Data visualization
Using **ggplot2** to work with plots

Kamarul Imran Musa
Associate Professor (Epidemiology and Statistics)

2019-02-17

Contents

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 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."

1.2 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

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.

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.

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.

5 Preparation

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`)

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?

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

6 Hands-on 1 (5 min)

6.1 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

```
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)`.

6.2 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

```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
##
```

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

```r
?gapminder
```

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

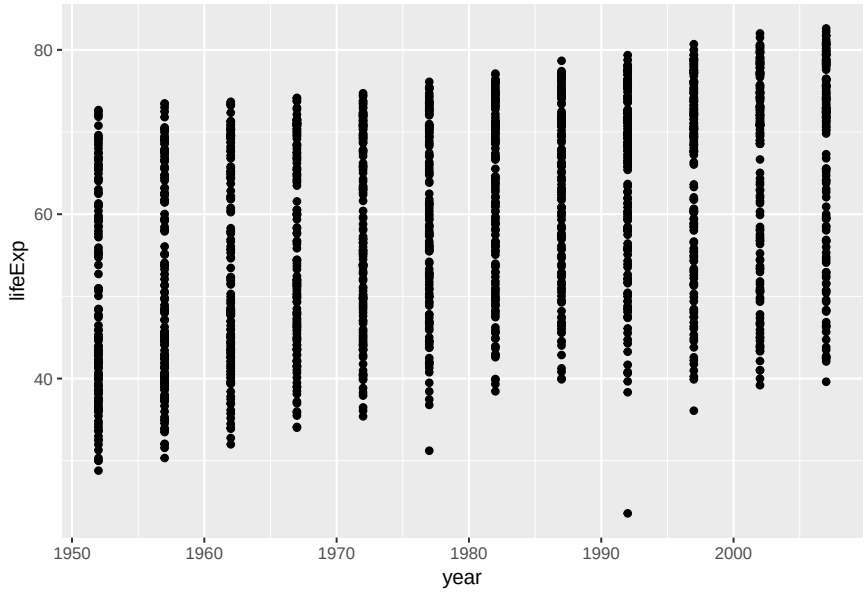
7 Hands-on 2 (5 min)

7.1 Basic plot

We can start create a basic plot

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

```r
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.

7.2 Adding another variable

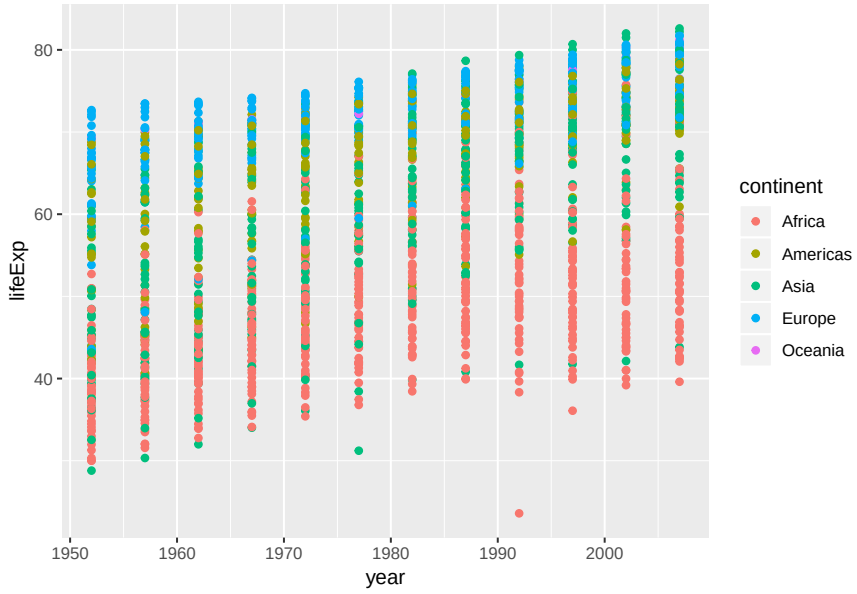
You realize that we plotted 2 variables based on `aes()`. We can add the third variable to make a more complicated plot.

For example:

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

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

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

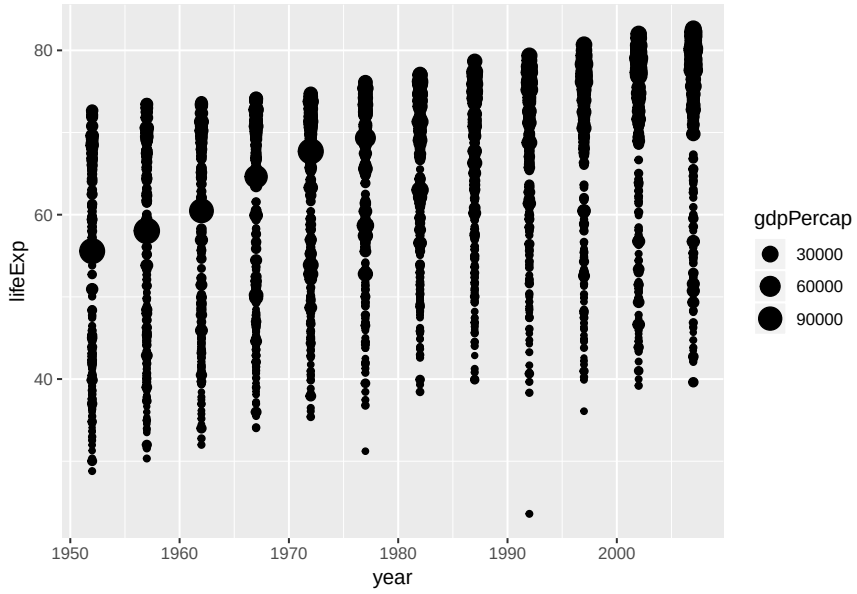


What can you see from the scatterplot.

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.

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



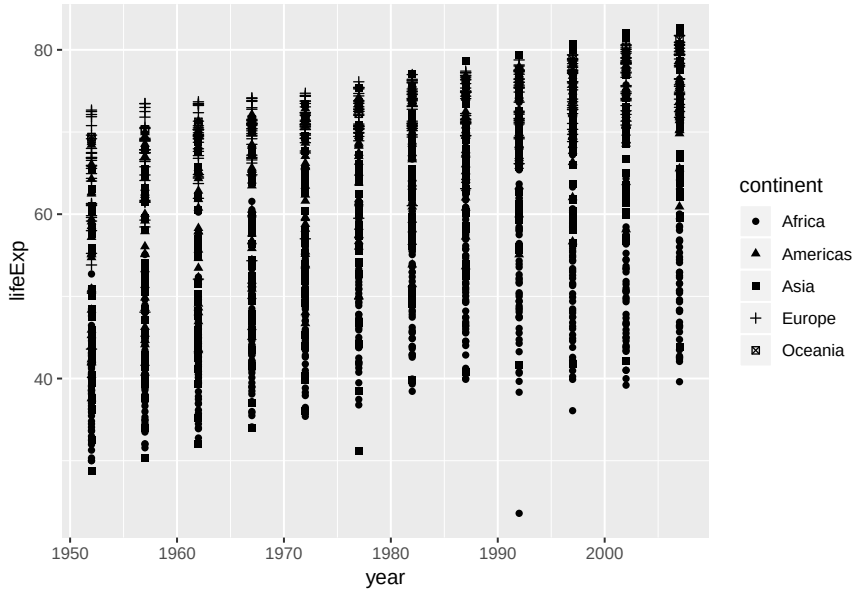
ggplot2 will automatically assign a unique level of the aesthetic (here a unique color) to each unique value of the variable, a process known as scaling.

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

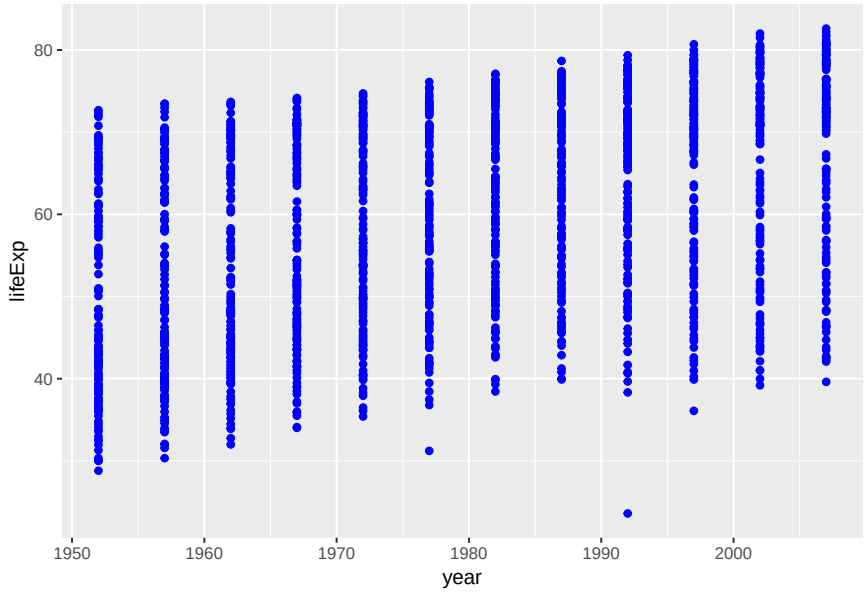
```
ggplot(data = gapminder) +
  geom_point(mapping = aes(x = year, y = lifeExp, shape = continent))
```



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

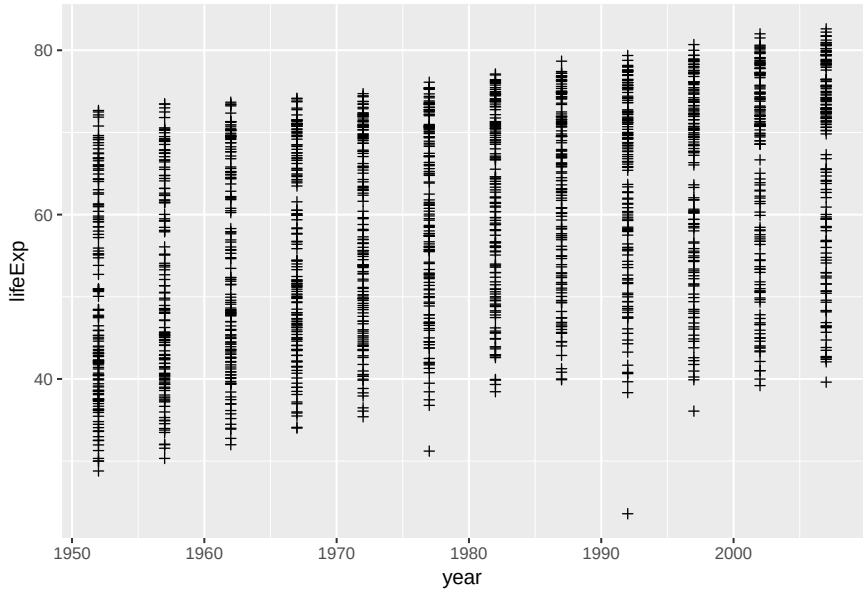
colour as blue

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



shape as plus

```
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

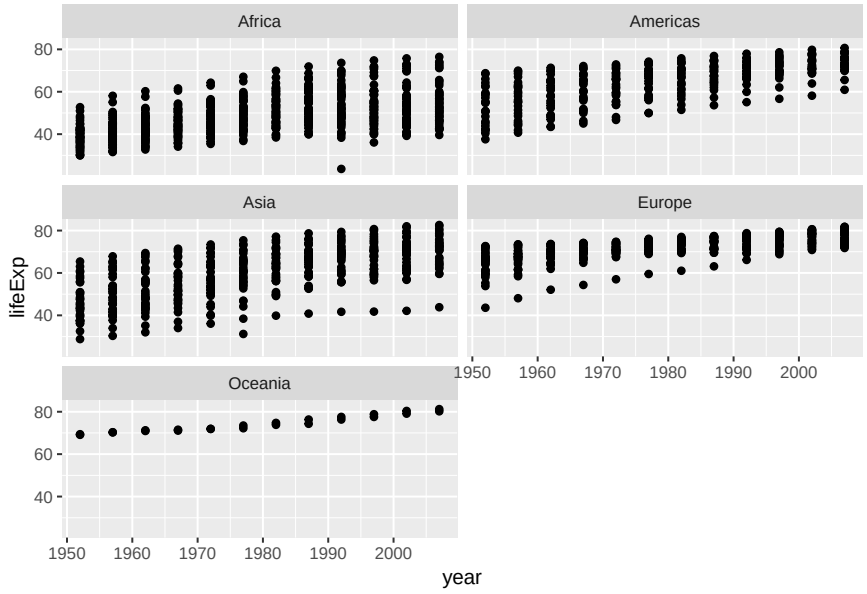
8 Hands-on 3 (5 min)

8.1 Making subplots

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

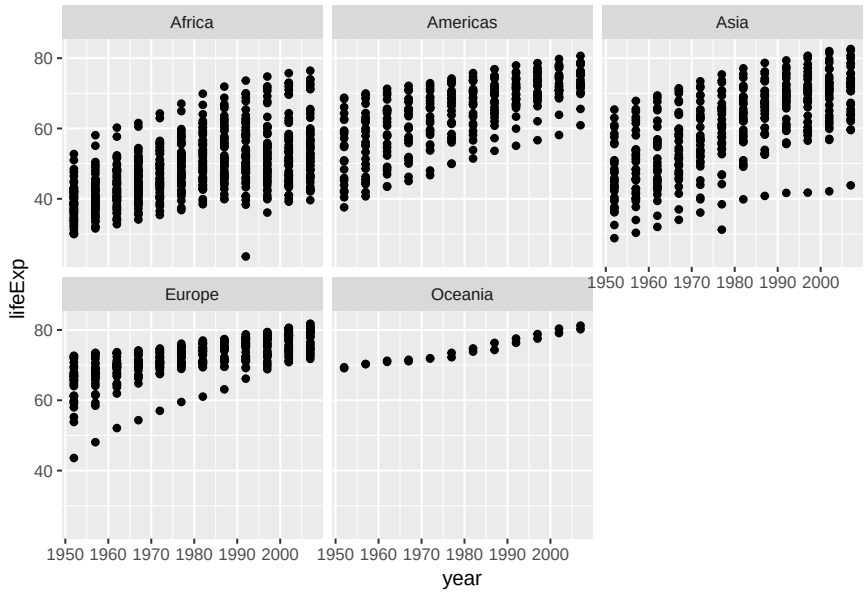
For example, if we want to make subplots based on continents, you can run these codes

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



and change the nrow

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



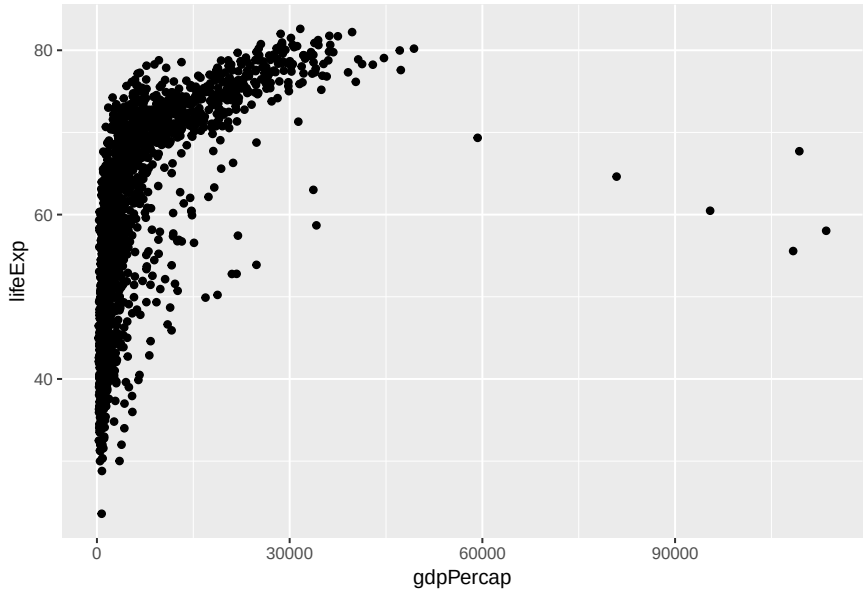
9 Hands-on 4 (10 min)

9.1 Overlaying plots

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

Scatterplot

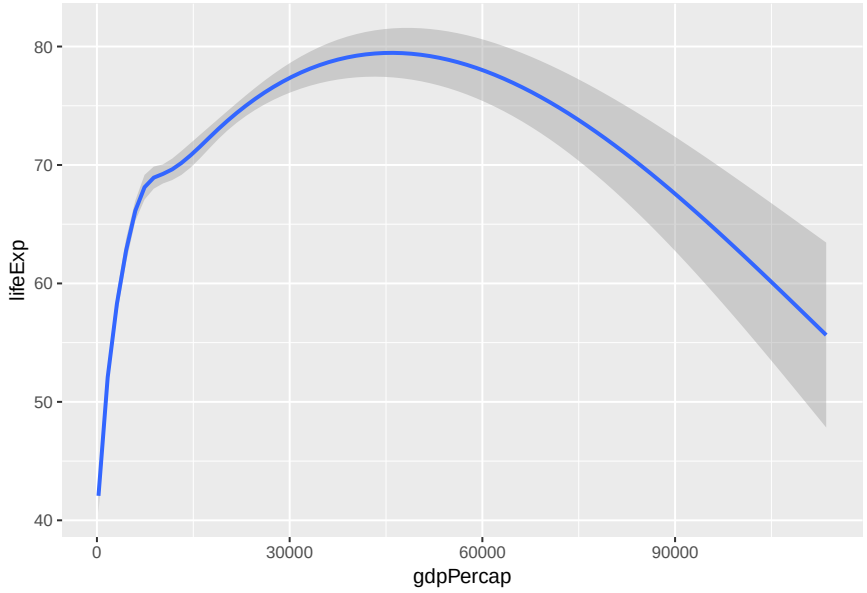
```
ggplot(data = gapminder) +
  geom_point(mapping = aes(x = gdpPercap, y = lifeExp))
```



Smooth line

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

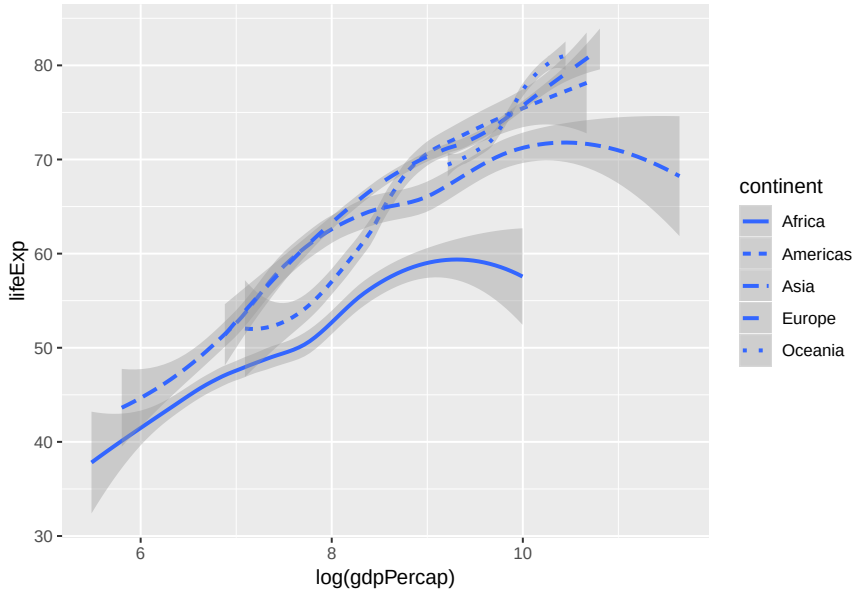
```
## `geom_smooth()` using method = 'gam' and formula 'y ~ s(x, bs = "cs")'
```



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

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

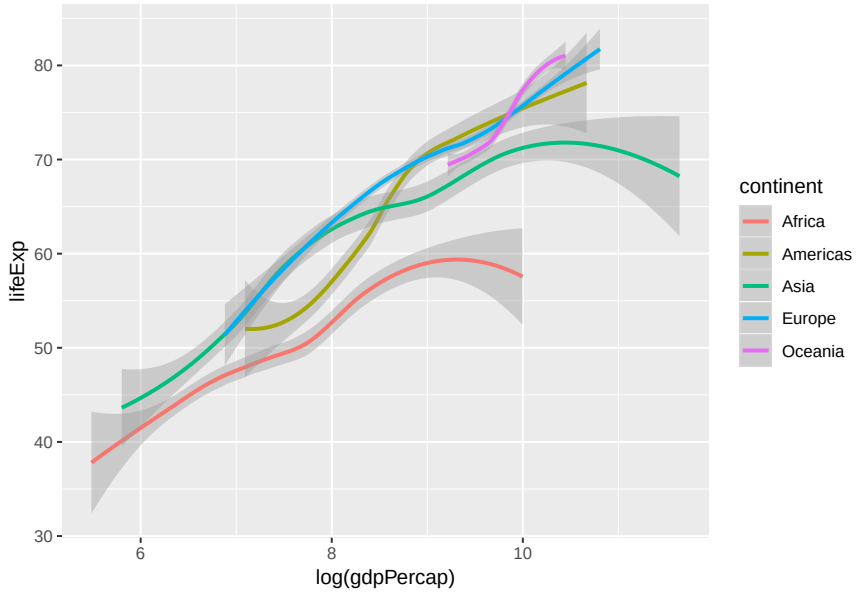
```
## `geom_smooth()` using method = 'loess' and formula 'y ~ x'
```



Another plot but using colour

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

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

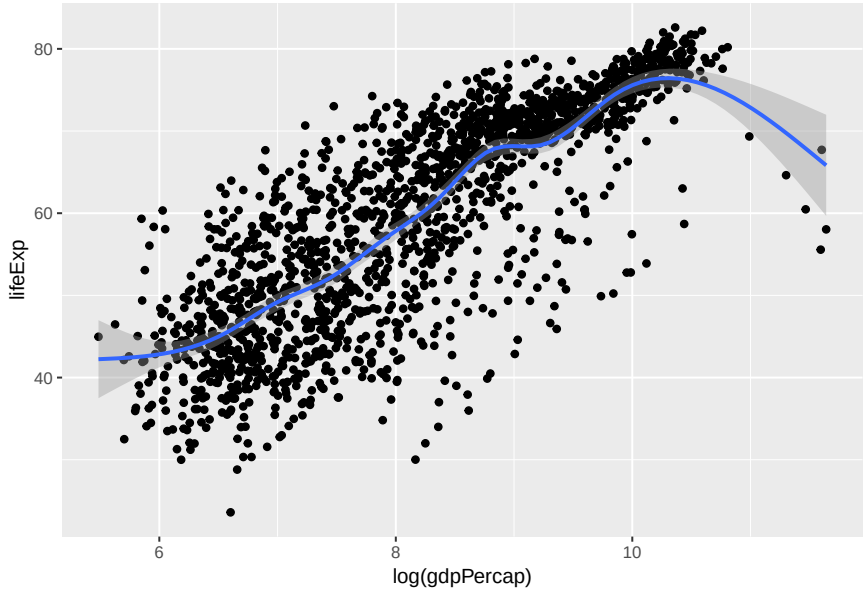


9.2 Combining geom

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

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

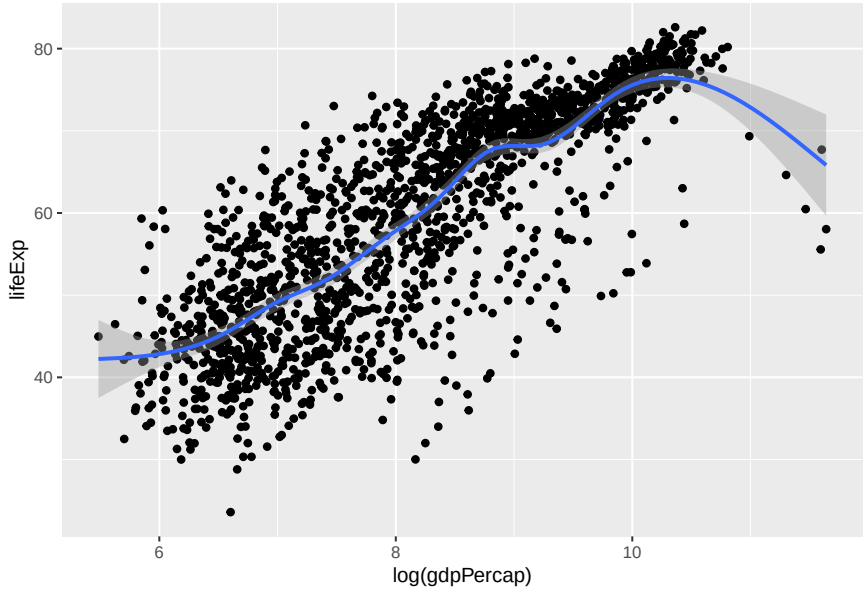
```
## `geom_smooth()` using method = 'gam' and formula 'y ~ s(x, bs = "cs")'
```



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

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

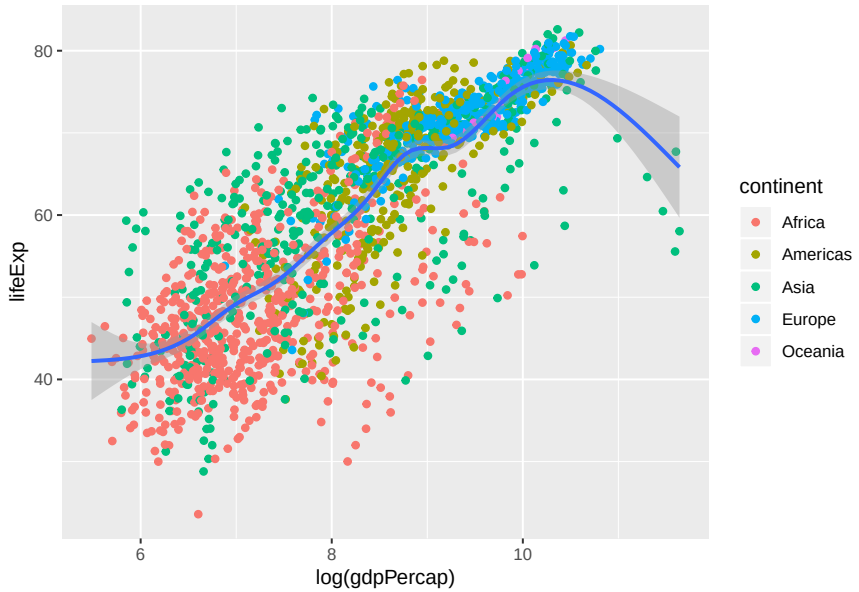
```
## `geom_smooth()` using method = 'gam' and formula 'y ~ s(x, bs = "cs")'
```



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

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

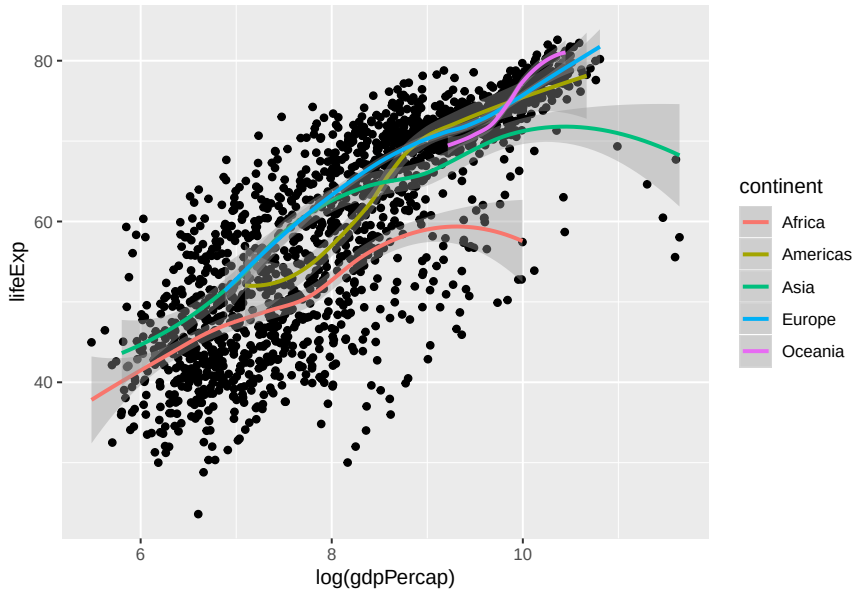
```
## `geom_smooth()` using method = 'gam' and formula 'y ~ s(x, bs = "cs")'
```



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

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

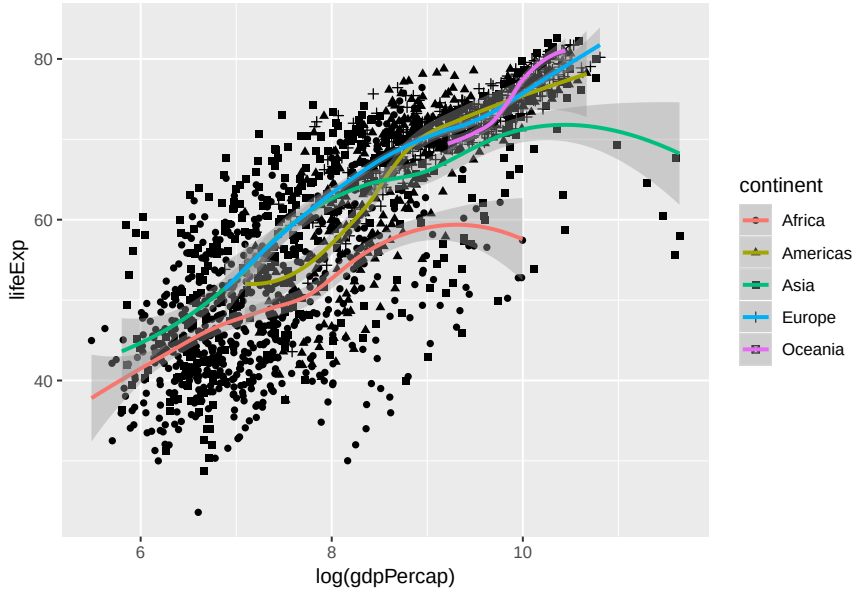
```
## `geom_smooth()` using method = 'loess' and formula 'y ~ x'
```



Or both the scatterplot and the smoothplot

```
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'
```

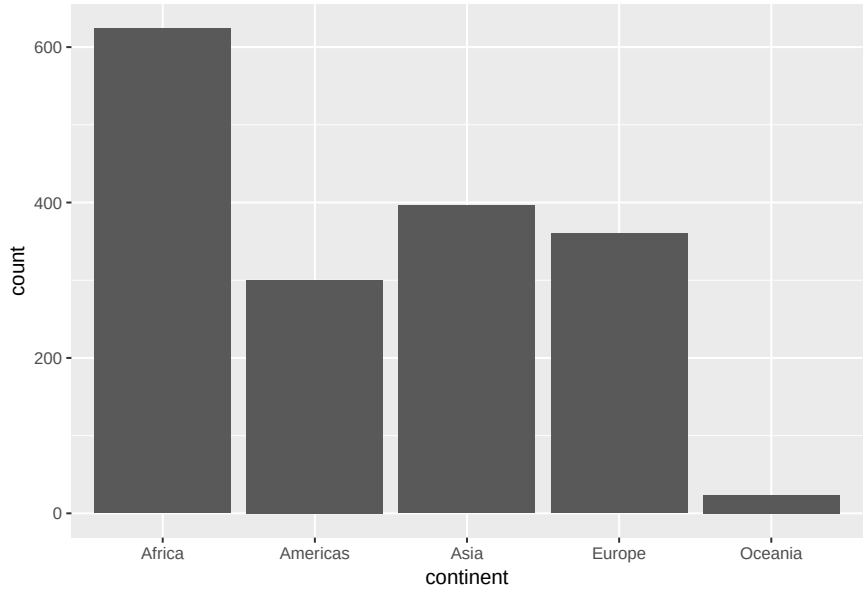


10 Hands-on 5 (5 min)

10.1 Statistical transformation

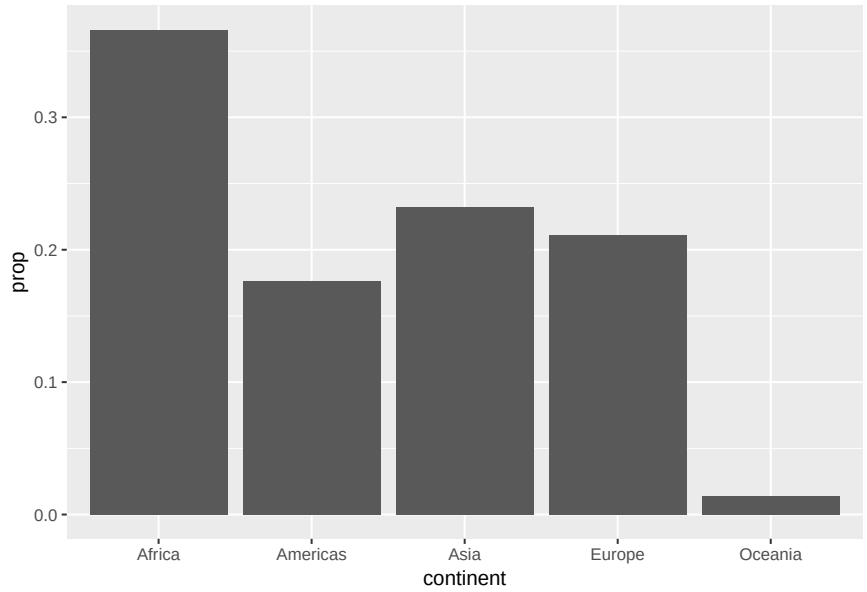
Let us create a bar chart, with y axis as the frequency.

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



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

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



11 Hands-on 6 (10 min)

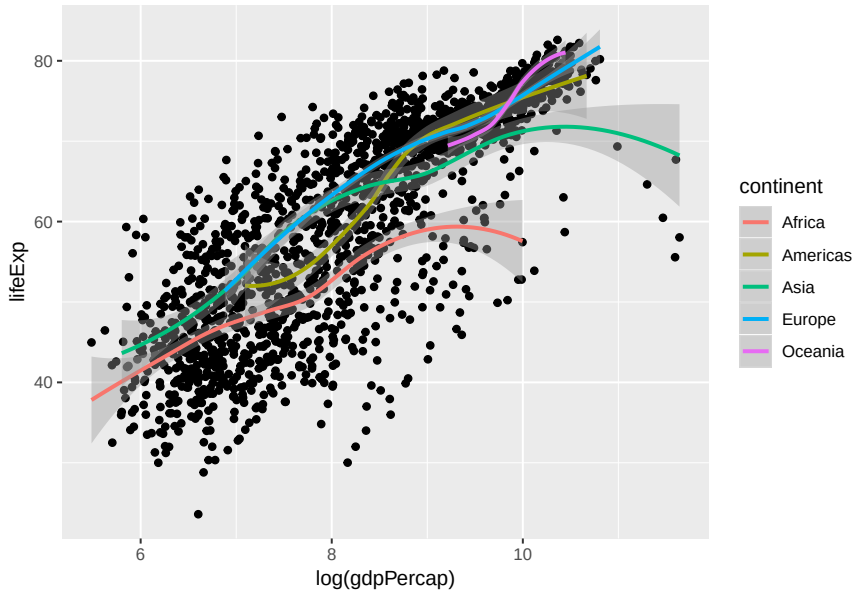
11.1 Customizing title

We can customize many aspects of the plot using ggplot package.

For example, from gapminder dataset, we choose GDP and log it (to reduce skewness) and life expectancy, and make a scatterplot. We named the plot as `my_pop`

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

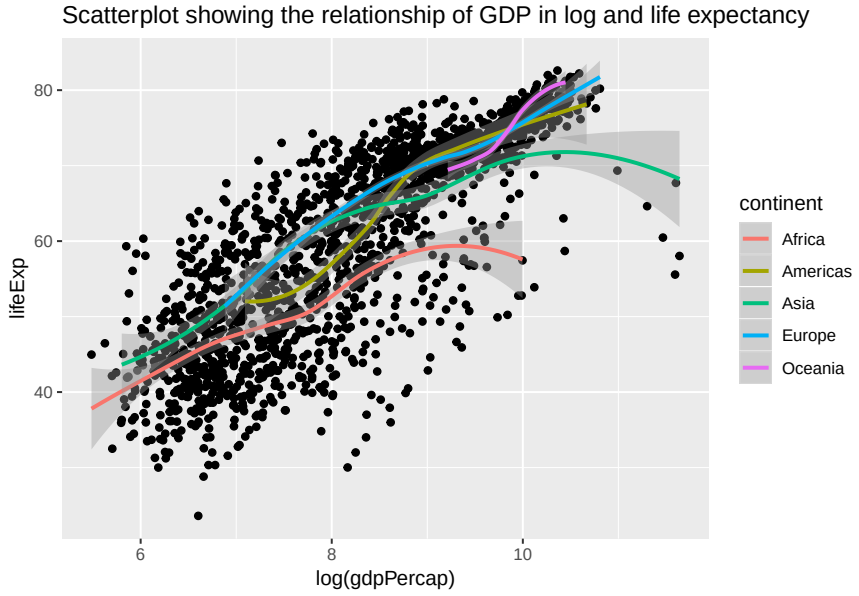
```
## `geom_smooth()` using method = 'loess' and formula 'y ~ x'
```



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")
```

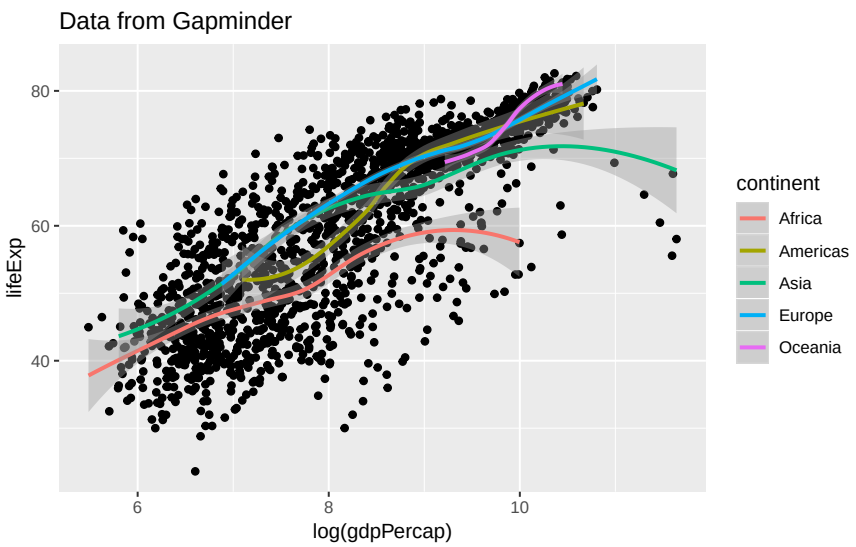
```
## `geom_smooth()` using method = 'loess' and formula 'y ~ x'
```



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'
```



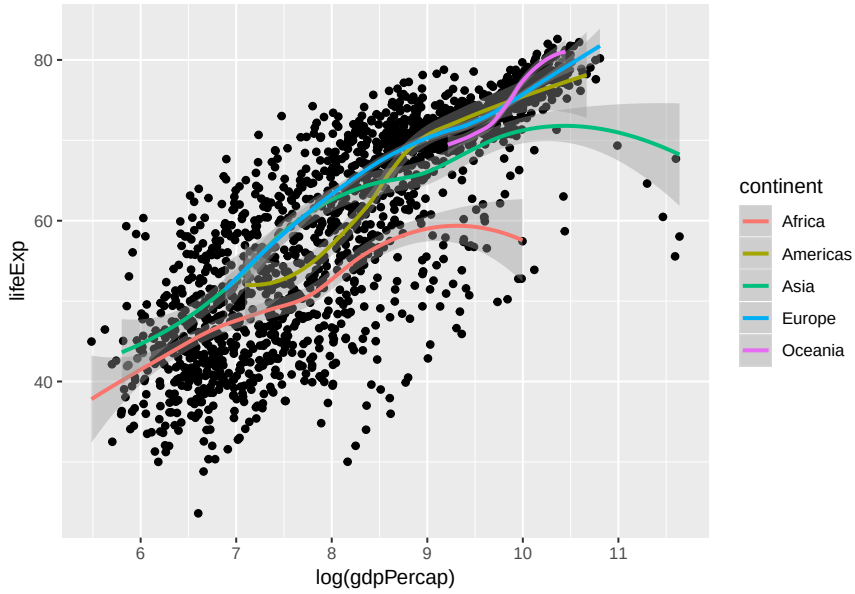
11.2 Adjusting axes

We can specify the tick marks

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

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

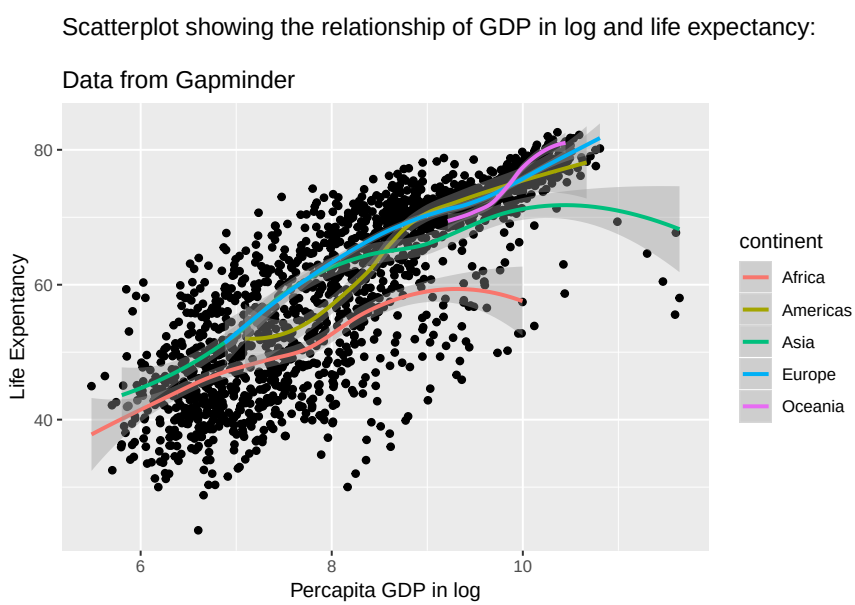
```
## `geom_smooth()` using method = 'loess' and formula 'y ~ x'
```



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

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

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



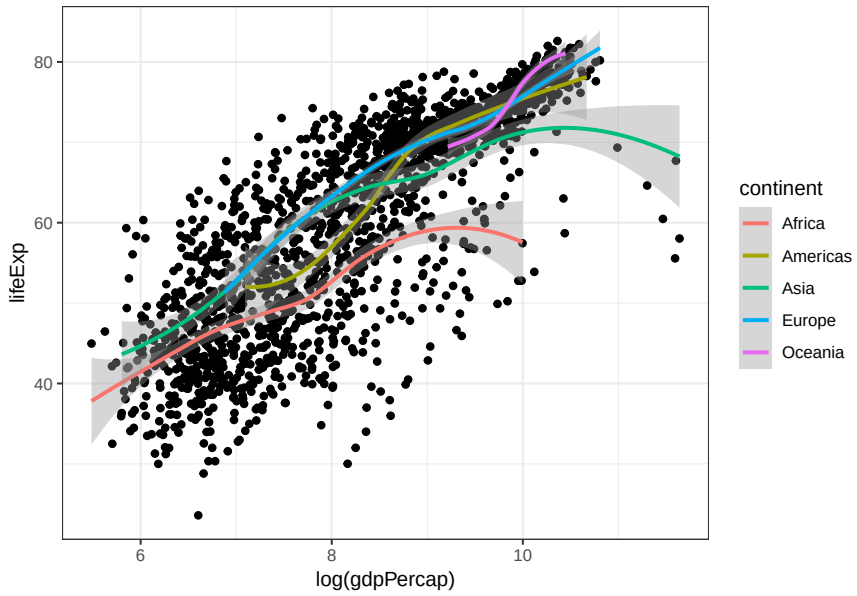
11.3 Choosing theme

The default is gray theme or `theme_gray()`

The black and white theme

```
mypop + theme_bw()
```

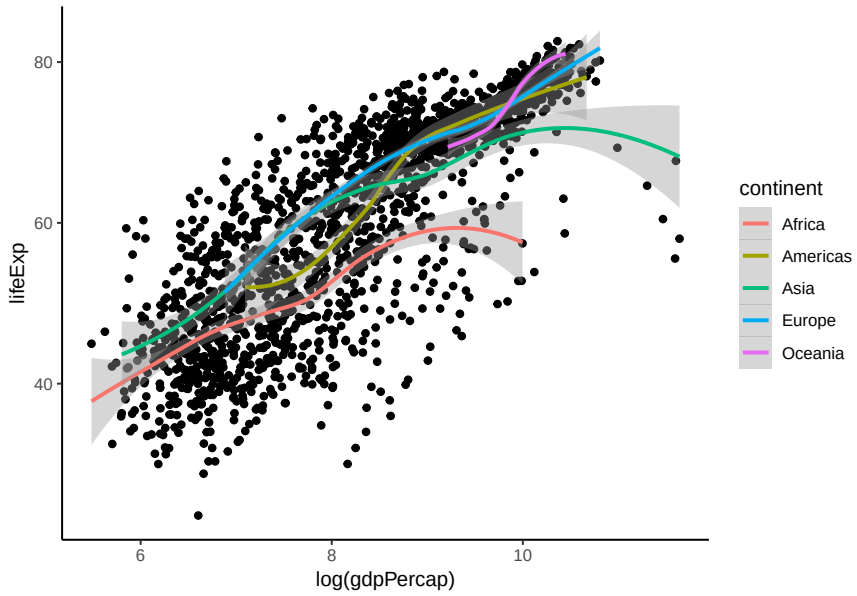
```
## `geom_smooth()` using method = 'loess' and formula 'y ~ x'
```



Classic theme

```
mypop + theme_classic()
```

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



12 Hands-on 7 (5 min)

12.1 Saving plot

The preferred format for saving file is PDF.

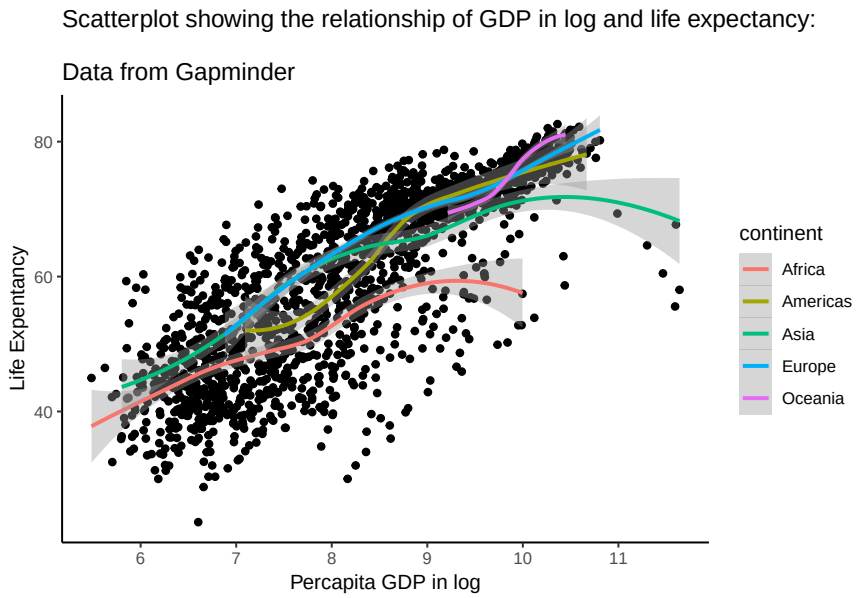
12.2 Saving plot using ggplot2

We can save the last plot (plot on the screen) to a file.

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

```
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()
```

```
## `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

```
ggsave("my_pdf_plot.pdf")
```

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

```
ggsave("my_png_plot.png")
```

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

```
ggsave("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
2. height = 6 cm
3. dpi = 100

```
ggsave("my_pdf_plot2.pdf", width = 10, height = 6, units = "cm")
```

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

```
ggsave("my_png_plot2.png", width = 10, height = 6, units = "cm")
```

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

```
ggsave("my_jpg_plot2.jpg", width = 10, height = 6, units = "cm")
```

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

13 Self-practice

14 Environment

```
sessionInfo()
```

```
## R version 3.5.2 (2018-12-20)
## Platform: x86_64-w64-mingw32/x64 (64-bit)
## Running under: Windows 10 x64 (build 17763)
##
## Matrix products: default
##
## locale:
## [1] LC_COLLATE=English_United States.1252
## [2] LC_CTYPE=English_United States.1252
## [3] LC_MONETARY=English_United States.1252
## [4] LC_NUMERIC=C
## [5] LC_TIME=English_United States.1252
##
## attached base packages:
## [1] stats     graphics  grDevices utils     datasets  methods   base
##
## other attached packages:
##  [1] gapminder_0.3.0 forcats_0.3.0   stringr_1.3.1   dplyr_0.7.8
##  [5] purrr_0.2.5     readr_1.3.1     tidyr_0.8.2     tibble_1.4.2
##  [9] ggplot2_3.1.0   tidyverse_1.2.1
##
## loaded via a namespace (and not attached):
##  [1] tidyselect_0.2.5 xfun_0.4         haven_2.0.0      lattice_0.20-38
##  [5] colorspace_1.3-2 generics_0.0.2   htmltools_0.3.6  mgcv_1.8-26
##  [9] yaml_2.2.0       utf8_1.1.4       rlang_0.3.0.1    pillar_1.3.1
## [13] glue_1.3.0       withr_2.1.2      modelr_0.1.2     readxl_1.2.0
## [17] bindrcpp_0.2.2   bindr_0.1.1      plyr_1.8.4       munsell_0.5.0
## [21] gtable_0.2.0     cellranger_1.1.0 rvest_0.3.2      evaluate_0.12
## [25] labeling_0.3     knitr_1.21       fansi_0.4.0      broom_0.5.1
## [29] Rcpp_1.0.0       scales_1.0.0     backports_1.1.3  jsonlite_1.6
## [33] hms_0.4.2        digest_0.6.18    stringi_1.2.4    grid_3.5.2
## [37] cli_1.0.1        tools_3.5.2      magrittr_1.5     lazyeval_0.2.1
## [41] crayon_1.3.4     pkgconfig_2.0.2  Matrix_1.2-15    xml2_1.2.0
## [45] lubridate_1.7.4  assertthat_0.2.0 rmarkdown_1.11   httr_1.4.0
## [49] rstudioapi_0.8   R6_2.3.0         nlme_3.1-137     compiler_3.5.2
```