Data Visualization, Exploration and Analysis *Practical Notes for Making Plots and Doing Regression Analysis in R*

Kamarul Imran Musa
Wan Nor Arifin

2019-02-19

Contents

1 Data Visualization

1.1 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".

References on data visualization:

1. Link 1 https://en.m.wikipedia.org/wiki/Data_visualization
2. Link 2 https://en.m.wikipedia.org/wiki/Michael_Friendly

1.1.1 History of data visualization

1983 book The Visual Display of Quantitative Information, Edward Tufte defines **graphical displays** and principles for effective graphical display

The book defines "Excellence in statistical graphics consists of complex ideas communicated with clarity, precision and efficiency."

1.1.2 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.

Reference: http://researcher.watson.ibm.com/researcher/view_group.php?id=143

1.2 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.

1.3 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.

1.4 Introduction to ggplot2 package

- **ggplot2** 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.

1.5 Preparation

1.5.1 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`)

1.5.2 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?

1.5.3 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

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. Some examples of databases

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

1.5.4 Load the library

ggplot2 is one of the core member of **tidyverse** package (https://www.tidyverse.org/).

Once we load the **tidyverse** package, we will also have access to

1. help pages
2. functions
3. datasets

```r
library(tidyverse)
```

```
## -- Attaching packages -------- tidyverse 1.2.1 --

## v ggplot2 3.1.0     v purrr   0.2.5
## v tibble  1.4.2     v dplyr   0.7.8
## v tidyr   0.8.2     v stringr 1.3.1
## v readr   1.3.1     v forcats 0.3.0

## -- Conflicts ----------- tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
```

If you run the code and you see *there is no package called tidyverse* then you need to install the **tidyverse** package.

to do that type `install.package("tidyverse")`, then run again `library(tidyverse)`.

1.5.5 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 website contains many useful datasets and show wonderful graphics. It is made popular by Dr Hans Rosling.

Load the package,

```
library(gapminder)
```

call the data into R and browse the data the top of the data

```
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)
```

```
## Observations: 1,704
## Variables: 6
## $ country   <fct> Afghanistan, Afghanistan, Afghanistan, Afghanistan, ...
## $ continent <fct> Asia, Asia, Asia, Asia, Asia, Asia, Asia, Asia, Asia...
## $ year      <int> 1952, 1957, 1962, 1967, 1972, 1977, 1982, 1987, 1992...
## $ lifeExp   <dbl> 28.801, 30.332, 31.997, 34.020, 36.088, 38.438, 39.8...
## $ pop       <int> 8425333, 9240934, 10267083, 11537966, 13079460, 1488...
## $ gdpPercap <dbl> 779.4453, 820.8530, 853.1007, 836.1971, 739.9811, 78...
```

The 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()`. It will list

1. frequencies
2. 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 the package, we can use ?

```
?gapminder
```

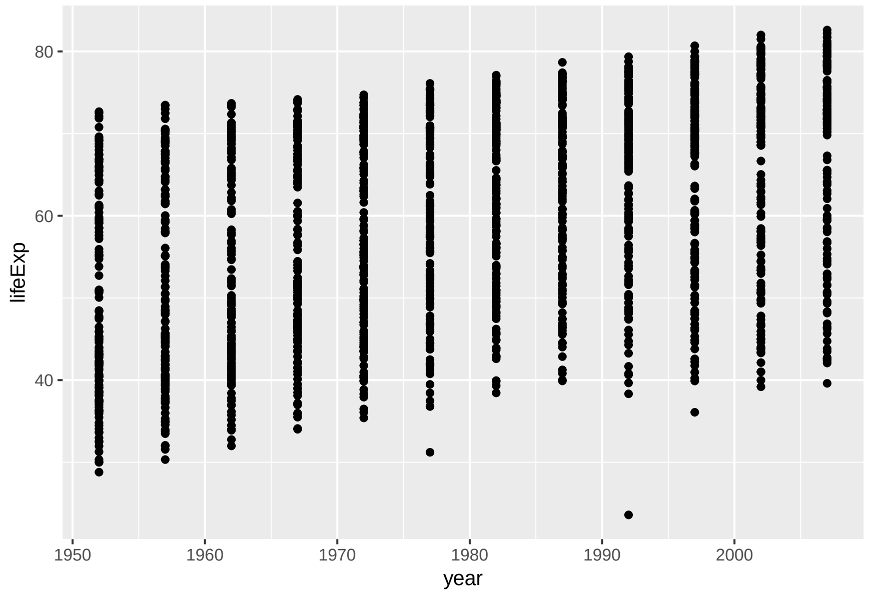
```
## starting httpd help server ... done
```

1.6 Basic plot

We can start create a basic plot

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

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



The plot shows:

1. the relationship between year and life expectancy.

2. as 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.

1.7 Adding another variable

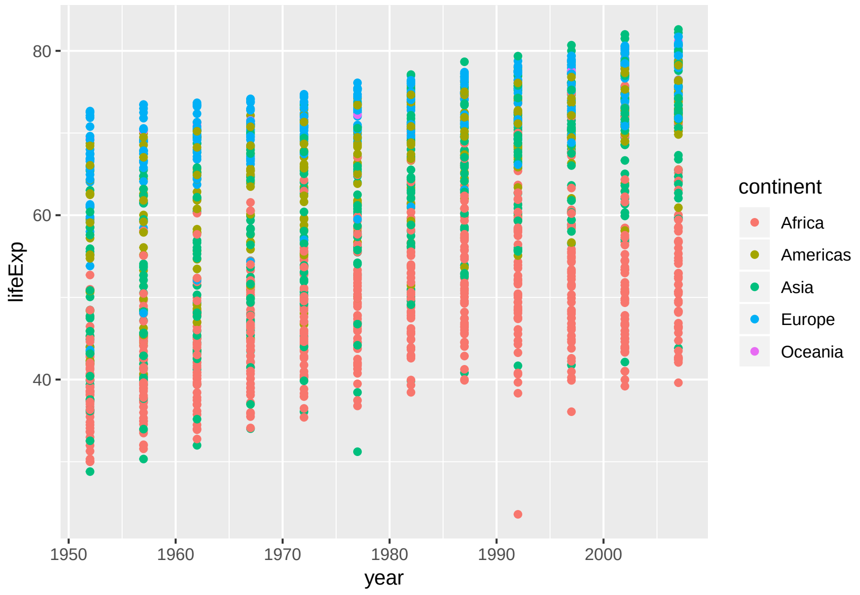
You realize that we plotted 2 variables based on `aes()`. We can add the third variable to make a more complicated plot.

For example:

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

Objective: to plot the relationship between year and life expectancy based on continent.

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

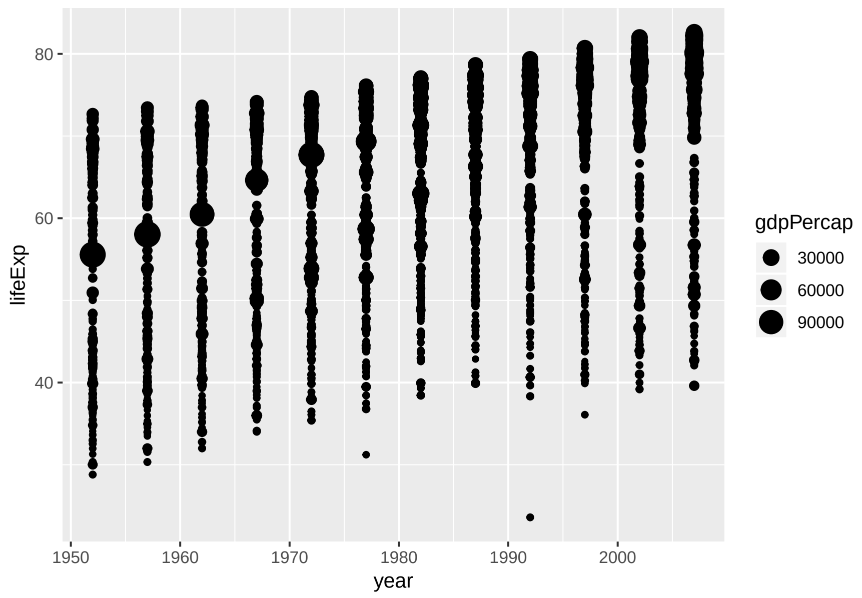


What can you see from the scatterplot.

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.

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



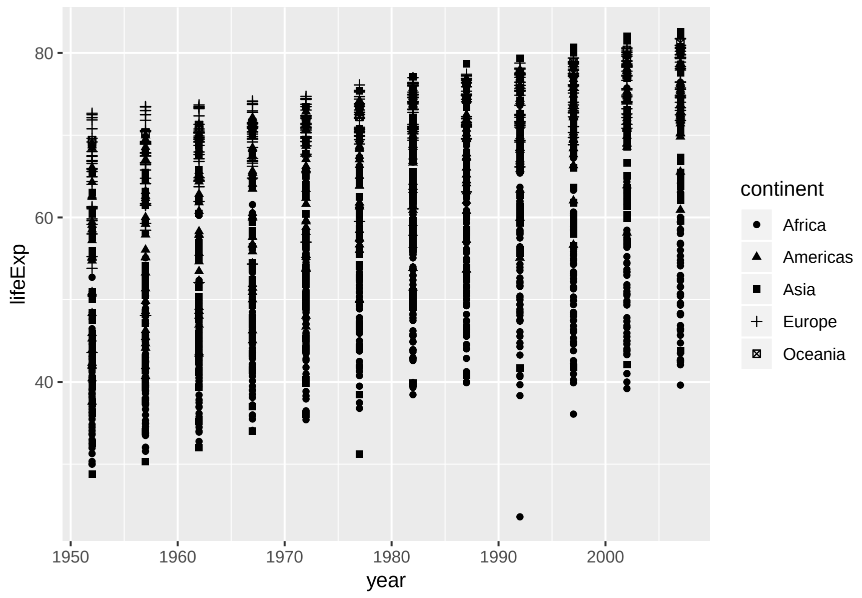
ggplot2 will automatically assign a unique level of the aesthetic (here a unique color) to each unique value of the variable, a process known as scaling.

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

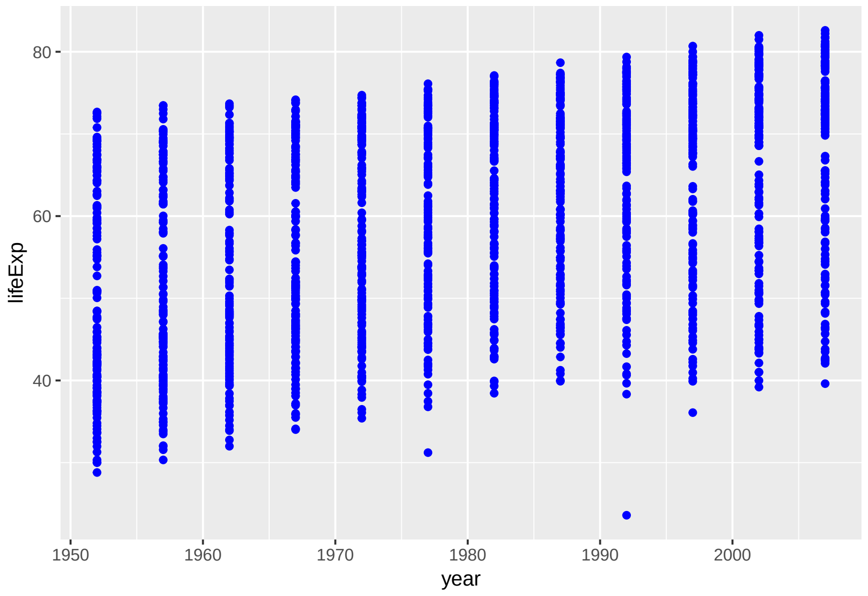
```
ggplot(data = gapminder) +
  geom_point(mapping = aes(x = year, y = lifeExp, shape = continent))
```



But, see what will happen if you set the colour and shape like below but outside the aes parentheses.

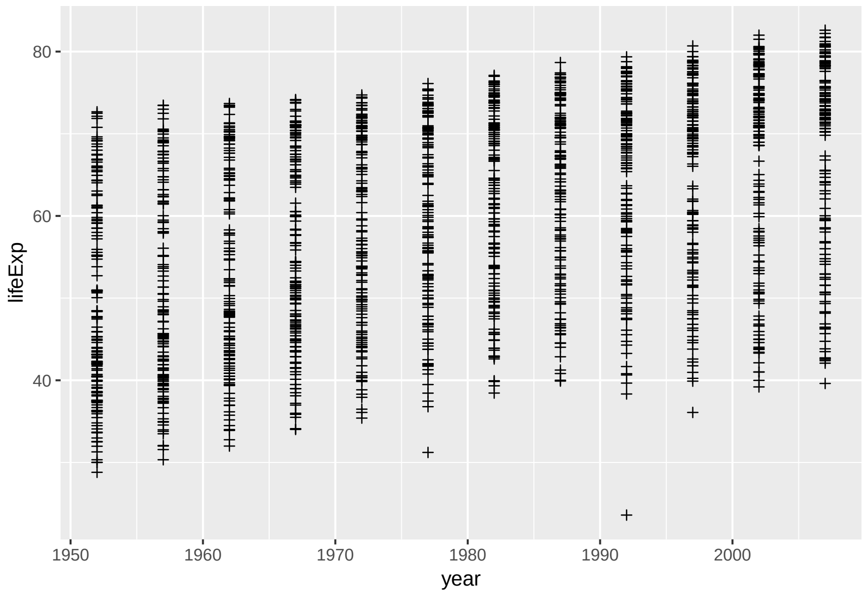
colour as blue

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



shape as plus

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



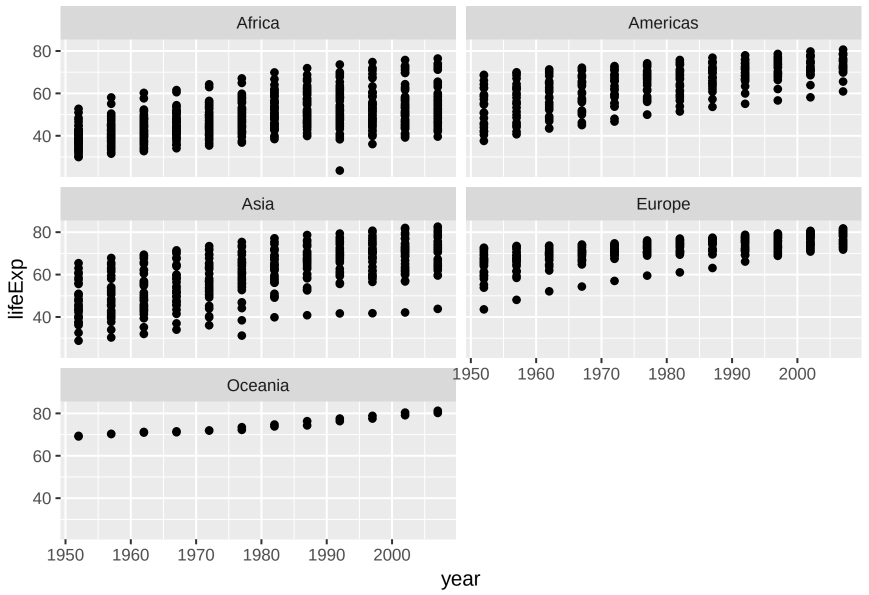
You can type ?*pch* to see the number that correspond to the shape

1.8 Making subplots

We can split our plots based on a factor variable and make subplots using the `facet()`.

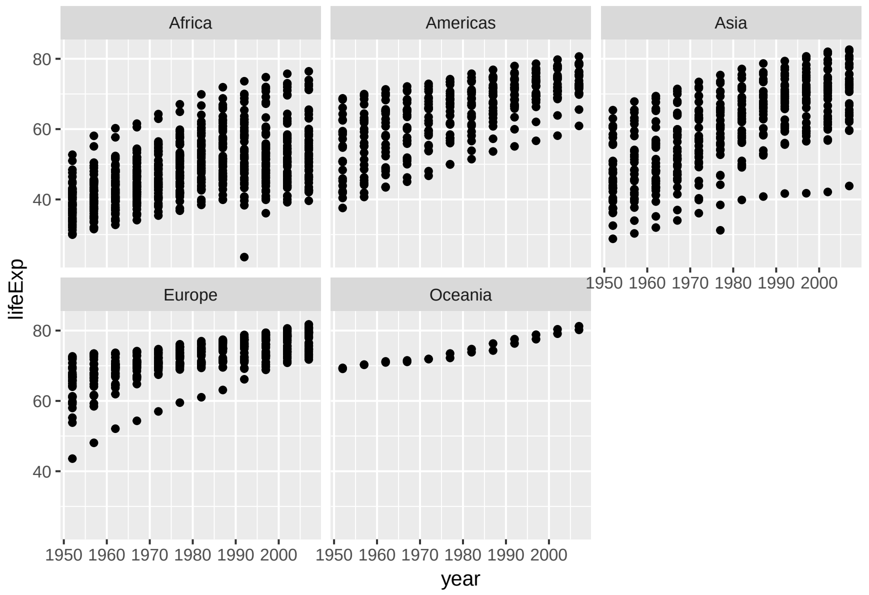
For example, if we want to make subplots based on continents, you can run these codes

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



and change the nrow

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

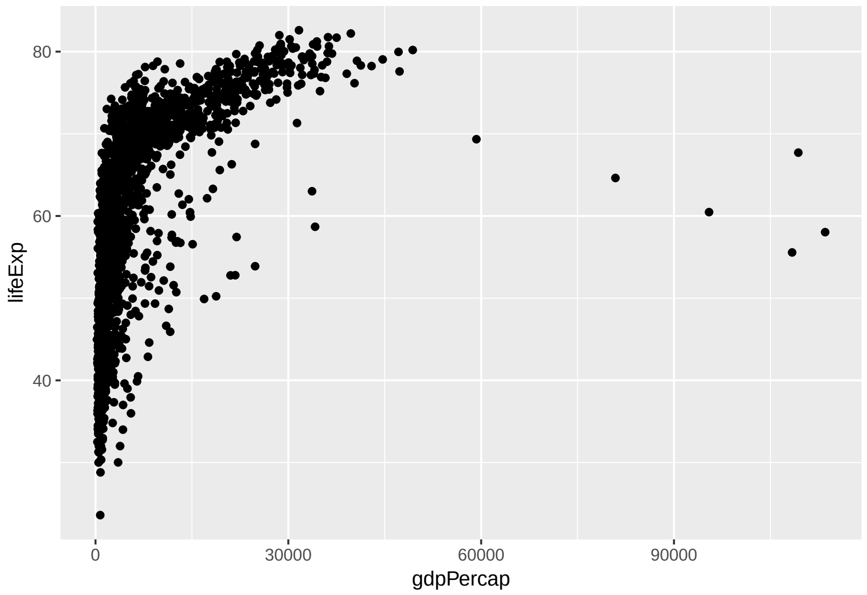


1.9 Overlaying plots

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

Scatterplot

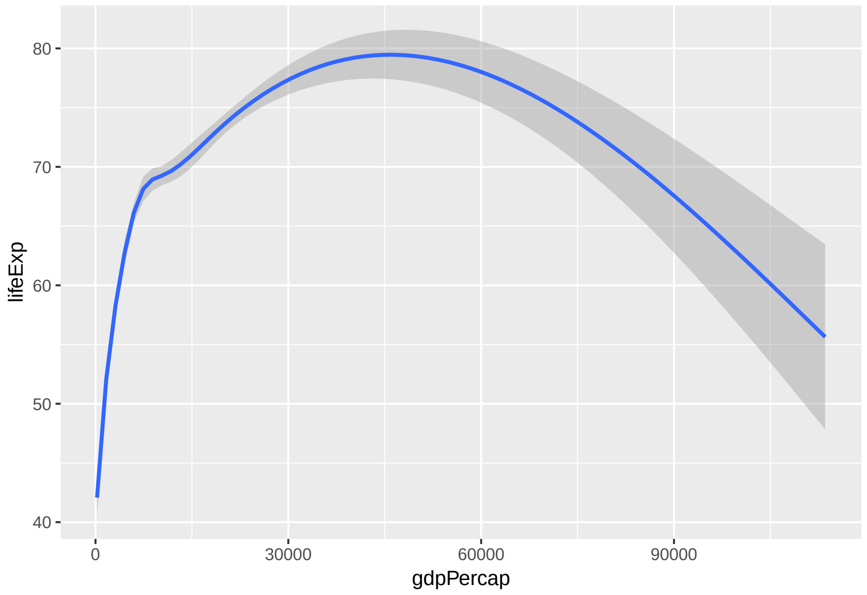
```r
ggplot(data = gapminder) +
  geom_point(mapping = aes(x = gdpPercap, y = lifeExp))
```



Smooth line

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

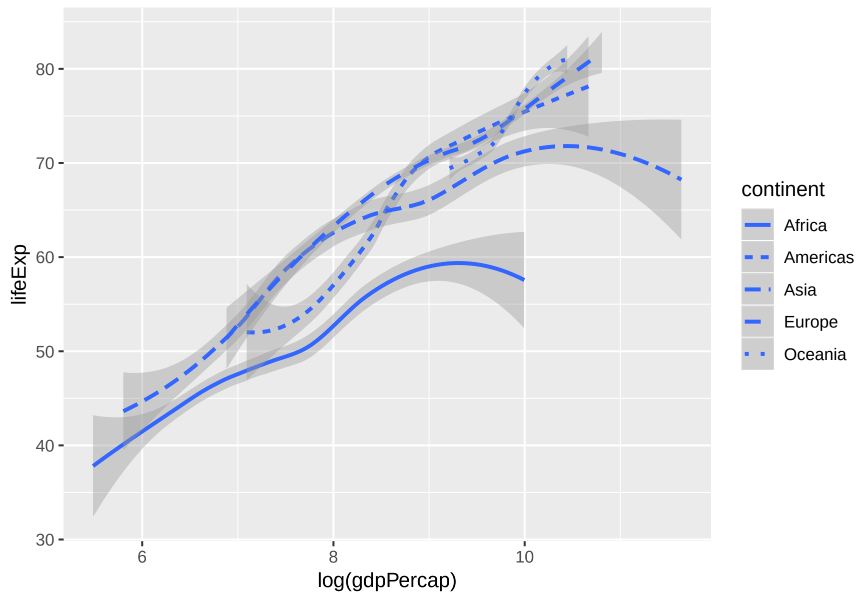
```
## `geom_smooth()` using method = 'gam' and formula 'y ~ s(x, bs = "cs")'
```



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

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

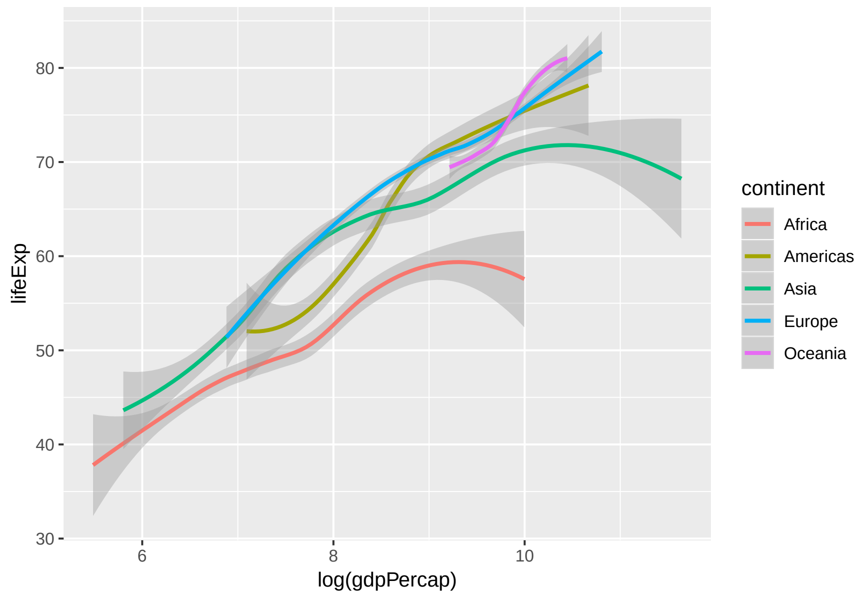
```
## `geom_smooth()` using method = 'loess' and formula 'y ~ x'
```



Another plot but using colour

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

```
## `geom_smooth()` using method = 'loess' and formula 'y ~ x'
```

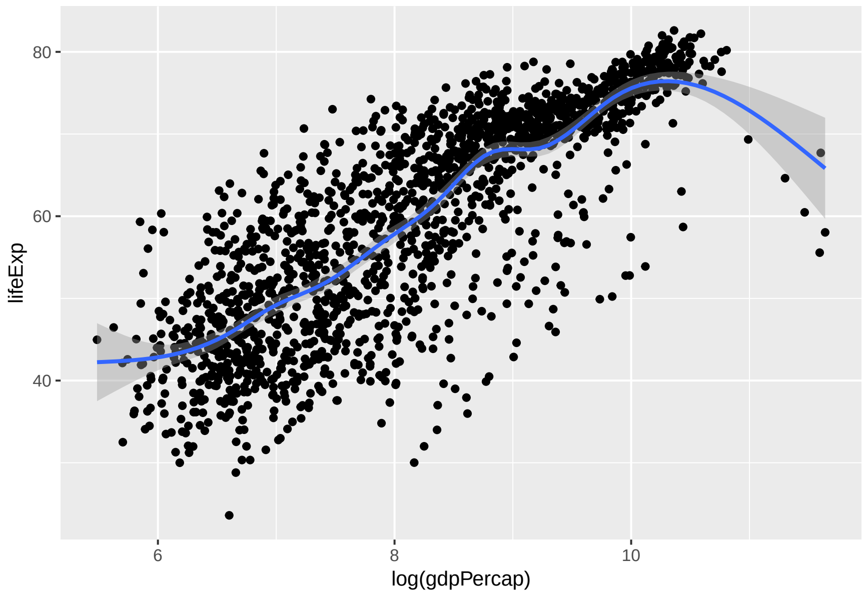


1.10 Combining geom

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

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

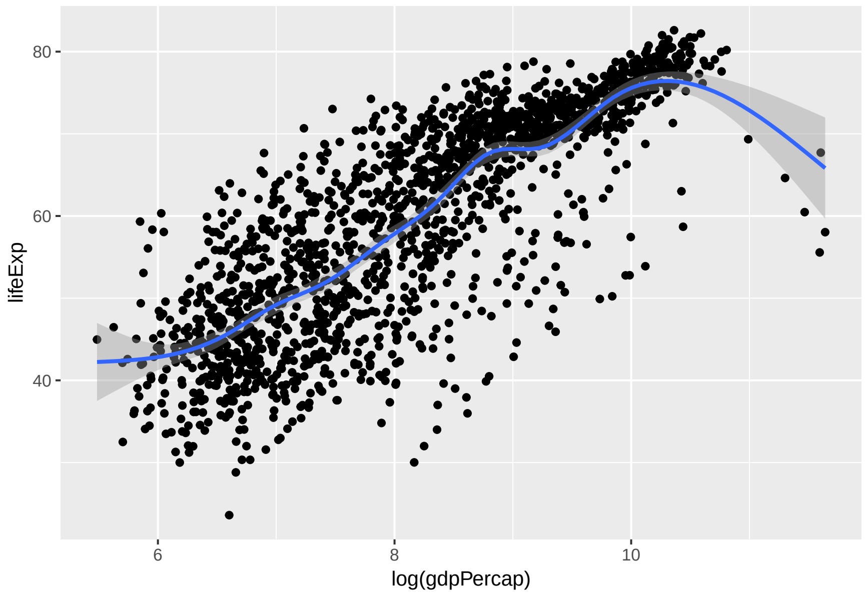
```
## `geom_smooth()` using method = 'gam' and formula 'y ~ s(x, bs = "cs")'
```



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

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

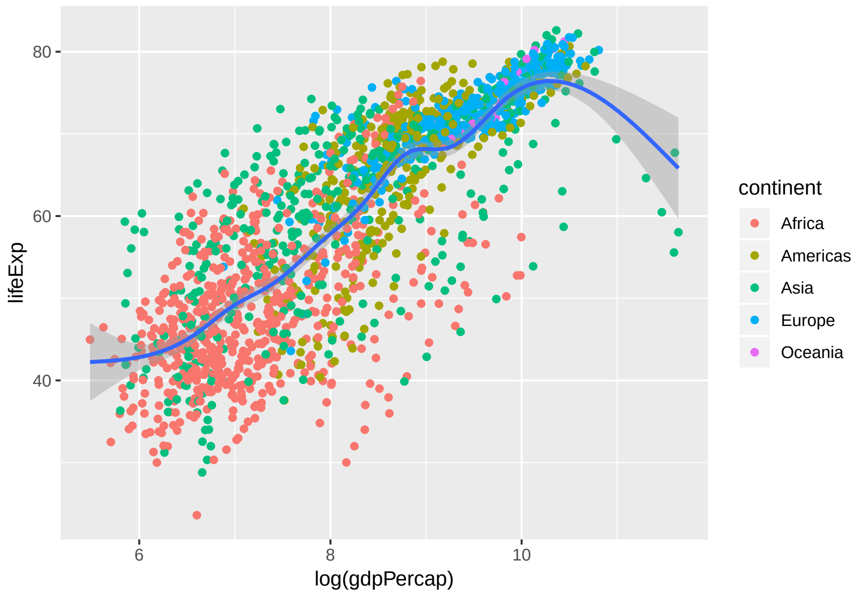
```
## `geom_smooth()` using method = 'gam' and formula 'y ~ s(x, bs = "cs")'
```



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

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

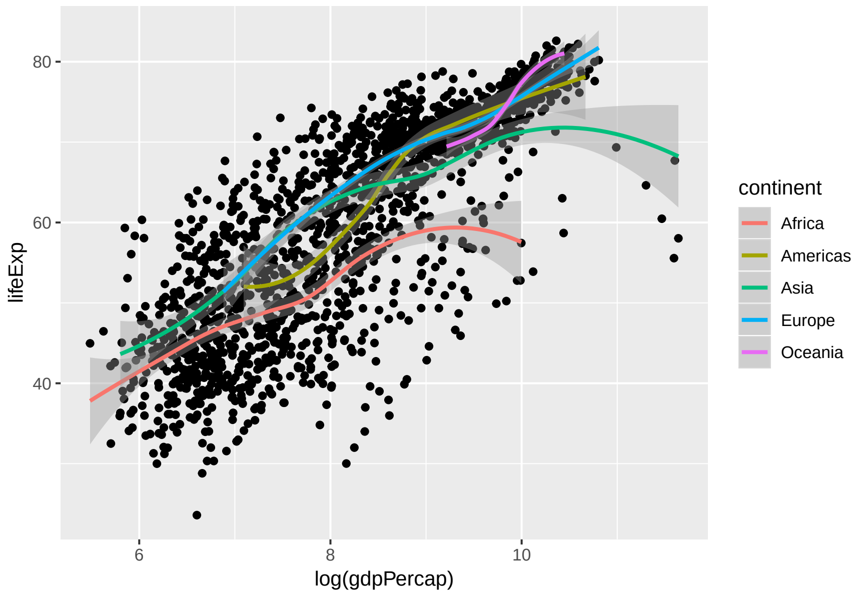
```
## `geom_smooth()` using method = 'gam' and formula 'y ~ s(x, bs = "cs")'
```



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

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

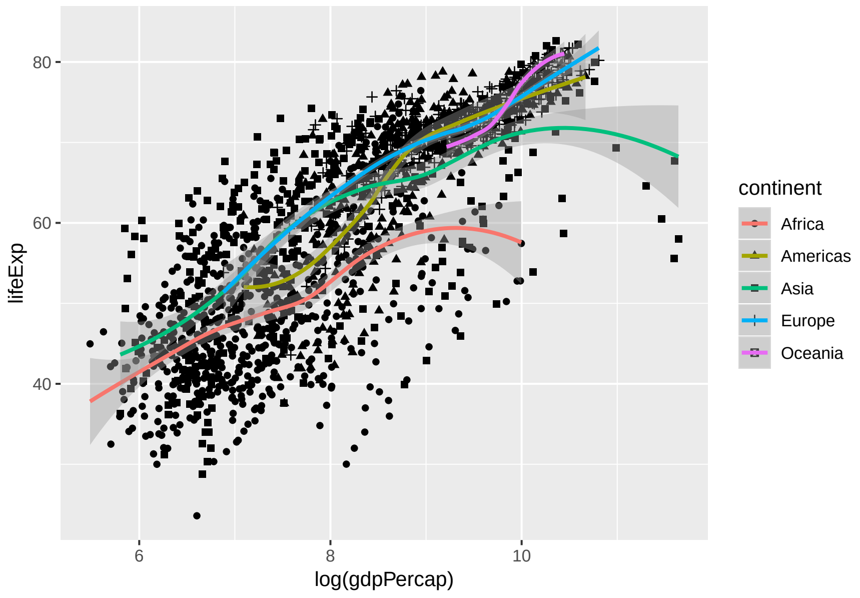
```
## `geom_smooth()` using method = 'loess' and formula 'y ~ x'
```



Or both the scatterplot and the smoothplot

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

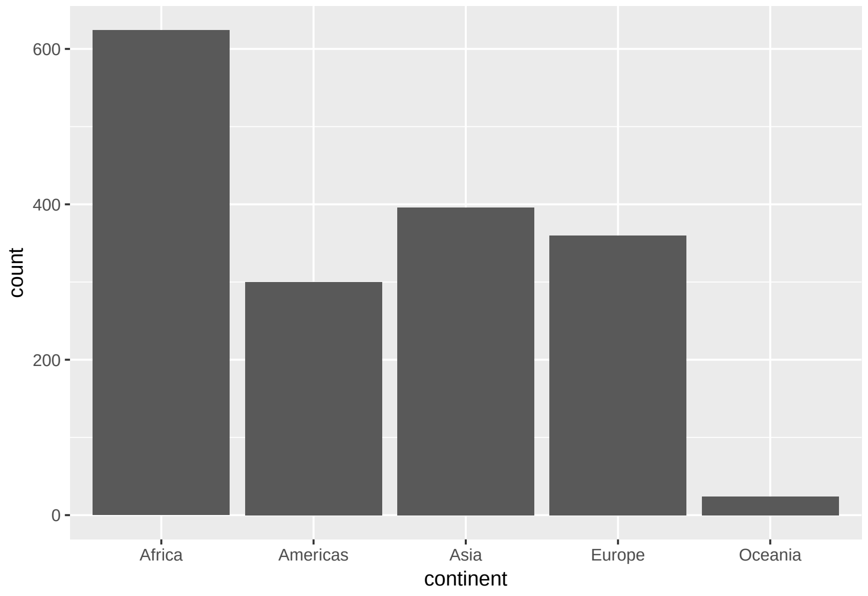
```
## `geom_smooth()` using method = 'loess' and formula 'y ~ x'
```



1.11 Statistical transformation

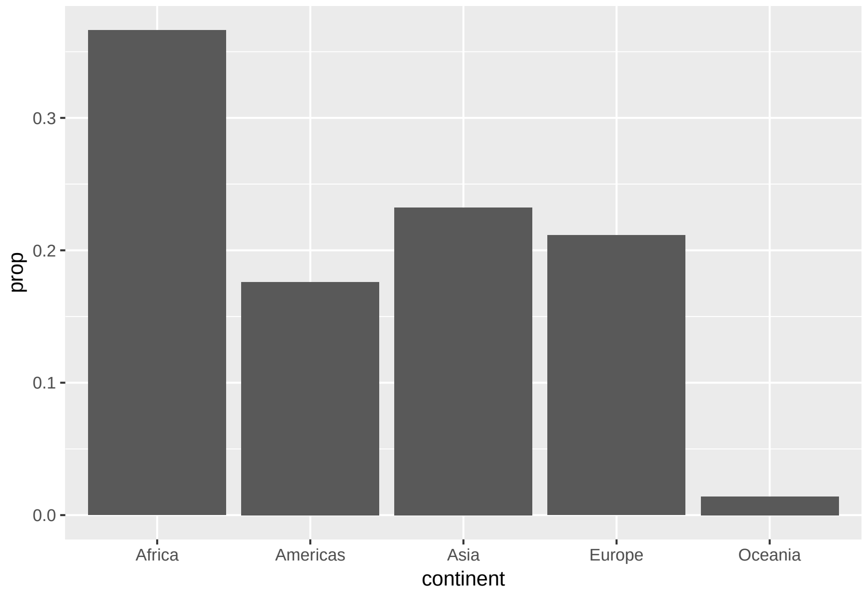
Let us create a bar chart, with y axis as the frequency.

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



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

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



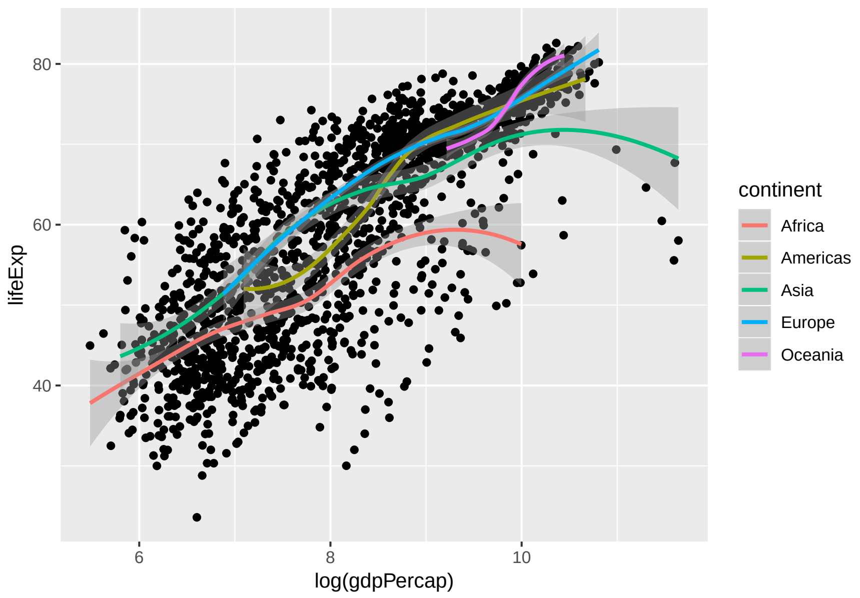
1.12 Customizing title

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`

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

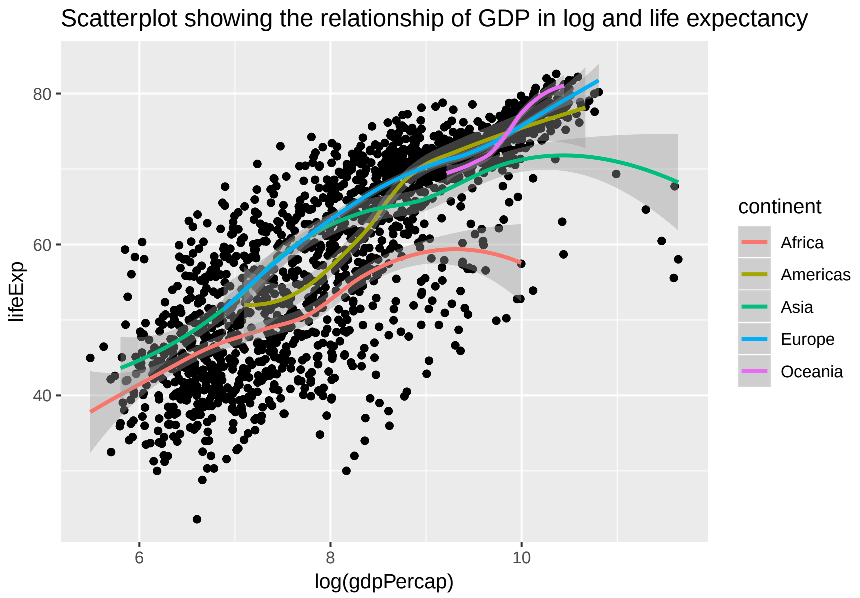
```
## `geom_smooth()` using method = 'loess' and formula 'y ~ x'
```



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")
```

```
## `geom_smooth()` using method = 'loess' and formula 'y ~ x'
```



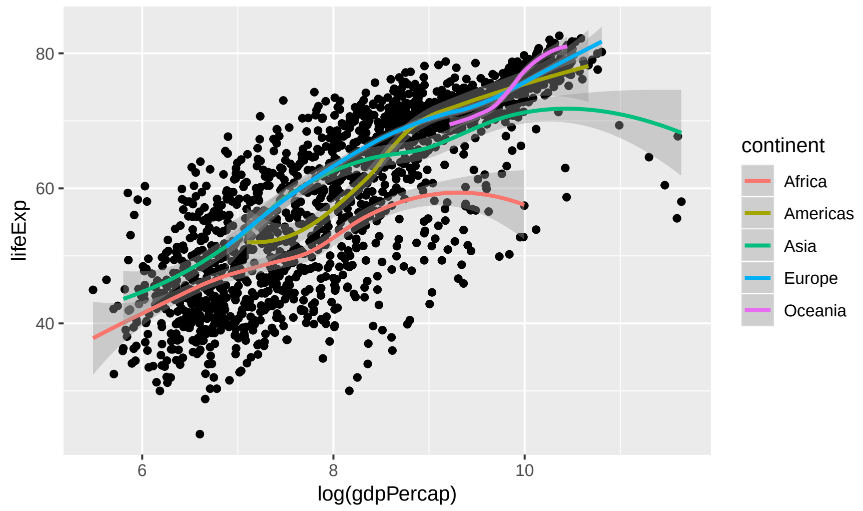
Title in multiple lines by adding \n

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

```
## `geom_smooth()` using method = 'loess' and formula 'y ~ x'
```

Scatterplot showing the relationship of GDP in log and life expectancy:

Data from Gapminder



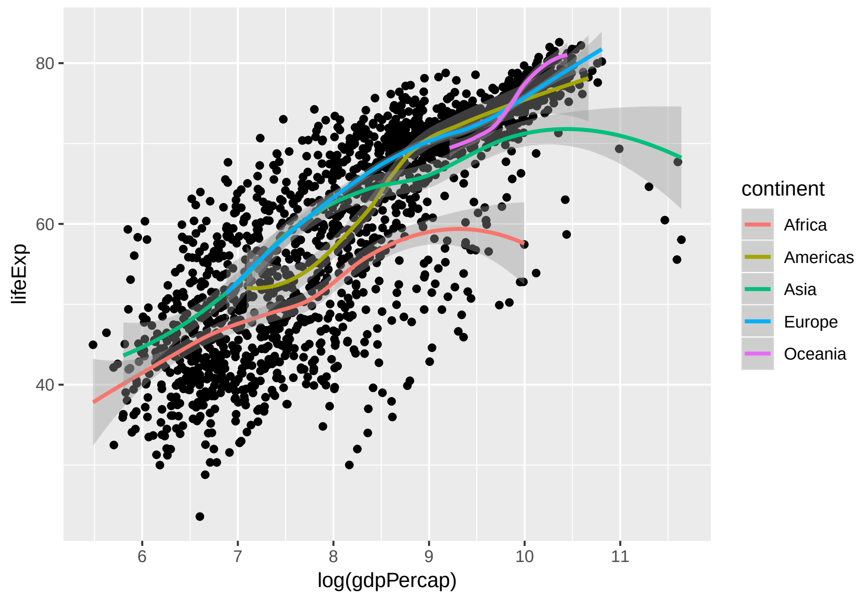
1.13 Adjusting axes

We can specify the tick marks

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

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

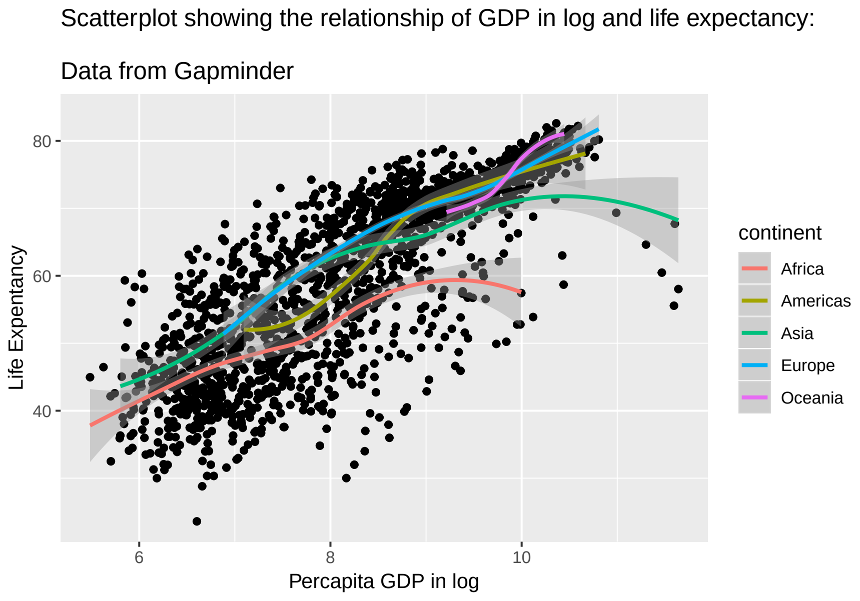
```
## `geom_smooth()` using method = 'loess' and formula 'y ~ x'
```



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

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

```
## `geom_smooth()` using method = 'loess' and formula 'y ~ x'
```



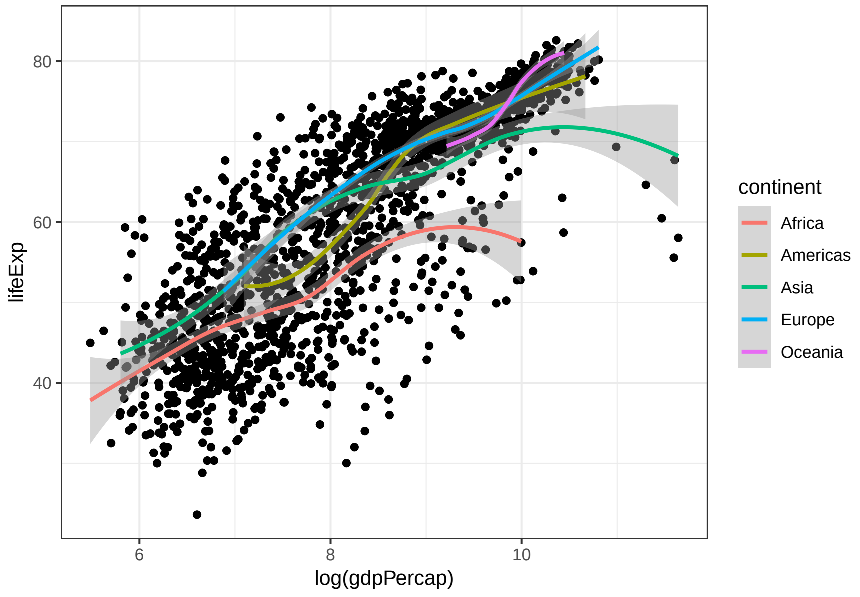
1.14 Choosing theme

The default is gray theme or `theme_gray()`

The black and white theme

```r
mypop + theme_bw()
```

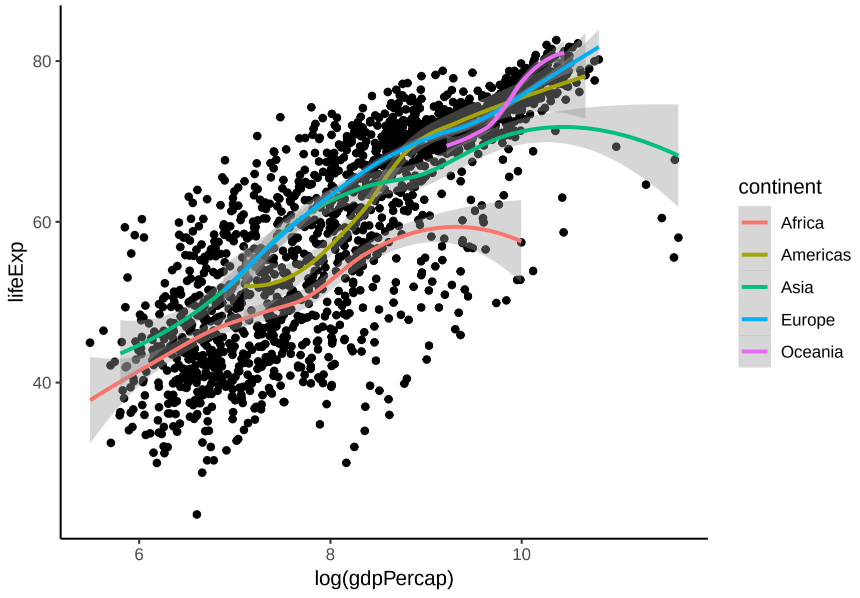
```
## `geom_smooth()` using method = 'loess' and formula 'y ~ x'
```



Classic theme

```
mypop + theme_classic()
```

```
## `geom_smooth()` using method = 'loess' and formula 'y ~ x'
```



1.15 Saving plot

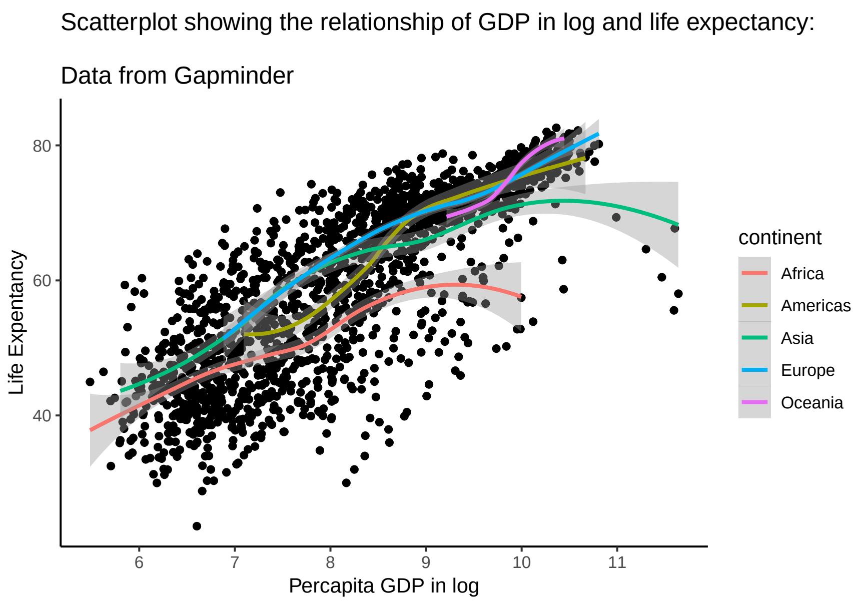
The preferred format for saving file is PDF.

1.16 Saving plot using ggplot2

For example, let us create a more complete plot with added title, x label and y label and a 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
```

```
## `geom_smooth()` using method = 'loess' and formula 'y ~ x'
```



And we want to save the plot (now on the screen) to these formats:

1. pdf format
2. png format
3. jpg format

```r
library(here)
```

```
## here() starts at D:/KIM/OneDrive/OneDrive - Universiti Sains Malaysia/R_Course_Book
```

```r
ggsave(plot = myplot, here("plots","my_pdf_plot.pdf"))
```

```
## Saving 6.5 x 4.5 in image
```

```
## `geom_smooth()` using method = 'loess' and formula 'y ~ x'
```

```r
ggsave(plot = myplot, here("plots","my_png_plot.png"))
```

```
## Saving 6.5 x 4.5 in image
## `geom_smooth()` using method = 'loess' and formula 'y ~ x'
```

```r
ggsave(plot = myplot, here("plots","my_jpg_plot.jpg"))
```

```
## Saving 6.5 x 4.5 in image
## `geom_smooth()` using method = 'loess' and formula 'y ~ x'
```

We can customize the

1. width = 10 cm (or in)
2. height = 6 cm (or in)
3. dpi = 150

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

```
## `geom_smooth()` using method = 'loess' and formula 'y ~ x'
```

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

```
## `geom_smooth()` using method = 'loess' and formula 'y ~ x'
```

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

```
## `geom_smooth()` using method = 'loess' and formula 'y ~ x'
```

2 Data transformation

2.1 Definition of data transformation

Data transformation is also known as Data Munging or Data Wrangling. 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.

For more information, please refer https://community.modeanalytics.com/sql/tutorial/data-wrangling-with-sql/

2.2 Data transformation with dplyr package

2.2.1 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

For more information, please read https://github.com/tidyverse/dplyr

2.3 Common procedures for doing data transformation

When we begin to work with data, common procedures include transforming variables in the dataset.

The common 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

2.4 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

2.5 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, I will emphasize again, every time you want to start processing your data, please make sure:

1. to use R project to work with your data or analysis
2. if you are not using R project, make sure you are inside the correct working directory. Type `getwd()` to display the active **working directory**. And to set a working directory use `setwd()`.
3. once you are know where your working directory is, you can start read or import data into your working directory. Remember, there are a number of packages you can use to read the data into R. It depends on the format of your data.

For example, we know that data format can be in:

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

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.

2.6 *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)`.

```r
library(dplyr)
```

Take a peek at the `starwars` data

```r
glimpse(starwars)
```

```
## Observations: 87
## Variables: 13
## $ name       <chr> "Luke Skywalker", "C-3PO", "R2-D2", "Darth Vader", ...
## $ height     <int> 172, 167, 96, 202, 150, 178, 165, 97, 183, 182, 188...
## $ mass       <dbl> 77.0, 75.0, 32.0, 136.0, 49.0, 120.0, 75.0, 32.0, 8...
## $ hair_color <chr> "blond", NA, NA, "none", "brown", "brown, grey", "b...
## $ skin_color <chr> "fair", "gold", "white, blue", "white", "light", "l...
## $ eye_color  <chr> "blue", "yellow", "red", "yellow", "brown", "blue",...
## $ birth_year <dbl> 19.0, 112.0, 33.0, 41.9, 19.0, 52.0, 47.0, NA, 24.0...
## $ gender     <chr> "male", NA, NA, "male", "female", "male", "female",...
## $ homeworld  <chr> "Tatooine", "Tatooine", "Naboo", "Tatooine", "Alder...
## $ species    <chr> "Human", "Droid", "Droid", "Human", "Human", "Human...
## $ films      <list> [<"Revenge of the Sith", "Return of the Jedi", "Th...
## $ vehicles   <list> [<"Snowspeeder", "Imperial Speeder Bike">, <>, <>,...
## $ starships  <list> [<"X-wing", "Imperial shuttle">, <>, <>, "TIE Adva...
```

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 13
##    name  height  mass hair_color skin_color eye_color birth_year gender
##    <chr>  <int> <dbl> <chr>      <chr>      <chr>          <dbl> <chr>
##  1 Luke~    172    77 blond      fair       blue            19   male
##  2 C-3PO    167    75 <NA>       gold       yellow         112   <NA>
##  3 R2-D2     96    32 <NA>       white, bl~ red             33   <NA>
##  4 Dart~    202   136 none       white      yellow          41.9 male
##  5 Leia~    150    49 brown      light      brown           19   female
##  6 Owen~    178   120 brown, gr~ light      blue            52   male
##  7 Beru~    165    75 brown      light      blue            47   female
##  8 R5-D4     97    32 <NA>       white, red red             NA   <NA>
##  9 Bigg~    183    84 black      light      brown           24   male
## 10 Obi-~    182    77 auburn, w~ fair       blue-gray       57   male
## # ... with 77 more rows, and 5 more variables: homeworld <chr>,
## #   species <chr>, films <list>, vehicles <list>, starships <list>
```

2.7 `dplyr::select()` , `dplyr::mutate()` and `dplyr::rename()`

2.7.1 `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.

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     male      172    77
##  2 C-3PO              <NA>      167    75
##  3 R2-D2              <NA>       96    32
##  4 Darth Vader        male      202   136
##  5 Leia Organa        female    150    49
##  6 Owen Lars          male      178   120
##  7 Beru Whitesun lars female    165    75
##  8 R5-D4              <NA>       97    32
##  9 Biggs Darklighter  male      183    84
## 10 Obi-Wan Kenobi     male      182    77
## # ... with 77 more rows
```

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

2.7.2 `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     male      172    77  26.0
##  2 C-3PO              <NA>      167    75  26.9
##  3 R2-D2              <NA>       96    32  34.7
##  4 Darth Vader        male      202   136  33.3
##  5 Leia Organa        female    150    49  21.8
##  6 Owen Lars          male      178   120  37.9
##  7 Beru Whitesun lars female    165    75  27.5
##  8 R5-D4              <NA>       97    32  34.0
##  9 Biggs Darklighter  male      183    84  25.1
## 10 Obi-Wan Kenobi     male      182    77  23.2
## # ... with 77 more rows
```

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

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

2.8 dplyr::arrange() and dplyr::filter()

2.8.1 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 hermaphrodite    175  1358 443.   196629.
##  2 Dud Bolt              male              94    45  50.9    2594.
##  3 Yoda                  male              66    17  39.0    1523.
##  4 Owen Lars             male             178   120  37.9    1434.
##  5 IG-88                 none             200   140  35      1225
##  6 R2-D2                 <NA>             96    32  34.7    1206.
##  7 Grievous              male            216   159  34.1    1161.
##  8 R5-D4                 <NA>             97    32  34.0    1157.
##  9 Jek Tono Porkins      male            180   110  34.0    1153.
## 10 Darth Vader           male            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).

```r
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     male      193    48  12.9  166.
##  2 Adi Gallia     female    184    50  14.8  218.
##  3 Sly Moore      female    178    48  15.1  230.
##  4 Roos Tarpals   male      224    82  16.3  267.
##  5 Padmé Amidala female    165    45  16.5  273.
##  6 Lama Su        male      229    88  16.8  282.
##  7 Jar Jar Binks male      196    66  17.2  295.
##  8 Ayla Secura    female    178    55  17.4  301.
##  9 Shaak Ti       female    178    57  18.0  324.
## 10 Barriss Offee female    166    50  18.1  329.
## # ... with 77 more rows
```

2.8.2 `dplyr::filter()`

To select observations based on certain criteria, we use the `dplyr::filter()` function.

Here, 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

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

```
## # A tibble: 1 x 6
##   name     gender height  mass   bmi bmisq
##   <chr>    <chr>   <int> <dbl> <dbl> <dbl>
## 1 Dud Bolt male       94    45  50.9 2594.
```

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`

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

2.9 dplyr::group_by() and dplyr::summarize

2.9.1 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)

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

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

2.9.2 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).

```r
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: 5 x 7
##   gender        meanbmi meanht meanmass sdbmi  sdht sdmass
##   <chr>           <dbl>  <dbl>    <dbl> <dbl> <dbl>  <dbl>
## 1 female           18.8   165.     54.0  3.71  23.0   8.37
## 2 hermaphrodite   443.     175     1358    NaN     NaN    NaN
## 3 male             25.7    179.      81.0    6.49   35.4   28.2
## 4 none             35      200      140    NaN     NaN    NaN
## 5 <NA>             31.9    120       46.3    4.33   40.7   24.8
```

To calculate the frequencies

- with 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        5
##  3 <NA>         5
##  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
```

- with two variables

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

2.10 More complicated dplyr verbs

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

```
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: 5 x 4
##   species    mean_ht mean_mass  freq
##   <chr>        <dbl>     <dbl> <int>
## 1 Besalisk      198        102     1
## 2 Human         187.       122     3
## 3 Kaleesh       216        159     1
## 4 Trandoshan    190        113     1
## 5 Wookiee       231        124     2
```

2.11 Data transformation for categorical variables

2.11.1 forcats package

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

2.11.2 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] 0 0 0 1 0 0 1 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     0     1
## 2     0     2
## 3     0     3
## 4     1     4
## 5     0     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)
```

```
## Classes 'tbl_df', 'tbl' and 'data.frame':    100 obs. of  2 variables:
##  $ sex1 : int  0 0 0 1 0 0 1 0 1 1 ...
##  $ race1: int  1 2 3 4 1 2 3 4 1 2 ...
```

2.11.3 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)
```

```
## Classes 'tbl_df', 'tbl' and 'data.frame':    100 obs. of  4 variables:
##  $ sex1 : int  0 0 0 1 0 0 1 0 1 1 ...
##  $ race1: int  1 2 3 4 1 2 3 4 1 2 ...
##  $ male : Factor w/ 2 levels "No","Yes": 1 1 1 2 1 1 2 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     0     1 No    Mal
## 2     0     2 No    Chi
## 3     0     3 No    Ind
## 4     1     4 Yes   Others
## 5     0     1 No    Mal
## 6     0     2 No    Chi
```

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

2.11.4 `forcats::fct_recode()`

Recode old levels to new levels

Our objectives:

1. For variable **male**, change from `No` vs `Yes` to `Fem` and `Male`
2. Create a new variable **malay** from variable **race2** and label `Chi` to `Non-Malay`, `Ind` to `Non-Malay` and `Others` to `Non-Malay`. But we keep `Mal` as it is

We will use **forcats** packages for that. Below we show two ways of recoding the variables.

```
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     0     1 No    Mal    Fem   Mal
## 2     0     2 No    Chi    Fem   Non-Malay
## 3     0     3 No    Ind    Fem   Non-Malay
## 4     1     4 Yes   Others Male  Non-Malay
## 5     0     1 No    Mal    Fem   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     0     3 No    Ind    Fem   Non-Malay
## 2     0     4 No    Others Fem   Non-Malay
## 3     0     1 No    Mal    Fem   Mal
## 4     1     2 Yes   Chi    Male  Non-Malay
## 5     1     3 Yes   Ind    Male  Non-Malay
## 6     1     4 Yes   Others Male  Non-Malay
```

2.12 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.

2.13 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()`)

2.14 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

3 Exploratory data analysis (EDA)

3.1 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

3.2 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_`

3.3 EDA using ggplot2 package

3.3.1 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()`).

3.4 Loading the library

3.4.1 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/

```r
library(tidyverse)
```

3.4.2 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

```r
library(gapminder)
```

Peek at the data

```r
glimpse(gapminder)
```

```
## Observations: 1,704
## Variables: 6
## $ country   <fct> Afghanistan, Afghanistan, Afghanistan, Afghanistan, ...
## $ continent <fct> Asia, Asia, Asia, Asia, Asia, Asia, Asia, Asia, Asia...
## $ year      <int> 1952, 1957, 1962, 1967, 1972, 1977, 1982, 1987, 1992...
## $ lifeExp   <dbl> 28.801, 30.332, 31.997, 34.020, 36.088, 38.438, 39.8...
## $ pop       <int> 8425333, 9240934, 10267083, 11537966, 13079460, 1488...
## $ gdpPercap <dbl> 779.4453, 820.8530, 853.1007, 836.1971, 739.9811, 78...
```

Summary statistics of data

```r
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.
```

3.5 One variable: Distribution of a categorical variable

3.5.1 Bar chart

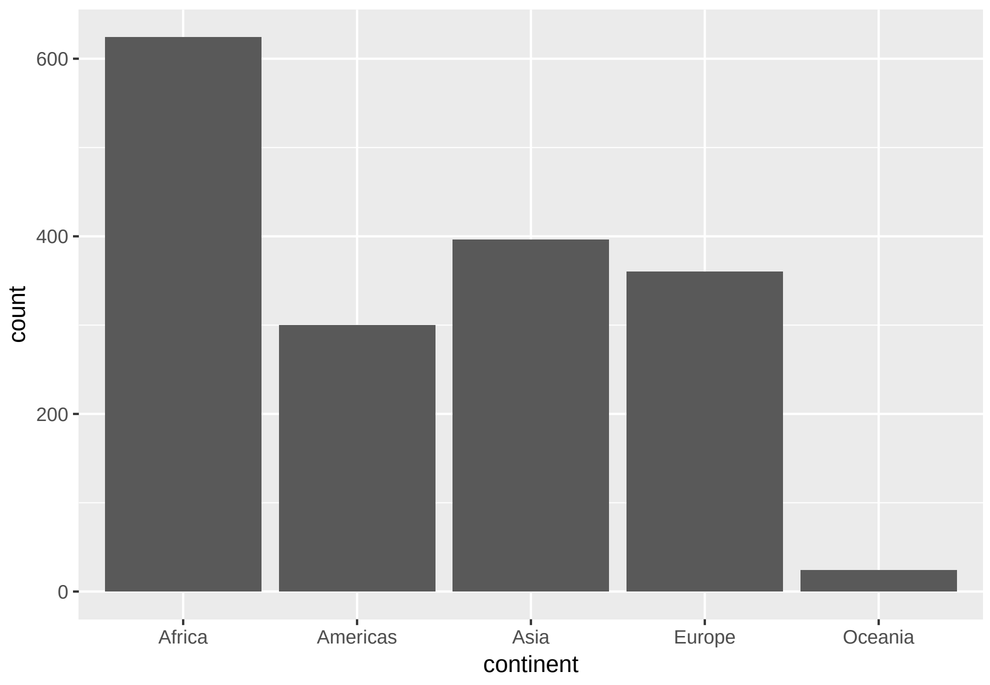
The frequencies (number of countries) based on continents

```
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
```

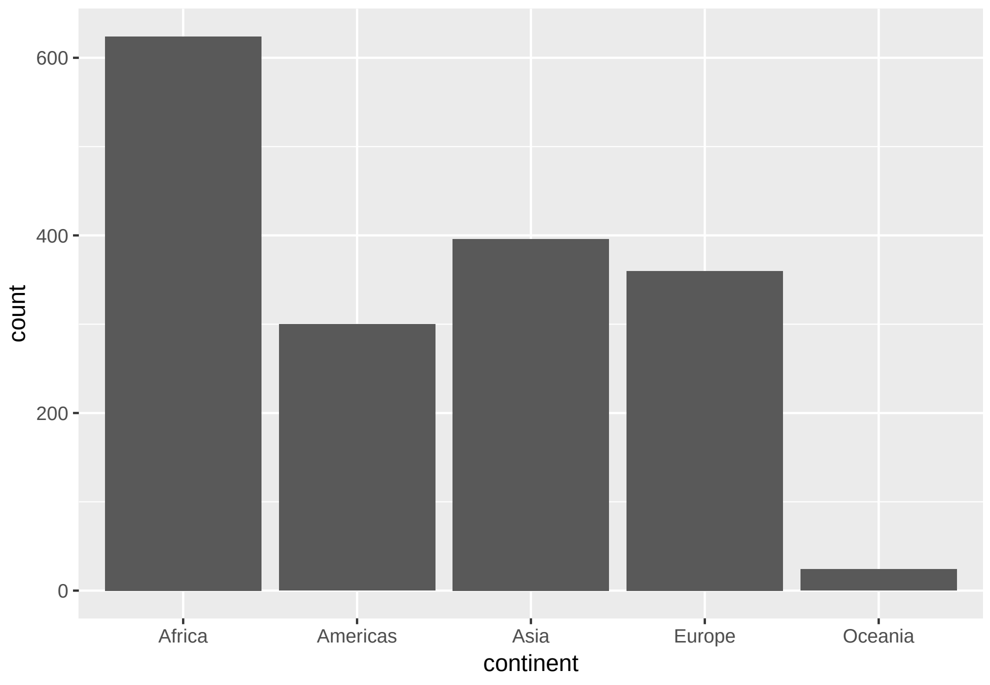
To examine the distribution of a categorical variable, we can use a bar chart.

```
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:

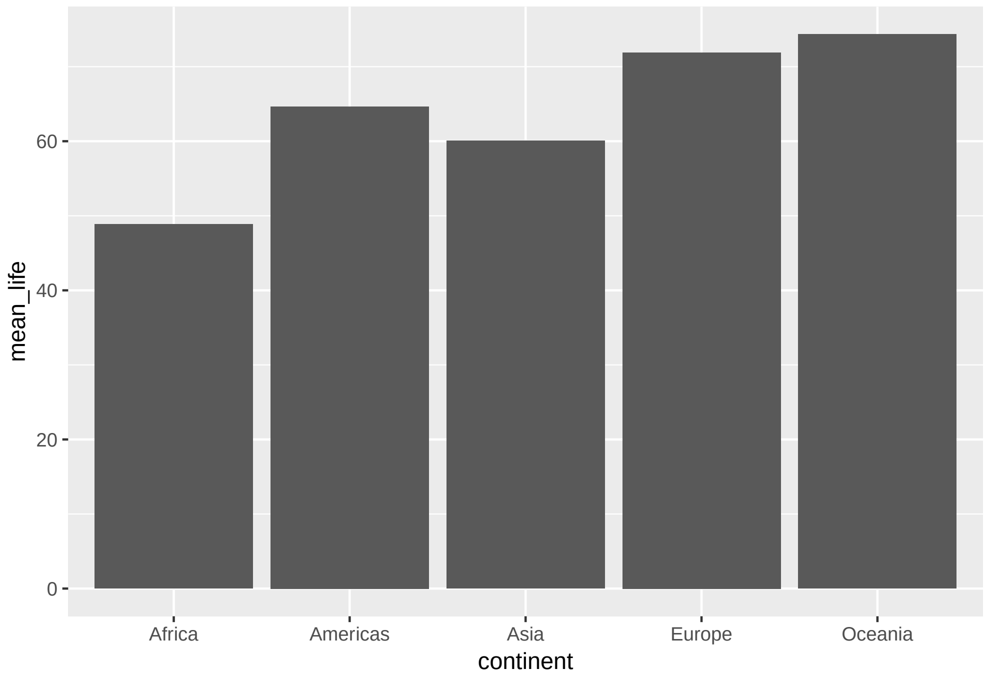
```
gapminder_life <- gapminder %>% group_by(continent) %>%
  summarize(mean_life = mean(lifeExp))
```

```
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
```

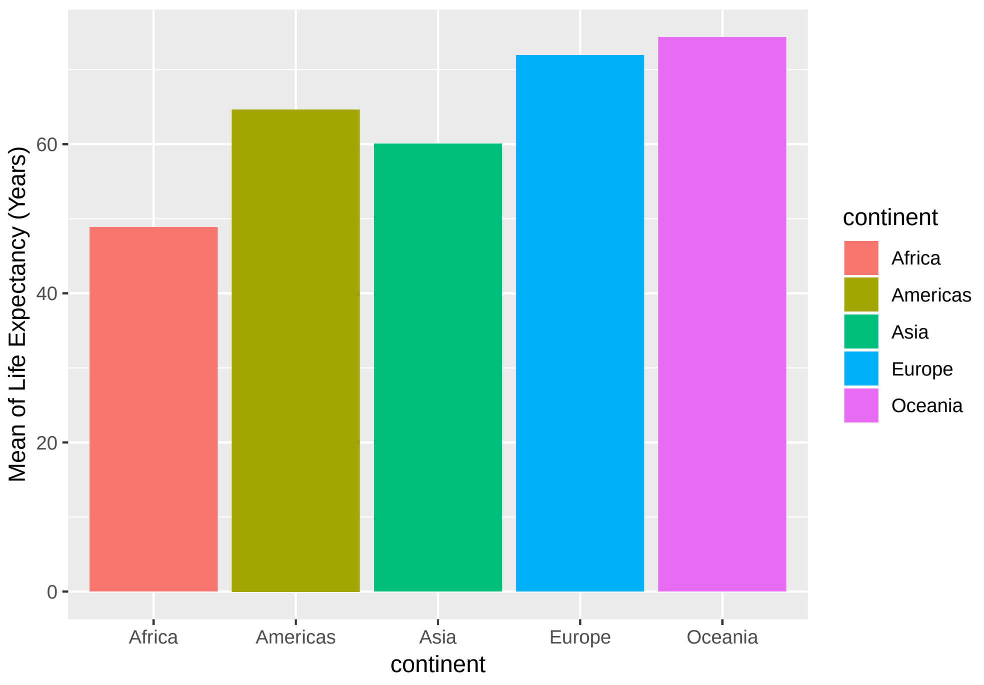
ggplot part:

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



dplyr and ggplot in action:

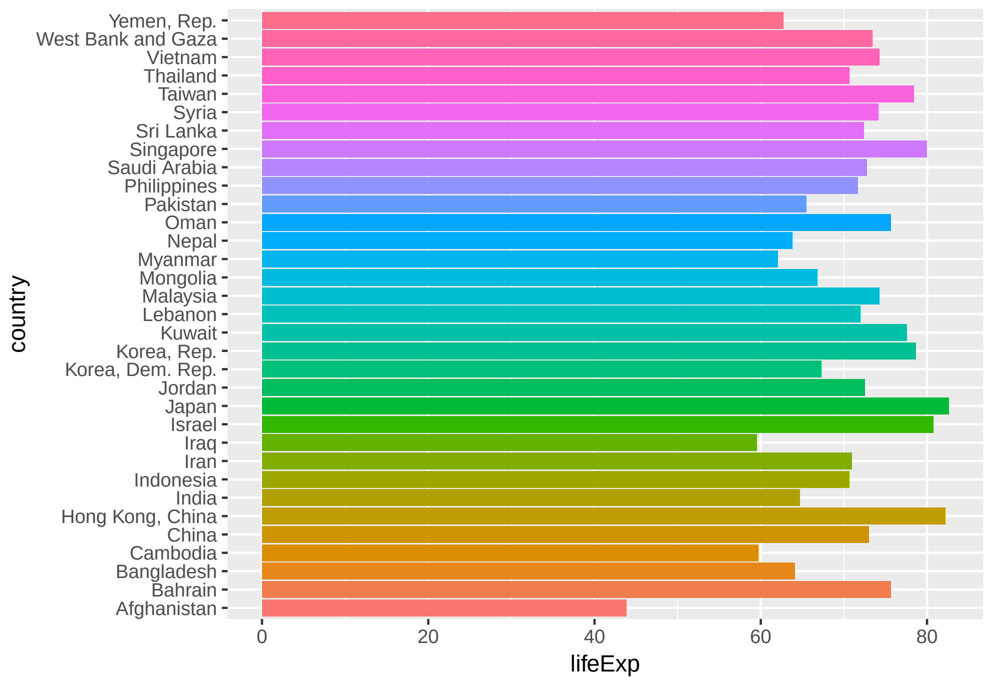
```
gapminder %>% group_by(continent) %>% summarize(mean_life = mean(lifeExp)) %>%
  ggplot(mapping = aes(x = continent, y = mean_life, fill = continent)) + geom_col() +
  ylab("Mean of Life Expectancy (Years)")
```



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!!

3.6 One variable: Distribution of a numerical variable

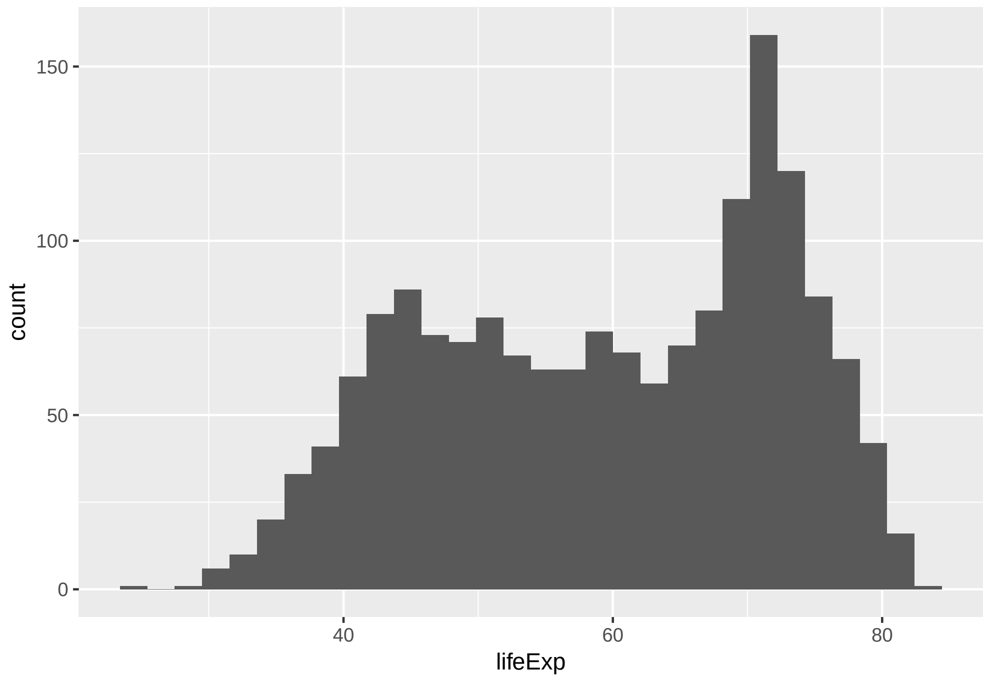
Plot distribution of values of a numerical variable

3.6.1 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.

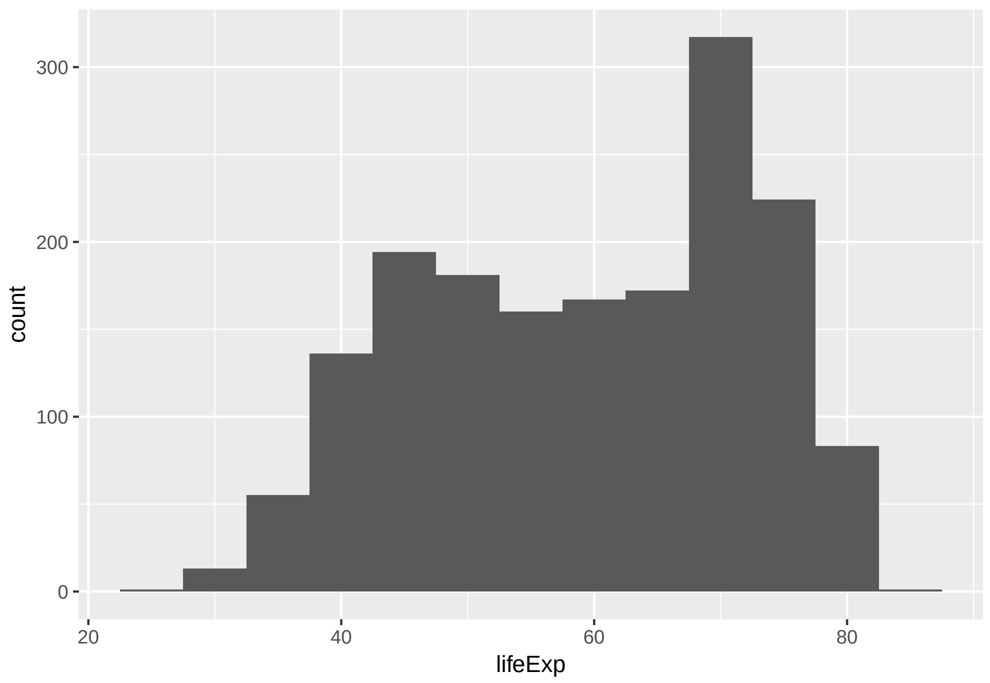
```r
ggplot(data = gapminder, mapping = aes(x = lifeExp)) + geom_histogram()
```

```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```



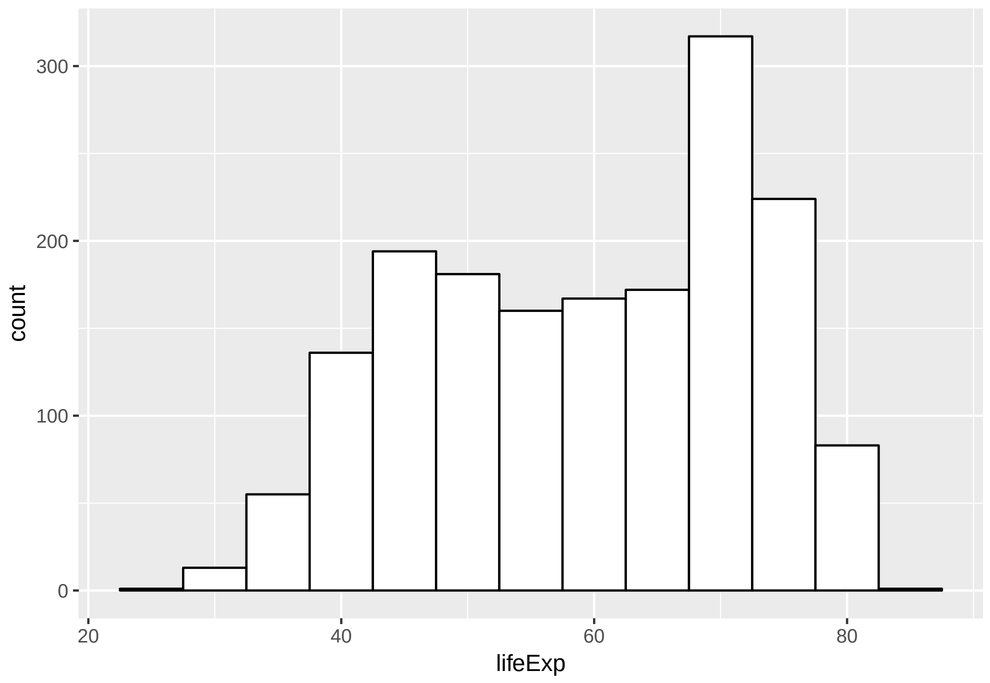
To specify the number of bin, we can use bindwidth

```r
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.

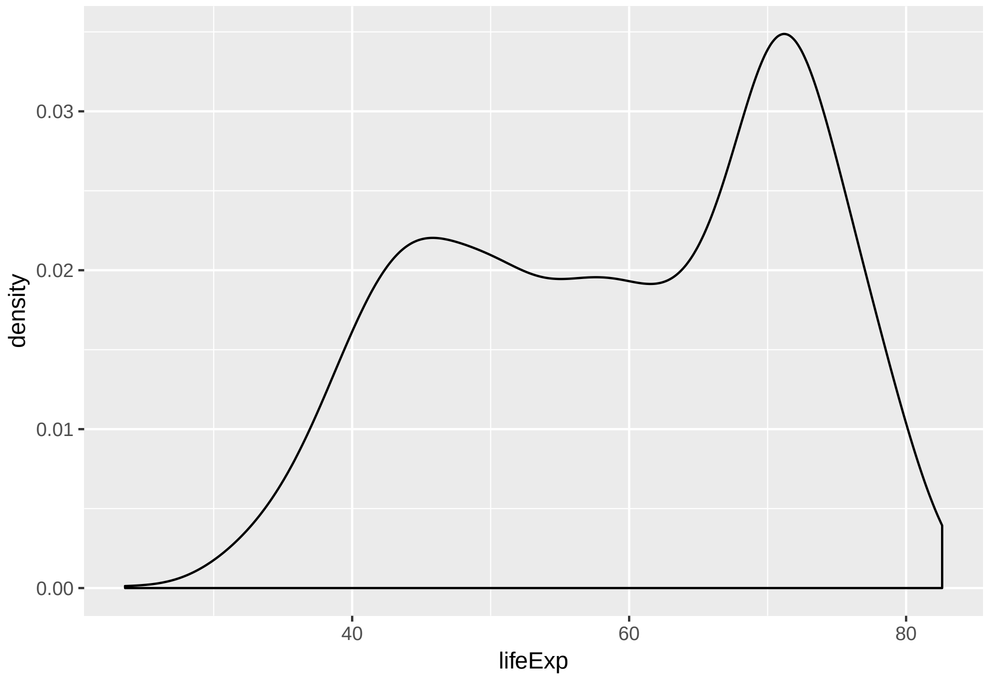
```
ggplot(data = gapminder, mapping = aes(x = lifeExp)) +
  geom_histogram(binwidth = 5, colour = "black", fill = "white")
```



3.6.2 Density curve

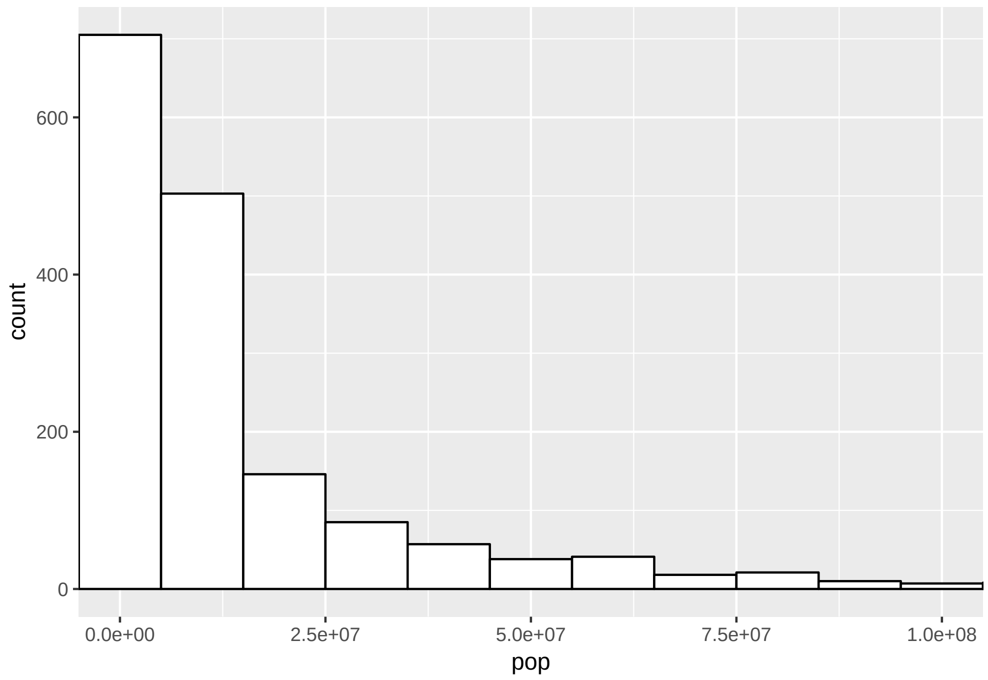
Let us create a density curve. Density curve is useful to examine the distribution of observations.

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



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

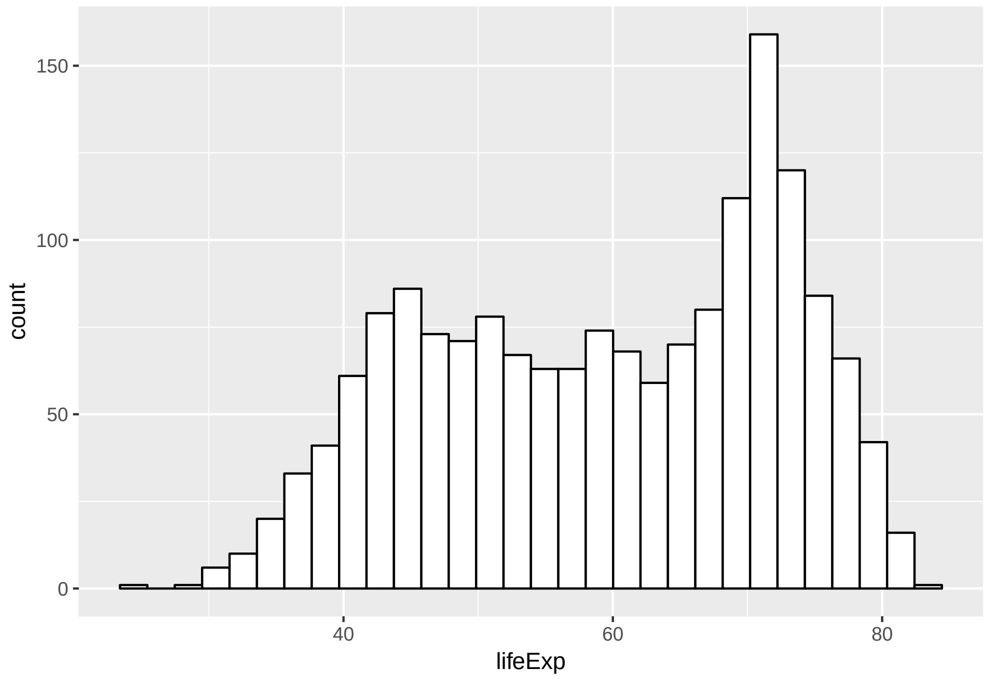
```r
ggplot(data = gapminder, mapping = aes(x = pop)) +
  geom_histogram(binwidth = 10000000, colour = "black", fill = "white") +
  coord_cartesian(xlim = c(0, 100000000))
```



Histogram for life expectancy

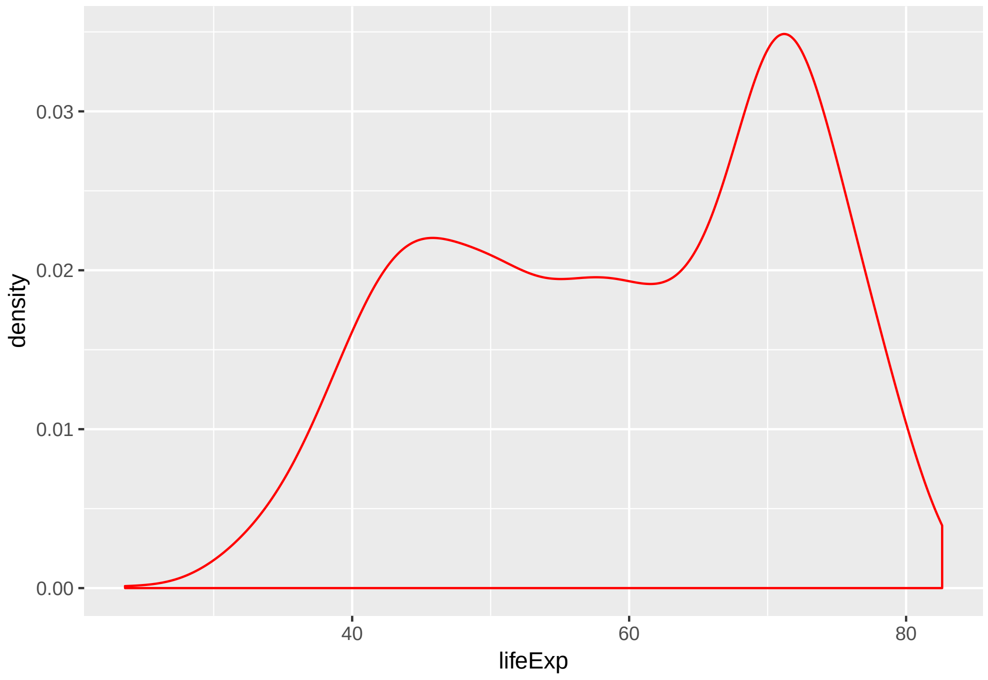
```r
ggplot(data = gapminder, mapping = aes(x = lifeExp)) +
  geom_histogram(fill = "white", colour = "black")
```

```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```



Density curve for life expectancy

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

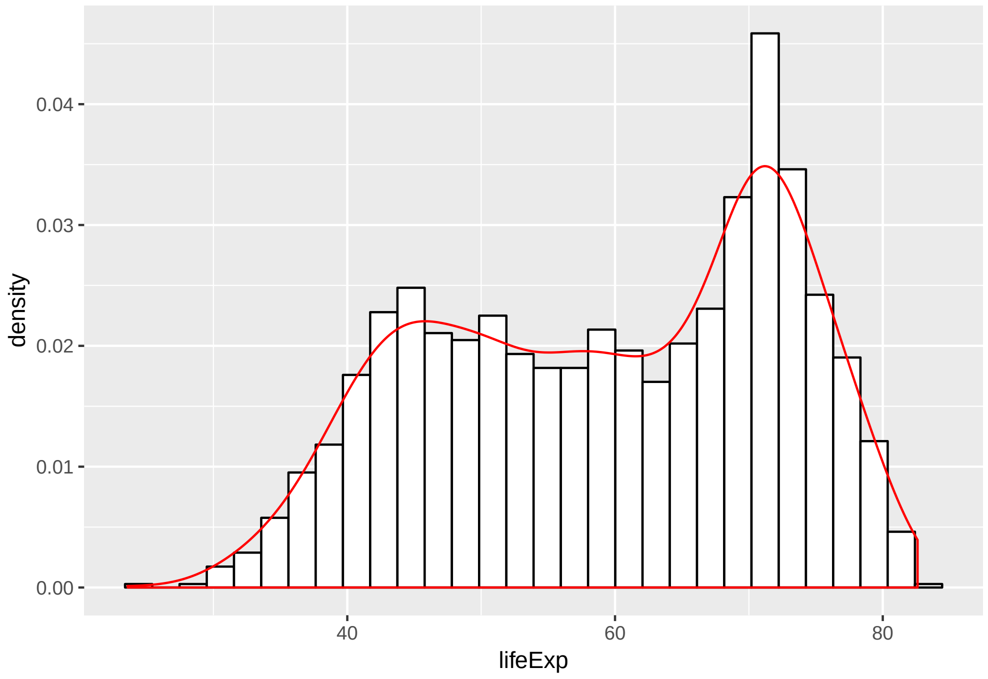


3.6.3 Histogram and density curve together

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")
```

```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```



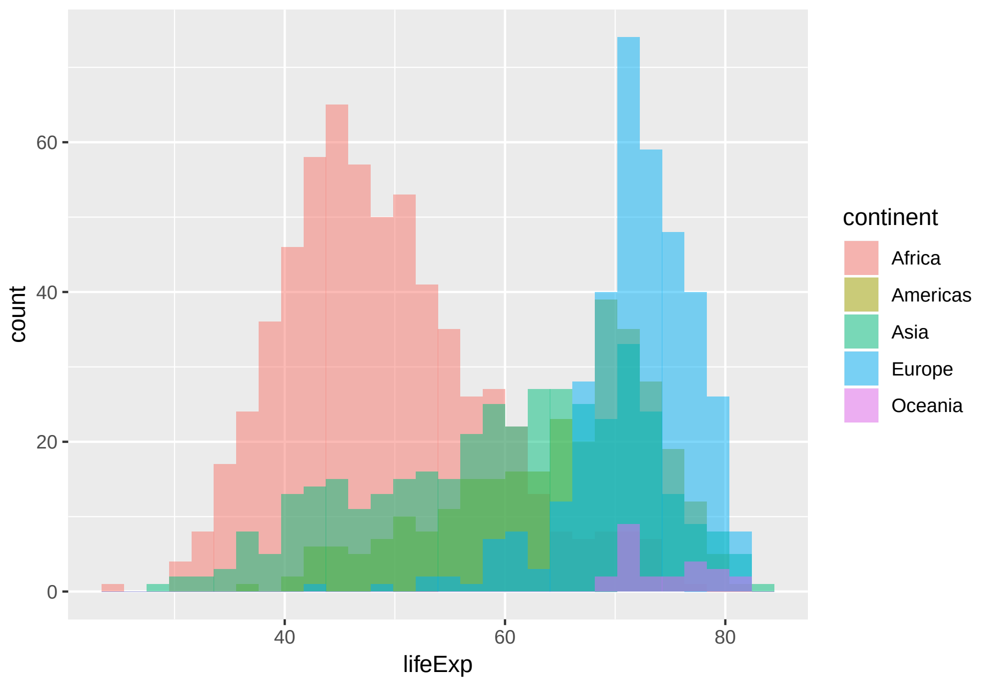
3.7 Two variables: Plotting a numerical and a categorical variable

3.7.1 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)
```

```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```



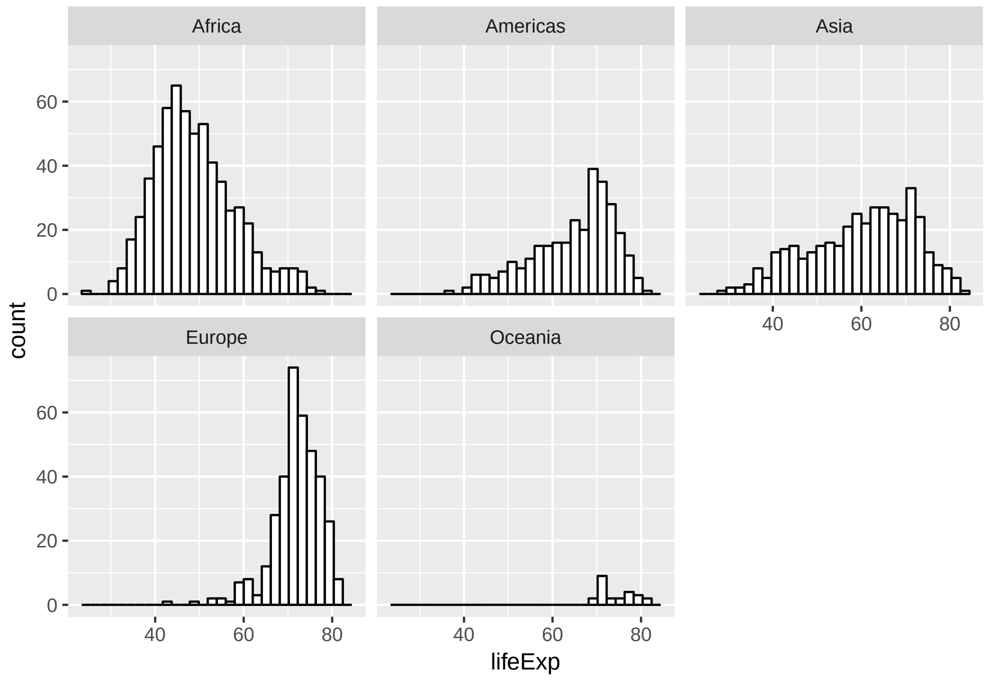
3.7.2 Using facet

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)
```

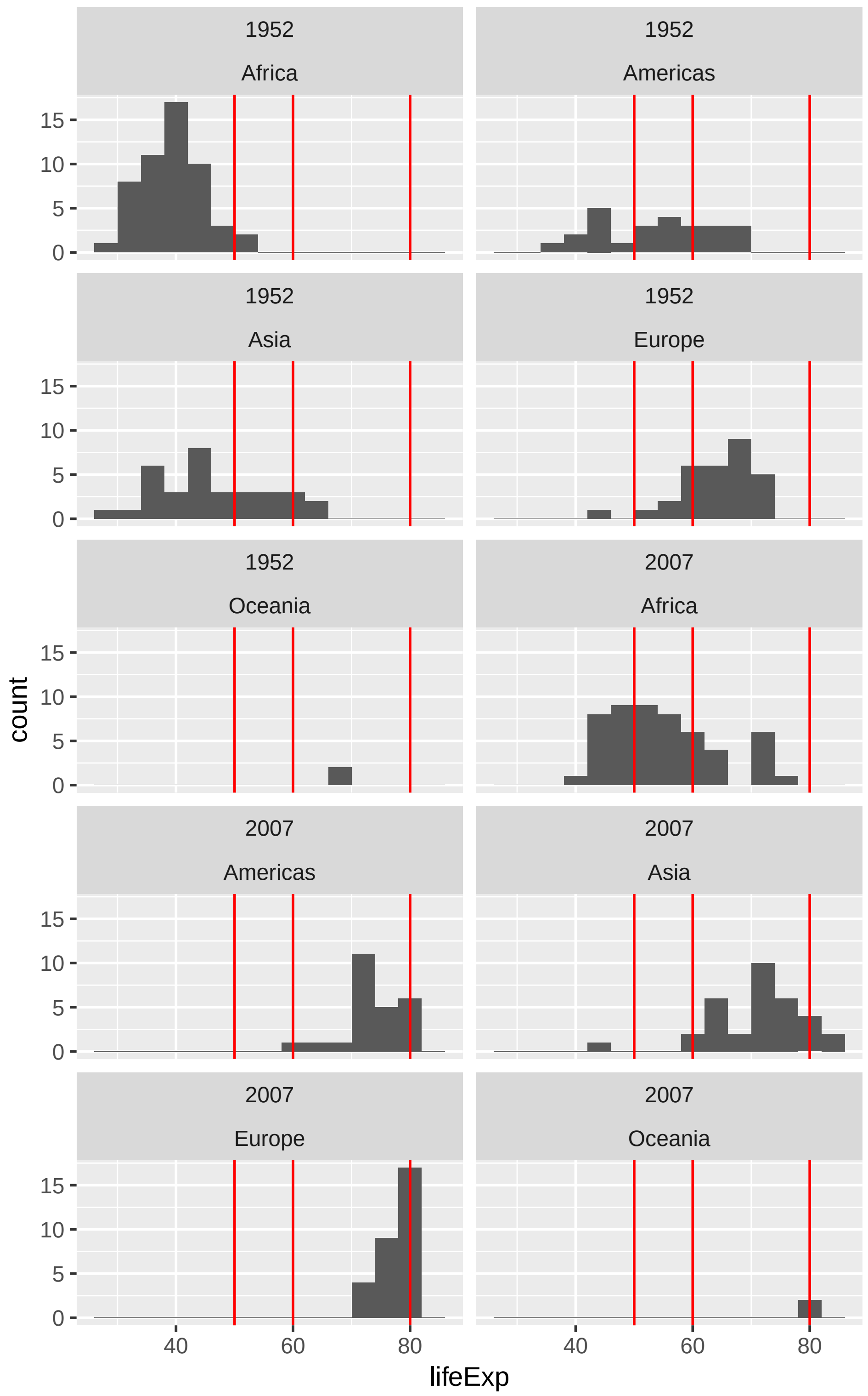
```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```



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")
```



3.8 Box plot

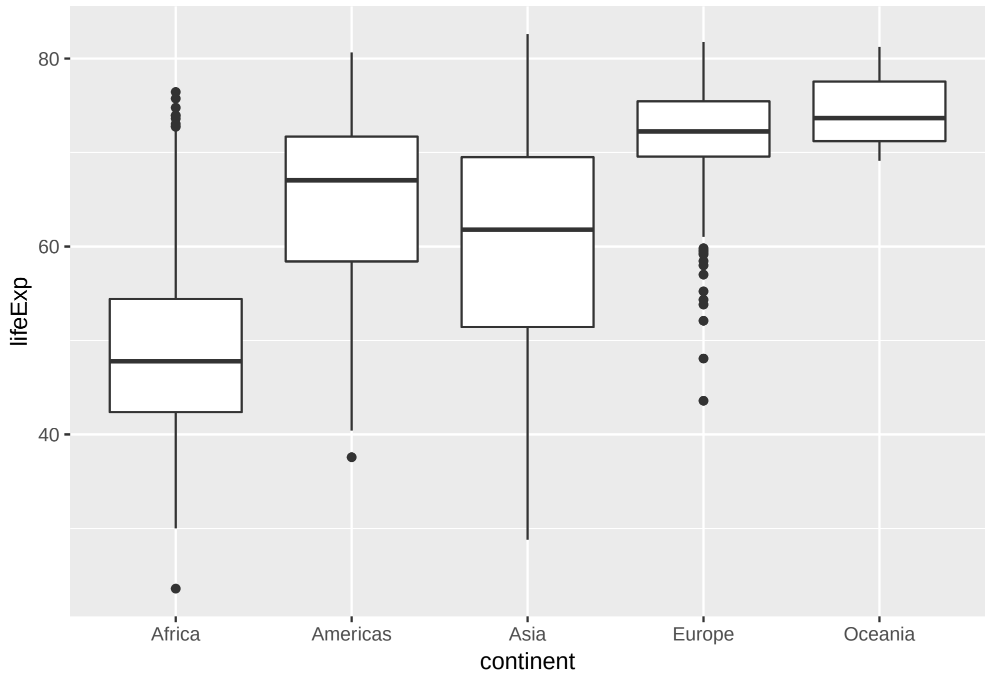
Box plot contains a box and whiskers. The box goes from the 25th percentile to the 75th percentile of the data, also known as the inter-quartile range (IQR).

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.

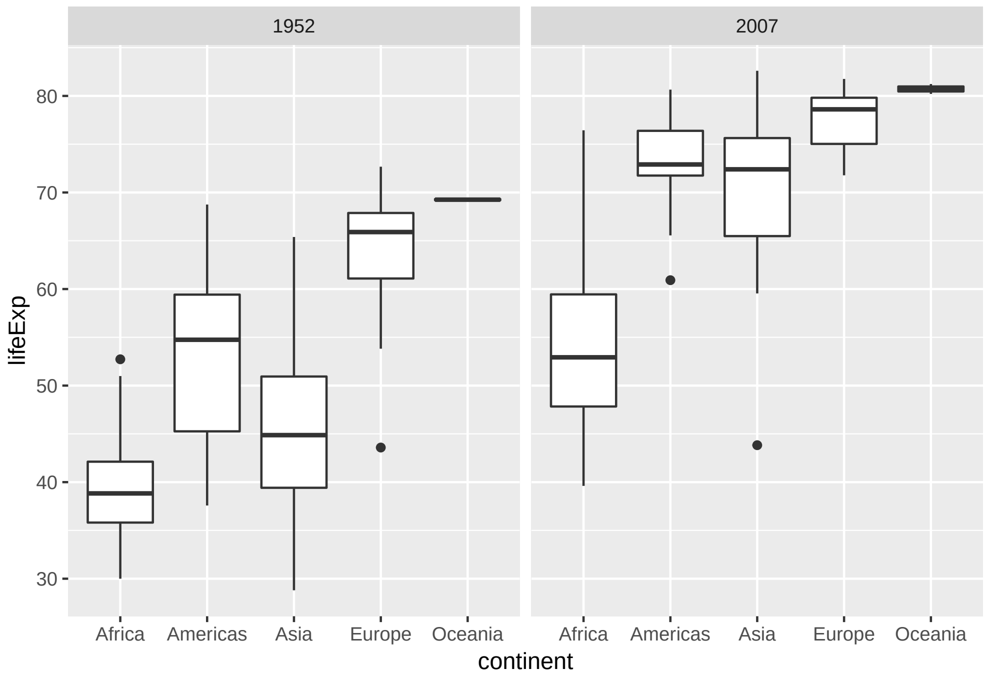
Outliers are points that stay beyond the end of whiskers

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



For 1952 and 2007

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



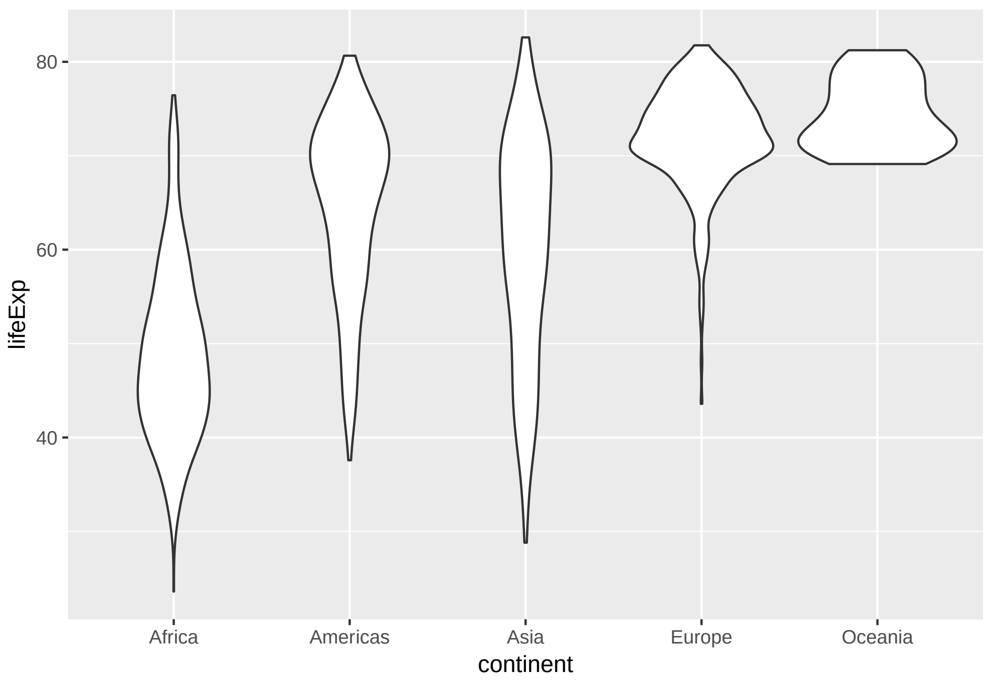
3.9 Violin plot

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

Violin plots are a way of comparing multiple data distributions.

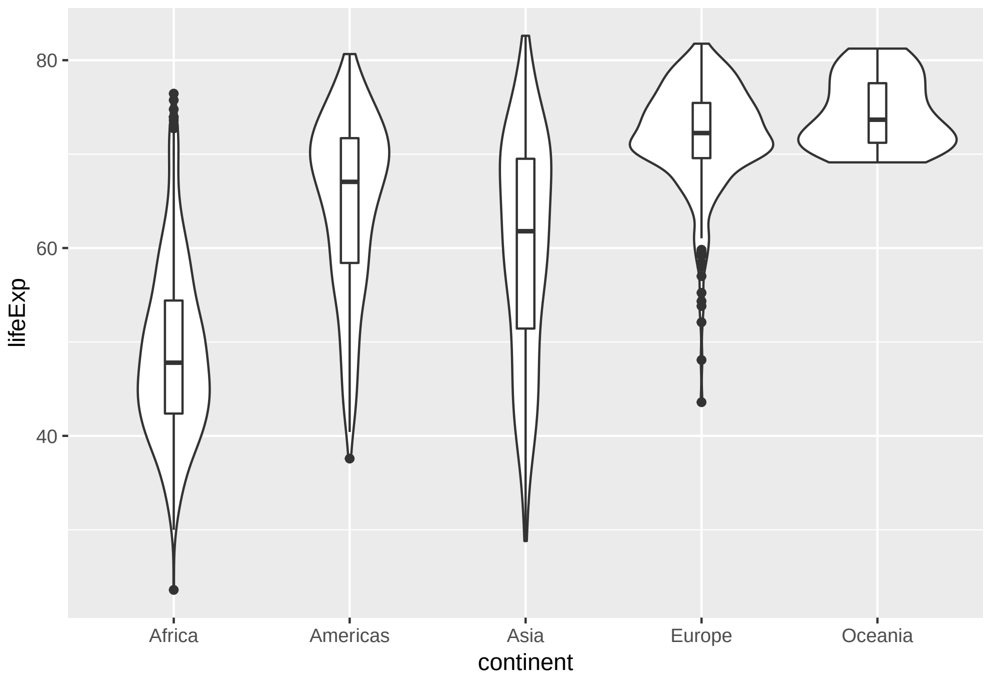
A violin plot is a kernel density estimate, mirrored so that it forms a symmetrical shape.

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



We can overlay with box plot

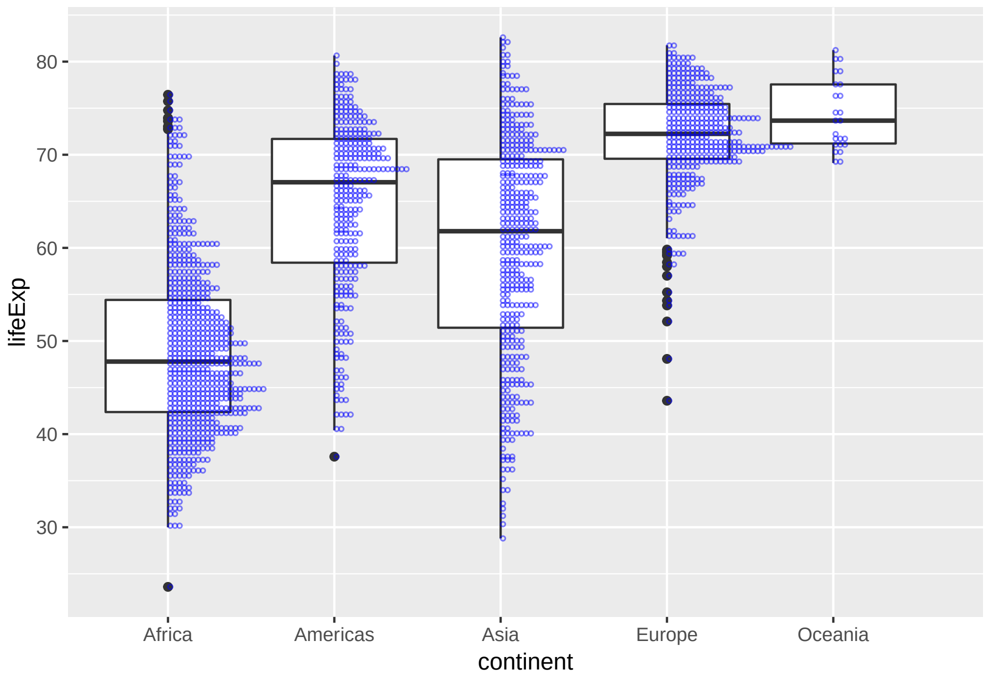
```
my_violin + geom_boxplot(width = .1)
```



3.10 Dot plot

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

```r
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)
```



3.11 Line plot

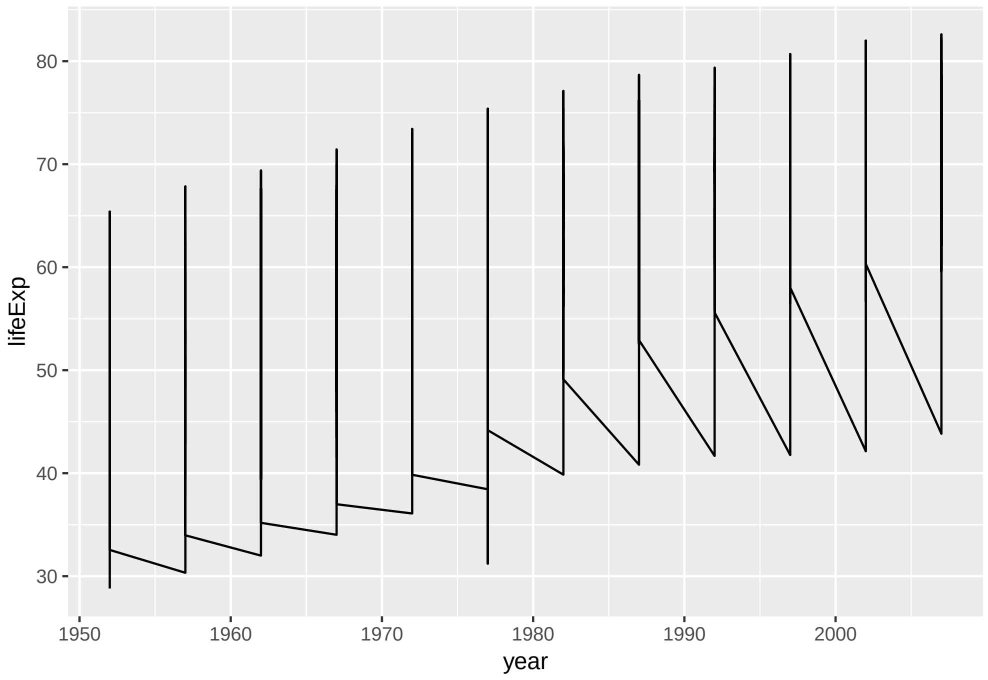
Line graphs are typically used for visualizing how one continuous variable, on the y-axis, changes in relation to another continuous variable, on the x-axis.

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

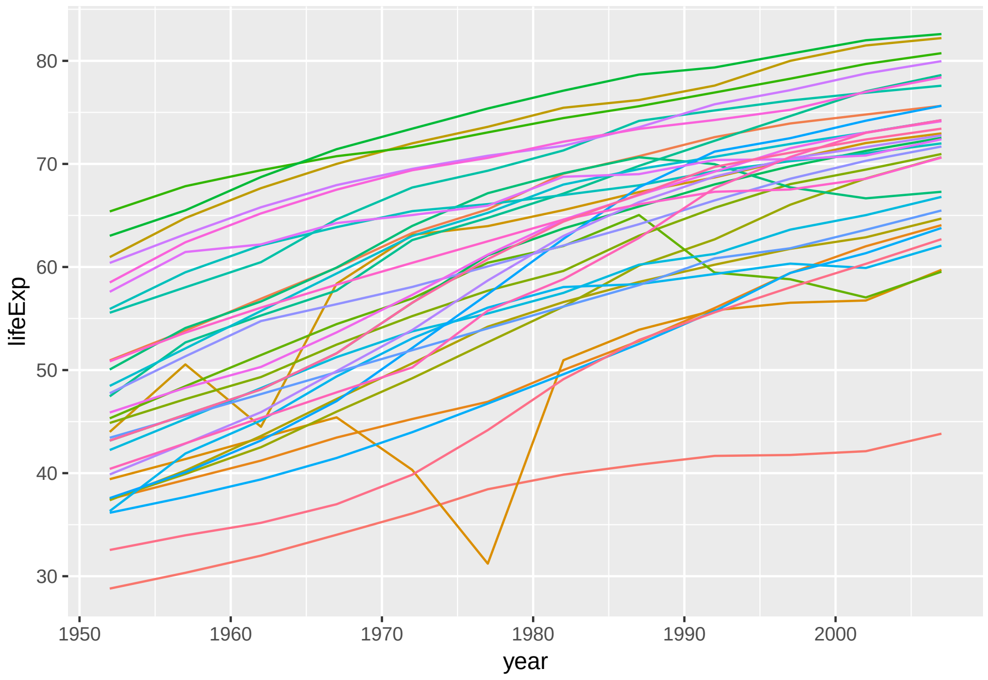
1. gapminder data
2. continent == “Asia”
3. life expectancy
4. trend

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



Does not seem right,

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



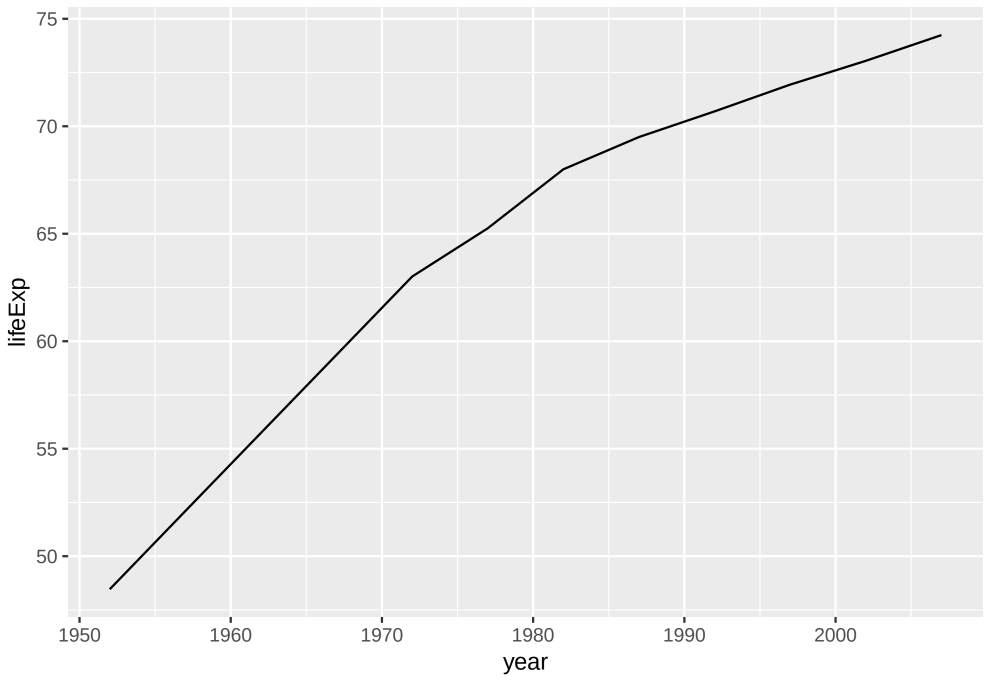
Let us make a line graph for only Malaysia

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()
```



3.12 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))
```

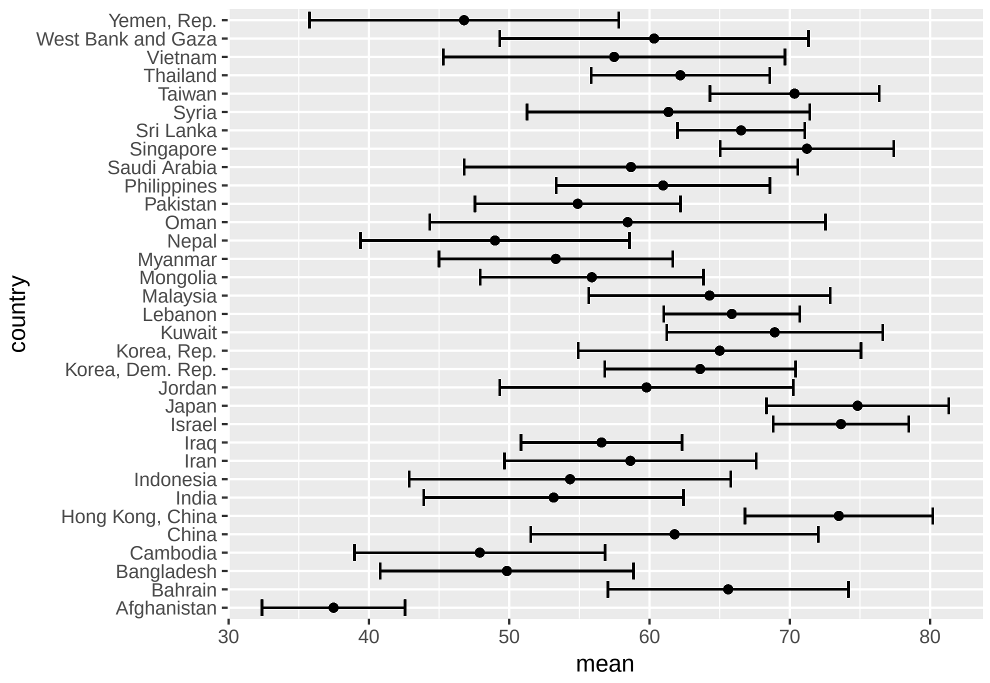
Let is check the result

```
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()
```

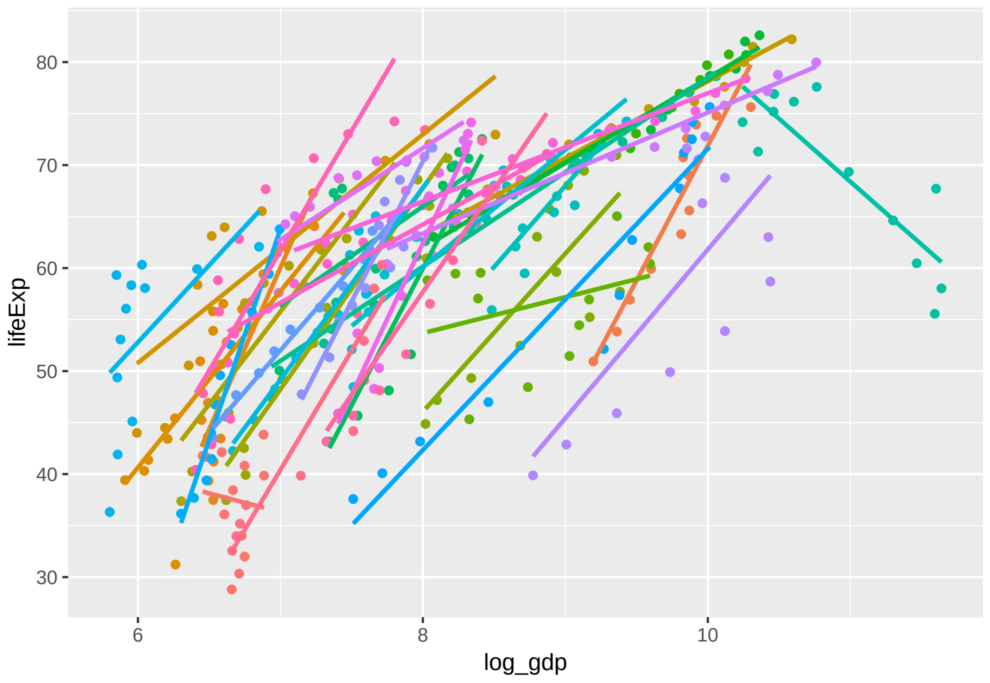


3.13 Scatterplot with fit line

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

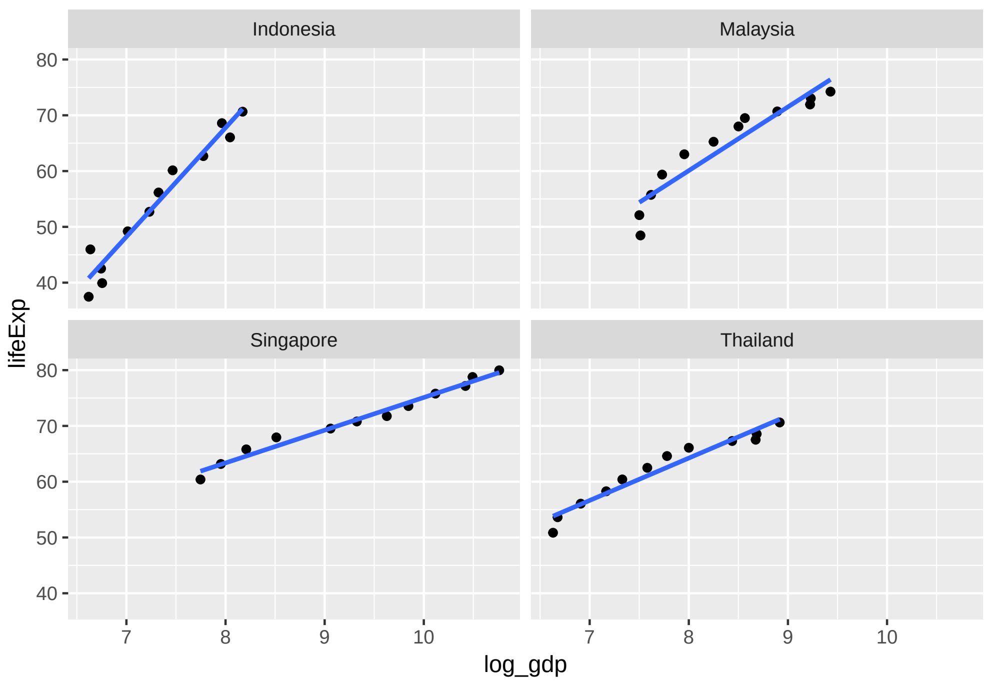
```
gapminder %>% mutate(log_gdp = log(gdpPercap)) %>%
  filter(continent == "Asia") %>%
ggplot(mapping = aes(x = log_gdp, y = lifeExp, colour = country)) +
  geom_point(show.legend = FALSE) +
  geom_smooth(method = lm, se = FALSE, show.legend = FALSE)
```



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)
```

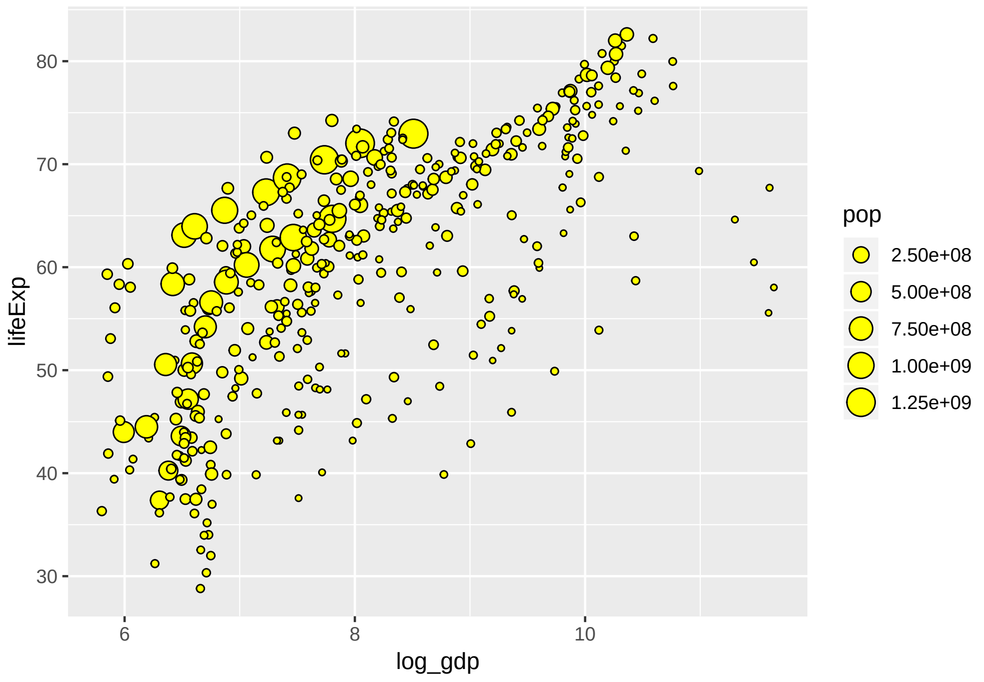


3.14 Balloon plot

Relationship between log_gdp and life expectancy

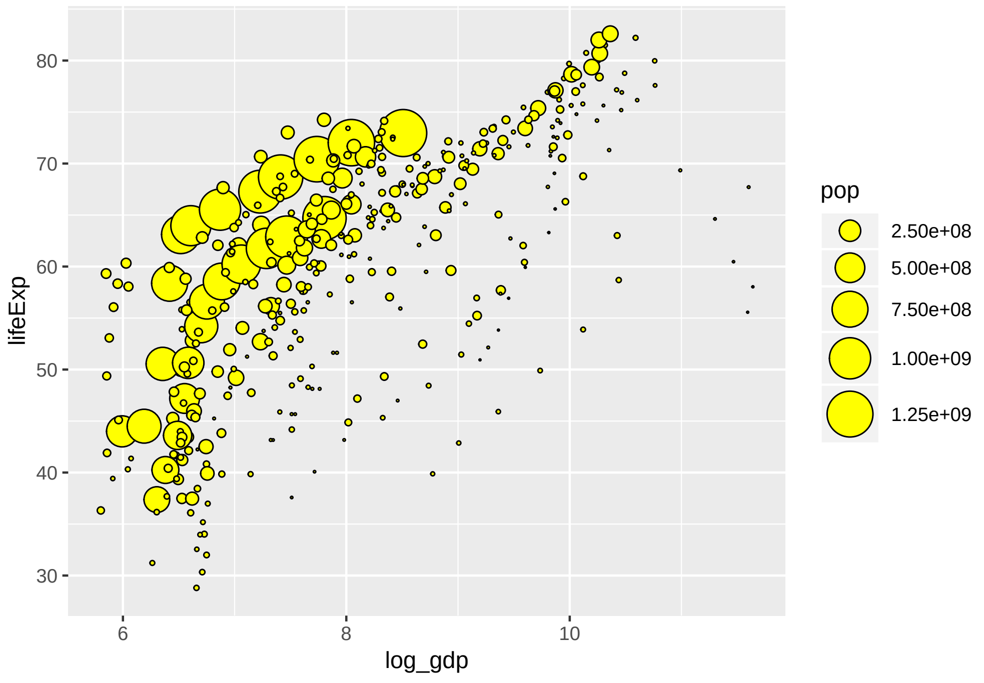
- gapminder data for countries in the Asia continent
- variable log_gdp as the predictor
- variable life expectancy as the outcome
- variable population for the size
- using scatterplot

```
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)
```



4 Linear regression

1. A statistical method to model relationship between:

- outcome: numerical variable.
- predictors/independent variables: numerical, categorical variables.

2. A type of Generalized Linear Models (GLMs), which also includes other outcome types, e.g. categorical and count.
3. Basically, the linear relationship is structured as follows,

$$numerical\ outcome = numerical\ predictors + categorical\ predictors$$

4.1 Simple linear regression (SLR)

4.1.1 About SLR

1. Model *linear* (straight line) relationship between:

   - outcome: numerical variable.
   - a predictor: numerical variable (only).

   Note: What if the predictor is a categorical variable? Remember, we already handled that with one-way ANOVA.

2. Formula,

   $$numerical\ outcome = intercept + coefficient \times numerical\ predictor$$

   in short,

   $$\hat{y} = \beta_0 + \beta_1 x_1$$

   where $\hat{y}$ is the predicted value of the outcome y.

4.1.2 Analysis

4.1.2.1 Libraries

```
# library
library(foreign)
library(epiDisplay)
```

```
## Loading required package: survival

## Loading required package: MASS

##
## Attaching package: 'MASS'

## The following object is masked from 'package:dplyr':
##
##     select

## Loading required package: nnet

##
## Attaching package: 'epiDisplay'

## The following object is masked from 'package:ggplot2':
##
##     alpha
```

```
library(psych)
```

```
##
## Attaching package: 'psych'

## The following objects are masked from 'package:epiDisplay':
##
##     alpha, cs, lookup

## The following objects are masked from 'package:ggplot2':
##
##     %+%, alpha
```

```r
library(lattice)
```

```
##
## Attaching package: 'lattice'

## The following object is masked from 'package:epiDisplay':
##
##     dotplot
```

```r
library(rsq)
library(MASS)
library(car)
```

```
## Loading required package: carData

##
## Attaching package: 'car'

## The following object is masked from 'package:psych':
##
##     logit

## The following object is masked from 'package:dplyr':
##
##     recode

## The following object is masked from 'package:purrr':
##
##     some
```

```r
library(broom)
```

4.1.2.2 Data set

```r
# data
coronary = read.dta(here::here("data", "coronary.dta"))
str(coronary)
```

```
## 'data.frame':    200 obs. of  9 variables:
##  $ id    : num  1 14 56 61 62 64 69 108 112 134 ...
##  $ cad   : Factor w/ 2 levels "no cad","cad": 1 1 1 1 1 1 2 1 1 1 ...
##  $ sbp   : num  106 130 136 138 115 124 110 112 138 104 ...
##  $ dbp   : num  68 78 84 100 85 72 80 70 85 70 ...
##  $ chol  : num  6.57 6.33 5.97 7.04 6.66 ...
##  $ age   : num  60 34 36 45 53 43 44 50 43 48 ...
##  $ bmi   : num  38.9 37.8 40.5 37.6 40.3 ...
##  $ race  : Factor w/ 3 levels "malay","chinese",..: 3 1 1 1 3 1 1 2 2 2 ...
##  $ gender: Factor w/ 2 levels "woman","man": 1 1 1 1 2 2 2 1 1 2 ...
##  - attr(*, "datalabel")= chr "Written by R.              "
##  - attr(*, "time.stamp")= chr ""
##  - attr(*, "formats")= chr  "%9.0g" "%9.0g" "%9.0g" "%9.0g" ...
##  - attr(*, "types")= int  100 108 100 100 100 100 100 108 108
##  - attr(*, "val.labels")= chr  "" "cad" "" "" ...
##  - attr(*, "var.labels")= chr  "id" "cad" "sbp" "dbp" ...
##  - attr(*, "version")= int 7
##  - attr(*, "label.table")=List of 3
##   ..$ cad   : Named int  1 2
##   .. ..- attr(*, "names")= chr  "no cad" "cad"
##   ..$ race  : Named int  1 2 3
##   .. ..- attr(*, "names")= chr  "malay" "chinese" "indian"
##   ..$ gender: Named int  1 2
##   .. ..- attr(*, "names")= chr  "woman" "man"
```

4.1.3 Data exploration

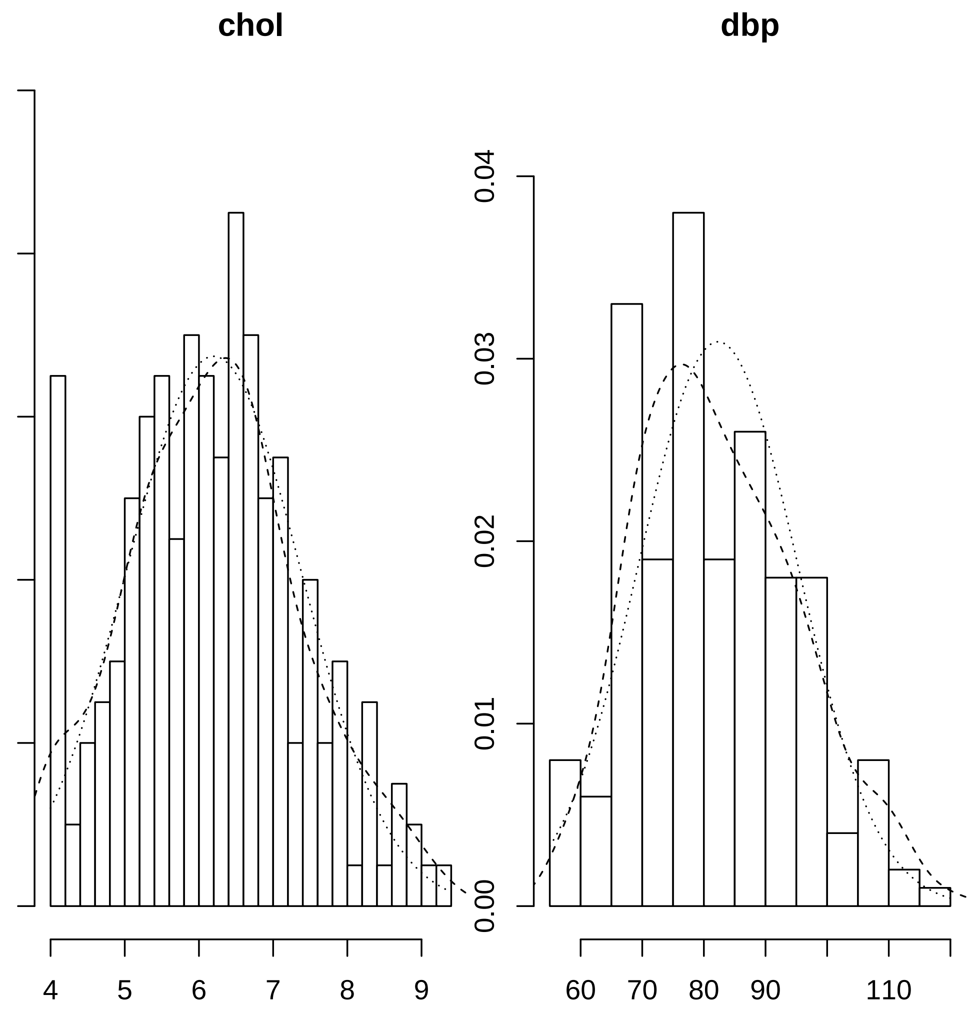
4.1.3.1 Descriptive statistics

```r
summ(coronary[c("chol", "dbp")])
```

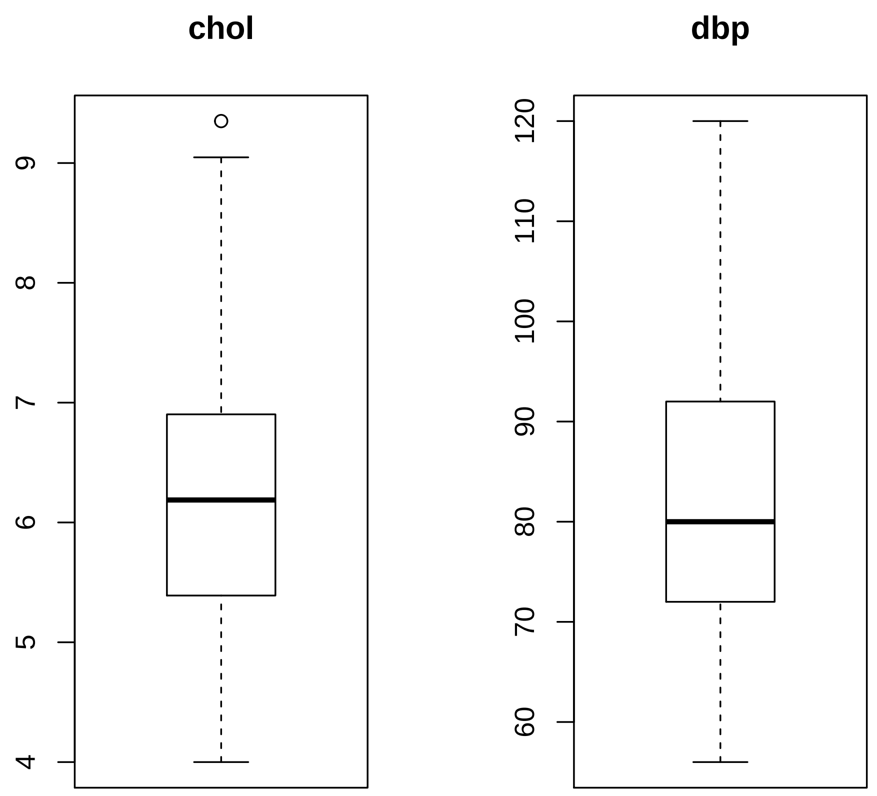
```
## 
## No. of observations = 200
## 
##   Var. name obs. mean   median  s.d.   min.   max.  
## 1 chol      200  6.2    6.19    1.18   4      9.35  
## 2 dbp       200  82.31  80      12.9   56     120
```

4.1.3.2 Plots

```r
multi.hist(coronary[c("chol", "dbp")], ncol = 2)
```



```r
par(mfrow = c(1, 2))
mapply(boxplot, coronary[c("chol", "dbp")], main = colnames(coronary[c("chol", "dbp")]))
```



```
##       chol      dbp
## stats Numeric,5 Numeric,5
## n     200       200
## conf  Numeric,2 Numeric,2
## out   9.35      Numeric,0
## group 1         Numeric,0
## names ""        ""
```

```
par(mfrow = c(1, 1))
```

4.1.4 Univariable

Fit model,

```
# model: chol ~ dbp
slr_chol = glm(chol ~ dbp, data = coronary)
summary(slr_chol)
```

```
##
## Call:
## glm(formula = chol ~ dbp, data = coronary)
##
```

```
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -1.9967  -0.8304  -0.1292   0.7734   2.8470
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) 2.995134   0.492092   6.087 5.88e-09 ***
## dbp         0.038919   0.005907   6.589 3.92e-10 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for gaussian family taken to be 1.154763)
##
##     Null deviance: 278.77  on 199  degrees of freedom
## Residual deviance: 228.64  on 198  degrees of freedom
## AIC: 600.34
##
## Number of Fisher Scoring iterations: 2
```

```
Confint(slr_chol)  # 95% CI
```

```
##               Estimate      2.5 %     97.5 %
## (Intercept) 2.99513427 2.03065127 3.95961727
## dbp         0.03891876 0.02734161 0.05049591
```

Important results,

- Coefficient, β.
- 95% CI.
- P-value.

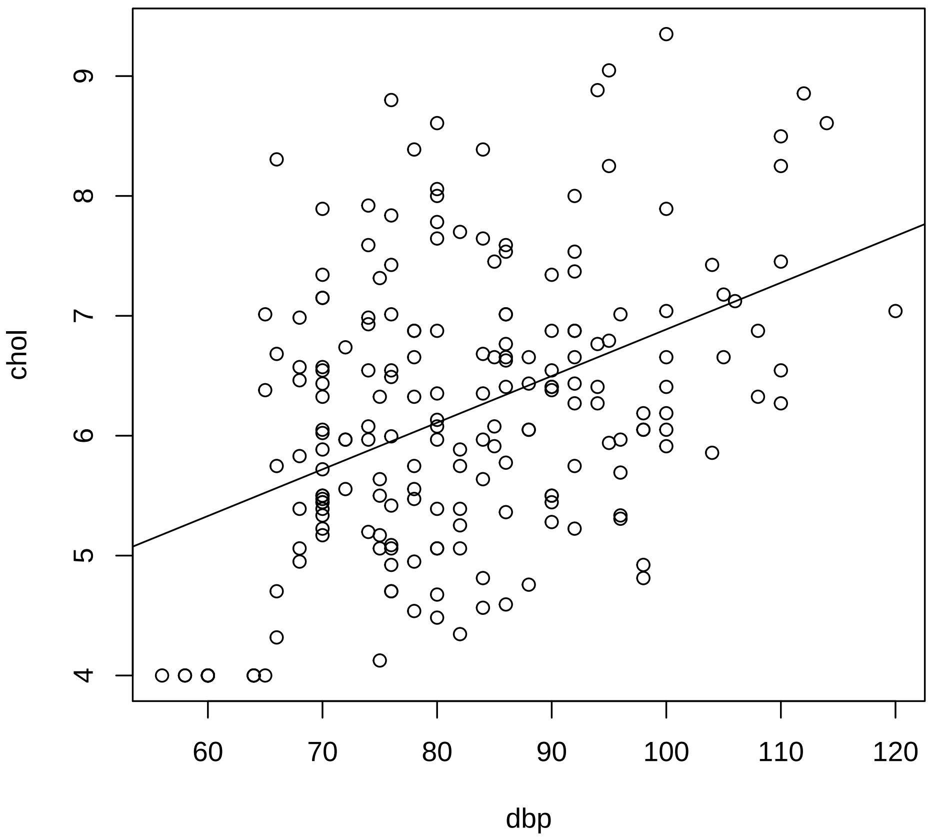
Obtain R^2, % of variance explained,

```
rsq(slr_chol, adj = T)
```

```
## [1] 0.1756834
```

Scatter plot,

```
plot(chol ~ dbp, data = coronary)
abline(slr_chol)
```



this allows assessment of normality, linearity and equal variance assumptions. We expect eliptical/oval shape (normality), equal scatter of dots on both sides of the prediction line (equal variance). Both these indicate linear relationship between `chol` and `dbp`.

4.1.4.1 Interpretation

- 1mmHg increase in DBP causes 0.04mmol/L increase in cholestrol.
- DBP explains 17.6% variance in cholestrol.

4.1.4.2 Model equation

$$chol = 3.0 + 0.04 \times dbp$$

4.2 Multiple linear regression (MLR)

4.2.1 About MLR

1. Model *linear* relationship between:
    - outcome: numerical variable.

- predictors: numerical, categorical variables.

Note: MLR is a term that refers to linear regression with two or more *numerical* variables. Whenever we have both numerical and categorical variables, the proper term for the regression model is *General Linear Model*. However, we will use the term MLR in this workshop.

2. Formula,

$$\begin{aligned} numerical\ outcome = intercept + coefficients \times numerical\ predictors \\ + coefficients \times categorical\ predictors \end{aligned}$$

in a shorter form,

$$\hat{y} = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + ... + \beta_k x_k$$

where we have k predictors.

Whenever the predictor is a categorical variable with more than two levels, we use dummy variable(s). This can be easily specified in R using `factor()` if the variable is not yet properly specified as such. There is no problem with binary categorical variable.

For a categorical variable with more than two levels, the number of dummy variables (i.e. once turned into several binary variables) equals number of levels minus one. For example, whenever we have four levels, we will obtain three dummy (binary) variables.

4.2.2 Analysis

4.2.2.1 Review data set

```r
# data
str(coronary)
```

```
## 'data.frame':	200 obs. of  9 variables:
##  $ id    : num  1 14 56 61 62 64 69 108 112 134 ...
##  $ cad   : Factor w/ 2 levels "no cad","cad": 1 1 1 1 1 1 2 1 1 1 ...
##  $ sbp   : num  106 130 136 138 115 124 110 112 138 104 ...
##  $ dbp   : num  68 78 84 100 85 72 80 70 85 70 ...
##  $ chol  : num  6.57 6.33 5.97 7.04 6.66 ...
##  $ age   : num  60 34 36 45 53 43 44 50 43 48 ...
##  $ bmi   : num  38.9 37.8 40.5 37.6 40.3 ...
##  $ race  : Factor w/ 3 levels "malay","chinese",..: 3 1 1 1 3 1 1 2 2 2 ...
##  $ gender: Factor w/ 2 levels "woman","man": 1 1 1 1 2 2 2 1 1 2 ...
##  - attr(*, "datalabel")= chr "Written by R.              "
##  - attr(*, "time.stamp")= chr ""
##  - attr(*, "formats")= chr  "%9.0g" "%9.0g" "%9.0g" "%9.0g" ...
##  - attr(*, "types")= int  100 108 100 100 100 100 100 108 108
##  - attr(*, "val.labels")= chr  "" "cad" "" "" ...
##  - attr(*, "var.labels")= chr  "id" "cad" "sbp" "dbp" ...
##  - attr(*, "version")= int 7
##  - attr(*, "label.table")=List of 3
##   ..$ cad   : Named int  1 2
##   .. ..- attr(*, "names")= chr  "no cad" "cad"
##   ..$ race  : Named int  1 2 3
##   .. ..- attr(*, "names")= chr  "malay" "chinese" "indian"
##   ..$ gender: Named int  1 2
##   .. ..- attr(*, "names")= chr  "woman" "man"
```

We exclude `id`, `cad` and `age` from our data for the purpose of this analysis, keeping only `sbp` , `dbp`, `bmi`, `race` and `gender`. We will add `age` later in the exercise.

```
coronary = subset(coronary, select = -c(id, cad, age))
# remove id, cad, age from our data since we're not going to use them, easier to specifiy
# multivariable model.
```

4.2.2.2 Data exploration

4.2.2.2.1 Descriptive statistics

```
summ(coronary[c("chol", "sbp", "dbp", "bmi")])
```

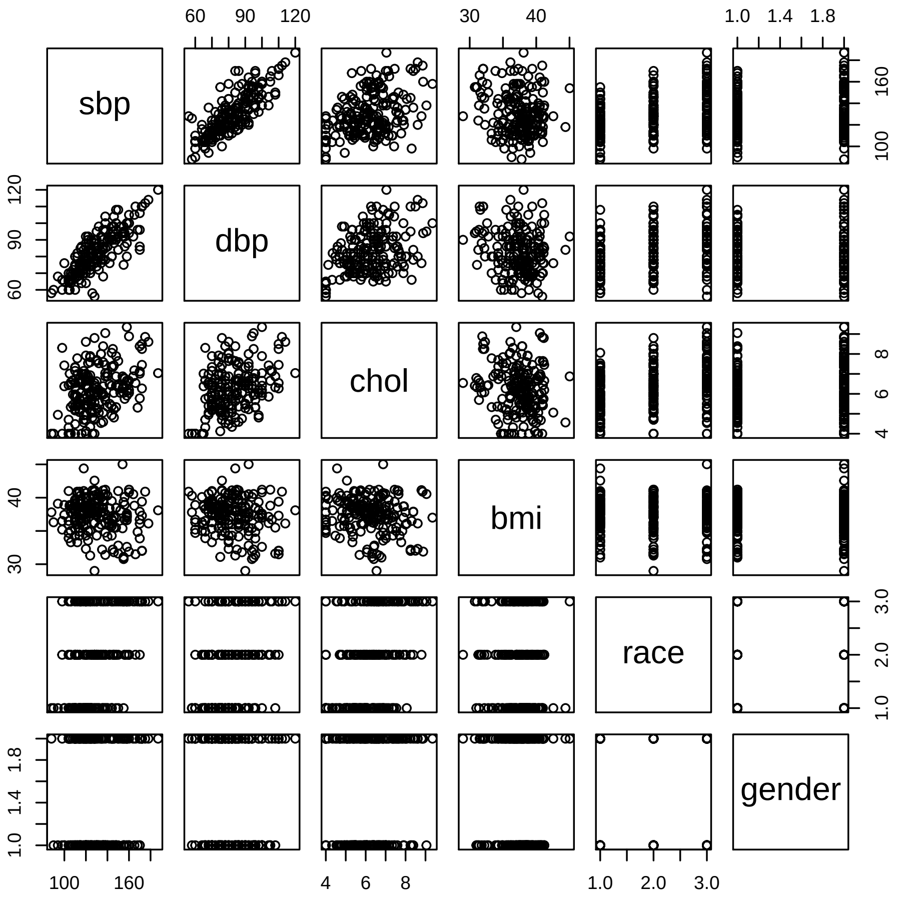
```
##
## No. of observations = 200
##
##   Var. name obs. mean   median  s.d.   min.   max.
## 1 chol      200  6.2    6.19    1.18   4      9.35
## 2 sbp       200  130.18 126     19.81  88     187
## 3 dbp       200  82.31  80      12.9   56     120
## 4 bmi       200  37.45  37.8    2.68   28.99  45.03
```

```
codebook(coronary[c("race", "gender")])
```

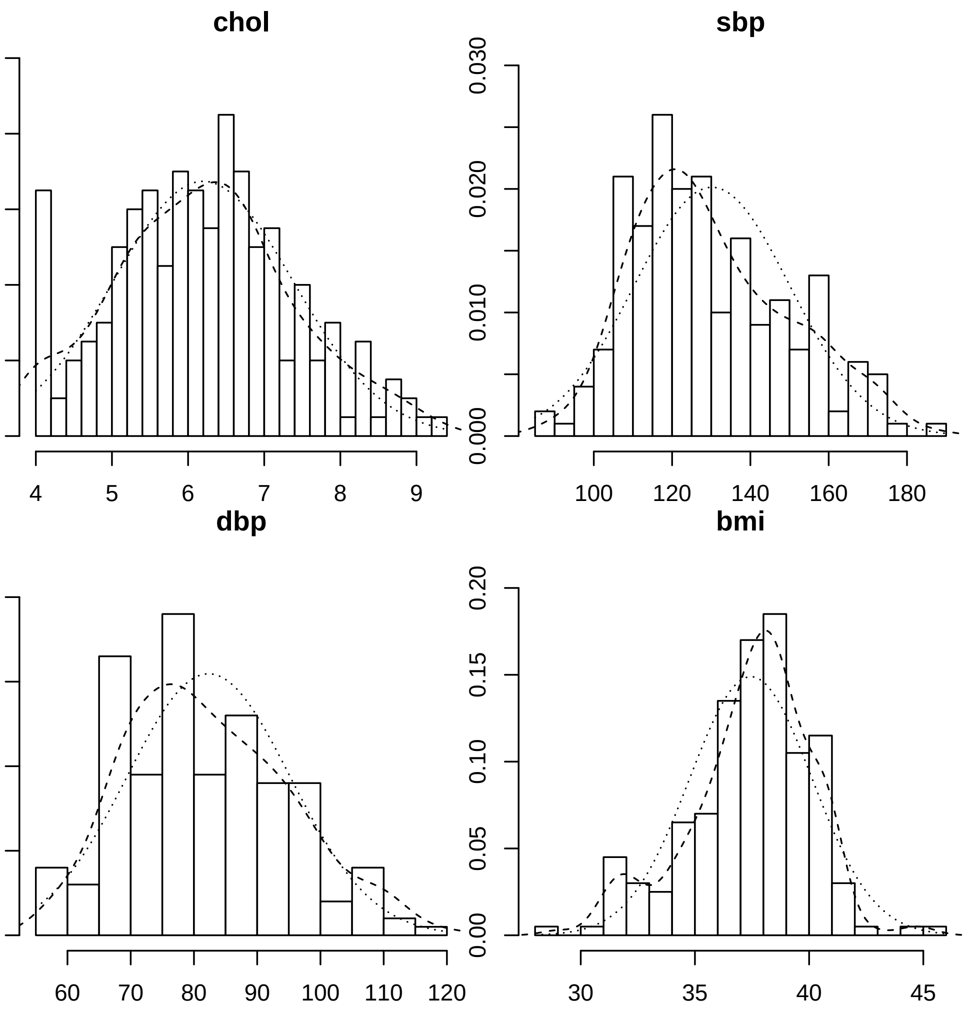
```
##
##
##
## race      :
##         Frequency Percent
## malay          73    36.5
## chinese        64    32.0
## indian         63    31.5
##
##  ==================
## gender    :
##       Frequency Percent
## woman       100      50
## man         100      50
##
##  ==================
```

4.2.2.2.2 Plots

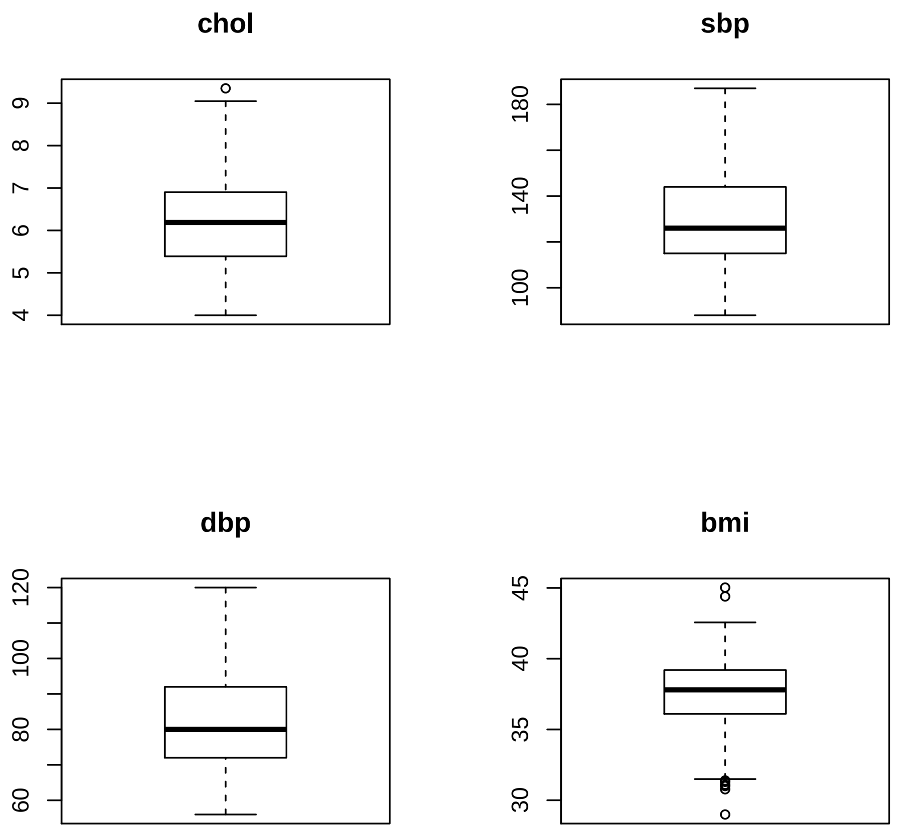
```
plot(coronary)
```



```
multi.hist(coronary[c("chol", "sbp", "dbp", "bmi")])
```

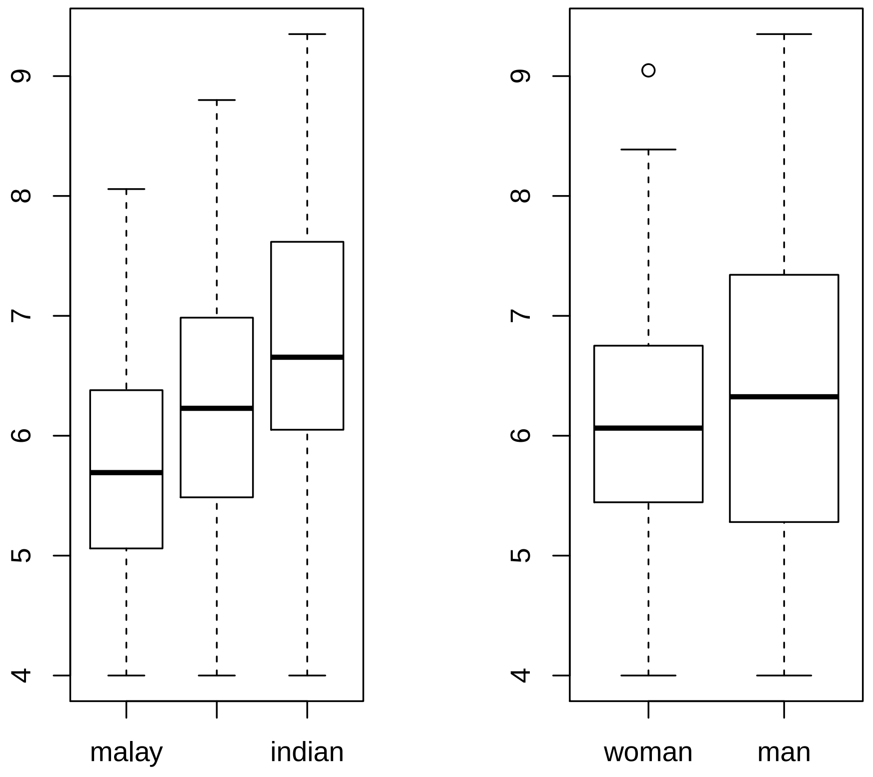


```r
par(mfrow = c(2, 2))
mapply(boxplot, coronary[c("chol", "sbp", "dbp", "bmi")], main = colnames(coronary[c("chol",
    "sbp", "dbp", "bmi")]))
```



```
##       chol      sbp       dbp       bmi
## stats Numeric,5 Numeric,5 Numeric,5 Numeric,5
## n     200       200       200       200
## conf  Numeric,2 Numeric,2 Numeric,2 Numeric,2
## out   9.35      Numeric,0 Numeric,0 Numeric,8
## group 1         Numeric,0 Numeric,0 Numeric,8
## names ""        ""        ""        ""
```

```
par(mfrow = c(1, 1))
par(mfrow = c(1, 2))
boxplot(chol ~ race, data = coronary)
boxplot(chol ~ gender, data = coronary)
```



```r
par(mfrow = c(1, 1))
```

4.2.2.3 Variable selection

4.2.2.3.1 Univariable

Perform SLR for `chol`, `sbp`, `dbp` and `bmi` on your own as shown above. Now, we are concerned with which variables are worthwhile to include in the multivariable models.

We want to choose only variables with P-values < 0.25 to be included in MLR. Obtaining the P-values for each variable is easy by LR test,

```r
slr_chol0 = glm(chol ~ 1, data = coronary)
summary(slr_chol0)
```

```
##
## Call:
## glm(formula = chol ~ 1, data = coronary)
##
## Deviance Residuals:
##      Min        1Q    Median        3Q       Max
## -2.19854  -0.80854  -0.01104   0.69021   3.15146
```

```
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  6.19854    0.08369   74.06   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for gaussian family taken to be 1.400874)
##
##     Null deviance: 278.77  on 199  degrees of freedom
## Residual deviance: 278.77  on 199  degrees of freedom
## AIC: 637.99
##
## Number of Fisher Scoring iterations: 2
```

```
names(coronary)
```

```
## [1] "sbp"    "dbp"    "chol"   "bmi"    "race"   "gender"
```

```
add1(slr_chol0, scope = ~sbp + dbp + bmi + race + gender, test = "LRT")
```

```
## Single term additions
##
## Model:
## chol ~ 1
##        Df Deviance    AIC scaled dev.  Pr(>Chi)
## <none>      278.77 637.99
## sbp     1   235.36 606.14      33.855 5.938e-09 ***
## dbp     1   228.64 600.34      39.648 3.042e-10 ***
## bmi     1   272.17 635.20       4.792   0.02859 *
## race    2   241.68 613.43      28.561 6.280e-07 ***
## gender  1   277.45 639.04       0.952   0.32933
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

All variables are significant and $< .25$ except gender. So proceed with the rest of the variables, excluding gender.

4.2.2.3.2 Multivariable

Perform MLR with *all* selected variables,

```
# all
mlr_chol = glm(chol ~ sbp + dbp + bmi + race, data = coronary)
# mlr_chol = glm(chol ~ ., data = coronary) # shortcut
summary(mlr_chol)
```

```
##
## Call:
## glm(formula = chol ~ sbp + dbp + bmi + race, data = coronary)
##
## Deviance Residuals:
##      Min        1Q    Median        3Q       Max
## -2.17751  -0.73860  -0.02674   0.63163   2.90926
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept)  4.842338   1.265149   3.827 0.000175 ***
## sbp          0.000975   0.006990   0.139 0.889210
## dbp          0.028350   0.010327   2.745 0.006615 **
## bmi         -0.038537   0.028170  -1.368 0.172879
## racechinese  0.354039   0.183169   1.933 0.054710 .
## raceindian   0.716327   0.200346   3.575 0.000441 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for gaussian family taken to be 1.089387)
##
##     Null deviance: 278.77  on 199  degrees of freedom
## Residual deviance: 211.34  on 194  degrees of freedom
## AIC: 592.61
##
## Number of Fisher Scoring iterations: 2
```

```r
rsq(mlr_chol, adj = T)
```

```
## [1] 0.2223518
```

Focus on,

- Coefficients, βs.
- 95% CI.
- P-values.

For model fit,

- R^2 – % of variance explained by the model.
- Akaike Information Criterion, AIC – for comparison with other models. This is not useful alone, but for comparison with other models. The model with the lowest AIC is the best model.

4.2.2.3.3 Stepwise

As you can see, not all variables are significant. How to select? We proceed with stepwise automatic selection,

```r
# stepwise both
mlr_chol_stepboth = step(mlr_chol, direction = "both")
```

```
## Start:  AIC=592.61
## chol ~ sbp + dbp + bmi + race
##
##        Df Deviance    AIC
## - sbp   1   211.36 590.63
## - bmi   1   213.38 592.53
## <none>      211.34 592.61
## - dbp   1   219.55 598.23
## - race  2   225.30 601.40
##
## Step:  AIC=590.63
## chol ~ dbp + bmi + race
##
##        Df Deviance    AIC
## - bmi   1   213.40 590.55
## <none>      211.36 590.63
## + sbp   1   211.34 592.61
## - race  2   227.04 600.94
## - dbp   1   235.88 610.58
##
## Step:  AIC=590.55
## chol ~ dbp + race
##
##        Df Deviance    AIC
## <none>      213.40 590.55
## + bmi   1   211.36 590.63
## + sbp   1   213.38 592.53
## - race  2   228.64 600.34
## - dbp   1   241.68 613.43
```

```
summary(mlr_chol_stepboth)  # racechinese marginally sig.
```

```
##
## Call:
## glm(formula = chol ~ dbp + race, data = coronary)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -2.1378  -0.7068  -0.0289   0.5997   2.7778
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) 3.298028   0.486213   6.783 1.36e-10 ***
## dbp         0.031108   0.006104   5.096 8.14e-07 ***
## racechinese 0.359964   0.182149   1.976 0.049534 *
## raceindian  0.713690   0.190883   3.739 0.000243 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for gaussian family taken to be 1.088777)
##
##     Null deviance: 278.77  on 199  degrees of freedom
## Residual deviance: 213.40  on 196  degrees of freedom
## AIC: 590.55
##
## Number of Fisher Scoring iterations: 2
```

```
# forward
mlr_chol_stepforward = step(slr_chol0, scope = ~sbp + dbp + bmi + race + gender, direction = "forward")
```

```
## Start:  AIC=637.99
## chol ~ 1
##
##          Df Deviance    AIC
## + dbp     1   228.64 600.34
## + sbp     1   235.36 606.14
## + race    2   241.68 613.43
## + bmi     1   272.17 635.20
## <none>        278.77 637.99
## + gender  1   277.45 639.04
##
## Step:  AIC=600.34
## chol ~ dbp
##
##          Df Deviance    AIC
## + race    2   213.40 590.55
## <none>        228.64 600.34
## + gender  1   226.64 600.58
## + sbp     1   226.96 600.87
## + bmi     1   227.04 600.94
##
## Step:  AIC=590.55
## chol ~ dbp + race
##
##          Df Deviance    AIC
## <none>        213.40 590.55
## + bmi     1   211.36 590.63
## + gender  1   212.47 591.67
## + sbp     1   213.38 592.53
```

```
summary(mlr_chol_stepforward)  # same with both
```

```
##
## Call:
## glm(formula = chol ~ dbp + race, data = coronary)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -2.1378  -0.7068  -0.0289   0.5997   2.7778
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) 3.298028   0.486213   6.783 1.36e-10 ***
## dbp         0.031108   0.006104   5.096 8.14e-07 ***
## racechinese 0.359964   0.182149   1.976 0.049534 *
## raceindian  0.713690   0.190883   3.739 0.000243 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for gaussian family taken to be 1.088777)
##
##     Null deviance: 278.77  on 199  degrees of freedom
## Residual deviance: 213.40  on 196  degrees of freedom
## AIC: 590.55
##
## Number of Fisher Scoring iterations: 2
```

```
# backward
mlr_chol_stepback = step(mlr_chol, direction = "backward")
```

```
## Start:  AIC=592.61
## chol ~ sbp + dbp + bmi + race
##
##        Df Deviance    AIC
## - sbp   1   211.36 590.63
## - bmi   1   213.38 592.53
## <none>      211.34 592.61
## - dbp   1   219.55 598.23
## - race  2   225.30 601.40
##
## Step:  AIC=590.63
## chol ~ dbp + bmi + race
##
##        Df Deviance    AIC
## - bmi   1   213.40 590.55
## <none>      211.36 590.63
## - race  2   227.04 600.94
## - dbp   1   235.88 610.58
##
## Step:  AIC=590.55
## chol ~ dbp + race
##
##        Df Deviance    AIC
## <none>      213.40 590.55
## - race  2   228.64 600.34
## - dbp   1   241.68 613.43
```

```
summary(mlr_chol_stepback)  # same with both
```

```
##
## Call:
## glm(formula = chol ~ dbp + race, data = coronary)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -2.1378  -0.7068  -0.0289   0.5997   2.7778
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) 3.298028   0.486213   6.783 1.36e-10 ***
## dbp         0.031108   0.006104   5.096 8.14e-07 ***
## racechinese 0.359964   0.182149   1.976 0.049534 *
## raceindian  0.713690   0.190883   3.739 0.000243 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for gaussian family taken to be 1.088777)
##
##     Null deviance: 278.77  on 199  degrees of freedom
## Residual deviance: 213.40  on 196  degrees of freedom
## AIC: 590.55
##
## Number of Fisher Scoring iterations: 2
```

Looking at all these results, we choose:

```
chol ~ dbp + race
```

which has the lowest AIC.

```
mlr_chol1 = glm(chol ~ dbp + race, data = coronary)
summary(mlr_chol1)
```

```
##
```

```
## Call:
## glm(formula = chol ~ dbp + race, data = coronary)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -2.1378  -0.7068  -0.0289   0.5997   2.7778
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) 3.298028   0.486213   6.783 1.36e-10 ***
## dbp         0.031108   0.006104   5.096 8.14e-07 ***
## racechinese 0.359964   0.182149   1.976 0.049534 *
## raceindian  0.713690   0.190883   3.739 0.000243 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for gaussian family taken to be 1.088777)
##
##     Null deviance: 278.77  on 199  degrees of freedom
## Residual deviance: 213.40  on 196  degrees of freedom
## AIC: 590.55
##
## Number of Fisher Scoring iterations: 2
```

4.2.3 Confounder

If we include a variable and it causes notable change ($> 20\%$) in the coefficients of other variables, it is a confounder. When the confounder is significant and the main effect variable is also significant, we keep the confounder in the model.

Formula for % change,

```
100 * (model_small - model_large) / model_large
```

@hosmer2013

Start by including common demographic adjustment, gender,

```
# + gender
mlr_chol2 = glm(chol ~ dbp + race + gender, data = coronary)
summary(mlr_chol2)  # higher AIC, gender insig.
```

```
##
## Call:
## glm(formula = chol ~ dbp + race + gender, data = coronary)
##
## Deviance Residuals:
##      Min        1Q    Median        3Q       Max
## -2.06350  -0.71634  -0.04471   0.64533   2.70974
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) 3.203032   0.497111   6.443 8.94e-10 ***
## dbp         0.031533   0.006124   5.149 6.37e-07 ***
## racechinese 0.353052   0.182369   1.936   0.0543 .
## raceindian  0.692724   0.192293   3.602   0.0004 ***
## genderman   0.137663   0.148790   0.925   0.3560
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for gaussian family taken to be 1.089578)
##
##     Null deviance: 278.77  on 199  degrees of freedom
## Residual deviance: 212.47  on 195  degrees of freedom
## AIC: 591.67
##
## Number of Fisher Scoring iterations: 2
```

```
coef(mlr_chol2)
```

```
## (Intercept)         dbp racechinese  raceindian   genderman
##   3.2030318   0.0315331   0.3530516   0.6927239   0.1376627
```

```
coef(mlr_chol1)
```

```
## (Intercept)         dbp racechinese  raceindian
##  3.29802826  0.03110811  0.35996365  0.71369024
```

```
100 * (coef(mlr_chol1) - coef(mlr_chol2)[1:4])/coef(mlr_chol2)[1:4]  # change < 20%
```

```
## (Intercept)         dbp racechinese  raceindian
##    2.965828   -1.347773    1.957792    3.026647
```

```
# no notable change in coeffs, gender is not a confounder
```

Now, we can try adding `sbp` & `bmi` to `mlr_chol1` and see what happens to the coefficients. We will use `update()` function here.

```
mlr_chol3 = update(mlr_chol1, . ~ . + sbp)
summary(mlr_chol3)  # higher AIC, sbp insig.
```

```
##
## Call:
## glm(formula = chol ~ dbp + race + sbp, data = coronary)
##
## Deviance Residuals:
##      Min        1Q    Median        3Q       Max
## -2.12850  -0.71572  -0.03242   0.59676   2.77189
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) 3.269724   0.529556   6.174 3.78e-09 ***
## dbp         0.029978   0.010281   2.916 0.003963 **
## racechinese 0.357407   0.183561   1.947 0.052963 .
## raceindian  0.705445   0.200635   3.516 0.000545 ***
## sbp         0.000958   0.007005   0.137 0.891365
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for gaussian family taken to be 1.094256)
##
##     Null deviance: 278.77  on 199  degrees of freedom
## Residual deviance: 213.38  on 195  degrees of freedom
## AIC: 592.53
```

```
##
## Number of Fisher Scoring iterations: 2
```

```
coef(mlr_chol3)
```

```
##  (Intercept)          dbp  racechinese   raceindian          sbp
## 3.2697237312 0.0299783153 0.3574065705 0.7054452332 0.0009580065
```

```
coef(mlr_chol1)
```

```
## (Intercept)         dbp racechinese  raceindian
##  3.29802826  0.03110811  0.35996365  0.71369024
```

```
100 * (coef(mlr_chol1) - coef(mlr_chol3)[1:4])/coef(mlr_chol3)[1:4]  # change < 20%
```

```
## (Intercept)         dbp racechinese  raceindian
##   0.8656550   3.7687027   0.7154536   1.1687670
```

```
# no notable change in coeffs, sbp is not a confounder
```

```
mlr_chol4 = update(mlr_chol1, . ~ . + bmi)
summary(mlr_chol4)  # slighly higher AIC, bmi insig.
```

```
##
## Call:
## glm(formula = chol ~ dbp + race + bmi, data = coronary)
##
## Deviance Residuals:
##      Min        1Q    Median        3Q       Max
## -2.18698  -0.73076  -0.01935   0.63476   2.91524
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept)  4.870859   1.245373   3.911 0.000127 ***
## dbp          0.029500   0.006203   4.756 3.83e-06 ***
## racechinese  0.356642   0.181757   1.962 0.051164 .
## raceindian   0.724716   0.190625   3.802 0.000192 ***
## bmi         -0.038530   0.028099  -1.371 0.171871
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for gaussian family taken to be 1.083909)
##
##     Null deviance: 278.77  on 199  degrees of freedom
## Residual deviance: 211.36  on 195  degrees of freedom
## AIC: 590.63
##
## Number of Fisher Scoring iterations: 2
```

```
coef(mlr_chol4)
```

```
## (Intercept)         dbp racechinese  raceindian         bmi
##  4.87085865  0.02950027  0.35664168  0.72471631 -0.03853042
```

```
coef(mlr_chol1)
```

```
## (Intercept)         dbp racechinese  raceindian
##  3.29802826  0.03110811  0.35996365  0.71369024
```

```
100 * (coef(mlr_chol1) - coef(mlr_chol4)[1:4])/coef(mlr_chol4)[1:4]  # change < 20%
```

```
## (Intercept)         dbp racechinese  raceindian 
##  -32.290619    5.450250    0.931459   -1.521432
```

```
# no notable change in coeffs of other vars (ignore intercept!)  bmi is not a confounder
```

Our chosen model:

```
mlr_chol1: chol ~ dbp + race
```

```
summary(mlr_chol1)
```

```
## 
## Call:
## glm(formula = chol ~ dbp + race, data = coronary)
## 
## Deviance Residuals: 
##     Min       1Q   Median       3Q      Max  
## -2.1378  -0.7068  -0.0289   0.5997   2.7778  
## 
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)    
## (Intercept) 3.298028   0.486213   6.783 1.36e-10 ***
## dbp         0.031108   0.006104   5.096 8.14e-07 ***
## racechinese 0.359964   0.182149   1.976 0.049534 *  
## raceindian  0.713690   0.190883   3.739 0.000243 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
## 
## (Dispersion parameter for gaussian family taken to be 1.088777)
## 
##     Null deviance: 278.77  on 199  degrees of freedom
## Residual deviance: 213.40  on 196  degrees of freedom
## AIC: 590.55
## 
## Number of Fisher Scoring iterations: 2
```

```
Confint(mlr_chol1)  # 95% CI of the coefficients
```

```
##               Estimate       2.5 %     97.5 %
## (Intercept) 3.29802826 2.345067995 4.25098852
## dbp         0.03110811 0.019143668 0.04307255
## racechinese 0.35996365 0.002958566 0.71696873
## raceindian  0.71369024 0.339566932 1.08781356
```

Compare this model with the no-variable model and all-variable model by LR test and AIC comparison,

```
# LR test
anova(slr_chol0, mlr_chol1, test = "LRT")  # sig. better than no var at all!
```

```
## Analysis of Deviance Table
## 
## Model 1: chol ~ 1
## Model 2: chol ~ dbp + race
##   Resid. Df Resid. Dev Df Deviance  Pr(>Chi)    
## 1       199     278.77                          
## 2       196     213.40  3   65.373 5.755e-13 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```
# model with no var at all is called Null Model
anova(mlr_chol, mlr_chol1, test = "LRT")  # no sig. dif with all vars model,
```

```
## Analysis of Deviance Table
##
## Model 1: chol ~ sbp + dbp + bmi + race
## Model 2: chol ~ dbp + race
##   Resid. Df Resid. Dev Df Deviance Pr(>Chi)
## 1       194     211.34
## 2       196     213.40 -2  -2.0593   0.3886
```

```
# model with 2 vars (dbp & race) is just as good as full model (with all the vars) model
# with all vars is called Saturated Model
```

```
# AIC
AIC(slr_chol0, mlr_chol1, mlr_chol)
```

```
##           df      AIC
## slr_chol0  2 637.9921
## mlr_chol1  5 590.5459
## mlr_chol   7 592.6065
```

```
# our final model has the lowest AIC
```

4.2.4 Multicollinearity, MC

Multicollinearity is the problem of repetitive/redundant variables – high correlations between predictors. MC is checked by Variance Inflation Factor (VIF). VIF > 10 indicates MC problem.

```
vif(mlr_chol1)  # all < 10
```

```
##          GVIF Df GVIF^(1/(2*Df))
## dbp  1.132753  1        1.064309
## race 1.132753  2        1.031653
```

4.2.5 Interaction, *

Interaction is the predictor variable combination that requires interpretation of regression coefficients separately based on the levels of the predictor (e.g. separate analysis for each race group, Malay vs Chinese vs Indian). This makes interpreting our analysis complicated. So, most of the time, we pray not to have interaction in our regression model.

```
summary(glm(chol ~ dbp * race, data = coronary))  # dbp*race not sig.
```

```
##
## Call:
## glm(formula = chol ~ dbp * race, data = coronary)
##
## Deviance Residuals:
##      Min        1Q    Median        3Q       Max
## -2.10485  -0.77524  -0.02423   0.58059   2.74380
##
## Coefficients:
```

```
##                   Estimate Std. Error t value Pr(>|t|)    
## (Intercept)        2.11114    0.92803   2.275 0.024008 *  
## dbp                0.04650    0.01193   3.897 0.000134 ***
## racechinese        1.95576    1.28477   1.522 0.129572    
## raceindian         2.41530    1.25766   1.920 0.056266 .  
## dbp:racechinese   -0.02033    0.01596  -1.273 0.204376    
## dbp:raceindian    -0.02126    0.01529  -1.391 0.165905    
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for gaussian family taken to be 1.087348)
##
##     Null deviance: 278.77  on 199  degrees of freedom
## Residual deviance: 210.95  on 194  degrees of freedom
## AIC: 592.23
##
## Number of Fisher Scoring iterations: 2
```

```
# in R, it is easy to fit interaction by * dbp*race will automatically include all vars
# involved i.e. equal to glm(chol ~ dbp + race + dbp:race, data = coronary) use : to just
# include just the interaction
```

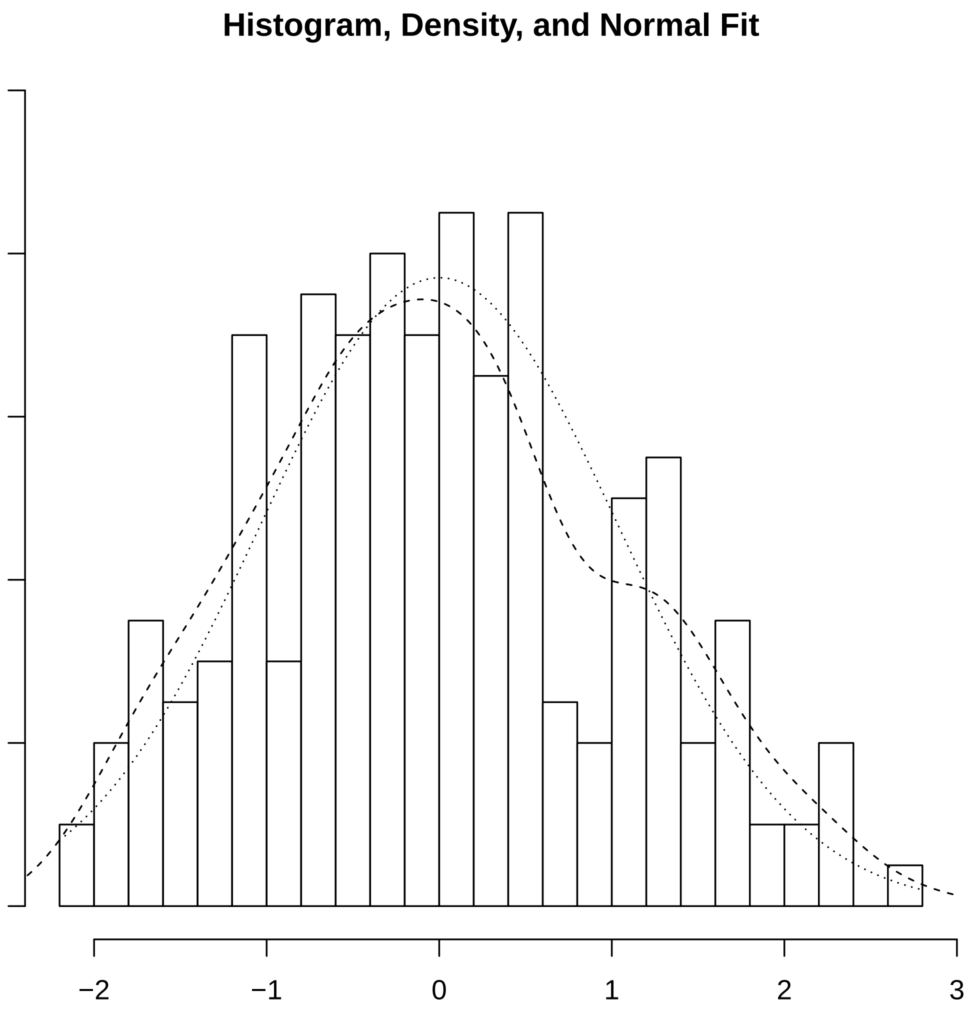
There is no interaction here because the included interaction term was insignificant.

4.3 Model fit assessment: Residuals

Histogram

Raw residuals: Normality assumption.

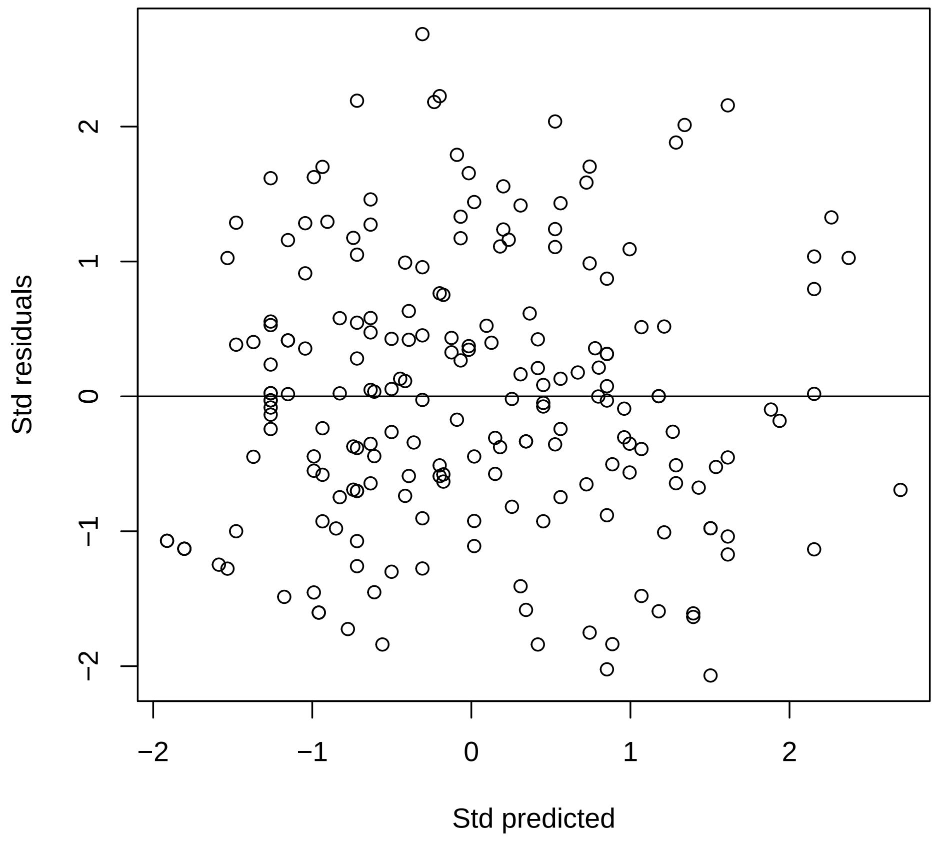
```
rraw_chol = resid(mlr_chol1)  # unstandardized
multi.hist(rraw_chol)
```



Scatter plots

Standardized residuals vs Standardized predicted values: Overall – normality, linearity and equal variance assumptions.

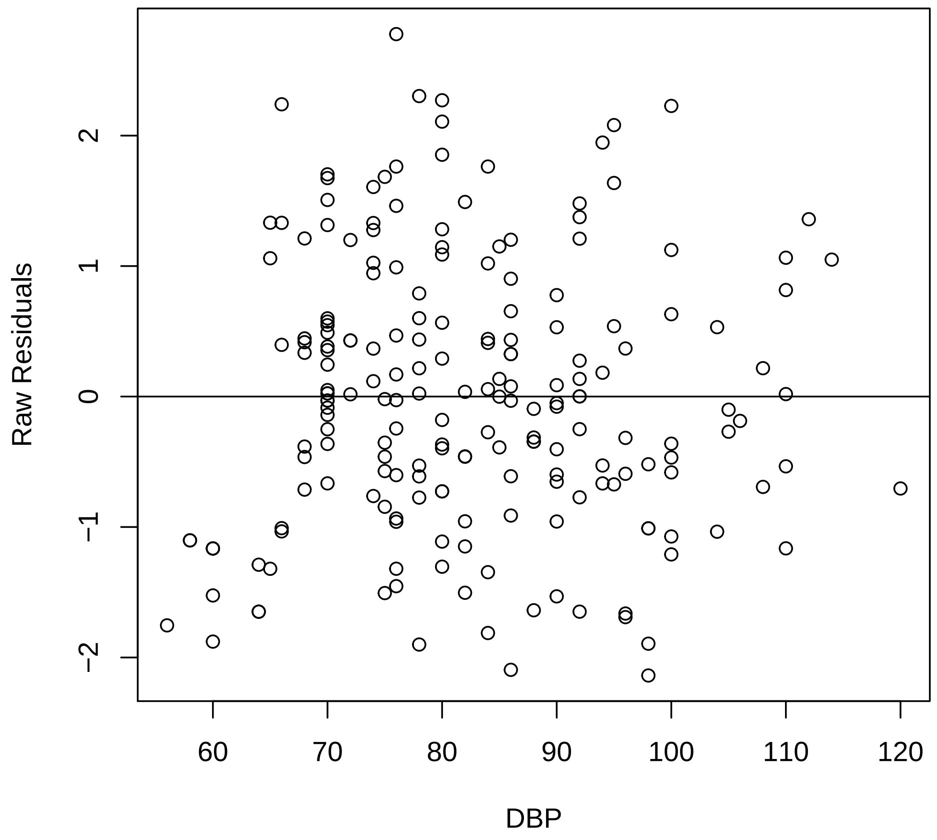
```
rstd_chol = rstandard(mlr_chol1)  # standardized residuals
pstd_chol = scale(predict(mlr_chol1))  # standardized predicted values
plot(rstd_chol ~ pstd_chol, xlab = "Std predicted", ylab = "Std residuals")
abline(0, 0)  # normal, linear, equal variance
```



The dots should form elliptical/oval shape (normality) and scattered roughly equal above and below the zero line (equal variance). Both these indicate linearity.

Raw residuals vs Numerical predictor by each predictors: Linearity assumption.

```r
plot(rraw_chol ~ coronary$dbp, xlab = "DBP", ylab = "Raw Residuals")
abline(0, 0)
```



4.4 Interpretation

Now we have decided on our final model, rename the model,

```
# rename the selected model
mlr_chol_final = mlr_chol1
```

and interpret the model,

```
summary(mlr_chol_final)
```

```
##
## Call:
## glm(formula = chol ~ dbp + race, data = coronary)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -2.1378  -0.7068  -0.0289   0.5997   2.7778
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) 3.298028   0.486213   6.783 1.36e-10 ***
## dbp         0.031108   0.006104   5.096 8.14e-07 ***
## racechinese 0.359964   0.182149   1.976 0.049534 *
## raceindian  0.713690   0.190883   3.739 0.000243 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for gaussian family taken to be 1.088777)
##
##     Null deviance: 278.77  on 199  degrees of freedom
## Residual deviance: 213.40  on 196  degrees of freedom
## AIC: 590.55
##
## Number of Fisher Scoring iterations: 2
```

```
Confint(mlr_chol_final)  # 95% CI of the coefficients
```

```
##               Estimate       2.5 %     97.5 %
## (Intercept) 3.29802826 2.345067995 4.25098852
## dbp         0.03110811 0.019143668 0.04307255
## racechinese 0.35996365 0.002958566 0.71696873
## raceindian  0.71369024 0.339566932 1.08781356
```

```
rsq(mlr_chol_final, adj = T)
```

```
## [1] 0.2227869
```

- 1mmHg increase in DBP causes 0.03mmol/L increase in cholestrol, controlling for the effect of race.
- Being Chinese causes 0.36mmol/L increase in cholestrol in comparison to Malay, controlling for the effect of DBP.
- Being Indian causes 0.71mmol/L increase in cholestrol in comparison to Malay, controlling for the effect of DBP.
- DBP and race explains 22.3% variance in cholestrol.

Turn the results into data frames results using `broom`,

```
tib_mlr = tidy(mlr_chol_final, conf.int = T)
tib_mlr
```

```
## # A tibble: 4 x 7
##   term        estimate std.error statistic  p.value conf.low conf.high
##   <chr>          <dbl>     <dbl>     <dbl>    <dbl>    <dbl>     <dbl>
## 1 (Intercept)   3.30     0.486        6.78 1.36e-10  2.35       4.25
## 2 dbp           0.0311   0.00610      5.10 8.14e- 7  0.0191     0.0431
## 3 racechinese   0.360    0.182        1.98 4.95e- 2  0.00296    0.717
## 4 raceindian    0.714    0.191        3.74 2.43e- 4  0.340      1.09
```

Display the results using `kable` in a nice table,

```
knitr::kable(tib_mlr)
```

term	estimate	std.error	statistic	p.value	conf.low	conf.high
(Intercept)	3.2980283	0.4862132	6.783091	0.0000000	2.3450680	4.2509885
dbp	0.0311081	0.0061044	5.095998	0.0000008	0.0191437	0.0430726
racechinese	0.3599636	0.1821488	1.976207	0.0495342	0.0029586	0.7169687
raceindian	0.7136902	0.1908827	3.738893	0.0002425	0.3395669	1.0878136

We can export the results into a .csv file for use later (e.g. to prepare a table for journal articles etc.),

```r
write.csv(tib_mlr, "mlr_final.csv")
```

4.5 Model equation

Cholestrol level in mmol/L can be predicted by its predictors as given by,

$$chol = 3.30 + 0.03 \times dbp + 0.36 \times race\ (chinese) + 0.71 \times race\ (indian)$$

4.6 Prediction

It is easy to predict in R using our fitted model above. First we view the predicted values for our sample,

```r
coronary$pred_chol = predict(mlr_chol_final)
head(coronary)
```

```
##   sbp dbp   chol  bmi   race gender pred_chol
## 1 106  68 6.5725 38.9 indian  woman  6.127070
## 2 130  78 6.3250 37.8  malay  woman  5.724461
## 3 136  84 5.9675 40.5  malay  woman  5.911109
## 4 138 100 7.0400 37.6  malay  woman  6.408839
## 5 115  85 6.6550 40.3 indian    man  6.655908
## 6 124  72 5.9675 37.6  malay    man  5.537812
```

Now let us try predicting for any values for `dbp` and `race`,

```r
str(coronary[c("dbp", "race")])
```

```
## 'data.frame':    200 obs. of  2 variables:
##  $ dbp : num  68 78 84 100 85 72 80 70 85 70 ...
##  $ race: Factor w/ 3 levels "malay","chinese",..: 3 1 1 1 3 1 1 2 2 2 ...
```

```r
# simple, dbp = 90, race = indian
predict(mlr_chol_final, list(dbp = 90, race = "indian"))
```

```
##        1
## 6.811448
```

More data points

```r
new_data = data.frame(dbp = c(90, 90, 90), race = c("malay", "chinese", "indian"))
new_data
```

```
##   dbp    race
## 1  90   malay
## 2  90 chinese
## 3  90  indian
```

```r
predict(mlr_chol_final, new_data)
```

```
##        1        2        3
## 6.097758 6.457722 6.811448
```

```r
new_data$pred_chol = predict(mlr_chol_final, new_data)
new_data
```

```
##   dbp    race pred_chol
## 1  90   malay  6.097758
## 2  90 chinese  6.457722
## 3  90  indian  6.811448
```