Data Visualization, Exploration and Regression Analysis

A brief book using R

Kamarul Imran Wan Nor Arifin

2021-06-18

Kim would like to dedicate this book to his parents (Arwah Hj Musa and Napisah Mohamed Nor) and his parents in law, his wife, Juhara Haron, his sons, Afif and Iman and to his students, pass, current and future.

Arifin would like to

Contents

List of Tables

List of Figures

Preface

This is the **draft** version of our book on making plots, exploring data and doing regression analysis in R.

All in all, we hope you enjoy this book!

Kamarul Imran Musa

Wan Nor Arifin Wan Mansor

School of Medical Sciences,

Universiti Sains Malaysia

Chapter 1

Introduction to R, RStudio and RStudio Cloud

This chapter will introduce to readers to

- R and RStudio
- RStudio Cloud
- Installation for R and RStudio
- Optional installation for Miktex or Texlive and MacTex

- R scripts, R packages, R Taskview

1.1 RStudio Cloud

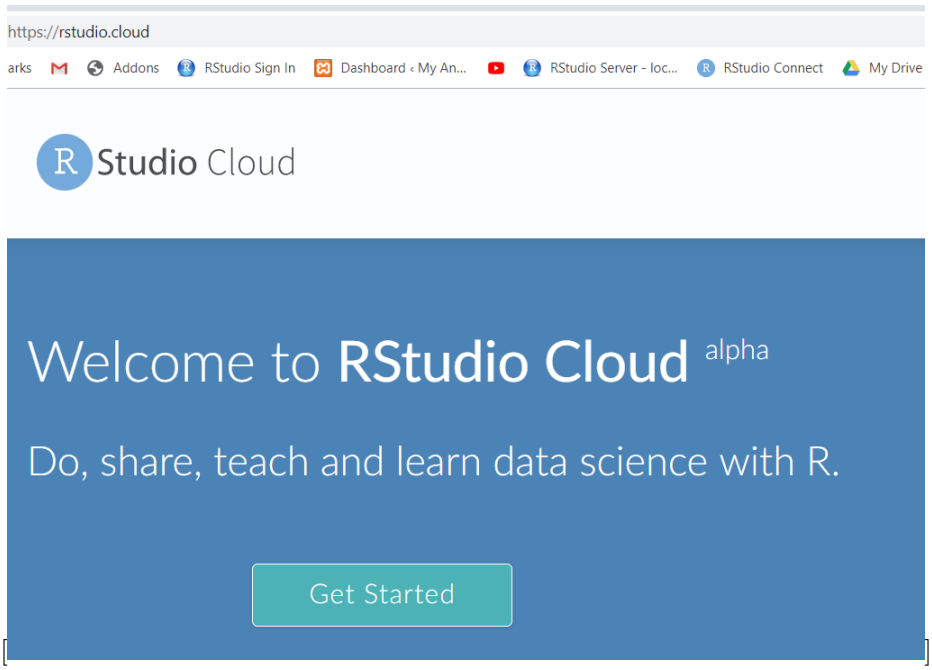
RStudio cloud facilitates the learning of R. Anyone can sign up and start using RStudio on the cloud.

It is one of the the quickest way to learn R. You do not need to install R on your machine. RStudio Cloud allows collaborration between teachers and students. It also helps colleagues working together on R codes.

RStudio Cloud is free for now. To start, you need to visit `https://rstudio.cloud/` and register.

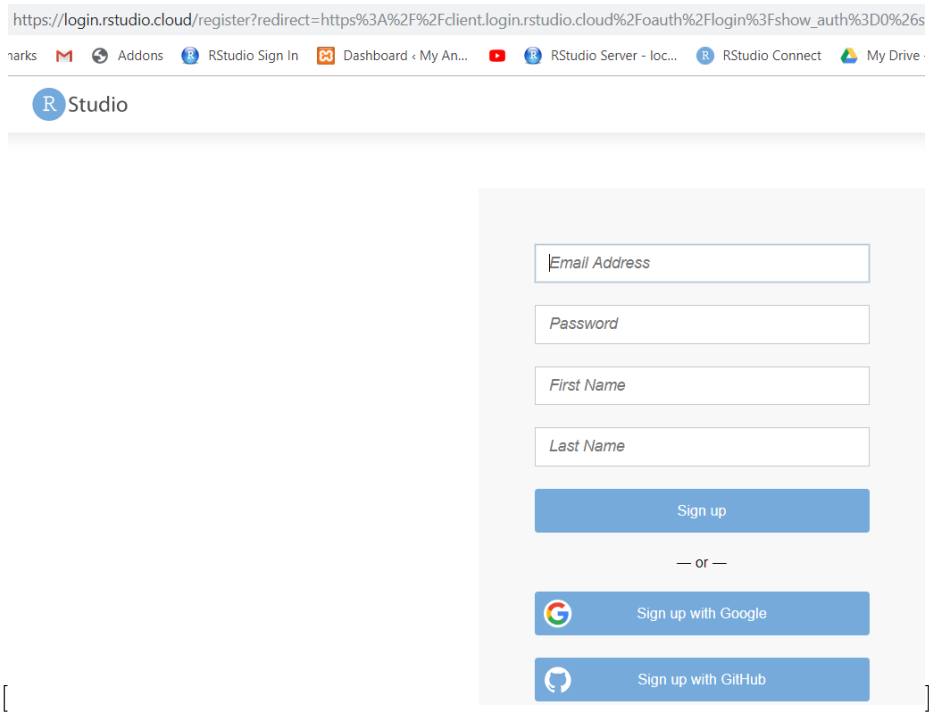
1.1.1 The RStudio Cloud Interface

This is the interface for RStudio Cloud.



1.1.2 Register and log in

You can register now. After registration, you can log in.

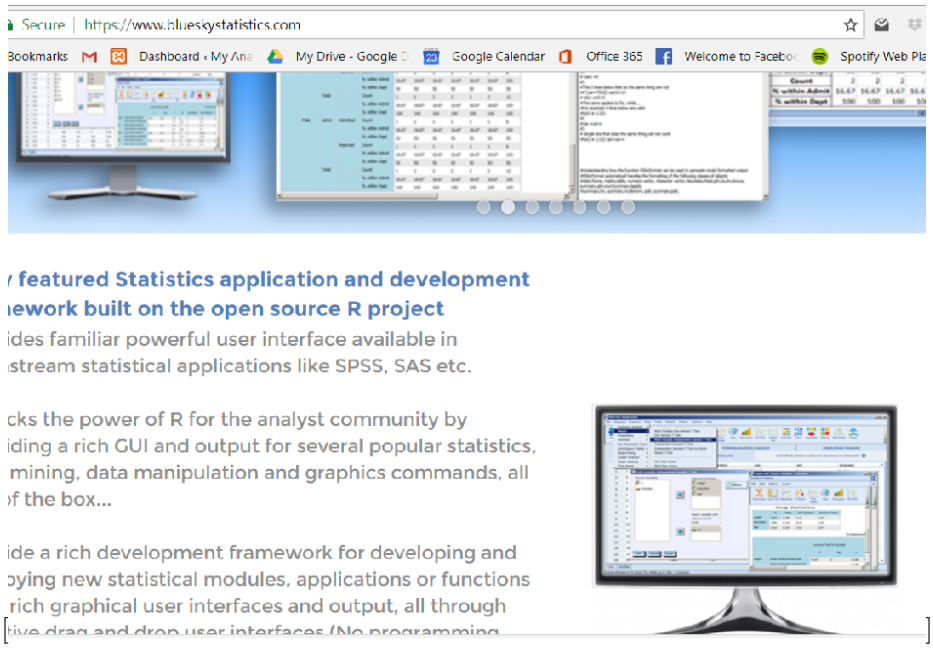


1.2 Point and click R GUI

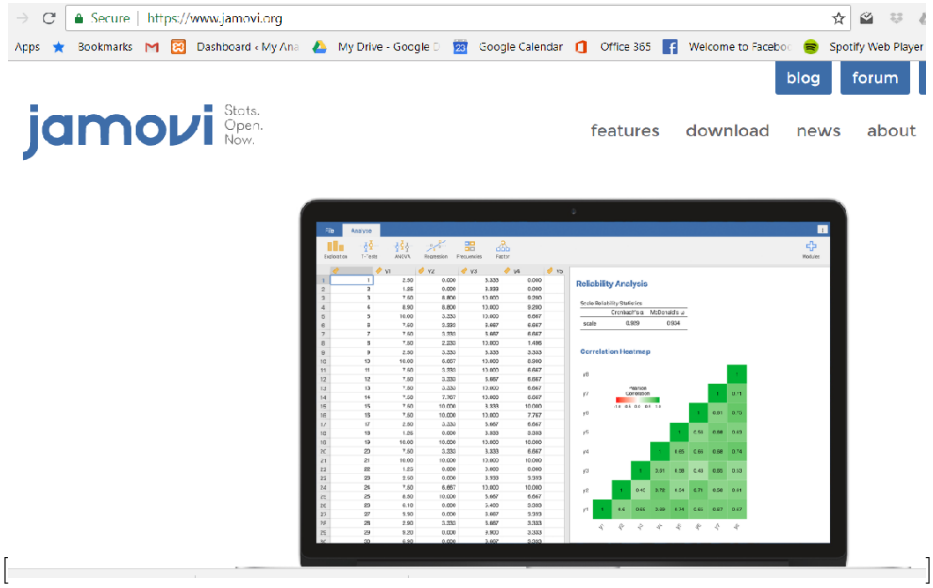
There are a number of SPSS-like GUI for R. For example

- Bluesky statistics https://www.blueskystatistics.com/
- JAMOVI - https://www.jamovi.org/

This is Bluesky statistics



And this is JAMOVI



1.3 RStudio Server

You can run R and RStudio on the server. To do this you have to install RStudio Server. having RStudio Server enables you to do analysis on the server.

This will give a taste of working on BIG DATA. There are two versions of RStudio Server

- RStudio Server
- RStudio Server Professional

For example, we have our RStudio Server Professional Edition (courtesy of RStudio) running on our server here https://healthdata.usm.my/rstudio/auth-sign-in

1.4 Installation

To install R on your machine, you have to have **Admin Right** to your machine. We recommend that you install

- R
- RStudio

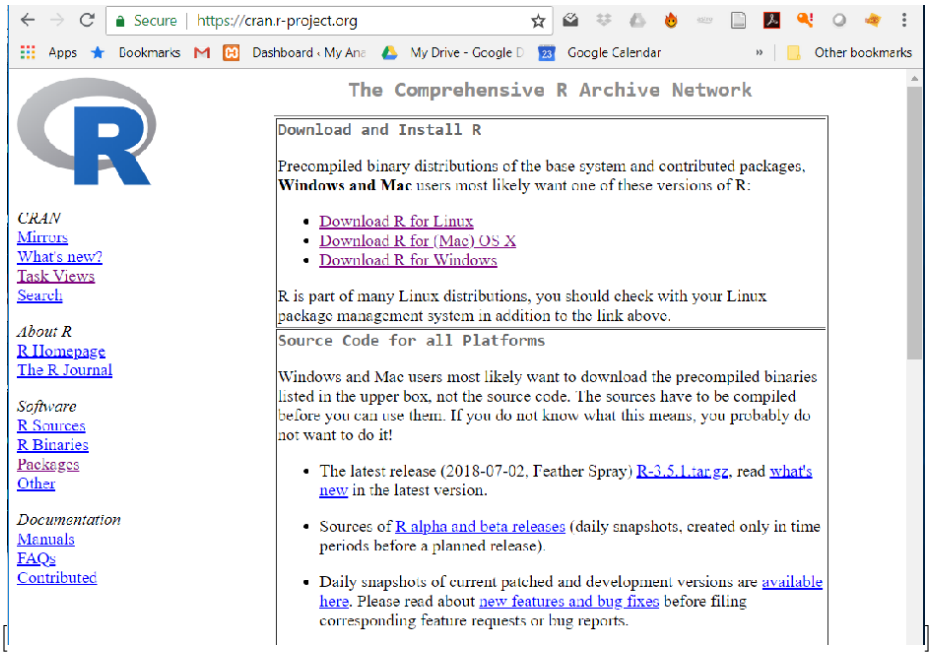
It is optional to install Latex editor for example MiKTeX and TeXLive for Windows and MacTex for Mac OS.

1.4.1 Installation for R

Though you can use R to run R codes. But we highly enciurage you to install both R and RStudio.

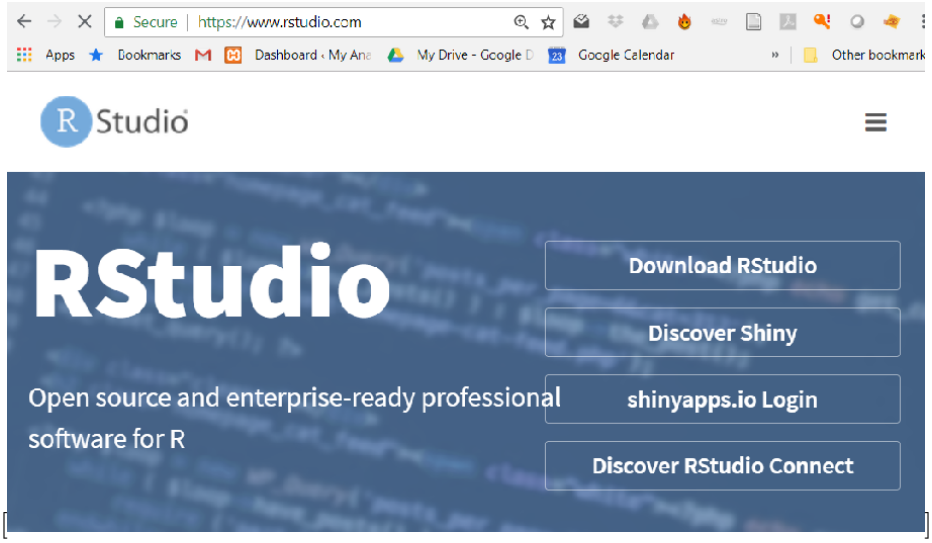
To install R, go to cran[1]. Then choose the correct R version for your machine OS. For example, for Windows OS the link is https://cran.r-project.org/bin/windows/base/R-3.6.1-win.exe. And for Mac OS, the download link is https://cran.r-project.org/bin/macosx/R-3.6.1.pkg. Similarly, if you are using Linux, follow the steps as listed before.

[1] https://cran.r-project.org/



1.4.2 Installation for RStudio

You can install RStudio for your OS from here https://www.rstudio.com/products/rstudio/download/#download. Choose the supported platforms. The size of download will be around 70-90 MB.



1.4.3 Check R and RStudio on your machine

Now, we assume you have installed R and RStudio. Please, check

- Do you have R?
- what version of R do you have?
- Do you have RStudio?
- what version of RStudio do you have?
- Do you need to update R and RStudio?

1.4.4 Installation of MiKTeX, TeXLive and MacTex

It is necessary to install Latex editor if you want to convert the outputs to pdf. Especially, if you run your codes on RMarkdown. It is because RMarkdown can produce different types of documents.

]

This is MiKTeX, for Window OS

And this is MacTeX, for Mac OS

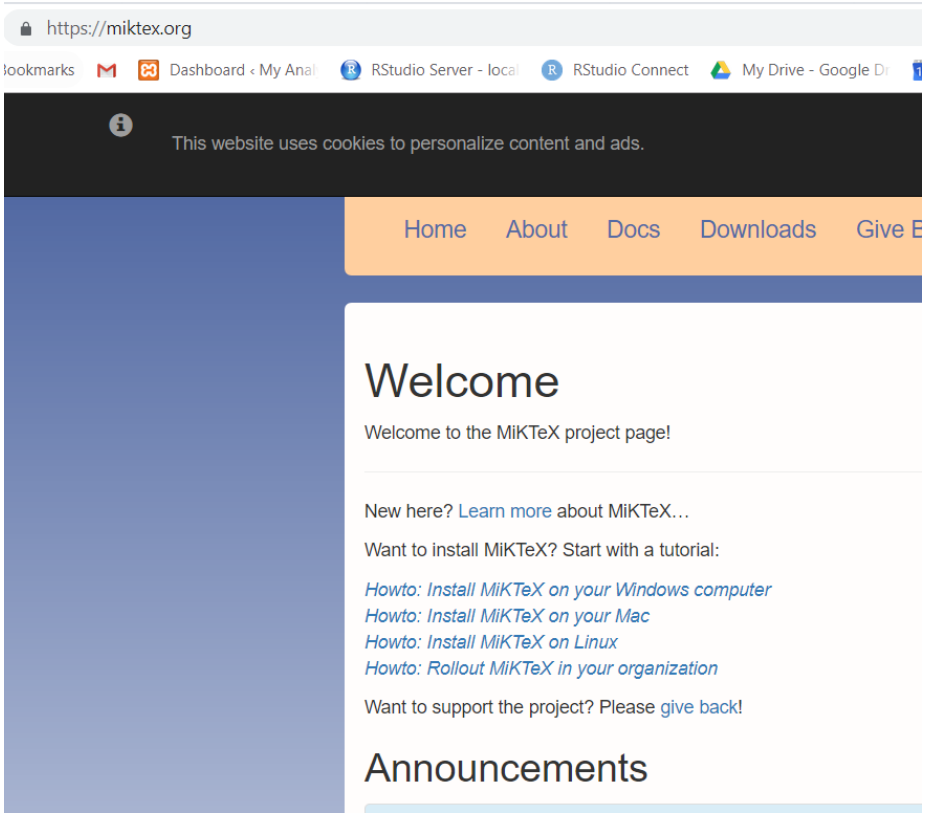


Figure 1.1: MikTeX webpage

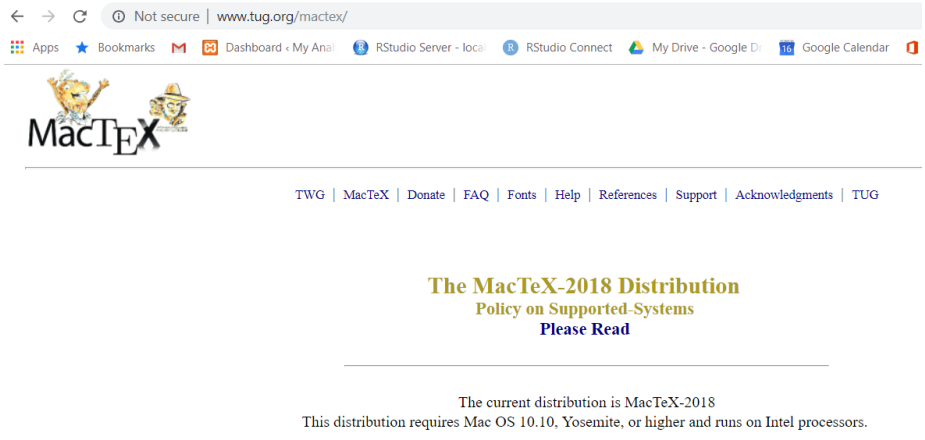
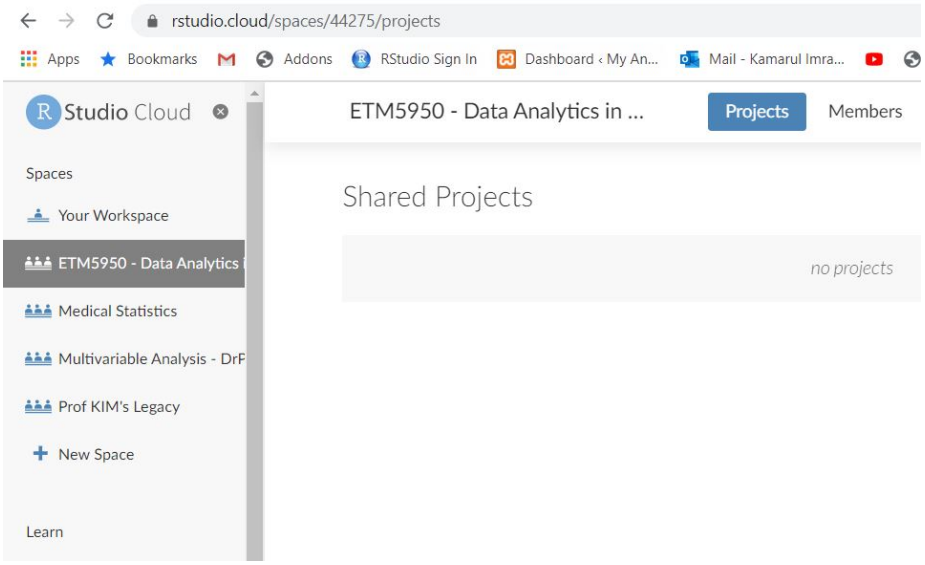


Figure 1.2: MacTeX webpage

1.5 Start your RStudio

You can either login to RStudio Cloud OR start RStudio on your machine. remember, to login to RStudio Cloud, go to https://rstudio.cloud. Then you will be asked for your username and password.

Click this link https://rstudio.cloud/spaces/44275/join?access_code=LPLoq5Q4kSdtBv1AN8kcHP%2FHGODiW1kGj4jVtG4k



To start R on your machine, find the Rstudio program in your start bar in your machine

What you see on RStudio now? You should see three panes if you start Rstudio for the first time or four panes if you have used RStudio before.

1.5.1 Console tab

In Console tab, this is where we will see most of the results

1.5.2 Files, Plots, Packages, Help and Viewer Pane

In this console, you will see

- List of objects
- R files, datasets, tables, list etc

1.5.3 Environment, History, Connection and Build Pane

In the environment, history, connection and build pane, you will see this

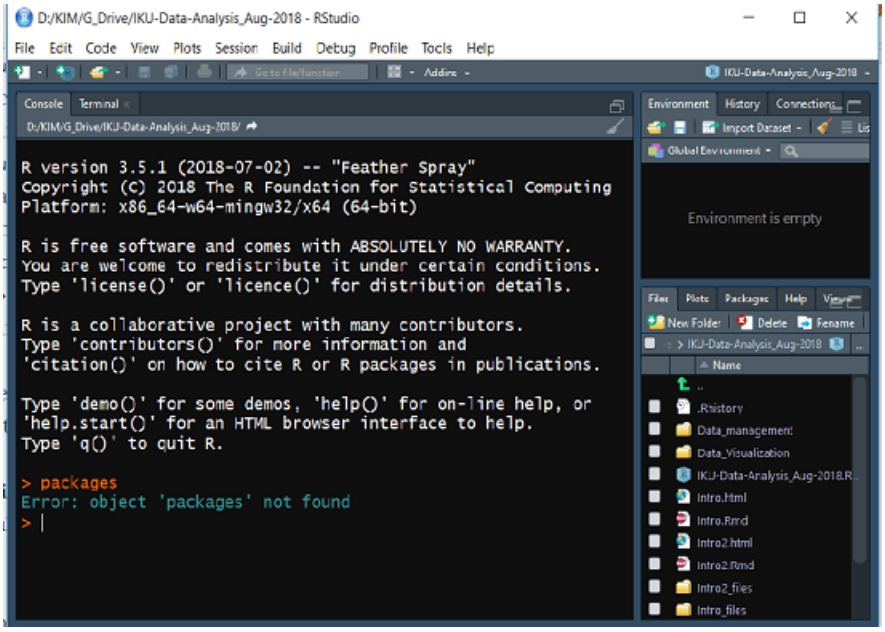


Figure 1.3: Rstudio

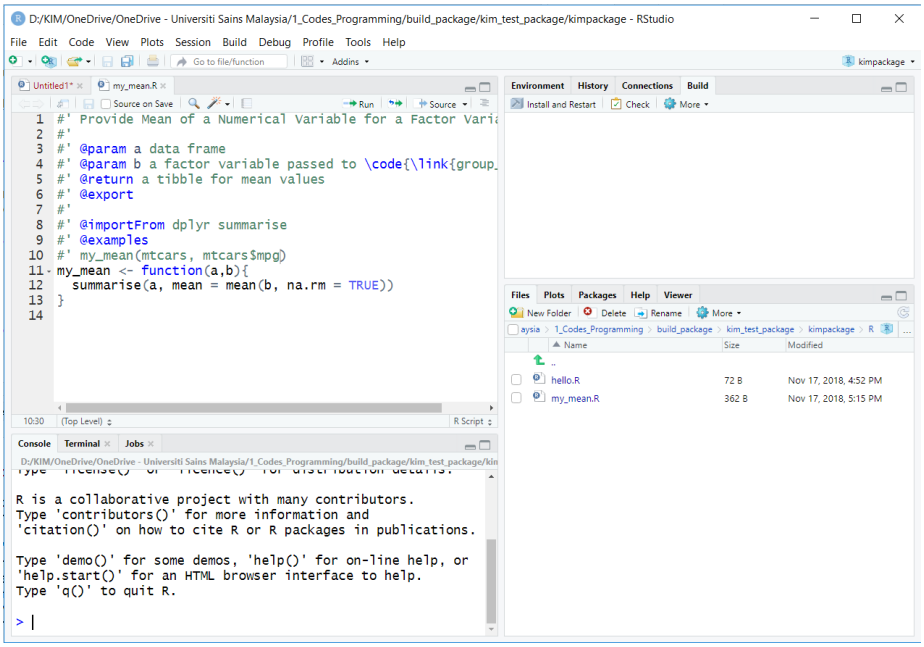


Figure 1.4: RStudio Panes

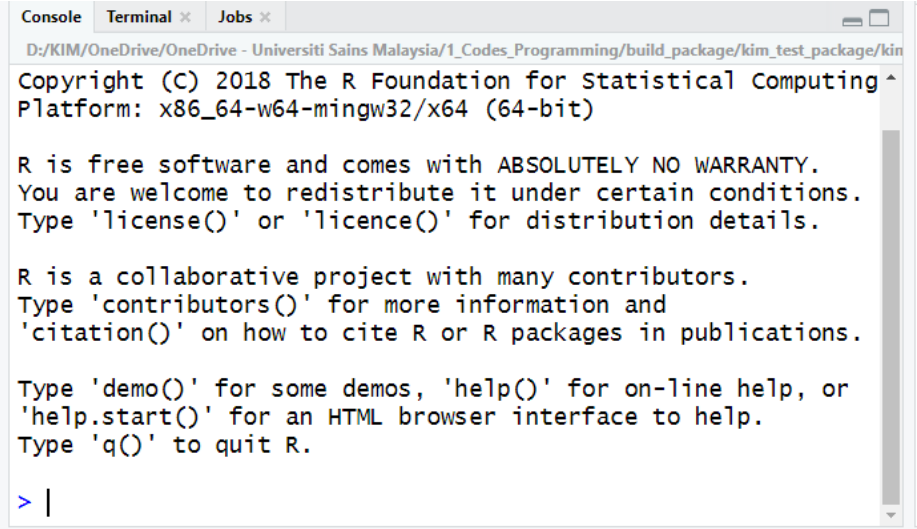


Figure 1.5: Console Pane

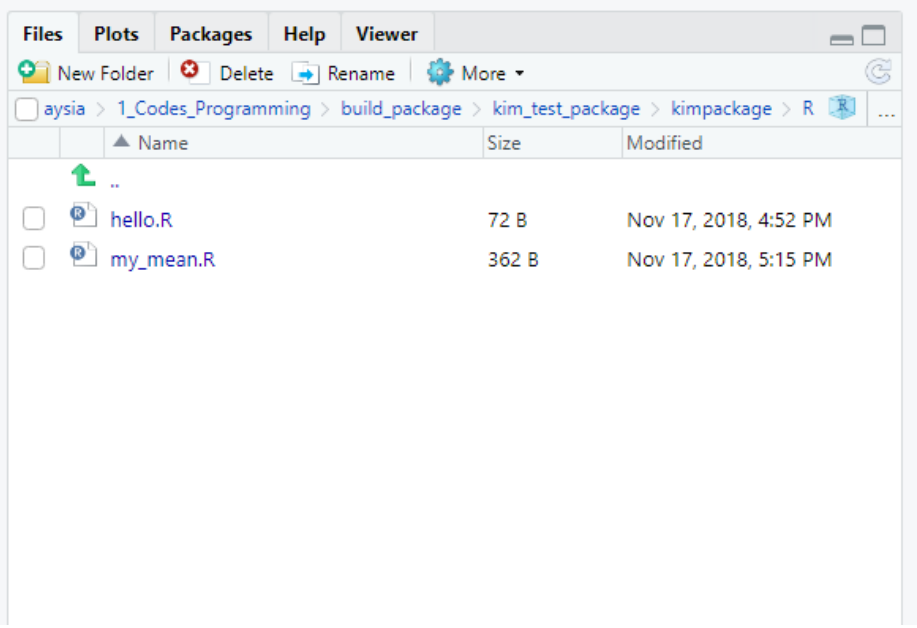


Figure 1.6: File, Plots and Viewer Pane

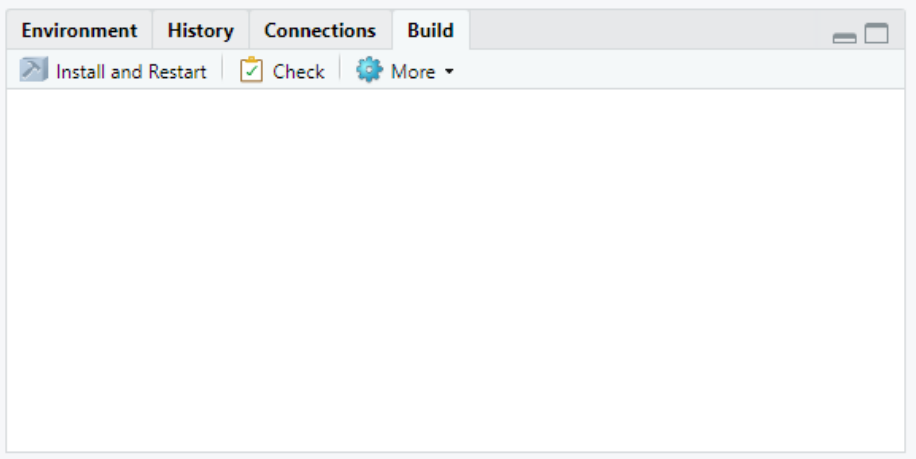


Figure 1.7: Environment Pane

1.5.4 Source Pane

In the Source pane, you can create R files and write your R codes

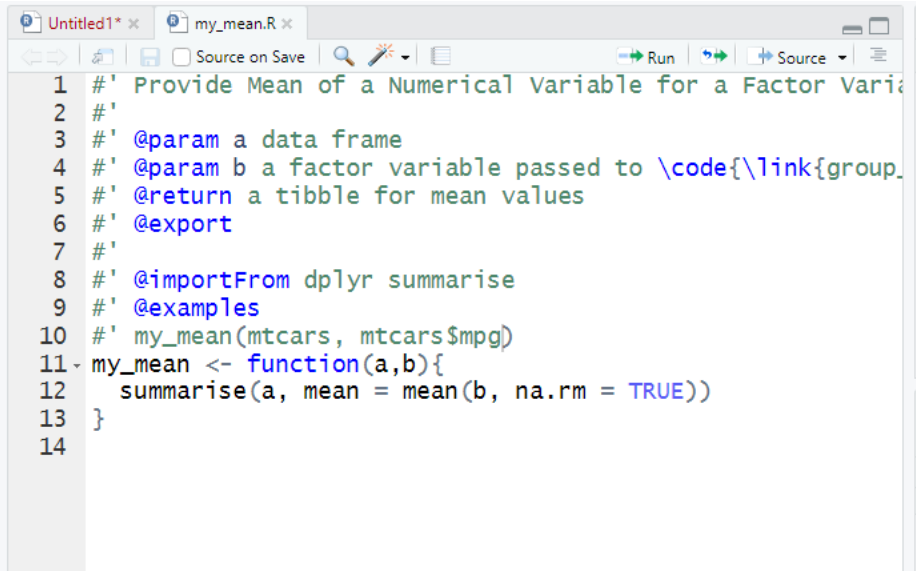


Figure 1.8: Source Pane

1.6 Open a new R script

To open a new R script

- File -> R Script
- In Window OS, CTRL-SHIFT-N

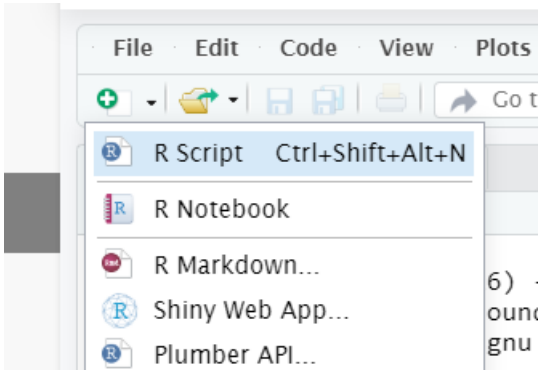


Figure 1.9: New R Script

1.6.1 Our first R script

Let us write our first R script.

- In Line 1, type `2 + 3`
- click CTRL-ENTER or CMD-ENTER
- see the outputs in the Console Pane

```
2 + 3
```

```
## [1] 5
```

After writing your R script, you can save it. This will allow to use or run the R script the next time you run RStudio.

To save R script

- File ->
- Save As ->
- Choose folder ->

- Name the file

Now, types this to check the version of R

```
version[6:7]
```

```
##       _
## major 4
## minor 1.0
```

The current version for R is 4, 1.0

If you lower version, then you want to upgrade. To upgrade

- for Windows, you can use **installr** package
- for Mac OS, you can use some functions

More info here https://www.linkedin.com/pulse/3-methods-update-r-rstudio-windows-mac-woratana-ngarmtrakulchol/

1.6.2 function, argument and parameters

R codes contain

- function
- argument
- parameters

```
f <- function(<arguments>) {
## Do something interesting
}
```

For example, for the function `lm()` to estimate parameters for linear regression model

```
args(lm)
```

```
## function (formula, data, subset, weights, na.action, method = "qr",
##     model = TRUE, x = FALSE, y = FALSE, qr = TRUE, singular.ok = TRUE,
##     contrasts = NULL, offset, ...)
## NULL
```

For example:

```
lm(weight ~ Time, data = ChickWeight)
```

```
##
## Call:
## lm(formula = weight ~ Time, data = ChickWeight)
##
## Coefficients:
## (Intercept)         Time
##      27.467        8.803
```

Ref:

- https://www.stat.auckland.ac.nz/~ihaka/downloads/Waikato-WRUG.pdf

- https://www.stat.berkeley.edu/~statcur/Workshop2/Presentations/functions.pdf

1.6.3 Need more help?

Then type the ? before the function

```
?lm
```

See what will be displayed in Help Pane

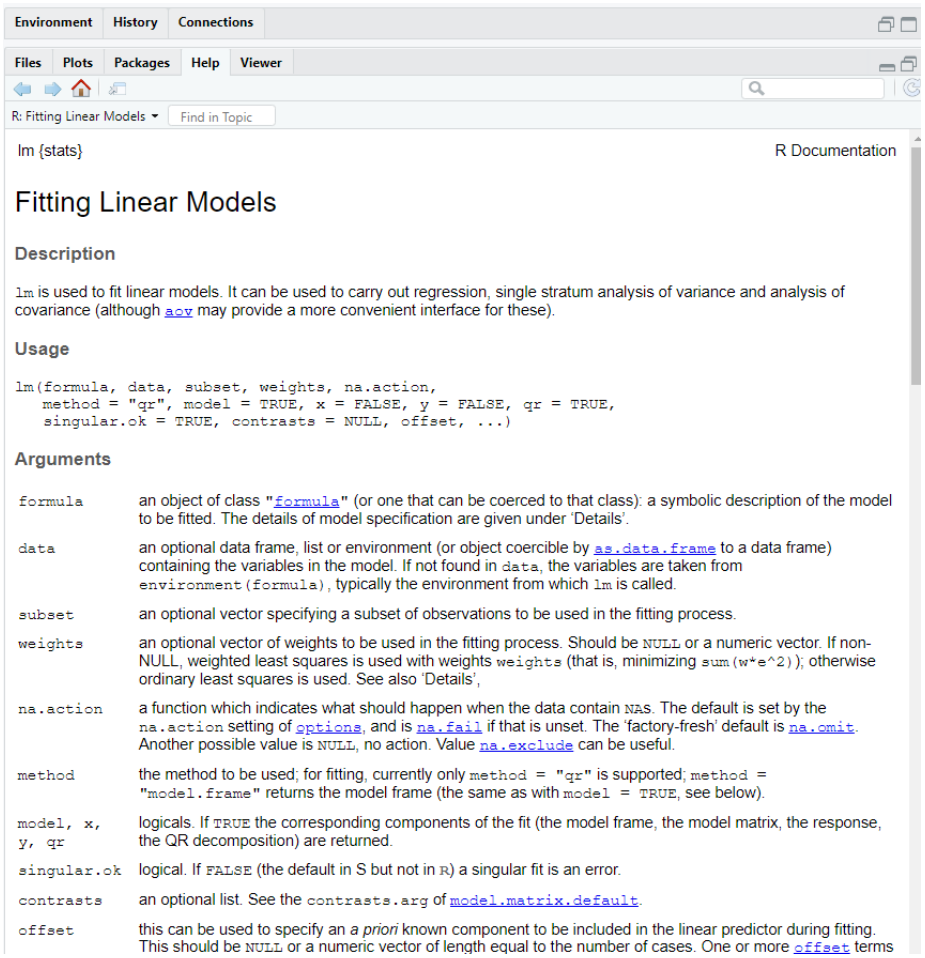


Figure 1.10: Help Pane

1.7 Packages

1.7.1 Packages on CRAN

https://cran.r-project.org/

- Currently, the CRAN package repository features 12784 available packages
- Cran Task Views

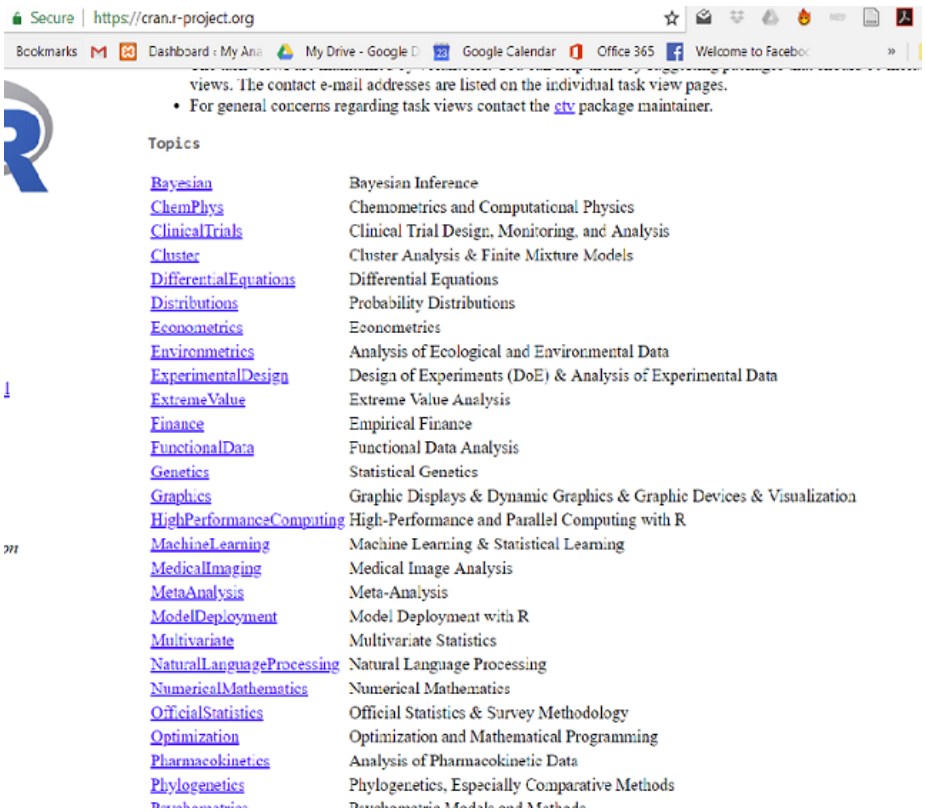


Figure 1.11: Task Views

1.7.2 Check if the package you need is available in your R library

Type this inside your console

```
library(ggplot2)
```

You should not receive any error message. If you have not installed the package, you will receive and error message. And it tells you that the package is not

available in your R. the package is stored in the R folder in your My Document or HOME directory

```
.libPaths()
```

```
## [1] "C:/Users/drkim/OneDrive/Documents/R/win-library/4.1"
## [2] "C:/Program Files/R/R-4.1.0/library"
```

1.7.3 Install an R package

To install an R package, you can type below (without the # tag)

```
# install.packages(foreign, dependencies = TRUE)
```

You need to have internet access. You can install from a zip file (from your machine or USB), from github and other repo

1.8 Directory

This is important. Not knowing your working directory will make you lost (you do not know where your R codes, R outputs, datasets etc)

You must know where your folder is located. The folder can contain many sub folders. The folder should contain dataset (if you want to analyze your data). It will later store the objects created during R session

```
getwd()
```

```
## [1] "C:/Users/drkim/OneDrive - Universiti Sains Malaysia/multivar_data_analysis"
```

You have to know to write file path. It is written differently for Window OS and other OS

1.8.1 Starting your R job

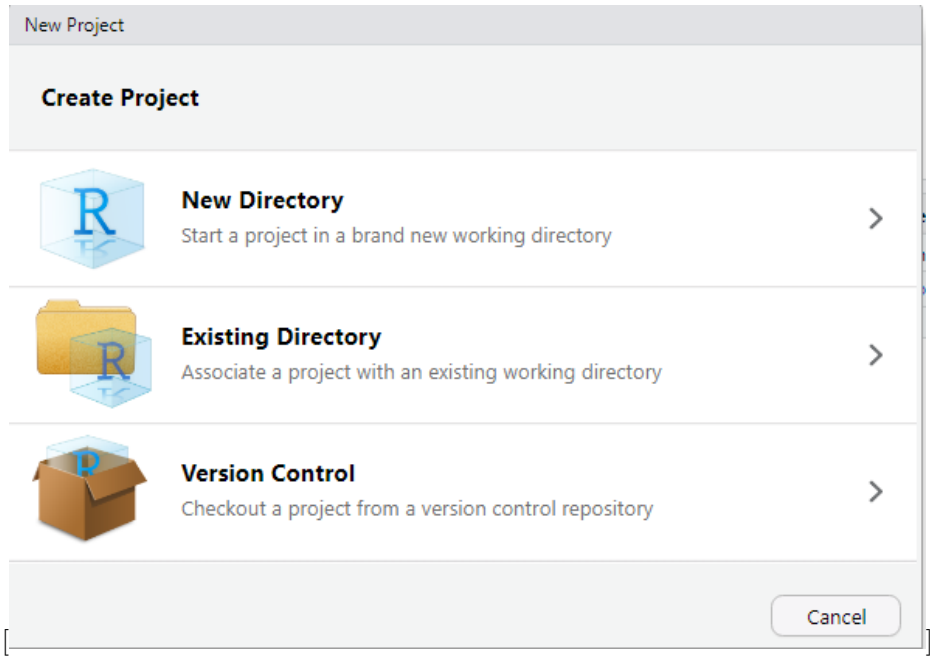
There are 2 ways to start your job:

- create a new project (recommended)
- setting your working directory using `setwd()` (not recommended)

1.8.2 Create new project

Always create a new project (This is the recommended way). This can be by

- Go to `File -> New Project`



When you see project type, click New Project

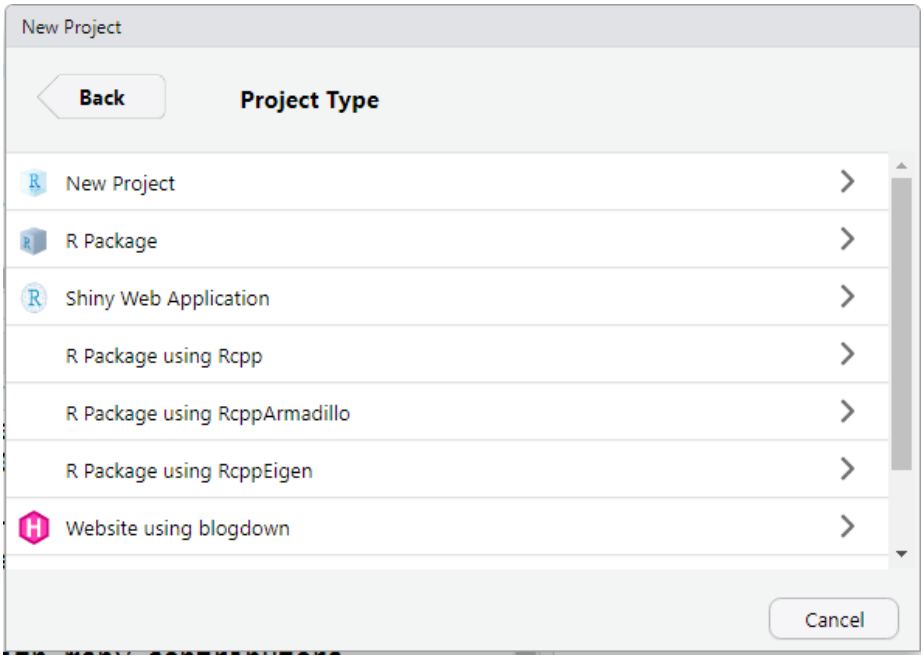


Figure 1.12: Project Type

1.8.3 Where is my data?

Datasets for analysis in R and usually in data frame format. You can see the datasets in the environment pane. Your data is read from the original dataset to a memory. SO you must know the size of your computer RAM. How much your RAM for your machine? The bigger the RAM, thelarger R can read and store your data.

The data that is read (in memory) will dissaper once you close RStudio. But the original stays in its location. This will not change your original data (so be happy!)

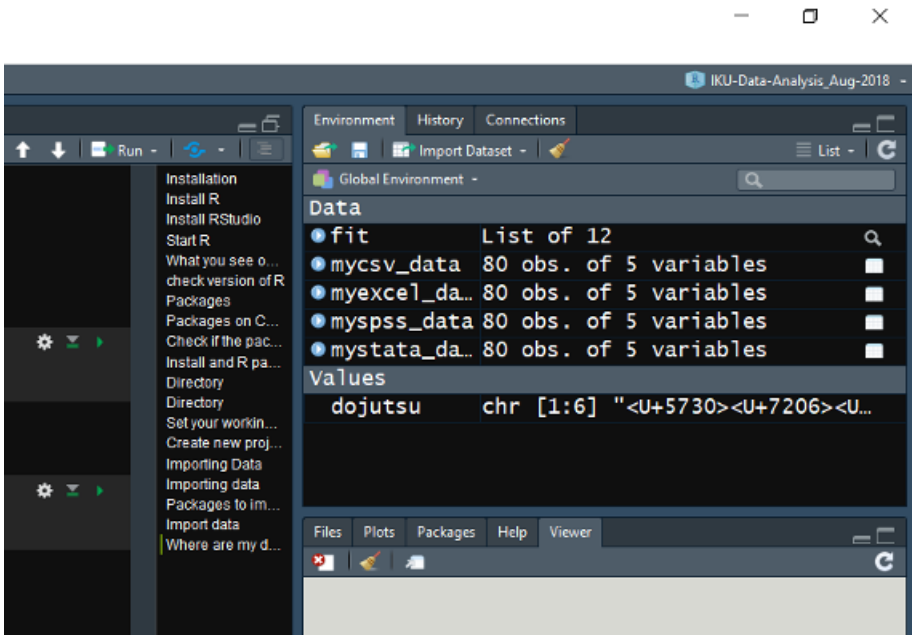


Figure 1.13: My Data

1.9 Upload data to RStudio Cloud

You have to upload data to RStudio Cloud Or link data to dropbox folder

1.10 More resources on RStudio Cloud

You can learn more about RStudio Cloud here

- on YouTube : RStudio Cloud for educationn https://www.youtube.com/watch?v=PviVimazpz8

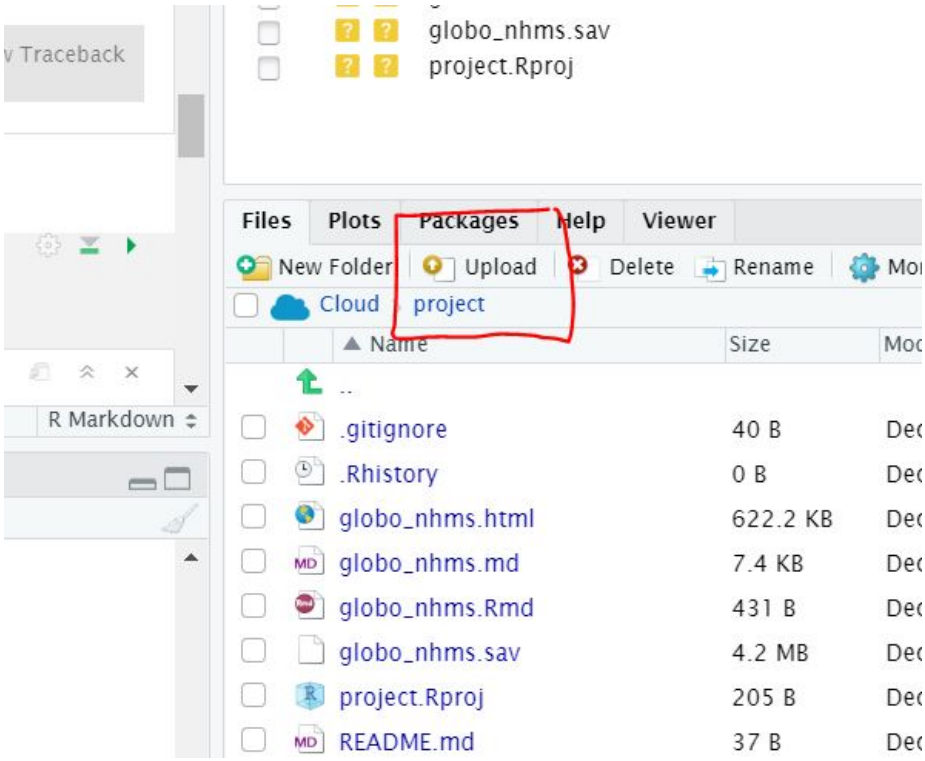


Figure 1.14: Upload Data in RStudio Cloud

- YouTube: Working with R in Cloud https://www.youtube.com/watch?v=SFpzr21Pavg

1.11 Need help?

If you need help you can

- Type a question mark infront of a function

```
?plot
```

Other options are these:

- register and join RStudio Community here https://community.rstudio.com/
- Ask questions on Stack Overflow https://stackoverflow.com/
- Search for mailing list and subscribe to it
- Books on R https://bookdown.org/

1.12 Bookdown

This webpage contains many useful books that use R codes https://bookdown.org/

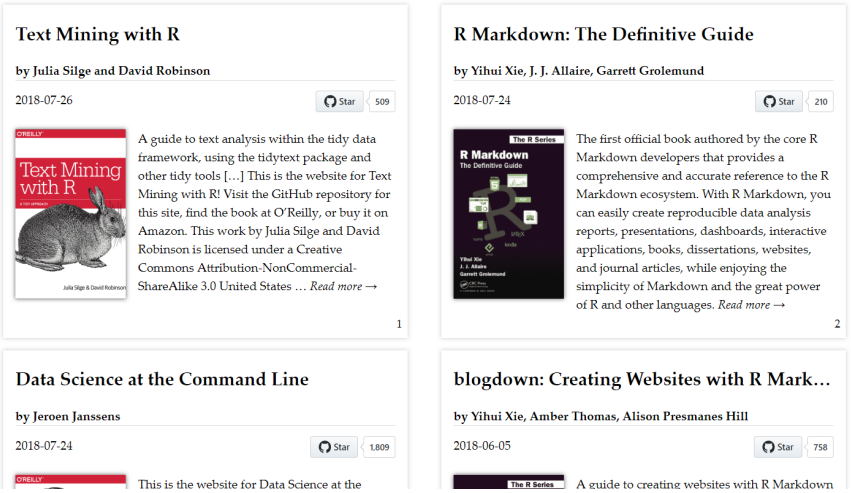


Figure 1.15: Bookdown

Chapter 2

Data Visualization

2.1 Introduction to visualization

Data visualization is viewed by many disciplines as a modern equivalent of visual communication. It involves the creation and study of the visual representation of data. Data visualization requires "information that has been abstracted in some schematic form, including attributes or variables for the units of information". You can read more about data visualization here https://en.m.wikipedia.org/wiki/Data_visualization and here https://en.m.wikipedia.org/wiki/Michael_Friendly

2.1.1 History of data visualization

In his 1983 book which carried the title *The Visual Display of Quantitative Information*, the author Edward Tufte defines **graphical displays** and principles for effective graphical display. The book mentioned that "Excellence in statistical graphics consists of complex ideas communicated with clarity, precision and efficiency."

2.1.2 Processes and Objectives of visualization

Visualization is the process of representing data graphically and interacting with these representations. The objective is to gain insight into the data. Some of the processes are outlined here http://researcher.watson.ibm.com/researcher/view_group.php?id=143

2.2 What makes good graphics

You may require these to make good graphics:

1. Data
2. Substance rather than about methodology, graphic design, the technology of graphic production or something else
3. No distortion to what the data has to say
4. Presence of many numbers in a small space
5. Coherence for large data sets
6. Encourage the eye to compare different pieces of data
7. Reveal the data at several levels of detail, from a broad overview to the fine structure
8. Serve a reasonably clear purpose: description, exploration, tabulation or decoration
9. Be closely integrated with the statistical and verbal descriptions of a data set.

2.3 Graphics packages in R

There are many **graphics packages** in R. Some packages are aimed to perform general tasks related with graphs. Some provide specific graphics for certain analyses. The popular general graphics packages in R are:

1. **graphics** : a base R package
2. **ggplot2** : a user-contributed package by Hadley Wickham
3. **lattice** : a user-contributed package

Except for **graphics** package (a a base R package), other packages need to downloaded and installed into your R library. Examples of other more specific packages - to run graphics for certain analyses - are:

1. **survminer::ggsurvlot**
2. **sjPlot**

For this course, we will focus on using the **ggplot2** package.

2.4 Introduction to ggplot2 package

The **ggplot2** package is an elegant, easy and versatile general graphics package in R. It implements the **grammar of graphics** concept. The advantage of this concept is that, it fasten the process of learning graphics. It also facilitates the process of creating complex graphics

To work with **ggplot2**, remember

- start with: `ggplot()`
- which data: `data = X`
- which variables: `aes(x = , y = )`
- which graph: `geom_histogram()`, `geom_points()`

The official website for ggplot2 is here http://ggplot2.org/.

ggplot2 is a plotting system for R, based on the grammar of graphics, which tries to take the good parts of base and lattice graphics and none of the bad parts. It takes care of many of the fiddly details that make plotting a hassle (like drawing legends) as well as providing a powerful model of graphics that makes it easy to produce complex multi-layered graphics.

2.5 Preparation

2.5.1 Set a new project or set the working directory

It is always recommended that to start working on data analysis in RStudio, you create first a new project.

Go to File, then click New Project.

You can create a new R project based on existing directory. This method is useful because an RStudio project keep your data, your analysis, and outputs in a clean dedicated folder or sets of folders.If you do not want to create a new project, then make sure you are inside the correct directory (the working directory). The working directory is a folder where you store.

Type `getwd()` in your Console to display your working directory. Inside your working directory, you should see and keep

1. dataset or datasets
2. outputs - plots
3. codes (R scripts `.R`, R markdown files `.Rmd`)

2.5.2 Questions to ask before making graphs

You must ask yourselves these:

1. Which variable or variables do I want to plot?
2. What is (or are) the type of that variable?

- Are they factor (categorical) variables ?
- Are they numerical variables?

3. Am I going to plot

- a single variable?
- two variables together?
- three variables together?

2.5.3 Read data

The common data formats include

1. comma separated files (`.csv`)
2. MS Excel file (`.xlsx`)

3. SPSS file (`.sav`)
4. Stata file (`.dta`)
5. SAS file

Packages that read these data include **haven** package. Below are the functions to read SAS, SPSS and Stata file.

1. SAS: `read_sas()` reads .sas7bdat + .sas7bcat files and read_xpt() reads SAS transport files (version 5 and version 8). write_sas() writes .sas7bdat files.
2. SPSS: `read_sav()` reads .sav files and read_por() reads the older .por files. write_sav() writes .sav files.
3. Stata: `read_dta()` reads .dta files (up to version 15). write_dta() writes .dta files (versions 8-15).

Data from databases are less common but are getting more important and more common. R can also read these data. Some examples of databases format are:

1. MySQL
2. SQLite
3. Postgresql
4. Mariadb

2.5.4 Load the library

The **ggplot2** package is one of the core member of **tidyverse** package (https://www.tidyverse.org/). So, if we load the **tidyverse** package, we will access to other packages under **tidyverse** which include **dplyr**, **readr**, **ggplot2**.

Loading a package will give you access to

1. help pages
2. functions
3. datasets

```
library(tidyverse)
```

If you run the code and you see *there is no package called tidyverse* then you need to install the **tidyverse** package. To install the package, type `install.package("tidyverse")` in the Console. Once the installation is complete, type `library(tidyverse)` to load the package.

2.5.5 Open dataset

For now, we will use the built-in dataset in the **gapminder** package. You can read more about *gapminder* from https://www.gapminder.org/. The gapminder website contains many useful datasets and show wonderful graphics. It is made popular by Dr Hans Rosling.

To load the package, type

```
library(gapminder)
```

call the data *gapminder* into R and browse the first 6 observations of the *gapminder* data

```
gapminder <- gapminder
head(gapminder)
```

```
## # A tibble: 6 x 6
##   country     continent  year lifeExp      pop gdpPercap
##   <fct>       <fct>     <int>   <dbl>    <int>     <dbl>
## 1 Afghanistan Asia       1952    28.8  8425333      779.
## 2 Afghanistan Asia       1957    30.3  9240934      821.
## 3 Afghanistan Asia       1962    32.0 10267083      853.
## 4 Afghanistan Asia       1967    34.0 11537966      836.
## 5 Afghanistan Asia       1972    36.1 13079460      740.
## 6 Afghanistan Asia       1977    38.4 14880372      786.
```

We can list the variables and look at the type of the variables in the dataset

```
glimpse(gapminder)
```

```
## Rows: 1,704
## Columns: 6
## $ country   <fct> "Afghanistan", "Afghanistan", "Afghanistan", "Afghanistan", ~
## $ continent <fct> Asia, Asia, Asia, Asia, Asia, Asia, Asia, Asia, Asia, Asia, ~
## $ year      <int> 1952, 1957, 1962, 1967, 1972, 1977, 1982, 1987, 1992, 1997, ~
## $ lifeExp   <dbl> 28.801, 30.332, 31.997, 34.020, 36.088, 38.438, 39.854, 40.8~
## $ pop       <int> 8425333, 9240934, 10267083, 11537966, 13079460, 14880372, 12~
## $ gdpPercap <dbl> 779.4453, 820.8530, 853.1007, 836.1971, 739.9811, 786.1134, ~
```

The *gapminder* data have

1. 6 variables
2. 1704 observations
3. There are 2 factor variables, 2 integer variables and 2 numeric variables

We can examine the basic statistics of the datasets by using `summary()`. This function will list

1. the frequencies
2. some descriptive statistics: min, 1st quartile, median, mean, 3rd quartile and max

```
summary(gapminder)
```

```
##         country        continent        year         lifeExp
##  Afghanistan:  12   Africa  :624   Min.   :1952   Min.   :23.60
##  Albania    :  12   Americas:300   1st Qu.:1966   1st Qu.:48.20
##  Algeria    :  12   Asia    :396   Median :1980   Median :60.71
##  Angola     :  12   Europe  :360   Mean   :1980   Mean   :59.47
##  Argentina  :  12   Oceania : 24   3rd Qu.:1993   3rd Qu.:70.85
##  Australia  :  12                  Max.   :2007   Max.   :82.60
##  (Other)    :1632
##       pop              gdpPercap
##  Min.   :6.001e+04   Min.   :   241.2
##  1st Qu.:2.794e+06   1st Qu.:  1202.1
##  Median :7.024e+06   Median :  3531.8
##  Mean   :2.960e+07   Mean   :  7215.3
##  3rd Qu.:1.959e+07   3rd Qu.:  9325.5
##  Max.   :1.319e+09   Max.   :113523.1
##
```

To know more about the package, we can use the ? mark

```
?gapminder
```

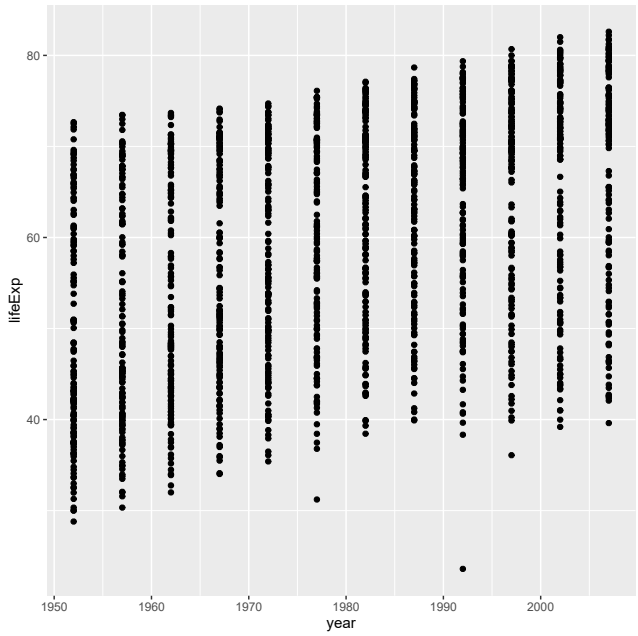
2.6 Basic plot

We can start create a basic plot by setting these parameters

- data = gapminder
- variables = year, lifeExp
- graph = scatterplot

In **ggplot2** which is a package under **tidyverse** package, you can use the + sign to connect the function. And in R, your codes can span multiple lines. This will increase the visibility of the codes.

```
ggplot(data = gapminder) +
  geom_point(mapping = aes(x = year, y = lifeExp))
```



Now, you can see that the plot shows:

1. the relationship between year and life expectancy.
2. as variable year advances, the life expectancy increases.

the `ggplot()` tells R to plot what variables from what data. And `geom_point()` tells R to make a scatter plot.

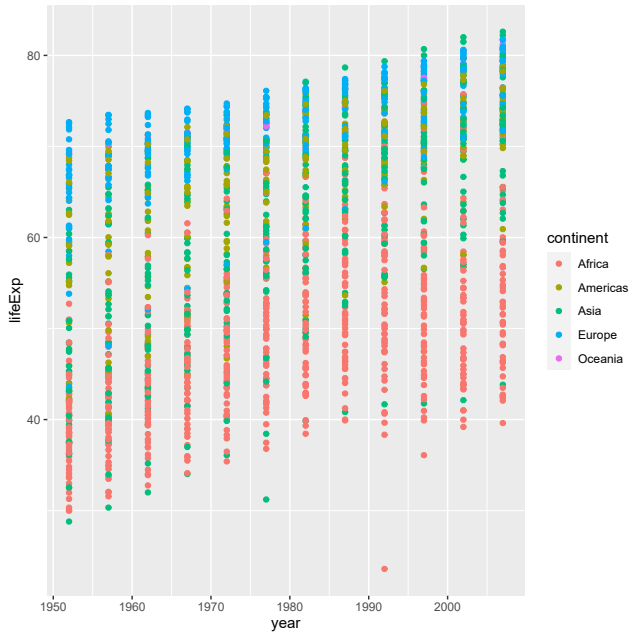
2.7 Adding another variable

You realize that we plotted 2 variables based on `aes()`. We can add the third variable to make a more complicated plot. For example:

1. data = gapminder
2. variables = year, life expectancy, continent

For this, the objective to create plot might be to see the relationship between year and life expectancy based on continent.

```
ggplot(data = gapminder) +
  geom_point(mapping = aes(x = year, y = lifeExp, colour = continent))
```

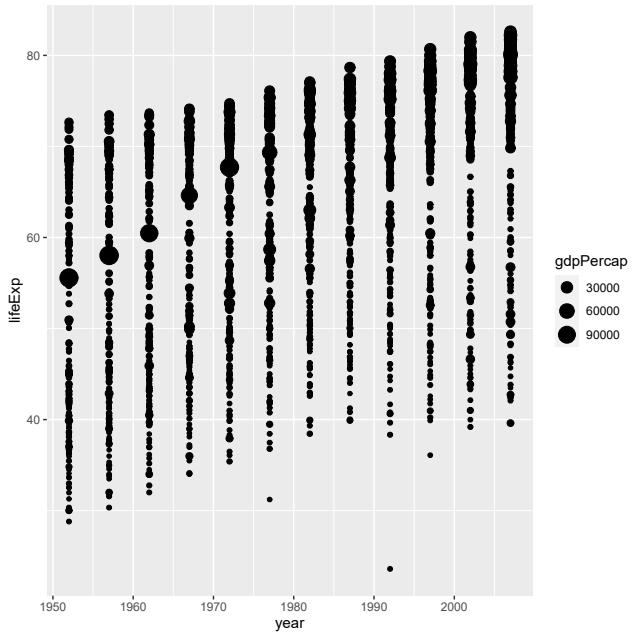


What can you see from the scatterplot? You may notice that

1. Europe countries have high life expectancy
2. Africa countries have lower life expectancy
3. One Asia country looks like an outlier (very low life expectancy)
4. One Africa country looks like an outlier (very low life expectancy)

Now, we will replace the 3rd variable with GDP (variable gdpPercap) and make the plot correlates with the size of GDP.

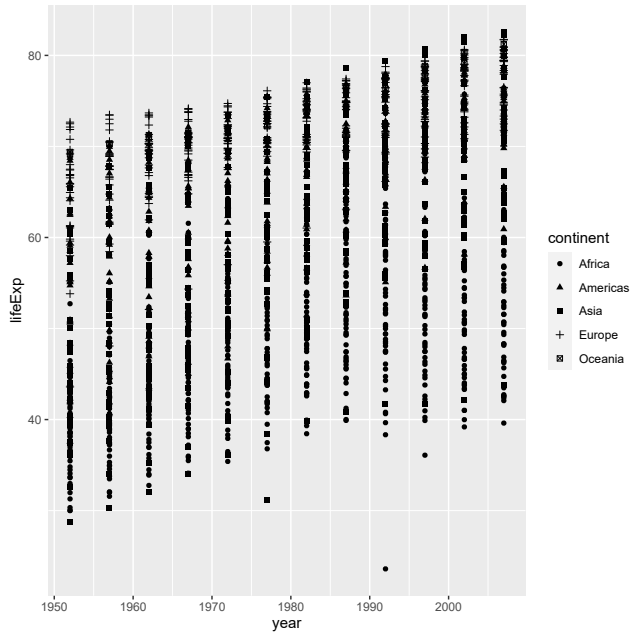
```
ggplot(data = gapminder) +
  geom_point(mapping = aes(x = year, y = lifeExp, size = gdpPercap))
```



ggplot2 will automatically assign a unique level of the aesthetic (here a unique color) to each unique value of the variable, a process known as scaling. *ggplot2* will also add a legend that explains which levels correspond to which values. The plot suggets that higher GDP countries have longer life expectancy.

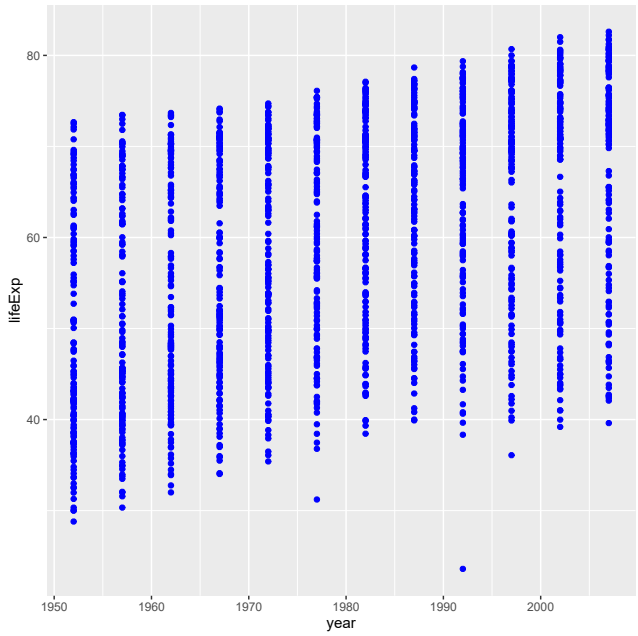
Instead of using colour, we can use shape especially in instances where there is no facility to print out colour plots.

```
ggplot(data = gapminder) +
  geom_point(mapping = aes(x = year, y = lifeExp, shape = continent))
```



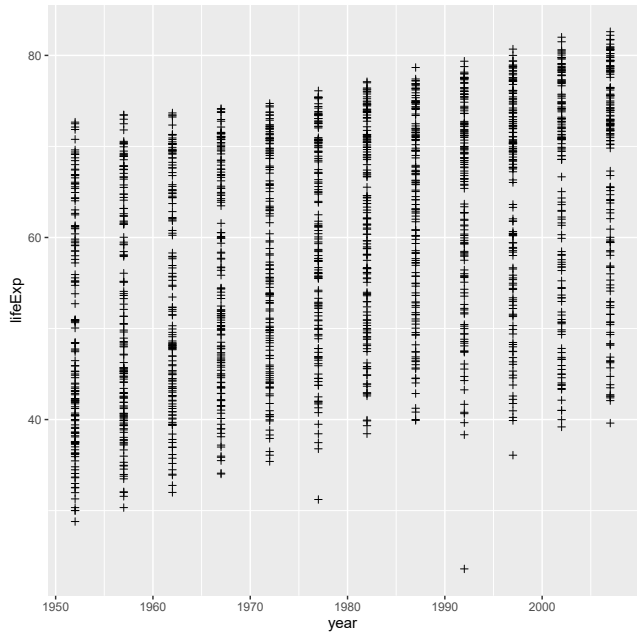
But, see what will happen if you set the colour and shape like below but outside the aes parentheses. For example, let set the parameter colour to blue

```
ggplot(data = gapminder) +
  geom_point(mapping = aes(x = year, y = lifeExp), colour = 'blue')
```



And then parameter shape to plus (which is represented by number 3).

```
ggplot(data = gapminder) +
  geom_point(mapping = aes(x = year, y = lifeExp), shape = 3)
```



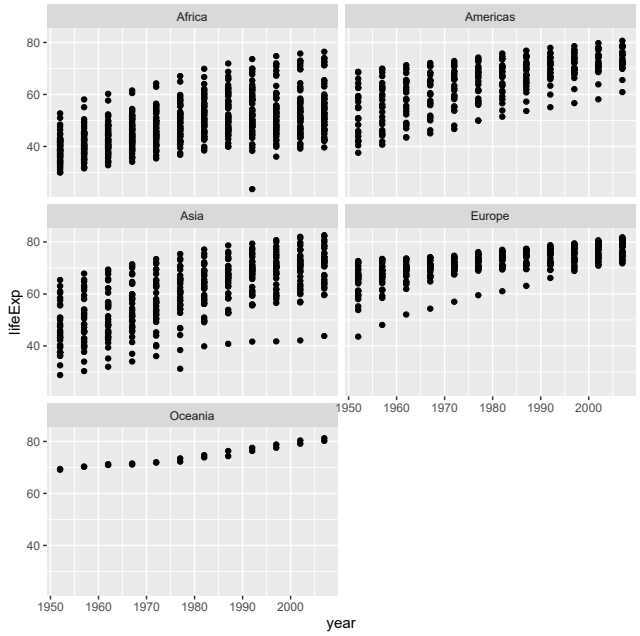
You may wonder what number corresponds to what type of shape. You can type ?*pch*. And you will see in the Viewer pane, the explanation about the shape available in R. It also shows what number that corresponds to what shape.

2.8 Making subplots

We can split our plots based on a factor variable and make subplots using the `facet()`. For example, if we want to make subplots based on continents, then you need to set these parameters:

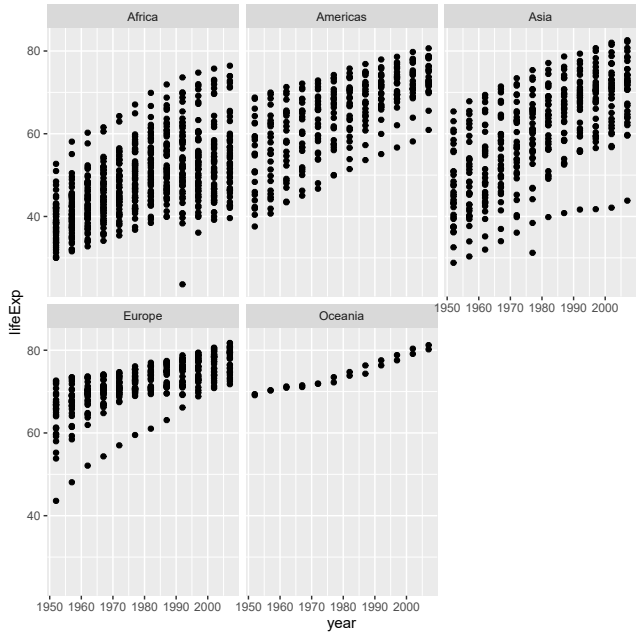
- data = gapminder
- variable year on the x-axis and lifeExp on the y-axis
- split the plot based on continent
- the number of rows for the plot are 3

```
ggplot(data = gapminder) +
  geom_point(mapping = aes(x = year, y = lifeExp)) +
  facet_wrap(~ continent, nrow = 3)
```



Now, what happen if we change the value for the nrow

```
ggplot(data = gapminder) +
  geom_point(mapping = aes(x = year, y = lifeExp)) +
  facet_wrap(~ continent, nrow = 2)
```

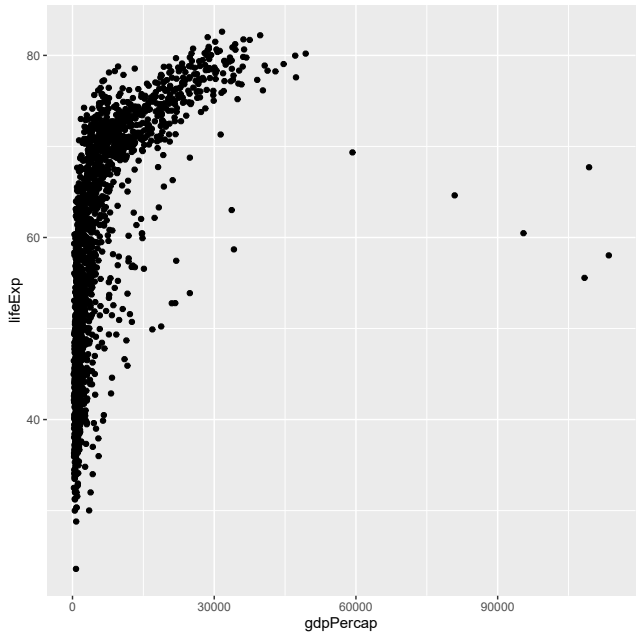


2.9 Overlaying plots

Each `geom_X()` in ggplot2 indicates different visual objects.

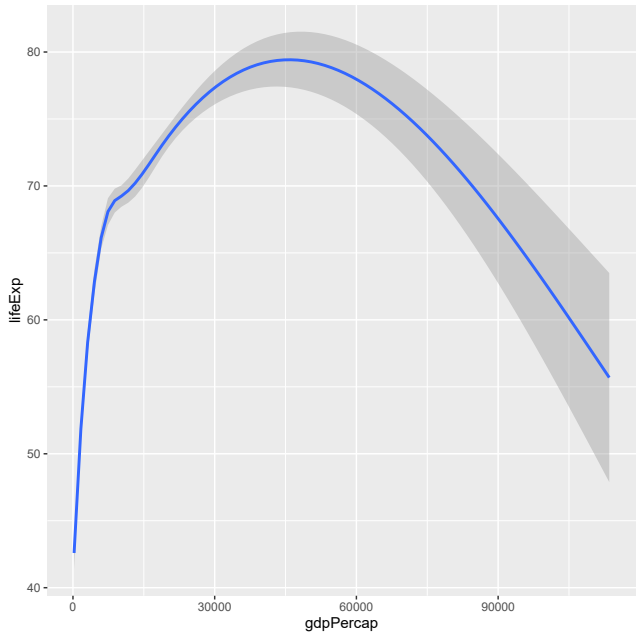
This is a scatterplot

```
ggplot(data = gapminder) +
  geom_point(mapping = aes(x = gdpPercap, y = lifeExp))
```



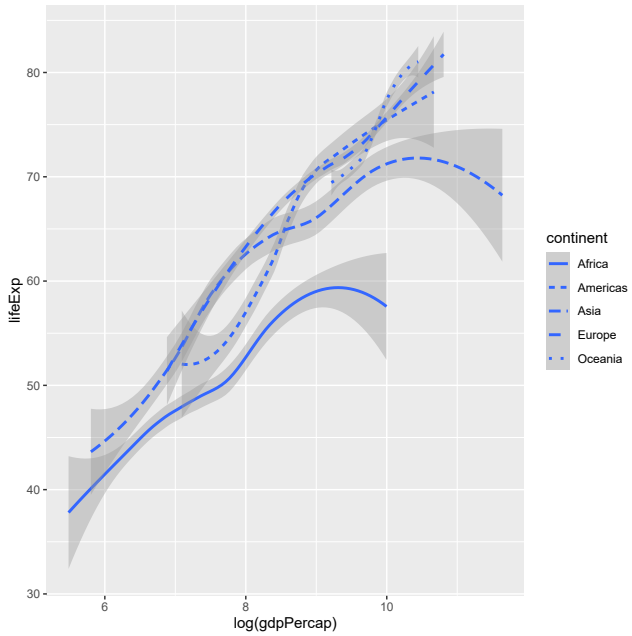
This is a smooth line

```
ggplot(data = gapminder) +
  geom_smooth(mapping = aes(x = gdpPercap, y = lifeExp))
```



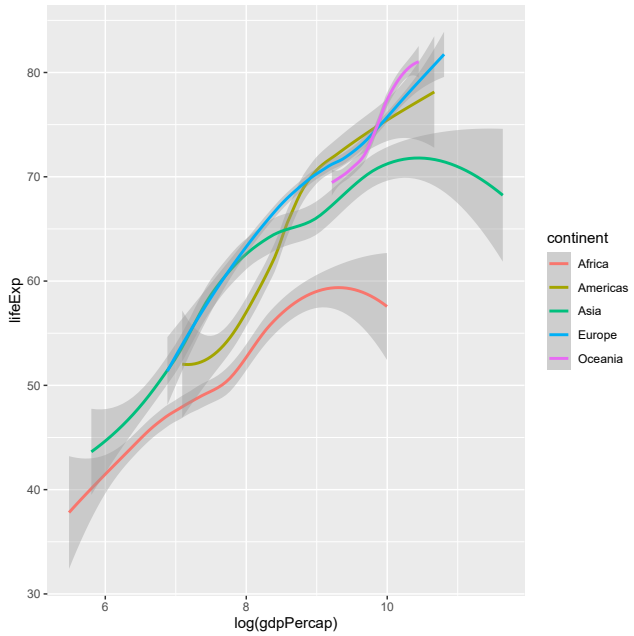
And we can regenerate the smooth plot based on continent using the `linetype()`. We use `log(gdpPercap)` to reduce the skewness of the data.

```
ggplot(data = gapminder) +
  geom_smooth(mapping = aes(x = log(gdpPercap), y = lifeExp, linetype = continent))
```



Another smooth plot but setting the parameter for colour

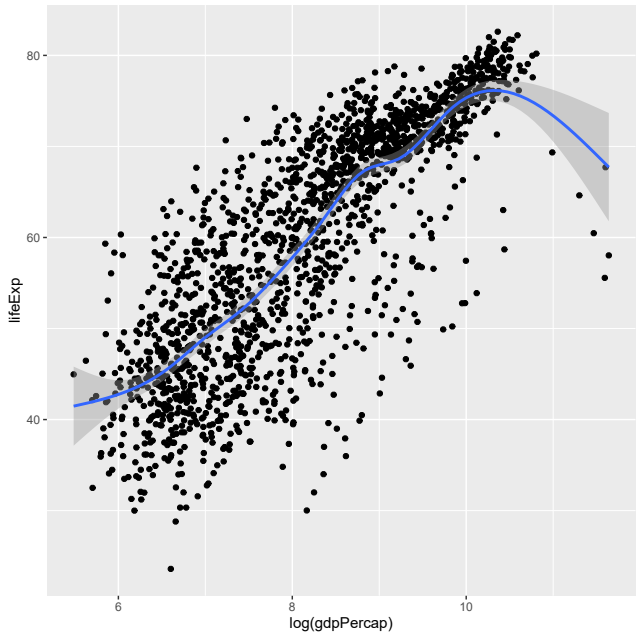
```
ggplot(data = gapminder) +
  geom_smooth(mapping = aes(x = log(gdpPercap), y = lifeExp, colour = continent))
```



2.10 Combining geom

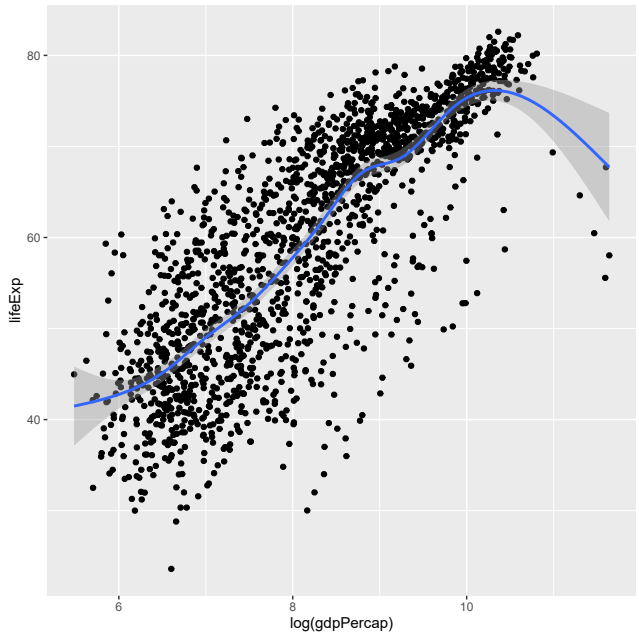
We can combine more than one geoms to overlay plots. The trick is to use multiple geoms in a single line of R code

```
ggplot(data = gapminder) +
  geom_point(mapping = aes(x = log(gdpPercap), y = lifeExp)) +
  geom_smooth(mapping = aes(x = log(gdpPercap), y = lifeExp))
```



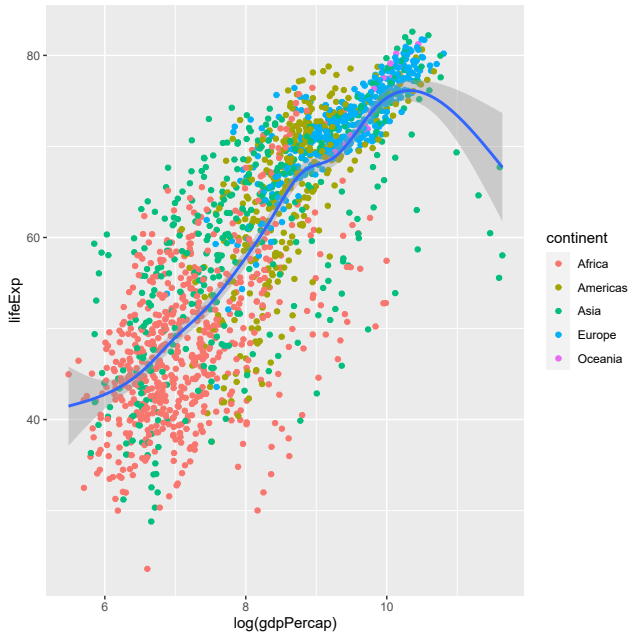
The codes above show duplication or repetition. To avoid this, we can pass the mapping to `ggplot()`.

```
ggplot(data = gapminder, mapping = aes(x = log(gdpPercap), y = lifeExp)) +
  geom_point() +
  geom_smooth()
```



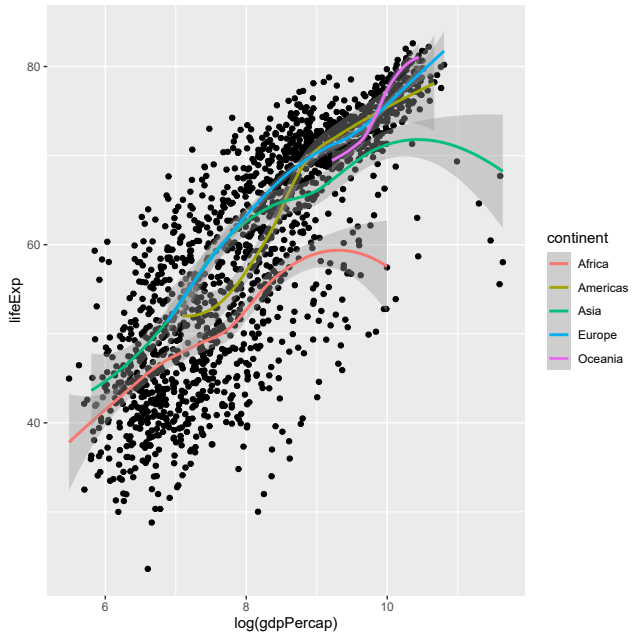
And we can expand this to make scatterplot shows different colour for continent

```
ggplot(data = gapminder, mapping = aes(x = log(gdpPercap), y = lifeExp)) +
  geom_point(mapping = aes(colour = continent)) +
  geom_smooth()
```



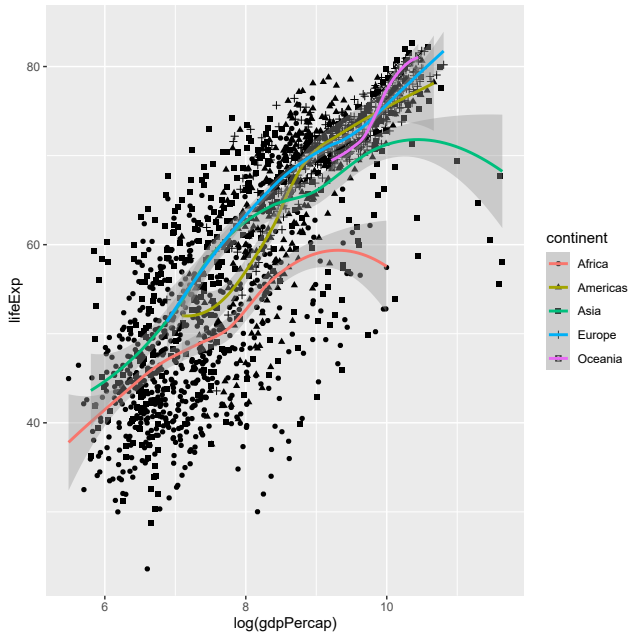
Or expand this to make the smooth plot shows different colour for continent

```
ggplot(data = gapminder, mapping = aes(x = log(gdpPercap), y = lifeExp)) +
  geom_point() +
  geom_smooth(mapping = aes(colour = continent))
```



Or both the scatterplot and the smoothplot

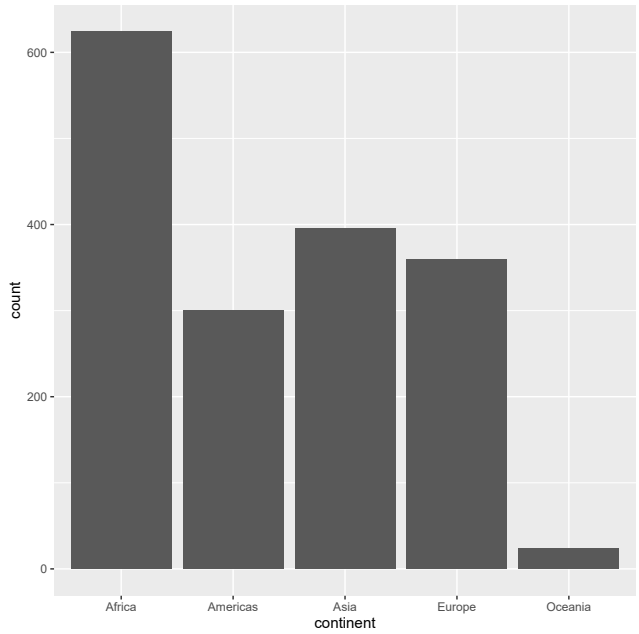
```
ggplot(data = gapminder, mapping = aes(x = log(gdpPercap), y = lifeExp)) +
  geom_point(mapping = aes(shape = continent)) +
  geom_smooth(mapping = aes(colour = continent))
```



2.11 Statistical transformation

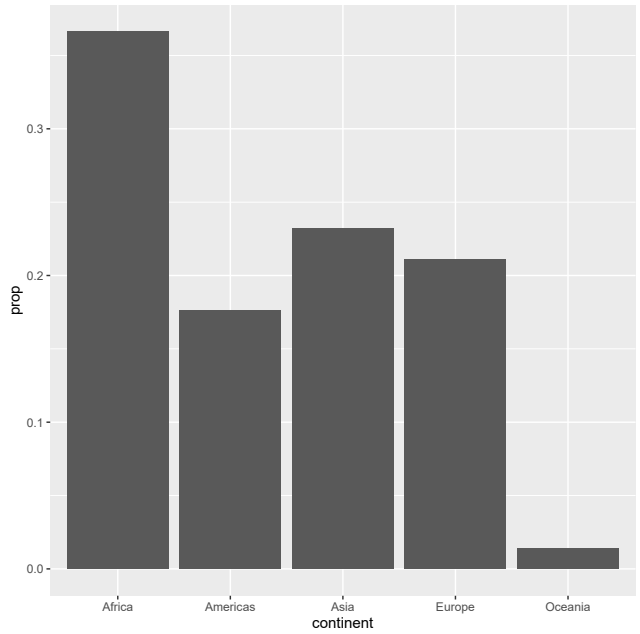
Let us create a bar chart, with y axis as the frequency.

```
ggplot(data = gapminder) +
  geom_bar(mapping = aes(x = continent))
```



If we want the y-axis to show proportion, we can use these codes

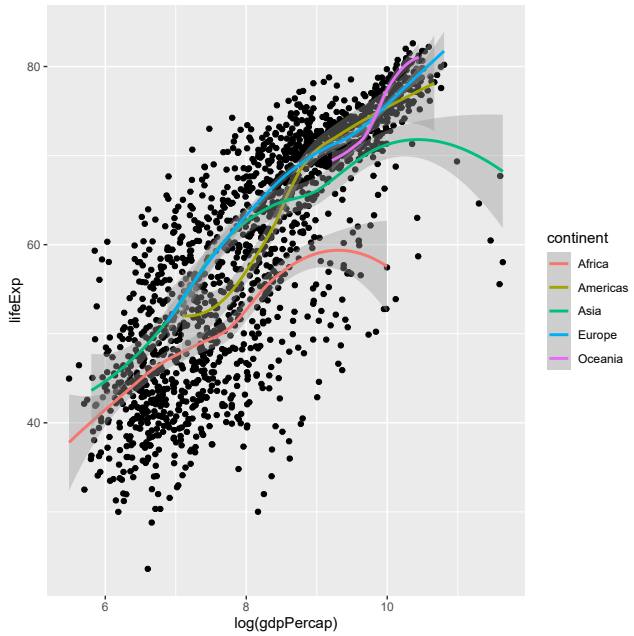
```
ggplot(data = gapminder) +
  geom_bar(mapping = aes(x = continent, y = ..prop..,
                         group = 1))
```



2.12 Customizing title

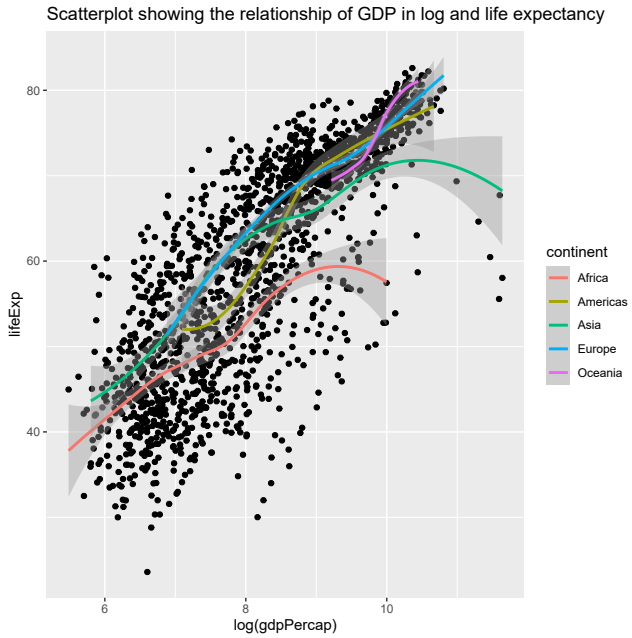
We can customize many aspects of the plot using ggplot package. For example, from gapminder dataset, we choose GDP and log it (to reduce skewness) and life expectancy, and make a scatterplot. We named the plot as `my_pop`

```
mypop <- ggplot(data = gapminder, mapping = aes(x = log(gdpPercap), y = lifeExp)) +
  geom_point() +
  geom_smooth(mapping = aes(colour = continent))
mypop
```



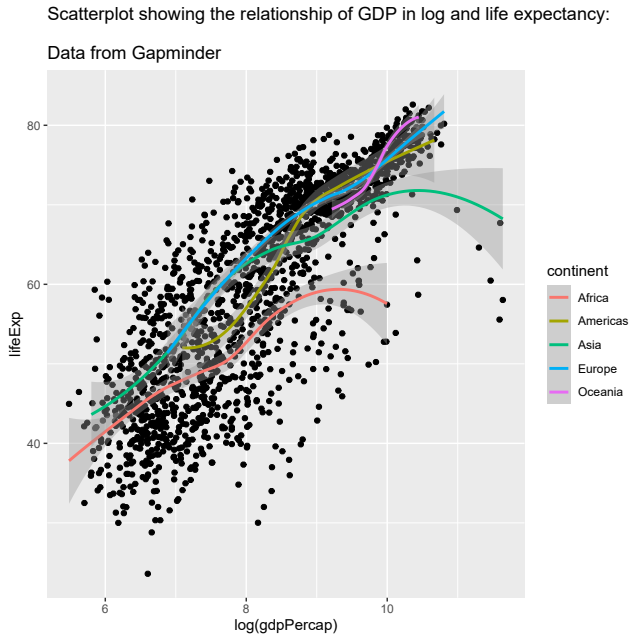
You will notice that there is no title in the plot. A title can be added to the plot.

```
mypop + ggtitle("Scatterplot showing the relationship of GDP in log and life expectancy
```



Title in multiple lines by adding `\n`

```
mypop + ggtitle("Scatterplot showing the relationship of GDP in log and life expectancy:
                \nData from Gapminder")
```

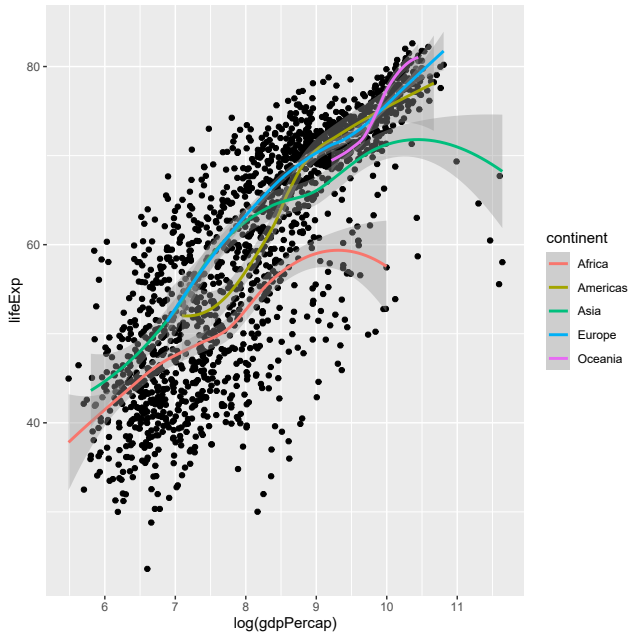


2.13 Adjusting axes

We can specify the tick marks

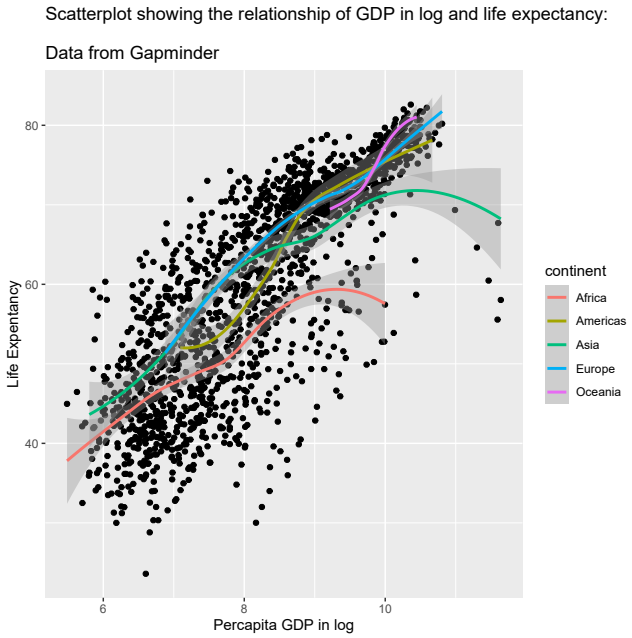
1. min = 0
2. max = 12
3. interval = 1

```
mypop + scale_x_continuous(breaks = seq(0,12,1))
```



And we can label the x-axis and y-axis

```
mypop + ggtitle("Scatterplot showing the relationship of GDP in log and life expectancy:
                \nData from Gapminder") + ylab("Life Expentancy") + xlab("Percapita GDP in log")
```

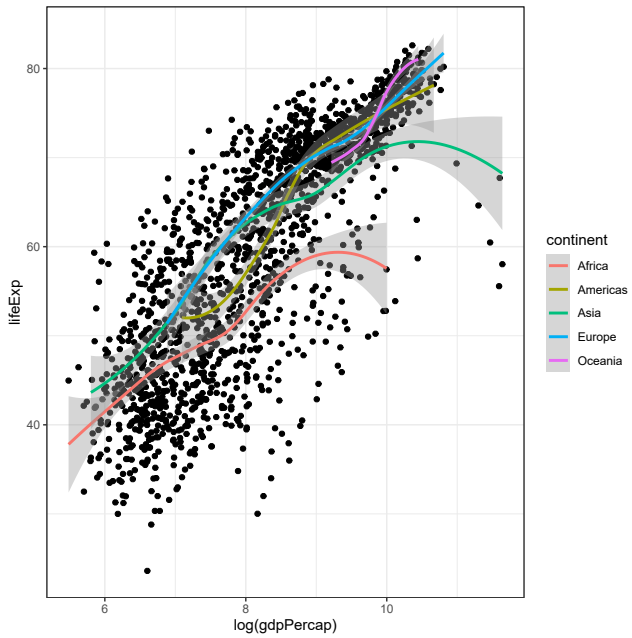


2.14 Choosing theme

The default is gray theme or `theme_gray()`

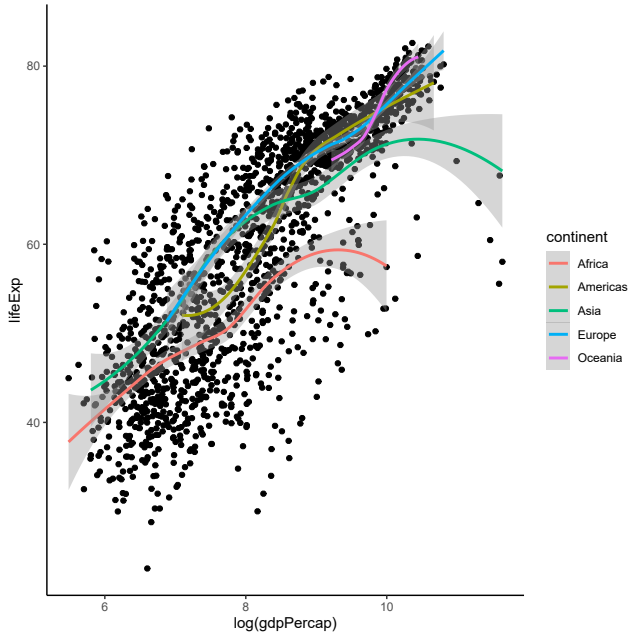
This is the black and white theme

```
mypop + theme_bw()
```



This is the classic theme

```
mypop + theme_classic()
```



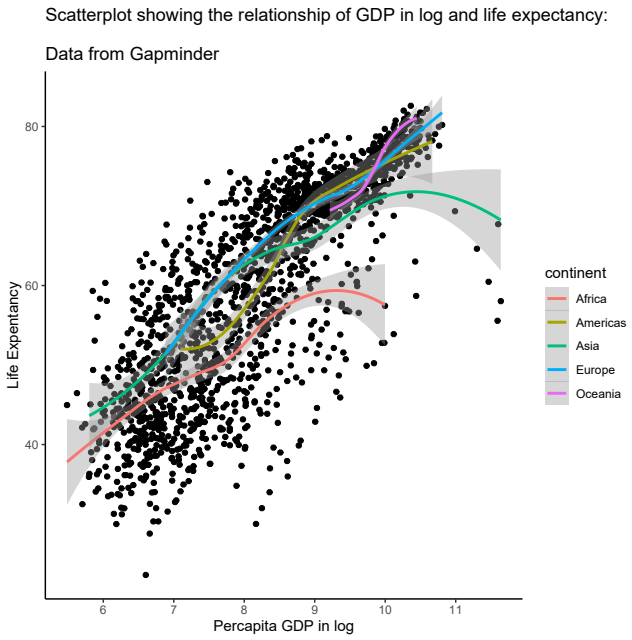
2.15 Saving plot

In R, you can save the plot into different format. You can also set other parameters such as the dpi and the size for the plot. One of the preferred formats for saving a plot is as a PDF format.

2.16 Saving plot using ggplot2

Here, we will show how to save plots in R. In this example, let us use the object for the plot named `mypop` and add a title, an x label, an y label and choose the classic theme,

```
myplot <- mypop +
ggtitle("Scatterplot showing the relationship of GDP in log and life expectancy:
                \nData from Gapminder") + ylab("Life Expentancy") +
  xlab("Percapita GDP in log") +
  scale_x_continuous(breaks = seq(0,12,1)) +
  theme_classic()
myplot
```



We now can see a nice plot. And next, we want to save the plot (currently on the screen) to these formats:

1. `pdf` format
2. `png` format

3. `jpg` format

The codes we can use are:

```
library(here)
ggsave(plot = myplot, here("plots","my_pdf_plot.pdf"))
ggsave(plot = myplot, here("plots","my_png_plot.png"))
ggsave(plot = myplot, here("plots","my_jpg_plot.jpg"))
```

If we want to add more customization before saving the plot, for example, we want to set these parameters:

1. width = 10 cm (or you can use `in`)
2. height = 6 cm (or you can use `in`)
3. dpi = 150. dpi is dots per inch

Now, you can run these codes

```
ggsave(plot = myplot, here('plots','my_pdf_plot2.pdf'),
                            width = 10, height = 6, units = "in",
                            dpi = 150, device = 'pdf')
ggsave(plot = myplot, here('plots','my_png_plot2.png'),
       width = 10, height = 6, units = "cm",
       dpi = 150, device = 'png')
ggsave(plot = myplot, here("plots","my_jpg_plot2.jpg"),
       width = 10, height = 6, units = "cm",
       dpi = 150, device = 'jpg')
```

Chapter 3

Data Wrangling

3.1 Definition of data wrangling

Data wrangling is also known as Data Munging or Data Transformation. It is loosely the process of manually converting or mapping data from one "raw" form into another format. The process allows for more convenient consumption of the data. In doing so, we will be using semi-automated tools in RStudio. You can find more information here https://community.modeanalytics.com/sql/tutorial/data-wrangling-with-sql/

3.2 Data wrangling with dplyr package

3.2.1 dplyr package

dplyr is a package grouped inside **tidyverse** collection of packages. **dplyr** package is a very useful package to munge or wrangle or to tranform your data. It is a grammar of data manipulation. It provides a consistent set of verbs that help you solve the most common data manipulation challenges. This **tidyverse** webpage https://github.com/tidyverse/dplyr has more information and examples.

3.3 Common procedures for doing data transformation

The common data wrangling procedures that data analyst does include:

1. reducing the size of dataset by selecting certain variables (or columns)
2. generating new variable from existing variables
3. sorting observation of a variable

4. grouping observations based on certain criteria
5. reducing variable to groups to in order to estimate summary statistic

3.4 Some dplyr functions

For the procedures listed above, the corresponding **dplyr** functions are

1. `dplyr::select()` - to select a number of variables from a dataframe
2. `dplyr::mutate()` - to generate a new variable from existing variables
3. `dplyr::arrange()` - to sort observation of a variable
4. `dplyr::filter()` - to group observations that fulfil certain criteria
5. `dplyr::group_by()` and `dplyr::summarize()` - to reduce variable to groups in order to provide summary statistic

3.5 Create a new project or set your working directory

It is very important to ensure you know where your working directory is. To do so, the best practice is *is to create a new project everytime you want to start new analysis with R*. To do so, create a new project by `File -> New Project`. If you do not start with a new project, you still need to know **Where is my working directory?**.

So, again we emphasize that every time you want to start processing your data, please make sure:

1. you use R project. It is much easier and cleaner to start your work with a new R project. Once you have done or need to log off your computer, close the project and reopen the project the next time you need to.

2. that if you are not using R project, you are inside the correct working directory. Type `getwd()` to display the active **working directory**. And to set a new working directory use the function `setwd()`. Once you are know where your working directory is, you can start read or import data into your working directory.

Once you are inside the project, you can import your data if necessary. Remember, there are a number of packages you can use to read the data into R. R can read almost all (if not all format) types of data format. For example, we know that common data format are the:

1. SPSS (`.sav`) format,
2. Stata (`.dta`) format,
3. SAS format,
4. MS Excel (`.xlsx`) format
5. Comma-separated-values `.csv` format.

But they are other formats too for examples data in DICOM format. DICOM format data includes data from CT scan and MRI images. There are data in shapefile format to store geographical information. Three packages - **haven**, **readr** and **foreign** packages - are very useful to read or import your data into R memory.

1. **readr** provides a fast and friendly way to read rectangular data (like csv, tsv, and fwf).
2. **readxl** reads .xls and .xlsx sheets.
3. **haven** reads SPSS, Stata, and SAS data.

3.6 *starwars* data

To make life easier and to facilitate reproducibility, we will use examples available from the public domains. We will produce and reproduce the outputs demonstrated on **tidyverse** website (https://github.com/tidyverse/dplyr). One of the useful datasets is `starwars` dataset. The `starwars` data comes together with **dplyr** package. This original source of data is from SWAPI, the Star Wars API accessible at http://swapi.co/.

The `starwars` data is class of `tibble`. The data have:

- 87 rows (observations)
- 13 columns (variables)

Now, let us:

1. load the **tidyverse** package
2. examine the column names (variable names)

Loading **tidyverse** packages will load **dplyr** automatically. If you want to load only **dplyr**, just type `library(dplyr)`.

```
library(dplyr)
```

Take a peek at the `starwars` data

```
glimpse(starwars)
```

```
## Rows: 87
## Columns: 14
## $ name       <chr> "Luke Skywalker", "C-3PO", "R2-D2", "Darth Vader", "Leia Or~
## $ height     <int> 172, 167, 96, 202, 150, 178, 165, 97, 183, 182, 188, 180, 2~
## $ mass       <dbl> 77.0, 75.0, 32.0, 136.0, 49.0, 120.0, 75.0, 32.0, 84.0, 77.~
## $ hair_color <chr> "blond", NA, NA, "none", "brown", "brown, grey", "brown", N~
## $ skin_color <chr> "fair", "gold", "white, blue", "white", "light", "light", "~
## $ eye_color  <chr> "blue", "yellow", "red", "yellow", "brown", "blue", "blue",~
```

```
## $ birth_year <dbl> 19.0, 112.0, 33.0, 41.9, 19.0, 52.0, 47.0, NA, 24.0, 57.0, ~
## $ sex        <chr> "male", "none", "none", "male", "female", "male", "female",~
## $ gender     <chr> "masculine", "masculine", "masculine", "masculine", "femini~
## $ homeworld  <chr> "Tatooine", "Tatooine", "Naboo", "Tatooine", "Alderaan", "T~
## $ species    <chr> "Human", "Droid", "Droid", "Human", "Human", "Human", "Huma~
## $ films      <list> <"The Empire Strikes Back", "Revenge of the Sith", "Return~
## $ vehicles   <list> <"Snowspeeder", "Imperial Speeder Bike">, <>, <>, <>, "Imp~
## $ starships  <list> <"X-wing", "Imperial shuttle">, <>, <>, "TIE Advanced x1",~
```

Next, we examine the first 10 observations of the data. There are 77 more rows NOT SHOWN. You can also see the types of the variables:

1. `chr` (character),
2. `int` (integer),
3. `dbl` (double)

```
starwars
```

```
## # A tibble: 87 x 14
##    name    height  mass hair_color  skin_color eye_color birth_year sex   gender
##    <chr>    <int> <dbl> <chr>       <chr>      <chr>          <dbl> <chr> <chr>
##  1 Luke S~    172    77 blond       fair       blue            19   male  mascu~
##  2 C-3PO      167    75 <NA>        gold       yellow         112   none  mascu~
##  3 R2-D2       96    32 <NA>        white, bl~ red             33   none  mascu~
##  4 Darth ~    202   136 none        white      yellow          41.9 male  mascu~
##  5 Leia O~    150    49 brown       light      brown           19   fema~ femin~
##  6 Owen L~    178   120 brown, grey light      blue            52   male  mascu~
##  7 Beru W~    165    75 brown       light      blue            47   fema~ femin~
##  8 R5-D4       97    32 <NA>        white, red red             NA   none  mascu~
##  9 Biggs ~    183    84 black       light      brown           24   male  mascu~
## 10 Obi-Wa~    182    77 auburn, wh~ fair       blue-gray       57   male  mascu~
## # ... with 77 more rows, and 5 more variables: homeworld <chr>, species <chr>,
## #   films <list>, vehicles <list>, starships <list>
```

3.7 Select variables, generate new variable and rename variable

We will work with these functions

- `dplyr::select()`
- `dplyr::mutate()` and
- `dplyr::rename()`

3.7.1 Select variables using `dplyr::select()`

When you work with large datasets with many columns, sometimes it is easier to select only the necessary columns to reduce the size of dataset. This is possible by creating a smaller dataset (less variables). Then you can work on at the initial part of data analysis with this smaller dataset. This will greatly help data exploration.

To create smaller datasets, select some of the columns (variables) in the dataset. For example, in `starwars` data, we have 13 variables. From this dataset, let us generate a new dataset named as `mysw` with only these 4 variables , 1. name 2. height 3. mass 4. gender

```
mysw <- starwars %>% select(name, gender, height, mass)
mysw
```

```
## # A tibble: 87 x 4
##    name               gender    height  mass
##    <chr>              <chr>      <int> <dbl>
##  1 Luke Skywalker     masculine    172    77
##  2 C-3PO              masculine    167    75
##  3 R2-D2              masculine     96    32
##  4 Darth Vader        masculine    202   136
##  5 Leia Organa        feminine     150    49
##  6 Owen Lars          masculine    178   120
##  7 Beru Whitesun lars feminine     165    75
##  8 R5-D4              masculine     97    32
##  9 Biggs Darklighter  masculine    183    84
## 10 Obi-Wan Kenobi     masculine    182    77
## # ... with 77 more rows
```

The new dataset `mysw` is now created. You can see it in the `Environment` pane.

3.7.2 Generate varibale using `dplyr::mutate()`

With `dplyr::mutate()`, you can generate new variable. For example, in the dataset `mysw`, we want to create a new variable named as `bmi`. This variable equals mass in kg divided by squared height (in meter)

$$bmi = \frac{kg}{m^2}$$

```
mysw <- mysw %>% mutate(bmi = mass/(height/100)^2)
mysw
```

```
## # A tibble: 87 x 5
```

```
##    name               gender    height  mass   bmi
##    <chr>              <chr>      <int> <dbl> <dbl>
##  1 Luke Skywalker     masculine    172    77  26.0
##  2 C-3PO              masculine    167    75  26.9
##  3 R2-D2              masculine     96    32  34.7
##  4 Darth Vader        masculine    202   136  33.3
##  5 Leia Organa        feminine     150    49  21.8
##  6 Owen Lars          masculine    178   120  37.9
##  7 Beru Whitesun lars feminine     165    75  27.5
##  8 R5-D4              masculine     97    32  34.0
##  9 Biggs Darklighter  masculine    183    84  25.1
## 10 Obi-Wan Kenobi     masculine    182    77  23.2
## # ... with 77 more rows
```

Now, your dataset `mysw` has 5 columns (variables). The last variable is `bmi`

3.7.3 Rename variable using `dplyr::rename()`

Now,

1. create a new variable *bmi2* which equals bmi^2.
2. rename *bmi2* to *bmisq*

```
mysw <- mysw %>% mutate(bmi2 = bmi^2)
mysw
```

```
## # A tibble: 87 x 6
##    name               gender    height  mass   bmi  bmi2
##    <chr>              <chr>      <int> <dbl> <dbl> <dbl>
##  1 Luke Skywalker     masculine    172    77  26.0  677.
##  2 C-3PO              masculine    167    75  26.9  723.
##  3 R2-D2              masculine     96    32  34.7 1206.
##  4 Darth Vader        masculine    202   136  33.3 1111.
##  5 Leia Organa        feminine     150    49  21.8  474.
##  6 Owen Lars          masculine    178   120  37.9 1434.
##  7 Beru Whitesun lars feminine     165    75  27.5  759.
##  8 R5-D4              masculine     97    32  34.0 1157.
##  9 Biggs Darklighter  masculine    183    84  25.1  629.
## 10 Obi-Wan Kenobi     masculine    182    77  23.2  540.
## # ... with 77 more rows
```

```
mysw <- mysw %>% rename(bmisq = bmi2)
mysw
```

```
## # A tibble: 87 x 6
##    name               gender    height  mass   bmi bmisq
```

```
##    <chr>              <chr>      <int> <dbl> <dbl> <dbl>
##  1 Luke Skywalker     masculine    172    77  26.0  677.
##  2 C-3PO              masculine    167    75  26.9  723.
##  3 R2-D2              masculine     96    32  34.7 1206.
##  4 Darth Vader        masculine    202   136  33.3 1111.
##  5 Leia Organa        feminine     150    49  21.8  474.
##  6 Owen Lars          masculine    178   120  37.9 1434.
##  7 Beru Whitesun lars feminine     165    75  27.5  759.
##  8 R5-D4              masculine     97    32  34.0 1157.
##  9 Biggs Darklighter  masculine    183    84  25.1  629.
## 10 Obi-Wan Kenobi     masculine    182    77  23.2  540.
## # ... with 77 more rows
```

3.8 Sorting data and select observation

The function `dplyr::arrange()` can sort the data. And the function `dplyr::filter()` allows you to select observations based on your criteria.

3.8.1 Sorting data using `dplyr::arrange()`

We can sort data in ascending or descending order. To do that, we will use `dplyr::arrange()`. It will sort the observation based on the values of the specified variable. For dataset `mysw`, let us sort the `bmi` from the biggest bmi (descending).

```
mysw <- mysw %>% arrange(desc(bmi))
mysw
```

```
## # A tibble: 87 x 6
##    name                  gender    height  mass   bmi   bmisq
##    <chr>                 <chr>      <int> <dbl> <dbl>   <dbl>
##  1 Jabba Desilijic Tiure masculine    175  1358 443.  196629.
##  2 Dud Bolt              masculine     94    45  50.9   2594.
##  3 Yoda                  masculine     66    17  39.0   1523.
##  4 Owen Lars             masculine    178   120  37.9   1434.
##  5 IG-88                 masculine    200   140  35     1225
##  6 R2-D2                 masculine     96    32  34.7   1206.
##  7 Grievous              masculine    216   159  34.1   1161.
##  8 R5-D4                 masculine     97    32  34.0   1157.
##  9 Jek Tono Porkins      masculine    180   110  34.0   1153.
## 10 Darth Vader           masculine    202   136  33.3   1111.
## # ... with 77 more rows
```

Now, we will replace the dataset `mysw` with data that contain the `bmi` values from the lowest to the biggest bmi (ascending).

```
mysw <- mysw %>% arrange(bmi)
mysw
```

```
## # A tibble: 87 x 6
##    name          gender    height  mass   bmi bmisq
##    <chr>         <chr>      <int> <dbl> <dbl> <dbl>
##  1 Wat Tambor    masculine    193    48  12.9  166.
##  2 Adi Gallia    feminine     184    50  14.8  218.
##  3 Sly Moore     <NA>         178    48  15.1  230.
##  4 Roos Tarpals  masculine    224    82  16.3  267.
##  5 Padmé Amidala feminine     165    45  16.5  273.
##  6 Lama Su       masculine    229    88  16.8  282.
##  7 Jar Jar Binks masculine    196    66  17.2  295.
##  8 Ayla Secura   feminine     178    55  17.4  301.
##  9 Shaak Ti      feminine     178    57  18.0  324.
## 10 Barriss Offee feminine     166    50  18.1  329.
## # ... with 77 more rows
```

3.8.2 Select observation using `dplyr::filter()`

To select observations based on certain criteria, we use the `dplyr::filter()` function. Here, in this example, we will create a new dataset (which we will name as `mysw_m_40`) that contains observations with these criteria:

- gender is male AND
- bmi at or above 40

```
mysw_m_40 <- mysw %>% filter(gender == 'male', bmi >= 40)
mysw_m_40
```

```
## # A tibble: 0 x 6
## # ... with 6 variables: name <chr>, gender <chr>, height <int>, mass <dbl>,
## #   bmi <dbl>, bmisq <dbl>
```

Next, we will create a new dataset (named as `mysw_ht_45`) that contain

- `height` above 200 OR `BMI` above 45, AND
- does not include `NA` (which is missing value) observation for `bmi`

```
mysw_ht_45 <- mysw %>% filter(height >200 | bmi >45, bmi != 'NA')
mysw_ht_45
```

```
## # A tibble: 9 x 6
##   name                 gender    height  mass   bmi   bmisq
##   <chr>                <chr>      <int> <dbl> <dbl>   <dbl>
```

```
## 1 Roos Tarpals          masculine    224    82  16.3    267.
## 2 Lama Su               masculine    229    88  16.8    282.
## 3 Tion Medon            masculine    206    80  18.9    355.
## 4 Chewbacca             masculine    228   112  21.5    464.
## 5 Tarfful               masculine    234   136  24.8    617.
## 6 Darth Vader           masculine    202   136  33.3   1111.
## 7 Grievous              masculine    216   159  34.1   1161.
## 8 Dud Bolt              masculine     94    45  50.9   2594.
## 9 Jabba Desilijic Tiure masculine    175  1358 443.  196629.
```

3.9 Group data and get summary statistics

The`dplyr::group_by()` function allows us to group data based on categorical variable. Using the `dplyr::summarize` we do summary statistics for the overall data or for groups created using `group_by()` function.

3.9.1 Group data using `dplyr::group_by()`

The `group_by` function will prepare the data for group analysis. For example,

1. to get summary values for mean `bmi`, mean `ht` and mean `mass`
2. for male, female, hermaphrodite and none (`gender` variable)

```
mysw_g <- mysw %>% group_by(gender)
mysw_g
```

```
## # A tibble: 87 x 6
## # Groups:   gender [3]
##    name          gender    height  mass   bmi bmisq
##    <chr>         <chr>      <int> <dbl> <dbl> <dbl>
##  1 Wat Tambor    masculine    193    48  12.9  166.
##  2 Adi Gallia    feminine     184    50  14.8  218.
##  3 Sly Moore     <NA>         178    48  15.1  230.
##  4 Roos Tarpals  masculine    224    82  16.3  267.
##  5 Padmé Amidala feminine     165    45  16.5  273.
##  6 Lama Su       masculine    229    88  16.8  282.
##  7 Jar Jar Binks masculine    196    66  17.2  295.
##  8 Ayla Secura   feminine     178    55  17.4  301.
##  9 Shaak Ti      feminine     178    57  18.0  324.
## 10 Barriss Offee feminine     166    50  18.1  329.
## # ... with 77 more rows
```

3.9.2 Summary statistic using `dplyr::summarize()`

Now that we have a group data named `mysw_g`, now, we would summarize our data using the mean and standard deviation (SD) for the groups specified above.

```
mysw_g %>% summarise(meanbmi = mean(bmi, na.rm = TRUE),
                     meanht  = mean(height, na.rm = TRUE),
                     meanmass = mean(mass, na.rm = TRUE),
                     sdbmi = sd(bmi, na.rm = TRUE),
                     sdht = sd(height, na.rm = TRUE),
                     sdmass = sd(mass, na.rm = TRUE))
```

```
## # A tibble: 3 x 7
##   gender    meanbmi meanht meanmass sdbmi  sdht sdmass
##   <chr>       <dbl>  <dbl>    <dbl> <dbl> <dbl>  <dbl>
## 1 feminine     19.2   165.     54.7  3.69 23.6    8.59
## 2 masculine    34.7   177.    106.  60.0  37.6  185.
## 3 <NA>         15.1   181.     48   NA     2.89  NA
```

To calculate the frequencies for one variable

```
freq_species <- starwars %>% count(species, sort = TRUE)
freq_species
```

```
## # A tibble: 38 x 2
##    species      n
##    <chr>    <int>
##  1 Human       35
##  2 Droid        6
##  3 <NA>         4
##  4 Gungan       3
##  5 Kaminoan     2
##  6 Mirialan     2
##  7 Twi'lek      2
##  8 Wookiee      2
##  9 Zabrak       2
## 10 Aleena       1
## # ... with 28 more rows
```

And to calculate the frequencies for two variables. You can add the categorical or factor variable to get the frequencies for more groups.

```
freq_species_home <- starwars %>% count(species, homeworld, sort = TRUE)
freq_species_home
```

```
## # A tibble: 58 x 3
```

```
##    species  homeworld     n
##    <chr>    <chr>     <int>
##  1 Human    Tatooine      8
##  2 Human    Naboo         5
##  3 Human    <NA>          5
##  4 Droid    <NA>          3
##  5 Gungan   Naboo         3
##  6 Human    Alderaan      3
##  7 Droid    Tatooine      2
##  8 Human    Corellia      2
##  9 Human    Coruscant     2
## 10 Kaminoan Kamino        2
## # ... with 48 more rows
```

3.10 More complicated dplyr verbs

To be more efficent, use multiple **dplyr** functions in one line of R code. For example,

```
starwars %>% filter(gender == "male", height > 100, mass > 100) %>%
  select(height, mass, species) %>%
  group_by(species) %>%
  summarize(mean_ht = mean(height, na.rm = TRUE),
            mean_mass = mean(mass, na.rm = TRUE),
            freq = n())
```

```
## # A tibble: 0 x 4
## # ... with 4 variables: species <chr>, mean_ht <dbl>, mean_mass <dbl>,
## #   freq <int>
```

3.11 Data transformation for categorical variables

3.11.1 forcats package

Data transformation for categorical variables (factor variables) can be facilitated using the **forcats** package.

3.11.2 Create a dataset

Let us create create a dataset to demonstrate **forcats** package. The dataset will contain

1. a vector column named as **sex1** , values = 0,1

2. a vector column named as **race1** , values = 1,2,3,4
3. a tibble dataframe (dataset) named as **data_f**

```
sex1 <- rbinom(n = 100, size = 1, prob = 0.5)
str(sex1)
```

```
##  int [1:100] 1 0 1 1 1 0 0 0 1 1 ...
```

```
race1 <- rep(seq(1:4), 25)
str(race1)
```

```
##  int [1:100] 1 2 3 4 1 2 3 4 1 2 ...
```

```
data_f <- tibble(sex1, race1)
head(data_f)
```

```
## # A tibble: 6 x 2
##    sex1 race1
##   <int> <int>
## 1     1     1
## 2     0     2
## 3     1     3
## 4     1     4
## 5     1     1
## 6     0     2
```

Now let us see the structure of the dataset. You should see that they are all in the integer (numerical) format

```
str(data_f)
```

```
## tibble [100 x 2] (S3: tbl_df/tbl/data.frame)
##  $ sex1 : int [1:100] 1 0 1 1 1 0 0 0 1 1 ...
##  $ race1: int [1:100] 1 2 3 4 1 2 3 4 1 2 ...
```

3.11.3 Conversion from numeric to factor variables

Now, we will convert the integer (numerical) variable to a factor (categorical) variable. For example, we will generate a new factor (categorical) variable named as `male` from variable `sex1` (which is an integer variable). We will label `male`as *No* or *Yes*.

We then generate a new factor (categorical) variable named as `race2` from `race1` (which is an integer variable) and label as *Mal, Chi, Ind, Others*

```
data_f$male <- factor(data_f$sex1, labels = c('No', 'Yes'))
data_f$race2 <- factor(data_f$race1, labels = c('Mal', 'Chi', 'Ind', 'Others'))
str(data_f)
```

```
## tibble [100 x 4] (S3: tbl_df/tbl/data.frame)
##  $ sex1 : int [1:100] 1 0 1 1 1 0 0 0 1 1 ...
##  $ race1: int [1:100] 1 2 3 4 1 2 3 4 1 2 ...
##  $ male : Factor w/ 2 levels "No","Yes": 2 1 2 2 2 1 1 1 2 2 ...
##  $ race2: Factor w/ 4 levels "Mal","Chi","Ind",..: 1 2 3 4 1 2 3 4 1 2 ...
```

```
head(data_f) ; tail(data_f)
```

```
## # A tibble: 6 x 4
##    sex1 race1 male  race2
##   <int> <int> <fct> <fct>
## 1     1     1 Yes   Mal
## 2     0     2 No    Chi
## 3     1     3 Yes   Ind
## 4     1     4 Yes   Others
## 5     1     1 Yes   Mal
## 6     0     2 No    Chi
```

```
## # A tibble: 6 x 4
##    sex1 race1 male  race2
##   <int> <int> <fct> <fct>
## 1     1     3 Yes   Ind
## 2     1     4 Yes   Others
## 3     1     1 Yes   Mal
## 4     0     2 No    Chi
## 5     1     3 Yes   Ind
## 6     1     4 Yes   Others
```

3.11.4 Recoding variables using `forcats::fct_recode()`

We use this function to recode variables from the old levels to the new levels. Let us say that our objectives is to:

1. modify the levels for variable **male** by changing the levels from `No` vs `Yes` to `Fem` and `Male`, respectively.
2. create a new variable **malay** from variable **race2**. We then want to label `Chi` to `Non-Malay`, `Ind` to `Non-Malay` and `Others` to `Non-Malay`. But we keep `Mal` as it is.

To do that, use the codes below,

```
library(forcats)
data_f$male2 <- data_f$male %>% fct_recode('Fem' = 'No', 'Male' = 'Yes')
data_f <- data_f %>% mutate(malay = fct_recode(race2,
                                               'Non-Malay' = 'Chi',
                                               'Non-Malay' = 'Ind',
                                               'Non-Malay' = 'Others'))
head(data_f) ; tail(data_f)
```

```
## # A tibble: 6 x 6
##    sex1 race1 male  race2  male2 malay
##   <int> <int> <fct> <fct>  <fct> <fct>
## 1     1     1 Yes   Mal    Male  Mal
## 2     0     2 No    Chi    Fem   Non-Malay
## 3     1     3 Yes   Ind    Male  Non-Malay
## 4     1     4 Yes   Others Male  Non-Malay
## 5     1     1 Yes   Mal    Male  Mal
## 6     0     2 No    Chi    Fem   Non-Malay
```

```
## # A tibble: 6 x 6
##    sex1 race1 male  race2  male2 malay
##   <int> <int> <fct> <fct>  <fct> <fct>
## 1     1     3 Yes   Ind    Male  Non-Malay
## 2     1     4 Yes   Others Male  Non-Malay
## 3     1     1 Yes   Mal    Male  Mal
## 4     0     2 No    Chi    Fem   Non-Malay
## 5     1     3 Yes   Ind    Male  Non-Malay
## 6     1     4 Yes   Others Male  Non-Malay
```

3.12 Summary

dplyr package is a very useful package that encourages users to use proper verb when manipulating variables (columns) and observations (rows). We have learned to use 5 functions but there are more functions available. Other useful functions include:

1. `dplyr::distinct()`
2. `dplyr::transmutate()`
3. `dplyr::sample_n()` and `dplyr::sample_frac()`

Also note that, package **dplyr** is very useful when it is combined with another function that is `group_by`

If you working with database, you can use **dbplyr** which has been developed to perform very effectively with databases.

For categorical variables, you can use **forcats** package.

3.13 Self-practice

If you have completed the tutorial above, you may:

1. Read your own data (hints: **haven**, **foreign**) or you can download data from https://www.kaggle.com/datasets . Maybe can try this dataset https://www.kaggle.com/blastchar/telco-customer-churn
2. Create a smaller dataset by selecting some variable (hints: `dplyr::select()`)
3. Creating a dataset with some selection (hints: `dplyr::filter()`)
4. Generate a new variable (hints`dplyr::mutate()`)
5. Creata an object using pipe and combining `dplyr::select()`, `dplyr::filter()` and `dplyr::mutate()` in one single line of R code
6. Summarise the mean, standard deviation and median for numerical variables `dplyr::group_by()` and `dplyr::summarize()`
7. Calculare the number of observations for categorical variables (hints: `dplyr::count()`)
8. Recode a categorical variable (hints: `forcats::fct_recode()`)

3.14 References

1. dplyr vignettes here https://cran.r-project.org/web/packages/dplyr/vignettes/dplyr.html
2. forcats examples here http://r4ds.had.co.nz/factors.html
3. reading data into R https://garthtarr.github.io/meatR/rio.html

Chapter 4

Exploratory data analysis (EDA)

4.1 Motivation

In statistics, exploratory data analysis (EDA) is an approach in data analysis in order to summarize their main characteristics, often with visual methods.

A statistical model can be used or not, but primarily EDA is for seeing what the data can tell us beyond the formal modeling or hypothesis testing task.

Exploratory data analysis was promoted by John Tukey to encourage statisticians to explore the data, and possibly formulate hypotheses that could lead to new data collection

Source: https://en.wikipedia.org/wiki/Exploratory_data_analysis

4.2 Questions to ask before making graphs

You must ask yourselves these:

- Which data do I want to use? `data =`
- Which variable or variables do I want to plot? `mapping = aes()`
- What is (or are) the type of that variable?
  - Are they factor (categorical) variables ?
  - Are they numerical variables?
- Am I going to plot
  - a single variable?
  - two variables together?
  - three variables together?
- What plot? `geom_`

4.3 EDA using ggplot2 package

4.3.1 Usage of ggplot2

1. start with `ggplot()`
2. supply a dataset `data =`
3. and aesthetic mapping (with `aes()`) - variables
4. add on layers

- geom_X : `geom_point()`, `geom_histogram()`
- scales (like `scale_colour_brewer()`)
- faceting specifications (like `facet_wrap()`)
- coordinate systems (like `coord_flip()`).

4.4 Loading the library

4.4.1 tidyverse package for EDA

We will load the **tidyverse** library

The tidyverse is an opinionated collection of R packages designed for data science. All packages share an underlying design philosophy, grammar, and data structures.

https://www.tidyverse.org/

```
library(tidyverse)
```

4.4.2 the gapminder for dataset

And load **gapminder** package to use the *gapminder* dataset

Excerpt from the Gapminder data. The main object in this package is the gapminder data frame or *tibble*.

There are other goodies in the package, such as the data in tab delimited form, a larger unfiltered dataset, premade color schemes for the countries and continents, and ISO 3166-1 country codes.

Variables:

- country
- continent
- year
- lifeExp: life expectancy at birth
- pop: total population
- gdpPercap: per-capita GDP

Load the library

```
library(gapminder)
```

Peek at the data

```
glimpse(gapminder)
```

```
## Rows: 1,704
## Columns: 6
## $ country   <fct> "Afghanistan", "Afghanistan", "Afghanistan", "Afghanistan", ~
## $ continent <fct> Asia, Asia, Asia, Asia, Asia, Asia, Asia, Asia, Asia, Asia, ~
## $ year      <int> 1952, 1957, 1962, 1967, 1972, 1977, 1982, 1987, 1992, 1997, ~
## $ lifeExp   <dbl> 28.801, 30.332, 31.997, 34.020, 36.088, 38.438, 39.854, 40.8~
## $ pop       <int> 8425333, 9240934, 10267083, 11537966, 13079460, 14880372, 12~
## $ gdpPercap <dbl> 779.4453, 820.8530, 853.1007, 836.1971, 739.9811, 786.1134, ~
```

Summary statistics of data

```
summary(gapminder)
```

```
##         country        continent        year         lifeExp
##  Afghanistan:  12   Africa  :624   Min.   :1952   Min.   :23.60
##  Albania    :  12   Americas:300   1st Qu.:1966   1st Qu.:48.20
##  Algeria    :  12   Asia    :396   Median :1980   Median :60.71
##  Angola     :  12   Europe  :360   Mean   :1980   Mean   :59.47
##  Argentina  :  12   Oceania : 24   3rd Qu.:1993   3rd Qu.:70.85
##  Australia  :  12                  Max.   :2007   Max.   :82.60
##  (Other)    :1632
##       pop              gdpPercap
##  Min.   :6.001e+04   Min.   :   241.2
##  1st Qu.:2.794e+06   1st Qu.:  1202.1
##  Median :7.024e+06   Median :  3531.8
##  Mean   :2.960e+07   Mean   :  7215.3
##  3rd Qu.:1.959e+07   3rd Qu.:  9325.5
##  Max.   :1.319e+09   Max.   :113523.1
##
```

More information about *gapminder* data is here https://www.gapminder.org/

We can see the top of our *gapminder* data and the bottom of our *gapminder* data

```
head(gapminder) ; tail(gapminder)
```

```
## # A tibble: 6 x 6
##   country     continent  year lifeExp      pop gdpPercap
##   <fct>       <fct>     <int>   <dbl>    <int>     <dbl>
## 1 Afghanistan Asia       1952    28.8  8425333      779.
## 2 Afghanistan Asia       1957    30.3  9240934      821.
## 3 Afghanistan Asia       1962    32.0 10267083      853.
## 4 Afghanistan Asia       1967    34.0 11537966      836.
## 5 Afghanistan Asia       1972    36.1 13079460      740.
## 6 Afghanistan Asia       1977    38.4 14880372      786.
```

```
## # A tibble: 6 x 6
##   country  continent  year lifeExp      pop gdpPercap
##   <fct>    <fct>     <int>   <dbl>    <int>     <dbl>
## 1 Zimbabwe Africa     1982    60.4  7636524      789.
## 2 Zimbabwe Africa     1987    62.4  9216418      706.
## 3 Zimbabwe Africa     1992    60.4 10704340      693.
## 4 Zimbabwe Africa     1997    46.8 11404948      792.
## 5 Zimbabwe Africa     2002    40.0 11926563      672.
## 6 Zimbabwe Africa     2007    43.5 12311143      470.
```

4.5 One variable: Distribution of a categorical variable

4.5.1 Bar chart

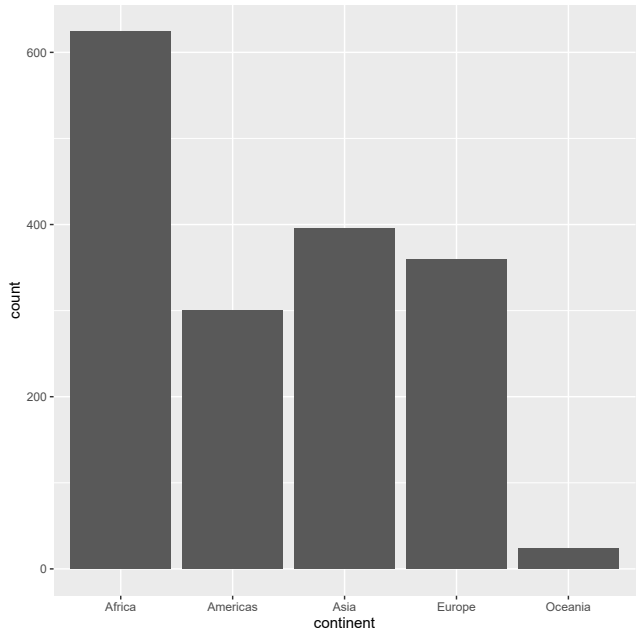
The frequencies (number of countries) based on continents

```
gapminder %>% group_by(continent) %>% summarise(freq = n())
```

```
## # A tibble: 5 x 2
##   continent  freq
##   <fct>     <int>
## 1 Africa      624
## 2 Americas    300
## 3 Asia        396
## 4 Europe      360
## 5 Oceania      24
```

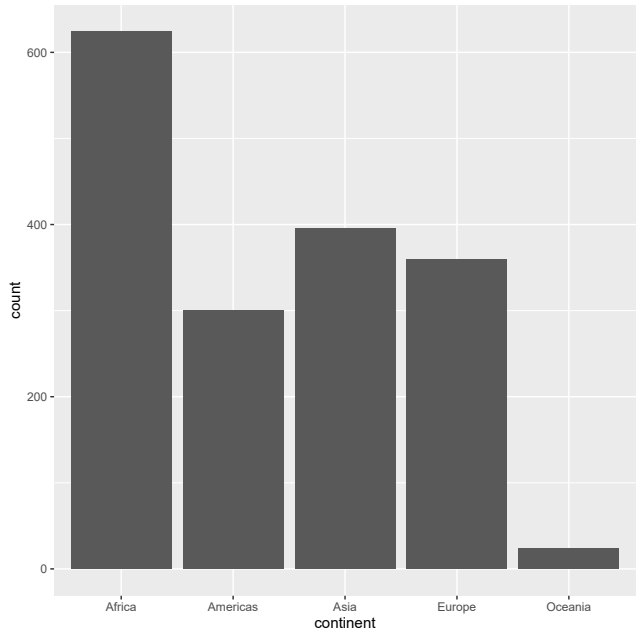
To examine the distribution of a categorical variable, we can use a bar chart.

```
ggplot(data = gapminder) +
  geom_bar(mapping = aes(x = continent))
```



But we can also pass the `aes()` to ggplot

```
ggplot(data = gapminder, mapping = aes(x = continent)) +
  geom_bar()
```



Combining dplyr and ggplot

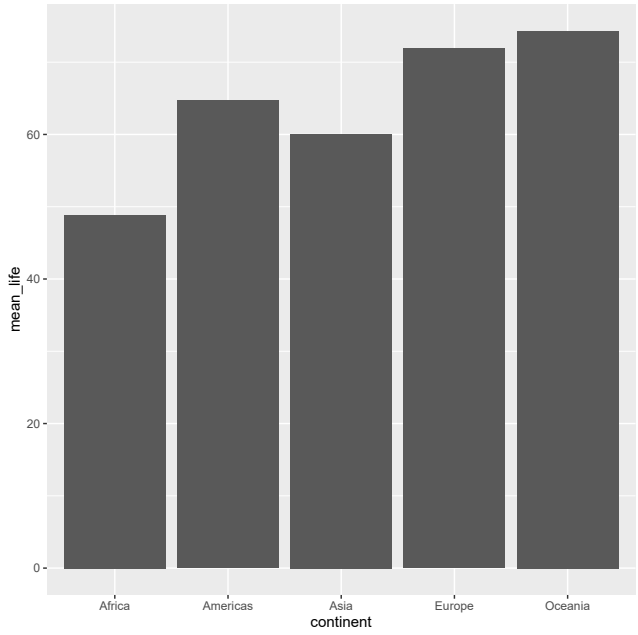
dplyr part:

```
gapminder_life <- gapminder %>% group_by(continent) %>%
  summarize(mean_life = mean(lifeExp))
gapminder_life
```

```
## # A tibble: 5 x 2
##   continent mean_life
##   <fct>         <dbl>
## 1 Africa         48.9
## 2 Americas       64.7
## 3 Asia           60.1
## 4 Europe         71.9
## 5 Oceania        74.3
```

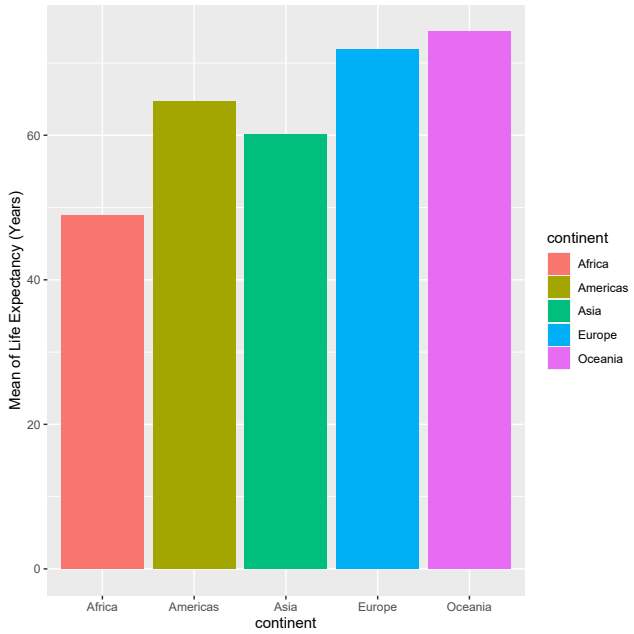
ggplot part:

```
ggplot(gapminder_life, mapping = aes(x = continent, y = mean_life)) + geom_col()
```



dplyr and ggplot in action:

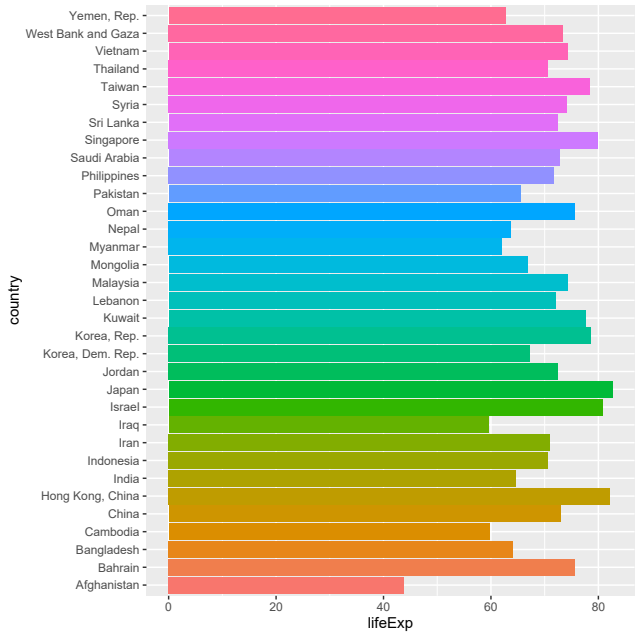
```
gapminder %>% group_by(continent) %>% summarize(mean_life = mean(lifeExp)) %>%
  ggplot(mapping = aes(x = continent, y = mean_life, fill = continent)) + geom_col() +
  ylab("Mean of Life Expectancy (Years)")
```



Personalize the plot

- using `coord_flip()`

```
gapminder %>% filter(continent == "Asia", year == 2007) %>% arrange(lifeExp) %>%
  ggplot(mapping = aes(x = country, y = lifeExp, fill = country)) +
  geom_col(position = "dodge", show.legend = FALSE) +
  coord_flip()
```



Excellent resource from this website http://www.cookbook-r.com/Graphs/Bar_and_line_graphs_(ggplot2)/. It is a must go to website!!

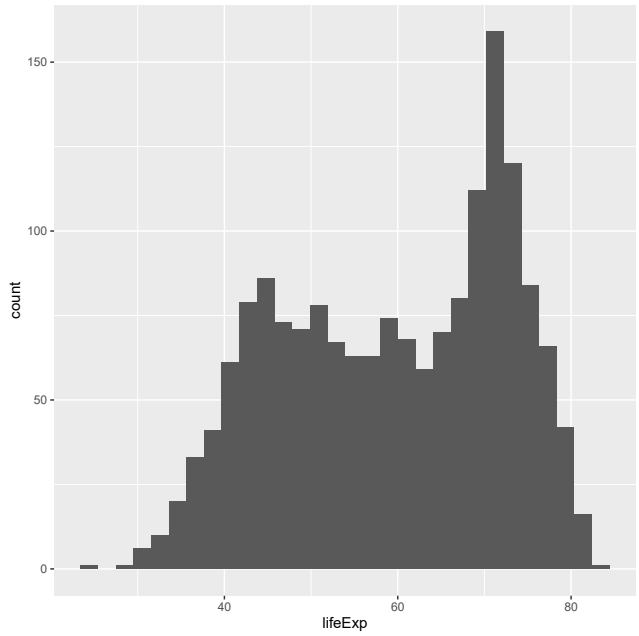
4.6 One variable: Distribution of a numerical variable

Plot distribution of values of a numerical variable

4.6.1 Histogram

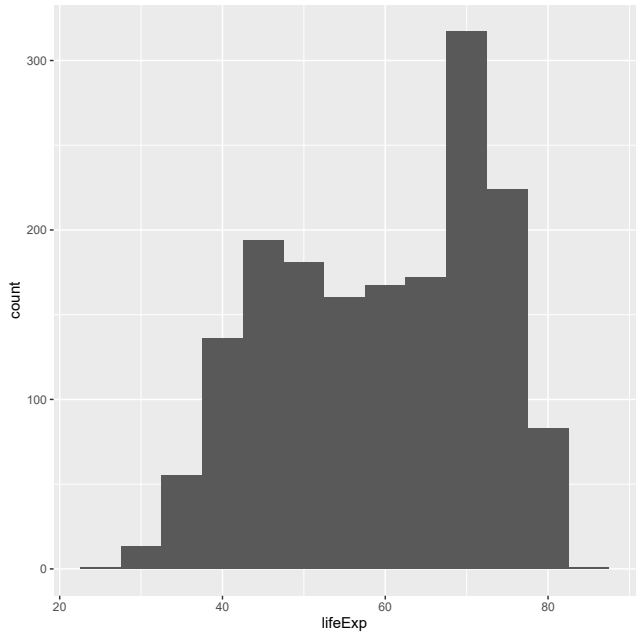
To see the distribution of a numerical variable, we can plot a histogram. And this is bu using the default number of bin widths.

```
ggplot(data = gapminder, mapping = aes(x = lifeExp)) + geom_histogram()
```



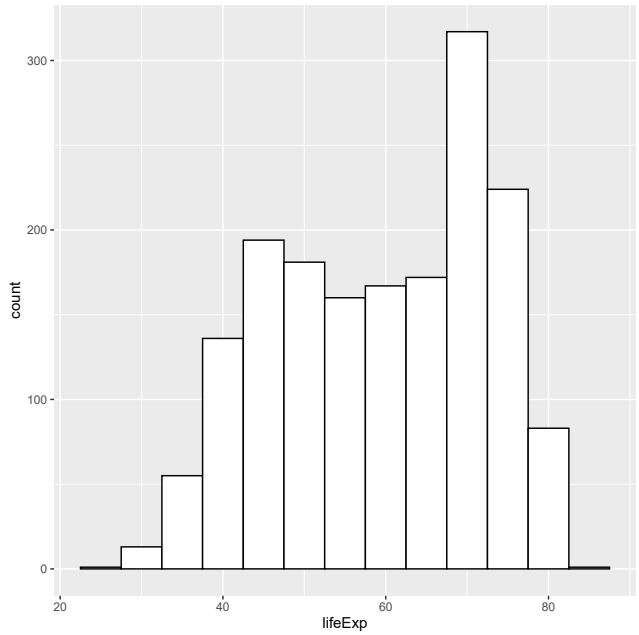
To specify the number of bin, we can use bindwidth

```
ggplot(data = gapminder, mapping = aes(x = lifeExp)) +
  geom_histogram(binwidth = 5)
```



ggplot2 has lots of flexibility and personalization. For example, the histogram above is very plain. We can improve it by setting the line color and fill color, the theme, the size, the symbols and many other parameters.

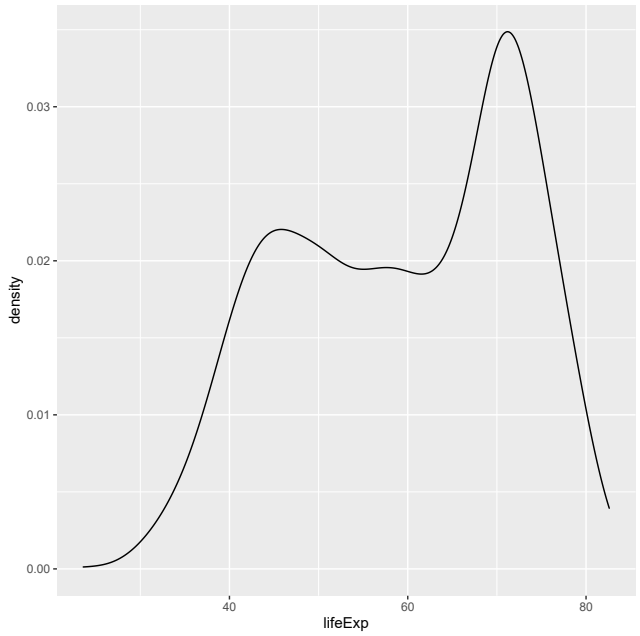
```
ggplot(data = gapminder, mapping = aes(x = lifeExp)) +
  geom_histogram(binwidth = 5, colour = "black", fill = "white")
```



4.6.2 Density curve

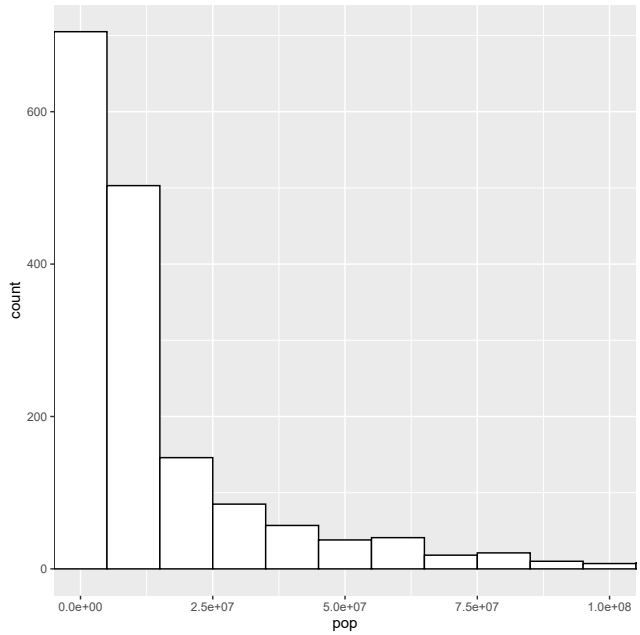
Let us create a density curve. Density curve is useful to examine the distribution of observations.

```
ggplot(data = gapminder, mapping = aes(x = lifeExp)) + geom_density()
```



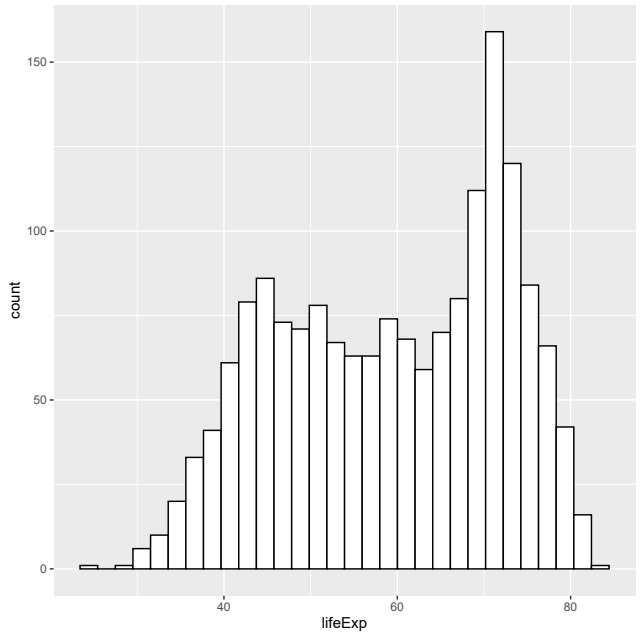
Very skewed. Let us focus (zoom) to the x axis

```
ggplot(data = gapminder, mapping = aes(x = pop)) +
  geom_histogram(binwidth = 10000000, colour = "black", fill = "white") +
  coord_cartesian(xlim = c(0, 100000000))
```



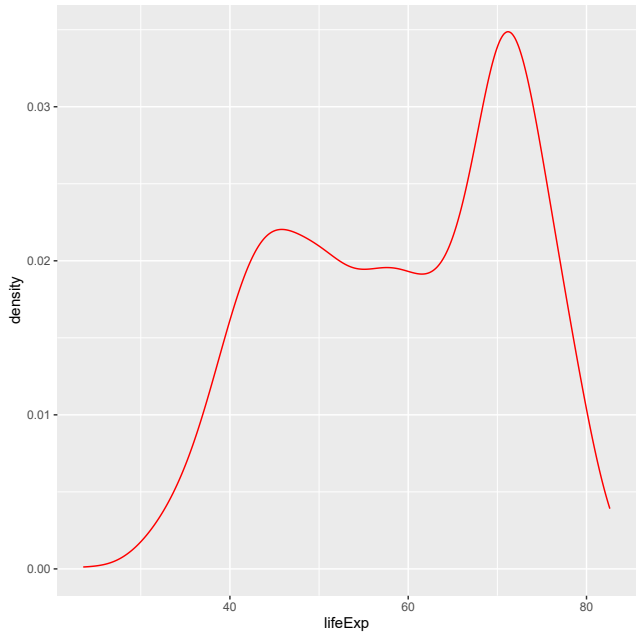
Histogram for life expectancy

```
ggplot(data = gapminder, mapping = aes(x = lifeExp)) +
  geom_histogram(fill = "white", colour = "black")
```



Density curve for life expectancy

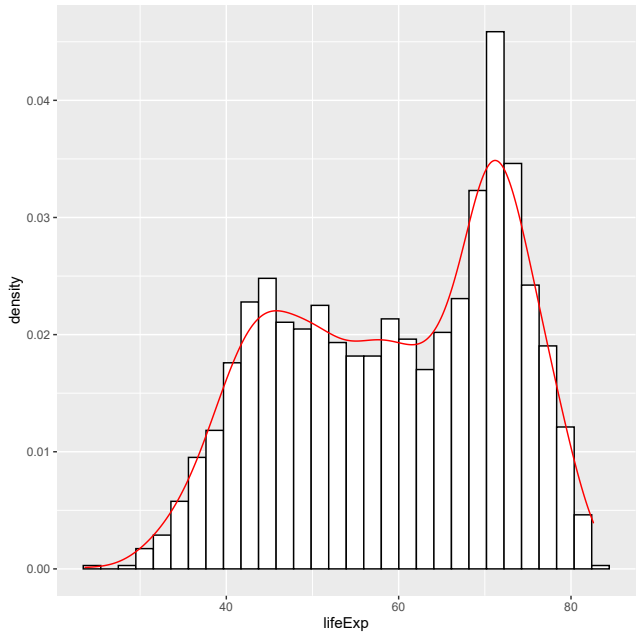
```
ggplot(data = gapminder, mapping = aes(x = lifeExp)) +
  geom_density(colour = "red")
```



4.6.3 Histogram and density curve together

If we want to display both the histogram and the density curve, we can use `geom_histogram()` and `geom_density()` in the single line of codes.

```
ggplot(data = gapminder, mapping = aes(x = lifeExp)) +
  geom_histogram(mapping = aes(y = ..density..),
                 colour = "black", fill = "white") +
  geom_density(colour = "red")
```

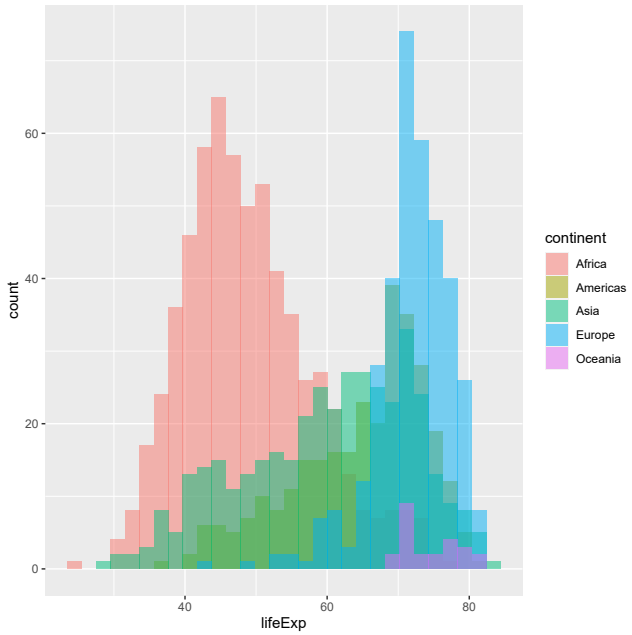


4.7 Two variables: Plotting a numerical and a categorical variable

4.7.1 Overlaying histogramss

Now, examine the distribution of a numerical variable (var **pop**) based on variable **continent** by overlaying histograms.

```
ggplot(data = gapminder, aes(x = lifeExp, fill = continent)) +
    geom_histogram(position = "identity", alpha = 0.5)
```

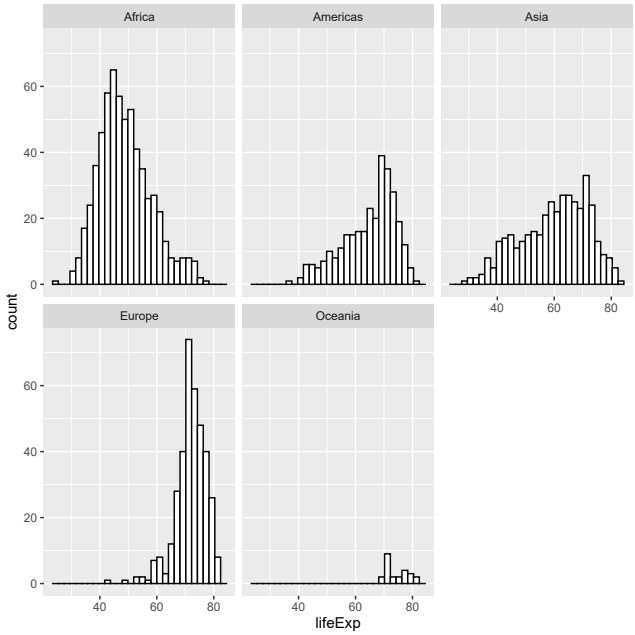


4.7.2 Using facet

We can see better plots if we splot the histogram based on certain grouping.

In this example, we stratify the distibution of life expectancy (a numerical variable) based on continent (a categorical variable)

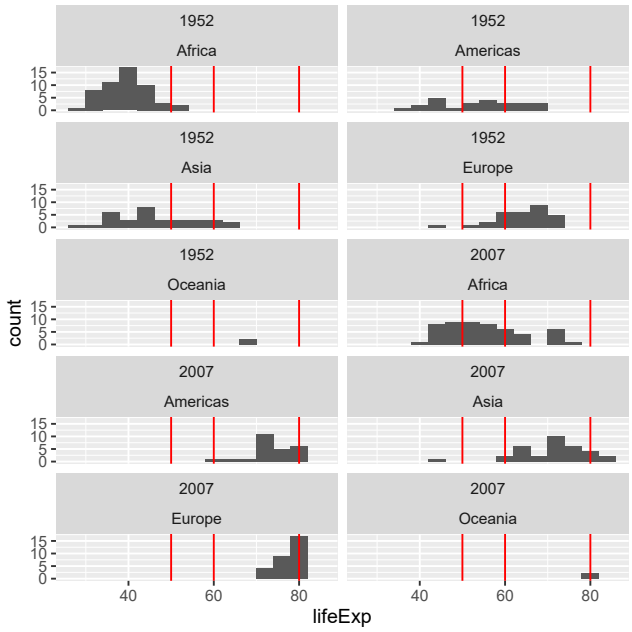
```
ggplot(data = gapminder, aes(x = lifeExp)) +
        geom_histogram(position = "identity", fill = "white", colour = "black") +
        facet_wrap(~ continent)
```



We can draw histogram life expectancy, and this plot can be split based on year and continent. We will also add a vertical line using `geom_vline()`.

But, first, we use filter (part of **dplyr** verb) to choose observations from variable year that equal year 1952 and 2007. We want to reduce the number of plots created afterwards. We named this data as `gapminder_52_07`

```
gapminder_52_07 <- gapminder %>% filter(year == 1952 | year == 2007)
ggplot(data = gapminder_52_07) +
  geom_histogram(mapping = aes(x = lifeExp), binwidth = 4) +
  facet_wrap(~ year + continent, nrow = 5) +
  geom_vline(xintercept = c(50, 60, 80), colour = "red")
```



4.8 Box plot

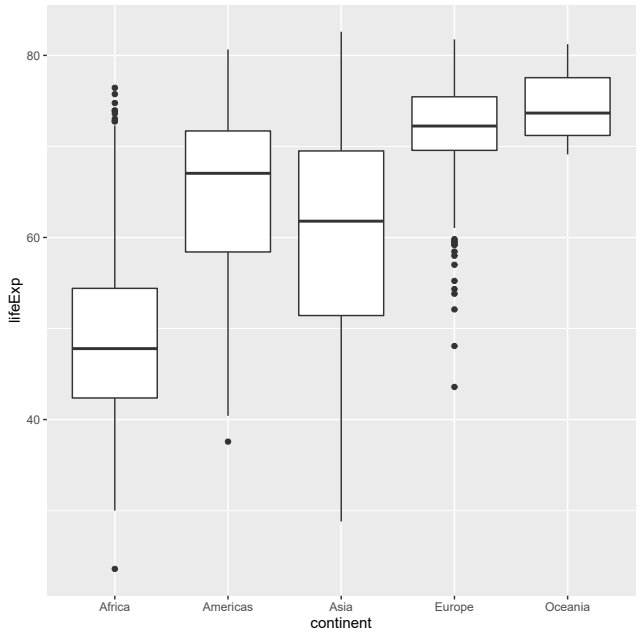
Box plot contains a box and whiskers. The box goes from the 25th percentile to the 75th percentile of the data, also known as the inter-quartile range (IQR).

There's a line indicating the median, or the 50th percentile of the data.

The whiskers start from the edge of the box and extend to the furthest data point that is within 1.5 times the IQR.

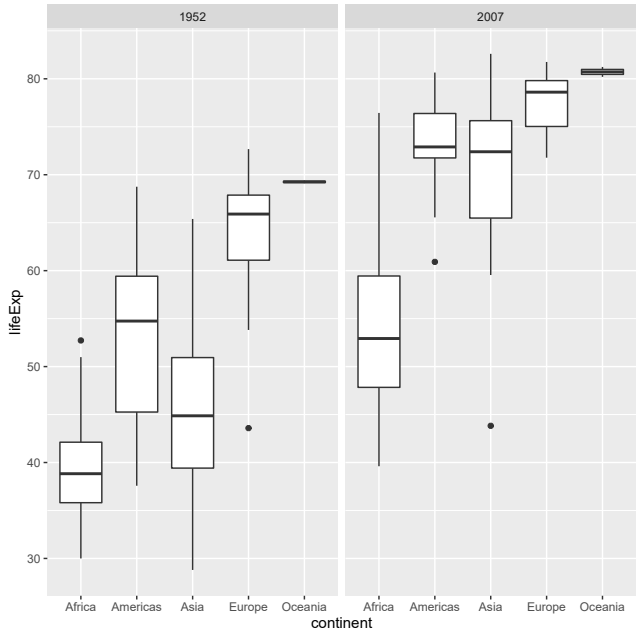
Outliers are points that stay beyond the end of whiskers

```
ggplot(data = gapminder, mapping = aes(x = continent, y = lifeExp)) +
  geom_boxplot()
```



For 1952 and 2007

```
ggplot(data = gapminder_52_07, mapping = aes(x = continent, y = lifeExp)) +
  geom_boxplot() +
  facet_wrap(~ year)
```



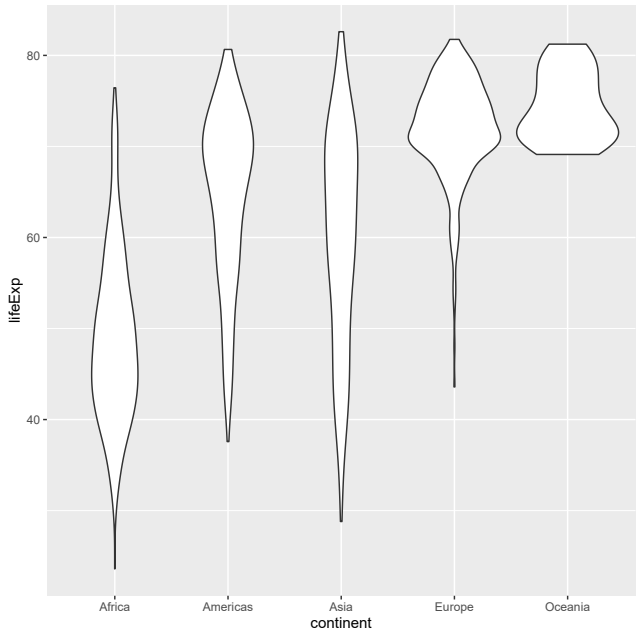
4.9 Violin plot

You want to make a violin plot to compare density estimates of different groups.

Violin plots are a way of comparing multiple data distributions.

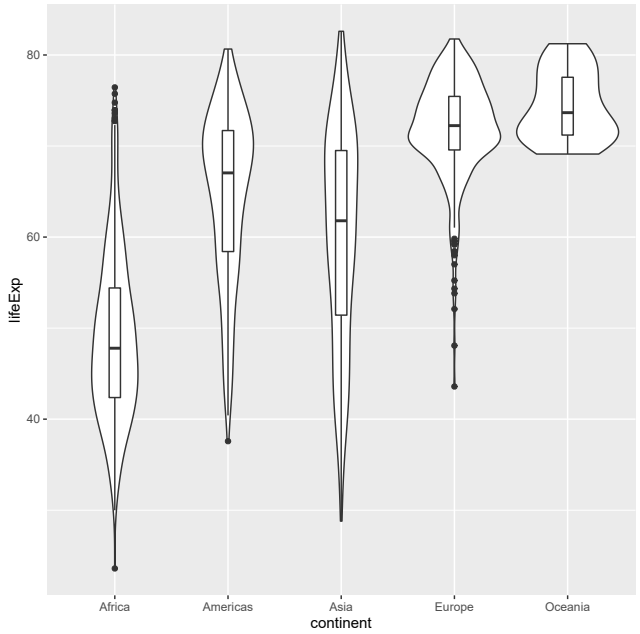
A violin plot is a kernel density estimate, mirrored so that it forms a symmetrical shape.

```
my_violin <- ggplot(data = gapminder, mapping = aes(x = continent, y = lifeExp)) +
  geom_violin()
my_violin
```



We can overlay with box plot

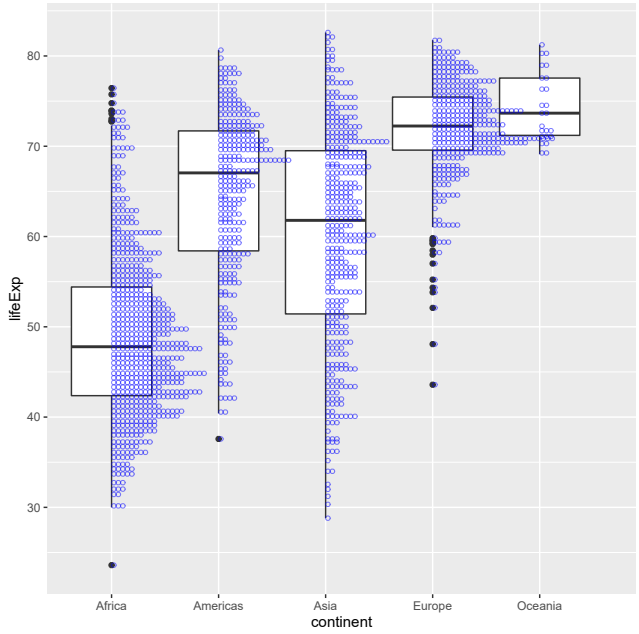
```
my_violin + geom_boxplot(width = .1)
```



4.10 Dot plot

Dot plots are sometimes overlaid on box plots. In these cases, it may be helpful to make the dots hollow

```
ggplot(data = gapminder, mapping = aes(x = continent, y = lifeExp)) +
  geom_boxplot() +
  scale_y_continuous(breaks = seq(30,80,10)) +
  geom_dotplot(binaxis = "y", binwidth = 0.5, fill = NA, colour = "blue", alpha = 0.5)
```



4.11 Line plot

Line graphs are typically used for visualizing how one continuous variable, on the y-axis, changes in relation to another continuous variable, on the x-axis.

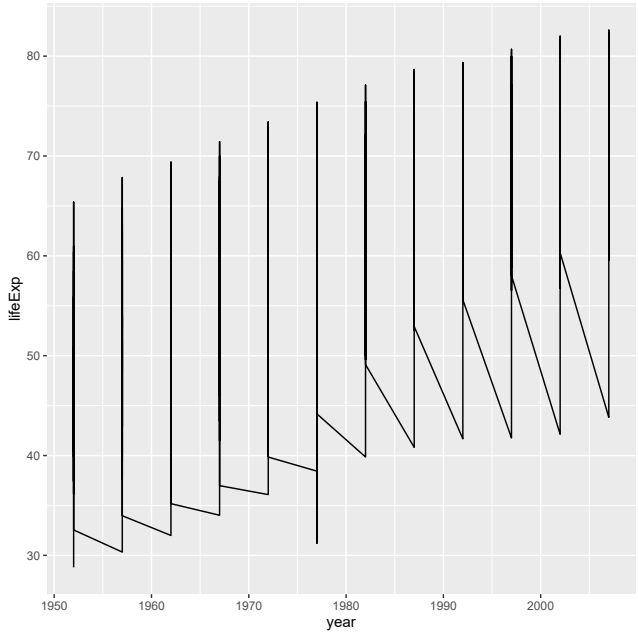
As with bar graphs, there are exceptions. Line graphs can also be used with a discrete variable on the x-axis. This is appropriate when the variable is ordered (e.g., “small,” “medium,” “large”).

How about the life expectancy for Asia continent and also Malaysia in comparison

1. gapminder data
2. continent == “Asia”
3. life expectancy

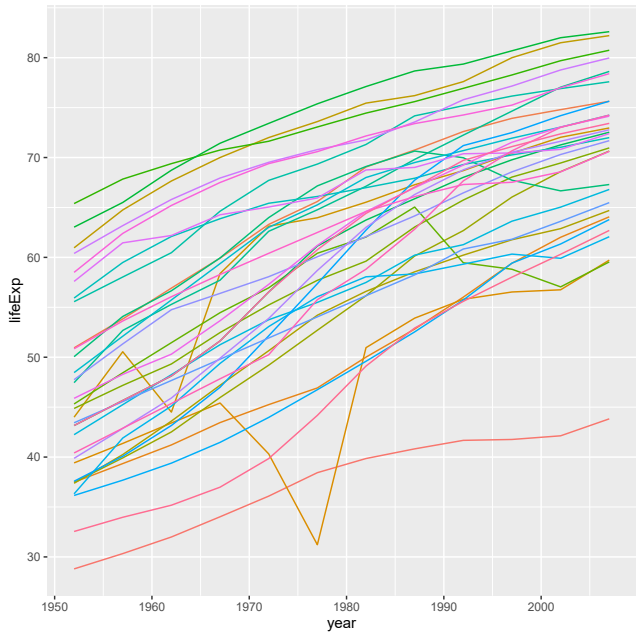
4. trend

```
gapminder %>% filter(continent == "Asia") %>%
ggplot(mapping = aes(x = year, y = lifeExp)) +
  geom_line()
```



Does not seem right,

```
gapminder %>% filter(continent == "Asia") %>%
ggplot(mapping = aes(x = year, y = lifeExp, colour = country)) +
  geom_line(show.legend = FALSE)
```

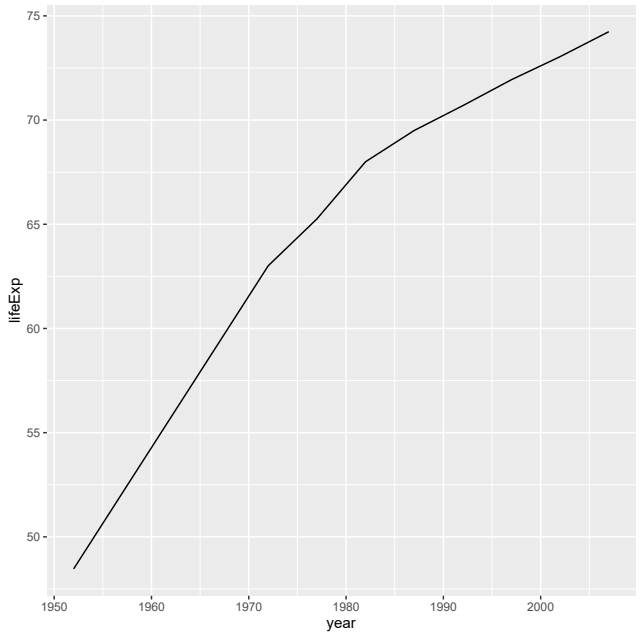


Let us make a line graph for only Malaysia

1. gapminder data
2. country == “Malaysia”
3. life expectancy
4. trend

Pay attention to the dplyr codes and ggplot2 codes:

```
gapminder %>% filter(country == "Malaysia") %>%
  ggplot(mapping = aes(x = year, y = lifeExp)) +
  geom_line()
```



4.12 Plotting means and error bars (10 min)

We want to error bar for life expectancy for Asia continent only. Error bar that will contain

- mean
- standard deviation

Our approach is:

- use filter to choose Asia continent only `filter()`
- generate the mean and SD for life expectancy using `mutate()`
- plot the scatterplot (country vs mean life expectancy) `geom_point()`
- plot errorbar `geom_errorbar()`

```
gap_continent <- gapminder %>% filter(continent == "Asia") %>%
  group_by(country) %>% mutate(mean = mean(lifeExp), sd = sd(lifeExp)) %>%
  arrange(desc(mean))
```

Let is check the result

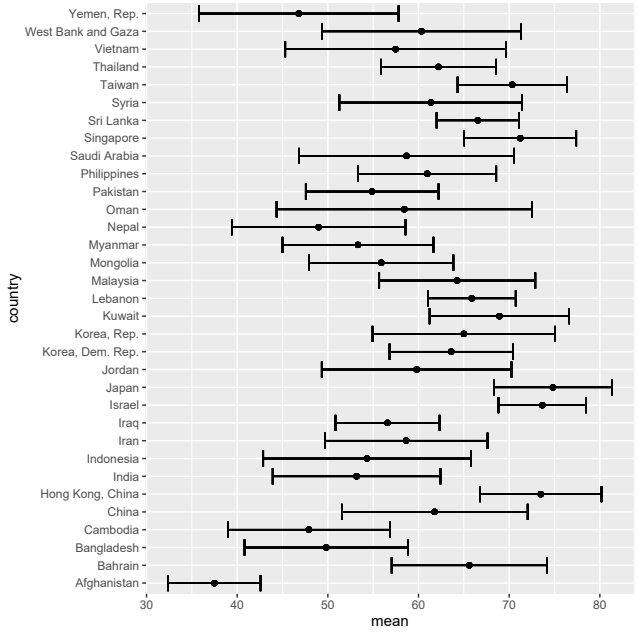
```
head(gap_continent)
```

```
## # A tibble: 6 x 8
```

```
## # Groups:   country [1]
##   country continent  year lifeExp       pop gdpPercap  mean    sd
##   <fct>   <fct>     <int>   <dbl>     <int>     <dbl> <dbl> <dbl>
## 1 Japan   Asia       1952    63.0  86459025     3217.  74.8  6.49
## 2 Japan   Asia       1957    65.5  91563009     4318.  74.8  6.49
## 3 Japan   Asia       1962    68.7  95831757     6577.  74.8  6.49
## 4 Japan   Asia       1967    71.4 100825279     9848.  74.8  6.49
## 5 Japan   Asia       1972    73.4 107188273    14779.  74.8  6.49
## 6 Japan   Asia       1977    75.4 113872473    16610.  74.8  6.49
```

Plot them with `coord_flip()`

```
gap_continent %>%
  ggplot(mapping = aes(x = country, y = mean)) +
  geom_point(mapping = aes(x = country, y = mean)) +
  geom_errorbar(mapping = aes(ymin = mean - sd, ymax = mean + sd),
                position = position_dodge()) +
  coord_flip()
```



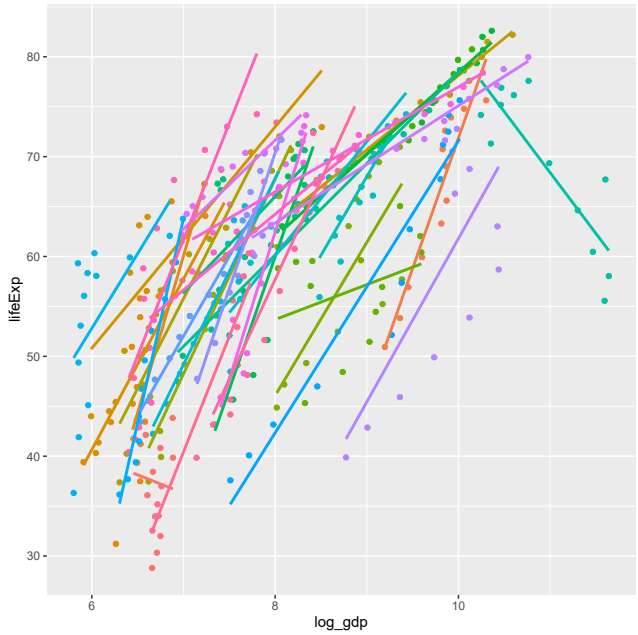
4.13 Scatterplot with fit line

The steps below shows that:

- data is gapminder

- a new variable log_gdp which equals log of gdpPercap
- filter to Asia continent
- make a plot
- choose variables log_gdp and lifeExp based on country
- plot scatterplot
- plot fit line

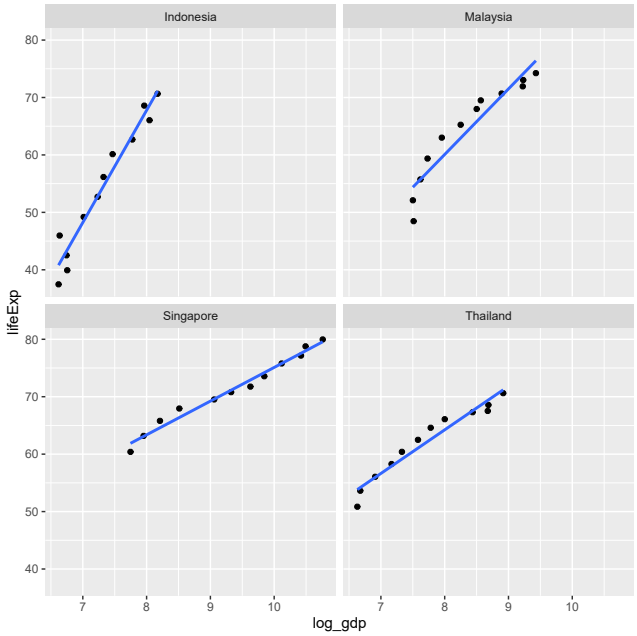
```
gapminder %>% mutate(log_gdp = log(gdpPercap)) %>%
  filter(continent == "Asia") %>%
ggplot(mapping = aes(x = log_gdp, y = lifeExp, colour = country)) +
  geom_point(show.legend = FALSE) +
  geom_smooth(method = lm, se = FALSE, show.legend = FALSE)
```



Now,

- choose selected countries: Malaysia, Indonesia, Singapore and Thailand.
- use `facet_wrap()` to split the plots based on the 4 countries

```
gapminder %>% mutate(log_gdp = log(gdpPercap)) %>%
  filter(country %in% c("Malaysia", "Indonesia", "Singapore", "Thailand")) %>%
ggplot(mapping = aes(x = log_gdp, y = lifeExp)) +
  geom_point(show.legend = FALSE) +
  geom_smooth(method = lm, se = FALSE, show.legend = FALSE) +
  facet_wrap(~ country, ncol = 2)
```

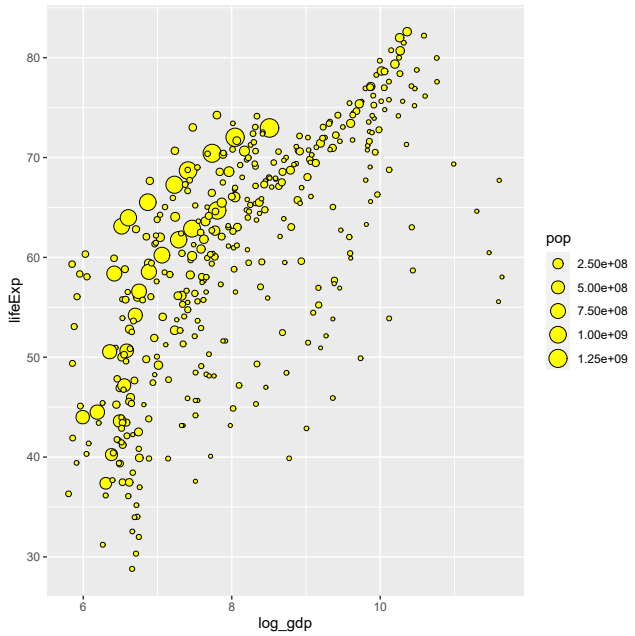


4.14 Balloon plot

Relationship between log_gdp and life expectancy

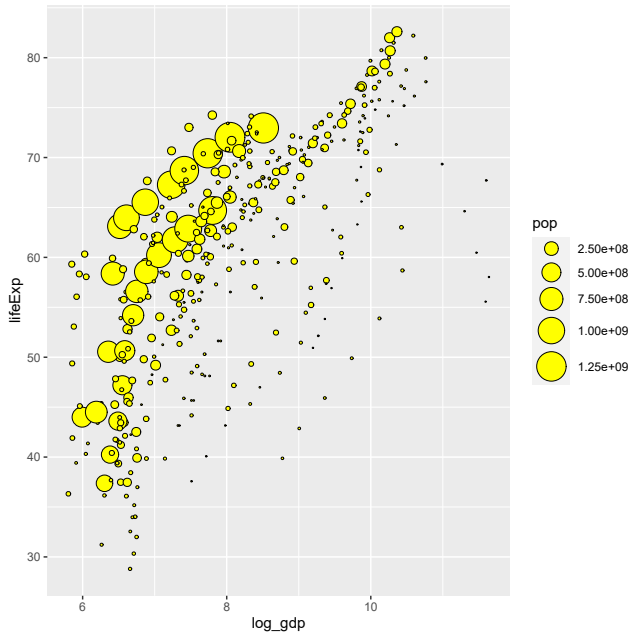
- gapminder data for countries in the Asia continent
- variable log_gdp as the predictor
- variable life expectancy as the outcome
- variable population for the size
- using scatterplot

```
ballon_plot <- gapminder %>% mutate(log_gdp = log(gdpPercap)) %>%
  filter(continent == "Asia") %>%
  ggplot(mapping = aes(x = log_gdp, y = lifeExp, size = pop)) +
  geom_point(shape = 21, colour = "black", fill = "yellow")
ballon_plot
```



We instead want to map the value of population to the area of the dots, which we can do using scale_size_area()

```
ballon_plot + scale_size_area(max_size = 10)
```


80
70
60
50
40
30
lifeExp
6
8
10
log_gdp
pop
2.50e+08
5.00e+08
7.50e+08
1.00e+09
1.25e+09