Advanced R

Kálmán Abari

2021-10-22

Contents

Chapter 1

Introduction

Welcome to the second book in R Fundamentals series! This second book takes you through how to do manipulation of tabular data and how to create modern graphics in R. We'll primarily be using capabilities from the set of packages called the tidyverse within the book. The book is aimed at beginners to R who understand the basics (check out the Basic R).

Chapter 2

Warm-up exercise

Loftus, S. C. (2021). Basic Statistics with R: Reaching Decisions with Data. Retrieved from https://books.google.hu/books?id=vTASEAAAQBAJ

2.1 Data structures

2.1.1 Problems

Consider the following set of attributes about the American Film Institute's top-five movies ever from their 2007 list.

1. What code would you use to create a vector named `Movie` with the values `Citizen Kane`, `The Godfather`, `Casablanca`, `Raging Bull`, and `Singing in the Rain`? (Hints: `object <- c()`, Working with character in R)
2. What code would you use to create a vector — giving the year that the movies in Problem 1 were made — named `Year` with the values 1941, 1972, 1942, 1980, and 1952?
3. What code would you use to create a vector — giving the run times in minutes of the movies in Problem 1 — named `RunTime` with the values 119, 177, 102, 129, and 103?
4. What code would you use to find the run times of the movies in hours and save them in a vector called `RunTimeHours`? (Hints: Numeric tranformation)
5. What code would you use to create a data frame named `MovieInfo` containing the vectors created in Problem 1, Problem 2, and Problem 3? (Hints: `data.frame()`)

2.2 Manipulation

2.2.1 Problems

Suppose we have the following data frame named `colleges` (download here):

College	Employees	TopSalary	MedianSalary
William and Mary	2104	425000	56496
Christopher Newport	922	381486	47895
George Mason	4043	536714	63029
James Madison	2833	428400	53080
Longwood	746	328268	52000
Norfolk State	919	295000	49605
Old Dominion	2369	448272	54416
Radford	1273	312080	51000
Mary Washington	721	449865	53045
Virginia	7431	561099	60048
Virginia Commonwealth	5825	503154	55000
Virginia Military Institute	550	364269	44999
Virginia Tech	7303	500000	51656
Virginia State	761	356524	55925

1. What code would you use to select the first, third, tenth, and twelfth entries in the `TopSalary` vector from the `Colleges` data frame? (Hints: Indexing with `[]` operator)
2. What code would you use to select the elements of the `MedianSalary` vector where the `TopSalary` is greater than $400,000? (Hints: `d$MedianSalary[d$TopSalary>400000]`)
3. What code would you use to select the rows of the data frame for colleges with less than or equal to 1000 employees? (Hints: `d[condition, ]`)
4. What code would you use to select a sample of 5 colleges from this data frame (there are 14 rows)? (Hints: `d[sample(x = 1:14, size = 5, replace = F),]`)

Suppose we have the following data frame named Countries (download here):

Nation	Region	Population	PctIncrease	GDPcapita
China	Asia	1409517397	0.4	8582
India	Asia	1339180127	1.1	1852
United States	North America	324459463	0.7	57467
Indonesia	Asia	263991379	1.1	3895
Brazil	South America	209288278	0.8	10309
Pakistan	Asia	197015955	2.0	1629
Nigeria	Africa	190886311	2.6	2640
Bangladesh	Asia	164669751	1.1	1524
Russia	Europe	143989754	0.0	10248
Mexico	North America	129163276	1.3	8562

5. What could would you use to select the rows of the data frame that have GDP per capita less than 10000 and are not in the Asia region?
6. What code would you use to select a sample of three nations from this data frame (There are 10 rows)?
7. What code would you use to select which nations saw a population percent increase greater that 1.5%?

Suppose we have the following data frame named Olympics (download here):

Year	Type	Host	Competitors	Events	Nations	Leader
1992	Summer	Spain	9356	257	169	Unified Team
1992	Winter	France	1801	57	64	Germany
1994	Winter	Norway	1737	61	67	Russia
1996	Summer	United States	10318	271	197	United States
1998	Winter	Japan	2176	68	72	Germany
2000	Summer	Australia	10651	300	199	United States
2002	Winter	United States	2399	78	78	Norway
2004	Summer	Greece	10625	301	201	United States
2006	Winter	Italy	2508	84	80	Germany
2008	Summer	China	10942	302	204	China
2010	Winter	Canada	2566	86	82	Canada
2012	Summer	United Kingdom	10768	302	204	United States
2014	Winter	Russia	2873	98	88	Russia
2016	Summer	Brazil	11238	306	207	United States
2018	Winter	South Korea	2922	102	92	Norway

8. What code would you use to select the rows of the data frame where the host nation was also the medal leader?
9. What code would you use to select the rows of the data frame where the number of competitors per event is greater than 35?
10. What code would you use to select the rows of the data frame where the number of competing nations in the Winter Olympics is at least 80?

2.3 Packages

2.3.1 Problems

1. Install the **Ecdat** package. (Hints: `install.packages()`)
2. Say that we previously installed the **Ecdat** library into R and wanted to call the library to access datasets from it. What code would we use to call the library? (Hints: `library()`)
3. Say that we then wanted to call the dataset `Diamond` from the **Ecdat** library. What code would we use to load this dataset into R? (Hints: `data()`)

2.4 Frequency and numerical exploratory analyses

2.4.1 Problems

Load the `leuk` dataset from the *MASS* library. This dataset is the survival times (`time`), white blood cell count (`wbc`), and the presence of a morphologic characteristic of white blood cells (`ag`).

1. Generate the frequency table for the presence of the morphologic characteristic.
2. Find the median and mean for survival time.
3. Find the range, IQR, variance, and standard deviation for white blood cell count.
4. Find the correlation between white blood cell count and survival time.

Load the `survey` dataset from the *MASS* library. This dataset contains the survey responses of a class of college students.

5. Create the contingency table of whether or not the student smoked (`Smoke`) and the student's exercise regimen (`Exer`). (Hints: `table()`, `DescTools::Desc()`)
6. Find the mean and median of the student's heart rate (`Pulse`). (Hints: `summary()`, `DescTools::Desc()`, `psych::describe()`)
7. Find the range, IQR, variance, and standard deviation for student age (`Age`).
8. Find the correlation between the span of the student's writing hand (`Wr.Hnd`) and nonwriting hand (`NW.Hnd`). (Hints: `cor()`, `DescTools::Desc()`)

Load the `Housing` dataset from the *Ecdat* library. This dataset looks at the variables that affect the sales price of houses.

9. Create the contingency table of whether or not the house has a recreation room (`recroom`) and whether or not the house had a full basement (`fullbase`).
10. Find the mean and median of the house's lot size (`lotsize`).
11. Find the range, IQR, variance, and standard deviation for the sales price (`price`).
12. Find the correlation between the sales price of the house (`price`) and the number of bedrooms (`bedrooms`).

2.5 Graphical exploratory analyses

Load the `Star` dataset from the *Ecdat* library. This dataset looks at the affect on class sizes on student learning.

1. Generate the scatterplot of the student's math score `tmathssk` and reading score `treadssk`. (Hints: `plot()`, `ggplot() + geom_point()`)
2. Generate the histogram of the years of teaching experience `totexpk`. (Hints: `hist()`, `ggplot() + geom_histogram()`)
3. Create a new variable in the `Star` dataset called `totalscore` that is the sum of the student's math score `tmathssk` and reading score `treadssk`. (Hints: tranformation)
4. Generate a boxplot of the student's total score `totalscore` split out by the class size type `classk`. (Hints: `boxplot()`, `ggplot() + geom_boxplot()`)

Load the `survey` dataset from the *MASS* library. This dataset contains the survey responses of a class of college students.

5. Generate the scatterplot of the student's height `Height` and writing hand span `Wr.Hnd`.
6. Generate the histogram of student age `Age`.
7. Generate a boxplot of the student's heart rate `Pulse` split out by the student's exercise regimen `Exer`.

Chapter 3

RMarkdown

RMarkdown is a framework from RStudio for easily combining your code, data, text and interactive charts into both reports and slide decks. RMarkdown is based on Markdown.

3.1 Markdown

Markdown is a markup language. It is an extremely simple markup language, so it is very popular on the Web and in other application. Markdown is used to format text on GitHub, Reddit, Stack Exchange, and Trello, and in RMarkdown. Markap laguages allow authors to annotate content. The content could be anything from reports to websites. HTML is the most widely used markup language.

Markdown was created by John Gruber and Aaron Swartz in 2004. Markup was designed that a human reader could easily parse the content.

You can download an example Markdown file to illustrate the markdown syntax:

- Headings
- Paragraphs
- Line Breaks
- Emphasis (Bold, Italic)
- Blockquotes
- Lists (Ordered, Unordered)
- Code
- Horizontal Rules
- Links
- Images
- Tables
- Footnotes

- Definition Lists

It's important to note that Markdown comes in many different flavors (versions). There are several lightweight markup languages that are supersets of Markdown. They include Gruber's basic syntax and build upon it by adding additional elements.

Many of the most popular Markdown applications use one of the following lightweight markup languages:

- CommonMark
- GitHub Flavored Markdown (GFM)
- Markdown Extra
- MultiMarkdown
- R Markdown - Pandoc

If you are not familiar with Markdown yet, or do not prefer writing Markdown code, RStudio v1.4 has included an experimental visual editor for Markdown documents, which feels similar to traditional WYSIWYG editors like Word. You can find the full documentation at RStudio Visual R Markdown. With the visual editor, you can visually edit almost any Markdown elements supported by Pandoc, such as section headers, figures, tables, footnotes, and so on.

Additional resources about Markdown:

- Markdown Cheat Sheet A quick reference to the Markdown syntax.
- Basic Syntax The Markdown elements outlined in John Gruber's design document.
- Extended Syntax Advanced features that build on the basic Markdown syntax.

3.2 RMarkdown

R Markdown understands Pandoc's Markdown, a version of Markdown with more features. This Pandoc guide provides and extensive resource for formatting options.

Rmarkdown files are plain text files that contain all of the information necessary for RStudio to generate our output files, using **rmarkdown** and **knitr** package. There are three distinct parts to the document, and in fact, each is written in a different language.

- The file header tells the **rmarkdown** package what type of file to create. In this case, an HTML document. And it's worth noting that this header is written in YAML.
- The text in the document is written in Pandoc flavored Markdown.

- Any R code that we want to include or evaluate in a document is contained within code chunks. These are delimited by pairs of three back ticks. Note that these back ticks are actually part of the Pandoc Markdown syntax. This is the beauty of RMarkdown. It allows us to combine text, images, code, and output together into a huge variety of different output formats to create rich reports and presentations.

To use RMarkdown we need an R package available from CRAN, called **rmarkdown** that you need to install to use. And you install it in the same way as you install any R package, with the function `install.packages()`. The **rmarkdown** package is developed by the folks at RStudio. Therefore, the RStudio application is designed as the document editor for RMarkdown. R Markdown files have the extension `.Rmd`. It's not impossible to use R Markdown without RStudio, but RStudio makes it a real delight to use. The **rmarkdown** package is a collection of many different tools that work together to convert your RMarkdown files, into HTML, PDF, Microsoft Word documents, and many other file types.

There are therefore two components of R Markdown: `.Rmd` file, which contains all of our content, and the **rmarkdown** package that passes the `.Rmd` file and generates to specify output files.

The basic workflow structure for an RMarkdown document is shown in Figure 3.1, highlighting the steps (arrows) and the intermediate files that are created before producing the output. The whole process is implemented via the function `rmarkdown::render()`. Each stage is explained in further detail below.

3.3 Code Chunks

To run blocks of code in RMarkdown, use code chunks. Insert a new code chunk with:

- `Command + Option + I` on a Mac, or `Ctrl + Alt + I` on Linux and Windows.
- Another option is the "Insert" drop-down Icon in the toolbar and selecting R.

We recommend learning the shortcut to save time! We'll insert a new code chunk in our R Markdown Guide in a moment.

3.3.1 Running Code

RStudio provides many options for running code chunks in the "Run" drop-down tab on the toolbar.

Before running code chunks it is often a good idea to restart your R session and start with a clean environment. Do this with `Command + Shift + F10` on a Mac or `Control + Shift + F10` on Linux and Windows.

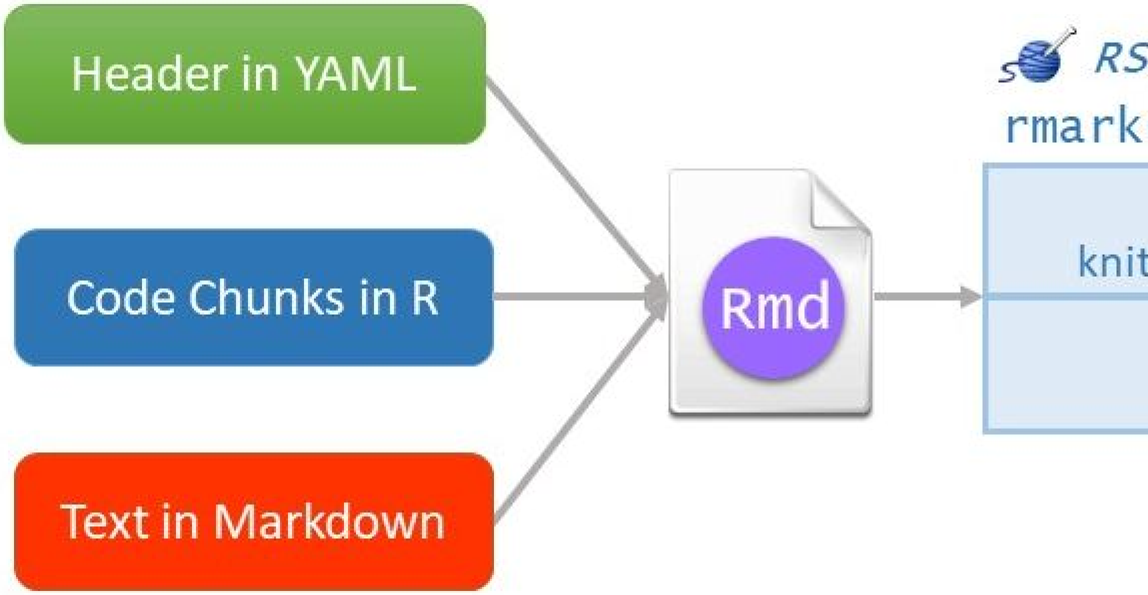


Figure 3.1: A diagram illustrating how an R Markdown document is converted to the final output document.

To save time, it's worth learning these shortcuts to run code:

- Run all chunks above the current chunk with `Command + Option + P` on a Mac, or `Ctrl + Alt + P` on Linux and Windows.
- Run the current chunk with `Command + Option + C` or `Command + Shift + Enter` on a Mac. On Linux and Windows, use `Ctrl + Alt + C` or `Ctrl + Shift + Enter` to run the current chunk.
- Run the next chunk with `Command + Option + N` on a Mac, or `Ctrl + Alt + N` on Linux and Windows.
- Run all chunks with `Command + Option + R` or `Command + A + Enter` on a Mac. On Linux and Windows, use `Ctrl + Alt + R` or `Ctrl + A + Enter` to run all chunks.

3.3.2 Control Behavior with Code Chunk Options

One of the great things about R Markdown is that you have many options to control how each chunk of code is evaluated and presented. This allows you to build presentations and reports from the ground up — including code, plots, tables, and images — while only presenting the essential information to the intended audience. For example, you can include a plot of your results without showing the code used to generate it.

Mastering code chunk options is essential to becoming a proficient RMarkdown user. The best way to learn chunk options is to try them as you need them in your reports, so don't worry about memorizing all of this now. Here are the key chunk options to learn:

- `echo = FALSE`: Do not show code in the output, but run code and produce all outputs, plots, warnings and messages. The code chunk to generate a plot in the image below is an example of this.
- `eval = FALSE`: Show code, but do not evaluate it.
- `fig.show = "hide"`: Hide plots.
- `results = "hide"`: Hides printed output.
- `include = FALSE`: Run code, but suppress all output. This is helpful for setup code.
- `message = FALSE`: Prevent packages from printing messages when they load. This also suppress messages generated by functions.
- `warning = FALSE`: Prevent packages and functions from displaying warnings.

3.3.3 Navigating Sections and Code Chunks

Naming code chunks is useful for long documents with many chunks. With R code chunks, name the chunk like this: `{r my_boring_chunk_name}`.

With named code chunks, you can navigate between chunks in the navigator included at the bottom of the R Markdown window pane. This can also make plots easy to identify by name so they can be used in other sections of your

Table 3.1: The First Few Rows of the Cars Dataset

speed	dist
4	2
4	10
7	4
7	22
8	16
9	10

document. This navigator is also useful for quickly jumping to another section of your document.

3.3.4 Table Formatting

Tables in R Markdown are displayed as you see them in the R console by default. To improve the aesthetics of a table in an RMarkdown document, use the function `knitr::kable()`. Here's an example:

```
knitr::kable(head(cars), caption = "The First Few Rows of the Cars Dataset")
```

There are many other packages for creating tables in R Markdown.

3.4 Inline Code

Directly embed R code into an R Markdown document with inline code. This is useful when you want to include information about your data in the written summary. We'll add a few examples of inline code to our R Markdown Guide to illustrate how it works.

Use inline code with `r` and add the code to evaluate within the backticks. For example, here's how we can summarize the number of rows and the number of columns in the cars dataset that's built-in to R:

```
## Inline Code

The `cars` dataset contains 50 rows and 2 columns.
```

The example above highlights how it's possible to reduce errors in reports by summarizing information programmatically. If we alter the dataset and change the number of rows and columns, we only need to rerun the code for an accurate result. This is much better than trying to remember where in the document we need to update the results, determining the new numbers, and manually changing the results. RMarkdown is a powerful because it can save time and improve the quality and accuracy of reports.

3.5 Output Format Options

Now that we have a solid understanding about how to format an RMarkdown document, let's discuss format options. Format options that apply to the entire document are specified in the YAML header. R Markdown supports many types of output formats.

The metadata specified in the YAML header controls the output. A single RMarkdown document can support many output formats. There are two types of output formats in the **rmarkdown** package: documents, and presentations. All available formats are listed below:

- `beamer_presentation`
- `context_document`
- `github_document`
- `html_document`
- `ioslides_presentation`
- `latex_document`
- `md_document`
- `odt_document`
- `pdf_document`
- `powerpoint_presentation`
- `rtf_document`
- `slidy_presentation`
- `word_document`

More details in https://bookdown.org/yihui/rmarkdown/documents.html#documents and https://bookdown.org/yihui/rmarkdown/presentations.html#presentations. There are more output formats provided in other extension packages. For the output format names in the YAML metadata of an Rmd file, you need to include the package name if a format is from an extension package, e.g.,

```
output: tufte::tufte_html
```

If the format is from the **rmarkdown** package, you do not need the `rmarkdown::` prefix (although it will not hurt).

Other packages provide even more output formats:

- The **bookdown** package, https://github.com/rstudio/bookdown, makes it easy to write books, like this one. To learn more, read *Authoring Books with R Markdown*, by Yihui Xie, which is, of course, written in bookdown.

Visit <http://www.bookdown.org> to see other bookdown books written by the wider R community.

- The **prettydoc** package, https://github.com/yixuan/prettydoc/, provides lightweight document formats with a range of attractive themes.

- The **rticles** package, https://github.com/rstudio/rticles, compiles a selection of formats tailored for specific scientific journals.

See http://rmarkdown.rstudio.com/formats.html for a list of even more formats. Also see R Markdown Theme Gallery.

3.6 Further topics and links

- Word documents https://bookdown.org/yihui/rmarkdown-cookbook/word.html https://rmarkdown.rstudio.com/articles_docx.html
- Bibliography https://bookdown.org/yihui/rmarkdown-cookbook/bibliography.html Citation Style Language - Style Repository
- Cross-referencing within documents https://bookdown.org/yihui/rmarkdown-cookbook/cross-ref.html
- Create diagrams https://bookdown.org/yihui/rmarkdown-cookbook/diagrams.html

3.7 Additional Resources

- R Markdown Cookbook A comprehensive free online book that contains almost everything you need to know about RMarkdown.
- RMarkdown for Scientists
- RStudio Articles for RMarkdown RStudio has published a few in-depth how to articles about using RMarkdown.
- R for Data Science Hadley Wickham provides a great overview of authoring with RMarkdown.
- R Markdown: The Definitive Guide It contains a large number of technical details, it may serve you better as a reference book than a textbook.
- Online lesson from RStudio
- R Markdown Cheatsheet. RStudio has published numerous cheatsheets for working with R, including a detailed cheatsheet on using R Markdown! The R Markdown cheatsheet can be accessed from within RStudio by selecting `Help > Cheatsheets > R Markdown Cheat Sheet`.

Chapter 4

Advanced data manipulation

This chapter focuses exclusively on advanced data manipulation. I therefore assume a basic level of comfort with data manipulation.

4.1 Importing data

Most of the data used for analysis is found in the outside world and needs to be imported into R. Data comes in different formats.

- **Delimited text files** are the most common way of transferring data between systems in general. They are files that store tabular data using special characters (known as delimiters) to indicate rows and columns. These delimiters include commas, tabs, space, semicolons (;), pipes (|), etc. The function `read.table()` is used to read delimited text files. It accepts as argument, the file path of the file and returns as output a data frame.

- **Binary files** are more complex than plain text files and accessing the information in binary files requires the use of special software. Some examples of binary files that we will frequently see include Microsoft Excel spreadsheets, SAS data sets, Stata data sets, and SPSS data set. The **foreign** package contains functions that may be used to import SAS data sets and Stata data sets, and is installed by default when you install R on your computer. We can use the **readxl** package to import Microsoft Excel files, and the **haven** package to import SAS and Stata data sets. We aren't going to use these packages in this chapter. Instead, we're going to use the best **rio** package to import data in the examples below.

```
# Description of gapminder:
# help(gapminder, package = "gapminder")

# importing the gapminder dataset - Delimited text files - ANSI (CP1250)
gapminder_cp1250 <- read.table(file = "data/gapminder_ext_CP1250.txt", header = T, sep

# importing the gapminder dataset - Delimited text files - UTF-8
gapminder_utf8 <- read.table(file = "data/gapminder_ext_UTF-8.txt", header = T, sep = "

# importing the gapminder dataset - Binary files
library(rio)
gapminder_xlsx <- import(file = "data/gapminder_ext.xlsx")

# checking class
class(gapminder_xlsx)
#> [1] "data.frame"
```

4.1.1 Import files directly from the web

The functions `read.table()` and `rio::import()` accept a URL in the place of a dataset and downloads the dataset directly.

```
# NCHS - Death rates and life expectancy at birth:
# https://data.cdc.gov/NCHS/NCHS-Death-rates-and-life-expectancy-at-birth/w9j2-ggv5

# storing URL
data_url <- 'https://data.cdc.gov/api/views/w9j2-ggv5/rows.csv?accessType=DOWNLOAD'

# reading in data from the URL - Delimited text file
life_expectancy <- read.table(data_url, header = T, sep = ",", dec = ".")

head(life_expectancy, 3)
#>   Year      Race        Sex Average.Life.Expectancy..Years.
#> 1 1900 All Races Both Sexes                            47.3
#> 2 1901 All Races Both Sexes                            49.1
#> 3 1902 All Races Both Sexes                            51.5
#>   Age.adjusted.Death.Rate
#> 1                  2518.0
#> 2                  2473.1
#> 3                  2301.3
nrow(life_expectancy)
#> [1] 1071


# Description of Potthoff-Roy data:
```

```
# help(potthoffroy, package = "mice")

# storing URL
data_url <- "https://raw.github.com/abarik/rdata/master/r_alapok/pothoff2.xlsx"
library(rio)
pothoff <- import(file = data_url)
str(pothoff)
#> 'data.frame':	108 obs. of  5 variables:
#>  $ person: num  1 1 1 1 2 2 2 2 3 3 ...
#>  $ sex   : chr  "F" "F" "F" "F" ...
#>  $ age   : num  8 10 12 14 8 10 12 14 8 10 ...
#>  $ y     : num  21 20 21.5 23 21 21.5 24 25.5 20.5 24 ...
#>  $ agefac: num  8 10 12 14 8 10 12 14 8 10 ...
```

4.2 Exporting data

The function `write.table()` are used to export data to delimited text file. The function `rio::export()` is used to export data to worksheets in an Excel file (or other binary file). The type of the binary file will depend on the extension given to the file name.

```
# exporting the gapminder dataset - Delimited text files - ANSI (CP1250)
write.table(x = gapminder_xlsx, file = "output/data/gapminder_CP1250.csv", quote = F, sep = ";",

# exporting the gapminder dataset - Delimited text files - UTF-8
write.table(x = gapminder_xlsx, file = "output/data/gapminder_UTF-8.csv", quote = F, sep = ";",

# exporting the gapminder dataset - Binary files
library(rio)
export(x = gapminder_xlsx, file = "output/data/gapminder.xlsx", overwrite = T)
export(x = gapminder_xlsx, file = "output/data/gapminder.sav")
```

4.3 Inspecting a data frame

We use the following functions to inspect a data frame:

- `dim()` returns dimensions
- `nrow()` returns number of rows
- `ncol()` returns number of columns
- `str()` returns column names and their data types plus some first few values
- `head()` returns the first six rows by default but can be changed using the argument `n`
- `tail()` returns the last six rows by default but can be changed using the

argument `n`

```
dim(gapminder_xlsx)
#> [1] 1704    8
nrow(gapminder_xlsx)
#> [1] 1704
ncol(gapminder_xlsx)
#> [1] 8
str(gapminder_xlsx)
#> 'data.frame':    1704 obs. of  8 variables:
#>  $ country      : chr  "Afghanistan" "Afghanistan" "Afghanistan" "Afghanistan" ...
#>  $ continent    : chr  "Asia" "Asia" "Asia" "Asia" ...
#>  $ year         : num  1952 1957 1962 1967 1972 ...
#>  $ lifeExp      : num  28.8 30.3 32 34 36.1 ...
#>  $ pop          : num  8425333 9240934 10267083 11537966 13079460 ...
#>  $ gdpPercap    : num  779 821 853 836 740 ...
#>  $ country_hun  : chr  "Afganisztán" "Afganisztán" "Afganisztán" "Afganisztán" ...
#>  $ continent_hun: chr  "Ázsia" "Ázsia" "Ázsia" "Ázsia" ...
head(gapminder_xlsx)
#>       country continent year lifeExp      pop gdpPercap
#> 1 Afghanistan      Asia 1952  28.801  8425333  779.4453
#> 2 Afghanistan      Asia 1957  30.332  9240934  820.8530
#> 3 Afghanistan      Asia 1962  31.997 10267083  853.1007
#> 4 Afghanistan      Asia 1967  34.020 11537966  836.1971
#> 5 Afghanistan      Asia 1972  36.088 13079460  739.9811
#> 6 Afghanistan      Asia 1977  38.438 14880372  786.1134
#>   country_hun continent_hun
#> 1 Afganisztán         Ázsia
#> 2 Afganisztán         Ázsia
#> 3 Afganisztán         Ázsia
#> 4 Afganisztán         Ázsia
#> 5 Afganisztán         Ázsia
#> 6 Afganisztán         Ázsia
tail(gapminder_xlsx, n = 4)
#>       country continent year lifeExp      pop gdpPercap
#> 1701 Zimbabwe    Africa 1992  60.377 10704340  693.4208
#> 1702 Zimbabwe    Africa 1997  46.809 11404948  792.4500
#> 1703 Zimbabwe    Africa 2002  39.989 11926563  672.0386
#> 1704 Zimbabwe    Africa 2007  43.487 12311143  469.7093
#>      country_hun continent_hun
#> 1701    Zimbabwe        Afrika
#> 1702    Zimbabwe        Afrika
#> 1703    Zimbabwe        Afrika
#> 1704    Zimbabwe        Afrika
```

4.4 Manipulating Columns

4.4.1 Changing column type

After importing data, column types can be changed by assigning new data types to them.

```
str(gapminder_xlsx)
#> 'data.frame':	1704 obs. of  8 variables:
#>  $ country      : chr  "Afghanistan" "Afghanistan" "Afghanistan" "Afghanistan" ...
#>  $ continent    : chr  "Asia" "Asia" "Asia" "Asia" ...
#>  $ year         : num  1952 1957 1962 1967 1972 ...
#>  $ lifeExp      : num  28.8 30.3 32 34 36.1 ...
#>  $ pop          : num  8425333 9240934 10267083 11537966 13079460 ...
#>  $ gdpPercap    : num  779 821 853 836 740 ...
#>  $ country_hun  : chr  "Afganisztán" "Afganisztán" "Afganisztán" "Afganisztán" ...
#>  $ continent_hun: chr  "Ázsia" "Ázsia" "Ázsia" "Ázsia" ...

# changing column type
gapminder_xlsx$country <- factor(gapminder_xlsx$country)
gapminder_xlsx$continent <- factor(gapminder_xlsx$continent)
gapminder_xlsx$country_hun <- factor(gapminder_xlsx$country_hun)
gapminder_xlsx$continent_hun <- factor(gapminder_xlsx$continent_hun)

str(gapminder_xlsx)
#> 'data.frame':	1704 obs. of  8 variables:
#>  $ country      : Factor w/ 142 levels "Afghanistan",..: 1 1 1 1 1 1 1 1 1 1 ...
#>  $ continent    : Factor w/ 5 levels "Africa","Americas",..: 3 3 3 3 3 3 3 3 3 3 ...
#>  $ year         : num  1952 1957 1962 1967 1972 ...
#>  $ lifeExp      : num  28.8 30.3 32 34 36.1 ...
#>  $ pop          : num  8425333 9240934 10267083 11537966 13079460 ...
#>  $ gdpPercap    : num  779 821 853 836 740 ...
#>  $ country_hun  : Factor w/ 142 levels "Afganisztán",..: 1 1 1 1 1 1 1 1 1 1 ...
#>  $ continent_hun: Factor w/ 5 levels "Afrika","Amerika",..: 3 3 3 3 3 3 3 3 3 3 ...
```

4.4.2 Renaming columns

After importing data, columns can be renamed by assigning new names to them.

```
names(gapminder_utf8)
#> [1] "country"       "continent"     "year"
#> [4] "lifeExp"       "pop"           "gdpPercap"
#> [7] "country_hun"   "continent_hun"
names(gapminder_utf8)[1] <- "orszag"
names(gapminder_utf8)[2] <- "kontinens"
names(gapminder_utf8)
#> [1] "orszag"        "kontinens"     "year"
```

```
#> [4] "lifeExp"       "pop"           "gdpPercap"
#> [7] "country_hun"   "continent_hun"

names(gapminder_utf8)
#> [1] "orszag"        "kontinens"     "year"
#> [4] "lifeExp"       "pop"           "gdpPercap"
#> [7] "country_hun"   "continent_hun"
names(gapminder_utf8)[7:8] <- c("orszag_hun", "kontinens_hun")
names(gapminder_utf8)
#> [1] "orszag"        "kontinens"     "year"
#> [4] "lifeExp"       "pop"           "gdpPercap"
#> [7] "orszag_hun"    "kontinens_hun"
```

4.4.3 Insert and derive new columns

```
# Here's a data set of 1,000 most popular movies on IMDB in the last 10 years.
# https://www.kaggle.com/PromptCloudHQ/imdb-data/version/1
mov <- read.table(file = "data/IMDB-Movie-Data.csv", header = T, sep = ",", dec = ".",
                  comment.char = "")
str(mov)
#> 'data.frame':    1000 obs. of  12 variables:
#>  $ Rank              : int  1 2 3 4 5 6 7 8 9 10 ...
#>  $ Title             : chr  "Guardians of the Galaxy" "Prometheus" "Split" "Sing" .
#>  $ Genre             : chr  "Action,Adventure,Sci-Fi" "Adventure,Mystery,Sci-Fi" "H
#>  $ Description       : chr  "A group of intergalactic criminals are forced to work
#>  $ Director          : chr  "James Gunn" "Ridley Scott" "M. Night Shyamalan" "Chris
#>  $ Actors            : chr  "Chris Pratt, Vin Diesel, Bradley Cooper, Zoe Saldana"
#>  $ Year              : int  2014 2012 2016 2016 2016 2016 2016 2016 2016 2016 ...
#>  $ Runtime..Minutes. : int  121 124 117 108 123 103 128 89 141 116 ...
#>  $ Rating            : num  8.1 7 7.3 7.2 6.2 6.1 8.3 6.4 7.1 7 ...
#>  $ Votes             : int  757074 485820 157606 60545 393727 56036 258682 2490 718
#>  $ Revenue..Millions.: num  333 126 138 270 325 ...
#>  $ Metascore         : int  76 65 62 59 40 42 93 71 78 41 ...
names(mov) <- c('Rank', 'Title', 'Genre', 'Description', 'Director', 'Actors', 'Year',
                'Runtime', 'Rating', 'Votes', 'Revenue', 'Metascore')
```

4.4.3.1 Inserting a new column

To insert a new column, we index the data frame by the new column name and assign it values.

```
# adding a new column known as example
movies <- mov[,c(2, 7, 11, 12)]
set.seed(123)
movies$Example <- sample(x = 1000)
```

```
head(movies)
#>                     Title Year Revenue Metascore Example
#> 1 Guardians of the Galaxy 2014  333.13        76     415
#> 2              Prometheus 2012  126.46        65     463
#> 3                   Split 2016  138.12        62     179
#> 4                    Sing 2016  270.32        59     526
#> 5           Suicide Squad 2016  325.02        40     195
#> 6          The Great Wall 2016   45.13        42     938
```

4.4.3.2 Duplicating a column

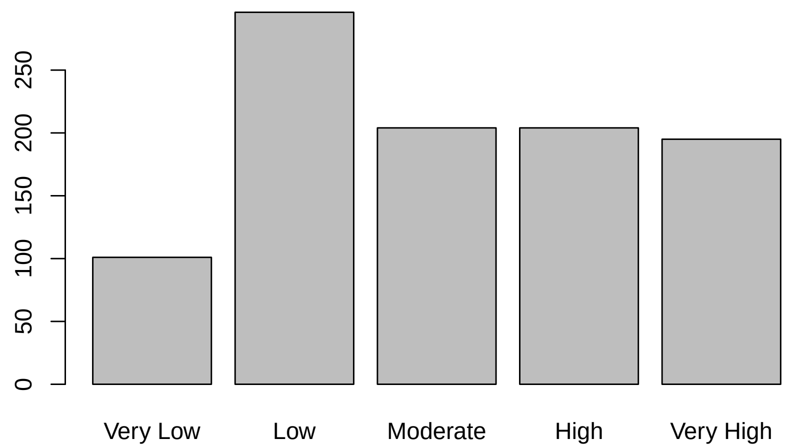
Duplicating a column is like inserting a new one. We simply select it and assign it a new name.

```
movies <- mov[, c(2, 7, 11, 12)]
movies$Metascore.2 <- movies$Metascore
head(movies)
#>                     Title Year Revenue Metascore
#> 1 Guardians of the Galaxy 2014  333.13        76
#> 2              Prometheus 2012  126.46        65
#> 3                   Split 2016  138.12        62
#> 4                    Sing 2016  270.32        59
#> 5           Suicide Squad 2016  325.02        40
#> 6          The Great Wall 2016   45.13        42
#>   Metascore.2
#> 1          76
#> 2          65
#> 3          62
#> 4          59
#> 5          40
#> 6          42
```

4.4.3.3 Deriving a new column from an existing one

```
movies <- mov[, c(2, 7, 9, 12)]
movies$Movie.Class <-
cut(movies$Rating,
    breaks = c(0, 5.5, 6.5, 7, 7.5, 10),
    labels = c("Very Low", "Low", "Moderate", "High", "Very High"))
head(movies)
#>                     Title Year Rating Metascore Movie.Class
#> 1 Guardians of the Galaxy 2014    8.1        76   Very High
#> 2              Prometheus 2012    7.0        65    Moderate
#> 3                   Split 2016    7.3        62        High
#> 4                    Sing 2016    7.2        59        High
```

```
#> 5            Suicide Squad 2016    6.2        40          Low
#> 6           The Great Wall 2016    6.1        42          Low

# plotting the new column
plot(movies$Movie.Class)
```



4.4.3.4 Deriving a new column from a calculation

```
movies <- mov[, c(2, 5, 7, 8, 11)]
movies$Rev.Run <- round(movies$Revenue/movies$Runtime, 2)
head(movies)
#>                     Title             Director Year Runtime
#> 1 Guardians of the Galaxy           James Gunn 2014     121
#> 2              Prometheus         Ridley Scott 2012     124
#> 3                   Split   M. Night Shyamalan 2016     117
#> 4                    Sing Christophe Lourdelet 2016     108
#> 5           Suicide Squad           David Ayer 2016     123
#> 6          The Great Wall          Yimou Zhang 2016     103
#>   Revenue Rev.Run
#> 1  333.13    2.75
#> 2  126.46    1.02
#> 3  138.12    1.18
#> 4  270.32    2.50
#> 5  325.02    2.64
```

```
#> 6   45.13    0.44
```

4.4.3.5 Updating a column

```
movies <- mov[,c(2, 5, 7, 9, 11, 12)]
movies$Director <- toupper(movies$Director)
movies$Title <- tolower(movies$Title)
head(movies)
#>                     Title            Director Year Rating
#> 1 guardians of the galaxy          JAMES GUNN 2014    8.1
#> 2              prometheus        RIDLEY SCOTT 2012    7.0
#> 3                   split   M. NIGHT SHYAMALAN 2016    7.3
#> 4                    sing CHRISTOPHE LOURDELET 2016    7.2
#> 5           suicide squad          DAVID AYER 2016    6.2
#> 6          the great wall         YIMOU ZHANG 2016    6.1
#>   Revenue Metascore
#> 1  333.13        76
#> 2  126.46        65
#> 3  138.12        62
#> 4  270.32        59
#> 5  325.02        40
#> 6   45.13        42
```

4.4.4 Sorting and ranking

4.4.4.1 Sorting a data frame

The `order()` function is used to sort a data frame. It takes a column and returns indices in ascending order. To reverse this, use `decreasing = TRUE`. Once the indices are sorted, they are used to index the data frame. The function `order()` also works on character columns as well and on multiple columns.

```
# sorting by revenue
movies <- mov[, c(2, 7, 11, 12)]
movies_ordered <- movies[order(movies$Revenue),]
head(movies_ordered)
#>                Title Year Revenue Metascore
#> 232 A Kind of Murder 2016    0.00        50
#> 28        Dead Awake 2016    0.01        NA
#> 69         Wakefield 2016    0.01        61
#> 322         Lovesong 2016    0.01        74
#> 678      Love, Rosie 2014    0.01        44
#> 962  Into the Forest 2015    0.01        59
tail(movies_ordered)
#>                              Title Year Revenue Metascore
#> 977                   Dark Places 2015      NA        39
```

```
#> 978                  Amateur Night 2016      NA        38
#> 979 It's Only the End of the World 2016      NA        48
#> 989                        Martyrs 2008      NA        89
#> 996            Secret in Their Eyes 2015      NA        45
#> 999                   Search Party 2014      NA        22

# sort decreasing
movies_ordered <- movies[order(movies$Revenue, decreasing = T),]
head(movies_ordered)
#>                                         Title Year Revenue
#> 51 Star Wars: Episode VII - The Force Awakens 2015  936.63
#> 88                                     Avatar 2009  760.51
#> 86                             Jurassic World 2015  652.18
#> 77                               The Avengers 2012  623.28
#> 55                            The Dark Knight 2008  533.32
#> 13                                  Rogue One 2016  532.17
#>    Metascore
#> 51        81
#> 88        83
#> 86        59
#> 77        69
#> 55        82
#> 13        65
tail(movies_ordered)
#>                               Title Year Revenue Metascore
#> 977                     Dark Places 2015      NA        39
#> 978                   Amateur Night 2016      NA        38
#> 979 It's Only the End of the World 2016      NA        48
#> 989                         Martyrs 2008      NA        89
#> 996            Secret in Their Eyes 2015      NA        45
#> 999                    Search Party 2014      NA        22

# sort decreasing using the negative sign
movies_ordered <- movies[order(-movies$Revenue),]
head(movies_ordered)
#>                                         Title Year Revenue
#> 51 Star Wars: Episode VII - The Force Awakens 2015  936.63
#> 88                                     Avatar 2009  760.51
#> 86                             Jurassic World 2015  652.18
#> 77                               The Avengers 2012  623.28
#> 55                            The Dark Knight 2008  533.32
#> 13                                  Rogue One 2016  532.17
#>    Metascore
#> 51        81
#> 88        83
```

```
#> 86        59
#> 77        69
#> 55        82
#> 13        65
tail(movies_ordered)
#>                                Title Year Revenue Metascore
#> 977                     Dark Places 2015      NA        39
#> 978                   Amateur Night 2016      NA        38
#> 979 It's Only the End of the World 2016      NA        48
#> 989                         Martyrs 2008      NA        89
#> 996             Secret in Their Eyes 2015      NA        45
#> 999                    Search Party 2014      NA        22
```

By default, `NA` values appear at the end of the sorted column, but this can be changed by setting `na.last = FALSE` so that they appear first.

```
# placing NA at the beginning
movies_ordered <- movies[order(movies$Revenue, na.last = FALSE),]
head(movies_ordered)
#>                      Title Year Revenue Metascore
#> 8                 Mindhorn 2016      NA        71
#> 23          Hounds of Love 2016      NA        72
#> 26         Paris pieds nus 2016      NA        NA
#> 40              5- 25- 77 2007      NA        NA
#> 43 Don't Fuck in the Woods 2016      NA        NA
#> 48                  Fallen 2016      NA        NA
tail(movies_ordered)
#>                                           Title Year Revenue
#> 13                                    Rogue One 2016  532.17
#> 55                              The Dark Knight 2008  533.33
#> 77                                 The Avengers 2012  623.28
#> 86                               Jurassic World 2015  652.18
#> 88                                       Avatar 2009  760.51
#> 51 Star Wars: Episode VII - The Force Awakens 2015  936.63
#>    Metascore
#> 13        65
#> 55        82
#> 77        69
#> 86        59
#> 88        83
#> 51        81

# sorting on multiple columns
movies_ordered <- movies[order(movies$Metascore, movies$Revenue, decreasing = T),]
head(movies_ordered, 10)
#>                       Title Year Revenue Metascore
```

```
#> 657              Boyhood 2014   25.36       100
#> 42             Moonlight 2016   27.85        99
#> 231      Pan's Labyrinth 2006   37.62        98
#> 510              Gravity 2013  274.08        96
#> 490          Ratatouille 2007  206.44        96
#> 112       12 Years a Slave 2013   56.67        96
#> 22  Manchester by the Sea 2016   47.70        96
#> 325     The Social Network 2010   96.92        95
#> 407       Zero Dark Thirty 2012   95.72        95
#> 502                 Carol 2015    0.25        95
```

4.4.4.2 Ranking

The function `rank()` ranks column values. It does this in ascending order but can be reversed by placing a negative sign in front of the ranking column as there is no decreasing argument here as was the case with the `order()` function.

```
# returning ranks by revenue
rank(movies$Revenue)[1:10]
#>  [1] 841 678 702 819 839 419 724 873 182 623

# adding a rank to the data frame
movies <- mov[, c(2, 7, 11, 12)]
movies$Ranking <- rank(movies$Revenue)
head(movies)
#>                     Title Year Revenue Metascore Ranking
#> 1 Guardians of the Galaxy 2014  333.13        76     841
#> 2              Prometheus 2012  126.46        65     678
#> 3                   Split 2016  138.12        62     702
#> 4                    Sing 2016  270.32        59     819
#> 5           Suicide Squad 2016  325.02        40     839
#> 6          The Great Wall 2016   45.13        42     419

# sorting by rank
movies <- mov[, c(2, 7, 11, 12)]
movies$Ranking <- rank(movies$Revenue)
movies <- movies[order(movies$Ranking), ]
head(movies)
#>                  Title Year Revenue Metascore Ranking
#> 232 A Kind of Murder 2016    0.00        50       1
#> 28        Dead Awake 2016    0.01        NA       4
#> 69         Wakefield 2016    0.01        61       4
#> 322         Lovesong 2016    0.01        74       4
#> 678      Love, Rosie 2014    0.01        44       4
#> 962  Into the Forest 2015    0.01        59       4
```

```
# placing NA values at the beginning
movies <- mov[, c(2, 7, 11, 12)]
movies$Ranking <- rank(movies$Revenue, na.last = F)
movies <- movies[order(movies$Ranking), ]
head(movies)
#>                      Title Year Revenue Metascore Ranking
#> 8                 Mindhorn 2016      NA        71       1
#> 23          Hounds of Love 2016      NA        72       2
#> 26         Paris pieds nus 2016      NA        NA       3
#> 40               5- 25- 77 2007      NA        NA       4
#> 43 Don't Fuck in the Woods 2016      NA        NA       5
#> 48                  Fallen 2016      NA        NA       6
```

There is no decreasing argument with `rank()`, hence our only chance of performing a decreasing rank is to use the negative sign.

```
# performing a decreasing rank
movies <- mov[, c(2, 7, 8, 11)]
movies$Ranking <- rank(-movies$Revenue)
movies <- movies[order(movies$Ranking), ]
head(movies)
#>                                          Title Year Runtime
#> 51 Star Wars: Episode VII - The Force Awakens 2015     136
#> 88                                      Avatar 2009     162
#> 86                              Jurassic World 2015     124
#> 77                                The Avengers 2012     143
#> 55                             The Dark Knight 2008     152
#> 13                                   Rogue One 2016     133
#>    Revenue Ranking
#> 51  936.63       1
#> 88  760.51       2
#> 86  652.18       3
#> 77  623.28       4
#> 55  533.32       5
#> 13  532.17       6
```

4.4.5 Splitting and Merging columns

4.4.5.1 Splitting columns

To split a data frame, we do the following

- select the column concerned and pass it to the function `strsplit()` together with the string to split on. This will return a list
- using the function `do.call('rbind', dfs)` convert the list to a data frame
- rename the columns of the new data frame

- finally using `cbind()`, combine the new data frame to the original one

```
# Airports are ranked by travellers and experts based on various measures.
# https://www.kaggle.com/jonahmary17/airports

# reading data
busiestAirports <- read.table(file = "data/busiestAirports.csv",
                              header = T,
                              sep=",",
                              dec = ".",
                              quote = "\"")

busiestAirports <- busiestAirports[-c(1, 2, 3, 4, 8)]
head(busiestAirports, 3)
#>   code.iata.icao.                 location
#> 1        ATL/KATL         Atlanta, Georgia
#> 2        PEK/ZBAA Chaoyang-Shunyi, Beijing
#> 3        DXB/OMDB           Garhoud, Dubai
#>                country
#> 1        United States
#> 2                China
#> 3 United Arab Emirates

# splitting column
strsplit(busiestAirports$code.iata.icao.,'/')[1:3]
#> [[1]]
#> [1] "ATL"  "KATL"
#>
#> [[2]]
#> [1] "PEK"  "ZBAA"
#>
#> [[3]]
#> [1] "DXB"  "OMDB"

# converting to a data frame
iata_icao <-
data.frame(do.call('rbind', strsplit(busiestAirports$code.iata.icao., '/')))
head(iata_icao, 3)
#>    X1   X2
#> 1 ATL KATL
#> 2 PEK ZBAA
#> 3 DXB OMDB

# renaming columns
names(iata_icao) <- c('iata', 'icao')
head(iata_icao, 3)
```

```
#>   iata icao
#> 1  ATL KATL
#> 2  PEK ZBAA
#> 3  DXB OMDB

# combining both data frames
busiest_Airports <- cbind(busiestAirports[-1], iata_icao)
head(busiest_Airports)
#>                    location              country iata icao
#> 1         Atlanta, Georgia        United States  ATL KATL
#> 2 Chaoyang-Shunyi, Beijing                China  PEK ZBAA
#> 3           Garhoud, Dubai United Arab Emirates  DXB OMDB
#> 4  Los Angeles, California        United States  LAX KLAX
#> 5               Ota, Tokyo                Japan  HND RJTT
#> 6        Chicago, Illinois        United States  ORD KORD
```

4.4.5.2 Merging columns

The function `paste()` is used to merge columns.

```
# merging iata and icao into iata_icao
busiest_Airports$iata_icao <-
paste(busiest_Airports$iata, busiest_Airports$icao, sep = '-')
head(busiest_Airports)
#>                    location              country iata icao
#> 1         Atlanta, Georgia        United States  ATL KATL
#> 2 Chaoyang-Shunyi, Beijing                China  PEK ZBAA
#> 3           Garhoud, Dubai United Arab Emirates  DXB OMDB
#> 4  Los Angeles, California        United States  LAX KLAX
#> 5               Ota, Tokyo                Japan  HND RJTT
#> 6        Chicago, Illinois        United States  ORD KORD
#>   iata_icao
#> 1  ATL-KATL
#> 2  PEK-ZBAA
#> 3  DXB-OMDB
#> 4  LAX-KLAX
#> 5  HND-RJTT
#> 6  ORD-KORD
```

4.5 Selecting columns

The function `subset()` or `[` is used to select columns.

```
head(gapminder_cp1250[, c(1, 3)])
#>       country year
#> 1 Afghanistan 1952
```

```
#> 2 Afghanistan 1957
#> 3 Afghanistan 1962
#> 4 Afghanistan 1967
#> 5 Afghanistan 1972
#> 6 Afghanistan 1977
head(gapminder_cp1250[, c("country", "gdpPercap")])
#>       country gdpPercap
#> 1 Afghanistan  779.4453
#> 2 Afghanistan  820.8530
#> 3 Afghanistan  853.1007
#> 4 Afghanistan  836.1971
#> 5 Afghanistan  739.9811
#> 6 Afghanistan  786.1134
head(subset(gapminder_cp1250, select = c("country", "gdpPercap")))
#>       country gdpPercap
#> 1 Afghanistan  779.4453
#> 2 Afghanistan  820.8530
#> 3 Afghanistan  853.1007
#> 4 Afghanistan  836.1971
#> 5 Afghanistan  739.9811
#> 6 Afghanistan  786.1134
```

4.6 Deleting columns

There is no special function to delete columns but `[` and `NULL` can be used to drop unwanted columns.

```
str(gapminder_cp1250)
#> 'data.frame':    1704 obs. of  8 variables:
#>  $ country      : chr  "Afghanistan" "Afghanistan" "Afghanistan" "Afghanistan" ...
#>  $ continent    : chr  "Asia" "Asia" "Asia" "Asia" ...
#>  $ year         : int  1952 1957 1962 1967 1972 1977 1982 1987 1992 1997 ...
#>  $ lifeExp      : num  28.8 30.3 32 34 36.1 ...
#>  $ pop          : int  8425333 9240934 10267083 11537966 13079460 14880372 12881816
#>  $ gdpPercap    : num  779 821 853 836 740 ...
#>  $ country_hun  : chr  "Afganisztán" "Afganisztán" "Afganisztán" "Afganisztán" ...
#>  $ continent_hun: chr  "Ázsia" "Ázsia" "Ázsia" "Ázsia" ...
gapminder_cp1250$pop <- NULL
str(gapminder_cp1250)
#> 'data.frame':    1704 obs. of  7 variables:
#>  $ country      : chr  "Afghanistan" "Afghanistan" "Afghanistan" "Afghanistan" ...
#>  $ continent    : chr  "Asia" "Asia" "Asia" "Asia" ...
#>  $ year         : int  1952 1957 1962 1967 1972 1977 1982 1987 1992 1997 ...
#>  $ lifeExp      : num  28.8 30.3 32 34 36.1 ...
#>  $ gdpPercap    : num  779 821 853 836 740 ...
```

```
#>  $ country_hun  : chr  "Afganisztán" "Afganisztán" "Afganisztán" "Afganisztán" ...
#>  $ continent_hun: chr  "Ázsia" "Ázsia" "Ázsia" "Ázsia" ...

str(gapminder_cp1250)
#> 'data.frame':    1704 obs. of  7 variables:
#>  $ country      : chr  "Afghanistan" "Afghanistan" "Afghanistan" "Afghanistan" ...
#>  $ continent    : chr  "Asia" "Asia" "Asia" "Asia" ...
#>  $ year         : int  1952 1957 1962 1967 1972 1977 1982 1987 1992 1997 ...
#>  $ lifeExp      : num  28.8 30.3 32 34 36.1 ...
#>  $ gdpPercap    : num  779 821 853 836 740 ...
#>  $ country_hun  : chr  "Afganisztán" "Afganisztán" "Afganisztán" "Afganisztán" ...
#>  $ continent_hun: chr  "Ázsia" "Ázsia" "Ázsia" "Ázsia" ...
gapminder_cp1250 <- gapminder_cp1250[, c(1, 2, 5, 6)]
str(gapminder_cp1250)
#> 'data.frame':    1704 obs. of  4 variables:
#>  $ country    : chr  "Afghanistan" "Afghanistan" "Afghanistan" "Afghanistan" ...
#>  $ continent  : chr  "Asia" "Asia" "Asia" "Asia" ...
#>  $ gdpPercap  : num  779 821 853 836 740 ...
#>  $ country_hun: chr  "Afganisztán" "Afganisztán" "Afganisztán" "Afganisztán" ...
```

4.7 Manipulating Rows

4.7.1 Renaming rows

After importing data, rows can be renamed by assigning new names to them.

```
rownames(gapminder_utf8)[1:6]
#> [1] "1" "2" "3" "4" "5" "6"
rownames(gapminder_utf8) <- paste0("RN-", 1:nrow(gapminder_utf8))
head(gapminder_utf8)
#>           orszag kontinens year lifeExp      pop gdpPercap
#> RN-1 Afghanistan      Asia 1952  28.801  8425333  779.4453
#> RN-2 Afghanistan      Asia 1957  30.332  9240934  820.8530
#> RN-3 Afghanistan      Asia 1962  31.997 10267083  853.1007
#> RN-4 Afghanistan      Asia 1967  34.020 11537966  836.1971
#> RN-5 Afghanistan      Asia 1972  36.088 13079460  739.9811
#> RN-6 Afghanistan      Asia 1977  38.438 14880372  786.1134
#>        orszag_hun kontinens_hun
#> RN-1 Afganisztán         Ázsia
#> RN-2 Afganisztán         Ázsia
#> RN-3 Afganisztán         Ázsia
#> RN-4 Afganisztán         Ázsia
#> RN-5 Afganisztán         Ázsia
#> RN-6 Afganisztán         Ázsia
rownames(gapminder_utf8) <- 1:nrow(gapminder_utf8) # reset row names
```

```
head(gapminder_utf8)
#>        orszag kontinens year lifeExp      pop gdpPercap
#> 1 Afghanistan      Asia 1952  28.801  8425333  779.4453
#> 2 Afghanistan      Asia 1957  30.332  9240934  820.8530
#> 3 Afghanistan      Asia 1962  31.997 10267083  853.1007
#> 4 Afghanistan      Asia 1967  34.020 11537966  836.1971
#> 5 Afghanistan      Asia 1972  36.088 13079460  739.9811
#> 6 Afghanistan      Asia 1977  38.438 14880372  786.1134
#>    orszag_hun kontinens_hun
#> 1 Afganisztán         Ázsia
#> 2 Afganisztán         Ázsia
#> 3 Afganisztán         Ázsia
#> 4 Afganisztán         Ázsia
#> 5 Afganisztán         Ázsia
#> 6 Afganisztán         Ázsia
```

4.7.2 Adding rows

4.7.2.1 Adding rows by assignment

```
movies <- mov[, c(2, 5, 7, 9, 11, 12)]
tail(movies, 3)
#>                       Title         Director Year Rating
#> 998  Step Up 2: The Streets       Jon M. Chu 2008    6.2
#> 999            Search Party   Scot Armstrong 2014    5.6
#> 1000             Nine Lives Barry Sonnenfeld 2016    5.3
#>      Revenue Metascore
#> 998    58.01        50
#> 999       NA        22
#> 1000   19.64        11

# inserting rows
movies[1001,] <- c("the big g", "goro lovic", 2015, 9.9, 1000, 100)
movies[1002,] <- c("luv of my life", "nema lovic", 2016, 7.9, 150, 65)
movies[1003,] <- c("everyday", "goro lovic", 2014, 4.4, 170, 40)
tail(movies)
#>                       Title         Director Year Rating
#> 998  Step Up 2: The Streets       Jon M. Chu 2008    6.2
#> 999            Search Party   Scot Armstrong 2014    5.6
#> 1000             Nine Lives Barry Sonnenfeld 2016    5.3
#> 1001              the big g       goro lovic 2015    9.9
#> 1002         luv of my life       nema lovic 2016    7.9
#> 1003               everyday       goro lovic 2014    4.4
#>      Revenue Metascore
#> 998    58.01        50
```

```
#> 999     <NA>         22
#> 1000   19.64         11
#> 1001    1000        100
#> 1002     150         65
#> 1003     170         40

# using nrow
movies <- mov[, c(2, 5, 7, 9, 11, 12)]
movies[nrow(movies) + 1,] <- c("the big g", "goro lovic", 2015, 9.9, 1000, 100)
movies[nrow(movies) + 1,] <- c("luv of my life", "nema lovic", 2016, 7.9, 150, 65)
movies[nrow(movies) + 1,] <- c("everyday", "goro lovic", 2014, 4.4, 170, 40)
tail(movies)
#>                         Title         Director Year Rating
#> 998  Step Up 2: The Streets       Jon M. Chu 2008    6.2
#> 999            Search Party   Scot Armstrong 2014    5.6
#> 1000             Nine Lives Barry Sonnenfeld 2016    5.3
#> 1001              the big g       goro lovic 2015    9.9
#> 1002         luv of my life       nema lovic 2016    7.9
#> 1003               everyday       goro lovic 2014    4.4
#>      Revenue Metascore
#> 998    58.01        50
#> 999     <NA>        22
#> 1000   19.64        11
#> 1001    1000       100
#> 1002     150        65
#> 1003     170        40
```

The function `rbind()` can combine both a list or a vector to a data frame. Generally, avoid using vectors as they may change the data type of the data frame.

4.7.2.2 Adding rows using rbind()

```
# binding a list to a data frame
movies <- mov[, c(2, 5, 7, 9, 11, 12)]
movies <- rbind(movies, list("the big g", "goro lovic", 2015, 9.9, 1000, 100))
movies <- rbind(movies, list("luv of my life", "nema lovic", 2016, 7.9, 150, 65))
movies <- rbind(movies, list("everyday", "goro lovic", 2014, 4.4, 170, 40))
tail(movies)
#>                         Title         Director Year Rating
#> 998  Step Up 2: The Streets       Jon M. Chu 2008    6.2
#> 999            Search Party   Scot Armstrong 2014    5.6
#> 1000             Nine Lives Barry Sonnenfeld 2016    5.3
#> 1001              the big g       goro lovic 2015    9.9
#> 1002         luv of my life       nema lovic 2016    7.9
```

```
#> 1003                everyday       goro lovic 2014    4.4
#>      Revenue Metascore
#> 998    58.01        50
#> 999       NA        22
#> 1000   19.64        11
#> 1001 1000.00       100
#> 1002  150.00        65
#> 1003  170.00        40

movies <- mov[, c(2, 5, 7, 9, 11, 12)]
sapply(movies, class)
#>       Title    Director        Year      Rating     Revenue
#> "character" "character"   "integer"   "numeric"   "numeric"
#>   Metascore
#>   "integer"

# using a vector
movies <- rbind(movies, c("the big g", "goro lovic", 2015, 9.9, 1000, 100))
sapply(movies, class)
#>       Title    Director        Year      Rating     Revenue
#> "character" "character" "character" "character" "character"
#>   Metascore
#> "character"
```

4.7.2.3 Adding rows using do.call()

The function `do.call('rbind', dfs)` combines a list of data frames, list, and vectors. Again, avoid using vectors as they may change the data type of the data frames.

```
movies <- subset(mov, select = c(2, 5, 7, 9, 11, 12))
movies <- do.call('rbind', list(movies,
                                list("the big g", "goro lovic", 2015, 9.9, 1000, 100),
                                list("luv of my life", "nema lovic", 2016, 7.9, 150, 65
                                list("everyday", "goro lovic", 2014, 4.4, 170, 40)))
tail(movies)
#>                         Title         Director Year Rating
#> 998  Step Up 2: The Streets       Jon M. Chu 2008    6.2
#> 999            Search Party   Scot Armstrong 2014    5.6
#> 1000             Nine Lives Barry Sonnenfeld 2016    5.3
#> 1001              the big g       goro lovic 2015    9.9
#> 1002         luv of my life       nema lovic 2016    7.9
#> 1003               everyday       goro lovic 2014    4.4
#>      Revenue Metascore
#> 998    58.01        50
#> 999       NA        22
```

```
#> 1000   19.64        11
#> 1001 1000.00       100
#> 1002  150.00        65
#> 1003  170.00        40
```

4.7.3 Updating rows of data

To update a row, we simply select it and give it a new list of values. Vectors can be used also but should be avoided as they may change the data type of the data frame.

```
movies <- mov[, c(1, 2, 5, 7, 9, 11, 12)]
movies[6,]
#>   Rank          Title    Director Year Rating Revenue
#> 6    6 The Great Wall Yimou Zhang 2016    6.1   45.13
#>   Metascore
#> 6        42

# updating a row by indexing
movies[6,] <- list(6, 'I am coming home', 'goro lovic', 2020, 9.8, 850, 85)
movies[6,]
#>   Rank            Title   Director Year Rating Revenue
#> 6    6 I am coming home goro lovic 2020    9.8     850
#>   Metascore
#> 6        85

# updating a row by filtering
movies <- mov[, c(1, 2, 5, 7, 9, 11, 12)]
movies[movies$Rank == 6,] <- list(6, 'I am coming home', 'goro lovic', 2020, 9.8, 850, 85)
movies[movies$Rank == 6,]
#>   Rank            Title   Director Year Rating Revenue
#> 6    6 I am coming home goro lovic 2020    9.8     850
#>   Metascore
#> 6        85
```

4.7.4 Updating a single value

To update a single value, we select it through subsetting and assign it a new value.

```
movies <- mov[, c(1, 2, 5, 7, 9, 11, 12)]
movies[movies$Director == 'Christopher Nolan',]
#>     Rank                 Title          Director Year
#> 37    37          Interstellar Christopher Nolan 2014
#> 55    55       The Dark Knight Christopher Nolan 2008
#> 65    65          The Prestige Christopher Nolan 2006
```

```
#> 81    81            Inception Christopher Nolan 2010
#> 125  125 The Dark Knight Rises Christopher Nolan 2012
#>     Rating Revenue Metascore
#> 37     8.6  187.99        74
#> 55     9.0  533.32        82
#> 65     8.5   53.08        66
#> 81     8.8  292.57        74
#> 125    8.5  448.13        78

# changing from 'Christopher Nolan' to 'C Nolan'
movies[movies$Director == 'Christopher Nolan', 'Director'] <- 'C Nolan'
movies[c(37, 55, 65, 81, 125),]
#>     Rank                 Title Director Year Rating Revenue
#> 37    37          Interstellar  C Nolan 2014    8.6  187.99
#> 55    55       The Dark Knight  C Nolan 2008    9.0  533.32
#> 65    65          The Prestige  C Nolan 2006    8.5   53.08
#> 81    81             Inception  C Nolan 2010    8.8  292.57
#> 125  125 The Dark Knight Rises  C Nolan 2012    8.5  448.13
#>     Metascore
#> 37         74
#> 55         82
#> 65         66
#> 81         74
#> 125        78
```

4.7.5 Randomly selecting rows

To select a random sample of rows, we use the function `sample()`.

```
# selecting 10 random rows
movies <- mov[, c(2, 7, 11, 12)]
movies[sample(x = nrow(movies), size = 10), ]
#>                   Title Year Revenue Metascore
#> 535      A Quiet Passion 2016    1.08        77
#> 471     American Gangster 2007  130.13        76
#> 728       The Illusionist 2006   39.83        68
#> 789 Hotel Transylvania 2 2015  169.69        44
#> 978         Amateur Night 2016      NA        38
#> 275             Ballerina 2016      NA        NA
#> 905               RoboCop 2014   58.61        52
#> 723             Grown Ups 2010  162.00        30
#> 958          End of Watch 2012   40.98        68
#> 211          San Andreas 2015  155.18        43
```

4.7.6 Filtering rows

The function `subset()` or `[` is used to filter rows.

```
head(gapminder_cp1250[gapminder_cp1250$continent == "Europe", c("country", "gdpPercap", "continen
#>    country gdpPercap continent
#> 13 Albania  1601.056    Europe
#> 14 Albania  1942.284    Europe
#> 15 Albania  2312.889    Europe
#> 16 Albania  2760.197    Europe
#> 17 Albania  3313.422    Europe
#> 18 Albania  3533.004    Europe
gapminder_cp1250[gapminder_cp1250$continent == "Europe" & gapminder_cp1250$gdpPercap > 2000 & gap
#>                     country gdpPercap continent
#> 15                  Albania  2312.889    Europe
#> 16                  Albania  2760.197    Europe
#> 17                  Albania  3313.422    Europe
#> 18                  Albania  3533.004    Europe
#> 19                  Albania  3630.881    Europe
#> 20                  Albania  3738.933    Europe
#> 21                  Albania  2497.438    Europe
#> 22                  Albania  3193.055    Europe
#> 148  Bosnia and Herzegovina  2172.352    Europe
#> 149  Bosnia and Herzegovina  2860.170    Europe
#> 150  Bosnia and Herzegovina  3528.481    Europe
#> 153  Bosnia and Herzegovina  2546.781    Europe
#> 181                Bulgaria  2444.287    Europe
#> 182                Bulgaria  3008.671    Europe
#> 373                 Croatia  3119.237    Europe
#> 589                  Greece  3530.690    Europe
#> 1009             Montenegro  2647.586    Europe
#> 1010             Montenegro  3682.260    Europe
#> 1237               Portugal  3068.320    Europe
#> 1238               Portugal  3774.572    Europe
#> 1273                Romania  3144.613    Europe
#> 1274                Romania  3943.370    Europe
#> 1333                 Serbia  3581.459    Europe
#> 1417                  Spain  3834.035    Europe
#> 1574                 Turkey  2218.754    Europe
#> 1575                 Turkey  2322.870    Europe
#> 1576                 Turkey  2826.356    Europe
#> 1577                 Turkey  3450.696    Europe
subset(gapminder_cp1250,
       subset =  continent == "Europe" & gdpPercap > 2000 & gdpPercap < 4000,
       select = c("country", "gdpPercap", "continent"))
#>                     country gdpPercap continent
```

```
#> 15                 Albania  2312.889    Europe
#> 16                 Albania  2760.197    Europe
#> 17                 Albania  3313.422    Europe
#> 18                 Albania  3533.004    Europe
#> 19                 Albania  3630.881    Europe
#> 20                 Albania  3738.933    Europe
#> 21                 Albania  2497.438    Europe
#> 22                 Albania  3193.055    Europe
#> 148  Bosnia and Herzegovina  2172.352    Europe
#> 149  Bosnia and Herzegovina  2860.170    Europe
#> 150  Bosnia and Herzegovina  3528.481    Europe
#> 153  Bosnia and Herzegovina  2546.781    Europe
#> 181                Bulgaria  2444.287    Europe
#> 182                Bulgaria  3008.671    Europe
#> 373                 Croatia  3119.237    Europe
#> 589                  Greece  3530.690    Europe
#> 1009             Montenegro  2647.586    Europe
#> 1010             Montenegro  3682.260    Europe
#> 1237               Portugal  3068.320    Europe
#> 1238               Portugal  3774.572    Europe
#> 1273                Romania  3144.613    Europe
#> 1274                Romania  3943.370    Europe
#> 1333                 Serbia  3581.459    Europe
#> 1417                  Spain  3834.035    Europe
#> 1574                 Turkey  2218.754    Europe
#> 1575                 Turkey  2322.870    Europe
#> 1576                 Turkey  2826.356    Europe
#> 1577                 Turkey  3450.696    Europe
```

4.7.7 Deleting rows

There is no special function to delete rows, but they can be filtered out using `[`.

```
movies_without_first10 <- movies[11:nrow(movies), ]
nrow(movies)
#> [1] 1000
nrow(movies_without_first10)
#> [1] 990
```

4.8 SQL like joins

At the most basic level there are four types of SQL joins:

- Inner join: which returns only rows matched in both data frames
- Left join (left outer join): which returns all rows found in the left data

frame irrespective of whether they are matched to rows in the right data frame. If rows do not match values in the right data frames, NA values are returned instead.

- Right join (right outer join): which is the reverse of the left join, that is it returns all rows found on the right data frame irrespective of whether they are matched on the left data frame.
- Outer join (full outer join): returns all rows from both data frames irrespective of whether they are matched or not

4.8.1 Inner join

```
# preparing data
employees <- data.frame(
  name = c('john', 'mary', 'david', 'paul', 'susan', 'cynthia', 'Joss', 'dennis'),
  age = c(45, 55, 35, 58, 40, 30, 39, 25),
  gender = c('m', 'f', 'm', 'm', 'f', 'f', 'm', 'm'),
  salary =c(40000, 50000, 35000, 25000, 48000, 32000, 20000, 45000),
  department = c('commercial', 'production', NA, 'human resources',
                 'commercial', 'commercial', 'production', NA))
employees
#>      name age gender salary      department
#> 1    john  45      m  40000      commercial
#> 2    mary  55      f  50000      production
#> 3   david  35      m  35000            <NA>
#> 4    paul  58      m  25000 human resources
#> 5   susan  40      f  48000      commercial
#> 6 cynthia  30      f  32000      commercial
#> 7    Joss  39      m  20000      production
#> 8  dennis  25      m  45000            <NA>
departments <- data.frame(
  department = c('commercial', 'human resources', 'production', 'finance', 'maintenance'),
  location = c('washington', 'london', 'paris', 'dubai', 'dublin'))
departments
#>        department   location
#> 1      commercial washington
#> 2 human resources     london
#> 3      production      paris
#> 4         finance      dubai
#> 5     maintenance     dublin

# returns only rows that are matched in both data frames
merge(employees, departments, by = "department")
#>        department    name age gender salary   location
#> 1      commercial    john  45      m  40000 washington
#> 2      commercial   susan  40      f  48000 washington
```

```
#> 3      commercial cynthia  30      f  32000 washington
#> 4 human resources    paul  58      m  25000     london
#> 5      production    mary  55      f  50000      paris
#> 6      production    Joss  39      m  20000      paris
```

4.8.2 Left join

To perform a left join, the argument `all.x = TRUE` is used.

```
# returns all the values of the left data frame
merge(employees, departments, by = "department", all.x = TRUE)
#>        department    name age gender salary   location
#> 1      commercial    john  45      m  40000 washington
#> 2      commercial   susan  40      f  48000 washington
#> 3      commercial cynthia  30      f  32000 washington
#> 4 human resources    paul  58      m  25000     london
#> 5      production    mary  55      f  50000      paris
#> 6      production    Joss  39      m  20000      paris
#> 7            <NA>   david  35      m  35000       <NA>
#> 8            <NA>  dennis  25      m  45000       <NA>
```

4.8.3 Right join

To perform a right join, the argument `all.y = TRUE` is used.

```
# returns all the values of the right table
merge(employees, departments, by = "department", all.y = TRUE)
#>        department    name age gender salary   location
#> 1      commercial    john  45      m  40000 washington
#> 2      commercial   susan  40      f  48000 washington
#> 3      commercial cynthia  30      f  32000 washington
#> 4         finance    <NA>  NA   <NA>     NA      dubai
#> 5 human resources    paul  58      m  25000     london
#> 6     maintenance    <NA>  NA   <NA>     NA     dublin
#> 7      production    mary  55      f  50000      paris
#> 8      production    Joss  39      m  20000      paris

# reversing the tables in the right join produces the same results as the left join
merge(departments, employees , by = "department", all.y = TRUE)
#>        department   location    name age gender salary
#> 1      commercial washington    john  45      m  40000
#> 2      commercial washington   susan  40      f  48000
#> 3      commercial washington cynthia  30      f  32000
#> 4 human resources     london    paul  58      m  25000
#> 5      production      paris    mary  55      f  50000
#> 6      production      paris    Joss  39      m  20000
```

```
#> 7             <NA>       <NA>   david  35      m  35000
#> 8             <NA>       <NA>  dennis  25      m  45000
```

4.8.4 Full outer join

To perform a full join, the argument `all = TRUE` is used.

```
# returns all rows
merge(employees, departments, by = "department", all = TRUE)
#>          department    name age gender salary   location
#> 1        commercial    john  45      m  40000 washington
#> 2        commercial   susan  40      f  48000 washington
#> 3        commercial cynthia  30      f  32000 washington
#> 4           finance    <NA>  NA   <NA>     NA      dubai
#> 5   human resources    paul  58      m  25000     london
#> 6       maintenance    <NA>  NA   <NA>     NA     dublin
#> 7        production    mary  55      f  50000      paris
#> 8        production    Joss  39      m  20000      paris
#> 9              <NA>   david  35      m  35000       <NA>
#> 10             <NA>  dennis  25      m  45000       <NA>
```

4.8.5 Joining data frames with different column names

The arguments `by.x=` and `by.y=` are used to declare the joining column(s) for the left and right data frames, respectively.

```
# recreating the employee table
employees <- data.frame(
  name = c('john', 'mary', 'david', 'paul', 'susan', 'cynthia', 'Joss', 'dennis'),
  age = c(45, 55, 35, 58, 40, 30, 39, 25),
  gender = c('m', 'f', 'm', 'm', 'f', 'f', 'm', 'm'),
  salary =c(40000, 50000, 35000, 25000, 48000, 32000, 20000, 45000),
  dep_name = c('commercial', 'production', NA, 'human resources', 'commercial',
               'commercial', 'production', NA))
head(employees, 2)
#>   name age gender salary   dep_name
#> 1 john  45      m  40000 commercial
#> 2 mary  55      f  50000 production
head(departments, 2)
#>        department   location
#> 1      commercial washington
#> 2 human resources     london

# joining on columns with different names
merge(employees, departments, by.x = 'dep_name', by.y = 'department')
#>          dep_name    name age gender salary   location
```

```
#> 1       commercial    john  45      m  40000 washington
#> 2       commercial   susan  40      f  48000 washington
#> 3       commercial cynthia  30      f  32000 washington
#> 4 human resources    paul  58      m  25000     london
#> 5      production    mary  55      f  50000      paris
#> 6      production    Joss  39      m  20000      paris
```

4.8.6 Joining data frames on one more than one joining column

If both data frames contain two or more columns with the same name, `merge()` will try performing the join using those column names.

```
# recreating the employees table
employees <- data.frame(
  name = c('john', 'mary', 'david', 'paul', 'susan', 'cynthia', 'Joss', 'dennis'),
  age = c(45, 55, 35, 58, 40, 30, 39, 25),
  gender = c('m', 'f', 'm', 'm', 'f', 'f', 'm', 'm'),
  salary =c(40000, 50000, 35000, 25000, 48000, 32000, 20000, 45000),
  department = c('commercial', 'production', NA, 'human resources', 'commercial',
                 'commercial', 'production', NA),
  subdepartment = c('marketing', 'production', NA, 'human resources', 'sales', 'sales'
                    'production', NA))
head(employees, 2)
#>   name age gender salary department subdepartment
#> 1 john  45      m  40000 commercial     marketing
#> 2 mary  55      f  50000 production    production

# creating the departments? table
departments <- data.frame(
  department = c('commercial', 'commercial', 'human resources', 'production', 'finance
                 'finance', 'maintenance'),
  subdepartment = c('marketing', 'sales', 'human resources', 'production', 'finance',
                    'accounting', 'maintenance'),
  location = c('washington', 'washington', 'london', 'paris', 'dubai', 'dubai', 'dubli
)
head(departments, 2)
#>   department subdepartment   location
#> 1 commercial     marketing washington
#> 2 commercial         sales washington

# because they both contain the same name, the join is performed automatically
merge(employees, departments)
#>        department   subdepartment    name age gender salary
#> 1      commercial       marketing    john  45      m  40000
```

```
#> 2       commercial            sales   susan  40      f  48000
#> 3       commercial            sales cynthia  30      f  32000
#> 4 human resources human resources    paul  58      m  25000
#> 5       production      production    mary  55      f  50000
#> 6       production      production    Joss  39      m  20000
#>     location
#> 1 washington
#> 2 washington
#> 3 washington
#> 4     london
#> 5      paris
#> 6      paris
```

If the data frames had columns of different names to join on, we would have used the arguments `by.x=` and `by.y=` to specify them as below.

```
# specifying joining columns
merge(employees, departments,
      by.x = c('department', 'subdepartment'),
      by.y =c('department', 'subdepartment'))
#>        department   subdepartment    name age gender salary
#> 1       commercial       marketing    john  45      m  40000
#> 2       commercial            sales   susan  40      f  48000
#> 3       commercial            sales cynthia  30      f  32000
#> 4 human resources human resources    paul  58      m  25000
#> 5       production      production    mary  55      f  50000
#> 6       production      production    Joss  39      m  20000
#>     location
#> 1 washington
#> 2 washington
#> 3 washington
#> 4     london
#> 5      paris
#> 6      paris
```

4.9 Aggregating and grouping data

The function `aggregate()` groups a data frame by a specific column value and performs summarization (sum, mean, median, length, min, max, etc.) based on those groups. It does a split-apply-combine, that is splitting a data frame by groups (category) after which it applies a calculation on each group and finally combines the results back together to create a single data frame which is presented as output.

```
# preparing data
gapminder_xlsx_2007 <- gapminder_xlsx[gapminder_xlsx$year == 2007, ]
head(gapminder_xlsx_2007)
#>        country continent year lifeExp      pop  gdpPercap
#> 12 Afghanistan      Asia 2007  43.828 31889923   974.5803
#> 24     Albania    Europe 2007  76.423  3600523  5937.0295
#> 36     Algeria    Africa 2007  72.301 33333216  6223.3675
#> 48      Angola    Africa 2007  42.731 12420476  4797.2313
#> 60   Argentina  Americas 2007  75.320 40301927 12779.3796
#> 72   Australia   Oceania 2007  81.235 20434176 34435.3674
#>    country_hun continent_hun
#> 12 Afganisztán         Ázsia
#> 24      Albánia        Európa
#> 36      Algéria        Afrika
#> 48       Angola        Afrika
#> 60   Argentína       Amerika
#> 72  Ausztrália       Óceánia

# population by continent
aggregate(pop ~ continent, gapminder_xlsx_2007, sum)
#>   continent        pop
#> 1    Africa  929539692
#> 2  Americas  898871184
#> 3      Asia 3811953827
#> 4    Europe  586098529
#> 5   Oceania   24549947
aggregate(pop ~ continent, gapminder_xlsx_2007, mean)
#>   continent       pop
#> 1    Africa  17875763
#> 2  Americas  35954847
#> 3      Asia 115513752
#> 4    Europe  19536618
#> 5   Oceania  12274974
```

The `aggregate()` function above, groups the data frame `gapminder_xlsx_2007` by continent, after which it applies sum to each group.

Rather than filtering the data before passing it to the `aggregate()` function, we can filter the data directly inside `aggregate()` using the `subset=` argument.

```
# filtering with the subset argument
aggregate(pop ~ continent, gapminder_xlsx,
          subset = year == 2007,
          sum)
#>   continent        pop
#> 1    Africa  929539692
```

```
#> 2  Americas  898871184
#> 3      Asia 3811953827
#> 4    Europe  586098529
#> 5   Oceania   24549947
```

The + sign is used to group by more than one categorical column.

```
# pop by continent and year
aggregate(pop ~ continent + year,
          gapminder_xlsx,
          subset = year %in% c(1987, 2007),
          sum)
#>    continent year        pop
#> 1     Africa 1987  574834110
#> 2   Americas 1987  682753971
#> 3       Asia 1987 2871220762
#> 4     Europe 1987  543094160
#> 5    Oceania 1987   19574415
#> 6     Africa 2007  929539692
#> 7   Americas 2007  898871184
#> 8       Asia 2007 3811953827
#> 9     Europe 2007  586098529
#> 10   Oceania 2007   24549947
# using mean
aggregate(pop ~ continent + year,
          gapminder_xlsx,
          subset = year %in% c(1987, 2007),
          mean)
#>    continent year       pop
#> 1     Africa 1987  11054502
#> 2   Americas 1987  27310159
#> 3       Asia 1987  87006690
#> 4     Europe 1987  18103139
#> 5    Oceania 1987   9787208
#> 6     Africa 2007  17875763
#> 7   Americas 2007  35954847
#> 8       Asia 2007 115513752
#> 9     Europe 2007  19536618
#> 10   Oceania 2007  12274974
```

The function `cbind()` is used to aggregate on multiple columns, the only problem is that only one summarisation function can be used.

```
# aggregating on two numeric columns (lifeExp and gdpPercap)
aggregate(cbind(lifeExp, gdpPercap) ~ continent + year,
          gapminder_xlsx,
          subset = year %in% c(1987, 2007),
```

```
          mean)
#>    continent year  lifeExp gdpPercap
#> 1     Africa 1987 53.34479  2282.669
#> 2   Americas 1987 68.09072  7793.400
#> 3       Asia 1987 64.85118  7608.227
#> 4     Europe 1987 73.64217 17214.311
#> 5    Oceania 1987 75.32000 20448.040
#> 6     Africa 2007 54.80604  3089.033
#> 7   Americas 2007 73.60812 11003.032
#> 8       Asia 2007 70.72848 12473.027
#> 9     Europe 2007 77.64860 25054.482
#> 10   Oceania 2007 80.71950 29810.188
# rounding with customized function
aggregate(cbind(lifeExp, gdpPercap) ~ continent + year,
          gapminder_xlsx,
          subset = year %in% c(1987, 2007),
          function(x){round(mean(x), 1)})
#>    continent year lifeExp gdpPercap
#> 1     Africa 1987    53.3    2282.7
#> 2   Americas 1987    68.1    7793.4
#> 3       Asia 1987    64.9    7608.2
#> 4     Europe 1987    73.6   17214.3
#> 5    Oceania 1987    75.3   20448.0
#> 6     Africa 2007    54.8    3089.0
#> 7   Americas 2007    73.6   11003.0
#> 8       Asia 2007    70.7   12473.0
#> 9     Europe 2007    77.6   25054.5
#> 10   Oceania 2007    80.7   29810.2
```

4.10 Pivoting and unpivoting data

Tabular data exist in two forms: long and wide. The wide form is ideal for reporting while the long form is ideal for the computer. Most often, when performing data analysis, data in the wide form has to be converted to the long form (unpivoting) while when preparing reports, data in the long has to be converted to the wide form (pivoting).

wide data

Person	Age	Weight	Height
Bob	32	168	180
Alice	24	150	175
Steve	64	144	165

long data

Person	Variable	Value
Bob	Age	32
Bob	Weight	168
Bob	Height	180
Alice	Age	24
Alice	Weight	150
Alice	Height	175
Steve	Age	64
Steve	Weight	144
Steve	Height	165

4.10.1 Pivoting

Pivoting converts data frame rows to columns.

4.10.1.1 Pivoting using the reshape package

The **reshape** package is a package created for restructuring and aggregating data using just two functions: `melt()` and `cast()`.

The function `cast()` pivots data while `melt()` unpivots data.

```
# preparing long data
dt <- aggregate(cbind(lifeExp, gdpPercap) ~ continent + year,
                gapminder_xlsx,
                subset = year >= 1987,
                mean)
head(dt,3)
#>   continent year  lifeExp gdpPercap
#> 1    Africa 1987 53.34479  2282.669
#> 2  Americas 1987 68.09072  7793.400
#> 3      Asia 1987 64.85118  7608.227
tail(dt,3)
#>    continent year  lifeExp gdpPercap
#> 23      Asia 2007 70.72848  12473.03
#> 24    Europe 2007 77.64860  25054.48
#> 25   Oceania 2007 80.71950  29810.19

library(reshape)
# converting from long to wide
cast(data = dt,
     formula = continent ~ year,
     value = 'lifeExp')
#>   continent     1987     1992     1997     2002     2007
```

```
#> 1    Africa 53.34479 53.62958 53.59827 53.32523 54.80604
#> 2  Americas 68.09072 69.56836 71.15048 72.42204 73.60812
#> 3      Asia 64.85118 66.53721 68.02052 69.23388 70.72848
#> 4    Europe 73.64217 74.44010 75.50517 76.70060 77.64860
#> 5   Oceania 75.32000 76.94500 78.19000 79.74000 80.71950
```

The function `cast()` can perform aggregation through the `fun.aggregate=` argument and filtering through the subset argument.

```
# summarization
cast(data = gapminder_xlsx_2007,
     formula = continent ~ year,
     value = 'pop',
     fun.aggregate = sum)
#>   continent       2007
#> 1    Africa  929539692
#> 2  Americas  898871184
#> 3      Asia 3811953827
#> 4    Europe  586098529
#> 5   Oceania   24549947

# filtering with subset
cast(data = gapminder_xlsx,
     continent ~ year,
     subset = year >= 1987,
     value = 'lifeExp',
     fun.aggregate = mean)
#>   continent     1987     1992     1997     2002     2007
#> 1    Africa 53.34479 53.62958 53.59827 53.32523 54.80604
#> 2  Americas 68.09072 69.56836 71.15048 72.42204 73.60812
#> 3      Asia 64.85118 66.53721 68.02052 69.23388 70.72848
#> 4    Europe 73.64217 74.44010 75.50517 76.70060 77.64860
#> 5   Oceania 75.32000 76.94500 78.19000 79.74000 80.71950

# rounding numbers
cast(data = gapminder_xlsx,
     continent ~ year,
     subset = year >= 1987,
     value = 'lifeExp',
     fun.aggregate = function(x)round(mean(x), 1))
#>   continent 1987 1992 1997 2002 2007
#> 1    Africa 53.3 53.6 53.6 53.3 54.8
#> 2  Americas 68.1 69.6 71.2 72.4 73.6
#> 3      Asia 64.9 66.5 68.0 69.2 70.7
#> 4    Europe 73.6 74.4 75.5 76.7 77.6
#> 5   Oceania 75.3 76.9 78.2 79.7 80.7
```

```
# population by year by continent
cast(data = gapminder_xlsx,
     year ~ continent,
     subset = year >= 1987,
     value = 'pop',
     fun.aggregate = sum)
#>   year    Africa  Americas       Asia    Europe  Oceania
#> 1 1987 574834110 682753971 2871220762 543094160 19574415
#> 2 1992 659081517 739274104 3133292191 558142797 20919651
#> 3 1997 743832984 796900410 3383285500 568944148 22241430
#> 4 2002 833723916 849772762 3601802203 578223869 23454829
#> 5 2007 929539692 898871184 3811953827 586098529 24549947
```

4.10.1.2 Pivoting using the reshape2 package

The **reshape2** package is a reboot of the reshape package.

The function `acast()` and `dcast()` are used to pivot data with the former returning a matrix while the later a data frame.

```
dt_wide <- reshape2::acast(data = dt,
                           formula = continent ~ year,
                           value.var = 'lifeExp')
dt_wide
#>              1987     1992     1997     2002     2007
#> Africa   53.34479 53.62958 53.59827 53.32523 54.80604
#> Americas 68.09072 69.56836 71.15048 72.42204 73.60812
#> Asia     64.85118 66.53721 68.02052 69.23388 70.72848
#> Europe   73.64217 74.44010 75.50517 76.70060 77.64860
#> Oceania  75.32000 76.94500 78.19000 79.74000 80.71950
class(dt_wide)
#> [1] "matrix" "array"

dt_wide <- reshape2::dcast(data = dt,
                           formula = continent ~ year,
                           value.var = 'lifeExp')
dt_wide
#>   continent     1987     1992     1997     2002     2007
#> 1    Africa 53.34479 53.62958 53.59827 53.32523 54.80604
#> 2  Americas 68.09072 69.56836 71.15048 72.42204 73.60812
#> 3      Asia 64.85118 66.53721 68.02052 69.23388 70.72848
#> 4    Europe 73.64217 74.44010 75.50517 76.70060 77.64860
#> 5   Oceania 75.32000 76.94500 78.19000 79.74000 80.71950
class(dt_wide)
#> [1] "data.frame"
```

```
# filtering by year
reshape2::dcast(data = gapminder_xlsx[gapminder_xlsx$year >= 1987,],
                formula = continent ~ year,
                value.var = 'lifeExp',
                fun.aggregate = function(x)round(mean(x), 1))
#>   continent 1987 1992 1997 2002 2007
#> 1    Africa 53.3 53.6 53.6 53.3 54.8
#> 2  Americas 68.1 69.6 71.2 72.4 73.6
#> 3      Asia 64.9 66.5 68.0 69.2 70.7
#> 4    Europe 73.6 74.4 75.5 76.7 77.6
#> 5   Oceania 75.3 76.9 78.2 79.7 80.7
```

4.10.2 Unpivoting

Unpivoting converts data frame columns to rows.

The function `melt()` is used to unpivot data. It accepts the following:

- `id.vars=`: columns not to be moved
- `measure.vars=`: columns to move to rows

but can guess both by default.

It is the same function name for **reshape** and **reshape2**.

```
dt_long <- melt(dt_wide)
head(dt_long)
#>   continent variable    value
#> 1    Africa     1987 53.34479
#> 2  Americas     1987 68.09072
#> 3      Asia     1987 64.85118
#> 4    Europe     1987 73.64217
#> 5   Oceania     1987 75.32000
#> 6    Africa     1992 53.62958

dt_long <- reshape2::melt(dt_wide)
head(dt_long)
#>   continent variable    value
#> 1    Africa     1987 53.34479
#> 2  Americas     1987 68.09072
#> 3      Asia     1987 64.85118
#> 4    Europe     1987 73.64217
#> 5   Oceania     1987 75.32000
#> 6    Africa     1992 53.62958
```

With the argument `measure.vars=`, we can filter the data frame.

```
# adding a variable name and filtering data
dt_long <- melt(dt_wide,
                id.vars = 'continent',
                variable_name = 'Year',
                measure.vars = c('1997', '2002', '2007'))
head(dt_long)
#>   continent Year    value
#> 1    Africa 1997 53.59827
#> 2  Americas 1997 71.15048
#> 3      Asia 1997 68.02052
#> 4    Europe 1997 75.50517
#> 5   Oceania 1997 78.19000
#> 6    Africa 2002 53.32523

# adding value, variable name, and filtering data
dt_long <- reshape2::melt(dt_wide,
                          id.vars = 'continent',
                          variable.name = 'Year',
                          value.name = 'lifeExp',
                          measure.vars = c('1997', '2002', '2007'))
head(dt_long)
#>   continent Year  lifeExp
#> 1    Africa 1997 53.59827
#> 2  Americas 1997 71.15048
#> 3      Asia 1997 68.02052
#> 4    Europe 1997 75.50517
#> 5   Oceania 1997 78.19000
#> 6    Africa 2002 53.32523
```

4.11 Detecting and dealing with missing values

The functions `anyNA()` and `is.na()` are used to check for `NA` values and return `TRUE` for `NA` value and `FALSE` for non-NA value. While the former checks if an object contains any missing value, the latter checks for missing values within an object.

```
movies <- mov[, c(2,7,11,12)]
head(movies)
#>                     Title Year Revenue Metascore
#> 1 Guardians of the Galaxy 2014  333.13        76
#> 2              Prometheus 2012  126.46        65
#> 3                   Split 2016  138.12        62
#> 4                    Sing 2016  270.32        59
#> 5           Suicide Squad 2016  325.02        40
#> 6          The Great Wall 2016   45.13        42
```

```
# checking if an object contains any NA
anyNA(NA)
#> [1] TRUE
anyNA(list(1, 3, 5, NA))
#> [1] TRUE
anyNA(c(1, 3, 5, NA))
#> [1] TRUE
# checking if data frame contains any NA values
anyNA(movies)
#> [1] TRUE
apply(movies, 2, anyNA)
#>     Title      Year   Revenue Metascore
#>     FALSE     FALSE      TRUE      TRUE
# checking for NA values within an object
is.na(NA)
#> [1] TRUE
is.na(list(1, 3, 5, NA))
#> [1] FALSE FALSE FALSE  TRUE
is.na(c(1, 3, 5, NA))
#> [1] FALSE FALSE FALSE  TRUE
head(is.na(movies))
#>      Title  Year Revenue Metascore
#> [1,] FALSE FALSE   FALSE     FALSE
#> [2,] FALSE FALSE   FALSE     FALSE
#> [3,] FALSE FALSE   FALSE     FALSE
#> [4,] FALSE FALSE   FALSE     FALSE
#> [5,] FALSE FALSE   FALSE     FALSE
#> [6,] FALSE FALSE   FALSE     FALSE
```

Since logical can be added, with `FALSE` $= 0$ and `TRUE` $= 1$, the results of `is.na()` can be added to determine the number of `NA` values in the dataset.

To get the total number of `NA` values by columns, the function `colSums()` is used instead as it does addition by columns rather than the whole data frame.

```
# number of na values in a dataset
sum(is.na(movies))
#> [1] 192

# number of na values in each column
colSums(is.na(movies))
#>     Title      Year   Revenue Metascore
#>         0         0       128        64
```

To get the number of non-NA values within each column, we simply reverse the results of `is.na()` with the not operator (!) or subtract from the total number of rows in the data frame.

```
# number of non-NA values within each column
colSums(!is.na(movies))
#>     Title      Year   Revenue Metascore
#>      1000      1000       872       936
nrow(movies) - colSums(is.na(movies))
#>     Title      Year   Revenue Metascore
#>      1000      1000       872       936
```

To get the number of rows containing non-NA values, we use the function `complete.cases()` which returns `TRUE` for rows without `NA` values and `FALSE` for rows with `NA` values. Summing its result gives us the number of rows without `NA` values (complete cases). We can equally reverse `complete.cases()` with the not operator to obtain the number of rows with NA values or subtract from the total number of rows.

```
# number of rows without NA values
sum(complete.cases(movies))
#> [1] 838
# number of rows with one or more NA values
sum(!complete.cases(movies))
#> [1] 162
nrow(movies) - sum(complete.cases(movies))
#> [1] 162
```

Using `complete.cases()`, we can filter out either rows with NA values or rows without `NA` values.

```
# selecting rows without NA
no_na_movies <- movies[complete.cases(movies), ]
head(no_na_movies, 10)
#>                                      Title Year Revenue
#> 1                  Guardians of the Galaxy 2014  333.13
#> 2                               Prometheus 2012  126.46
#> 3                                    Split 2016  138.12
#> 4                                     Sing 2016  270.32
#> 5                            Suicide Squad 2016  325.02
#> 6                           The Great Wall 2016   45.13
#> 7                               La La Land 2016  151.06
#> 9                       The Lost City of Z 2016    8.01
#> 10                              Passengers 2016  100.01
#> 11 Fantastic Beasts and Where to Find Them 2016  234.02
#>    Metascore
#> 1         76
#> 2         65
#> 3         62
#> 4         59
```

```
#> 5         40
#> 6         42
#> 7         93
#> 9         78
#> 10        41
#> 11        66

# selecting rows with NA
na_movies <- movies[!complete.cases(movies), ]
head(na_movies, 10)
#>                      Title Year Revenue Metascore
#> 8                 Mindhorn 2016      NA        71
#> 23          Hounds of Love 2016      NA        72
#> 26         Paris pieds nus 2016      NA        NA
#> 27 Bahubali: The Beginning 2015    6.50        NA
#> 28              Dead Awake 2016    0.01        NA
#> 40              5- 25- 77 2007      NA        NA
#> 43 Don't Fuck in the Woods 2016      NA        NA
#> 48                  Fallen 2016      NA        NA
#> 50           The Last Face 2016      NA        16
#> 62 The Autopsy of Jane Doe 2016      NA        65
```

4.12 Detecting and dealing with outliers

4.12.1 What is an outlier?

Outliers also known as anomalies are values that deviate extremely from other values within the same group of data. They occur because of errors committed while collecting or recording data, performing calculations or are just data points with extreme values.

4.12.2 Identifying outlier

4.12.2.1 Using summary statistics

The first step in outlier detection is to look at summary statistics, most especially the minimum, maximum, median, and mean. For example, with a dataset of people's ages, if the maximum is 200 or the minimum is negative, then there is a problem.

```
gapminder_xlsx_2007 <- gapminder_xlsx[gapminder_xlsx$year == 2007, ]
head(gapminder_xlsx_2007)
#>        country continent year lifeExp      pop  gdpPercap
#> 12 Afghanistan      Asia 2007  43.828 31889923   974.5803
#> 24     Albania    Europe 2007  76.423  3600523  5937.0295
#> 36     Algeria    Africa 2007  72.301 33333216  6223.3675
```

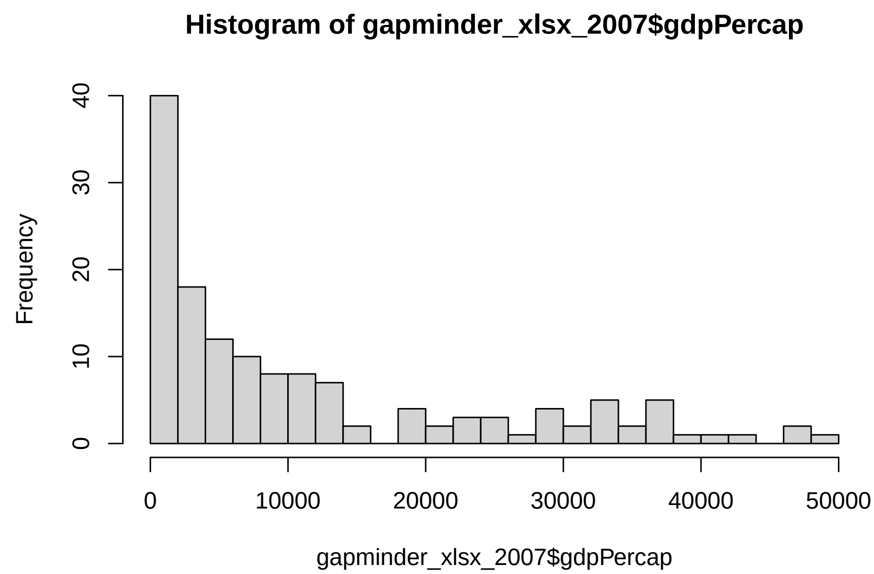
```
#> 48      Angola    Africa 2007  42.731 12420476  4797.2313
#> 60   Argentina  Americas 2007  75.320 40301927 12779.3796
#> 72   Australia   Oceania 2007  81.235 20434176 34435.3674
#>    country_hun continent_hun
#> 12 Afganisztán         Ázsia
#> 24     Albánia        Európa
#> 36     Algéria        Afrika
#> 48      Angola        Afrika
#> 60   Argentína       Amerika
#> 72  Ausztrália       Óceánia
summary(gapminder_xlsx_2007$pop/1e6)
#>      Min.   1st Qu.    Median      Mean   3rd Qu.      Max.
#>    0.1996    4.5080   10.5175   44.0212   31.2100 1318.6831
```

From the above, we see that the median and mean are 10 million and 44 million respectively while the maximum value is 1.3 billion. This tells us that there are some outliers since the maximum value varies greatly from the centre of the data.

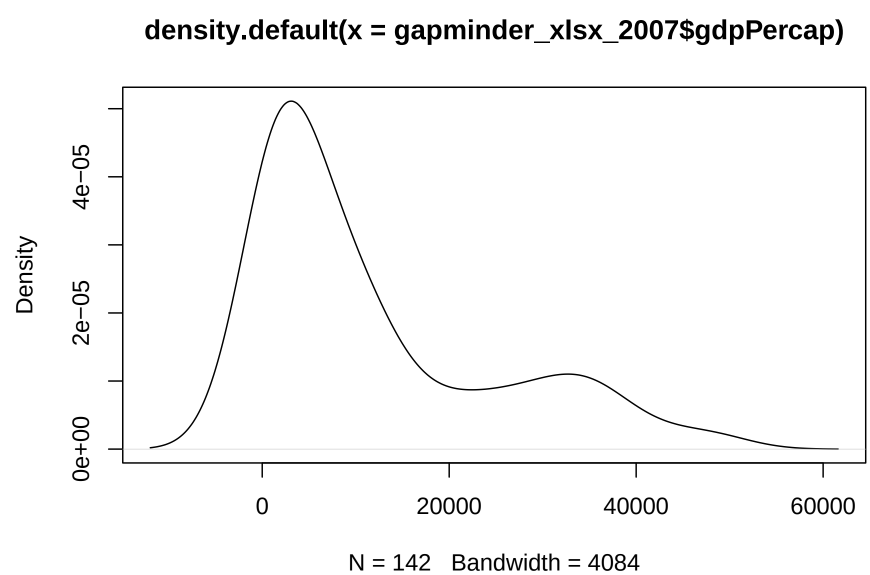
4.12.2.2 Using plots

Outliers are identified using univariate plots such as histogram, density plot and boxplot.

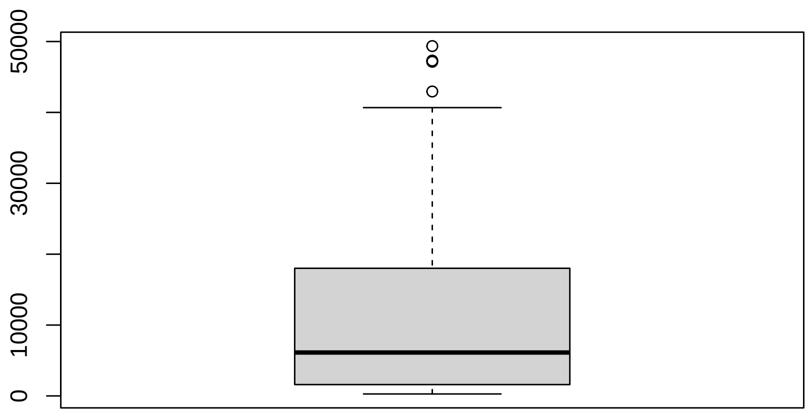
```
# plotting variable using histogram
hist(gapminder_xlsx_2007$gdpPercap, breaks = 18)
```




```
# density plot
plot(density(gapminder_xlsx_2007$gdpPercap))
```



```
# boxplot of population
boxplot(gapminder_xlsx_2007$gdpPercap)
```



Of the above data visualizations, the boxplot is the most relevant as it shows both the spread of data and outliers. The boxplot reveals the following:

- minimum value,
- first quantile (Q1),
- median (second quantile),
- third quantile (Q3),
- maximum value excluding outliers and
- outliers.

The difference between Q3 and Q1 is known as the Interquartile Range (IQR). The outliers within the box plot are calculated as any value that falls beyond 1.5 * IQR.

The function `boxplot.stats()` computes the data that is used to draw the box plot. Using this function, we can get our outliers.

```
boxplot.stats(gapminder_xlsx_2007$gdpPercap)
#> $stats
#> [1]   277.5519  1598.4351  6124.3711 18008.9444 40675.9964
#>
#> $n
#> [1] 142
```

```
#>
#> $conf
#> [1] 3948.491 8300.251
#>
#> $out
#> [1] 47306.99 49357.19 47143.18 42951.65
```

The first element returned is the summary statistic as was calculated with `summary()`.

```
boxplot.stats(gapminder_xlsx_2007$gdpPercap)$stats
#> [1]   277.5519  1598.4351  6124.3711 18008.9444 40675.9964
summary(gapminder_xlsx_2007$gdpPercap)
#>    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
#>   277.6  1624.8  6124.4 11680.1 18008.8 49357.2
```

The last element returned are the outliers.

```
boxplot.stats(gapminder_xlsx_2007$gdpPercap)$out
#> [1] 47306.99 49357.19 47143.18 42951.65
```

Recall outliers are calculated as 1.5 * IQR, this can be changed using the argument coef. By default, it is set to 1.5 but can be changed as need be.

```
# changing coef
boxplot.stats(gapminder_xlsx_2007$gdpPercap, coef = 0.8)$out
#>  [1] 34435.37 36126.49 33692.61 36319.24 35278.42 33207.08
#>  [7] 32170.37 39724.98 36180.79 40676.00 31656.07 47306.99
#> [13] 36797.93 49357.19 47143.18 33859.75 37506.42 33203.26
#> [19] 42951.65
boxplot.stats(gapminder_xlsx_2007$gdpPercap, coef = 1)$out
#>  [1] 34435.37 36126.49 36319.24 35278.42 39724.98 36180.79
#>  [7] 40676.00 47306.99 36797.93 49357.19 47143.18 37506.42
#> [13] 42951.65
boxplot.stats(gapminder_xlsx_2007$gdpPercap, coef = 1.2)$out
#> [1] 39724.98 40676.00 47306.99 49357.19 47143.18 42951.65

# selecting outliers
gapminder_xlsx_2007[gapminder_xlsx_2007$gdpPercap >= min(boxplot.stats(gapminder_xlsx_2
#>            country continent year lifeExp       pop
#> 864         Kuwait      Asia 2007  77.588   2505559
#> 1152        Norway    Europe 2007  80.196   4627926
#> 1368     Singapore      Asia 2007  79.972   4553009
#> 1620 United States  Americas 2007  78.242 301139947
#>      gdpPercap      country_hun continent_hun
#> 864   47306.99           Kuvait         Ázsia
#> 1152  49357.19         Norvégia        Európa
```

```
#> 1368  47143.18        Szingapúr        Ázsia
#> 1620  42951.65 Egyesült Államok       Amerika
```

4.13 Dealing with duplicate values

4.13.1 Determining duplicate values

The function `duplicated()` determines which elements are duplicates in a vector or data frame while the function `anyDuplicated()` returns the index position of the first duplicate.

```
# checking for duplicates
duplicated(1:10)
#>  [1] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
#> [10] FALSE

duplicated(c(2, 1, 3, 6, 2, 4, 7, 0, 3, 3, 2, 2, 8, 4, 0))
#>  [1] FALSE FALSE FALSE FALSE  TRUE FALSE FALSE FALSE  TRUE
#> [10]  TRUE  TRUE  TRUE FALSE  TRUE  TRUE

# get duplicate values
vt <- c(2, 1, 3, 6, 2, 4, 7, 0, 3, 3, 2, 2, 8, 4, 0)
vt[duplicated(c(2, 1, 3, 6, 2, 4, 7, 0, 3, 3, 2, 2, 8, 4, 0))]
#> [1] 2 3 3 2 2 4 0

# checking if an object contains any duplicates
any(duplicated(1:10))
#> [1] FALSE

any(duplicated(c(2, 1, 3, 6, 2, 4, 7, 0, 3, 3, 2, 2, 8, 4, 0)))
#> [1] TRUE

# get the first duplicate position
anyDuplicated(1:10)
#> [1] 0

anyDuplicated(c(2, 1, 3, 6, 2, 4, 7, 0, 3, 3, 2, 2, 8, 4, 0))
#> [1] 5
```

The function `duplicated()` and `anyDuplicated()` also work on data frames. The former drops unique rows while keeping duplicate rows.

```
movies_2006 <- mov[mov$Year == 2006, c(7,12)]
movies_2006 <- movies_2006[order(movies_2006$Year, movies_2006$Metascore),]
head(movies_2006)
#>      Year Metascore
```

```
#> 774 2006        36
#> 309 2006        45
#> 551 2006        45
#> 594 2006        45
#> 734 2006        46
#> 531 2006        47

# checking for any duplicates
any(duplicated(movies_2006))
#> [1] TRUE

anyDuplicated(movies_2006)
#> [1] 3

# checking for duplicates
duplicated(movies_2006)
#>  [1] FALSE FALSE  TRUE  TRUE FALSE FALSE FALSE FALSE FALSE
#> [10]  TRUE FALSE  TRUE FALSE FALSE  TRUE FALSE FALSE FALSE
#> [19]  TRUE  TRUE FALSE FALSE  TRUE  TRUE  TRUE FALSE  TRUE
#> [28] FALSE  TRUE FALSE FALSE FALSE FALSE  TRUE FALSE  TRUE
#> [37] FALSE FALSE  TRUE FALSE FALSE FALSE  TRUE  TRUE

# returning duplicates
movies_2006_dup <- movies_2006 [duplicated(movies_2006), ]
head(movies_2006_dup)
#>     Year Metascore
#> 551 2006        45
#> 594 2006        45
#> 859 2006        52
#> 960 2006        53
#> 902 2006        58
#> 670 2006        64
```

4.13.2 Get unique values

The function `unique()` extracts unique values from a vector or data frame.

```
# return unique values
unique(1:10)
#>  [1]  1  2  3  4  5  6  7  8  9 10

unique(c(2, 1, 3, 6, 2, 4, 7, 0, 3, 3, 2, 2, 8, 4, 0))
#> [1] 2 1 3 6 4 7 0 8

# return unique values using duplicated()
```

```
vt[!duplicated(c(2, 1, 3, 6, 2, 4, 7, 0, 3, 3, 2, 2, 8, 4, 0))]
#> [1] 2 1 3 6 4 7 0 8

# returning unique rows
movies_2006_uni <- unique(movies_2006)
head(movies_2006_uni)
#>     Year Metascore
#> 774 2006        36
#> 309 2006        45
#> 734 2006        46
#> 531 2006        47
#> 321 2006        48
#> 775 2006        51

# returning unique rows using duplicated()
movies_2006_uni <- subset(movies_2006, !duplicated(movies_2006))
head(movies_2006_uni)
#>     Year Metascore
#> 774 2006        36
#> 309 2006        45
#> 734 2006        46
#> 531 2006        47
#> 321 2006        48
#> 775 2006        51
```

4.14 Factors in Base R

4.14.1 What are factors?

Factors are variables in R which take on a limited number of different values which are usually known as categorical values e.g. male and female or months of the year. They can contain either strings or integers but are stored internally as a vector of integers with each integer corresponding to one category. Factors can either be ordered or unordered e.g. low, medium, high for ordered and male or female for unordered.

4.14.2 Creating a factor

While the function `factor()` is used to create a factor, the function `is.factor()` is used to check for factor.

```
# creating a factor
(fac <- factor(c('female', 'male', 'male', 'female', 'male', 'male', 'male', 'female')))
#> [1] female male   male   female male   male   male   female
#> Levels: female male
```

```r
# looking at type and class
typeof(fac)
#> [1] "integer"
class(fac)
#> [1] "factor"

# checking if the object is a factor
is.factor(fac)
#> [1] TRUE
```

4.14.3 Factor attributes and structure

A factor has as attribute levels which represent the categories of the factor. The function `levels()` is used to get and set levels while `nlevels()` returns the number of categories.

```r
# get levels
levels(fac)
#> [1] "female" "male"

# set levels
(levels(fac) <- c('f', 'm'))
#> [1] "f" "m"

# resetting levels
(levels(fac) <- c('female', 'male'))
#> [1] "female" "male"

# number of categories
nlevels(fac)
#> [1] 2

# structure of the factor
str(fac)
#>  Factor w/ 2 levels "female","male": 1 2 2 1 2 2 2 1

attributes(fac)
#> $levels
#> [1] "female" "male"
#>
#> $class
#> [1] "factor"

# count of elements by category
table(fac)
```

```
#> fac
#> female   male
#>      3      5

# internally factors are stored as integers
unclass(fac)
#> [1] 1 2 2 1 2 2 2 1
#> attr(,"levels")
#> [1] "female" "male"
```

4.14.4 Rearranging levels

The argument levels is used to rearrange the levels of a factor.

```
lev <- c('male', 'female')
(fac1 <- factor(c('female', 'male', 'male', 'female', 'male', 'male', 'male', 'female'),
               levels = lev))
#> [1] female male   male   female male   male   male   female
#> Levels: male female

# comparing fac and fac1
attributes(fac)
#> $levels
#> [1] "female" "male"
#>
#> $class
#> [1] "factor"
attributes(fac1)
#> $levels
#> [1] "male"   "female"
#>
#> $class
#> [1] "factor"

table(fac)
#> fac
#> female   male
#>      3      5
table(fac1)
#> fac1
#>   male female
#>      5      3
```

4.14.5 Dropping levels

The function `droplevels()` is used to drop unused levels from a factor.

```
(fac1 <- factor(c('female', 'male', 'male', 'female', 'male', 'male', 'male', 'female')
               levels = c('male', 'female', 'boy', 'girl')))
#> [1] female male   male   female male   male   male   female
#> Levels: male female boy girl
(fac1 <- droplevels(fac1))
#> [1] female male   male   female male   male   male   female
#> Levels: male female
```

4.14.6 Changing labels

The argument label is used to change the labels of a factor.

```
# changing from male to M and from female to F
(fac1 <- factor(c('female', 'male', 'male', 'female', 'male', 'male', 'male', 'female')
               levels = c('male', 'female'),
               label = c('M', 'F')))
#> [1] F M M F M M M F
#> Levels: M F
```

4.14.7 Ordered factors

Ordered factors are factors whose orders matter for example with grading; A is greater than B and B greater than C, and so forth. The argument `order = TRUE` is used to create an ordered factor. Also, the function `ordered()` can be used to create an ordered factor while the function `is.ordered()` is used to check for ordered factor. With ordered factors, we can use the function `min()` and `max()` on them to determine the minimum and maximum values, respectively.

```
(fac2 <- factor(c('female', 'male', 'male', 'female', 'male', 'male', 'male', 'female')
               levels = lev,
               ordered = T))
#> [1] female male   male   female male   male   male   female
#> Levels: male < female
attributes(fac)
#> $levels
#> [1] "female" "male"
#>
#> $class
#> [1] "factor"
attributes(fac1)
#> $levels
#> [1] "M" "F"
#>
#> $class
#> [1] "factor"
attributes(fac2)
```

```
#> $levels
#> [1] "male"   "female"
#>
#> $class
#> [1] "ordered" "factor"

# getting minimum and maximum values
min(fac2)
#> [1] male
#> Levels: male < female
max(fac2)
#> [1] female
#> Levels: male < female

# checking for ordered factor
is.ordered(fac2)
#> [1] TRUE

ordered(c('female', 'male', 'male', 'female', 'male', 'male', 'male', 'female'))
#> [1] female male   male   female male   male   male   female
#> Levels: female < male
ordered(c('female', 'male', 'male', 'female', 'male', 'male', 'male', 'female'),
        levels = c('male', 'female'))
#> [1] female male   male   female male   male   male   female
#> Levels: male < female
```

The functions `max()` and `min()` do not work for `fac` and `fac1` because they are not ordered, hence have no minimum or maximum.

4.14.8 Converting from character to factor

The function `as.factor()` converts to a factor, if possible but is less flexible than `factor()`. It is used when we do not care about levels, label or order.

```
month.name
#>  [1] "January"   "February"  "March"     "April"
#>  [5] "May"       "June"      "July"      "August"
#>  [9] "September" "October"   "November"  "December"
class(month.name)
#> [1] "character"

# converting to factor
(month_fac <- as.factor(month.name))
#>  [1] January   February  March     April     May
#>  [6] June      July      August    September October
#> [11] November  December
```

```
#> 12 Levels: April August December February January ... September
```

4.14.9 Converting from factor to character

The function `as.character()` converts from factor to character.

```
(month_char <- as.character(month_fac))
```

4.14.10 Converting from numeric to factor

The function `cut()` is used to convert from numeric vector to factor. It bins numbers into ranges which can be treated as categories.

```
scores <- c(15,65,68,46,15,61,32,13,15,46,13,21,89,89,44,51,32,16,18,95,46,16,65,46)

# create factors from numeric
cut(scores, breaks = 5)
#>  [1] (12.9,29.4] (62.2,78.6] (62.2,78.6] (45.8,62.2]
#>  [5] (12.9,29.4] (45.8,62.2] (29.4,45.8] (12.9,29.4]
#>  [9] (12.9,29.4] (45.8,62.2] (12.9,29.4] (12.9,29.4]
#> [13] (78.6,95.1] (78.6,95.1] (29.4,45.8] (45.8,62.2]
#> [17] (29.4,45.8] (12.9,29.4] (12.9,29.4] (78.6,95.1]
#> [21] (45.8,62.2] (12.9,29.4] (62.2,78.6] (45.8,62.2]
#> 5 Levels: (12.9,29.4] (29.4,45.8] ... (78.6,95.1]

# return categories
levels(cut(scores, breaks = 5))
#> [1] "(12.9,29.4]" "(29.4,45.8]" "(45.8,62.2]" "(62.2,78.6]"
#> [5] "(78.6,95.1]"

# number of levels
nlevels(cut(scores, breaks = 5))
#> [1] 5

# check class
class(cut(scores, breaks = 5))
#> [1] "factor"

# controlling breaks
cut(scores, breaks = c(0, 40 , 50, 60, 80, 100))
#>  [1] (0,40]   (60,80]  (60,80]  (40,50]  (0,40]   (60,80]
#>  [7] (0,40]   (0,40]   (0,40]   (40,50]  (0,40]   (0,40]
#> [13] (80,100] (80,100] (40,50]  (50,60]  (0,40]   (0,40]
#> [19] (0,40]   (80,100] (40,50]  (0,40]   (60,80]  (40,50]
#> Levels: (0,40] (40,50] (50,60] (60,80] (80,100]
```

```
# adding labels
cut(scores, breaks = c(0, 40 , 50, 60, 80, 100), labels = c('F', 'D', 'C', 'B', 'A'))
#>  [1] F B B D F B F F F D F F A A D C F F F A D F B D
#> Levels: F D C B A

# majority of the students failed
table(cut(scores, breaks = c(0, 40 , 50, 60, 80, 100), labels = c('F', 'D', 'C', 'B', 'A')))
#> 
#>  F  D  C  B  A 
#> 11  5  1  4  3
```

4.14.11 Converting from factor to numeric

To convert from a factor to numeric, the function `as.numeric()` does not work. To use it, we first have to convert the factor to character using `as.character()` or `levels(fac)[fac]`. Below we make use of both methods to achieve our objective.

```
num_vec <- c(15,65,68,46,15,61,32,13,15,46,13,21,89,89,44,51,32,16,18,95,46,16,65,46)
mean(num_vec)
#> [1] 42.375

#converting to factor
(fac3 <- factor(num_vec))
#>  [1] 15 65 68 46 15 61 32 13 15 46 13 21 89 89 44 51 32 16
#> [19] 18 95 46 16 65 46
#> Levels: 13 15 16 18 21 32 44 46 51 61 65 68 89 95

# calculating mean
mean(fac3)
#> [1] NA


# as.numeric() doesn't seem to work
mean(as.numeric(fac3))
#> [1] 6.958333


# using as.character
mean(as.numeric(as.character(fac3)))
#> [1] 42.375

# using levels()
mean(as.numeric(levels(fac3)[fac3]))
#> [1] 42.375
```

4.15 String manipulation with base R

4.15.1 String length and character count

The function `length()` returns the count of elements in a vector. The function `nchar()` returns the count of letters in a string.

```
month.name
#>  [1] "January"   "February"  "March"     "April"
#>  [5] "May"       "June"      "July"      "August"
#>  [9] "September" "October"   "November"  "December"

#count of elements
length(month.name)
#> [1] 12

# count of letters
month.name
#>  [1] "January"   "February"  "March"     "April"
#>  [5] "May"       "June"      "July"      "August"
#>  [9] "September" "October"   "November"  "December"
nchar(month.name)
#>  [1] 7 8 5 5 3 4 4 6 9 7 8 8
```

4.15.2 Strings formatting (case-folding)

The functions `toupper()` and `tolower()` are used to convert to upper and lower cases, respectively while `casefold()` is a wrapper to these functions.

```
# uppercase
toupper(month.name)
#>  [1] "JANUARY"   "FEBRUARY"  "MARCH"     "APRIL"
#>  [5] "MAY"       "JUNE"      "JULY"      "AUGUST"
#>  [9] "SEPTEMBER" "OCTOBER"   "NOVEMBER"  "DECEMBER"
casefold(month.name, upper = TRUE)
#>  [1] "JANUARY"   "FEBRUARY"  "MARCH"     "APRIL"
#>  [5] "MAY"       "JUNE"      "JULY"      "AUGUST"
#>  [9] "SEPTEMBER" "OCTOBER"   "NOVEMBER"  "DECEMBER"

# lowercase
tolower(month.name)
#>  [1] "january"   "february"  "march"     "april"
#>  [5] "may"       "june"      "july"      "august"
#>  [9] "september" "october"   "november"  "december"
casefold(month.name, upper = FALSE)
#>  [1] "january"   "february"  "march"     "april"
#>  [5] "may"       "june"      "july"      "august"
```

```
#>  [9] "september" "october"   "november"  "december"
```

4.15.3 Join and Split strings

4.15.3.1 Joining strings using cat()

The function `cat()` converts its arguments to strings and concatenates them after appending a separator string (given by sep) to them.

```
a <- month.name[1]
b <- month.name[2]
c <- month.name[3]
cat(b,'comes after', a ,'but comes before', c)
#> February comes after January but comes before March
cat(b,'comes before', a ,'but comes after', c, sep = '/')
#> February/comes before/January/but comes after/March
cat(month.name[1:6], sep = ' - ')
#> January - February - March - April - May - June
cat(month.name[1:6], sep = ' <> ')
#> January <> February <> March <> April <> May <> June
```

Newlines and tabs can be added by using `\n` for newline and `\t` for tabs.

```
# adding a new line
cat(b,'comes after\n', a ,'but comes before', c)
#> February comes after
#>  January but comes before March

# adding a tab
cat(b,'comes after\t', a ,'but comes before', c)
#> February comes after  January but comes before March
```

The function `cat()` can write its output directly to a file if a file name is passed to it.

```
# writing to disc
cat(month.name, sep = ' <> ', file = "output/data/months.txt")

# checking if file exists
file.exists('output/data/months.txt')
#> [1] TRUE

# removing file
file.remove('output/data/months.txt')
#> [1] TRUE
```

4.15.3.2 Joining strings using paste() and paste0()

The function `paste()` concatenate vectors after converting them to character and separating them by a string given by sep. It concatenates multiple vectors element by element to give a new character vector and if one is shorter, recycling occurs with zero-length arguments being recycled to " ". With a single vector, it is simply converted to a character vector and if the argument collapse is set, the elements are condensed into a single string.

The function `paste0(...)` is equivalent to `paste(…, sep = '')`, but slightly more efficient.

```
# combining elements into a character vector
paste('a', 'b')
#> [1] "a b"
paste(1, 2, 3, 4)
#> [1] "1 2 3 4"


# using a sep
paste('a', 'b', sep = '')
#> [1] "ab"
paste(1, 2, 3, 4, sep = '')
#> [1] "1234"


# using paste0
paste0('a', 'b')
#> [1] "ab"
paste0(1, 2, 3 ,4)
#> [1] "1234"

# on a single vector
paste(c('a', 'b'), sep = ' <> ')
#> [1] "a" "b"
paste(c(1, 2), sep = ' <> ')
#> [1] "1" "2"

# two or more vectors
paste(c('a', 'b'), c('c', 'd'), sep = ' <> ')
#> [1] "a <> c" "b <> d"
paste0(c('a', 'b'), c('c', 'd'))
#> [1] "ac" "bd"
paste0(1:5, 6:10)
#> [1] "16"  "27"  "38"  "49"  "510"
paste(1:5, 10:20)
#>  [1] "1 10" "2 11" "3 12" "4 13" "5 14" "1 15" "2 16" "3 17"
```

```
#>  [9] "4 18" "5 19" "1 20"
paste(1:5, 10:20, c('a','b','c'))
#>  [1] "1 10 a" "2 11 b" "3 12 c" "4 13 a" "5 14 b" "1 15 c"
#>  [7] "2 16 a" "3 17 b" "4 18 c" "5 19 a" "1 20 b"

# combining character and variables with paste
paste(b,'comes after', a ,'but comes before', c)
#> [1] "February comes after January but comes before March"
paste(b,'comes after', a ,'but comes before', c, sep = "    ")
#> [1] "February    comes after    January    but comes before    March"
paste(b,'comes after', a ,'but comes before', c, sep = "/")
#> [1] "February/comes after/January/but comes before/March"
paste('version 1.', 1:5, sep = '')
#> [1] "version 1.1" "version 1.2" "version 1.3" "version 1.4"
#> [5] "version 1.5"

# combining character and variables with paste0
paste0(b,' comes after ', a ,' but comes before ', c)
#> [1] "February comes after January but comes before March"
paste0(b,'    comes after    ', a ,'    but comes before    ', c)
#> [1] "February    comes after    January    but comes before    March"

paste0(b,'/comes after/', a ,'/but comes before/', c)
#> [1] "February/comes after/January/but comes before/March"
paste0('version 1.', 1:5)
#> [1] "version 1.1" "version 1.2" "version 1.3" "version 1.4"
#> [5] "version 1.5"
```

The collapse argument is used to collapse elements returned into a single string.

```
# collapsing vectors
paste(1:10, collapse = '~')
#> [1] "1~2~3~4~5~6~7~8~9~10"
paste(c('a', 'b'), c('c', 'd'), collapse = ' <> ')
#> [1] "a c <> b d"
paste0(c('a', 'b'), c('c', 'd'), collapse = ' <> ')
#> [1] "ac <> bd"

paste0(1:5, 6:10, collapse = '--')
#> [1] "16--27--38--49--510"
paste(month.name[1:6], collapse = " - ")
#> [1] "January - February - March - April - May - June"
```

4.15.3.3 Joining strings using sprintf()

The function `sprintf()` returns a character vector containing a formatted combination of text and variable values. The format of the variables is passed using one of the following characters `aAdifeEgGosxX%` and should start with %.

4.15.3.3.1 Formatting with integers The command `%d` is used for formatting integers.

```
# using an integer as a variable
x <- 2
sprintf('%d * %d = %d', x, x, x ** 2)
#> [1] "2 * 2 = 4"
x <- c(1:4)
y <- x ** 2
sprintf('%d squared is equal to %d', x, y)
#> [1] "1 squared is equal to 1"  "2 squared is equal to 4"
#> [3] "3 squared is equal to 9"  "4 squared is equal to 16"

### padding integers with zeros
num <- c(123, 1, 100, 200, 10200, 25000)
sprintf('my registration number is %05d', num)
#> [1] "my registration number is 00123"
#> [2] "my registration number is 00001"
#> [3] "my registration number is 00100"
#> [4] "my registration number is 00200"
#> [5] "my registration number is 10200"
#> [6] "my registration number is 25000"
```

4.15.3.3.2 Formatting with strings The command `%s` is used for formatting strings.

```
# using a string as a variable
x <- 'my name is'
y <- 'james'
z <- 'london'
sprintf('%s %s and i live and work in %s', x, y, z)
#> [1] "my name is james and i live and work in london"

# combining strings and integers
x <- 'my name is'
y <- 'james'
z <- 35
sprintf('%s %s and i am %d years', x, y, z)
#> [1] "my name is james and i am 35 years"

names = c('paul', 'alphonse', 'michael', 'james', 'samson', 'terence', 'derin')
```

```
age = c(30, 35, 32, 37, 29, 40, 30)
sprintf('i am %s and i am %d years old', names, age)
#> [1] "i am paul and i am 30 years old"
#> [2] "i am alphonse and i am 35 years old"
#> [3] "i am michael and i am 32 years old"
#> [4] "i am james and i am 37 years old"
#> [5] "i am samson and i am 29 years old"
#> [6] "i am terence and i am 40 years old"
#> [7] "i am derin and i am 30 years old"
```

4.15.3.3.3 Formatting with doubles or floating-points The command `%f` is used for formatting doubles while either `%e` or `%E` for formatting exponential.

```
# using doubles as a variable
x <- 1000/6
sprintf('1000 divided by 3 is %f', x)
#> [1] "1000 divided by 3 is 166.666667"

# rounding a double to the nearest decimal
sprintf('1000 divided by 3 is %.3f', x)
#> [1] "1000 divided by 3 is 166.667"
sprintf('1000 divided by 3 is %.2f', x)
#> [1] "1000 divided by 3 is 166.67"
sprintf('1000 divided by 3 is %.1f', x)
#> [1] "1000 divided by 3 is 166.7"

# rounding a double to the nearest whole number
sprintf('1000 divided by 3 is %1.f', x)
#> [1] "1000 divided by 3 is 167"

# printing a plus (+) in front of a double
sprintf('+1000 divided by 3 is %+.1f', x)
#> [1] "+1000 divided by 3 is +166.7"

# printing space in front of a double
sprintf('1000 divided by 3 is %f', x)
#> [1] "1000 divided by 3 is 166.666667"
sprintf('1000 divided by 3 is % f', x)
#> [1] "1000 divided by 3 is  166.666667"

# exponential
sprintf("%e", pi)
#> [1] "3.141593e+00"
```

4.15.3.4 Splitting strings using strsplit()

The function `strsplit()` splits the elements of a character vector into substrings by a specific split character. It returns a list.

```
str(strsplit(c('2020-01-01', '2019-03-31', '2018-06-30'), split = "-"))
#> List of 3
#>  $ : chr [1:3] "2020" "01" "01"
#>  $ : chr [1:3] "2019" "03" "31"
#>  $ : chr [1:3] "2018" "06" "30"
str(strsplit(c('2020 01 01', '2019 03 31', '2018 06 30'), split = " "))
#> List of 3
#>  $ : chr [1:3] "2020" "01" "01"
#>  $ : chr [1:3] "2019" "03" "31"
#>  $ : chr [1:3] "2018" "06" "30"
str(strsplit(c('2020, 01, 01', '2019, 03, 31', '2018, 06, 30'), split = ", "))
#> List of 3
#>  $ : chr [1:3] "2020" "01" "01"
#>  $ : chr [1:3] "2019" "03" "31"
#>  $ : chr [1:3] "2018" "06" "30"
```

4.15.4 Extract and Replace part of strings

4.15.4.1 Extracting substring using substr()

The function `substr()` extracts a substring from a string by indexing. It uses start for the beginning position and stop for the ending position. It is like indexing but applied to a string.

```
var <- c('2020-01-01', '2019-03-31', '2018-06-30')
substr(var, start = 1, stop = 4)
#> [1] "2020" "2019" "2018"
substr(var, start = 6, stop = 7)
#> [1] "01" "03" "06"
substr(var, start = 9, stop = 10)
#> [1] "01" "31" "30"
```

4.15.4.2 Replacing substring using substr()

The function `substr()` is also used to replace substring in a string by assigning a different string to the extracted substring.

```
var <- c('2020-01-01', '2019-03-31', '2018-06-30')
substr(var, start = 1, stop = 4) <- c('2010', '2011', '2012')
var
#> [1] "2010-01-01" "2011-03-31" "2012-06-30"

weekdays <- c('monday', 'tuesday', 'wednesday', 'thursday', 'friday', 'saturday', 'sund
```

```
substr(weekdays, start = 1, stop = 1) <- toupper(substr(weekdays, start = 1, stop = 1))
weekdays
#> [1] "Monday"    "Tuesday"   "Wednesday" "Thursday"
#> [5] "Friday"    "Saturday"  "Sunday"
```

4.15.4.3 Replacing substrings using sub()

The function `sub()` replaces a substring at first occurrence in a string.

```
var <- c('2020-01-01', '2019-03-31', '2018-06-30')
sub("-", "", var)
#> [1] "202001-01" "201903-31" "201806-30"
sub("-", " ", var)
#> [1] "2020 01-01" "2019 03-31" "2018 06-30"
```

4.15.4.4 Replacing substrings using gsub()

The function `gsub()` replaces a substring throughout a string.

```
var <- c('2020-01-01', '2019-03-31', '2018-06-30')
gsub("-", "/", var)
#> [1] "2020/01/01" "2019/03/31" "2018/06/30"
gsub("-", " ", var)
#> [1] "2020 01 01" "2019 03 31" "2018 06 30"
```

4.15.4.5 Replacing substring using chartr()

The function `chartr()` replaces a substring throughout a string.

```
var <- c('2020-01-01', '2019-03-31', '2018-06-30')
chartr(old = "-", new = "/", var)
#> [1] "2020/01/01" "2019/03/31" "2018/06/30"
chartr(old = "-", new = " ", var)
#> [1] "2020 01 01" "2019 03 31" "2018 06 30"
```

4.15.4.6 Remove white spaces and clean string values

The function `trimws()` removes white spaces.

```
trimws(c(' 2020-01-01 ', ' 2019-03-31 ', ' 2018-06-30 '))
#> [1] "2020-01-01" "2019-03-31" "2018-06-30"
```

4.15.5 Pattern matching using regular expression

4.15.5.1 Regex functions

- `grep()`
- `grepl()`

- `regexpr()`
- `gregexpr()`
- `regexec()`
- `sub()`
- `gsub()`

4.15.5.1.1 The `grep()` function The function `grep()` returns the index position or value of elements that match a pattern.

```
# returning index position
month.name
#>  [1] "January"   "February"  "March"     "April"
#>  [5] "May"       "June"      "July"      "August"
#>  [9] "September" "October"   "November"  "December"
grep(pattern = 'uary', month.name)
#> [1] 1 2

# returning values
grep('uary', month.name, value = TRUE)
#> [1] "January"  "February"

# ignoring case
grep('ju', month.name, value = TRUE)
#> character(0)
grep('ju', month.name, ignore.case = TRUE, value = TRUE)
#> [1] "June" "July"
```

4.15.5.2 The grepl() function

The function `grepl()` returns `TRUE` for pattern match and FALSE for no pattern match.

```
grepl('uary', month.name)
#>  [1]  TRUE  TRUE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
#> [10] FALSE FALSE FALSE
grepl('ju', month.name, ignore.case = TRUE)
#>  [1] FALSE FALSE FALSE FALSE FALSE  TRUE  TRUE FALSE FALSE
#> [10] FALSE FALSE FALSE
```

4.15.5.3 The `regexpr()` function

The function `regexpr()` returns the position of the first pattern match in an element with -1 representing no pattern match.

```
regexpr('ber', month.name, ignore.case = TRUE)
#>  [1] -1 -1 -1 -1 -1 -1 -1 -1  7  5  6  6
#> attr(,"match.length")
```

```
#>  [1] -1 -1 -1 -1 -1 -1 -1 -1  3  3  3  3
#> attr(,"index.type")
#> [1] "chars"
#> attr(,"useBytes")
#> [1] TRUE
(st <- state.name[20:25])
#> [1] "Maryland"      "Massachusetts" "Michigan"
#> [4] "Minnesota"     "Mississippi"   "Missouri"
regexpr('ss', st, ignore.case = TRUE)
#> [1] -1  3 -1 -1  3  3
#> attr(,"match.length")
#> [1] -1  2 -1 -1  2  2
#> attr(,"index.type")
#> [1] "chars"
#> attr(,"useBytes")
#> [1] TRUE
```

4.15.5.4 The `gregexpr()` function

The function `gregexpr()` returns the position of all pattern matches in an element with -1 representing no pattern match.

```
# same as as.list(regexpr())
(st <- state.name[23:25])
#> [1] "Minnesota"   "Mississippi" "Missouri"
gregexpr('ss', st, ignore.case = TRUE)
#> [[1]]
#> [1] -1
#> attr(,"match.length")
#> [1] -1
#> attr(,"index.type")
#> [1] "chars"
#> attr(,"useBytes")
#> [1] TRUE
#>
#> [[2]]
#> [1] 3 6
#> attr(,"match.length")
#> [1] 2 2
#> attr(,"index.type")
#> [1] "chars"
#> attr(,"useBytes")
#> [1] TRUE
#>
#> [[3]]
#> [1] 3
```

```
#> attr(,"match.length")
#> [1] 2
#> attr(,"index.type")
#> [1] "chars"
#> attr(,"useBytes")
#> [1] TRUE
```

4.15.5.5 Regex Operations

4.15.5.5.1 Matching spaces

- [[:blank:]] matches space and tab characters
- [[:space:]] matches tab, newline, vertical tab, form feed, carriage return, and space
- \s matches space character
- \S matches non-space character

```
# creating a character vector
var <- c('2020 01 01', '2019 03 31', '2018 06 30')

# POSIX Character
gsub('[[:space:]]', '-', var)
#> [1] "2020-01-01" "2019-03-31" "2018-06-30"
gsub('[[:space:]]', '/', var)
#> [1] "2020/01/01" "2019/03/31" "2018/06/30"
gsub('[[:blank:]]', '_', var)
#> [1] "2020_01_01" "2019_03_31" "2018_06_30"
gsub('[[:blank:]]', '/', var)
#> [1] "2020/01/01" "2019/03/31" "2018/06/30"

# Sequences
gsub('\\s', '-', var)
#> [1] "2020-01-01" "2019-03-31" "2018-06-30"
gsub('\\s', '/', var)
#> [1] "2020/01/01" "2019/03/31" "2018/06/30"
gsub('\\s', '_', var)
#> [1] "2020_01_01" "2019_03_31" "2018_06_30"
gsub('\\s', '/', var)
#> [1] "2020/01/01" "2019/03/31" "2018/06/30"

# using strsplit() to split based on a pattern
var <- c('2020 01 01', '2019 03 31', '2018 06 30')
str(strsplit(var, split = '\\s'))
#> List of 3
#>  $ : chr [1:3] "2020" "01" "01"
#>  $ : chr [1:3] "2019" "03" "31"
```

```
#>  $ : chr [1:3] "2018" "06" "30"

# matching non-space character with \\S
var <- c('2020 01 01', '2019 03 31', '2018 06 30')
gsub('\\S', '-', var)
#> [1] "---- -- --" "---- -- --" "---- -- --"
gsub('\\S', '/', var)
#> [1] "//// // //" "//// // //" "//// // //"
gsub('\\S', '_', var)
#> [1] "____ __ __" "____ __ __" "____ __ __"
gsub('\\S', '/', var)
#> [1] "//// // //" "//// // //" "//// // //"
```

Matching alphabetic characters

- [[:alpha:]] matches alphabetic characters
- [[:lower:]] matches lowercase characters
- [[:upper:]] matches uppercase characters

```
var <- 'a1b2c3d4e5f'

# matching alphabetic characters
gsub('[[:alpha:]]', '', var)
#> [1] "12345"
gsub('[[:alpha:]]', '-', var)
#> [1] "-1-2-3-4-5-"

# matching lowercase letters
gsub('[[:lower:]]', '', month.name)
#>  [1] "J" "F" "M" "A" "M" "J" "J" "A" "S" "O" "N" "D"

# matching uppercase letters
gsub('[[:upper:]]', '', month.name)
#>  [1] "anuary"   "ebruary"  "arch"     "pril"     "ay"
#>  [6] "une"      "uly"      "ugust"    "eptember" "ctober"
#> [11] "ovember"  "ecember"
```

Matching numerical digits

- [[:digit:]] and \d matches numbers from 0-9.

```
var <- 'a1b2c3d4e5f'

# POSIX Character
gsub('[[:digit:]]', '', var)
#> [1] "abcdef"
gsub('[[:digit:]]', '-', var)
#> [1] "a-b-c-d-e-f"
```

```
# Sequences
gsub('\\d', '', var)
#> [1] "abcdef"
gsub('\\d', '-', var)
#> [1] "a-b-c-d-e-f"
```

Matching letters and numbers (alphanumeric characters)

- [[:alnum:]] matches alphanumeric characters ([[:alpha:]] and [[:digit:]])
- [[:xdigit:]] matches Hexadecimal digits (0 1 2 3 4 5 6 7 8 9 A B C D E F a b c d e f)
- \w matches word characters

```
var <- 'a1@; 2#4c $8`*%f^!1~0&^h*()j'

# alphanumeric characters
gsub('[[:alnum:]]', '', var)
#> [1] "@; # $`*%^!~&^*()"
gsub('[[:alnum:]]', '-', var)
#> [1] "--@; -#-- $-`*%-^!-~-&^-*()-"

# Hexadecimal digits
gsub('[[:xdigit:]]', '', var)
#> [1] "@; # $`*%^!~&^h*()j"
gsub('[[:xdigit:]]', '-', var)
#> [1] "--@; -#-- $-`*%-^!-~-&^h*()j"

# matching word characters
gsub('\\w', '', var)
#> [1] "@; # $`*%^!~&^*()"
gsub('\\w', '-', var)
#> [1] "--@; -#-- $-`*%-^!-~-&^-*()-"
```

Matching punctuation

- [[:punct:]] matches punctuation characters.
- \W matches non-word characters.

```
var <- 'a1@; 2#4c $8`*%f^!1~0&^h*()j'

# matching punctuation characters
gsub('[[:punct:]]', '', var)
#> [1] "a1 24c 8f10hj"
gsub('[[:punct:]]', '-', var)
#> [1] "a1-- 2-4c -8---f--1-0--h---j"

# matching non-word characters
```

```
gsub('\\W', '', var)
#> [1] "a124c8f10hj"
gsub('\\W', '-', var)
#> [1] "a1---2-4c--8---f--1-0--h---j"
```

Matching letters, numbers, and punctuation

- [[:graph:]] matches graphical characters ([[:alpha:]] and [[:punct:]])
- . matches any character (except newline character)

```
# matching graphical characters
var <- 'a1@; 2#4c $8%f^!10&^h*()j'
gsub('[[:graph:]]', ' ', var)
#> [1] "                         "

# matching anything but newline characters
var <- 'a1@; 2#4c $8%f^!10&^h*()j'
gsub('.', ' ', var)
#> [1] "                         "
```

Matching whitespace

- \s is used to match whitespaces.

```
# removing whitespace
gsub('\\s', '', c(' 2020-01-01 ', ' 2019-03-31 ', ' 2018-06-30 '))
#> [1] "2020-01-01" "2019-03-31" "2018-06-30"
```

Matching a newline

- \n is used to match a newline.

```
cat('good morning \n i am fru kinglsy \n i will be your instructor')
#> good morning
#>  i am fru kinglsy
#>  i will be your instructor

# replacing newline with tab
gsub('\\n', '\t', 'good morning \n i am fru kinglsy \n i will be your instructor')
#> [1] "good morning \t i am fru kinglsy \t i will be your instructor"

# print it out
cat(gsub('\\n', '\t', 'good morning \n i am fru kinglsy \n i will be your instructor'))
#> good morning      i am fru kinglsy    i will be your instructor
```

Matching tab

- \t is used to match tabs.

```
# replacing tab by newline
gsub('\\t', '\n', 'good morning \t i am fru kinglsy \t i will your instructor')
#> [1] "good morning \n i am fru kinglsy \n i will your instructor"

# printing it out
cat(gsub('\\t', '\n', 'good morning \t i am fru kinglsy \t i will your instructor'))
#> good morning
#>  i am fru kinglsy
#>  i will your instructor
```

4.15.5.6 Matching metacharacters

Metacharacters consist of non-alphanumeric symbols such as $. ^ * | + ! ? () {} []. They are matched, by escaping them with a double backslash \.

```
# matching $
sales <-
  c('$25000', '$20000', '$22500', '$24000', '$30000', '$35000')
sub('\\$', '', sales)
#> [1] "25000" "20000" "22500" "24000" "30000" "35000"

# matching +
sales <-
  c('+25000', '+20000', '+22500', '+24000', '+30000', '+35000')
sub('\\+', '', sales)
#> [1] "25000" "20000" "22500" "24000" "30000" "35000"

# matching .
dates <-
  c('01.01.2012', '01.02.2012', '01.03.2012', '01.04.2012', '01.05.2012', '01.06.2012')
gsub('\\.', '-', dates)
#> [1] "01-01-2012" "01-02-2012" "01-03-2012" "01-04-2012"
#> [5] "01-05-2012" "01-06-2012"

# matching *
dates <-
  c('01*01*2012', '01*02*2012', '01*03*2012', '01*04*2012', '01*05*2012', '01*06*2012')
gsub('\\*', '-', dates)
#> [1] "01-01-2012" "01-02-2012" "01-03-2012" "01-04-2012"
#> [5] "01-05-2012" "01-06-2012"
#> [1] "01-01-2012" "01-02-2012" "01-03-2012" "01-04-2012" "01-05-2012"
#> [6] "01-06-2012"

# matching ^
dates <-
  c('01^01^2012', '01^02^2012', '01^03^2012', '01^04^2012', '01^05^2012', '01^06^2012')
```

```r
gsub('\\^', '-', dates)
#> [1] "01-01-2012" "01-02-2012" "01-03-2012" "01-04-2012"
#> [5] "01-05-2012" "01-06-2012"

# matching |
dates <-
  c('01|01|2012', '01|02|2012', '01|03|2012', '01|04|2012', '01|05|2012', '01|06|2012')
gsub('\\|', '-', dates)
#> [1] "01-01-2012" "01-02-2012" "01-03-2012" "01-04-2012"
#> [5] "01-05-2012" "01-06-2012"

# matching \
dates <-
  c('01\\01\\2012', '01\\02\\2012', '01\\03\\2012',
    '01\\04\\2012', '01\\05\\2012', '01\\06\\2012')
gsub('\\\\', '-', dates)
#> [1] "01-01-2012" "01-02-2012" "01-03-2012" "01-04-2012"
#> [5] "01-05-2012" "01-06-2012"

# matching \\.
dates <-
  c('01\\.01\\.2012', '01\\.02\\.2012', '01\\.03\\.2012',
    '01\\.04\\.2012', '01\\.05\\.2012', '01\\.06\\.2012')
gsub('\\\\\\.', '-', dates)
#> [1] "01-01-2012" "01-02-2012" "01-03-2012" "01-04-2012"
#> [5] "01-05-2012" "01-06-2012"
```

Alternates and ranges

Either or (|)

```r
# replacing either uary or ember or ober
gsub('uary|ember|ober', '-', month.name)
#>  [1] "Jan-"    "Febr-"  "March"  "April"  "May"    "June"
#>  [7] "July"    "August" "Sept-"  "Oct-"   "Nov-"   "Dec-"
```

set of characters ([]) matches a set of characters.

ranges (-) matches a range of characters.

```r
# matching vowels
gsub('[aeiou]', '*', month.name)
#>  [1] "J*n**ry"   "F*br**ry"  "M*rch"     "Apr*l"
#>  [5] "M*y"       "J*n*"      "J*ly"      "A*g*st"
#>  [9] "S*pt*mb*r" "Oct*b*r"   "N*v*mb*r"  "D*c*mb*r"

# matching lower cases
gsub('[a-z]', '*', month.name)
```

```
#>  [1] "J******"   "F*******"  "M****"     "A****"    
#>  [5] "M**"       "J***"      "J***"      "A*****"   
#>  [9] "S********" "O******"   "N*******"  "D*******"

# matching upper cases
gsub('[A-Z]', '*', month.name)
#>  [1] "*anuary"   "*ebruary"  "*arch"     "*pril"    
#>  [5] "*ay"       "*une"      "*uly"      "*ugust"   
#>  [9] "*eptember" "*ctober"   "*ovember"  "*ecember"

# matching the letters m to z
gsub('[m-z]', '*', month.name)
#>  [1] "Ja**a**"   "Feb**a**"  "Ma*ch"     "A**il"    
#>  [5] "Ma*"       "J**e"      "J*l*"      "A*g***"   
#>  [9] "Se**e*be*" "Oc**be*"   "N**e*be*"  "Dece*be*"

# matching the numbers 0 to 9
gsub('[0-9]', '*', c('1a8g9u93l48p51359p78'))
#> [1] "*a*g*u**l**p*****p**"

# matching the numbers 1 to 5
gsub('[1-5]', '*', c('1a8g9u93l48p51359p78'))
#> [1] "*a8g9u9*l*8p****9p78"

# matching alphanumeric
gsub('[a-zA-Z0-9]', '*', c('1a8#g9u/93l48p51*395(9p78'))
#> [1] "***#***/***********(****"
```

Not [^abc]

```
# matching everything but vowels
gsub('[^aeiou]', '*', month.name)
#>  [1] "*a*ua**"   "*e**ua**"  "*a***"     "***i*"    
#>  [5] "*a*"       "*u*e"      "*u**"      "*u*u**"   
#>  [9] "*e****e*"  "***o*e*"   "*o*e**e*"  "*e*e**e*"

# matching everything but lowercase letters
gsub('[^a-z]', '*', month.name)
#>  [1] "*anuary"   "*ebruary"  "*arch"     "*pril"    
#>  [5] "*ay"       "*une"      "*uly"      "*ugust"   
#>  [9] "*eptember" "*ctober"   "*ovember"  "*ecember"
```

Anchors

- ^ matches a pattern at the start of a string. / $ matches a pattern at the end of a string.

```
# start of a string
gsub('^J', 'j', month.name)
#>  [1] "january"   "February"  "March"     "April"
#>  [5] "May"       "june"      "july"      "August"
#>  [9] "September" "October"   "November"  "December"

# end of a string
gsub('ber$', 'ba', month.name)
#>  [1] "January"  "February" "March"    "April"    "May"
#>  [6] "June"     "July"     "August"   "Septemba" "Octoba"
#> [11] "Novemba"  "Decemba"
```

Quantifiers

- – matches a pattern 0 or more times
- – matches a pattern 1 or more times
- ? matches a pattern 0 or one time
- x{m} matches x exactly m times
- x{m,} matches x exactly m or more times
- x{m,n} matches x exactly m or n times

```
# match 's' zero or one time
grep('s?', month.name, value = TRUE)
#>  [1] "January"   "February"  "March"     "April"
#>  [5] "May"       "June"      "July"      "August"
#>  [9] "September" "October"   "November"  "December"

# match 'J' one or more times
grep('J+', month.name, value = TRUE)
#> [1] "January" "June"    "July"

# match 'e' one or more times
grep('e+', state.name, value = TRUE)
#>  [1] "Connecticut"   "Delaware"      "Georgia"
#>  [4] "Kentucky"      "Maine"         "Massachusetts"
#>  [7] "Minnesota"     "Nebraska"      "Nevada"
#> [10] "New Hampshire" "New Jersey"    "New Mexico"
#> [13] "New York"      "Oregon"        "Pennsylvania"
#> [16] "Rhode Island"  "Tennessee"     "Texas"
#> [19] "Vermont"       "West Virginia"

# matched 'y', zero or more times
grep('y*', month.name, value = TRUE)
#>  [1] "January"   "February"  "March"     "April"
```

```
#>  [5] "May"       "June"      "July"      "August"
#>  [9] "September" "October"   "November"  "December"

# matched 'a', zero or more times
grep('a*', month.name, value = TRUE)
#>  [1] "January"   "February"  "March"     "April"
#>  [5] "May"       "June"      "July"      "August"
#>  [9] "September" "October"   "November"  "December"

# match 'a' zero or more times and 'y'
grep('a*y', month.name, value = TRUE)
#> [1] "January"  "February" "May"      "July"

# match 'y' zero or more times and 'a'
grep('y*a', month.name, value = TRUE)
#> [1] "January"  "February" "March"    "May"

# match 's', exactly 2 times
grep(pattern = "s{2}", state.name, value = TRUE)
#> [1] "Massachusetts" "Mississippi"   "Missouri"
#> [4] "Tennessee"

# match 's', exactly 1 or more times
grep(pattern = "s{1,}", state.name, value = TRUE)
#>  [1] "Alaska"        "Arkansas"      "Illinois"
#>  [4] "Kansas"        "Louisiana"     "Massachusetts"
#>  [7] "Minnesota"     "Mississippi"   "Missouri"
#> [10] "Nebraska"      "New Hampshire" "New Jersey"
#> [13] "Pennsylvania"  "Rhode Island"  "Tennessee"
#> [16] "Texas"         "Washington"    "West Virginia"
#> [19] "Wisconsin"

# match 's', exactly 1 or 2 times
grep(pattern = "s{1,2}", state.name, value = TRUE)
#>  [1] "Alaska"        "Arkansas"      "Illinois"
#>  [4] "Kansas"        "Louisiana"     "Massachusetts"
#>  [7] "Minnesota"     "Mississippi"   "Missouri"
#> [10] "Nebraska"      "New Hampshire" "New Jersey"
#> [13] "Pennsylvania"  "Rhode Island"  "Tennessee"
#> [16] "Texas"         "Washington"    "West Virginia"
#> [19] "Wisconsin"
```

Groups

() matches group patterns.

```
# match 2 repeating 's' followed by an 'e'
grep(pattern = '(s{2})e', state.name, value = TRUE)
#> [1] "Tennessee"
```

Chapter 5

Tidyverse R

Hadley Wickham and Garrett Grolemund, in their excellent and freely available book R for Data Science, promote the concept of "tidy data." The Tidyverse collection of R packages attempt to realize this concept in concrete libraries.

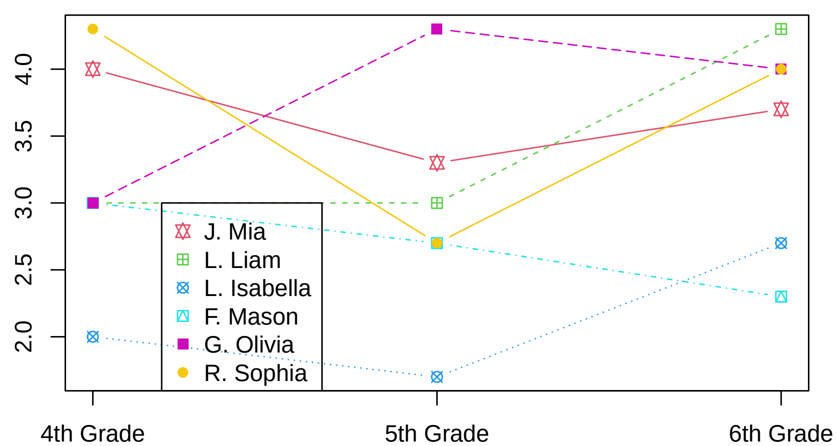
In brief, tidy data carefully separates variables (the columns of a table, also called features or fields) from observations (the rows of a table, also called samples). At the intersection of these two, we find values, one data item (datum) in each cell. Unfortunately, the data we encounter is often not arranged in this useful way, and it requires normalization. In particular, what are really values are often represented either as columns or as rows instead. To demonstrate what this means, let us consider an example (a small elementary school class).

```
library(tidyverse)
# inline reading, tibble version
students <- tribble(
  ~'Last Name', ~'First Name', ~'4th Grade', ~'5th Grade', ~'6th Grade',
  "Johnson", "Mia", "A", "B+", "A-",
  "Lopez", "Liam", "B", "B", "A+",
  "Lee", "Isabella", "C", "C-", "B-",
  "Fisher", "Mason", "B", "B-", "C+",
  "Gupta", "Olivia", "B", "A+", "A",
  "Robinson", "Sophia", "A+", "B-", "A"
)
students
#> # A tibble: 6 x 5
#>   `Last Name` `First Name` `4th Grade` `5th Grade`
#>   <chr>       <chr>        <chr>       <chr>
#> 1 Johnson     Mia          A           B+
#> 2 Lopez       Liam         B           B
#> 3 Lee         Isabella     C           C-
```

```
#> 4 Fisher      Mason       B           B-
#> 5 Gupta       Olivia      B           A+
#> 6 Robinson    Sophia      A+          B-
#> # ... with 1 more variable: 6th Grade <chr>
```

This view of the data is easy for humans to read. We can see trends in the scores each student received over several years of education. Moreover, this format might lend itself to useful visualizations fairly easily:

```
# Generic conversion of letter grades to numbers
recodes.str <- "'A+'=4.3;'A'=4;'A-'=3.7;'B+'=3.3;'B'=3;'B-'=2.7;'C+'=2.3;'C'= 2;'C-'=1
students$`4th Grade` <- car::recode(students$`4th Grade`, recodes.str)
students$`5th Grade` <- car::recode(students$`5th Grade`, recodes.str)
students$`6th Grade` <- car::recode(students$`6th Grade`, recodes.str)

# create plot
matplot(t(students[,c(3:5)]), type = "b", pch = 11:16, col = 2:7, xaxt="n", ylab="")
Axis(labels = names(students)[3:5], side=1, at = 1:3)
legend(1.2, 3, paste(substr(x = students$`Last Name`, 1, 1), students$`First Name`, sep
       pch = 11:16, col = 2:7)
```



This data layout exposes its limitations once the class advances to 7th grade, or if we were to obtain 3rd grade information. To accommodate such additional data, we would need to change the number and position of columns, not simply add additional rows. It is natural to make new observations or identify new samples (rows) but usually awkward to change the underlying variables (columns).

The particular class level (e.g. 4th grade) that a letter grade pertains to is, at heart, a value, not a variable. Another way to think of this is in terms of independent variables versus dependent variables, or in machine learning terms, features versus target. In some ways, the class level might correlate with or influence the resulting letter grade; perhaps the teachers at the different levels have different biases, or children of a certain age lose or gain interest in schoolwork, for example.

For most analytic purposes, this data would be more useful if we made it tidy (normalized) before further processing. In Base R, the `reshape2::melt()` method can perform this tidying. We pin some of the columns as id_vars, and we set a name for the combined columns as a variable and the letter grade as a single new column.

```
reshape2::melt(data = students, id=c("Last Name", "First Name"))
#>    Last Name First Name  variable value
#> 1    Johnson        Mia 4th Grade   4.0
#> 2      Lopez       Liam 4th Grade   3.0
#> 3        Lee   Isabella 4th Grade   2.0
#> 4     Fisher      Mason 4th Grade   3.0
#> 5      Gupta     Olivia 4th Grade   3.0
#> 6   Robinson     Sophia 4th Grade   4.3
#> 7    Johnson        Mia 5th Grade   3.3
#> 8      Lopez       Liam 5th Grade   3.0
#> 9        Lee   Isabella 5th Grade   1.7
#> 10    Fisher      Mason 5th Grade   2.7
#> 11     Gupta     Olivia 5th Grade   4.3
#> 12  Robinson     Sophia 5th Grade   2.7
#> 13   Johnson        Mia 6th Grade   3.7
#> 14     Lopez       Liam 6th Grade   4.3
#> 15       Lee   Isabella 6th Grade   2.7
#> 16    Fisher      Mason 6th Grade   2.3
#> 17     Gupta     Olivia 6th Grade   4.0
#> 18  Robinson     Sophia 6th Grade   4.0
```

Within the Tidyverse, specifically within the tidyr package, there is a function `pivot_longer()` that is similar to Base R's `reshape2::melt()`. The aggregation names and values have parameters spelled `names_to=` and `values_to=`, but the operation is the same:

```
s.l <- students %>%
 pivot_longer(c('4th Grade', '5th Grade', '6th Grade'),
 names_to = "Level",
 values_to = "Score")
s.l
#> # A tibble: 18 x 4
#>    `Last Name` `First Name` Level     Score
```

```
#>    <chr>       <chr>        <chr>     <dbl>
#>  1 Johnson     Mia          4th Grade   4
#>  2 Johnson     Mia          5th Grade   3.3
#>  3 Johnson     Mia          6th Grade   3.7
#>  4 Lopez       Liam         4th Grade   3
#>  5 Lopez       Liam         5th Grade   3
#>  6 Lopez       Liam         6th Grade   4.3
#>  7 Lee         Isabella     4th Grade   2
#>  8 Lee         Isabella     5th Grade   1.7
#>  9 Lee         Isabella     6th Grade   2.7
#> 10 Fisher      Mason        4th Grade   3
#> 11 Fisher      Mason        5th Grade   2.7
#> 12 Fisher      Mason        6th Grade   2.3
#> 13 Gupta       Olivia       4th Grade   3
#> 14 Gupta       Olivia       5th Grade   4.3
#> 15 Gupta       Olivia       6th Grade   4
#> 16 Robinson    Sophia       4th Grade   4.3
#> 17 Robinson    Sophia       5th Grade   2.7
#> 18 Robinson    Sophia       6th Grade   4
```

The simple example above gives you a first feel for tidying tabular data. To reverse the tidying operation that moves variables (columns) to values (rows), the `pivot_wider()` function in **tidyr** can be used. In Base R there are several related methods on data frames, including `reshape::cast()` and `reshape2::dcast()`.

```
s.l %>%
  pivot_wider(names_from = Level, values_from = Score)
#> # A tibble: 6 x 5
#>   `Last Name` `First Name` `4th Grade` `5th Grade`
#>   <chr>       <chr>              <dbl>       <dbl>
#> 1 Johnson     Mia                  4           3.3
#> 2 Lopez       Liam                 3           3
#> 3 Lee         Isabella             2           1.7
#> 4 Fisher      Mason                3           2.7
#> 5 Gupta       Olivia               3           4.3
#> 6 Robinson    Sophia               4.3         2.7
#> # ... with 1 more variable: 6th Grade <dbl>
```

5.1 The tibble

Tibbles inherits the attributes of a data frame and enhances some of them. The tibble is the central data structure for a set of packages known as the **tidyverse**.

Tibbles when printed out returns:

- the first 10 rows and
- all the columns that can fit on screen and
- column types.

5.1.1 Importing data

The functions `read_csv()`, `read_delim()`, `read_excel_csv()`, `read_tsv()` are used to import data.

```
# loading package
library(readr)

# reading data
gapminder <- read_delim(file = 'data/gapminder_ext_UTF-8.txt',
                        delim = "\t",
                        col_names = T,
                        locale = locale(decimal_mark = ",",  encoding = "UTF-8"))
head(gapminder, 3)
#> # A tibble: 3 x 8
#>   country     continent  year lifeExp      pop gdpPercap
#>   <chr>       <chr>     <dbl>   <dbl>    <dbl>     <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.
#> # ... with 2 more variables: country_hun <chr>,
#> #   continent_hun <chr>

# class checking
class(gapminder)
#> [1] "spec_tbl_df" "tbl_df"      "tbl"         "data.frame"

# checking for data frame
is.data.frame(gapminder)
#> [1] TRUE
```

5.1.2 Tibbles are data frames

Since Tibbles are data frames, functions which operate on data frames also operate on them.

```
head(gapminder, 3)
#> # A tibble: 3 x 8
#>   country     continent  year lifeExp      pop gdpPercap
#>   <chr>       <chr>     <dbl>   <dbl>    <dbl>     <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.
```

```
#> # ... with 2 more variables: country_hun <chr>,
#> #   continent_hun <chr>
tail(gapminder, 3)
#> # A tibble: 3 x 8
#>   country  continent  year lifeExp      pop gdpPercap
#>   <chr>    <chr>     <dbl>   <dbl>    <dbl>     <dbl>
#> 1 Zimbabwe Africa     1997    46.8 11404948      792.
#> 2 Zimbabwe Africa     2002    40.0 11926563      672.
#> 3 Zimbabwe Africa     2007    43.5 12311143      470.
#> # ... with 2 more variables: country_hun <chr>,
#> #   continent_hun <chr>
nrow(gapminder)
#> [1] 1704
ncol(gapminder)
#> [1] 8
summary(gapminder)
#>    country           continent              year
#>  Length:1704        Length:1704        Min.   :1952
#>  Class :character   Class :character   1st Qu.:1966
#>  Mode  :character   Mode  :character   Median :1980
#>                                        Mean   :1980
#>                                        3rd Qu.:1993
#>                                        Max.   :2007
#>     lifeExp          pop              gdpPercap
#>  Min.   :23.60   Min.   :6.001e+04   Min.   :   241.2
#>  1st Qu.:48.20   1st Qu.:2.794e+06   1st Qu.:  1202.1
#>  Median :60.71   Median :7.024e+06   Median :  3531.8
#>  Mean   :59.47   Mean   :2.960e+07   Mean   :  7215.3
#>  3rd Qu.:70.85   3rd Qu.:1.959e+07   3rd Qu.:  9325.5
#>  Max.   :82.60   Max.   :1.319e+09   Max.   :113523.1
#>  country_hun        continent_hun
#>  Length:1704        Length:1704
#>  Class :character   Class :character
#>  Mode  :character   Mode  :character
#>
#>
#>
```

5.1.3 Exporting data

The functions `write_csv()`, `write_delim()`, `write_excel_csv()`, `write_tsv()` are used to export data. To export Tibbles, they have first to be converted into data frames.

```
# exporting Tibbles
write_delim(x = data.frame(gapminder), delim = " ", file = 'output/data/gapminderfixedw
```

```
write_csv(x = data.frame(gapminder), file = 'output/data/gapminder_csv.txt')
write_tsv(x = data.frame(gapminder), file = 'output/data/gapminder_tsv.txt')

# checking if files exist?
file.exists(c('output/data/gapminderfixedwidth.txt', 'output/data/gapminder_csv.txt', 'output/dat
#> [1] TRUE TRUE TRUE

# removing files
file.remove(c('output/data/gapminderfixedwidth.txt', 'output/data/gapminder_csv.txt', 'output/dat
#> [1] TRUE TRUE TRUE
```

5.1.4 Check for tibble

Tibbles come from the package **tibble**.

The function `is_tibble()` is used to check for tibble. The function `glimpse()` is a better option of `str()`.

```
# loading tibble
library(tibble)

# glimpse() a better option to str()
glimpse(gapminder)
#> Rows: 1,704
#> Columns: 8
#> $ country       <chr> "Afghanistan", "Afghanistan", "Afgha~
#> $ continent     <chr> "Asia", "Asia", "Asia", "Asia", "Asi~
#> $ year          <dbl> 1952, 1957, 1962, 1967, 1972, 1977, ~
#> $ lifeExp       <dbl> 28.801, 30.332, 31.997, 34.020, 36.0~
#> $ pop           <dbl> 8425333, 9240934, 10267083, 11537966~
#> $ gdpPercap     <dbl> 779.4453, 820.8530, 853.1007, 836.19~
#> $ country_hun   <chr> "Afganisztán", "Afganisztán", "Afgan~
#> $ continent_hun <chr> "Ázsia", "Ázsia", "Ázsia", "Ázsia", ~

# checking whether an object is a tibble
is_tibble(gapminder)
#> [1] TRUE
```

5.1.5 Creating a tibble

The function `tibble()` is like `data.frame()` but creates a tibble.

```
# creating named vectors
country <- c('China', 'India', 'United States', 'Indonesia', 'Brazil',
             'Pakistan', 'Bangladesh', 'Nigeria', 'Japan', 'Mexico')
continent <- c('Asia', 'Asia', 'Americas', 'Asia', 'Americas',
               'Asia', 'Asia', 'Africa', 'Asia', 'Americas')
```

```
population <- c(1318683096, 1110396331, 301139947, 223547000, 190010647,
                169270617, 150448339, 135031164, 127467972, 108700891)
lifeExpectancy <- c(72.961, 64.698, 78.242, 70.65, 72.39,
                    65.483, 64.062, 46.859, 82.603, 76.195)
percapita <- c(4959, 2452, 42952, 3541, 9066, 2606, 1391, 2014, 31656, 11978)

# creating a tibble from named vectors
top_10 <- tibble(country, population, lifeExpectancy)
head(top_10, 3)
#> # A tibble: 3 x 3
#>   country       population lifeExpectancy
#>   <chr>              <dbl>          <dbl>
#> 1 China         1318683096           73.0
#> 2 India         1110396331           64.7
#> 3 United States  301139947           78.2
class(top_10)
#> [1] "tbl_df"     "tbl"        "data.frame"
```

5.1.6 Adding columns

The function `add_column()` is used to add columns to a tibble or data frames.

```
# adding a column to a tibble
# defaults to the last column
add_column(top_10, continent)
#> # A tibble: 10 x 4
#>    country       population lifeExpectancy continent
#>    <chr>              <dbl>          <dbl> <chr>
#>  1 China         1318683096           73.0 Asia
#>  2 India         1110396331           64.7 Asia
#>  3 United States  301139947           78.2 Americas
#>  4 Indonesia      223547000           70.6 Asia
#>  5 Brazil         190010647           72.4 Americas
#>  6 Pakistan       169270617           65.5 Asia
#>  7 Bangladesh     150448339           64.1 Asia
#>  8 Nigeria        135031164           46.9 Africa
#>  9 Japan          127467972           82.6 Asia
#> 10 Mexico         108700891           76.2 Americas

# also works for data frames
add_column(as.data.frame(top_10), continent)
#>          country population lifeExpectancy continent
#> 1          China 1318683096         72.961      Asia
#> 2          India 1110396331         64.698      Asia
#> 3  United States  301139947         78.242  Americas
```

```
#> 4      Indonesia  223547000         70.650      Asia
#> 5         Brazil  190010647         72.390  Americas
#> 6       Pakistan  169270617         65.483      Asia
#> 7     Bangladesh  150448339         64.062      Asia
#> 8        Nigeria  135031164         46.859    Africa
#> 9          Japan  127467972         82.603      Asia
#> 10        Mexico  108700891         76.195  Americas

# adding multiple columns
add_column(top_10, continent, percapita)
#> # A tibble: 10 x 5
#>    country       population lifeExpectancy continent percapita
#>    <chr>              <dbl>          <dbl> <chr>         <dbl>
#>  1 China         1318683096           73.0 Asia           4959
#>  2 India         1110396331           64.7 Asia           2452
#>  3 United States  301139947           78.2 Americas      42952
#>  4 Indonesia      223547000           70.6 Asia           3541
#>  5 Brazil         190010647           72.4 Americas       9066
#>  6 Pakistan       169270617           65.5 Asia           2606
#>  7 Bangladesh     150448339           64.1 Asia           1391
#>  8 Nigeria        135031164           46.9 Africa         2014
#>  9 Japan          127467972           82.6 Asia          31656
#> 10 Mexico         108700891           76.2 Americas      11978

# adding multiple columns directly
add_column(top_10,
           continent = c('Asia', 'Asia', 'Americas', 'Asia', 'Americas',
                         'Asia', 'Asia', 'Africa', 'Asia', 'Americas'),
           percapita = c(4959, 2452, 42952, 3541, 9066, 2606, 1391, 2014, 31656, 11978))
#> # A tibble: 10 x 5
#>    country       population lifeExpectancy continent percapita
#>    <chr>              <dbl>          <dbl> <chr>         <dbl>
#>  1 China         1318683096           73.0 Asia           4959
#>  2 India         1110396331           64.7 Asia           2452
#>  3 United States  301139947           78.2 Americas      42952
#>  4 Indonesia      223547000           70.6 Asia           3541
#>  5 Brazil         190010647           72.4 Americas       9066
#>  6 Pakistan       169270617           65.5 Asia           2606
#>  7 Bangladesh     150448339           64.1 Asia           1391
#>  8 Nigeria        135031164           46.9 Africa         2014
#>  9 Japan          127467972           82.6 Asia          31656
#> 10 Mexico         108700891           76.2 Americas      11978

# add a column before an index position
add_column(top_10, continent, .before = 2)
```

```
#> # A tibble: 10 x 4
#>    country       continent population lifeExpectancy
#>    <chr>         <chr>          <dbl>          <dbl>
#>  1 China         Asia      1318683096           73.0
#>  2 India         Asia      1110396331           64.7
#>  3 United States Americas   301139947           78.2
#>  4 Indonesia     Asia       223547000           70.6
#>  5 Brazil        Americas   190010647           72.4
#>  6 Pakistan      Asia       169270617           65.5
#>  7 Bangladesh    Asia       150448339           64.1
#>  8 Nigeria       Africa     135031164           46.9
#>  9 Japan         Asia       127467972           82.6
#> 10 Mexico        Americas   108700891           76.2

# add a column after an index position
top_10 <- add_column(top_10, continent, .after = 1)
top_10
#> # A tibble: 10 x 4
#>    country       continent population lifeExpectancy
#>    <chr>         <chr>          <dbl>          <dbl>
#>  1 China         Asia      1318683096           73.0
#>  2 India         Asia      1110396331           64.7
#>  3 United States Americas   301139947           78.2
#>  4 Indonesia     Asia       223547000           70.6
#>  5 Brazil        Americas   190010647           72.4
#>  6 Pakistan      Asia       169270617           65.5
#>  7 Bangladesh    Asia       150448339           64.1
#>  8 Nigeria       Africa     135031164           46.9
#>  9 Japan         Asia       127467972           82.6
#> 10 Mexico        Americas   108700891           76.2
```

5.1.7 Adding rows

The function `add_row()` is used to add rows to a tibble or a data frame.

```
# adding a row
# defaults to the tail of the data frame
add_row(top_10,
        country = 'Philippines',
        continent = 'Asia',
        population = 91077287,
        lifeExpectancy = 71.688)
#> # A tibble: 11 x 4
#>    country       continent population lifeExpectancy
#>    <chr>         <chr>          <dbl>          <dbl>
#>  1 China         Asia      1318683096           73.0
```

```
#>  2 India         Asia      1110396331          64.7
#>  3 United States Americas   301139947          78.2
#>  4 Indonesia     Asia       223547000          70.6
#>  5 Brazil        Americas   190010647          72.4
#>  6 Pakistan      Asia       169270617          65.5
#>  7 Bangladesh    Asia       150448339          64.1
#>  8 Nigeria       Africa     135031164          46.9
#>  9 Japan         Asia       127467972          82.6
#> 10 Mexico        Americas   108700891          76.2
#> 11 Philippines   Asia        91077287          71.7

# adding rows before an index position
add_row(top_10,
        country = 'Philippines',
        continent = 'Asia',
        population = 91077287,
        lifeExpectancy = 71.688,
        .before = 2)
#> # A tibble: 11 x 4
#>    country       continent population lifeExpectancy
#>    <chr>         <chr>          <dbl>          <dbl>
#>  1 China         Asia      1318683096          73.0
#>  2 Philippines   Asia        91077287          71.7
#>  3 India         Asia      1110396331          64.7
#>  4 United States Americas   301139947          78.2
#>  5 Indonesia     Asia       223547000          70.6
#>  6 Brazil        Americas   190010647          72.4
#>  7 Pakistan      Asia       169270617          65.5
#>  8 Bangladesh    Asia       150448339          64.1
#>  9 Nigeria       Africa     135031164          46.9
#> 10 Japan         Asia       127467972          82.6
#> 11 Mexico        Americas   108700891          76.2

# adding rows after an index position
add_row(top_10,
        country = 'Philippines',
        continent = 'Asia',
        population = 91077287,
        lifeExpectancy = 71.688,
        .after = 2)
#> # A tibble: 11 x 4
#>    country       continent population lifeExpectancy
#>    <chr>         <chr>          <dbl>          <dbl>
#>  1 China         Asia      1318683096          73.0
#>  2 India         Asia      1110396331          64.7
```

```
#>  3 Philippines   Asia        91077287          71.7
#>  4 United States Americas   301139947          78.2
#>  5 Indonesia     Asia       223547000          70.6
#>  6 Brazil        Americas   190010647          72.4
#>  7 Pakistan      Asia       169270617          65.5
#>  8 Bangladesh    Asia       150448339          64.1
#>  9 Nigeria       Africa     135031164          46.9
#> 10 Japan         Asia       127467972          82.6
#> 11 Mexico        Americas   108700891          76.2

# adding multiple rows
add_row(top_10,
        country = c('Philippines', 'Vietnam', 'Germany', 'Egypt', 'Ethiopia',
                    'Turkey', 'Iran', 'Thailand', 'Congo, Dem. Rep.', 'France'),
        continent = c('Asia', 'Asia', 'Europe', 'Africa', 'Africa',
                      'Europe', 'Asia', 'Asia', 'Africa', 'Europe'),
        population = c(91077287, 85262356, 82400996, 80264543, 76511887,
                       71158647, 69453570, 65068149, 64606759, 61083916),
        lifeExpectancy = c(71.688, 74.249, 79.406, 71.338, 52.947,
                           71.777, 70.964, 70.616, 46.462, 80.657)
       )
#> # A tibble: 20 x 4
#>    country          continent population lifeExpectancy
#>    <chr>            <chr>          <dbl>          <dbl>
#>  1 China            Asia      1318683096           73.0
#>  2 India            Asia      1110396331           64.7
#>  3 United States    Americas   301139947           78.2
#>  4 Indonesia        Asia       223547000           70.6
#>  5 Brazil           Americas   190010647           72.4
#>  6 Pakistan         Asia       169270617           65.5
#>  7 Bangladesh       Asia       150448339           64.1
#>  8 Nigeria          Africa     135031164           46.9
#>  9 Japan            Asia       127467972           82.6
#> 10 Mexico           Americas   108700891           76.2
#> 11 Philippines      Asia        91077287           71.7
#> 12 Vietnam          Asia        85262356           74.2
#> 13 Germany          Europe      82400996           79.4
#> 14 Egypt            Africa      80264543           71.3
#> 15 Ethiopia         Africa      76511887           52.9
#> 16 Turkey           Europe      71158647           71.8
#> 17 Iran             Asia        69453570           71.0
#> 18 Thailand         Asia        65068149           70.6
#> 19 Congo, Dem. Rep. Africa      64606759           46.5
#> 20 France           Europe      61083916           80.7
```

5.1.8 Converting to tibble

The function `as_tibble()` is used to convert to a tibble, if possible.

```
# creating a matrix
mat = matrix(seq(1,12), 3, 4,
             dimnames = list('a' = c('a1', 'a2', 'a3'), 'b' = c('b1', 'b2', 'b3', 'b4')))
mat
#>     b
#> a    b1 b2 b3 b4
#>   a1  1  4  7 10
#>   a2  2  5  8 11
#>   a3  3  6  9 12

# converting a matrix to tibble
# removes the rownames
mat_tbl <- as_tibble(mat)
mat_tbl
#> # A tibble: 3 x 4
#>      b1    b2    b3    b4
#>   <int> <int> <int> <int>
#> 1     1     4     7    10
#> 2     2     5     8    11
#> 3     3     6     9    12
class(mat_tbl)
#> [1] "tbl_df"     "tbl"        "data.frame"

# creating a data frame
top_10_df <- data.frame(
country = c('China', 'India', 'United States', 'Indonesia', 'Brazil',
            'Pakistan', 'Bangladesh', 'Nigeria', 'Japan', 'Mexico'),
continent = c('Asia', 'Asia', 'Americas', 'Asia', 'Americas',
              'Asia', 'Asia', 'Africa', 'Asia', 'Americas'),
population = c(1318683096, 1110396331, 301139947, 223547000, 190010647,
               169270617, 150448339, 135031164, 127467972, 108700891),
lifeExpectancy = c(72.961, 64.698, 78.242, 70.65, 72.39,
                   65.483, 64.062, 46.859, 82.603, 76.195)
    )
head(top_10_df, 3)
#>         country continent population lifeExpectancy
#> 1         China      Asia 1318683096         72.961
#> 2         India      Asia 1110396331         64.698
#> 3 United States  Americas  301139947         78.242
class(top_10_df)
#> [1] "data.frame"
```

```
# converting data frame to tibble
top_tbl <- as_tibble(top_10_df)
top_tbl
#> # A tibble: 10 x 4
#>    country       continent population lifeExpectancy
#>    <chr>         <chr>          <dbl>          <dbl>
#>  1 China         Asia      1318683096           73.0
#>  2 India         Asia      1110396331           64.7
#>  3 United States Americas   301139947           78.2
#>  4 Indonesia     Asia       223547000           70.6
#>  5 Brazil        Americas   190010647           72.4
#>  6 Pakistan      Asia       169270617           65.5
#>  7 Bangladesh    Asia       150448339           64.1
#>  8 Nigeria       Africa     135031164           46.9
#>  9 Japan         Asia       127467972           82.6
#> 10 Mexico        Americas   108700891           76.2
class(top_tbl)
#> [1] "tbl_df"     "tbl"        "data.frame"
```

5.1.9 Manipulating row names

Tibble does not support row names but the package tibble has the following functions for dealing with row names:

- `has_rownames()` checks if a data frame has row names.
- `remove_rownames()` removes row names.
- `column_to_rownames()` moves a column to row names.
- `rowid_to_column()` moves a row index to column.

```
# creating a data frame
top_10_df <- data.frame(
continent = c('Asia', 'Asia', 'Americas', 'Asia', 'Americas',
              'Asia', 'Asia', 'Africa', 'Asia', 'Americas'),
population = c(1318683096, 1110396331, 301139947, 223547000, 190010647,
               169270617, 150448339, 135031164, 127467972, 108700891),
lifeExpectancy = c(72.961, 64.698, 78.242, 70.65, 72.39,
                   65.483, 64.062, 46.859, 82.603, 76.195)
    )
top_10_df
#>   continent population lifeExpectancy
#> 1      Asia 1318683096         72.961
#> 2      Asia 1110396331         64.698
#> 3  Americas  301139947         78.242
#> 4      Asia  223547000         70.650
#> 5  Americas  190010647         72.390
#> 6      Asia  169270617         65.483
```

```
#> 7       Asia  150448339         64.062
#> 8     Africa  135031164         46.859
#> 9       Asia  127467972         82.603
#> 10  Americas  108700891         76.195

# vector of country names
country <- c('China', 'India', 'United States', 'Indonesia', 'Brazil',
             'Pakistan', 'Bangladesh', 'Nigeria', 'Japan', 'Mexico')

# adding row names
rownames(top_10_df) <- country
top_10_df
#>               continent population lifeExpectancy
#> China              Asia 1318683096         72.961
#> India              Asia 1110396331         64.698
#> United States  Americas  301139947         78.242
#> Indonesia          Asia  223547000         70.650
#> Brazil         Americas  190010647         72.390
#> Pakistan           Asia  169270617         65.483
#> Bangladesh         Asia  150448339         64.062
#> Nigeria          Africa  135031164         46.859
#> Japan              Asia  127467972         82.603
#> Mexico         Americas  108700891         76.195

# check if the data frame contains row names
has_rownames(top_10_df)
#> [1] TRUE

# delete row names
remove_rownames(top_10_df)
#>    continent population lifeExpectancy
#> 1       Asia 1318683096         72.961
#> 2       Asia 1110396331         64.698
#> 3   Americas  301139947         78.242
#> 4       Asia  223547000         70.650
#> 5   Americas  190010647         72.390
#> 6       Asia  169270617         65.483
#> 7       Asia  150448339         64.062
#> 8     Africa  135031164         46.859
#> 9       Asia  127467972         82.603
#> 10  Americas  108700891         76.195

# convert row names to a column
top_10_df <- rownames_to_column(top_10_df, var = "country")
top_10_df
```

```
#>          country continent population lifeExpectancy
#> 1          China      Asia 1318683096         72.961
#> 2          India      Asia 1110396331         64.698
#> 3  United States  Americas  301139947         78.242
#> 4      Indonesia      Asia  223547000         70.650
#> 5         Brazil  Americas  190010647         72.390
#> 6       Pakistan      Asia  169270617         65.483
#> 7     Bangladesh      Asia  150448339         64.062
#> 8        Nigeria    Africa  135031164         46.859
#> 9          Japan      Asia  127467972         82.603
#> 10        Mexico  Americas  108700891         76.195

# convert a column to row names
column_to_rownames(top_10_df, var = "country")
#>               continent population lifeExpectancy
#> China              Asia 1318683096         72.961
#> India              Asia 1110396331         64.698
#> United States  Americas  301139947         78.242
#> Indonesia          Asia  223547000         70.650
#> Brazil         Americas  190010647         72.390
#> Pakistan           Asia  169270617         65.483
#> Bangladesh         Asia  150448339         64.062
#> Nigeria          Africa  135031164         46.859
#> Japan              Asia  127467972         82.603
#> Mexico         Americas  108700891         76.195

# convert row index to a column
rowid_to_column(top_10_df, var = "rank")
#>    rank       country continent population lifeExpectancy
#> 1     1         China      Asia 1318683096         72.961
#> 2     2         India      Asia 1110396331         64.698
#> 3     3 United States  Americas  301139947         78.242
#> 4     4     Indonesia      Asia  223547000         70.650
#> 5     5        Brazil  Americas  190010647         72.390
#> 6     6      Pakistan      Asia  169270617         65.483
#> 7     7    Bangladesh      Asia  150448339         64.062
#> 8     8       Nigeria    Africa  135031164         46.859
#> 9     9         Japan      Asia  127467972         82.603
#> 10   10        Mexico  Americas  108700891         76.195
```

5.2 Manipulating categorical data with forcats

The package **forcats** comes with a series of functions all beginning with fct_ for working with categorical data. This package is developed and maintained by Hadley Wickham and is part of the tidyverse universe of packages.

Categorical data in R is represented by factors.

```
# install.packages(forcats)
library(forcats)
library(gapminder)
# loading data
data(gapminder)
# preparing data
gapminder_2007 <- subset(gapminder, year == 2007, -3)
head(gapminder_2007)
#> # A tibble: 6 x 5
#>   country     continent lifeExp      pop gdpPercap
#>   <fct>       <fct>       <dbl>    <int>     <dbl>
#> 1 Afghanistan Asia         43.8 31889923      975.
#> 2 Albania     Europe       76.4  3600523     5937.
#> 3 Algeria     Africa       72.3 33333216     6223.
#> 4 Angola      Africa       42.7 12420476     4797.
#> 5 Argentina   Americas     75.3 40301927    12779.
#> 6 Australia   Oceania      81.2 20434176    34435.
sapply(gapminder_2007, class)
#>   country continent   lifeExp       pop gdpPercap
#>  "factor"  "factor" "numeric" "integer" "numeric"
```

5.2.1 Inspecting factors

5.2.1.1 Get categories

The functions `levels()` and `fct_unique()` are used to get levels or categories.

```
# get levels using base R
levels(gapminder_2007$continent)
#> [1] "Africa"   "Americas" "Asia"     "Europe"   "Oceania"

# get levels using forcats
fct_unique(gapminder_2007$continent)
#> [1] Africa   Americas Asia     Europe   Oceania
#> Levels: Africa Americas Asia Europe Oceania
```

5.2.1.2 Get the number of categories

The functions `nlevels()` and `length(fct_unique())` are used to get the number of categories or levels.

```
# get the number of categories using base R
nlevels(gapminder_2007$continent)
#> [1] 5
```

```
# get the number of categories using forcats
length(fct_unique(gapminder_2007$continent))
#> [1] 5
```

5.2.1.3 Count of values by categories

The function `table()` and `fct_count()` are used to get count of values by categories with the later returning a tibble.

```
# count of elements by categories using base R
table(gapminder_2007$continent)
#>
#>   Africa Americas     Asia   Europe  Oceania
#>       52       25       33       30        2

# count of elements by categories using forcats
fct_count(gapminder_2007$continent)
#> # A tibble: 5 x 2
#>   f            n
#>   <fct>    <int>
#> 1 Africa      52
#> 2 Americas    25
#> 3 Asia        33
#> 4 Europe      30
#> 5 Oceania      2
```

5.2.1.4 Reordering levels

```
# get levels
table(gapminder_2007$continent)
#>
#>   Africa Americas     Asia   Europe  Oceania
#>       52       25       33       30        2
```

5.2.1.4.1 Manually reordering levels The function `fct_relevel()` is used to manually reorder levels.

```
# manually reorder levels
gapminder_2007$continent <- fct_relevel(gapminder_2007$continent,
                                        c('Asia', 'Africa', 'Americas', 'Europe', 'Ocea
table(gapminder_2007$continent)
#>
#>     Asia   Africa Americas   Europe  Oceania
#>       33       52       25       30        2
# oceania first
```

```
gapminder_2007$continent <- fct_relevel(gapminder_2007$continent, 'Oceania')
table(gapminder_2007$continent)
#>
#>  Oceania     Asia   Africa Americas   Europe
#>        2       33       52       25       30
```

5.2.1.5 Reordering levels by frequency of occurrence

The function `fct_infreq()` reorders levels by the number of times they occur in the data with the highest first.

```
# ordering levels by the frequency they appear in a dataset
gapminder_2007$continent <- fct_infreq(gapminder_2007$continent, ordered = NA)
table(gapminder_2007$continent)
#>
#>   Africa     Asia   Europe Americas  Oceania
#>       52       33       30       25        2
```

The argument `ordered = TRUE` returns an ordered factor.

```
# unordered factor
class(fct_infreq(gapminder_2007$continent, ordered = NA))
#> [1] "factor"

# ordered factor
class(fct_infreq(gapminder_2007$continent, ordered = TRUE))
#> [1] "ordered" "factor"
```

5.2.2 Reordering levels by their order in data

The function `fct_inorder()` reorders levels by the order in which they appear in the data set.

```
# ordering levels by the order in which they appear in a dataset
gapminder_2007$continent <- fct_inorder(gapminder_2007$continent, ordered = NA)
table(gapminder_2007$continent)
#>
#>     Asia   Europe   Africa Americas  Oceania
#>       33       30       52       25        2
```

5.2.2.1 Reversing the order

The function `fct_rev()` reverses the order of the levels.

```
# reversing level order
gapminder_2007$continent <- fct_rev(gapminder_2007$continent)
table(gapminder_2007$continent)
#>
```

```
#>  Oceania Americas   Africa   Europe     Asia
#>        2       25       52       30       33
```

5.2.2.2 Random order

The function `fct_shuffle()` randomly shuffles levels.

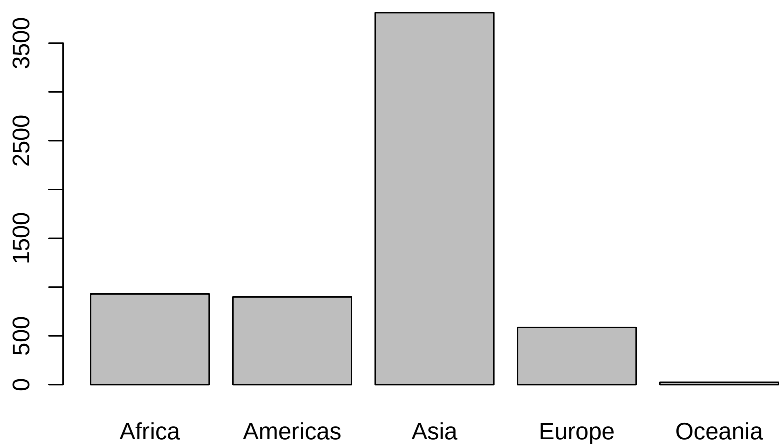
```
# randomly shuffling level order
gapminder_2007$continent <- fct_shuffle(gapminder_2007$continent)
table(gapminder_2007$continent)
#>
#>     Asia  Oceania   Europe Americas   Africa
#>       33        2       30       25       52
```

5.2.2.3 Reordering level by another column

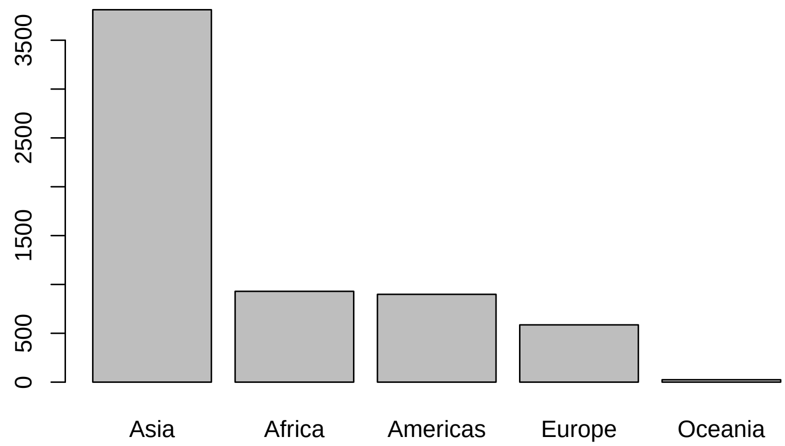
The function `fct_reorder()` reorders levels by another column or vector.

```
# ordering levels by another column
gapminder_2007$continent <-
  fct_reorder(gapminder_2007$continent, gapminder_2007$pop, .fun = sum, .desc = TRUE)
levels(gapminder_2007$continent)
#> [1] "Asia"     "Africa"   "Americas" "Europe"   "Oceania"


# using median
gapminder_2007$continent <-
  fct_reorder(gapminder_2007$continent, gapminder_2007$pop, .fun = median, .desc = TRU
levels(gapminder_2007$continent)
#> [1] "Asia"     "Oceania"  "Africa"   "Europe"   "Americas"

# ascending
gapminder_2007$continent <-
  fct_reorder(gapminder_2007$continent, gapminder_2007$pop, .fun = median, .desc = FALS
levels(gapminder_2007$continent)
#> [1] "Americas" "Europe"   "Africa"   "Oceania"  "Asia"

# population by continent
(pop_cont <- aggregate(pop ~ continent, gapminder, sum, subset = year == 2007))
#>   continent        pop
#> 1    Africa  929539692
#> 2  Americas  898871184
#> 3      Asia 3811953827
#> 4    Europe  586098529
#> 5   Oceania   24549947

# plotting a barchart
```

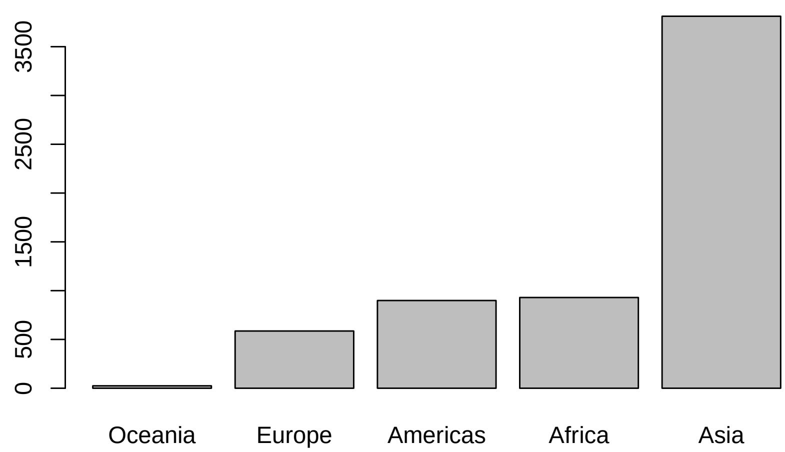
```
with(pop_cont, barplot(pop/1e6, names.arg = continent))
```




```
# reordering continent by population
pop_cont$continent <- fct_reorder(pop_cont$continent, pop_cont$pop, .desc = TRUE)
levels(pop_cont$continent)
#> [1] "Asia"     "Africa"   "Americas" "Europe"   "Oceania"

# sorting data frame by continent
pop_cont <- with(pop_cont, pop_cont[order(continent),])
pop_cont
#>   continent        pop
#> 3      Asia 3811953827
#> 1    Africa  929539692
#> 2  Americas  898871184
#> 4    Europe  586098529
#> 5   Oceania   24549947

# plotting barplot
with(pop_cont, barplot(pop/1e6, names.arg = continent))
```




```
# producing an ascending bar chart
pop_cont$continent <- fct_reorder(pop_cont$continent, pop_cont$pop, .desc = FALSE)
pop_cont <- with(pop_cont, pop_cont[order(continent),])
with(pop_cont, barplot(pop/1e6, names.arg = continent))
```



5.2.3 Restructuring levels and their labels

5.2.3.1 Renaming labels

The function `fct_recode()` is used to rename levels. It takes the form `new_name = old_name`.

```
levels(fct_recode(gapminder_2007$continent, 'AS' = 'Asia', 'Af' = 'Africa', 'Eu' = 'Europe'))
#> [1] "Americas" "Eu"       "Af"       "Oceania"  "AS"
```

5.2.3.2 collapsing levels

The function `fct_collapse()` is used to collapse levels into a new one.

```
# collapsing europe and africa into euroafrica
gapminder_2007$continent <-
  fct_collapse(gapminder_2007$continent, Euroafrica = c('Africa', 'Europe'))
table(gapminder_2007$continent)
#>
#>   Americas Euroafrica    Oceania       Asia
#>         25         82          2         33

# population by continent
(pop_cont <- aggregate(pop ~ continent, gapminder_2007, sum))
#>    continent        pop
#> 1   Americas  898871184
#> 2 Euroafrica 1515638221
#> 3    Oceania   24549947
#> 4       Asia 3811953827
```

5.2.3.3 combining levels

The functions `fct_lump()` and `fct_lump_min()` combines levels together based on the frequency of occurrence of each level.

```
# combining the least frequent levels
gapminder_2007 <- subset(gapminder, year == 2007, -3)
table(fct_lump(gapminder_2007$continent))
#>
#> Africa   Asia Europe  Other
#>     52     33     30     27
```

Using the arguments `n=` and `p=` we can specify the type of combining to perform; with positive values indicating combining rarest levels while negative values indicate combining most common levels.

```
# combining all except the first most common
table(fct_lump(gapminder_2007$continent, n = 1))
#>
```

```
#> Africa  Other
#>     52     90

# combining all except the first 2 most common
table(fct_lump(gapminder_2007$continent, n = 2))
#>
#> Africa   Asia  Other
#>     52     33     57

# combining all except the first 3 most common
table(fct_lump(gapminder_2007$continent, n = 3))
#>
#> Africa   Asia Europe  Other
#>     52     33     30     27

# combining all except the first rarest
table(fct_lump(gapminder_2007$continent, n = -1))
#>
#> Oceania   Other
#>       2     140

# combining all except the first 2 rarest
table(fct_lump(gapminder_2007$continent, n = -2))
#>
#> Americas  Oceania    Other
#>       25        2      115

# combining all except the first 3 rarest
table(fct_lump(gapminder_2007$continent, n = -3))
#>
#> Americas   Europe  Oceania    Other
#>       25       30        2       85

# using prop positive
table(fct_lump(gapminder_2007$continent, prop = 0.25))
#>
#> Africa  Other
#>     52     90
table(fct_lump(gapminder_2007$continent, prop = 0.22))
#>
#> Africa   Asia  Other
#>     52     33     57
table(fct_lump(gapminder_2007$continent, prop = 0.2))
#>
#> Africa   Asia Europe  Other
```

```
#>     52     33     30     27

# using prop negative
table(fct_lump(gapminder_2007$continent, prop = -0.25))
#>
#> Americas     Asia   Europe  Oceania    Other
#>       25       33       30        2       52
table(fct_lump(gapminder_2007$continent, prop = -0.22))
#>
#> Americas   Europe  Oceania    Other
#>       25       30        2       85
table(fct_lump(gapminder_2007$continent, prop = -0.2))
#>
#> Americas  Oceania    Other
#>       25        2      115
```

With `fct_lump_min()` combining is done based on whether a threshold declared by the min argument is met.

```
table(gapminder_2007$continent)
#>
#>   Africa Americas     Asia   Europe  Oceania
#>       52       25       33       30        2

# combining levels with less than 25 counts
table(fct_lump_min(gapminder_2007$continent, min = 25))
#>
#>   Africa Americas     Asia   Europe    Other
#>       52       25       33       30        2

# combining levels with less than 30 counts
table(fct_lump_min(gapminder_2007$continent, min = 30))
#>
#> Africa   Asia Europe  Other
#>     52     33     30     27

# combining levels with less than 33 counts
table(fct_lump_min(gapminder_2007$continent, min = 33))
#>
#> Africa   Asia  Other
#>     52     33     57
```

5.2.4 Remove and add levels

5.2.4.1 dropping levels

The function `fct_other()` will drop levels and replace them with the argument other_level = other by default.

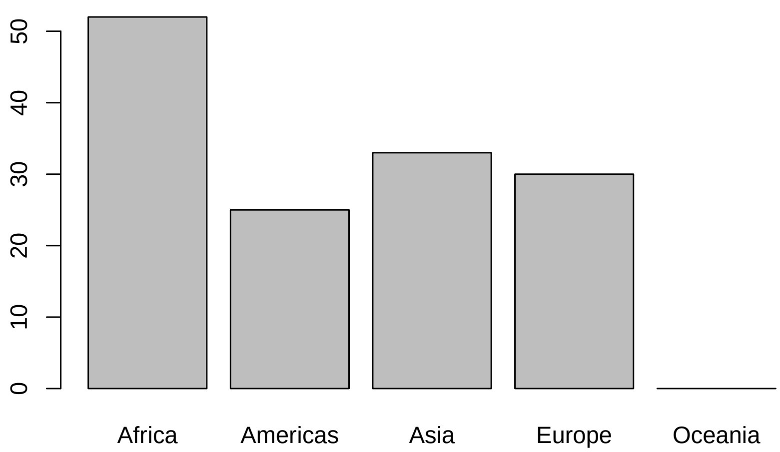
```
# keeping asia and europe
table(fct_other(gapminder_2007$continent, keep = c('Asia', 'Europe')))
#>
#>   Asia Europe  Other
#>     33     30     79

# dropping asia and europe
table(fct_other(gapminder_2007$continent, drop = c('Asia', 'Europe')))
#>
#>   Africa Americas  Oceania    Other
#>       52       25        2       63

# replacing other continents with nonEurasia
table(fct_other(gapminder_2007$continent,
                keep = c('Asia', 'Europe'),
                other_level = 'nonEurasia'))
#>
#>       Asia     Europe nonEurasia
#>         33         30         79

# replacing europe and asia with Eurasia
table(fct_other(gapminder_2007$continent,
                drop = c('Asia', 'Europe'),
                other_level = 'Eurasia'))
#>
#>   Africa Americas  Oceania  Eurasia
#>       52       25        2       63
```

5.2.5 dropping unused levels

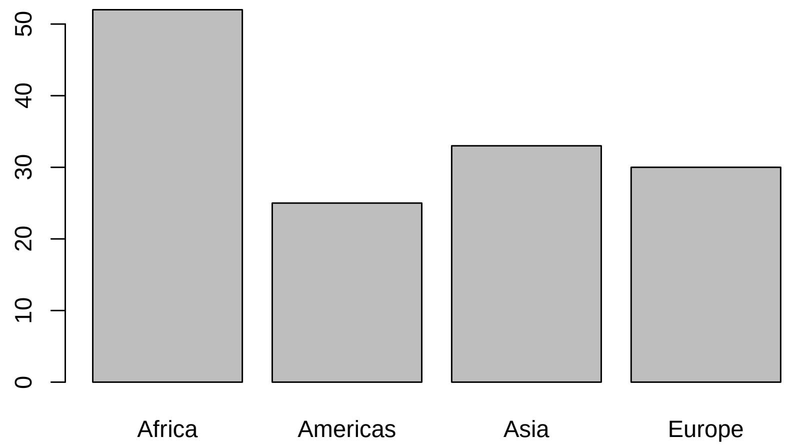
The function `fct_drop()` is used to drop unused levels. Unused levels are usually a problem while plotting as they appear on the graph though they contain no data.

```
# dropping Oceania
gapminder_oc <- subset(gapminder_2007, continent != 'Oceania')
table(gapminder_oc$continent)
#>
#>   Africa Americas     Asia   Europe  Oceania
#>       52       25       33       30        0
# Because the level Oceania has not been dropped, it appears on the above plot.
```

```
plot(gapminder_oc$continent)
```



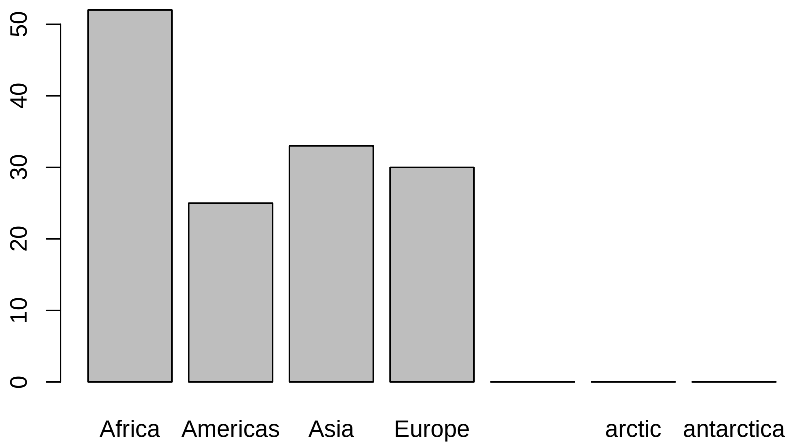

```
# dropping unused level
table(fct_drop(gapminder_oc$continent))
#>
#>   Africa Americas     Asia   Europe
#>       52       25       33       30
plot(fct_drop(gapminder_oc$continent))
```



5.2.5.1 adding levels

The function `fct_expand()` is used to add levels.

```
# adding the level arctic
table(fct_expand(gapminder_oc$continent, 'arctic'))
#>
#>   Africa Americas     Asia   Europe  Oceania   arctic
#>       52       25       33       30        0        0

# adding the levels arctic and antarctica
table(fct_expand(gapminder_oc$continent, c('arctic', 'antarctica')))
#>
#>     Africa   Americas       Asia     Europe    Oceania
#>         52         25         33         30          0
#>     arctic antarctica
#>          0          0
# newly added levels appear on the plot though they have no data
plot(fct_expand(gapminder_oc$continent, c('arctic', 'antarctica')))
```



5.3 Data Manipulation with dplyr and tidyr

The package **dplyr** is one of the core packages in a group of packages known as the tidyverse. It was developed and released in 2014 by Hadley Wickham and others. **dplyr** is meant to be for data manipulation what **ggplot2** is for data visualization, that is the grammar of data manipulation. It focuses solely on data frame manipulation and transformation using a set of verbs (functions) which are consistent and easy to understand.

Since **dyplr** belongs to the tidyverse world, it can be installed either by installing tidyverse or by installing **dplyr** itself.

5.3.1 Rename columns and rows

5.3.1.1 Renaming columns

The function `rename()` is used to rename columns.

`rename(new_name = old_name)`

```
library(readr)
library(dplyr)
library(gapminder)

# loading data
data(gapminder)
```

```
# get column names
names(gapminder)
#> [1] "country"   "continent" "year"      "lifeExp"
#> [5] "pop"       "gdpPercap"

# set column names
gapminder <- rename(gapminder,
                    Country = country,
                    Continent = continent,
                    Year = year,
                    `Life Expectancy` = lifeExp,
                    Population = pop,
                    `GDP per Capita` = gdpPercap)
# get column names
colnames(gapminder)
#> [1] "Country"         "Continent"       "Year"
#> [4] "Life Expectancy" "Population"      "GDP per Capita"
```

5.3.1.2 Renaming rows

Tibble does not support row names. See this.

5.3.2 Select columns and filter rows

5.3.2.1 Selecting and dropping columns

The function `select()` is used to select and rename columns.

```
# preparing data
column_names <- c('Rank', 'Title', 'Genre', 'Description', 'Director', 'Actors',
                  'Year', 'Runtime', 'Rating', 'Votes', 'Revenue', 'Metascore')
mov <- read.table(file = "data/IMDB-Movie-Data.csv", header = T, sep = ",", dec = ".",
                  comment.char = "")
head(mov, 3)
#>   Rank                   Title                    Genre
#> 1    1 Guardians of the Galaxy  Action,Adventure,Sci-Fi
#> 2    2              Prometheus Adventure,Mystery,Sci-Fi
#> 3    3                   Split          Horror,Thriller
#>
#> 1                               A group of intergalactic criminals are forced to wo
#> 2                               Following clues to the origin of mankind, a team fi
#> 3 Three girls are kidnapped by a man with a diagnosed 23 distinct personalities. Th
#>             Director
#> 1         James Gunn
#> 2       Ridley Scott
#> 3 M. Night Shyamalan
```

```
#>                                                                      Actors
#> 1                    Chris Pratt, Vin Diesel, Bradley Cooper, Zoe Saldana
#> 2 Noomi Rapace, Logan Marshall-Green, Michael Fassbender, Charlize Theron
#> 3        James McAvoy, Anya Taylor-Joy, Haley Lu Richardson, Jessica Sula
#>   Year Runtime..Minutes. Rating  Votes Revenue..Millions.
#> 1 2014               121    8.1 757074             333.13
#> 2 2012               124    7.0 485820             126.46
#> 3 2016               117    7.3 157606             138.12
#>   Metascore
#> 1        76
#> 2        65
#> 3        62
names(mov) <- c('Rank', 'Title', 'Genre', 'Description', 'Director', 'Actors',
                  'Year', 'Runtime', 'Rating', 'Votes', 'Revenue', 'Metascore')


# selecting columns by column names
movies <- select(mov, c('Title', 'Year', 'Revenue', 'Metascore'))
head(movies, 3)
#>                     Title Year Revenue Metascore
#> 1 Guardians of the Galaxy 2014  333.13        76
#> 2              Prometheus 2012  126.46        65
#> 3                   Split 2016  138.12        62

# columns can be passed directly without quotation marks
movies <- select(mov, Title, Year, Revenue, Metascore)
head(movies, 3)
#>                     Title Year Revenue Metascore
#> 1 Guardians of the Galaxy 2014  333.13        76
#> 2              Prometheus 2012  126.46        65
#> 3                   Split 2016  138.12        62

# renaming column
movies <- select(mov,
                 Title,
                 `Release Year` = Year,
                 `Revenue in Millions` = Revenue,
                 Metascore)
head(movies, 3)
#>                     Title Release Year Revenue in Millions
#> 1 Guardians of the Galaxy         2014              333.13
#> 2              Prometheus         2012              126.46
#> 3                   Split         2016              138.12
#>   Metascore
#> 1        76
```

```
#> 2        65
#> 3        62

# selecting columns by position
movies <- select(mov, 2, 7, 11, 12)
head(movies, 3)
#>                     Title Year Revenue Metascore
#> 1 Guardians of the Galaxy 2014  333.13        76
#> 2              Prometheus 2012  126.46        65
#> 3                   Split 2016  138.12        62

# selecting columns by sequencing
movies <- select(mov, 7:12)
head(movies, 3)
#>   Year Runtime Rating  Votes Revenue Metascore
#> 1 2014     121    8.1 757074  333.13        76
#> 2 2012     124    7.0 485820  126.46        65
#> 3 2016     117    7.3 157606  138.12        62

# : works with column names
movies <- select(mov, Year:Metascore)
head(movies, 3)
#>   Year Runtime Rating  Votes Revenue Metascore
#> 1 2014     121    8.1 757074  333.13        76
#> 2 2012     124    7.0 485820  126.46        65
#> 3 2016     117    7.3 157606  138.12        62

# dropping columns by column names
movies <- select(mov, -Rank, -Genre, -Description,
                 -Director, -Actors, -Runtime, -Rating, -Votes)
head(movies, 3)
#>                     Title Year Revenue Metascore
#> 1 Guardians of the Galaxy 2014  333.13        76
#> 2              Prometheus 2012  126.46        65
#> 3                   Split 2016  138.12        62

# dropping columns by sequence
movies <- select(mov, -(1:6))
head(movies, 3)
#>   Year Runtime Rating  Votes Revenue Metascore
#> 1 2014     121    8.1 757074  333.13        76
#> 2 2012     124    7.0 485820  126.46        65
#> 3 2016     117    7.3 157606  138.12        62

movies <- select(mov, -(Rank:Actors))
```

```
head(movies, 3)
#>   Year Runtime Rating  Votes Revenue Metascore
#> 1 2014     121    8.1 757074  333.13        76
#> 2 2012     124    7.0 485820  126.46        65
#> 3 2016     117    7.3 157606  138.12        62

# dropping columns by index position
movies <- select(mov, -c(1, 3, 4, 5, 6, 8, 9, 10))
head(movies, 3)
#>                     Title Year Revenue Metascore
#> 1 Guardians of the Galaxy 2014  333.13        76
#> 2              Prometheus 2012  126.46        65
#> 3                   Split 2016  138.12        62

# dropping columns by index position
movies <- select(mov, -1, -3, -4, -5, -6, -8, -9, -10)
head(movies, 3)
#>                     Title Year Revenue Metascore
#> 1 Guardians of the Galaxy 2014  333.13        76
#> 2              Prometheus 2012  126.46        65
#> 3                   Split 2016  138.12        62
```

5.3.2.2 Selecting column based on a condition

The functions `starts_with()`, `ends_with()`, `matches()`, and `contains()` are used to select columns based on a specific pattern. The function

- `starts_with()`: returns columns that start with a specific prefix
- `ends_with()`: returns columns that end with a specific suffix
- `matches()`: returns columns that match a particular regex pattern
- `contains()`: returns columns that contain a particular string

```
# selecting columns starting with R
movies <- select(mov, starts_with('R'))
head(movies, 3)
#>   Rank Runtime Rating Revenue
#> 1    1     121    8.1  333.13
#> 2    2     124    7.0  126.46
#> 3    3     117    7.3  138.12

# selecting columns starting with R and D
movies <- select(mov, starts_with(c('R', 'M')))
head(movies)
#>   Rank Runtime Rating Revenue Metascore
#> 1    1     121    8.1  333.13        76
#> 2    2     124    7.0  126.46        65
```

```
#> 3    3     117    7.3  138.12        62
#> 4    4     108    7.2  270.32        59
#> 5    5     123    6.2  325.02        40
#> 6    6     103    6.1   45.13        42

# selecting columns containing ea
movies <- select(mov, contains('ea'))
head(movies)
#>   Year
#> 1 2014
#> 2 2012
#> 3 2016
#> 4 2016
#> 5 2016
#> 6 2016

# selecting columns ending with r
movies <- select(mov, ends_with('r'))
head(movies)
#>               Director Year
#> 1           James Gunn 2014
#> 2         Ridley Scott 2012
#> 3   M. Night Shyamalan 2016
#> 4 Christophe Lourdelet 2016
#> 5           David Ayer 2016
#> 6          Yimou Zhang 2016

# selecting columns ending with r and e
movies <- select(mov, ends_with(c('k','r')))
head(movies)
#>   Rank             Director Year
#> 1    1           James Gunn 2014
#> 2    2         Ridley Scott 2012
#> 3    3   M. Night Shyamalan 2016
#> 4    4 Christophe Lourdelet 2016
#> 5    5           David Ayer 2016
#> 6    6          Yimou Zhang 2016
```

5.3.3 Selecting a single column

Selecting a single column with `select()` returns a one-column data frame. Often, a vector is wanted instead, to that end there is the function `pull()`.

The function `pull()` is used to select a single column and return a vector.

```r
movies <- select(mov, c('Title', 'Year', 'Revenue', 'Metascore'))

# using select returns a tibble
head(select(movies, 'Title'), 3)
#>                     Title
#> 1 Guardians of the Galaxy
#> 2              Prometheus
#> 3                   Split
class(select(movies, 'Title'))
#> [1] "data.frame"

# using pull returns a vector whose type depends on the data type of the column
head(pull(movies, var = 1))
#> [1] "Guardians of the Galaxy" "Prometheus"
#> [3] "Split"                   "Sing"
#> [5] "Suicide Squad"           "The Great Wall"
class(pull(movies, var = 1))
#> [1] "character"
```

5.3.4 Filtering rows

The function `filter()` is used to filter rows.

```r
movies <- select(mov, -c(1, 3, 4, 6, 8, 10))

# using the filter() function
movies. <- filter(movies, Year == 2006)
head(movies., 3)
#>                                         Title
#> 1                                The Prestige
#> 2 Pirates of the Caribbean: Dead Man's Chest
#> 3                                The Departed
#>             Director Year Rating Revenue Metascore
#> 1 Christopher Nolan 2006    8.5   53.08        66
#> 2    Gore Verbinski 2006    7.3  423.03        53
#> 3   Martin Scorsese 2006    8.5  132.37        85
tail(movies., 3)
#>                                          Title
#> 42 Talladega Nights: The Ballad of Ricky Bobby
#> 43                        Lucky Number Slevin
#> 44                              Inland Empire
#>          Director Year Rating Revenue Metascore
#> 42    Adam McKay 2006    6.6  148.21        66
#> 43 Paul McGuigan 2006    7.8   22.49        53
#> 44   David Lynch 2006    7.0      NA        NA
```

```r
# selecting movies released in 2006 with a rating above 8
filter(movies, Year == 2006 & Rating >= 8)
#>                       Title                          Director
#> 1              The Prestige                 Christopher Nolan
#> 2              The Departed                   Martin Scorsese
#> 3             Casino Royale                   Martin Campbell
#> 4           Pan's Labyrinth                Guillermo del Toro
#> 5       The Lives of Others Florian Henckel von Donnersmarck
#> 6 The Pursuit of Happyness                  Gabriele Muccino
#> 7             Blood Diamond                      Edward Zwick
#>   Year Rating Revenue Metascore
#> 1 2006    8.5   53.08        66
#> 2 2006    8.5  132.37        85
#> 3 2006    8.0  167.01        80
#> 4 2006    8.2   37.62        98
#> 5 2006    8.5   11.28        89
#> 6 2006    8.0  162.59        64
#> 7 2006    8.0   57.37        64

# without the & operator
filter(movies, Year == 2006, Rating >= 8)
#>                       Title                          Director
#> 1              The Prestige                 Christopher Nolan
#> 2              The Departed                   Martin Scorsese
#> 3             Casino Royale                   Martin Campbell
#> 4           Pan's Labyrinth                Guillermo del Toro
#> 5       The Lives of Others Florian Henckel von Donnersmarck
#> 6 The Pursuit of Happyness                  Gabriele Muccino
#> 7             Blood Diamond                      Edward Zwick
#>   Year Rating Revenue Metascore
#> 1 2006    8.5   53.08        66
#> 2 2006    8.5  132.37        85
#> 3 2006    8.0  167.01        80
#> 4 2006    8.2   37.62        98
#> 5 2006    8.5   11.28        89
#> 6 2006    8.0  162.59        64
#> 7 2006    8.0   57.37        64

# selecting rows with NA values on the Metascore column
movies. <- filter(movies, is.na(Metascore))
head(movies.)
#>                      Title            Director Year Rating
#> 1          Paris pieds nus      Dominique Abel 2016    6.8
#> 2 Bahubali: The Beginning      S.S. Rajamouli 2015    8.3
#> 3               Dead Awake       Phillip Guzman 2016    4.7
```

```
#> 4               5- 25- 77 Patrick Read Johnson 2007    7.1
#> 5 Don't Fuck in the Woods        Shawn Burkett 2016    2.7
#> 6                  Fallen          Scott Hicks 2016    5.6
#>   Revenue Metascore
#> 1      NA        NA
#> 2    6.50        NA
#> 3    0.01        NA
#> 4      NA        NA
#> 5      NA        NA
#> 6      NA        NA

# selecting rows with NA values on the Revenue and Metascore column
movies. <- filter(movies, is.na(Revenue), is.na(Metascore))
head(movies.)
#>                     Title             Director Year Rating
#> 1          Paris pieds nus       Dominique Abel 2016    6.8
#> 2               5- 25- 77 Patrick Read Johnson 2007    7.1
#> 3 Don't Fuck in the Woods        Shawn Burkett 2016    2.7
#> 4                  Fallen          Scott Hicks 2016    5.6
#> 5            Contratiempo          Oriol Paulo 2016    7.9
#> 6    Boyka: Undisputed IV      Todor Chapkanov 2016    7.4
#>   Revenue Metascore
#> 1      NA        NA
#> 2      NA        NA
#> 3      NA        NA
#> 4      NA        NA
#> 5      NA        NA
#> 6      NA        NA

# selecting rows with NA values on either the Revenue or Metascore column
movies. <- filter(movies, is.na(Revenue) | is.na(Metascore))
head(movies.)
#>                     Title             Director Year Rating
#> 1                Mindhorn           Sean Foley 2016    6.4
#> 2          Hounds of Love            Ben Young 2016    6.7
#> 3         Paris pieds nus       Dominique Abel 2016    6.8
#> 4 Bahubali: The Beginning       S.S. Rajamouli 2015    8.3
#> 5              Dead Awake       Phillip Guzman 2016    4.7
#> 6               5- 25- 77 Patrick Read Johnson 2007    7.1
#>   Revenue Metascore
#> 1      NA        71
#> 2      NA        72
#> 3      NA        NA
#> 4    6.50        NA
#> 5    0.01        NA
```

```r
#> 6      NA        NA

# selecting rows without NA values on the Metascore column
movies. <- filter(movies, !is.na(Metascore))
head(movies.)
#>                     Title            Director Year Rating
#> 1 Guardians of the Galaxy          James Gunn 2014    8.1
#> 2              Prometheus        Ridley Scott 2012    7.0
#> 3                   Split   M. Night Shyamalan 2016    7.3
#> 4                    Sing Christophe Lourdelet 2016    7.2
#> 5           Suicide Squad          David Ayer 2016    6.2
#> 6          The Great Wall         Yimou Zhang 2016    6.1
#>   Revenue Metascore
#> 1  333.13        76
#> 2  126.46        65
#> 3  138.12        62
#> 4  270.32        59
#> 5  325.02        40
#> 6   45.13        42

# selecting rows without NA values on the Revenue and Metascore columns
movies. <- filter(movies, !is.na(Revenue), !is.na(Metascore))
head(movies.)
#>                     Title            Director Year Rating
#> 1 Guardians of the Galaxy          James Gunn 2014    8.1
#> 2              Prometheus        Ridley Scott 2012    7.0
#> 3                   Split   M. Night Shyamalan 2016    7.3
#> 4                    Sing Christophe Lourdelet 2016    7.2
#> 5           Suicide Squad          David Ayer 2016    6.2
#> 6          The Great Wall         Yimou Zhang 2016    6.1
#>   Revenue Metascore
#> 1  333.13        76
#> 2  126.46        65
#> 3  138.12        62
#> 4  270.32        59
#> 5  325.02        40
#> 6   45.13        42
nrow(movies.)
#> [1] 838

# selecting rows without NA values on either the Revenue or Metascore columns
movies. <- filter(movies, !is.na(Revenue) | !is.na(Metascore))
head(movies.)
#>                     Title            Director Year Rating
#> 1 Guardians of the Galaxy          James Gunn 2014    8.1
```

```
#> 2             Prometheus         Ridley Scott 2012    7.0
#> 3                  Split   M. Night Shyamalan 2016    7.3
#> 4                   Sing Christophe Lourdelet 2016    7.2
#> 5          Suicide Squad           David Ayer 2016    6.2
#> 6         The Great Wall          Yimou Zhang 2016    6.1
#>   Revenue Metascore
#> 1  333.13        76
#> 2  126.46        65
#> 3  138.12        62
#> 4  270.32        59
#> 5  325.02        40
#> 6   45.13        42

nrow(movies.)
#> [1] 970

# selecting films released in 2006 and 2008
movies. <- filter(movies, Year %in% c(2006, 2008))
head(movies.)
#>                                        Title
#> 1                            The Dark Knight
#> 2                               The Prestige
#> 3 Pirates of the Caribbean: Dead Man's Chest
#> 4                               The Departed
#> 5                                        300
#> 6                                 Mamma Mia!
#>            Director Year Rating Revenue Metascore
#> 1 Christopher Nolan 2008    9.0  533.32        82
#> 2 Christopher Nolan 2006    8.5   53.08        66
#> 3    Gore Verbinski 2006    7.3  423.03        53
#> 4   Martin Scorsese 2006    8.5  132.37        85
#> 5       Zack Snyder 2006    7.7  210.59        52
#> 6    Phyllida Lloyd 2008    6.4  143.70        51

# selecting films released by 'James Gunn' or 'James Marsh'
movies. <- filter(movies, Director %in% c('James Gunn', 'James Marsh'))
head(movies.)
#>                      Title    Director Year Rating Revenue
#> 1  Guardians of the Galaxy  James Gunn 2014    8.1  333.13
#> 2 The Theory of Everything James Marsh 2014    7.7   35.89
#> 3                  Slither  James Gunn 2006    6.5    7.77
#> 4                    Super  James Gunn 2010    6.8    0.32
#>   Metascore
#> 1        76
#> 2        72
```

```
#> 3        69
#> 4        50

# selecting films released between 2006 and 2008
movies. <- filter(movies, between(Year, 2006, 2008))
head(movies., 3)
#>             Title             Director Year Rating Revenue
#> 1       5- 25- 77 Patrick Read Johnson 2007    7.1      NA
#> 2 The Dark Knight    Christopher Nolan 2008    9.0  533.32
#> 3    The Prestige    Christopher Nolan 2006    8.5   53.08
#>   Metascore
#> 1        NA
#> 2        82
#> 3        66
tail(movies., 3)
#>                        Title   Director Year Rating Revenue
#> 147       Taare Zameen Par Aamir Khan 2007    8.5    1.20
#> 148        Hostel: Part II   Eli Roth 2007    5.5   17.54
#> 149 Step Up 2: The Streets Jon M. Chu 2008    6.2   58.01
#>     Metascore
#> 147        42
#> 148        46
#> 149        50
```

5.3.4.1 Randomly selecting rows

The function `sample_frac()` randomly samples rows and returns a fixed fraction of them.

```
# sampling by a proportion
sample_frac(movies, 0.005, replace = TRUE)
#>                                         Title
#> 1                          Dallas Buyers Club
#> 2                                 Chalk It Up
#> 3                        Perfetti sconosciuti
#> 4 Star Wars: Episode VII - The Force Awakens
#> 5                           A United Kingdom
#>            Director Year Rating Revenue Metascore
#> 1 Jean-Marc Vallée 2013    8.0   27.30        84
#> 2  Hisonni Johnson 2016    4.8      NA        NA
#> 3   Paolo Genovese 2016    7.7      NA        43
#> 4      J.J. Abrams 2015    8.1  936.63        81
#> 5      Amma Asante 2016    6.8    3.90        65
```

The function `sample_n()` randomly samples rows and returns a fixed number of them.

```
# sampling by number
sample_n(movies, 5, replace = TRUE)
#>                  Title        Director Year Rating Revenue
#> 1                Mommy    Xavier Dolan 2014    8.1    3.49
#> 2 Ouija: Origin of Evil   Mike Flanagan 2016    6.1   34.90
#> 3        Blood Diamond    Edward Zwick 2006    8.0   57.37
#> 4        The Conjuring       James Wan 2013    7.5  137.39
#> 5            Concussion Peter Landesman 2015    7.1   34.53
#>   Metascore
#> 1        74
#> 2        65
#> 3        64
#> 4        68
#> 5        NA
```

5.3.5 Slicing

The function `slice()` is used to slice a data set.

```
slice(movies, 200:205)
#>                   Title               Director Year Rating
#> 1 Central Intelligence Rawson Marshall Thurber 2016    6.3
#> 2     Edge of Tomorrow              Doug Liman 2014    7.9
#> 3  A Cure for Wellness          Gore Verbinski 2016    6.5
#> 4              Snowden            Oliver Stone 2016    7.3
#> 5             Iron Man             Jon Favreau 2008    7.9
#> 6            Allegiant        Robert Schwentke 2016    5.7
#>   Revenue Metascore
#> 1  127.38        52
#> 2  100.19        71
#> 3    8.10        47
#> 4   21.48        58
#> 5  318.30        79
#> 6   66.00        33
```

5.3.6 Top values

The function `top_n()` returns the top nth number of elements in a column.

```
# top 5 movies by revenue
top_n(movies, 5, Revenue)
#>                                         Title
#> 1 Star Wars: Episode VII - The Force Awakens
#> 2                             The Dark Knight
#> 3                                The Avengers
#> 4                              Jurassic World
```

```
#> 5                                    Avatar
#>            Director Year Rating Revenue Metascore
#> 1        J.J. Abrams 2015    8.1  936.63        81
#> 2 Christopher Nolan 2008    9.0  533.32        82
#> 3        Joss Whedon 2012    8.1  623.28        69
#> 4    Colin Trevorrow 2015    7.0  652.18        59
#> 5      James Cameron 2009    7.8  760.51        83

# if no column is specified, the last is used.
top_n(movies, 5)
#>                  Title           Director Year Rating
#> 1 Manchester by the Sea   Kenneth Lonergan 2016    7.9
#> 2             Moonlight      Barry Jenkins 2016    7.5
#> 3      12 Years a Slave      Steve McQueen 2013    8.1
#> 4       Pan's Labyrinth Guillermo del Toro 2006    8.2
#> 5           Ratatouille          Brad Bird 2007    8.0
#> 6               Gravity     Alfonso Cuarón 2013    7.8
#> 7               Boyhood  Richard Linklater 2014    7.9
#>   Revenue Metascore
#> 1   47.70        96
#> 2   27.85        99
#> 3   56.67        96
#> 4   37.62        98
#> 5  206.44        96
#> 6  274.08        96
#> 7   25.36       100
```

The function `top_frac()` returns the top nth elements in a column by proportion.

```
# top 0.5% of movies by revenue
top_frac(movies, 0.005, Revenue)
#>                                          Title
#> 1 Star Wars: Episode VII - The Force Awakens
#> 2                              The Dark Knight
#> 3                                 The Avengers
#> 4                               Jurassic World
#> 5                                       Avatar
#>            Director Year Rating Revenue Metascore
#> 1        J.J. Abrams 2015    8.1  936.63        81
#> 2 Christopher Nolan 2008    9.0  533.32        82
#> 3        Joss Whedon 2012    8.1  623.28        69
#> 4    Colin Trevorrow 2015    7.0  652.18        59
#> 5      James Cameron 2009    7.8  760.51        83

# if no column is specified, the last is used.
```

```
top_frac(movies, 0.005)
#>                   Title           Director Year Rating
#> 1 Manchester by the Sea   Kenneth Lonergan 2016    7.9
#> 2             Moonlight     Barry Jenkins 2016    7.5
#> 3      12 Years a Slave     Steve McQueen 2013    8.1
#> 4       Pan's Labyrinth Guillermo del Toro 2006    8.2
#> 5           Ratatouille          Brad Bird 2007    8.0
#> 6               Gravity     Alfonso Cuarón 2013    7.8
#> 7               Boyhood  Richard Linklater 2014    7.9
#>   Revenue Metascore
#> 1   47.70        96
#> 2   27.85        99
#> 3   56.67        96
#> 4   37.62        98
#> 5  206.44        96
#> 6  274.08        96
#> 7   25.36       100
```

5.3.7 Using select and filter

```
select(filter(mov, Year == 2006, Rating >= 8), 2, 7, 9, 11, 12)
#>                      Title Year Rating Revenue Metascore
#> 1             The Prestige 2006    8.5   53.08        66
#> 2             The Departed 2006    8.5  132.37        85
#> 3            Casino Royale 2006    8.0  167.01        80
#> 4          Pan's Labyrinth 2006    8.2   37.62        98
#> 5      The Lives of Others 2006    8.5   11.28        89
#> 6 The Pursuit of Happyness 2006    8.0  162.59        64
#> 7            Blood Diamond 2006    8.0   57.37        64
filter(select(mov, 2, 7, 9, 11, 12), Year == 2006, Rating >= 8)
#>                      Title Year Rating Revenue Metascore
#> 1             The Prestige 2006    8.5   53.08        66
#> 2             The Departed 2006    8.5  132.37        85
#> 3            Casino Royale 2006    8.0  167.01        80
#> 4          Pan's Labyrinth 2006    8.2   37.62        98
#> 5      The Lives of Others 2006    8.5   11.28        89
#> 6 The Pursuit of Happyness 2006    8.0  162.59        64
#> 7            Blood Diamond 2006    8.0   57.37        64
```

With such an operation, it is better to use the pipe operator.

5.3.8 Pipe operator

The pipe operator (`%>%`) passes an object forward into a function. The shortcut `Ctrl + Shift + M` for PC and `Cmd + Shift + M` for Mac is used to insert this operator. Below, we pass the dataset mov into the function `filter()`, which after processing, passes its output to `select()`.

```
# passing movies dataset into filter and then to select
mov %>%
  filter(Year == 2006 & Rating >= 8) %>%
  select(2, 7, 9, 11, 12)
#>                          Title Year Rating Revenue Metascore
#> 1                 The Prestige 2006    8.5   53.08        66
#> 2                 The Departed 2006    8.5  132.37        85
#> 3                Casino Royale 2006    8.0  167.01        80
#> 4              Pan's Labyrinth 2006    8.2   37.62        98
#> 5         The Lives of Others 2006    8.5   11.28        89
#> 6 The Pursuit of Happyness 2006    8.0  162.59        64
#> 7                Blood Diamond 2006    8.0   57.37        64

mov %>%
  select(2, 7, 9, 11, 12) %>%
  filter(Year == 2006 & Rating >= 8)
#>                          Title Year Rating Revenue Metascore
#> 1                 The Prestige 2006    8.5   53.08        66
#> 2                 The Departed 2006    8.5  132.37        85
#> 3                Casino Royale 2006    8.0  167.01        80
#> 4              Pan's Labyrinth 2006    8.2   37.62        98
#> 5         The Lives of Others 2006    8.5   11.28        89
#> 6 The Pursuit of Happyness 2006    8.0  162.59        64
#> 7                Blood Diamond 2006    8.0   57.37        64
```

Using `.` as a placeholder for the data set. The period will be replaced in the function by the data frame or tibble.

```
mov %>%
  filter(.$Year == 2006 & .$Rating >= 8) %>%
  select(2, 7, 9, 11, 12)
#>                          Title Year Rating Revenue Metascore
#> 1                 The Prestige 2006    8.5   53.08        66
#> 2                 The Departed 2006    8.5  132.37        85
#> 3                Casino Royale 2006    8.0  167.01        80
#> 4              Pan's Labyrinth 2006    8.2   37.62        98
#> 5         The Lives of Others 2006    8.5   11.28        89
#> 6 The Pursuit of Happyness 2006    8.0  162.59        64
#> 7                Blood Diamond 2006    8.0   57.37        64
```

5.4 Manipulating Columns

5.4.1 Inserting a new column

The function `mutate()` and transmutate are used to manipulate columns. They are used to:

- insert new columns
- duplicate columns
- deriving new columns
- update existing ones

```
# adding a new column known as example
select(mov, c('Title', 'Year', 'Revenue', 'Metascore')) %>%
mutate(example = sample(1000)) %>%
  head()
#>                     Title Year Revenue Metascore example
#> 1 Guardians of the Galaxy 2014  333.13        76     526
#> 2              Prometheus 2012  126.46        65     223
#> 3                   Split 2016  138.12        62     414
#> 4                    Sing 2016  270.32        59     756
#> 5           Suicide Squad 2016  325.02        40     360
#> 6          The Great Wall 2016   45.13        42     512

# duplicating the Revenue column
select(mov, c('Title', 'Year', 'Revenue', 'Metascore')) %>%
mutate(Metascore.2 = Metascore) %>%
  head()
#>                     Title Year Revenue Metascore
#> 1 Guardians of the Galaxy 2014  333.13        76
#> 2              Prometheus 2012  126.46        65
#> 3                   Split 2016  138.12        62
#> 4                    Sing 2016  270.32        59
#> 5           Suicide Squad 2016  325.02        40
#> 6          The Great Wall 2016   45.13        42
#>   Metascore.2
#> 1          76
#> 2          65
#> 3          62
#> 4          59
#> 5          40
#> 6          42

# deriving the new column Movie Class
labels <- c('Very Low', 'Low', 'Moderate', 'High', 'Very High')

select(mov, c('Title', 'Year', 'Rating', 'Revenue', 'Metascore')) %>%
```

```
  mutate(`Movie Class` = cut(Rating, breaks = c(0, 5.5, 6.5, 7, 7.5, 10),
                             labels = labels)) %>%
  head()
#>                     Title Year Rating Revenue Metascore
#> 1 Guardians of the Galaxy 2014    8.1  333.13        76
#> 2              Prometheus 2012    7.0  126.46        65
#> 3                   Split 2016    7.3  138.12        62
#> 4                    Sing 2016    7.2  270.32        59
#> 5           Suicide Squad 2016    6.2  325.02        40
#> 6          The Great Wall 2016    6.1   45.13        42
#>   Movie Class
#> 1   Very High
#> 2    Moderate
#> 3        High
#> 4        High
#> 5         Low
#> 6         Low

# Updating the Director column to uppercase
select(mov, c(Title, Director, Year, Rating, Revenue, Metascore)) %>%
  mutate(Director = toupper(Director)) %>%
  head()
#>                     Title             Director Year Rating
#> 1 Guardians of the Galaxy           JAMES GUNN 2014    8.1
#> 2              Prometheus         RIDLEY SCOTT 2012    7.0
#> 3                   Split    M. NIGHT SHYAMALAN 2016    7.3
#> 4                    Sing CHRISTOPHE LOURDELET 2016    7.2
#> 5           Suicide Squad           DAVID AYER 2016    6.2
#> 6          The Great Wall          YIMOU ZHANG 2016    6.1
#>   Revenue Metascore
#> 1  333.13        76
#> 2  126.46        65
#> 3  138.12        62
#> 4  270.32        59
#> 5  325.02        40
#> 6   45.13        42

# using a customized function
# defining a function
fin_crisis <- function(x) {
    if(x < 2008){
        return('pre financial crisis')
    }else if(x < 2010 ){
        return('financial crisis')
    }else{
```

```
        return('post financial crisis')
    }
}

select(mov, 2, 7, 11, 12) %>%
  mutate('fin crisis Class' = sapply(Year, fin_crisis)) %>%
  head()
#>                     Title Year Revenue Metascore
#> 1 Guardians of the Galaxy 2014  333.13        76
#> 2              Prometheus 2012  126.46        65
#> 3                   Split 2016  138.12        62
#> 4                    Sing 2016  270.32        59
#> 5           Suicide Squad 2016  325.02        40
#> 6          The Great Wall 2016   45.13        42
#>        fin crisis Class
#> 1 post financial crisis
#> 2 post financial crisis
#> 3 post financial crisis
#> 4 post financial crisis
#> 5 post financial crisis
#> 6 post financial crisis

# deriving a new column from a calculation
select(mov, 2, 5, 7, 8, 11) %>%
  mutate('Rev/Run' = round(Revenue/Runtime, 2)) %>%
  head()
#>                     Title             Director Year Runtime
#> 1 Guardians of the Galaxy           James Gunn 2014     121
#> 2              Prometheus         Ridley Scott 2012     124
#> 3                   Split   M. Night Shyamalan 2016     117
#> 4                    Sing Christophe Lourdelet 2016     108
#> 5           Suicide Squad           David Ayer 2016     123
#> 6          The Great Wall          Yimou Zhang 2016     103
#>   Revenue Rev/Run
#> 1  333.13    2.75
#> 2  126.46    1.02
#> 3  138.12    1.18
#> 4  270.32    2.50
#> 5  325.02    2.64
#> 6   45.13    0.44
```

The function `case_when()` is a condensed form of if else statement or CASE THEN in SQL.

```
# classifying movies by ratings
select(mov, 2, 7, 9, 11, 12) %>%
```

```
  mutate(category = case_when(Rating < 5.5 ~ 'Very Low',
                              Rating < 6.5 ~ 'Low',
                              Rating < 7 ~ 'Moderate',
                              Rating < 7.5 ~ 'High',
                              Rating <= 10 ~ 'Very High')) %>%
  head()
#>                     Title Year Rating Revenue Metascore
#> 1 Guardians of the Galaxy 2014    8.1  333.13        76
#> 2              Prometheus 2012    7.0  126.46        65
#> 3                   Split 2016    7.3  138.12        62
#> 4                    Sing 2016    7.2  270.32        59
#> 5           Suicide Squad 2016    6.2  325.02        40
#> 6          The Great Wall 2016    6.1   45.13        42
#>    category
#> 1 Very High
#> 2      High
#> 3      High
#> 4      High
#> 5       Low
#> 6       Low
```

The function `coalesce()` which is modelled after the COALESCE function in SQL returns the first non-missing element. Using it, we can replace NA values in a column.

```
# selecting some rows containing NA values
select(mov, 2, 5, 7, 9, 11, 12) %>%
  filter(is.na(Revenue)) %>%
  slice(c(8, 23, 26, 40, 43, 48))
#>                      Title         Director Year Rating
#> 1  The Autopsy of Jane Doe   André Øvredal 2016    6.8
#> 2                  Old Boy        Spike Lee 2013    5.8
#> 3                  Satanic Jeffrey G. Hunt 2016    3.7
#> 4       Absolutely Anything      Terry Jones 2015    6.0
#> 5 The Headhunter's Calling   Mark Williams 2016    6.9
#> 6           Predestination Michael Spierig 2014    7.5
#>   Revenue Metascore
#> 1      NA        65
#> 2      NA        49
#> 3      NA        NA
#> 4      NA        31
#> 5      NA        85
#> 6      NA        69

# replacing NA values with a value
select(mov, 2, 5, 7, 9, 11, 12) %>%
```

```
  mutate(Revenue = coalesce(Revenue, 50)) %>%
  slice(c(8, 23, 26, 40, 43, 48))
#>                      Title             Director Year Rating
#> 1                 Mindhorn           Sean Foley 2016    6.4
#> 2           Hounds of Love            Ben Young 2016    6.7
#> 3          Paris pieds nus       Dominique Abel 2016    6.8
#> 4              5- 25- 77 Patrick Read Johnson 2007    7.1
#> 5 Don't Fuck in the Woods        Shawn Burkett 2016    2.7
#> 6                   Fallen          Scott Hicks 2016    5.6
#>   Revenue Metascore
#> 1      50        71
#> 2      50        72
#> 3      50        NA
#> 4      50        NA
#> 5      50        NA
#> 6      50        NA

# replacing NA values with a computed value (mean/median)
select(mov, 2, 5, 7, 9, 11, 12) %>%
  mutate(Revenue = coalesce(Revenue, round(median(Revenue, na.rm = T))),
         Metascore = coalesce(Metascore, round(mean(Metascore, na.rm = T)))) %>%
  slice(c(8, 23, 26, 40, 43, 48))
#>                      Title             Director Year Rating
#> 1                 Mindhorn           Sean Foley 2016    6.4
#> 2           Hounds of Love            Ben Young 2016    6.7
#> 3          Paris pieds nus       Dominique Abel 2016    6.8
#> 4              5- 25- 77 Patrick Read Johnson 2007    7.1
#> 5 Don't Fuck in the Woods        Shawn Burkett 2016    2.7
#> 6                   Fallen          Scott Hicks 2016    5.6
#>   Revenue Metascore
#> 1      48        71
#> 2      48        72
#> 3      48        59
#> 4      48        59
#> 5      48        59
#> 6      48        59
```

The function `transmutate()` behaves like `mutate()` but drops other columns that are not selected.

```
# transmutate drops unselected columns
select(mov, c(Title, Director, Year, Rating, Revenue, Metascore)) %>%
  transmute(Director = toupper(Director)) %>%
  head()
#>              Director
#> 1          JAMES GUNN
```

```
#> 2         RIDLEY SCOTT
#> 3   M. NIGHT SHYAMALAN
#> 4 CHRISTOPHE LOURDELET
#> 5           DAVID AYER
#> 6          YIMOU ZHANG

# transmutate keeps selected columns
select(mov, c(Title, Director, Year, Runtime, Revenue, Metascore)) %>%
  transmute(Director = toupper(Director),
            Year,
            Revenue = round(Revenue/Runtime, 2)) %>%
head()
#>               Director Year Revenue
#> 1           JAMES GUNN 2014    2.75
#> 2         RIDLEY SCOTT 2012    1.02
#> 3   M. NIGHT SHYAMALAN 2016    1.18
#> 4 CHRISTOPHE LOURDELET 2016    2.50
#> 5           DAVID AYER 2016    2.64
#> 6          YIMOU ZHANG 2016    0.44
```

5.5 Sorting and ranking

5.5.1 Sorting

The function `arrange()` is used to sort data frames. It does an ascending sort but to do a descending sort, we use the function `desc()` or the negative sign.

```
# sort increasing
select(mov, c(Title, Year, Runtime, Revenue, Metascore)) %>%
  arrange(Revenue) %>%
  head(10)
#>                         Title Year Runtime Revenue Metascore
#> 1           A Kind of Murder 2016      95    0.00        50
#> 2                 Dead Awake 2016      99    0.01        NA
#> 3                  Wakefield 2016     106    0.01        61
#> 4                   Lovesong 2016      84    0.01        74
#> 5               Love, Rosie 2014     102    0.01        44
#> 6          Into the Forest 2015     101    0.01        59
#> 7                 Stake Land 2010      98    0.02        66
#> 8             The First Time 2012      95    0.02        55
#> 9  The Blackcoat's Daughter 2015      93    0.02        68
#> 10         The Sea of Trees 2015     110    0.02        23

# sort decreasing using the negative sign
select(mov, c(Title, Year, Runtime, Revenue, Metascore)) %>%
```

```
  arrange(-Revenue) %>%
  head(10)
#>                                      Title Year Runtime
#> 1  Star Wars: Episode VII - The Force Awakens 2015     136
#> 2                                   Avatar 2009     162
#> 3                           Jurassic World 2015     124
#> 4                             The Avengers 2012     143
#> 5                          The Dark Knight 2008     152
#> 6                                Rogue One 2016     133
#> 7                             Finding Dory 2016      97
#> 8                  Avengers: Age of Ultron 2015     141
#> 9                    The Dark Knight Rises 2012     164
#> 10         The Hunger Games: Catching Fire 2013     146
#>    Revenue Metascore
#> 1   936.63        81
#> 2   760.51        83
#> 3   652.18        59
#> 4   623.28        69
#> 5   533.32        82
#> 6   532.17        65
#> 7   486.29        77
#> 8   458.99        66
#> 9   448.13        78
#> 10  424.65        76

# sort decreasing using desc()
select(mov, c(Title, Year, Runtime, Revenue, Metascore)) %>%
  arrange(desc(Revenue)) %>%
  head(10)
#>                                      Title Year Runtime
#> 1  Star Wars: Episode VII - The Force Awakens 2015     136
#> 2                                   Avatar 2009     162
#> 3                           Jurassic World 2015     124
#> 4                             The Avengers 2012     143
#> 5                          The Dark Knight 2008     152
#> 6                                Rogue One 2016     133
#> 7                             Finding Dory 2016      97
#> 8                  Avengers: Age of Ultron 2015     141
#> 9                    The Dark Knight Rises 2012     164
#> 10         The Hunger Games: Catching Fire 2013     146
#>    Revenue Metascore
#> 1   936.63        81
#> 2   760.51        83
#> 3   652.18        59
#> 4   623.28        69
```

```
#> 5   533.32        82
#> 6   532.17        65
#> 7   486.29        77
#> 8   458.99        66
#> 9   448.13        78
#> 10  424.65        76


# sorting on multiple columns
select(mov, c(Title, Year, Runtime, Revenue, Metascore)) %>%
  arrange(-Metascore, Revenue) %>%
  head(10)
#>                    Title Year Runtime Revenue Metascore
#> 1                Boyhood 2014     165   25.36       100
#> 2              Moonlight 2016     111   27.85        99
#> 3        Pan's Labyrinth 2006     118   37.62        98
#> 4  Manchester by the Sea 2016     137   47.70        96
#> 5       12 Years a Slave 2013     134   56.67        96
#> 6            Ratatouille 2007     111  206.44        96
#> 7                Gravity 2013      91  274.08        96
#> 8                  Carol 2015     118    0.25        95
#> 9       Zero Dark Thirty 2012     157   95.72        95
#> 10    The Social Network 2010     120   96.92        95
```

5.5.2 Ranking

The functions `row_number()`, `ntile()`, `min_rank()`, `dense_rank()`, `percent_rank()` and `cume_dist()` are used for ranking.

```
# ranking by revenue ascending
select(mov, Title, Year, Revenue, Metascore) %>%
  mutate(rank_by_revenue = dense_rank(Revenue)) %>%
  head()
#>                     Title Year Revenue Metascore
#> 1 Guardians of the Galaxy 2014  333.13        76
#> 2              Prometheus 2012  126.46        65
#> 3                   Split 2016  138.12        62
#> 4                    Sing 2016  270.32        59
#> 5           Suicide Squad 2016  325.02        40
#> 6          The Great Wall 2016   45.13        42
#>   rank_by_revenue
#> 1             783
#> 2             623
#> 3             646
#> 4             761
#> 5             781
```

```
#> 6            370

# ranking by revenue decreasing using desc()
select(mov, Title, Year, Revenue, Metascore) %>%
  mutate(rank_by_revenue = dense_rank(desc(Revenue))) %>%
  head()
#>                     Title Year Revenue Metascore
#> 1 Guardians of the Galaxy 2014  333.13        76
#> 2              Prometheus 2012  126.46        65
#> 3                   Split 2016  138.12        62
#> 4                    Sing 2016  270.32        59
#> 5           Suicide Squad 2016  325.02        40
#> 6          The Great Wall 2016   45.13        42
#>   rank_by_revenue
#> 1              32
#> 2             192
#> 3             169
#> 4              54
#> 5              34
#> 6             445

# ranking by revenue decreasing using negative sign
select(mov, Title, Year, Revenue, Metascore) %>%
  mutate(rank_by_revenue = dense_rank(-Revenue)) %>%
  head()
#>                     Title Year Revenue Metascore
#> 1 Guardians of the Galaxy 2014  333.13        76
#> 2              Prometheus 2012  126.46        65
#> 3                   Split 2016  138.12        62
#> 4                    Sing 2016  270.32        59
#> 5           Suicide Squad 2016  325.02        40
#> 6          The Great Wall 2016   45.13        42
#>   rank_by_revenue
#> 1              32
#> 2             192
#> 3             169
#> 4              54
#> 5              34
#> 6             445

# rank and arrange
select(mov, Title, Year, Revenue, Metascore) %>%
  mutate(rank_by_revenue = dense_rank(desc(Revenue))) %>%
  arrange(desc(Revenue)) %>%
  head()
```

```
#>                                         Title Year Revenue
#> 1 Star Wars: Episode VII - The Force Awakens 2015   936.63
#> 2                                      Avatar 2009   760.51
#> 3                              Jurassic World 2015   652.18
#> 4                                The Avengers 2012   623.28
#> 5                             The Dark Knight 2008   533.32
#> 6                                   Rogue One 2016   532.17
#>   Metascore rank_by_revenue
#> 1        81               1
#> 2        83               2
#> 3        59               3
#> 4        69               4
#> 5        82               5
#> 6        65               6

# adding row numbers
select(mov, Title, Year, Revenue, Metascore) %>%
  arrange(Year, Revenue) %>%
  mutate(`row number` = row_number()) %>%
  head()
#>                                 Title Year Revenue Metascore
#> 1                           Idiocracy 2006    0.44        66
#> 2                            The Host 2006    2.20        85
#> 3 Perfume: The Story of a Murderer 2006    2.21        56
#> 4                            The Fall 2006    2.28        64
#> 5                     She's the Man 2006    2.34        45
#> 6                         Rescue Dawn 2006    5.48        77
#>   row number
#> 1          1
#> 2          2
#> 3          3
#> 4          4
#> 5          5
#> 6          6

# dividing data into evenly sized buckets
movies <-
select(mov, Title, Year, Revenue, Metascore) %>%
  arrange(Year) %>%
  mutate(buckets = ntile(Revenue, 5))
head(movies)
#>                                         Title Year Revenue
#> 1                                The Prestige 2006    53.08
#> 2 Pirates of the Caribbean: Dead Man's Chest 2006   423.03
#> 3                               The Departed 2006   132.37
```

```
#> 4                                 300 2006  210.59
#> 5                       Casino Royale 2006  167.01
#> 6                                Cars 2006  244.05
#>   Metascore buckets
#> 1        66       3
#> 2        53       5
#> 3        85       4
#> 4        52       5
#> 5        80       5
#> 6        73       5

table(movies$buckets)
#>
#>   1   2   3   4   5
#> 175 175 174 174 174

# calculating mean by buckets
tapply(movies$Metascore, movies$buckets, function(x)round(mean(x, na.rm = T), 1))
#>    1    2    3    4    5
#> 62.4 57.7 54.3 58.5 64.9
```

5.6 Splitting and Merging columns

5.6.1 Splitting columns

The function `separate()` from the package tidyr is used to split columns.

```
# reading data
busiestAirports <- read.table(file = "data/busiestAirports.csv",
                              header = T,
                              sep=",",
                              dec = ".",
                              quote = "\"")

busiestAirports <- select(busiestAirports, c('iata_icao' = 5, 'location', 'country'))
head(busiestAirports)
#>   iata_icao                 location              country
#> 1  ATL/KATL         Atlanta, Georgia        United States
#> 2  PEK/ZBAA Chaoyang-Shunyi, Beijing                China
#> 3  DXB/OMDB           Garhoud, Dubai United Arab Emirates
#> 4  LAX/KLAX  Los Angeles, California        United States
#> 5  HND/RJTT               Ota, Tokyo                Japan
#> 6  ORD/KORD        Chicago, Illinois        United States

# splitting the column iata_icao into iata and icao
```

```
busiest_Airports <-
  tidyr::separate(busiestAirports, col = 'iata_icao', into = c('iata', 'icao'), sep =
head(busiest_Airports)
#>   iata icao                 location              country
#> 1  ATL KATL          Atlanta, Georgia        United States
#> 2  PEK ZBAA Chaoyang-Shunyi, Beijing                China
#> 3  DXB OMDB           Garhoud, Dubai United Arab Emirates
#> 4  LAX KLAX  Los Angeles, California        United States
#> 5  HND RJTT               Ota, Tokyo                Japan
#> 6  ORD KORD        Chicago, Illinois        United States
```

Also, we can make use of mutate() and substring() from base R or str_sub() from stringr to split by position.

```
# using substring
busiestAirports %>%
  mutate(iata = substring(iata_icao, 1, 3), icao = substring(iata_icao, 5, 7)) %>%
  select(-1) %>%
  head()
#>                  location              country iata icao
#> 1         Atlanta, Georgia        United States  ATL  KAT
#> 2 Chaoyang-Shunyi, Beijing                China  PEK  ZBA
#> 3           Garhoud, Dubai United Arab Emirates  DXB  OMD
#> 4  Los Angeles, California        United States  LAX  KLA
#> 5               Ota, Tokyo                Japan  HND  RJT
#> 6        Chicago, Illinois        United States  ORD  KOR

# using str_sub
busiestAirports %>%
  mutate(iata = stringr::str_sub(iata_icao, 1, 3), icao = stringr::str_sub(iata_icao, 5
  select(-1) %>%
  head()
#>                  location              country iata icao
#> 1         Atlanta, Georgia        United States  ATL  KAT
#> 2 Chaoyang-Shunyi, Beijing                China  PEK  ZBA
#> 3           Garhoud, Dubai United Arab Emirates  DXB  OMD
#> 4  Los Angeles, California        United States  LAX  KLA
#> 5               Ota, Tokyo                Japan  HND  RJT
#> 6        Chicago, Illinois        United States  ORD  KOR
```

5.6.2 Merging columns

The function `unite()` from the package tidyr is used to merge columns.

```
# reading data
busiestAirports <- read.table(file = "data/busiestAirports.csv",
```

```
                              header = T,
                              sep=",",
                              dec = ".",
                              quote = "\"")
busiestAirports <- select(busiestAirports, c('iata_icao' = 5, 'location', 'country'))
head(busiestAirports)
#>   iata_icao                 location              country
#> 1  ATL/KATL         Atlanta, Georgia        United States
#> 2  PEK/ZBAA Chaoyang-Shunyi, Beijing                China
#> 3  DXB/OMDB           Garhoud, Dubai United Arab Emirates
#> 4  LAX/KLAX  Los Angeles, California        United States
#> 5  HND/RJTT               Ota, Tokyo                Japan
#> 6  ORD/KORD        Chicago, Illinois        United States

# merging the columns iata, icao into iata_icao
busiestAirports <-
  tidyr::unite(busiestAirports, location, country, col = `location country`, sep = ', ')
head(busiestAirports)
#>   iata_icao                     location country
#> 1  ATL/KATL        Atlanta, Georgia, United States
#> 2  PEK/ZBAA        Chaoyang-Shunyi, Beijing, China
#> 3  DXB/OMDB   Garhoud, Dubai, United Arab Emirates
#> 4  LAX/KLAX Los Angeles, California, United States
#> 5  HND/RJTT                      Ota, Tokyo, Japan
#> 6  ORD/KORD       Chicago, Illinois, United States
```

5.6.3 Rearranging columns

The function `relocate()` is used to rearrange columns. It was added with dplr 1.0.0.

```
# rearranging columns
select(mov, c(Title, Year, Runtime, Revenue, Metascore)) %>%
  relocate(Revenue, Metascore) %>%
  head(3)
#>   Revenue Metascore                   Title Year Runtime
#> 1  333.13        76 Guardians of the Galaxy 2014     121
#> 2  126.46        65              Prometheus 2012     124
#> 3  138.12        62                   Split 2016     117

# placing year after metascore
select(mov, c(Title, Year, Runtime, Revenue, Metascore)) %>%
  relocate(Year, .after = Metascore) %>%
  head(3)
#>                     Title Runtime Revenue Metascore Year
```

```r
#> 1 Guardians of the Galaxy     121  333.13        76 2014
#> 2             Prometheus     124  126.46        65 2012
#> 3                  Split     117  138.12        62 2016

# placing year before title
select(mov, c(Title, Year, Runtime, Revenue, Metascore)) %>%
  relocate(Year, .before = Title) %>%
  head(3)
#>   Year                   Title Runtime Revenue Metascore
#> 1 2014 Guardians of the Galaxy     121  333.13        76
#> 2 2012              Prometheus     124  126.46        65
#> 3 2016                   Split     117  138.12        62

# placing year at the end
select(mov, c(Title, Year, Runtime, Revenue, Metascore)) %>%
  relocate(Year, .after = last_col()) %>%
  head(3)
#>                     Title Runtime Revenue Metascore Year
#> 1 Guardians of the Galaxy     121  333.13        76 2014
#> 2             Prometheus     124  126.46        65 2012
#> 3                  Split     117  138.12        62 2016

# numeric columns last
select(mov, c(Title, Year, Runtime, Revenue, Metascore)) %>%
  relocate(where(is.numeric), .after = where(is.character)) %>%
  head(3)
#>                     Title Year Runtime Revenue Metascore
#> 1 Guardians of the Galaxy 2014     121  333.13        76
#> 2             Prometheus 2012     124  126.46        65
#> 3                  Split 2016     117  138.12        62

# numeric columns first
select(mov, c(Title, Year, Runtime, Revenue, Metascore)) %>%
  relocate(where(is.numeric), .before = where(is.character)) %>%
  head(3)
#>   Year Runtime Revenue Metascore                   Title
#> 1 2014     121  333.13        76 Guardians of the Galaxy
#> 2 2012     124  126.46        65              Prometheus
#> 3 2016     117  138.12        62                   Split

# selecting character columns only
select(mov, c(Title, Year, Runtime, Revenue, Metascore)) %>%
  relocate(where(is.character)) %>%
  head(3)
#>                     Title Year Runtime Revenue Metascore
```

```
#> 1 Guardians of the Galaxy 2014      121   333.13         76
#> 2               Prometheus 2012      124   126.46         65
#> 3                    Split 2016      117   138.12         62
```

5.6.4 Deleting columns of data

There is no special function to delete columns but the function `select()` can be used to select or drop columns.

5.7 Manipulating Rows

5.7.1 Inserting rows

The function `add_row()` is used to add row(s) to data frames. It adds:

- single row with add_row(dt, column_name = value)
- multiple rows with add_row(dt, column_name = values)

```
# adding a single row of data
select(mov, c(2, 5, 7, 9, 11, 12)) %>%
add_row(Title = "the big g",
        Director = "goro lovic",
        Year = 2015,
        Rating = 9.9,
        Revenue = 1000,
        Metascore = 100) %>%
tail()
#>                        Title         Director Year Rating
#> 996    Secret in Their Eyes        Billy Ray 2015    6.2
#> 997         Hostel: Part II         Eli Roth 2007    5.5
#> 998  Step Up 2: The Streets       Jon M. Chu 2008    6.2
#> 999             Search Party   Scot Armstrong 2014    5.6
#> 1000              Nine Lives Barry Sonnenfeld 2016    5.3
#> 1001               the big g       goro lovic 2015    9.9
#>      Revenue Metascore
#> 996       NA        45
#> 997    17.54        46
#> 998    58.01        50
#> 999       NA        22
#> 1000   19.64        11
#> 1001 1000.00       100

# adding multiple rows of data
select(mov, c(2, 5, 7, 9, 11, 12)) %>%
  add_row(Title= c("the big g", "everyday", "true colors"),
          Director = c("goro lovic", "fk", "tupac"),
```

```
          Year = c(2015, 2016, 2014),
          Rating = c(9.9, 6.6, 8),
          Revenue = c(1000, 250, 350),
          Metascore = c(100, 60, 40)) %>%
  tail()
#>                        Title         Director Year Rating
#> 998  Step Up 2: The Streets       Jon M. Chu 2008    6.2
#> 999            Search Party   Scot Armstrong 2014    5.6
#> 1000             Nine Lives Barry Sonnenfeld 2016    5.3
#> 1001              the big g       goro lovic 2015    9.9
#> 1002               everyday               fk 2016    6.6
#> 1003            true colors            tupac 2014    8.0
#>      Revenue Metascore
#> 998    58.01        50
#> 999       NA        22
#> 1000   19.64        11
#> 1001 1000.00       100
#> 1002  250.00        60
#> 1003  350.00        40
```

Rows can also be added using the function `bind_rows()` which is an efficient implementation of `do.call(rbind, dfs)` in base R.

```
# adding a single row
select(mov, 2, 5, 7, 9, 11, 12) %>%
  bind_rows(list(Title = "the big g",
                 Director = "goro lovic",
                 Year = 2015,
                 Rating = 9.9,
                 Revenue = 1000,
                 Metascore = 100)) %>%
tail(3)
#>             Title         Director Year Rating Revenue
#> 999  Search Party   Scot Armstrong 2014    5.6      NA
#> 1000   Nine Lives Barry Sonnenfeld 2016    5.3   19.64
#> 1001    the big g       goro lovic 2015    9.9 1000.00
#>      Metascore
#> 999         22
#> 1000        11
#> 1001       100

# adding multiple rows
select(mov, 2, 5, 7, 9, 11, 12) %>%
  bind_rows(list(Rank= c(1, 3, 5),
                 Title= c("the big g", "everyday", "true colors"),
                 Director = c("goro lovic", "fk", "tupac"),
```

```
                  Year = c(2015, 2016, 2014),
                  Rating = c(9.9, 6.6, 8),
                  Revenue = c(1000, 250, 350),
                  Metascore = c(100, 60, 40))) %>%
tail()
#>                          Title          Director Year Rating
#> 998  Step Up 2: The Streets        Jon M. Chu 2008    6.2
#> 999            Search Party    Scot Armstrong 2014    5.6
#> 1000             Nine Lives Barry Sonnenfeld 2016    5.3
#> 1001              the big g        goro lovic 2015    9.9
#> 1002               everyday                fk 2016    6.6
#> 1003            true colors             tupac 2014    8.0
#>      Revenue Metascore Rank
#> 998    58.01        50   NA
#> 999       NA        22   NA
#> 1000   19.64        11   NA
#> 1001 1000.00       100    1
#> 1002  250.00        60    3
#> 1003  350.00        40    5
```

The function `rows_insert()` which is modelled after the SQL INSERT clause is also used to insert rows. It requires a column with unique value to uniquely identify each row. This function works on two tibbles, therefore the rows to be inserted must be converted to a tibble using the function `tibble()` before use. It should be noted that this function was added in version 1.0.0 of the package, therefore you should do well to update your package to make use of it. The function `rows_insert()` requires the argument by which identifies the unique row.

```
# creating a tibble
tb <-
  tibble(Rank = 1001,
         Title = "the big g",
         Director = "goro lovic",
         Year = 2015,
         Rating = 9.9,
         Revenue = 1000,
         Metascore = 100)

# inserting a single value
select(mov, 1, 2, 5, 7, 9, 11, 12) %>%
  rows_insert(tb, by = "Rank") %>%
  tail()
#>      Rank                   Title        Director Year
#> 996   996   Secret in Their Eyes       Billy Ray 2015
#> 997   997         Hostel: Part II        Eli Roth 2007
```

```
#> 998   998 Step Up 2: The Streets       Jon M. Chu 2008
#> 999   999           Search Party   Scot Armstrong 2014
#> 1000 1000             Nine Lives Barry Sonnenfeld 2016
#> 1001 1001              the big g       goro lovic 2015
#>      Rating Revenue Metascore
#> 996     6.2      NA        45
#> 997     5.5   17.54        46
#> 998     6.2   58.01        50
#> 999     5.6      NA        22
#> 1000    5.3   19.64        11
#> 1001    9.9 1000.00       100

# inserting multiple values
tb <-
  tibble(Rank= 1001:1003,
         Title= c("the big g", "everyday", "true colors"),
         Director = c("goro lovic", "fk", "tupac"),
         Year = c(2015, 2016, 2014),
         Rating = c(9.9, 6.6, 8),
         Revenue = c(1000, 250, 350),
         Metascore = c(100, 60, 40))

select(mov, 1, 2, 5, 7, 9, 11, 12) %>%
  rows_insert(tb, by = "Rank") %>%
  tail()
#>      Rank                  Title         Director Year
#> 998   998 Step Up 2: The Streets       Jon M. Chu 2008
#> 999   999           Search Party   Scot Armstrong 2014
#> 1000 1000             Nine Lives Barry Sonnenfeld 2016
#> 1001 1001              the big g       goro lovic 2015
#> 1002 1002               everyday               fk 2016
#> 1003 1003            true colors            tupac 2014
#>      Rating Revenue Metascore
#> 998     6.2   58.01        50
#> 999     5.6      NA        22
#> 1000    5.3   19.64        11
#> 1001    9.9 1000.00       100
#> 1002    6.6  250.00        60
#> 1003    8.0  350.00        40
```

5.7.2 Updating rows of data

The function `rows_update()` and `rows_upsert()` which are modelled after the SQL UPDATE and UPSERT clauses are used to update row values. While the former updates row values, the later updates existing rows and inserts new ones, if not present. They both required a column with unique value to uniquely identify each row. As with `rows_insert()`, these functions work on two tibbles, therefore the rows to be inserted must be converted to a tibble using the function `tibble()` before use.

```
# updating a single row
tb <-
  tibble(Rank = 5,
         Title = "the big g",
         Director = "goro lovic",
         Year = 2015,
         Rating = 9.9,
         Revenue = 1000,
         Metascore = 100)

select(mov, 1, 2, 5, 7, 9, 11, 12) %>%
  rows_update(tb, by = "Rank") %>%
  head()
#>   Rank                   Title             Director Year
#> 1    1 Guardians of the Galaxy           James Gunn 2014
#> 2    2              Prometheus         Ridley Scott 2012
#> 3    3                   Split   M. Night Shyamalan 2016
#> 4    4                    Sing Christophe Lourdelet 2016
#> 5    5               the big g           goro lovic 2015
#> 6    6           The Great Wall          Yimou Zhang 2016
#>   Rating Revenue Metascore
#> 1    8.1  333.13        76
#> 2    7.0  126.46        65
#> 3    7.3  138.12        62
#> 4    7.2  270.32        59
#> 5    9.9 1000.00       100
#> 6    6.1   45.13        42

# updating multiple rows
tb <-
  tibble(Rank= c(1, 3, 5),
         Title= c("the big g", "everyday", "true colors"),
         Director = c("goro lovic", "fk", "tupac"),
         Year = c(2015, 2016, 2014),
         Rating = c(9.9, 6.6, 8),
         Revenue = c(1000, 250, 350),
         Metascore = c(100, 60, 40))

select(mov, 1, 2, 5, 7, 9, 11, 12) %>%
  rows_update(tb, by = "Rank") %>%
  head()
```

```
#>   Rank           Title             Director Year Rating
#> 1    1       the big g           goro lovic 2015    9.9
#> 2    2      Prometheus         Ridley Scott 2012    7.0
#> 3    3        everyday                   fk 2016    6.6
#> 4    4            Sing Christophe Lourdelet 2016    7.2
#> 5    5     true colors                tupac 2014    8.0
#> 6    6 The Great Wall          Yimou Zhang 2016    6.1
#>   Revenue Metascore
#> 1 1000.00       100
#> 2  126.46        65
#> 3  250.00        60
#> 4  270.32        59
#> 5  350.00        40
#> 6   45.13        42

# update existing rows and insert new ones
tb <-
  tibble(Rank= c(2,3, 1001),
         Title= c("the big g", "everyday", "true colors"),
         Director = c("goro lovic", "fk", "tupac"),
         Year = c(2015, 2016, 2014),
         Rating = c(9.9, 6.6, 8),
         Revenue = c(1000, 250, 350),
         Metascore = c(100, 60, 40))

select(mov, 1, 2, 5, 7, 9, 11, 12) %>%
  rows_upsert(tb, by = "Rank") %>%
  slice(c(1:5, 1001))
#>   Rank                   Title             Director Year
#> 1    1 Guardians of the Galaxy           James Gunn 2014
#> 2    2               the big g           goro lovic 2015
#> 3    3                everyday                   fk 2016
#> 4    4                    Sing Christophe Lourdelet 2016
#> 5    5           Suicide Squad           David Ayer 2016
#> 6 1001             true colors                tupac 2014
#>   Rating Revenue Metascore
#> 1    8.1  333.13        76
#> 2    9.9 1000.00       100
#> 3    6.6  250.00        60
#> 4    7.2  270.32        59
#> 5    6.2  325.02        40
#> 6    8.0  350.00        40
```

5.7.3 Updating a single value

To update a single value, we make use of mutate() with either ifelse() from base R or if_else() from dplyr or replace() from base R. The function if_else() is an implementation of ifelse() in dplyr.

```
mov %>%
  select(1, 2, 5, 7, 9, 11, 12) %>%
  filter(Director == 'Christopher Nolan')
#>   Rank                 Title          Director Year Rating
#> 1   37          Interstellar Christopher Nolan 2014    8.6
#> 2   55       The Dark Knight Christopher Nolan 2008    9.0
#> 3   65          The Prestige Christopher Nolan 2006    8.5
#> 4   81             Inception Christopher Nolan 2010    8.8
#> 5  125 The Dark Knight Rises Christopher Nolan 2012    8.5
#>   Revenue Metascore
#> 1  187.99        74
#> 2  533.32        82
#> 3   53.08        66
#> 4  292.57        74
#> 5  448.13        78

# replacing a value using ifelse()
mov %>%
  select(1, 2, 5, 7, 9, 11, 12) %>%
  mutate(Director = ifelse(Director == 'Christopher Nolan', 'C. Nolan', Director)) %>%
  slice(c(37, 55, 65, 81, 125))
#>   Rank                 Title Director Year Rating Revenue
#> 1   37          Interstellar C. Nolan 2014    8.6  187.99
#> 2   55       The Dark Knight C. Nolan 2008    9.0  533.32
#> 3   65          The Prestige C. Nolan 2006    8.5   53.08
#> 4   81             Inception C. Nolan 2010    8.8  292.57
#> 5  125 The Dark Knight Rises C. Nolan 2012    8.5  448.13
#>   Metascore
#> 1        74
#> 2        82
#> 3        66
#> 4        74
#> 5        78

# increasing revenues for movies produced by Christopher Nolan by 20%
mov %>%
  select(1, 2, 5, 7, 9, 11, 12) %>%
  mutate(Revenue = ifelse(Director == 'Christopher Nolan', Revenue * 1.2, Revenue)) %>%
  slice(c(37, 55, 65, 81, 125))
#>   Rank                 Title          Director Year Rating
```

```
#> 1   37          Interstellar Christopher Nolan 2014    8.6
#> 2   55       The Dark Knight Christopher Nolan 2008    9.0
#> 3   65          The Prestige Christopher Nolan 2006    8.5
#> 4   81             Inception Christopher Nolan 2010    8.8
#> 5  125 The Dark Knight Rises Christopher Nolan 2012    8.5
#>   Revenue Metascore
#> 1 225.588        74
#> 2 639.984        82
#> 3  63.696        66
#> 4 351.084        74
#> 5 537.756        78

# replacing a value using if_else()
mov %>%
  select(1, 2, 5, 7, 9, 11, 12) %>%
  mutate(Director = if_else(Director == 'Christopher Nolan', 'C. Nolan', Director)) %>%
  slice(c(37, 55, 65, 81, 125))
#>   Rank                 Title Director Year Rating Revenue
#> 1   37          Interstellar C. Nolan 2014    8.6  187.99
#> 2   55       The Dark Knight C. Nolan 2008    9.0  533.32
#> 3   65          The Prestige C. Nolan 2006    8.5   53.08
#> 4   81             Inception C. Nolan 2010    8.8  292.57
#> 5  125 The Dark Knight Rises C. Nolan 2012    8.5  448.13
#>   Metascore
#> 1        74
#> 2        82
#> 3        66
#> 4        74
#> 5        78

# replacing a value using replace()
select(mov, 2, 5, 7, 9, 11, 12) %>%
  mutate(Director = replace(Director, Director == 'Christopher Nolan', 'C. Nolan')) %>%
  slice(c(37, 55, 65, 81, 125))
#>                  Title Director Year Rating Revenue
#> 1          Interstellar C. Nolan 2014    8.6  187.99
#> 2       The Dark Knight C. Nolan 2008    9.0  533.32
#> 3          The Prestige C. Nolan 2006    8.5   53.08
#> 4             Inception C. Nolan 2010    8.8  292.57
#> 5 The Dark Knight Rises C. Nolan 2012    8.5  448.13
#>   Metascore
#> 1        74
#> 2        82
#> 3        66
#> 4        74
```

```
#> 5        78

# increasing revenues for movies produced by Christopher Nolan by 20%
select(mov, 2, 5, 7, 9, 11, 12) %>%
  mutate(Revenue = replace(Director, Director == 'Christopher Nolan',
                           Revenue[Director == 'Christopher Nolan'] * 1.2)) %>%
  slice(c(37, 55, 65, 81, 125))
#>                   Title          Director Year Rating
#> 1          Interstellar Christopher Nolan 2014    8.6
#> 2       The Dark Knight Christopher Nolan 2008    9.0
#> 3          The Prestige Christopher Nolan 2006    8.5
#> 4             Inception Christopher Nolan 2010    8.8
#> 5 The Dark Knight Rises Christopher Nolan 2012    8.5
#>   Revenue Metascore
#> 1 225.588        74
#> 2 639.984        82
#> 3  63.696        66
#> 4 351.084        74
#> 5 537.756        78
```

5.7.4 Deleting rows of data

There is no special function to delete rows but the functions `filter()` and `slice()` can be used to keep or drop rows.

5.7.4.1 Unique rows

The function `distinct()` removes duplicate rows.

```
# selecting unique values
select(mov, 2, 5, 7, 9, 11, 12) %>%
distinct(Year) %>%
pull()
#>  [1] 2014 2012 2016 2015 2007 2011 2008 2006 2009 2010 2013

# selecting unique rows
select(mov, 2, 5, 7, 9, 11, 12) %>%
distinct(Year, Director) %>%
nrow()
#> [1] 987
```

5.8 Combine data: concatenate, join and merge

5.8.1 Concatenating tibbles

Combining datasets using `bind_rows()`

The function `bind_rows()` acts like `rbind()` in Base R.

```
top_5 <- tibble(country = c('China', 'India', 'United States', 'Indonesia', 'Brazil'),
                continent = c('Asia', 'Asia', 'Americas', 'Asia', 'Americas'),
                population = c(1318683096, 1110396331, 301139947, 223547000, 190010647)
                lifeExpectancy = c(72.961, 64.698, 78.242, 70.65, 72.39))
top_5
#> # A tibble: 5 x 4
#>   country       continent population lifeExpectancy
#>   <chr>         <chr>          <dbl>          <dbl>
#> 1 China         Asia      1318683096           73.0
#> 2 India         Asia      1110396331           64.7
#> 3 United States Americas   301139947           78.2
#> 4 Indonesia     Asia       223547000           70.6
#> 5 Brazil        Americas   190010647           72.4

top_5_10 <- tibble(country = c('Pakistan', 'Bangladesh', 'Nigeria', 'Japan', 'Mexico')
                   continent = c('Asia', 'Asia', 'Africa', 'Asia', 'Americas'),
                   population = c(169270617, 150448339, 135031164, 127467972, 10870089
                   lifeExpectancy = c(65.483, 64.062, 46.859, 82.603, 76.195))
top_5_10
#> # A tibble: 5 x 4
#>   country    continent population lifeExpectancy
#>   <chr>      <chr>          <dbl>          <dbl>
#> 1 Pakistan   Asia       169270617           65.5
#> 2 Bangladesh Asia       150448339           64.1
#> 3 Nigeria    Africa     135031164           46.9
#> 4 Japan      Asia       127467972           82.6
#> 5 Mexico     Americas   108700891           76.2

# binding data frames
bind_rows(top_5, top_5_10)
#> # A tibble: 10 x 4
#>    country       continent population lifeExpectancy
#>    <chr>         <chr>          <dbl>          <dbl>
#>  1 China         Asia      1318683096           73.0
#>  2 India         Asia      1110396331           64.7
#>  3 United States Americas   301139947           78.2
#>  4 Indonesia     Asia       223547000           70.6
#>  5 Brazil        Americas   190010647           72.4
#>  6 Pakistan      Asia       169270617           65.5
#>  7 Bangladesh    Asia       150448339           64.1
#>  8 Nigeria       Africa     135031164           46.9
#>  9 Japan         Asia       127467972           82.6
#> 10 Mexico        Americas   108700891           76.2
```

5.8.2 Combining datasets using `bind_cols()`

The function `bind_cols()` acts like `cbind()` in Base R.

```
country <-
tibble(country = c('China', 'India', 'United States', 'Indonesia', 'Brazil'),
       continent = c('Asia', 'Asia', 'Americas', 'Asia', 'Americas'))
country
#> # A tibble: 5 x 2
#>   country       continent
#>   <chr>         <chr>
#> 1 China         Asia
#> 2 India         Asia
#> 3 United States Americas
#> 4 Indonesia     Asia
#> 5 Brazil        Americas

variables <-
tibble(country = c('China', 'India', 'United States', 'Indonesia', 'Brazil'),
       population = c(1318683096, 1110396331, 301139947, 223547000, 190010647),
       lifeExpectancy = c(72.961, 64.698, 78.242, 70.65, 72.39),
       perCapita = c(4959, 2452, 42952, 3541, 9066))
variables
#> # A tibble: 5 x 4
#>   country       population lifeExpectancy perCapita
#>   <chr>              <dbl>          <dbl>     <dbl>
#> 1 China         1318683096           73.0      4959
#> 2 India         1110396331           64.7      2452
#> 3 United States  301139947           78.2     42952
#> 4 Indonesia      223547000           70.6      3541
#> 5 Brazil         190010647           72.4      9066

# binding data frames
bind_cols(country, variables[-1])
#> # A tibble: 5 x 5
#>   country       continent population lifeExpectancy perCapita
#>   <chr>         <chr>          <dbl>          <dbl>     <dbl>
#> 1 China         Asia      1318683096           73.0      4959
#> 2 India         Asia      1110396331           64.7      2452
#> 3 United States Americas   301139947           78.2     42952
#> 4 Indonesia     Asia       223547000           70.6      3541
#> 5 Brazil        Americas   190010647           72.4      9066
```

5.8.3 Joining data frames using Set Operations

```
group_one <-
tibble(country = c('Ethiopia', 'Congo, Dem. Rep.', 'Egypt', 'United States',
                   'Mexico', 'India', 'Pakistan', 'Thailand', 'Japan'),
       population = c(76511887, 64606759, 80264543, 301139947, 108700891,
                      1110396331, 169270617, 65068149, 127467972))
group_one
#> # A tibble: 9 x 2
#>   country          population
#>   <chr>                 <dbl>
#> 1 Ethiopia           76511887
#> 2 Congo, Dem. Rep.   64606759
#> 3 Egypt              80264543
#> 4 United States     301139947
#> 5 Mexico            108700891
#> 6 India            1110396331
#> 7 Pakistan          169270617
#> 8 Thailand           65068149
#> 9 Japan             127467972
group_two <-
tibble(country = c('Ethiopia', 'Vietnam', 'Bangladesh', 'Thailand', 'India'),
       population = c(76511887, 85262356, 150448339, 65068149, 111039633))
group_two
#> # A tibble: 5 x 2
#>   country    population
#>   <chr>           <dbl>
#> 1 Ethiopia     76511887
#> 2 Vietnam      85262356
#> 3 Bangladesh  150448339
#> 4 Thailand     65068149
#> 5 India       111039633
```

5.8.3.1 Intersection

The function `intersect()` keeps rows that appear in both datasets.

```
intersect(group_one, group_two)
#> # A tibble: 2 x 2
#>   country  population
#>   <chr>         <dbl>
#> 1 Ethiopia   76511887
#> 2 Thailand   65068149
```

5.8.4 Union

The function `union()` keeps rows that appear in either of the datasets.

```
union(group_one, group_two)
```

5.8.5 Differences

The function `setdiff()` keeps rows that appear in the first dataset but not in the second.

```
setdiff(group_one, group_two)
#> # A tibble: 7 x 2
#>   country          population
#>   <chr>                 <dbl>
#> 1 Congo, Dem. Rep.   64606759
#> 2 Egypt              80264543
#> 3 United States     301139947
#> 4 Mexico            108700891
#> 5 India            1110396331
#> 6 Pakistan          169270617
#> 7 Japan             127467972
```

5.8.5.1 SQL like joins

```
# preparing data
employees <- tibble(
    name = c('john', 'mary', 'david', 'paul', 'susan', 'cynthia', 'Joss', 'dennis'),
    age = c(45, 55, 35, 58, 40, 30, 39, 25),
    gender = c('m', 'f', 'm', 'm', 'f', 'f', 'm', 'm'),
    salary =c(40000, 50000, 35000, 25000, 48000, 32000, 20000, 45000),
    department = c('commercial', 'production', NA, 'human resources',
                   'commercial', 'commercial', 'production', NA))
employees
#> # A tibble: 8 x 5
#>   name      age gender salary department
#>   <chr>   <dbl> <chr>   <dbl> <chr>
#> 1 john       45 m       40000 commercial
#> 2 mary       55 f       50000 production
#> 3 david      35 m       35000 <NA>
#> 4 paul       58 m       25000 human resources
#> 5 susan      40 f       48000 commercial
#> 6 cynthia    30 f       32000 commercial
#> 7 Joss       39 m       20000 production
#> 8 dennis     25 m       45000 <NA>

departments <- tibble(
```

```
    department = c('commercial', 'human resources', 'production', 'finance', 'maintenan
    location = c('washington', 'london', 'paris', 'dubai', 'dublin'))
departments
#> # A tibble: 5 x 2
#>   department      location
#>   <chr>           <chr>
#> 1 commercial      washington
#> 2 human resources london
#> 3 production      paris
#> 4 finance         dubai
#> 5 maintenance     dublin
```

5.8.6 Left join

The left join returns all records from the left dataset and the matched records from the right dataset. The result is NULL from the right side if there is no match.

```
left_join(employees, departments)
#> # A tibble: 8 x 6
#>   name      age gender salary department      location
#>   <chr>   <dbl> <chr>   <dbl> <chr>           <chr>
#> 1 john       45 m       40000 commercial      washington
#> 2 mary       55 f       50000 production      paris
#> 3 david      35 m       35000 <NA>            <NA>
#> 4 paul       58 m       25000 human resources london
#> 5 susan      40 f       48000 commercial      washington
#> 6 cynthia    30 f       32000 commercial      washington
#> 7 Joss       39 m       20000 production      paris
#> 8 dennis     25 m       45000 <NA>            <NA>
```

5.8.7 Right join

The right join returns all records from the right dataset, and the matched records from the left dataset. The result is NULL from the left side when there is no match.

```
right_join(employees, departments)
#> # A tibble: 8 x 6
#>   name      age gender salary department      location
#>   <chr>   <dbl> <chr>   <dbl> <chr>           <chr>
#> 1 john       45 m       40000 commercial      washington
#> 2 mary       55 f       50000 production      paris
#> 3 paul       58 m       25000 human resources london
#> 4 susan      40 f       48000 commercial      washington
#> 5 cynthia    30 f       32000 commercial      washington
```

```
#> 6 Joss        39 m        20000 production      paris
#> 7 <NA>        NA <NA>        NA finance         dubai
#> 8 <NA>        NA <NA>        NA maintenance     dublin

# reversing tables produces the same results as a left join
right_join(departments, employees)
#> # A tibble: 8 x 6
#>   department      location   name       age gender salary
#>   <chr>           <chr>      <chr>    <dbl> <chr>   <dbl>
#> 1 commercial      washington john        45 m       40000
#> 2 commercial      washington susan       40 f       48000
#> 3 commercial      washington cynthia     30 f       32000
#> 4 human resources london     paul        58 m       25000
#> 5 production      paris      mary        55 f       50000
#> 6 production      paris      Joss        39 m       20000
#> 7 <NA>            <NA>       david       35 m       35000
#> 8 <NA>            <NA>       dennis      25 m       45000
```

5.8.8 Inner join

The inner join selects records that have matching values in both datasets

```
inner_join(employees, departments)
#> # A tibble: 6 x 6
#>   name       age gender salary department      location
#>   <chr>    <dbl> <chr>   <dbl> <chr>           <chr>
#> 1 john        45 m       40000 commercial      washington
#> 2 mary        55 f       50000 production      paris
#> 3 paul        58 m       25000 human resources london
#> 4 susan       40 f       48000 commercial      washington
#> 5 cynthia     30 f       32000 commercial      washington
#> 6 Joss        39 m       20000 production      paris
```

5.8.9 Full join

The full join returns all records between the left and right dataset

```
full_join(employees, departments)
#> # A tibble: 10 x 6
#>    name       age gender salary department      location
#>    <chr>    <dbl> <chr>   <dbl> <chr>           <chr>
#>  1 john        45 m       40000 commercial      washington
#>  2 mary        55 f       50000 production      paris
#>  3 david       35 m       35000 <NA>            <NA>
#>  4 paul        58 m       25000 human resources london
#>  5 susan       40 f       48000 commercial      washington
```

```
#>  6 cynthia    30 f        32000 commercial      washington
#>  7 Joss       39 m        20000 production      paris
#>  8 dennis     25 m        45000 <NA>            <NA>
#>  9 <NA>       NA <NA>        NA finance         dubai
#> 10 <NA>       NA <NA>        NA maintenance     dublin
```

5.8.10 Anti join

The anti join returns all records found on the left dataset but absent in the right one

```
anti_join(employees, departments)
#> # A tibble: 2 x 5
#>   name      age gender salary department
#>   <chr>   <dbl> <chr>   <dbl> <chr>
#> 1 david      35 m       35000 <NA>
#> 2 dennis     25 m       45000 <NA>
```

Tibbles with different column names

```
# recreating employee table with different column names
employees <- tibble(
    name = c('john', 'mary', 'david', 'paul', 'susan', 'cynthia', 'Joss', 'dennis'),
    age = c(45, 55, 35, 58, 40, 30, 39, 25),
    gender = c('m', 'f', 'm', 'm', 'f', 'f', 'm', 'm'),
    salary =c(40000, 50000, 35000, 25000, 48000, 32000, 20000, 45000),
    dep_name = c('commercial', 'production', NA, 'human resources',
                 'commercial', 'commercial', 'production', NA))
employees
#> # A tibble: 8 x 5
#>   name       age gender salary dep_name
#>   <chr>    <dbl> <chr>   <dbl> <chr>
#> 1 john        45 m       40000 commercial
#> 2 mary        55 f       50000 production
#> 3 david       35 m       35000 <NA>
#> 4 paul        58 m       25000 human resources
#> 5 susan       40 f       48000 commercial
#> 6 cynthia     30 f       32000 commercial
#> 7 Joss        39 m       20000 production
#> 8 dennis      25 m       45000 <NA>

left_join(employees, departments, by = c('dep_name' = 'department'))
#> # A tibble: 8 x 6
#>   name       age gender salary dep_name        location
#>   <chr>    <dbl> <chr>   <dbl> <chr>           <chr>
#> 1 john        45 m       40000 commercial      washington
```

```
#> 2 mary       55 f       50000 production      paris
#> 3 david      35 m       35000 <NA>            <NA>
#> 4 paul       58 m       25000 human resources london
#> 5 susan      40 f       48000 commercial      washington
#> 6 cynthia    30 f       32000 commercial      washington
#> 7 Joss       39 m       20000 production      paris
#> 8 dennis     25 m       45000 <NA>            <NA>
```

Joining on more than one joining column

```
# adding a subdepartment
employees <- tibble(
    name = c('john', 'mary', 'david', 'paul', 'susan', 'cynthia', 'Joss', 'dennis'),
    age = c(45, 55, 35, 58, 40, 30, 39, 25),
    gender = c('m', 'f', 'm', 'm', 'f', 'f', 'm', 'm'),
    salary =c(40000, 50000, 35000, 25000, 48000, 32000, 20000, 45000),
    department = c('commercial', 'production', NA, 'human resources', 'commercial',
                   'commercial', 'production', NA),
    subdepartment = c('marketing', 'production', NA, 'human resources', 'sales',
                      'sales', 'production', NA))
employees
#> # A tibble: 8 x 6
#>   name      age gender salary department      subdepartment
#>   <chr>   <dbl> <chr>   <dbl> <chr>           <chr>
#> 1 john       45 m       40000 commercial      marketing
#> 2 mary       55 f       50000 production      production
#> 3 david      35 m       35000 <NA>            <NA>
#> 4 paul       58 m       25000 human resources human resourc~
#> 5 susan      40 f       48000 commercial      sales
#> 6 cynthia    30 f       32000 commercial      sales
#> 7 Joss       39 m       20000 production      production
#> 8 dennis     25 m       45000 <NA>            <NA>

departments <- tibble(
    department = c('commercial', 'commercial', 'human resources', 'production',
                   'finance', 'finance', 'maintenance'),
    subdepartment = c('marketing', 'sales', 'human resources', 'production', 'finance',
                      'accounting', 'maintenance'),
    location = c('washington', 'washington', 'london', 'paris', 'dubai', 'dubai', 'dublin'))
departments
#> # A tibble: 7 x 3
#>   department      subdepartment   location
#>   <chr>           <chr>           <chr>
#> 1 commercial      marketing       washington
#> 2 commercial      sales           washington
#> 3 human resources human resources london
```

```
#> 4 production      production      paris
#> 5 finance         finance         dubai
#> 6 finance         accounting      dubai
#> 7 maintenance     maintenance     dublin

# since columns have the same names, joining is done automatically
left_join(employees, departments)
#> # A tibble: 8 x 7
#>   name      age gender salary department      subdepartment
#>   <chr>   <dbl> <chr>   <dbl> <chr>           <chr>
#> 1 john       45 m       40000 commercial      marketing
#> 2 mary       55 f       50000 production      production
#> 3 david      35 m       35000 <NA>            <NA>
#> 4 paul       58 m       25000 human resources human resourc~
#> 5 susan      40 f       48000 commercial      sales
#> 6 cynthia    30 f       32000 commercial      sales
#> 7 Joss       39 m       20000 production      production
#> 8 dennis     25 m       45000 <NA>            <NA>
#> # ... with 1 more variable: location <chr>


# declaring column names explicitly
left_join(employees, departments, by = c("department", "subdepartment"))
#> # A tibble: 8 x 7
#>   name      age gender salary department      subdepartment
#>   <chr>   <dbl> <chr>   <dbl> <chr>           <chr>
#> 1 john       45 m       40000 commercial      marketing
#> 2 mary       55 f       50000 production      production
#> 3 david      35 m       35000 <NA>            <NA>
#> 4 paul       58 m       25000 human resources human resourc~
#> 5 susan      40 f       48000 commercial      sales
#> 6 cynthia    30 f       32000 commercial      sales
#> 7 Joss       39 m       20000 production      production
#> 8 dennis     25 m       45000 <NA>            <NA>
#> # ... with 1 more variable: location <chr>

# with different names
employees <- tibble(
    name = c('john', 'mary', 'david', 'paul', 'susan', 'cynthia', 'Joss', 'dennis'),
    age = c(45, 55, 35, 58, 40, 30, 39, 25),
    gender = c('m', 'f', 'm', 'm', 'f', 'f', 'm', 'm'),
    salary =c(40000, 50000, 35000, 25000, 48000, 32000, 20000, 45000),
    dep = c('commercial', 'production', NA, 'human resources', 'commercial',
            'commercial', 'production', NA),
    sub = c('marketing', 'production', NA, 'human resources', 'sales',
```

```
             'sales', 'production', NA))
employees
#> # A tibble: 8 x 6
#>   name      age gender salary dep             sub
#>   <chr>   <dbl> <chr>   <dbl> <chr>           <chr>
#> 1 john       45 m       40000 commercial      marketing
#> 2 mary       55 f       50000 production      production
#> 3 david      35 m       35000 <NA>            <NA>
#> 4 paul       58 m       25000 human resources human resourc~
#> 5 susan      40 f       48000 commercial      sales
#> 6 cynthia    30 f       32000 commercial      sales
#> 7 Joss       39 m       20000 production      production
#> 8 dennis     25 m       45000 <NA>            <NA>

left_join(employees, departments, by = c('dep' = 'department', 'sub' = 'subdepartment'))
#> # A tibble: 8 x 7
#>   name      age gender salary dep             sub   location
#>   <chr>   <dbl> <chr>   <dbl> <chr>           <chr> <chr>
#> 1 john       45 m       40000 commercial      mark~ washing~
#> 2 mary       55 f       50000 production      prod~ paris
#> 3 david      35 m       35000 <NA>            <NA>  <NA>
#> 4 paul       58 m       25000 human resources huma~ london
#> 5 susan      40 f       48000 commercial      sales washing~
#> 6 cynthia    30 f       32000 commercial      sales washing~
#> 7 Joss       39 m       20000 production      prod~ paris
#> 8 dennis     25 m       45000 <NA>            <NA>  <NA>
```

5.9 Aggregating and grouping data

5.9.1 Aggregating

The function `summarise()` aggregates data using various summarization functions from both Base R and dplyr itself. In addition to the summarization functions like `mean()`, `median()`, `sum()`, etc. Which come with base R, dplyr comes with the following:

- `n()` for counts of rows,
- `n_distinct()` for counts of unique elements
- `first()` for first value
- `last()` for last value
- `nth()` for nth value

```
data(gapminder)

# performing aggregations
```

```
gapminder %>%
  filter(year == 2007) %>%
  summarize(`total pop` = sum(pop, na.rm = T),
            `mean pop` = mean(pop, na.rm = T),
            `median pop` = median(pop, na.rm = T),
            `country count` = n())
#> # A tibble: 1 x 4
#>   `total pop` `mean pop` `median pop` `country count`
#>         <dbl>      <dbl>        <dbl>           <int>
#> 1  6251013179  44021220.     10517531             142
```

The function `summarise_at()` affects variables selected with a character vector or `vars()`.

```
# using multiple summarization function
gapminder %>%
  filter(year == 2007) %>%
  summarise_at(vars(lifeExp), list(mean = mean, median = median, count = ~n()))
#> # A tibble: 1 x 3
#>    mean median count
#>   <dbl>  <dbl> <int>
#> 1  67.0   71.9   142

gapminder %>%
  filter(year == 2007) %>%
  summarise_at(vars(lifeExp), list(~ mean(.), ~ median(.), ~ n()))
#> # A tibble: 1 x 3
#>    mean median     n
#>   <dbl>  <dbl> <int>
#> 1  67.0   71.9   142

# multiple columns with vars
gapminder %>%
  filter(year == 2007) %>%
  summarise_at(vars(lifeExp, gdpPercap), list(mean = mean, median = median))
#> # A tibble: 1 x 4
#>   lifeExp_mean gdpPercap_mean lifeExp_median
#>          <dbl>          <dbl>          <dbl>
#> 1         67.0         11680.           71.9
#> # ... with 1 more variable: gdpPercap_median <dbl>

# multiple columns with vectors
gapminder %>%
  filter(year == 2007) %>%
  summarise_at(c('lifeExp', 'gdpPercap'), list(mean = mean, median = median))
#> # A tibble: 1 x 4
```

```
#>   lifeExp_mean gdpPercap_mean lifeExp_median
#>          <dbl>          <dbl>          <dbl>
#> 1         67.0         11680.           71.9
#> # ... with 1 more variable: gdpPercap_median <dbl>

# using a custom function
gapminder %>%
  filter(year == 2007) %>%
  summarise_at(vars(lifeExp, gdpPercap), list(mean = function(x)round(mean(x), 1),
                                              median = function(x)round(median(x), 1)))
#> # A tibble: 1 x 4
#>   lifeExp_mean gdpPercap_mean lifeExp_median
#>          <dbl>          <dbl>          <dbl>
#> 1           67         11680.           71.9
#> # ... with 1 more variable: gdpPercap_median <dbl>
```

5.9.2 Grouping data

The function `group_by()` is used to group data while the function `ungroup()` is used to ungroup data after applying grouping. It is always a good idea to ungroup data after working with groupings as functions in dplyr will behave differently with grouped data.

```
# grouping by single column (continent)
gapminder %>%
  filter(year == 2007) %>%
  group_by(continent) %>%
  summarize(`total pop` = sum(pop, na.rm = T),
            `mean pop` = mean(pop, na.rm = T),
            `median pop` = median(pop, na.rm = T),
            `country count` = n()) %>%
  ungroup()
#> # A tibble: 5 x 5
#>   continent `total pop` `mean pop` `median pop`
#>   <fct>           <dbl>      <dbl>        <dbl>
#> 1 Africa      929539692  17875763.    10093310.
#> 2 Americas    898871184  35954847.     9319622
#> 3 Asia       3811953827 115513752.    24821286
#> 4 Europe      586098529  19536618.     9493598
#> 5 Oceania      24549947  12274974.    12274974.
#> # ... with 1 more variable: country count <int>

# grouping by two categorical columns (continent and year)
gapminder %>%
  filter(year %in% c(1987, 2007)) %>%
```

```
  group_by(continent, year) %>%
  summarize(`total pop` = sum(pop, na.rm = T),
            `mean pop` = mean(pop, na.rm = T),
            `median pop` = median(pop, na.rm = T),
            `country count` = n()) %>%
  ungroup()
#> # A tibble: 10 x 6
#>    continent  year `total pop` `mean pop` `median pop`
#>    <fct>     <int>       <dbl>      <dbl>        <dbl>
#>  1 Africa     1987   574834110  11054502.     6635612.
#>  2 Africa     2007   929539692  17875763.    10093310.
#>  3 Americas   1987   682753971  27310159.     6655297
#>  4 Americas   2007   898871184  35954847.     9319622
#>  5 Asia       1987  2871220762  87006690.    16495304
#>  6 Asia       2007  3811953827 115513752.    24821286
#>  7 Europe     1987   543094160  18103139.     9101370.
#>  8 Europe     2007   586098529  19536618.     9493598
#>  9 Oceania    1987    19574415   9787208.     9787208.
#> 10 Oceania    2007    24549947  12274974.    12274974.
#> # ... with 1 more variable: country count <int>

# sorting by group
gap_data <-
gapminder %>%
  group_by(year) %>%
  arrange(pop, .by_group = TRUE) %>%
  ungroup()
head(gap_data)
#> # A tibble: 6 x 6
#>   country               continent  year lifeExp    pop gdpPercap
#>   <fct>                 <fct>     <int>   <dbl>  <int>     <dbl>
#> 1 Sao Tome and Principe Africa     1952    46.5  60011      880.
#> 2 Djibouti              Africa     1952    34.8  63149     2670.
#> 3 Bahrain               Asia       1952    50.9 120447     9867.
#> 4 Iceland               Europe     1952    72.5 147962     7268.
#> 5 Comoros               Africa     1952    40.7 153936     1103.
#> 6 Kuwait                Asia       1952    55.6 160000   108382.

tail(gap_data)
#> # A tibble: 6 x 6
#>   country       continent  year lifeExp        pop gdpPercap
#>   <fct>         <fct>     <int>   <dbl>      <int>     <dbl>
#> 1 Pakistan      Asia       2007    65.5  169270617     2606.
#> 2 Brazil        Americas   2007    72.4  190010647     9066.
#> 3 Indonesia     Asia       2007    70.6  223547000     3541.
```

```
#> 4 United States Americas   2007    78.2  301139947    42952.
#> 5 India         Asia       2007    64.7 1110396331     2452.
#> 6 China         Asia       2007    73.0 1318683096     4959.

# ranking by group
select(mov, Title, Year, Revenue, Metascore) %>%
  arrange(Year, Revenue) %>%
  group_by(Year) %>%
  mutate(rank_by_revenue = rank(Revenue, ties.method = "first")) %>%
  ungroup() %>%
  slice(43:47)
#> # A tibble: 5 x 5
#>   Title              Year Revenue Metascore rank_by_revenue
#>   <chr>             <int>   <dbl>     <int>           <int>
#> 1 Deja Vu            2006   NA           NA              43
#> 2 Inland Empire      2006   NA           NA              44
#> 3 The Babysitters    2007    0.04        35               1
#> 4 Taare Zameen Par   2007    1.2         42               2
#> 5 Funny Games        2007    1.29        NA               3

## NB: Notice as ranking restarts once as 2007 is reached.
```

5.9.3 Splitting data frame by groups

The `group_split()` is like `base::split()` in that it splits a data frame.

```
movies_year <-
select(mov, Title, Year, Revenue, Metascore) %>%
  group_split(Year)
length(movies_year)
#> [1] 11
movies_year[1]
#> <list_of<
#>   tbl_df<
#>     Title    : character
#>     Year     : integer
#>     Revenue  : double
#>     Metascore: integer
#>   >
#> >[1]>
#> [[1]]
#> # A tibble: 44 x 4
#>    Title                                Year Revenue Metascore
#>    <chr>                               <int>   <dbl>     <int>
#>  1 The Prestige                         2006    53.1        66
```

```
#>  2 Pirates of the Caribbean: Dead M~  2006   423.          53
#>  3 The Departed                       2006   132.          85
#>  4 300                                2006   211.          52
#>  5 Casino Royale                      2006   167.          80
#>  6 Cars                               2006   244.          73
#>  7 Pan's Labyrinth                    2006    37.6         98
#>  8 Apocalypto                         2006    50.9         68
#>  9 Children of Men                    2006    35.3         84
#> 10 The Devil Wears Prada              2006   125.          62
#> # ... with 34 more rows
```

5.10 Pivoting and unpivoting data with tidyr

5.10.1 Pivoting

The function `pivot_wider()` pivots data that is converting it from long to wide. It expects the following:

- names_from: rows to move to columns
- values_from: values to be placed between the intersection of rows and columns (cell values)

```
library(tidyr)

# preparing data
dt <-
gapminder %>%
  filter(year %in% c(1987, 1997, 2007)) %>%
  group_by(continent, year) %>%
  summarize(total_pop = sum(pop, na.rm = T)) %>%
  ungroup()
dt
#> # A tibble: 15 x 3
#>    continent  year  total_pop
#>    <fct>     <int>      <dbl>
#>  1 Africa     1987  574834110
#>  2 Africa     1997  743832984
#>  3 Africa     2007  929539692
#>  4 Americas   1987  682753971
#>  5 Americas   1997  796900410
#>  6 Americas   2007  898871184
#>  7 Asia       1987 2871220762
#>  8 Asia       1997 3383285500
#>  9 Asia       2007 3811953827
#> 10 Europe     1987  543094160
```

```
#> 11 Europe     1997  568944148
#> 12 Europe     2007  586098529
#> 13 Oceania    1987   19574415
#> 14 Oceania    1997   22241430
#> 15 Oceania    2007   24549947

# pivoting data
dt %>%
  pivot_wider(names_from = year, values_from = total_pop, names_prefix = 'Y')
#> # A tibble: 5 x 4
#>   continent       Y1987      Y1997      Y2007
#>   <fct>           <dbl>      <dbl>      <dbl>
#> 1 Africa      574834110  743832984  929539692
#> 2 Americas    682753971  796900410  898871184
#> 3 Asia       2871220762 3383285500 3811953827
#> 4 Europe      543094160  568944148  586098529
#> 5 Oceania      19574415   22241430   24549947
```

5.10.2 Unpivoting

The function `pivot_longer()` unpivots data, that is converting it from wide to long. It expects:

- cols: columns to move to row
- names_to: name of the new column for moved columns
- values_to: name of the new column for moved cell values

```
# preparing data
dt_wide <-
dt %>%
  pivot_wider(names_from = year, values_from = total_pop, names_prefix = 'Y')
dt_wide
#> # A tibble: 5 x 4
#>   continent       Y1987      Y1997      Y2007
#>   <fct>           <dbl>      <dbl>      <dbl>
#> 1 Africa      574834110  743832984  929539692
#> 2 Americas    682753971  796900410  898871184
#> 3 Asia       2871220762 3383285500 3811953827
#> 4 Europe      543094160  568944148  586098529
#> 5 Oceania      19574415   22241430   24549947

# unpivoting data
dt_wide %>%
  pivot_longer(cols = c(Y1987, Y1997, Y2007)) %>%
  head()
#> # A tibble: 6 x 3
```

```
#>   continent name      value
#>   <fct>     <chr>     <dbl>
#> 1 Africa    Y1987 574834110
#> 2 Africa    Y1997 743832984
#> 3 Africa    Y2007 929539692
#> 4 Americas  Y1987 682753971
#> 5 Americas  Y1997 796900410
#> 6 Americas  Y2007 898871184

# replacing name and value
dt_wide %>%
  pivot_longer(cols = c(Y1987, Y1997, Y2007), names_to = 'year', values_to = 'populatio
  head()
#> # A tibble: 6 x 3
#>   continent year  population
#>   <fct>     <chr>      <dbl>
#> 1 Africa    Y1987  574834110
#> 2 Africa    Y1997  743832984
#> 3 Africa    Y2007  929539692
#> 4 Americas  Y1987  682753971
#> 5 Americas  Y1997  796900410
#> 6 Americas  Y2007  898871184
```

5.11 Dealing with duplicate values with dplyr

The function `distinct()` is used to extract unique values while `n_distinct()` returns the count of unique values.

```
library(readr)
library(dplyr)

# reading data
movies <- read.table(file = "data/IMDB-Movie-Data.csv", header = T, sep = ",", dec = "
                  comment.char = "")


# preparing data
movies %>%
select(7, 12) %>%
filter(Year == 2006) %>%
arrange(Metascore) %>%
head()
#>   Year Metascore
#> 1 2006        36
#> 2 2006        45
```

```
#> 3 2006        45
#> 4 2006        45
#> 5 2006        46
#> 6 2006        47

# extracting unique values
movies %>%
select(7, 12) %>%
filter(Year == 2006) %>%
arrange(Metascore) %>%
distinct() %>%
head()
#>   Year Metascore
#> 1 2006        36
#> 2 2006        45
#> 3 2006        46
#> 4 2006        47
#> 5 2006        48
#> 6 2006        51

# count of unique values
movies %>%
select(7, 12) %>%
filter(Year == 2006) %>%
arrange(Year, Metascore) %>%
n_distinct()
#> [1] 27

# extracting unique values by column
movies %>%
arrange(Year, Metascore) %>%
distinct(Year)
#>    Year
#> 1  2006
#> 2  2007
#> 3  2008
#> 4  2009
#> 5  2010
#> 6  2011
#> 7  2012
#> 8  2013
#> 9  2014
#> 10 2015
#> 11 2016
```

```
# keeping other columns
movies %>%
select(7, 12) %>%
arrange(Year, Metascore) %>%
distinct(Year, .keep_all= TRUE)
#>    Year Metascore
#> 1  2006        36
#> 2  2007        29
#> 3  2008        15
#> 4  2009        23
#> 5  2010        20
#> 6  2011        31
#> 7  2012        31
#> 8  2013        18
#> 9  2014        22
#> 10 2015        18
#> 11 2016        11
```

5.12 Dealing with NA values with tidyr

5.12.1 Replacing missing values by LOCF

The function `fill()` performs NA replacement both by LOCF and NOCB.

```
library(tidyr)


# reading data
movies <- read.table(file = "data/IMDB-Movie-Data.csv", header = T, sep = ",", dec = "
                  comment.char = "")


names(movies)[c(2,7,11,12)] <- c('Title', 'Year', 'RevenueMillions', 'Metascore')

# replacing NA values to values that precede it
movies %>%
dplyr::arrange(Year) %>%
fill(RevenueMillions, .direction = "down") %>%
tail(10)
#>      Rank                          Title
#> 991   948                     King Cobra
#> 992   950                          Kicks
#> 993   965                        Custody
#> 994   967                      L'odyssée
#> 995   975                 Queen of Katwe
```

```
#> 996   976     My Big Fat Greek Wedding 2
#> 997   978                  Amateur Night
#> 998   979 It's Only the End of the World
#> 999   981           Miracles from Heaven
#> 1000 1000                     Nine Lives
#>                          Genre
#> 991             Crime,Drama
#> 992               Adventure
#> 993                   Drama
#> 994     Adventure,Biography
#> 995   Biography,Drama,Sport
#> 996   Comedy,Family,Romance
#> 997                  Comedy
#> 998                   Drama
#> 999  Biography,Drama,Family
#> 1000  Comedy,Family,Fantasy
#>
#> 991                            This ripped-from-the-headlines drama covers the early rise of g
#> 992  Brandon is a 15 year old whose dream is a pair of fresh Air Jordans. Soon after he gets h
#> 993
#> 994                Highly influential and a fearlessly ambitious pioneer, innovator, filmmake
#> 995
#> 996
#> 997                               Guy Carter is an award-winning graduate student of architect
#> 998
#> 999
#> 1000
#>              Director
#> 991       Justin Kelly
#> 992     Justin Tipping
#> 993       James Lapine
#> 994       Jérôme Salle
#> 995          Mira Nair
#> 996         Kirk Jones
#> 997       Lisa Addario
#> 998       Xavier Dolan
#> 999   Patricia Riggen
#> 1000 Barry Sonnenfeld
#>                                                                                     Actors
#> 991                 Garrett Clayton, Christian Slater, Molly Ringwald,James Kelley
#> 992  Jahking Guillory, Christopher Jordan Wallace,Christopher Meyer, Kofi Siriboe
#> 993        Viola Davis, Hayden Panettiere, Catalina Sandino Moreno, Ellen Burstyn
#> 994                    Lambert Wilson, Pierre Niney, Audrey Tautou,Laurent Lucas
#> 995                 Madina Nalwanga, David Oyelowo, Lupita Nyong'o, Martin Kabanza
#> 996                  Nia Vardalos, John Corbett, Michael Constantine, Lainie Kazan
```

```
#> 997                  Jason Biggs, Janet Montgomery,Ashley Tisdale, Bria L. Murphy
#> 998                  Nathalie Baye, Vincent Cassel, Marion Cotillard, Léa Seydoux
#> 999             Jennifer Garner, Kylie Rogers, Martin Henderson,Brighton Sharbino
#> 1000                     Kevin Spacey, Jennifer Garner, Robbie Amell,Cheryl Hines
#>      Year Runtime..Minutes. Rating Votes RevenueMillions
#> 991  2016                91    5.6  3990            0.03
#> 992  2016                80    6.1  2417            0.15
#> 993  2016               104    6.9   280            0.15
#> 994  2016               122    6.7  1810            0.15
#> 995  2016               124    7.4  6753            8.81
#> 996  2016                94    6.0 20966           59.57
#> 997  2016                92    5.0  2229           59.57
#> 998  2016                97    7.0 10658           59.57
#> 999  2016               109    7.0 12048           61.69
#> 1000 2016                87    5.3 12435           19.64
#>      Metascore
#> 991         48
#> 992         69
#> 993         72
#> 994         70
#> 995         73
#> 996         37
#> 997         38
#> 998         48
#> 999         44
#> 1000        11
```

5.12.2 Replacing missing values by NOCB

```
# replacing NA values with proceeding values
fill(movies, RevenueMillions, .direction = "up") %>%
head(10)
#>    Rank                   Title                     Genre
#> 1     1 Guardians of the Galaxy    Action,Adventure,Sci-Fi
#> 2     2              Prometheus   Adventure,Mystery,Sci-Fi
#> 3     3                   Split            Horror,Thriller
#> 4     4                    Sing    Animation,Comedy,Family
#> 5     5           Suicide Squad   Action,Adventure,Fantasy
#> 6     6          The Great Wall   Action,Adventure,Fantasy
#> 7     7              La La Land         Comedy,Drama,Music
#> 8     8                Mindhorn                     Comedy
#> 9     9      The Lost City of Z Action,Adventure,Biography
#> 10   10              Passengers    Adventure,Drama,Romance
#>
#> 1
```

```
#> 2
#> 3                                                                                           Three gir
#> 4                  In a city of humanoid animals, a hustling theater impresario's attempt to
#> 5                                                                            A secret government agency recr
#> 6
#> 7
#> 8  A has-been actor best known for playing the title character in the 1980s detective series "
#> 9                                                                                               A tru
#> 10                                                                       A spacecraft traveling to a distant
#>               Director
#> 1            James Gunn
#> 2          Ridley Scott
#> 3    M. Night Shyamalan
#> 4  Christophe Lourdelet
#> 5            David Ayer
#> 6           Yimou Zhang
#> 7       Damien Chazelle
#> 8            Sean Foley
#> 9            James Gray
#> 10        Morten Tyldum
#>                                                                                Actors
#> 1                            Chris Pratt, Vin Diesel, Bradley Cooper, Zoe Saldana
#> 2      Noomi Rapace, Logan Marshall-Green, Michael Fassbender, Charlize Theron
#> 3             James McAvoy, Anya Taylor-Joy, Haley Lu Richardson, Jessica Sula
#> 4  Matthew McConaughey,Reese Witherspoon, Seth MacFarlane, Scarlett Johansson
#> 5                             Will Smith, Jared Leto, Margot Robbie, Viola Davis
#> 6                                  Matt Damon, Tian Jing, Willem Dafoe, Andy Lau
#> 7                       Ryan Gosling, Emma Stone, Rosemarie DeWitt, J.K. Simmons
#> 8             Essie Davis, Andrea Riseborough, Julian Barratt,Kenneth Branagh
#> 9                Charlie Hunnam, Robert Pattinson, Sienna Miller, Tom Holland
#> 10           Jennifer Lawrence, Chris Pratt, Michael Sheen,Laurence Fishburne
#>    Year Runtime..Minutes. Rating  Votes RevenueMillions
#> 1  2014               121    8.1 757074          333.13
#> 2  2012               124    7.0 485820          126.46
#> 3  2016               117    7.3 157606          138.12
#> 4  2016               108    7.2  60545          270.32
#> 5  2016               123    6.2 393727          325.02
#> 6  2016               103    6.1  56036           45.13
#> 7  2016               128    8.3 258682          151.06
#> 8  2016                89    6.4   2490            8.01
#> 9  2016               141    7.1   7188            8.01
#> 10 2016               116    7.0 192177          100.01
#>    Metascore
#> 1         76
#> 2         65
```

```
#> 3         62
#> 4         59
#> 5         40
#> 6         42
#> 7         93
#> 8         71
#> 9         78
#> 10        41

# on more than one column
fill(movies, c(RevenueMillions, Metascore), .direction = "up") %>%
head(10)
#>    Rank                   Title                      Genre
#> 1     1 Guardians of the Galaxy    Action,Adventure,Sci-Fi
#> 2     2              Prometheus   Adventure,Mystery,Sci-Fi
#> 3     3                   Split            Horror,Thriller
#> 4     4                    Sing    Animation,Comedy,Family
#> 5     5           Suicide Squad   Action,Adventure,Fantasy
#> 6     6          The Great Wall   Action,Adventure,Fantasy
#> 7     7              La La Land         Comedy,Drama,Music
#> 8     8                Mindhorn                     Comedy
#> 9     9      The Lost City of Z Action,Adventure,Biography
#> 10   10              Passengers    Adventure,Drama,Romance
#>
#> 1
#> 2
#> 3
#> 4                   In a city of humanoid animals, a hustling theater impresario's 
#> 5                                                                       A secret government 
#> 6
#> 7
#> 8  A has-been actor best known for playing the title character in the 1980s detecti
#> 9
#> 10                                                                 A spacecraft traveling to
#>                Director
#> 1            James Gunn
#> 2          Ridley Scott
#> 3    M. Night Shyamalan
#> 4  Christophe Lourdelet
#> 5            David Ayer
#> 6           Yimou Zhang
#> 7       Damien Chazelle
#> 8            Sean Foley
#> 9            James Gray
#> 10        Morten Tyldum
```

```
#>                                                                    Actors
#> 1                          Chris Pratt, Vin Diesel, Bradley Cooper, Zoe Saldana
#> 2     Noomi Rapace, Logan Marshall-Green, Michael Fassbender, Charlize Theron
#> 3            James McAvoy, Anya Taylor-Joy, Haley Lu Richardson, Jessica Sula
#> 4  Matthew McConaughey,Reese Witherspoon, Seth MacFarlane, Scarlett Johansson
#> 5                            Will Smith, Jared Leto, Margot Robbie, Viola Davis
#> 6                                 Matt Damon, Tian Jing, Willem Dafoe, Andy Lau
#> 7                        Ryan Gosling, Emma Stone, Rosemarie DeWitt, J.K. Simmons
#> 8              Essie Davis, Andrea Riseborough, Julian Barratt,Kenneth Branagh
#> 9                 Charlie Hunnam, Robert Pattinson, Sienna Miller, Tom Holland
#> 10           Jennifer Lawrence, Chris Pratt, Michael Sheen,Laurence Fishburne
#>    Year Runtime..Minutes. Rating  Votes RevenueMillions
#> 1  2014               121    8.1 757074          333.13
#> 2  2012               124    7.0 485820          126.46
#> 3  2016               117    7.3 157606          138.12
#> 4  2016               108    7.2  60545          270.32
#> 5  2016               123    6.2 393727          325.02
#> 6  2016               103    6.1  56036           45.13
#> 7  2016               128    8.3 258682          151.06
#> 8  2016                89    6.4   2490            8.01
#> 9  2016               141    7.1   7188            8.01
#> 10 2016               116    7.0 192177          100.01
#>    Metascore
#> 1         76
#> 2         65
#> 3         62
#> 4         59
#> 5         40
#> 6         42
#> 7         93
#> 8         71
#> 9         78
#> 10        41

fill(movies, RevenueMillions:Metascore, .direction = "up") %>%
head()
#>   Rank                   Title                    Genre
#> 1    1 Guardians of the Galaxy  Action,Adventure,Sci-Fi
#> 2    2              Prometheus Adventure,Mystery,Sci-Fi
#> 3    3                   Split          Horror,Thriller
#> 4    4                    Sing  Animation,Comedy,Family
#> 5    5           Suicide Squad Action,Adventure,Fantasy
#> 6    6          The Great Wall Action,Adventure,Fantasy
#>
#> 1
```

```
#> 2
#> 3                                                                  Three girls are k
#> 4 In a city of humanoid animals, a hustling theater impresario's attempt to save his
#> 5                                             A secret government agency recruits som
#> 6                                                                        Europe
#>              Director
#> 1          James Gunn
#> 2        Ridley Scott
#> 3   M. Night Shyamalan
#> 4 Christophe Lourdelet
#> 5           David Ayer
#> 6         Yimou Zhang
#>                                                                          Actors
#> 1                         Chris Pratt, Vin Diesel, Bradley Cooper, Zoe Saldana
#> 2    Noomi Rapace, Logan Marshall-Green, Michael Fassbender, Charlize Theron
#> 3          James McAvoy, Anya Taylor-Joy, Haley Lu Richardson, Jessica Sula
#> 4 Matthew McConaughey,Reese Witherspoon, Seth MacFarlane, Scarlett Johansson
#> 5                           Will Smith, Jared Leto, Margot Robbie, Viola Davis
#> 6                              Matt Damon, Tian Jing, Willem Dafoe, Andy Lau
#>   Year Runtime..Minutes. Rating  Votes RevenueMillions
#> 1 2014               121    8.1 757074          333.13
#> 2 2012               124    7.0 485820          126.46
#> 3 2016               117    7.3 157606          138.12
#> 4 2016               108    7.2  60545          270.32
#> 5 2016               123    6.2 393727          325.02
#> 6 2016               103    6.1  56036           45.13
#>   Metascore
#> 1        76
#> 2        65
#> 3        62
#> 4        59
#> 5        40
#> 6        42
```

5.12.3 Replacing NA values by a constant

The function `replace_na()` replaces NA values with a constant value. It requires a named list of column names and values to replace NA values with. Pass in empty strings for the columns not to be affected.

```
# creating a named list of column values
lst <- list('','', 200, 50)
names(lst) <- names(movies)[1:3]
lst
#> $Rank
```

```
#> [1] ""
#>
#> $Title
#> [1] ""
#>
#> $Genre
#> [1] 200
#>
#> $<NA>
#> [1] 50

# replacing NA values with the named list
replace_na(movies, lst) %>%
head(10)
#>    Rank                   Title                     Genre
#> 1     1 Guardians of the Galaxy   Action,Adventure,Sci-Fi
#> 2     2              Prometheus  Adventure,Mystery,Sci-Fi
#> 3     3                   Split           Horror,Thriller
#> 4     4                    Sing   Animation,Comedy,Family
#> 5     5           Suicide Squad  Action,Adventure,Fantasy
#> 6     6          The Great Wall  Action,Adventure,Fantasy
#> 7     7              La La Land        Comedy,Drama,Music
#> 8     8                Mindhorn                    Comedy
#> 9     9      The Lost City of Z Action,Adventure,Biography
#> 10   10              Passengers   Adventure,Drama,Romance
#>
#> 1
#> 2
#> 3                                                                                     Three gir
#> 4                  In a city of humanoid animals, a hustling theater impresario's attempt to
#> 5                                                                A secret government agency recr
#> 6
#> 7
#> 8  A has-been actor best known for playing the title character in the 1980s detective series '
#> 9                                                                                          A tru
#> 10                                                            A spacecraft traveling to a distant
#>                 Director
#> 1             James Gunn
#> 2           Ridley Scott
#> 3     M. Night Shyamalan
#> 4   Christophe Lourdelet
#> 5             David Ayer
#> 6            Yimou Zhang
#> 7        Damien Chazelle
#> 8            Sean Foley
```

```
#> 9            James Gray
#> 10        Morten Tyldum
#>                                                                           Actors
#> 1                            Chris Pratt, Vin Diesel, Bradley Cooper, Zoe Saldana
#> 2      Noomi Rapace, Logan Marshall-Green, Michael Fassbender, Charlize Theron
#> 3             James McAvoy, Anya Taylor-Joy, Haley Lu Richardson, Jessica Sula
#> 4  Matthew McConaughey,Reese Witherspoon, Seth MacFarlane, Scarlett Johansson
#> 5                              Will Smith, Jared Leto, Margot Robbie, Viola Davis
#> 6                                   Matt Damon, Tian Jing, Willem Dafoe, Andy Lau
#> 7                        Ryan Gosling, Emma Stone, Rosemarie DeWitt, J.K. Simmons
#> 8              Essie Davis, Andrea Riseborough, Julian Barratt,Kenneth Branagh
#> 9                 Charlie Hunnam, Robert Pattinson, Sienna Miller, Tom Holland
#> 10           Jennifer Lawrence, Chris Pratt, Michael Sheen,Laurence Fishburne
#>    Year Runtime..Minutes. Rating  Votes RevenueMillions
#> 1  2014               121    8.1 757074          333.13
#> 2  2012               124    7.0 485820          126.46
#> 3  2016               117    7.3 157606          138.12
#> 4  2016               108    7.2  60545          270.32
#> 5  2016               123    6.2 393727          325.02
#> 6  2016               103    6.1  56036           45.13
#> 7  2016               128    8.3 258682          151.06
#> 8  2016                89    6.4   2490              NA
#> 9  2016               141    7.1   7188            8.01
#> 10 2016               116    7.0 192177          100.01
#>    Metascore
#> 1         76
#> 2         65
#> 3         62
#> 4         59
#> 5         40
#> 6         42
#> 7         93
#> 8         71
#> 9         78
#> 10        41

# creating named list of computed values
lst <- list('',
            '',
            round(median(movies$RevenueMillions, na.rm = T), 2),
            round(mean(movies$Metascore, na.rm = T)))
names(lst) <- names(movies)[1:4]
lst
#> $Rank
#> [1] ""
```

```
#>
#> $Title
#> [1] ""
#>
#> $Genre
#> [1] 47.98
#>
#> $Description
#> [1] 59

# replacing NA values
replace_na(movies, lst) %>%
head(10)
#>    Rank                   Title                     Genre
#> 1     1 Guardians of the Galaxy   Action,Adventure,Sci-Fi
#> 2     2              Prometheus  Adventure,Mystery,Sci-Fi
#> 3     3                   Split           Horror,Thriller
#> 4     4                    Sing   Animation,Comedy,Family
#> 5     5           Suicide Squad  Action,Adventure,Fantasy
#> 6     6          The Great Wall  Action,Adventure,Fantasy
#> 7     7              La La Land        Comedy,Drama,Music
#> 8     8                Mindhorn                    Comedy
#> 9     9      The Lost City of Z Action,Adventure,Biography
#> 10   10              Passengers   Adventure,Drama,Romance
#>
#> 1
#> 2
#> 3                                                                              Three gi
#> 4                  In a city of humanoid animals, a hustling theater impresario's attempt to
#> 5                                                                  A secret government agency rec
#> 6
#> 7
#> 8  A has-been actor best known for playing the title character in the 1980s detective series "
#> 9                                                                                        A tr
#> 10                                                         A spacecraft traveling to a distant
#>               Director
#> 1           James Gunn
#> 2         Ridley Scott
#> 3    M. Night Shyamalan
#> 4  Christophe Lourdelet
#> 5           David Ayer
#> 6          Yimou Zhang
#> 7      Damien Chazelle
#> 8           Sean Foley
#> 9           James Gray
```

```
#> 10       Morten Tyldum
#>                                                                           Actors
#> 1                         Chris Pratt, Vin Diesel, Bradley Cooper, Zoe Saldana
#> 2     Noomi Rapace, Logan Marshall-Green, Michael Fassbender, Charlize Theron
#> 3            James McAvoy, Anya Taylor-Joy, Haley Lu Richardson, Jessica Sula
#> 4  Matthew McConaughey,Reese Witherspoon, Seth MacFarlane, Scarlett Johansson
#> 5                           Will Smith, Jared Leto, Margot Robbie, Viola Davis
#> 6                                Matt Damon, Tian Jing, Willem Dafoe, Andy Lau
#> 7                      Ryan Gosling, Emma Stone, Rosemarie DeWitt, J.K. Simmons
#> 8              Essie Davis, Andrea Riseborough, Julian Barratt,Kenneth Branagh
#> 9                 Charlie Hunnam, Robert Pattinson, Sienna Miller, Tom Holland
#> 10           Jennifer Lawrence, Chris Pratt, Michael Sheen,Laurence Fishburne
#>    Year Runtime..Minutes. Rating  Votes RevenueMillions
#> 1  2014               121    8.1 757074          333.13
#> 2  2012               124    7.0 485820          126.46
#> 3  2016               117    7.3 157606          138.12
#> 4  2016               108    7.2  60545          270.32
#> 5  2016               123    6.2 393727          325.02
#> 6  2016               103    6.1  56036           45.13
#> 7  2016               128    8.3 258682          151.06
#> 8  2016                89    6.4   2490              NA
#> 9  2016               141    7.1   7188            8.01
#> 10 2016               116    7.0 192177          100.01
#>    Metascore
#> 1         76
#> 2         65
#> 3         62
#> 4         59
#> 5         40
#> 6         42
#> 7         93
#> 8         71
#> 9         78
#> 10        41
```

5.12.4 Replacing NA values by groups

```
# splitting data frame
movies_split <- base::split(movies, movies$Year)

# replacing NA values
lapply(movies_split, function(x) {
    lst <- list('',
                '',
```

```
                round(median(x[x$RevenueMillions, 'RevenueMillions'], na.rm = T),2),
                round(mean(x[x$Metascore, 'Metascore'], na.rm = T)))
    names(lst) <- names(movies)[1:4]
    x <- replace_na(x, lst)
    return(x)
}) %>%
dplyr::bind_rows() %>%
tail(10)
#>      Rank                          Title
#> 991   948                     King Cobra
#> 992   950                          Kicks
#> 993   965                        Custody
#> 994   967                       L'odyssée
#> 995   975                 Queen of Katwe
#> 996   976      My Big Fat Greek Wedding 2
#> 997   978                   Amateur Night
#> 998   979 It's Only the End of the World
#> 999   981             Miracles from Heaven
#> 1000 1000                      Nine Lives
#>                           Genre
#> 991                Crime,Drama
#> 992                  Adventure
#> 993                      Drama
#> 994        Adventure,Biography
#> 995     Biography,Drama,Sport
#> 996     Comedy,Family,Romance
#> 997                     Comedy
#> 998                      Drama
#> 999    Biography,Drama,Family
#> 1000    Comedy,Family,Fantasy
#>
#> 991                              This ripped-from-the-headlines drama covers the early rise of g
#> 992  Brandon is a 15 year old whose dream is a pair of fresh Air Jordans. Soon after he gets h
#> 993
#> 994                  Highly influential and a fearlessly ambitious pioneer, innovator, filmmake
#> 995
#> 996
#> 997                                 Guy Carter is an award-winning graduate student of architect
#> 998
#> 999
#> 1000
#>               Director
#> 991        Justin Kelly
#> 992      Justin Tipping
#> 993        James Lapine
```

```
#> 994      Jérôme Salle
#> 995         Mira Nair
#> 996        Kirk Jones
#> 997      Lisa Addario
#> 998      Xavier Dolan
#> 999   Patricia Riggen
#> 1000 Barry Sonnenfeld
#>                                                                                 Actors
#> 991                  Garrett Clayton, Christian Slater, Molly Ringwald,James Kelley
#> 992  Jahking Guillory, Christopher Jordan Wallace,Christopher Meyer, Kofi Siriboe
#> 993        Viola Davis, Hayden Panettiere, Catalina Sandino Moreno, Ellen Burstyn
#> 994                       Lambert Wilson, Pierre Niney, Audrey Tautou,Laurent Lucas
#> 995                  Madina Nalwanga, David Oyelowo, Lupita Nyong'o, Martin Kabanza
#> 996                   Nia Vardalos, John Corbett, Michael Constantine, Lainie Kazan
#> 997                    Jason Biggs, Janet Montgomery,Ashley Tisdale, Bria L. Murphy
#> 998                    Nathalie Baye, Vincent Cassel, Marion Cotillard, Léa Seydoux
#> 999              Jennifer Garner, Kylie Rogers, Martin Henderson,Brighton Sharbino
#> 1000                       Kevin Spacey, Jennifer Garner, Robbie Amell,Cheryl Hines
#>      Year Runtime..Minutes. Rating Votes RevenueMillions
#> 991  2016                91    5.6  3990            0.03
#> 992  2016                80    6.1  2417            0.15
#> 993  2016               104    6.9   280              NA
#> 994  2016               122    6.7  1810              NA
#> 995  2016               124    7.4  6753            8.81
#> 996  2016                94    6.0 20966           59.57
#> 997  2016                92    5.0  2229              NA
#> 998  2016                97    7.0 10658              NA
#> 999  2016               109    7.0 12048           61.69
#> 1000 2016                87    5.3 12435           19.64
#>      Metascore
#> 991         48
#> 992         69
#> 993         72
#> 994         70
#> 995         73
#> 996         37
#> 997         38
#> 998         48
#> 999         44
#> 1000        11
```

5.12.5 Dropping NA values

The function `drop_na()` drops all rows containing NA values.

```
drop_na(movies) %>%
head(10)
#>    Rank                                 Title
#> 1     1               Guardians of the Galaxy
#> 2     2                            Prometheus
#> 3     3                                 Split
#> 4     4                                  Sing
#> 5     5                         Suicide Squad
#> 6     6                        The Great Wall
#> 7     7                            La La Land
#> 8     9                    The Lost City of Z
#> 9    10                            Passengers
#> 10   11 Fantastic Beasts and Where to Find Them
#>                        Genre
#> 1     Action,Adventure,Sci-Fi
#> 2    Adventure,Mystery,Sci-Fi
#> 3             Horror,Thriller
#> 4     Animation,Comedy,Family
#> 5    Action,Adventure,Fantasy
#> 6    Action,Adventure,Fantasy
#> 7          Comedy,Drama,Music
#> 8  Action,Adventure,Biography
#> 9     Adventure,Drama,Romance
#> 10   Adventure,Family,Fantasy
#>
#> 1
#> 2
#> 3                                                                     Three girls are kidnapped
#> 4  In a city of humanoid animals, a hustling theater impresario's attempt to save his theater
#> 5                                                A secret government agency recruits some of the
#> 6                                                                                European mercen
#> 7
#> 8                                                                            A true-life drama, cen
#> 9                                           A spacecraft traveling to a distant colony planet and
#> 10                                                                           The adventures of wri
#>              Director
#> 1          James Gunn
#> 2        Ridley Scott
#> 3    M. Night Shyamalan
#> 4  Christophe Lourdelet
#> 5          David Ayer
#> 6         Yimou Zhang
#> 7      Damien Chazelle
#> 8          James Gray
#> 9        Morten Tyldum
```

```
#> 10          David Yates
#>                                                                        Actors
#> 1                        Chris Pratt, Vin Diesel, Bradley Cooper, Zoe Saldana
#> 2     Noomi Rapace, Logan Marshall-Green, Michael Fassbender, Charlize Theron
#> 3            James McAvoy, Anya Taylor-Joy, Haley Lu Richardson, Jessica Sula
#> 4  Matthew McConaughey,Reese Witherspoon, Seth MacFarlane, Scarlett Johansson
#> 5                          Will Smith, Jared Leto, Margot Robbie, Viola Davis
#> 6                              Matt Damon, Tian Jing, Willem Dafoe, Andy Lau
#> 7                    Ryan Gosling, Emma Stone, Rosemarie DeWitt, J.K. Simmons
#> 8                Charlie Hunnam, Robert Pattinson, Sienna Miller, Tom Holland
#> 9            Jennifer Lawrence, Chris Pratt, Michael Sheen,Laurence Fishburne
#> 10               Eddie Redmayne, Katherine Waterston, Alison Sudol,Dan Fogler
#>    Year Runtime..Minutes. Rating  Votes RevenueMillions
#> 1  2014               121    8.1 757074          333.13
#> 2  2012               124    7.0 485820          126.46
#> 3  2016               117    7.3 157606          138.12
#> 4  2016               108    7.2  60545          270.32
#> 5  2016               123    6.2 393727          325.02
#> 6  2016               103    6.1  56036           45.13
#> 7  2016               128    8.3 258682          151.06
#> 8  2016               141    7.1   7188            8.01
#> 9  2016               116    7.0 192177          100.01
#> 10 2016               133    7.5 232072          234.02
#>    Metascore
#> 1         76
#> 2         65
#> 3         62
#> 4         59
#> 5         40
#> 6         42
#> 7         93
#> 8         78
#> 9         41
#> 10        66

drop_na(movies) %>%
nrow()
#> [1] 838
```

5.13 Outliers

5.13.1 What is an outlier?

Outliers also known as anomalies are values that deviate extremely from other values within the same group of data. They occur because of errors committed while collecting or recording data, performing calculations or are just data points with extreme values.

5.13.2 Identifying outlier

5.13.2.1 Using summary statistics

The first step in outlier detection is to look at summary statistics, most especially the minimum, maximum, median, and mean. For example, with a dataset of people's ages, if the maximum is 200 or the minimum is negative, then there is a problem.

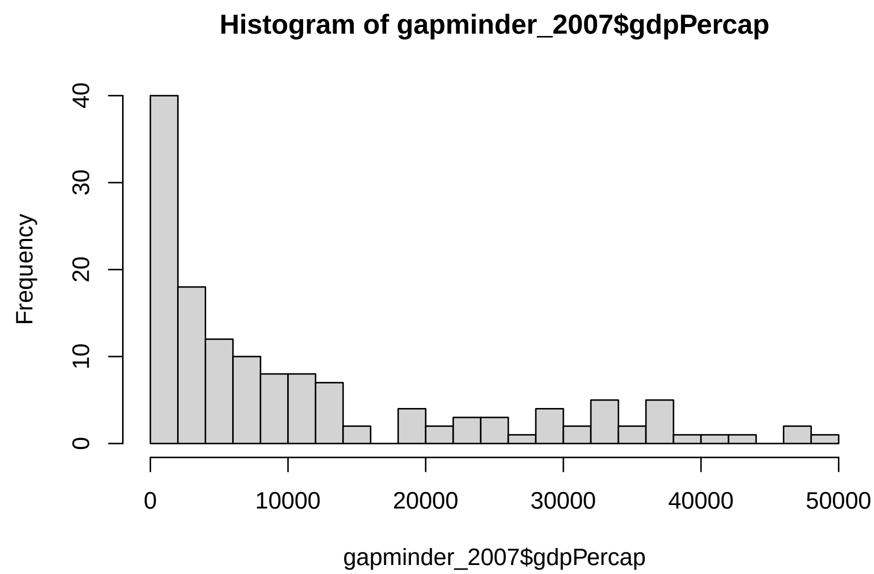
```
library(gapminder)
data(gapminder)
gapminder_2007 <- subset(gapminder, year == '2007', select = -year)
head(gapminder_2007)
#> # A tibble: 6 x 5
#>   country     continent lifeExp      pop gdpPercap
#>   <fct>       <fct>       <dbl>    <int>     <dbl>
#> 1 Afghanistan Asia         43.8 31889923      975.
#> 2 Albania     Europe       76.4  3600523     5937.
#> 3 Algeria     Africa       72.3 33333216     6223.
#> 4 Angola      Africa       42.7 12420476     4797.
#> 5 Argentina   Americas     75.3 40301927    12779.
#> 6 Australia   Oceania      81.2 20434176    34435.

summary(gapminder_2007$pop/1e6)
#>      Min.   1st Qu.    Median      Mean   3rd Qu.      Max.
#>    0.1996    4.5080   10.5175   44.0212   31.2100 1318.6831
```

From the above, we see that the median and mean are 10 million and 44 million respectively while the maximum value is 1.3 billion. This tells us that there are some outliers since the maximum value varies greatly from the centre of the data.

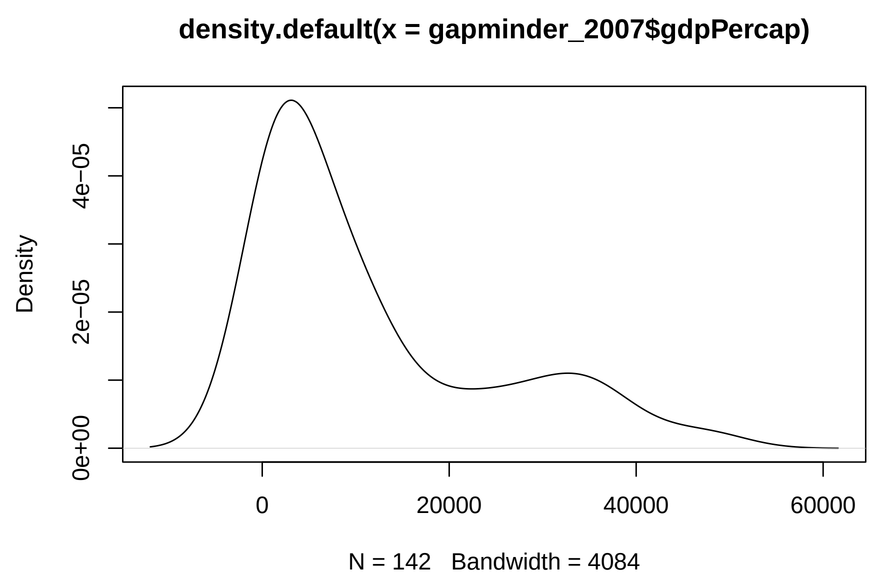
5.13.3 Using plots

Outliers are identified using univariate plots such as histogram, density plot and boxplot.

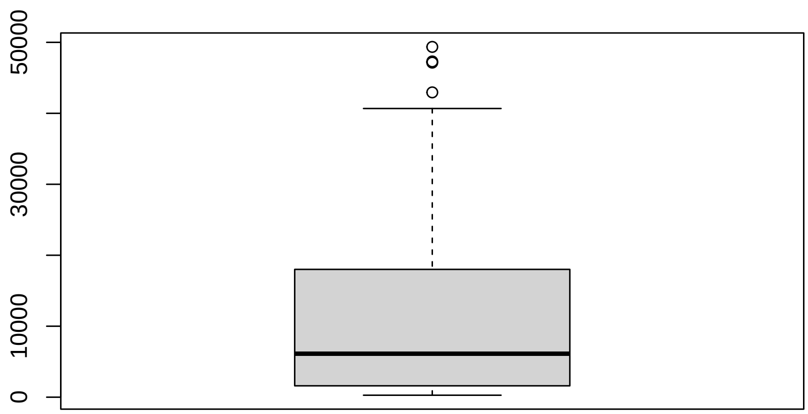
```
# plotting variable using histogram
hist(gapminder_2007$gdpPercap, breaks = 18)
```




```
# density plot
plot(density(gapminder_2007$gdpPercap))
```



```
# boxplot of population
boxplot(gapminder_2007$gdpPercap)
```



Of the above data visualizations, the boxplot is the most relevant as it shows both the spread of data and outliers. The boxplot reveals the following:

- minimum value,
- first quantile (Q1),
- median (second quantile),
- third quantile (Q3),
- maximum value excluding outliers and
- outliers.

The difference between Q3 and Q1 is known as the Interquartile Range (IQR).

The outliers within the box plot are calculated as any value that falls beyond 1.5 * IQR.

The function `boxplot.stats()` computes the data that is used to draw the box plot. Using this function, we can get our outliers.

```
boxplot.stats(gapminder_2007$gdpPercap)
#> $stats
#> [1]   277.5519  1598.4351  6124.3711 18008.9444 40675.9964
#>
#> $n
```

```
#> [1] 142
#>
#> $conf
#> [1] 3948.491 8300.251
#>
#> $out
#> [1] 47306.99 49357.19 47143.18 42951.65
```

The first element returned is the summary statistic as was calculated with `summary()`.

```
boxplot.stats(gapminder_2007$gdpPercap)$stats
#> [1]   277.5519  1598.4351  6124.3711 18008.9444 40675.9964
summary(gapminder_2007$gdpPercap)
#>    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
#>   277.6  1624.8  6124.4 11680.1 18008.8 49357.2
```

The last element returned are the outliers.

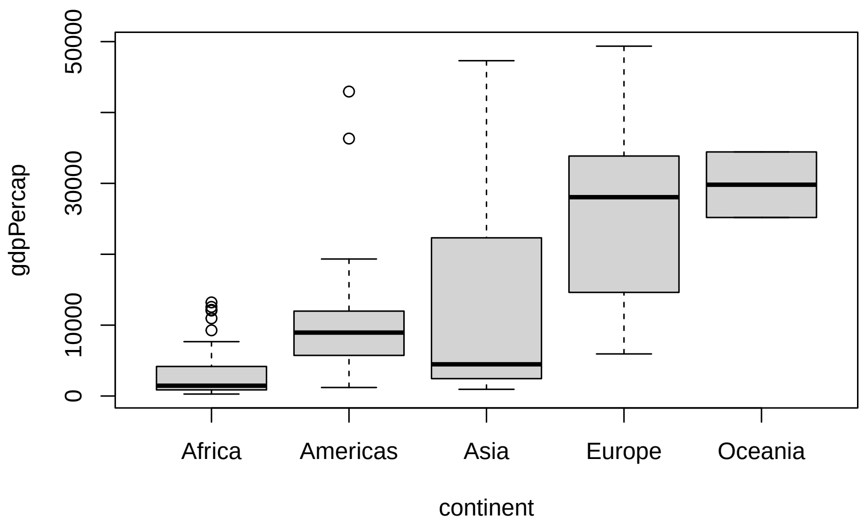
```
boxplot.stats(gapminder_2007$gdpPercap)$out
#> [1] 47306.99 49357.19 47143.18 42951.65
```

Recall outliers are calculated as 1.5 * IQR, this can be changed using the argument coef. By default, it is set to 1.5 but can be changed as need be.

```
# changing coef
boxplot.stats(gapminder_2007$gdpPercap, coef = 0.8)$out
#>  [1] 34435.37 36126.49 33692.61 36319.24 35278.42 33207.08
#>  [7] 32170.37 39724.98 36180.79 40676.00 31656.07 47306.99
#> [13] 36797.93 49357.19 47143.18 33859.75 37506.42 33203.26
#> [19] 42951.65
boxplot.stats(gapminder_2007$gdpPercap, coef = 1)$out
#>  [1] 34435.37 36126.49 36319.24 35278.42 39724.98 36180.79
#>  [7] 40676.00 47306.99 36797.93 49357.19 47143.18 37506.42
#> [13] 42951.65
boxplot.stats(gapminder_2007$gdpPercap, coef = 1.2)$out
#> [1] 39724.98 40676.00 47306.99 49357.19 47143.18 42951.65

# selecting outliers
subset(gapminder_2007, gdpPercap >= min(boxplot.stats(gdpPercap)$out))
#> # A tibble: 4 x 5
#>   country       continent lifeExp       pop gdpPercap
#>   <fct>         <fct>       <dbl>     <int>     <dbl>
#> 1 Kuwait        Asia         77.6   2505559    47307.
#> 2 Norway        Europe       80.2   4627926    49357.
#> 3 Singapore     Asia         80.0   4553009    47143.
#> 4 United States Americas     78.2 301139947    42952.
```

5.13.4 Outliers by groups

```r
# boxplot by continent
boxplot(gdpPercap ~ continent, gapminder_2007)
```




```r
# splitting data frame
gap_split <- split(gapminder_2007, gapminder_2007$continent)

outliers_2007 <-
lapply(gap_split, function(x) {
    x <- boxplot.stats(x$gdpPercap)$out
    return(x)
})
outliers_2007
#> $Africa
#> [1] 12569.852 12154.090 13206.485 12057.499 10956.991
#> [6]  9269.658
#>
#> $Americas
#> [1] 36319.24 42951.65
#>
#> $Asia
#> numeric(0)
#>
```

```
#> $Europe
#> numeric(0)
#>
#> $Oceania
#> numeric(0)
```

5.14 String manipulation with stringr

5.14.1 Determine string length

The function `str_length()` returns the count of letters in a string.

```
library(stringr)
month.name
#>  [1] "January"   "February"  "March"     "April"
#>  [5] "May"       "June"      "July"      "August"
#>  [9] "September" "October"   "November"  "December"
str_length(month.name)
#>  [1] 7 8 5 5 3 4 4 6 9 7 8 8
```

5.14.2 Strings formatting (case conversion)

The functions `str_to_upper()`, `str_to_lower()`, `str_to_title()` and `str_to_sentence()` are used to convert to upper, lower, title and sentence cases respectively.

The function `str_pad()` is used to pad characters before and/or after a string. The function `str_trunc()` is used to truncate a string.

```
# lowercase
str_to_lower('It is an everyday thing', locale = "en")
#> [1] "it is an everyday thing"

# uppercase
str_to_upper('It is an everyday thing', locale = "en")
#> [1] "IT IS AN EVERYDAY THING"

# title case
str_to_title('It is an everyday thing', locale = "en")
#> [1] "It Is An Everyday Thing"

# sentence case
str_to_sentence('iT is aN everyDay thIng', locale = "en")
#> [1] "It is an everyday thing"

# padding string
```

```
str_pad(c(12, 235, 'abd', 'ame'), width = 5, pad = '0')
#> [1] "00012" "00235" "00abd" "00ame"
str_pad(c(12, 235, 'abd', 'ame'), width = 5, pad = 'X', side = 'right')
#> [1] "12XXX" "235XX" "abdXX" "ameXX"
str_pad(c(12, 235, 'abd', 'ame'), width = 5, pad = '-', side = 'both')
#> [1] "-12--" "-235-" "-abd-" "-ame-"

# truncate a character string
str_trunc(state.name[1:8], width = 6)
#> [1] "Ala..." "Alaska" "Ari..." "Ark..." "Cal..." "Col..."
#> [7] "Con..." "Del..."
str_trunc(state.name[1:8], 6, side  = 'left')
#> [1] "...ama" "Alaska" "...ona" "...sas" "...nia" "...ado"
#> [7] "...cut" "...are"
str_trunc(state.name[1:8], 6, side  = 'right', ellipsis = '')
#> [1] "Alabam" "Alaska" "Arizon" "Arkans" "Califo" "Colora"
#> [7] "Connec" "Delawa"
```

5.14.3 Join and Split strings

5.14.3.1 joining strings with str_c()

The function `str_c()` joins two or more vectors element wise into a single character vector, optionally inserting separator (sep) between input vectors.

```
# combining elements into a character vector
str_c('a', 'b')
#> [1] "ab"
str_c(1, 2, 3, 4)
#> [1] "1234"

# using sep
str_c('a', 'b', sep = ' ')
#> [1] "a b"
str_c(1, 2, 3, 4, sep = ' ')
#> [1] "1 2 3 4"
str_c(1:10, sep = ' ')
#>  [1] "1"  "2"  "3"  "4"  "5"  "6"  "7"  "8"  "9"  "10"

# on a single vector
str_c(c('a', 'b'), sep = ' <> ')
#> [1] "a" "b"
str_c(c(1, 2), sep = ' <> ')
#> [1] "1" "2"

# two or more vectors
```

```
str_c(c('a', 'b'), c('c', 'd'), sep = ' <> ')
#> [1] "a <> c" "b <> d"
str_c(1:5, 10:20, sep = ' ')
#>  [1] "1 10" "2 11" "3 12" "4 13" "5 14" "1 15" "2 16" "3 17"
#>  [9] "4 18" "5 19" "1 20"
str_c(1:5, 10:20, c('a','b','c'), sep = ' ')
#>  [1] "1 10 a" "2 11 b" "3 12 c" "4 13 a" "5 14 b" "1 15 c"
#>  [7] "2 16 a" "3 17 b" "4 18 c" "5 19 a" "1 20 b"
# collapsing vectors
str_c(1:10, collapse = '~')
#> [1] "1~2~3~4~5~6~7~8~9~10"
str_c(c('a', 'b'), c('c', 'd'), collapse = ' <> ')
#> [1] "ac <> bd"
str_c(month.name[1:6], collapse = " - ")
#> [1] "January - February - March - April - May - June"

a <- month.name[1]
b <- month.name[2]
c <- month.name[3]

# combining character and variables
str_c(b,'comes after', a ,'but comes before', c, sep = " ")
#> [1] "February comes after January but comes before March"
str_c(b,'comes after', a ,'but comes before', c, sep = "/")
#> [1] "February/comes after/January/but comes before/March"
str_c('version 1.', 1:5, sep = '')
#> [1] "version 1.1" "version 1.2" "version 1.3" "version 1.4"
#> [5] "version 1.5"
```

5.14.4 Joining using str_glue()

The function `str_glue()` returns a character vector containing a formatted combination of text and variable values.

formatting with integers

```
x <- 2
str_glue('{x} * {x} = {x ** 2}')
#> 2 * 2 = 4

x <- c(1:4)
str_glue('{x} squared is equal to {x ** 2}')
#> 1 squared is equal to 1
#> 2 squared is equal to 4
#> 3 squared is equal to 9
#> 4 squared is equal to 16
```

```r
num <- c(123, 1, 100, 200, 10200, 25000)
str_glue('my registration number is {str_pad(num, 5, pad = "0")}')
#> my registration number is 00123
#> my registration number is 00001
#> my registration number is 00100
#> my registration number is 00200
#> my registration number is 10200
#> my registration number is 25000
```

Formatting with strings

```r
x <- 'my name is'
y <- 'james'
z <- 'london'
str_glue('{x} {y} and i live and work in {z}')
#> my name is james and i live and work in london


x <- 'my name is'
y <- 'james'
z <- 35
str_glue('{str_to_title(x)} {str_to_upper(y)} and i am {z} years')
#> My Name Is JAMES and i am 35 years

names <- c('paul', 'alphonse', 'michael', 'james', 'samson', 'terence', 'derin')
age <- c(30, 35, 32, 37, 29, 40, 30)
str_glue('i am {str_to_title(names)} and i am {age} years old')
#> i am Paul and i am 30 years old
#> i am Alphonse and i am 35 years old
#> i am Michael and i am 32 years old
#> i am James and i am 37 years old
#> i am Samson and i am 29 years old
#> i am Terence and i am 40 years old
#> i am Derin and i am 30 years old
```

Formatting with doubles or floating points

```r
x <- 1000/6
x
#> [1] 166.6667
str_glue('1000 divided by 3 is {x}')
#> 1000 divided by 3 is 166.666666666667
str_glue('1000 divided by 3 is {round(x, 3)}')
#> 1000 divided by 3 is 166.667
str_glue('1000 divided by 3 is {round(x)}')
#> 1000 divided by 3 is 167
```

```
str_glue('1000 divided by 3 is {paste0("+", round(x))}')
#> 1000 divided by 3 is +167
str_glue('1000 divided by 3 is{paste0(" ", round(x))}')
#> 1000 divided by 3 is 167
```

5.14.5 Splitting strings using str_split() and str_split_fixed()

The function `str_split()` splits the elements of a character vector into substrings by a specific pattern. The function `str_split_fixed()` splits up the elements of a character into a fixed number of pieces.

```
str(str_split(c('2020-01-01', '2019-03-31', '2018-06-30'), pattern = "-"))
#> List of 3
#>  $ : chr [1:3] "2020" "01" "01"
#>  $ : chr [1:3] "2019" "03" "31"
#>  $ : chr [1:3] "2018" "06" "30"
str(str_split(c('2020 01 01', '2019 03 31', '2018 06 30'), pattern = " "))
#> List of 3
#>  $ : chr [1:3] "2020" "01" "01"
#>  $ : chr [1:3] "2019" "03" "31"
#>  $ : chr [1:3] "2018" "06" "30"

# splitting into two substrings
str(str_split(c('2020-01-01', '2019-03-31', '2018-06-30'), pattern = "-", n = 2))
#> List of 3
#>  $ : chr [1:2] "2020" "01-01"
#>  $ : chr [1:2] "2019" "03-31"
#>  $ : chr [1:2] "2018" "06-30"
str(str_split(c('2020 01 01', '2019 03 31', '2018 06 30'), pattern = " ", n = 2))
#> List of 3
#>  $ : chr [1:2] "2020" "01 01"
#>  $ : chr [1:2] "2019" "03 31"
#>  $ : chr [1:2] "2018" "06 30"

# returning a matrix
str_split_fixed(c('2020-01-01', '2019-03-31', '2018-06-30'), '-', 2)
#>      [,1]   [,2]
#> [1,] "2020" "01-01"
#> [2,] "2019" "03-31"
#> [3,] "2018" "06-30"
str_split_fixed(c('2020-01-01', '2019-03-31', '2018-06-30'), '-', 3)
#>      [,1]   [,2] [,3]
#> [1,] "2020" "01" "01"
#> [2,] "2019" "03" "31"
#> [3,] "2018" "06" "30"
```

5.14.6 Extract and Replace part of a string

5.14.6.1 Extracting string values using str_sub()

The function `str_sub()` extracts a substring from a string by indexing. It uses start for the beginning position and stop for the ending position. It is like indexing but applied to a string.

```
var <- c('2020-01-01', '2019-03-31', '2018-06-30')
str_sub(var, start = 1, end = 4)
#> [1] "2020" "2019" "2018"
str_sub(var, 6, 7)
#> [1] "01" "03" "06"
str_sub(var, 9, 10)
#> [1] "01" "31" "30"

# using negative numbers
str_sub(var, -2, -1)
#> [1] "01" "31" "30"
str_sub(var, -5, -4)
#> [1] "01" "03" "06"
str_sub(var, -10, -7)
#> [1] "2020" "2019" "2018"
```

5.14.6.2 Replacing string values using str_sub()

The function `str_sub()` is also used to replace substring in a string by assigning a different string to the extracted substring.

```
var <- c('2020-01-01', '2019-03-31', '2018-06-30')
str_sub(var, 1, 4) <- c('2010', '2011', '2012')
var
#> [1] "2010-01-01" "2011-03-31" "2012-06-30"
```

5.14.7 Replacing string values using str_replace()

The function `str_replace()` replaces a substring at first occurrence.

```
var <- c('2020-01-01', '2019-03-31', '2018-06-30')
str_replace(var, "-", "")
#> [1] "202001-01" "201903-31" "201806-30"
str_replace(var, "-", "/")
#> [1] "2020/01-01" "2019/03-31" "2018/06-30"
```

5.14.8 Replacing string values using str_replace_all()

The function `str_replace_all()` replaces a substring throughout a string.

```
var <- c('2020-01-01', '2019-03-31', '2018-06-30')
str_replace_all(var, "-", " ")
#> [1] "2020 01 01" "2019 03 31" "2018 06 30"
str_replace_all(var, "-", "/")
#> [1] "2020/01/01" "2019/03/31" "2018/06/30"
```

5.14.8.1 Remove white spaces and clean string values

The function:

- `str_trim()` removes white spaces.
- `str_squish()` removes repeated spaces.
- `str_remove()` removes the first repeated spaces.
- `str_remove_all()` removes all repeated spaces.

```
# both sides
str_trim(c(' 2020-01-01 ', ' 2019-03-31 ', ' 2018-06-30 '))
#> [1] "2020-01-01" "2019-03-31" "2018-06-30"

# left side
str_trim(c(' 2020-01-01 ', ' 2019-03-31 ', ' 2018-06-30 '), side = 'left')
#> [1] "2020-01-01 " "2019-03-31 " "2018-06-30 "

# right side
str_trim(c(' 2020-01-01 ', ' 2019-03-31 ', ' 2018-06-30 '), side = 'right')
#> [1] " 2020-01-01" " 2019-03-31" " 2018-06-30"

str_squish('removing   all    repeated   spaces in a string  ')
#> [1] "removing all repeated spaces in a string"

str_remove('removing   first  repeated   spaces in a string  ', '  ')
#> [1] "removing first  repeated   spaces in a string  "
str_remove_all('removing   all  repeated   spaces in a string  ', '  ')
#> [1] "removing allrepeated spaces in a string"
```

5.14.9 Sorting

The function:

- `str_order()` sorts a character vector and returns sorted indices.
- `str_sort()` sorts a character vector and returns sorted values.

```
str_order(month.name)
#>  [1]  4  8 12  2  1  7  6  3  5 11 10  9
str_order(month.name, decreasing = T)
#>  [1]  9 10 11  5  3  6  7  1  2 12  8  4
```

```
str_sort(month.name)
#>  [1] "April"     "August"    "December"  "February"
#>  [5] "January"   "July"      "June"      "March"
#>  [9] "May"       "November"  "October"   "September"
str_sort(month.name, decreasing = T)
#>  [1] "September" "October"   "November"  "May"
#>  [5] "March"     "June"      "July"      "January"
#>  [9] "February"  "December"  "August"    "April"
```

5.14.10 Duplicating strings

The function `str_dup()` duplicates and concatenate strings within a character vector.

```
str_dup('jan', 2)
#> [1] "janjan"
str_dup('jan', 1:3)
#> [1] "jan"       "janjan"    "janjanjan"
```

5.14.11 Pattern matching using regular expression

5.14.11.1 Regex functions

- `str_which()`,str_detect()`and`str_subset()‘
- `str_count()`
- `str_starts()` and `str_ends()`
- `str_locate()` and `str_locate_all()`
- `str_extract()` and `str_extract_all()`
- `str_match()` and `str_match_all()`
- `str_view()` and `str_view_all()`
- `str_replace()` and `str_replace_all()`

5.14.11.1.1 The functions str_detect(), str_which() and str_subset() The function:

- `str_detect()` detects the presence or absence of a pattern in a string and is equivalent to grepl(pattern, x).
- `str_which()` detects the position of a matched pattern and is equivalent to grep(pattern, x).
- `str_subset()` keeps string matching a pattern and is equivalent to grep(pattern, x, value = TRUE).

```
str_detect(month.name, 'uary')
#>  [1]  TRUE  TRUE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
#> [10] FALSE FALSE FALSE
month.name[str_detect(month.name, 'uary')]
#> [1] "January"  "February"
```

```
str_detect(month.name, 'uary', negate = T)
#>  [1] FALSE FALSE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE
#> [10]  TRUE  TRUE  TRUE
month.name[str_detect(month.name, 'uary', negate = T)]
#>  [1] "March"     "April"     "May"       "June"
#>  [5] "July"      "August"    "September" "October"
#>  [9] "November"  "December"
str_which(month.name, 'uary')
#> [1] 1 2
month.name[str_which(month.name, 'uary')]
#> [1] "January"  "February"

str_which(month.name, 'uary', negate = T)
#>  [1]  3  4  5  6  7  8  9 10 11 12
month.name[str_which(month.name, 'uary', negate = T)]
#>  [1] "March"     "April"     "May"       "June"
#>  [5] "July"      "August"    "September" "October"
#>  [9] "November"  "December"

str_subset(month.name, pattern = 'ber')
#> [1] "September" "October"   "November"  "December"
str_subset(month.name, pattern = 'ber', negate = TRUE)
#> [1] "January"  "February" "March"    "April"    "May"
#> [6] "June"     "July"     "August"
```

5.14.11.1.2 The function str_count() The function `str_count()` counts the number of matches in a string.

```
var <- c('2020-01-01', '2019-03-31', '2018-06-30')
str_count(var, pattern = '-')
#> [1] 2 2 2
```

5.14.11.2 The functions str_starts() and str_ends()

The function:

- `str_starts()` detects the presence of a pattern at the beginning of a string.
- `str_ends()` detects the presence of a pattern at the end of a string.

```
str_starts(month.name, 'J')
#>  [1]  TRUE FALSE FALSE FALSE FALSE  TRUE  TRUE FALSE FALSE
#> [10] FALSE FALSE FALSE
month.name[str_starts(month.name, 'J')]
#> [1] "January" "June"    "July"
str_ends(month.name, 'ber')
```

```
#>  [1] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE  TRUE
#> [10]  TRUE  TRUE  TRUE
month.name[str_ends(month.name, 'ber')]
#> [1] "September" "October"   "November"  "December"
```

5.14.11.2.1 The functions str_locate() and str_locate_all()

The function:

- `str_locate()` locates the position of the first pattern match in a string.
- `str_locate_all()` locates the position of all pattern matches in a string.

```
str_locate(month.name, 'ber')
#>       start end
#>  [1,]    NA  NA
#>  [2,]    NA  NA
#>  [3,]    NA  NA
#>  [4,]    NA  NA
#>  [5,]    NA  NA
#>  [6,]    NA  NA
#>  [7,]    NA  NA
#>  [8,]    NA  NA
#>  [9,]     7   9
#> [10,]     5   7
#> [11,]     6   8
#> [12,]     6   8
```

5.14.11.2.2 The functions str_extract() and str_extract_all()

The function:

- `str_extract()` extracts the first matching pattern from a string.
- `str_extract_all()` extracts all matching patterns from a string.

```
str_extract(string = month.name, pattern = 'ber')
#>  [1] NA    NA    NA    NA    NA    NA    NA    NA    "ber"
#> [10] "ber" "ber" "ber"
```

5.14.11.2.3 The functions str_view() and str_view_all()

The functions `str_view()` and `str_view_all()` Views HTML rendering of regular expression match, with the first matching the first occurrence and the later all occurrences.

```
str_view(month.name, 'uary')
```

```
January
February
March
April
May
June
July
August
September
October
November
December
```

5.14.11.3 Regex Operations

matching spaces

```
var <- c('2020 01 01', '2019 03 31', '2018 06 30')
str_replace_all(var, '[[:space:]]', '-')
#> [1] "2020-01-01" "2019-03-31" "2018-06-30"
str_replace_all(var, '\\s', '-')
#> [1] "2020-01-01" "2019-03-31" "2018-06-30"
str(str_split(var, '\\s'))
#> List of 3
#>  $ : chr [1:3] "2020" "01" "01"
#>  $ : chr [1:3] "2019" "03" "31"
#>  $ : chr [1:3] "2018" "06" "30"
str_replace_all(var, '\\S', '-')
#> [1] "---- -- --" "---- -- --" "---- -- --"
```

matching alphabetic characters

```
var <- 'a1b2c3d4e5f'
str_replace_all(var, '[[:alpha:]]', '')
#> [1] "12345"
# lowercase letters
str_replace_all(month.name, '[[:lower:]]', '')
#>  [1] "J" "F" "M" "A" "M" "J" "J" "A" "S" "O" "N" "D"
```

matching numerical digits

```
var <- 'a1b2c3d4e5f'
str_replace_all(var, '[[:digit:]]', '')
#> [1] "abcdef"
str_replace_all(var, '\\d', '')
#> [1] "abcdef"
```

matching letters and numbers (alphanumeric characters)

```
var <- 'a1@; 2#4c $8`*%f^!1~0&^h*()j'
str_replace_all(var, '[[:alnum:]]', '')
```

```
#> [1] "@; # $`*%^!~&^*()"

str_replace_all(var, '[[:xdigit:]]', '')
#> [1] "@; # $`*%^!~&^h*()j"

str_replace_all(var, '\\w', '')
#> [1] "@; # $`*%^!~&^*()"
```

matching punctuation

```
var <- 'a1@; 2#4c $8`*%f^!1~0&^h*()j'
str_replace_all(var, '[[:punct:]]', '')
#> [1] "a1 24c $8`f^1~0^hj"

str_replace_all(var, '\\W', '')
#> [1] "a124c8f10hj"
```

matching letters, numbers, and punctuation

```
var <- 'a1@; 2#4c $8`*%f^!1~0&^h*()j'
str_replace_all(var, '[[:graph:]]', ' ')
#> [1] "                            "

str_replace_all(var, '.', ' ')
#> [1] "                            "
```

matching whitespace

```
str_replace_all(c(' 2020-01-01 ', ' 2019-03-31 ', ' 2018-06-30 '), '\\s', '')
#> [1] "2020-01-01" "2019-03-31" "2018-06-30"
```

matching newline and tap

```
cat('good morning \n i am fru kinglsy \n i will be your instructor')
#> good morning
#>  i am fru kinglsy
#>  i will be your instructor

# replacing new line
str_replace_all('good morning \n i am fru kinglsy \n i will be your instructor', '\\n', '\t')
#> [1] "good morning \t i am fru kinglsy \t i will be your instructor"
cat(str_replace_all('good morning \n i am fru kinglsy \n i will be your instructor', '\\n', '\t')
#> good morning      i am fru kinglsy    i will be your instructor

# replacing tab
str_replace_all('good morning \t i am fru kinglsy \t i will be your instructor', '\\t', '\n')
#> [1] "good morning \n i am fru kinglsy \n i will be your instructor"
cat(str_replace_all('good morning \t i am fru kinglsy \t i will be your instructor', '\\t', '\n')
```

```
#> good morning
#>  i am fru kinglsy
#>  i will be your instructor
```

matching metacharacters

```
sales <- c('$25000', '$20000', '$22500', '$24000', '$30000', '$35000')
str_replace(sales, '\\$', '')
#> [1] "25000" "20000" "22500" "24000" "30000" "35000"

sales <- c('+25000', '+20000', '+22500', '+24000', '+30000', '+35000')
str_replace(sales, '\\+', '')
#> [1] "25000" "20000" "22500" "24000" "30000" "35000"

dates <- c('01.01.2012', '01.02.2012', '01.03.2012', '01.04.2012', '01.05.2012', '01.0
str_replace_all(dates, '\\.', '-')
#> [1] "01-01-2012" "01-02-2012" "01-03-2012" "01-04-2012"
#> [5] "01-05-2012" "01-06-2012"

dates <- c('01*01*2012', '01*02*2012', '01*03*2012', '01*04*2012', '01*05*2012', '01*0
str_replace_all(dates, '\\*', '-')
#> [1] "01-01-2012" "01-02-2012" "01-03-2012" "01-04-2012"
#> [5] "01-05-2012" "01-06-2012"

dates <- c('01^01^2012', '01^02^2012', '01^03^2012', '01^04^2012', '01^05^2012', '01^0
str_replace_all(dates, '\\^', '-')
#> [1] "01-01-2012" "01-02-2012" "01-03-2012" "01-04-2012"
#> [5] "01-05-2012" "01-06-2012"

dates <- c('01|01|2012', '01|02|2012', '01|03|2012', '01|04|2012', '01|05|2012', '01|0
str_replace_all(dates, '\\|', '-')
#> [1] "01-01-2012" "01-02-2012" "01-03-2012" "01-04-2012"
#> [5] "01-05-2012" "01-06-2012"

dates <- c('01\\01\\2012', '01\\02\\2012', '01\\03\\2012', '01\\04\\2012', '01\\05\\20
str_replace_all(dates, '\\\\', '-')
#> [1] "01-01-2012" "01-02-2012" "01-03-2012" "01-04-2012"
#> [5] "01-05-2012" "01-06-2012"

dates <- c('01\\.01\\.2012', '01\\.02\\.2012', '01\\.03\\.2012', '01\\.04\\.2012', '01\
str_replace_all(dates, '\\\\\\.', '-')
#> [1] "01-01-2012" "01-02-2012" "01-03-2012" "01-04-2012"
#> [5] "01-05-2012" "01-06-2012"
```

alternates and ranges

```
# either or
str_view_all(month.name, 'uary|ember|ober', '*')
```

```
January
February
March
April
May
June
July
August
September
October
November
December
```

```
# ranges
str_replace_all(month.name, '[aeiou]', '*')
#>  [1] "J*n**ry"   "F*br**ry"  "M*rch"     "Apr*l"
#>  [5] "M*y"       "J*n*"      "J*ly"      "A*g*st"
#>  [9] "S*pt*mb*r" "Oct*b*r"   "N*v*mb*r"  "D*c*mb*r"
str_replace_all(month.name, '[a-z]', '*')
#>  [1] "J******"   "F*******"  "M****"     "A****"
#>  [5] "M**"       "J***"      "J***"      "A*****"
#>  [9] "S********" "O******"   "N*******"  "D*******"
str_replace_all(month.name, '[A-Z]', '*')
#>  [1] "*anuary"   "*ebruary"  "*arch"     "*pril"
#>  [5] "*ay"       "*une"      "*uly"      "*ugust"
#>  [9] "*eptember" "*ctober"   "*ovember"  "*ecember"
str_replace_all(month.name, '[m-z]', '*')
#>  [1] "Ja**a**"   "Feb**a**"  "Ma*ch"     "A**il"
#>  [5] "Ma*"       "J**e"      "J*l*"      "A*g***"
#>  [9] "Se**e*be*" "Oc**be*"   "N**e*be*"  "Dece*be*"
str_replace_all(month.name, '[0-9]', '*')
#>  [1] "January"   "February"  "March"     "April"
#>  [5] "May"       "June"      "July"      "August"
#>  [9] "September" "October"   "November"  "December"
str_replace_all(month.name, '[1-5]', '*')
#>  [1] "January"   "February"  "March"     "April"
#>  [5] "May"       "June"      "July"      "August"
#>  [9] "September" "October"   "November"  "December"
str_replace_all(month.name, '[a-zA-Z0-9]', '*')
#>  [1] "*******"   "********"  "*****"     "*****"
#>  [5] "***"       "****"      "****"      "******"
#>  [9] "*********" "*******"   "********"  "********"

# anything but
str_replace_all(month.name, '[^aeiou]', '*')
```

```
#>  [1] "*a*ua**"   "*e**ua**"  "*a***"     "***i*"
#>  [5] "*a*"       "*u*e"      "*u**"      "*u*u**"
#>  [9] "*e***e*" "***o*e*"   "*o*e**e*"  "*e*e**e*"
str_replace_all(month.name, '[^a-z]', '*')
#>  [1] "*anuary"   "*ebruary"  "*arch"     "*pril"
#>  [5] "*ay"       "*une"      "*uly"      "*ugust"
#>  [9] "*eptember" "*ctober"   "*ovember"  "*ecember"
```

groups

```
str_subset(pattern = '(s{2})e', state.name)
#> [1] "Tennessee"
```

anchors

```
# start of a string
str_replace_all(month.name, '^J', 'j')
#>  [1] "january"   "February"  "March"     "April"
#>  [5] "May"       "june"      "july"      "August"
#>  [9] "September" "October"   "November"  "December"

# end of a string
str_replace_all(month.name, 'ber$', 'ba')
#>  [1] "January"  "February" "March"    "April"    "May"
#>  [6] "June"     "July"     "August"   "Septemba" "Octoba"
#> [11] "Novemba"  "Decemba"
```

quantifiers

```
# match 's' zero or one time
str_subset(month.name, 's?')
#>  [1] "January"   "February"  "March"     "April"
#>  [5] "May"       "June"      "July"      "August"
#>  [9] "September" "October"   "November"  "December"
# match 'J' one or more times
str_subset(month.name, 'J+')
#> [1] "January" "June"    "July"
# match 'e' one or more times
str_subset(state.name, 'e+')
#>  [1] "Connecticut"   "Delaware"      "Georgia"
#>  [4] "Kentucky"      "Maine"         "Massachusetts"
#>  [7] "Minnesota"     "Nebraska"      "Nevada"
#> [10] "New Hampshire" "New Jersey"    "New Mexico"
#> [13] "New York"      "Oregon"        "Pennsylvania"
#> [16] "Rhode Island"  "Tennessee"     "Texas"
#> [19] "Vermont"       "West Virginia"
# matched 'y', zero or more times
```

```
str_subset(month.name, 'y*')
#>  [1] "January"   "February"  "March"     "April"
#>  [5] "May"       "June"      "July"      "August"
#>  [9] "September" "October"   "November"  "December"
# matched 'a', zero or more times
str_subset(month.name, 'a*')
#>  [1] "January"   "February"  "March"     "April"
#>  [5] "May"       "June"      "July"      "August"
#>  [9] "September" "October"   "November"  "December"
# match 'a' zero or more times and 'y'
str_subset(month.name, 'a*y')
#> [1] "January"  "February" "May"      "July"
# match 'y' zero or more times and 'a'
str_subset(month.name, 'y*a')
#> [1] "January"  "February" "March"    "May"
# match 's', exactly 2 times
str_subset(state.name, "s{2}")
#> [1] "Massachusetts" "Mississippi"   "Missouri"
#> [4] "Tennessee"
# match 's', exactly 1 or more times
str_subset(state.name, "s{1,}")
#>  [1] "Alaska"        "Arkansas"      "Illinois"
#>  [4] "Kansas"        "Louisiana"     "Massachusetts"
#>  [7] "Minnesota"     "Mississippi"   "Missouri"
#> [10] "Nebraska"      "New Hampshire" "New Jersey"
#> [13] "Pennsylvania"  "Rhode Island"  "Tennessee"
#> [16] "Texas"         "Washington"    "West Virginia"
#> [19] "Wisconsin"
# match 's', exactly 1 or 2 times
str_subset(state.name, "s{1,2}")
#>  [1] "Alaska"        "Arkansas"      "Illinois"
#>  [4] "Kansas"        "Louisiana"     "Massachusetts"
#>  [7] "Minnesota"     "Mississippi"   "Missouri"
#> [10] "Nebraska"      "New Hampshire" "New Jersey"
#> [13] "Pennsylvania"  "Rhode Island"  "Tennessee"
#> [16] "Texas"         "Washington"    "West Virginia"
#> [19] "Wisconsin"
```

Chapter 6

Modern graphics

6.1 Overview

ggplot2 was created by Hadley Wickham back in 2005 as an implementation of Leland Wilkinson's grammar of graphics. The general idea behind the grammar of graphics is that a plot can be broken down into different elements and assembled by adding elements together. This reasoning is the foundation of the popular data visualization package ggplot2.

ggplot2 is built on the premise that graphically data can be represented as either:

- Points e.g. in the case of scatter plots
- Lines e.g. in the case of line plots
- Bars e.g. in the case of histograms and bar plots
- Or a combination of some or all of them e.g. dot plot

These are collectively known as geometric objects. These geometric objects can have different attributes (colours, shape, and size). These attributes can either be mapped or set during plotting.

Mapping simply means colour, shape and size are added in such a manner that they are linked to the underlying data represented by the geometric objects. In so doing they add more information and understanding to the plot and most often changes if the underlying data changes.

While setting, on the other hand, is not linked to the underlying data but rather adds more beauty than information. Because they add little or no information, setting should be done with care most especially when using size and shape.

ggplot2 consist of seven layers which are:

- data: holds data to be plotted

- geom: determines the type of plot, that is the type of geometric object to be used e.g. geom_point(), geom_line(), geom_bar(), etc.
- aesthetics: maps data and attributes (colour, shape, and size) to the geom
- stat: performs a statistical transformation
- position adjustment: determines where elements are positioned on the plot relative to others
- coordinate-system: manipulates the coordinate system
- faceting: used for creating subplots

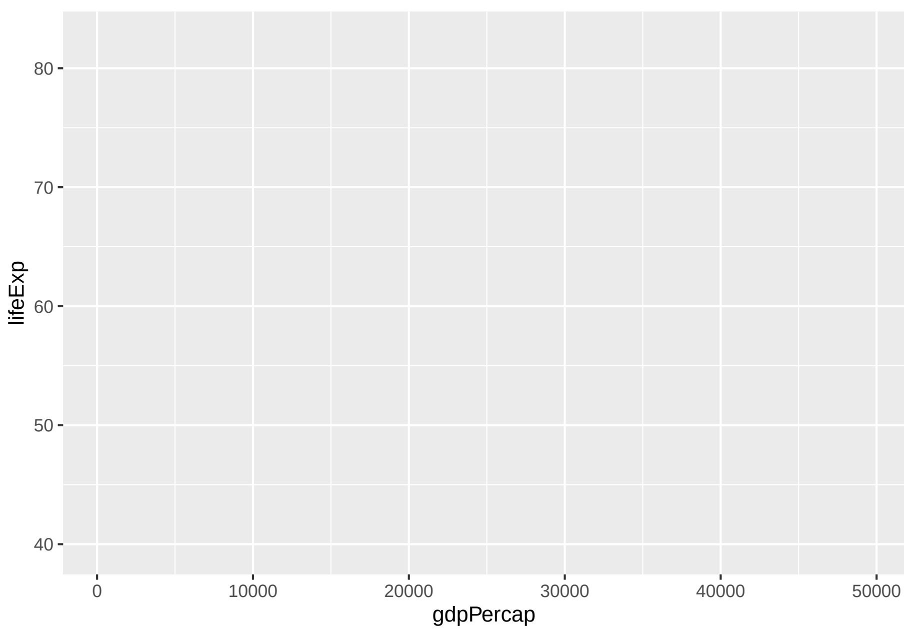
```
library(ggplot2)
library(dplyr)
library(gapminder)
data(gapminder)

# data preparation
gapminder_2007 <- gapminder %>%
  filter(year == '2007' & continent != 'Oceania') %>%
  select(-3) %>%
  mutate(pop = round(pop/1e6, 2))

head(gapminder_2007)
#> # A tibble: 6 x 5
#>   country     continent lifeExp   pop gdpPercap
#>   <fct>       <fct>       <dbl> <dbl>     <dbl>
#> 1 Afghanistan Asia         43.8  31.9      975.
#> 2 Albania     Europe       76.4   3.6     5937.
#> 3 Algeria     Africa       72.3  33.3     6223.
#> 4 Angola      Africa       42.7  12.4     4797.
#> 5 Argentina   Americas     75.3  40.3    12779.
#> 6 Austria     Europe       79.8   8.2    36126.
```

6.2 The data layer

The function `ggplot()` initializes a ggplot object. It can be used to pass in both data and aesthetic. Data and aesthetic passed in here becomes available to all subsequent layers but can be overridden if need be within subsequent layers.

```
# initializing plot with data
ggplot(data = gapminder_2007)
```

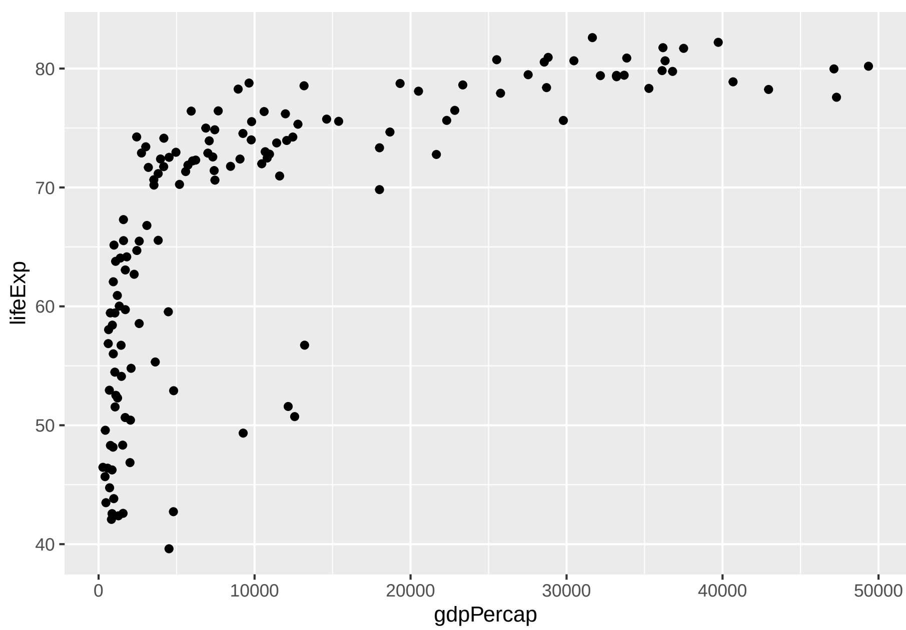
```
# mapping data to x and y-axis
ggplot(data = gapminder_2007,  mapping = aes(y = lifeExp, x = gdpPercap))
```



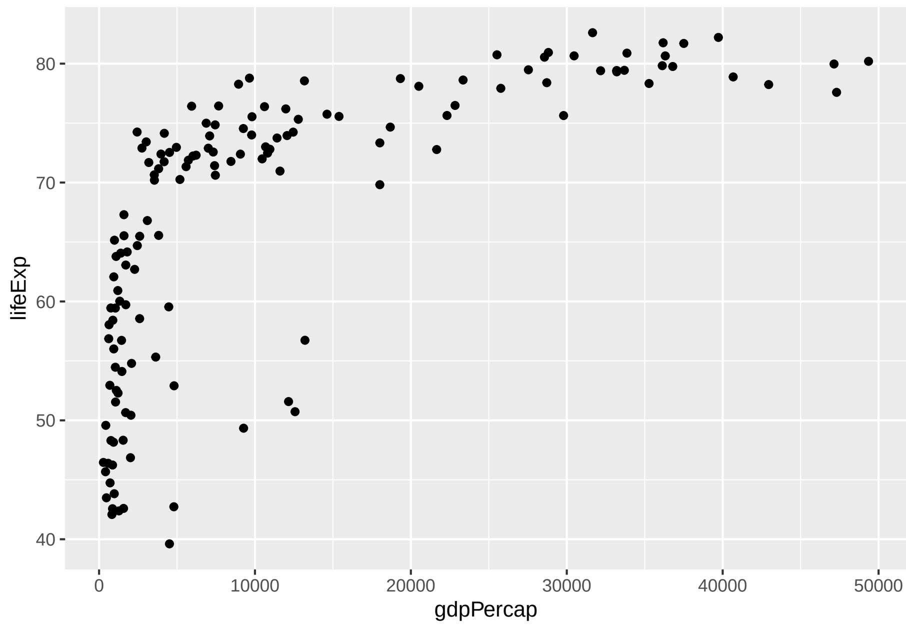
6.3 The geom layer

The geom layer declares the type of plot to be produced. More on this in the next chapter.

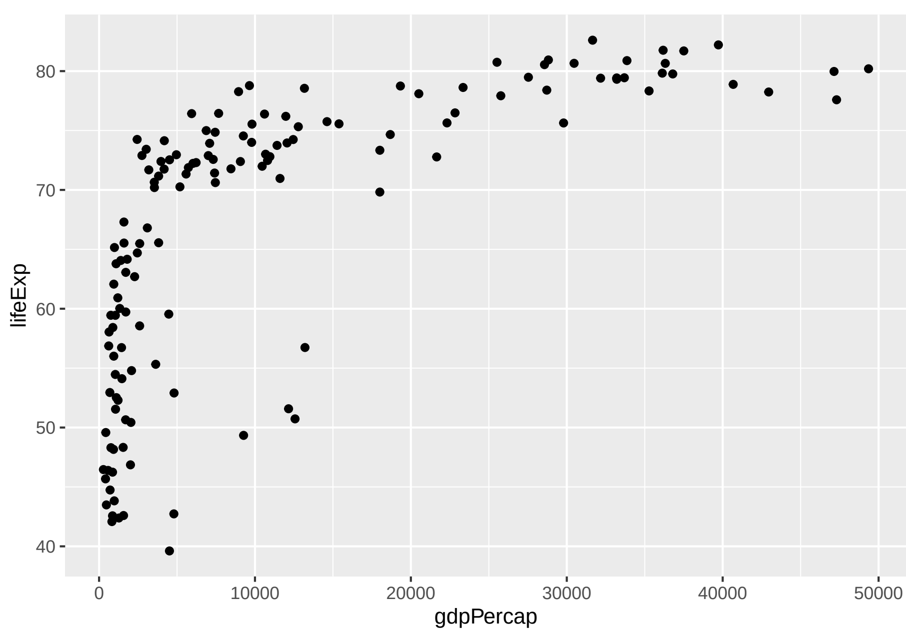
```
# adding the geom layer
ggplot(data = gapminder_2007, aes(y = lifeExp, x = gdpPercap)) +
  geom_point()
```




```
# Both data and axis can be declared within the geom layer.
ggplot(data = gapminder_2007) +
  geom_point(mapping = aes(y = lifeExp, x = gdpPercap))
```




```
ggplot() +
  geom_point(data = gapminder_2007, aes(y = lifeExp, x = gdpPercap))
```



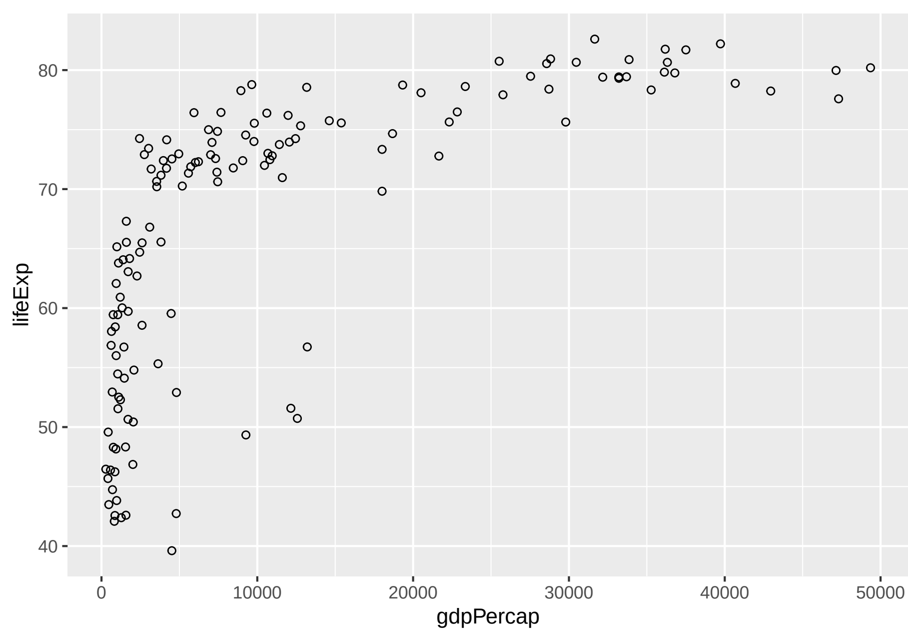
6.4 Shape

Shapes are controlled using the argument shape.

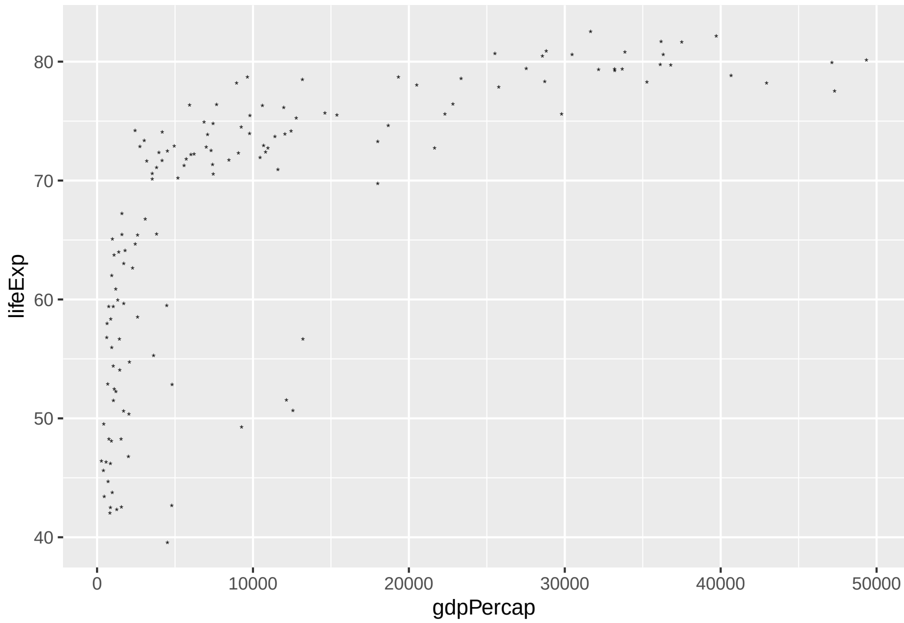
6.4.1 Setting shapes

Shapes are set by passing shape to geom_* but must be placed outside aes() as aes() is meant for mapping. Shape expects the same arguments as pch in base graphics that is, integers ranging from 1 to 25 or characters.

```
# changing shapes
ggplot(gapminder_2007) +
  geom_point(aes(y = lifeExp, x = gdpPercap), shape = 21)
```



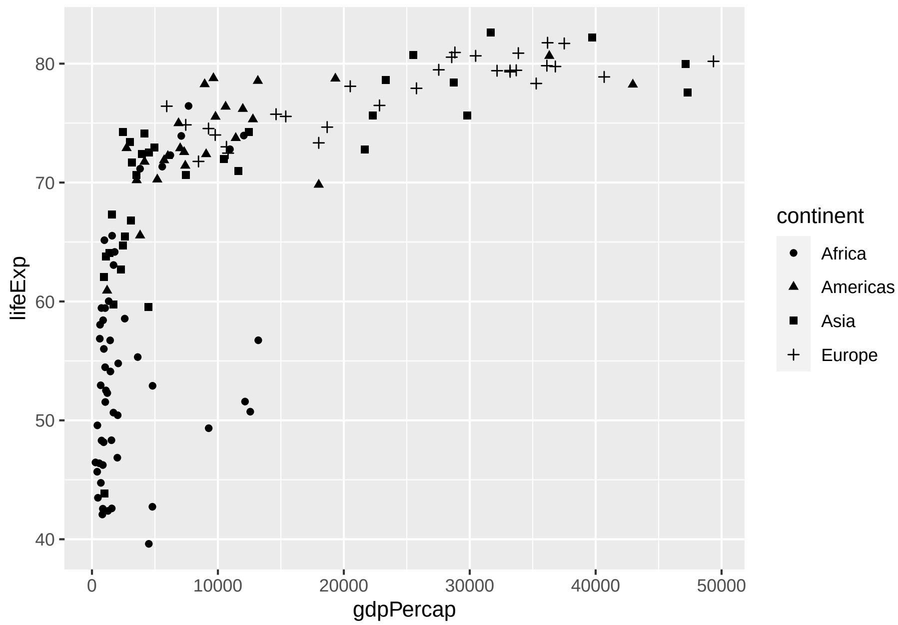

```
# using a character
ggplot(gapminder_2007) +
  geom_point(aes(y = lifeExp, x = gdpPercap), shape = '*')
```



6.4.2 Mapping shapes

The mapping of data to shapes allows us to have shapes by groups or categories for example having different shapes for different continents. To map data to shapes, the shape argument is passed a categorical variable and placed within `aes()`.

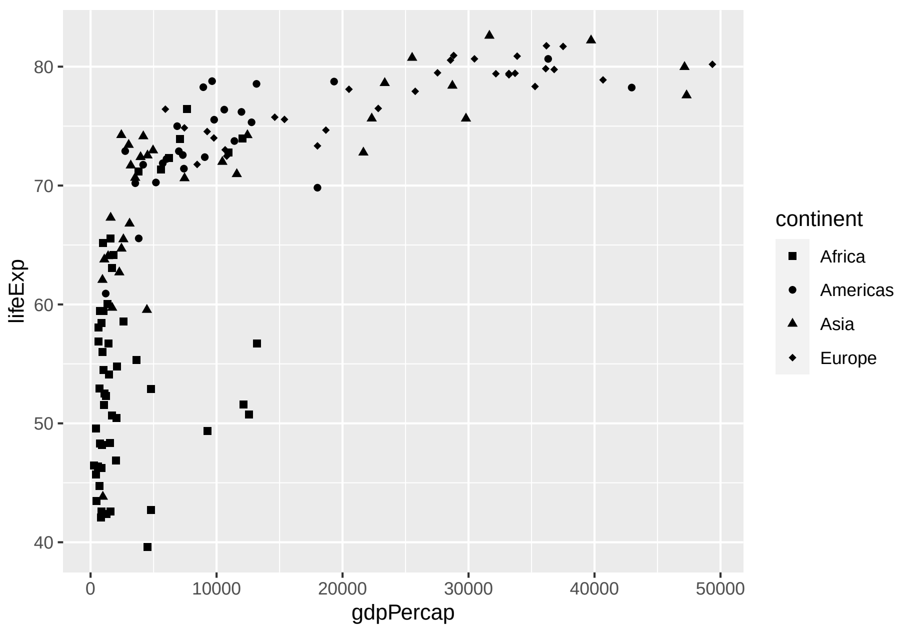
```
# shapes by continent
ggplot(gapminder_2007) +
  geom_point(aes(y = lifeExp, x = gdpPercap, shape = continent))
```



6.4.3 Scaling shapes

The function `scale_shape_manual()` is used to scale shapes that is determine the shapes to use in the plot.

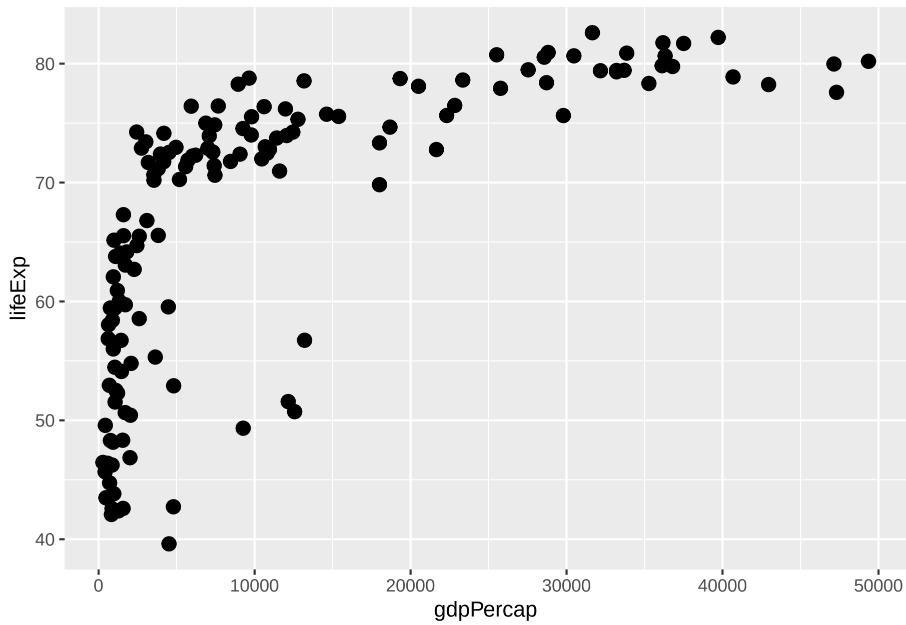
```
# using shapes ranging from 15 to 19
ggplot(gapminder_2007) +
  geom_point(aes(y = lifeExp, x = gdpPercap, shape = continent)) +
  scale_shape_manual(values = 15:19)
```



6.5 Size

size is controlled using the argument `size=`.

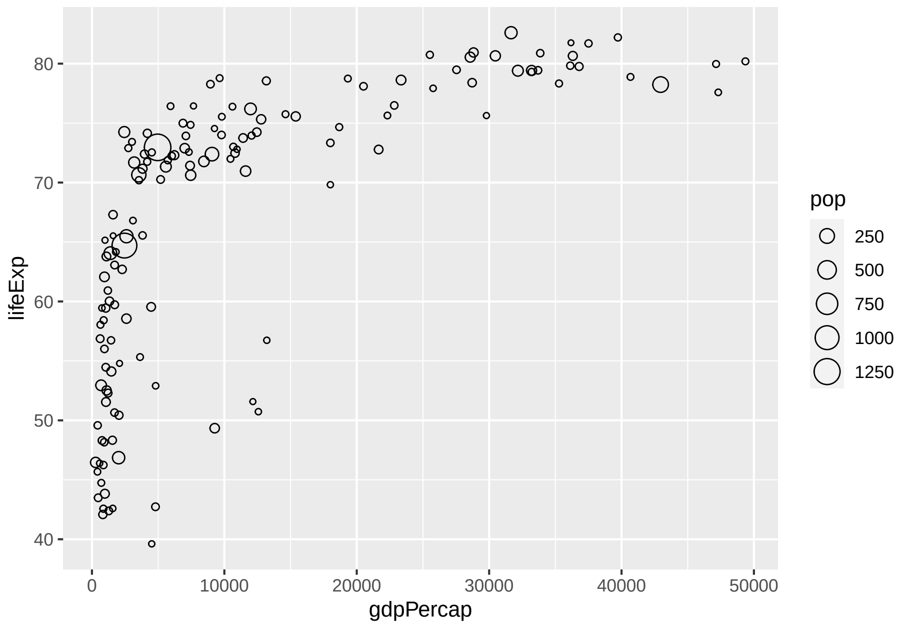
```
# adjusting size
ggplot(gapminder_2007) +
  geom_point(aes(y = lifeExp, x = gdpPercap), size = 3)
```



6.5.2 Mapping size

Size is mapped by assigning them a continuous variable and placing them within `aes()`.

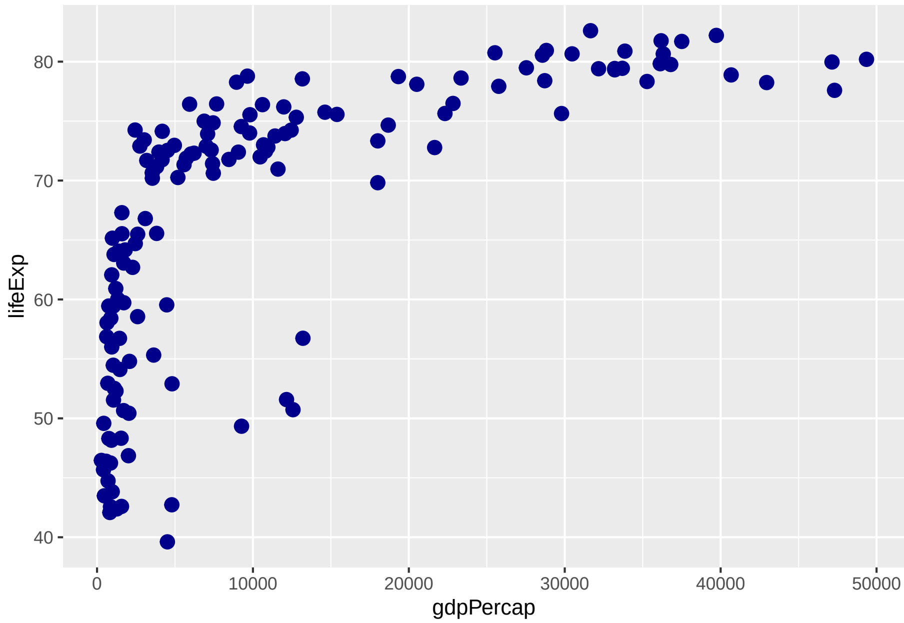
```
# size by population
ggplot(gapminder_2007) +
  geom_point(aes(y = lifeExp, x = gdpPercap, size = pop), shape = 21)
```



6.6 Colour

Colour is controlled using the argument `color=` or `colour=`.

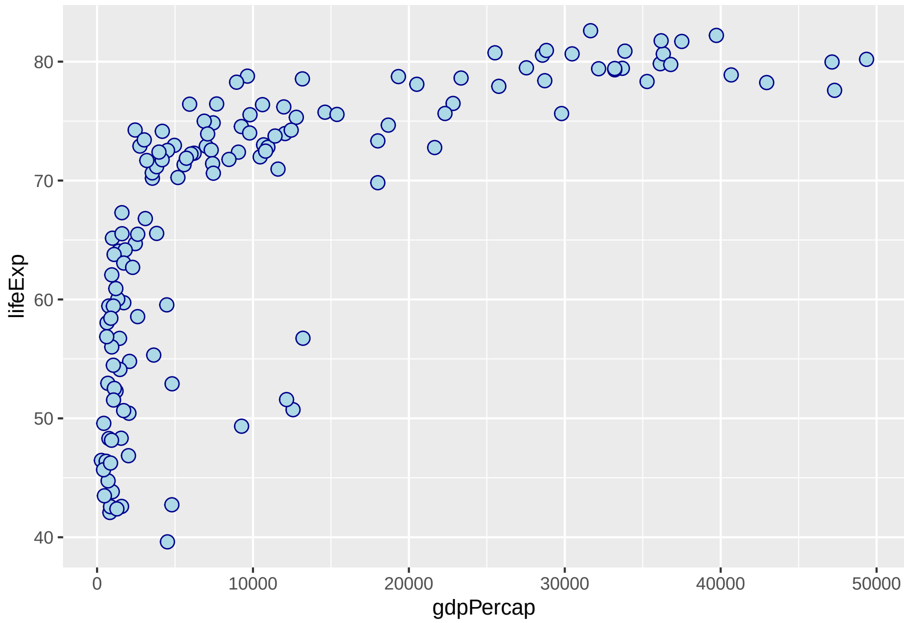
```
ggplot(gapminder_2007) +
  geom_point(aes(y = lifeExp, x = gdpPercap), colour = 'darkblue', size = 3, shape = 19)
```



6.6.2 Fill vs colour

With shapes between 21 to 25 and bars, the argument fill is used to fill shapes while colour is used to colour borders (outlines).

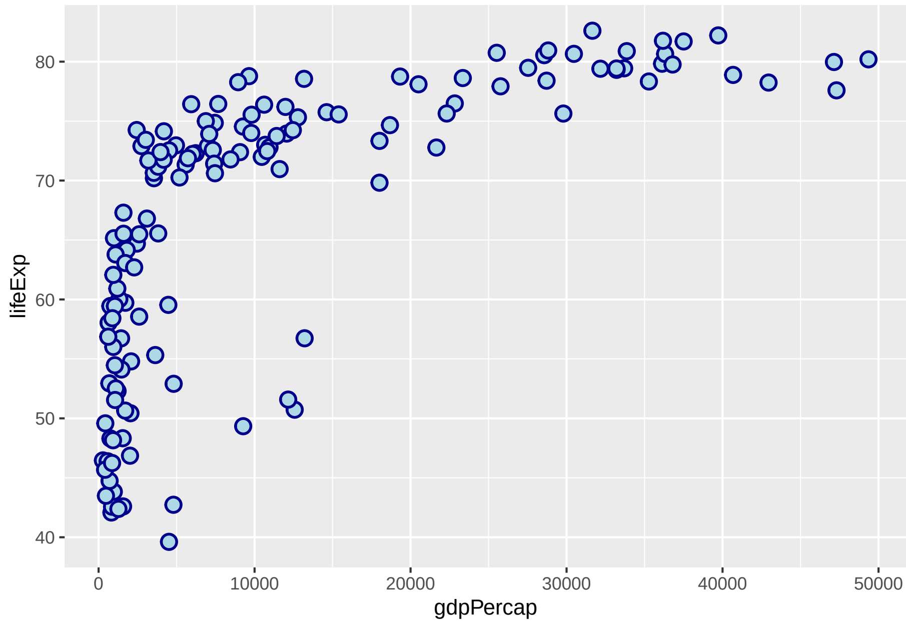
```
# using colour and fill
ggplot(gapminder_2007) +
geom_point(aes(y = lifeExp, x = gdpPercap), colour = 'darkblue', fill = 'lightblue',
           size = 3, shape = 21)
```



6.6.3 Stroke

The border or outline size is controlled using the argument `stroke=`.

```
ggplot(gapminder_2007) +
  geom_point(aes(y = lifeExp, x = gdpPercap), colour = 'darkblue', fill = 'lightblue',
             size = 3, shape = 21, stroke = 1)
```



6.6.4 Transparency

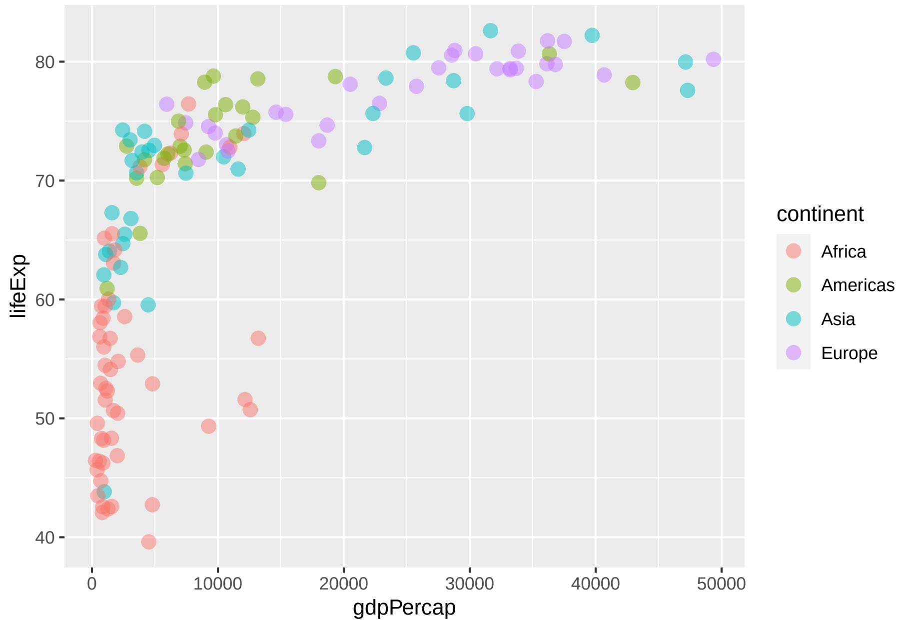
Transparency is controlled by the argument `alpha=`. It accepts values from 0 to 1.

```
ggplot(gapminder_2007) +
  geom_point(aes(y = lifeExp, x = gdpPercap),
             colour = 'darkblue', fill = 'lightblue', size = 3, shape = 21,
             stroke = 1, alpha = 0.5)
```

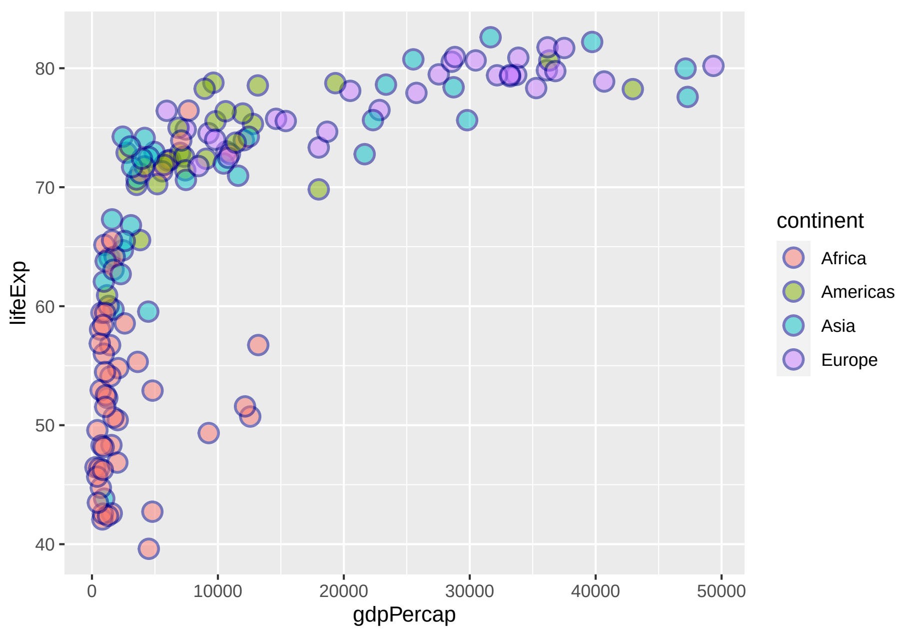
6.6.5 Mapping colours to discrete variables

As with shapes, colours are mapped by assigning a discrete variable to them and placing them within `aes()`.

```
# colour by continent
ggplot(gapminder_2007) +
  geom_point(aes(y = lifeExp, x = gdpPercap, colour = continent), size = 3,
             shape = 19, alpha = 0.5)
```



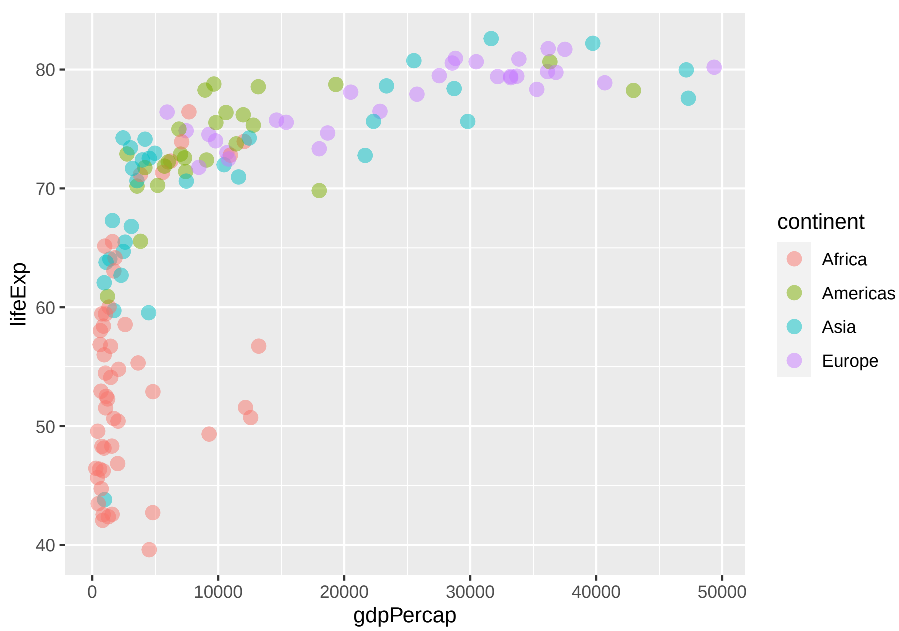

```
# fill by continent
ggplot(gapminder_2007) +
  geom_point(aes(y = lifeExp, x = gdpPercap, fill = continent),
             colour = 'darkblue', size = 4, shape = 21, alpha = 0.5, stroke = 1)
```



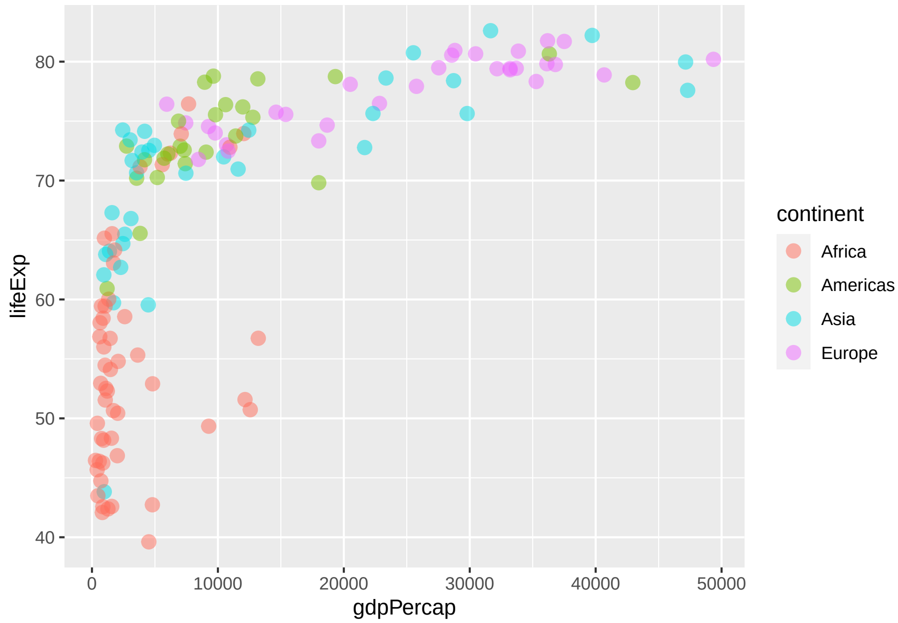
6.6.6 Default colours

The functions `scale_colour_hue()` and `scale_fill_hue()` sets the default colour and fill scale for discrete variables.

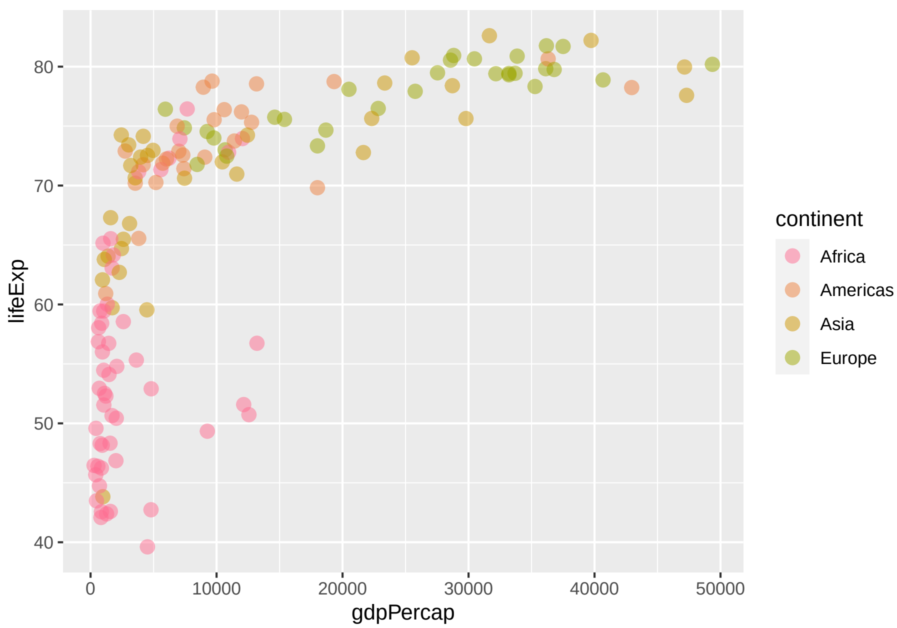
```
ggplot(gapminder_2007) +
  geom_point(aes(y = lifeExp, x = gdpPercap, colour = continent), size = 3,
             shape = 19, alpha = 0.5) +
  scale_colour_hue()
```




```
# Adjust luminosity and chroma
ggplot(gapminder_2007) +
  geom_point(aes(y = lifeExp, x = gdpPercap, colour = continent), size = 3,
             shape = 19, alpha = 0.5) +
  scale_colour_hue(l = 70, c = 150)
```



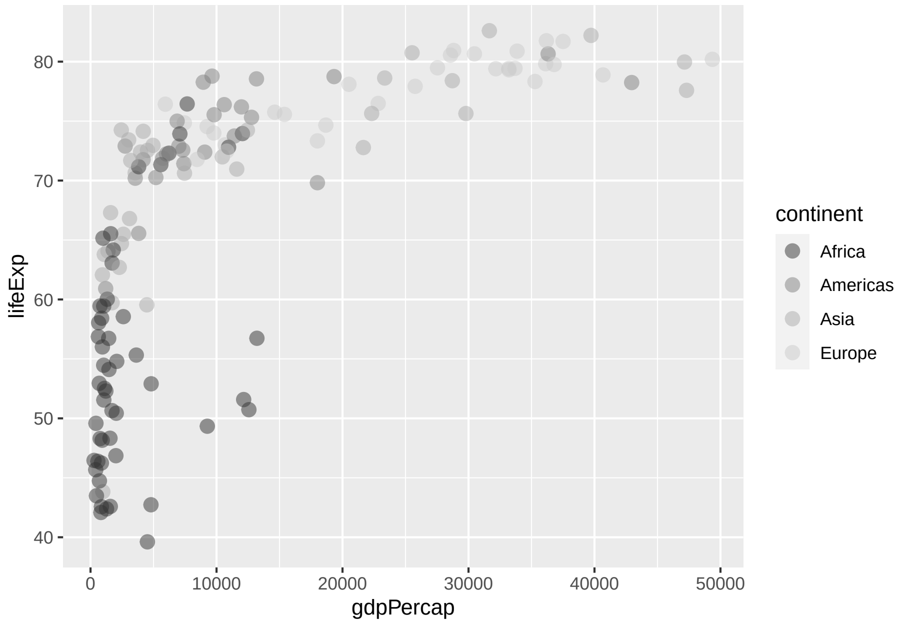

```
# Changing the range of hues used
ggplot(gapminder_2007) +
  geom_point(aes(y = lifeExp, x = gdpPercap, colour = continent), size = 3,
             shape = 19, alpha = 0.5) +
  scale_colour_hue(h = c(0, 90))
```



6.6.7 Grey colours

The function `scale_colour_grey()` defines grey colours for discrete variables.

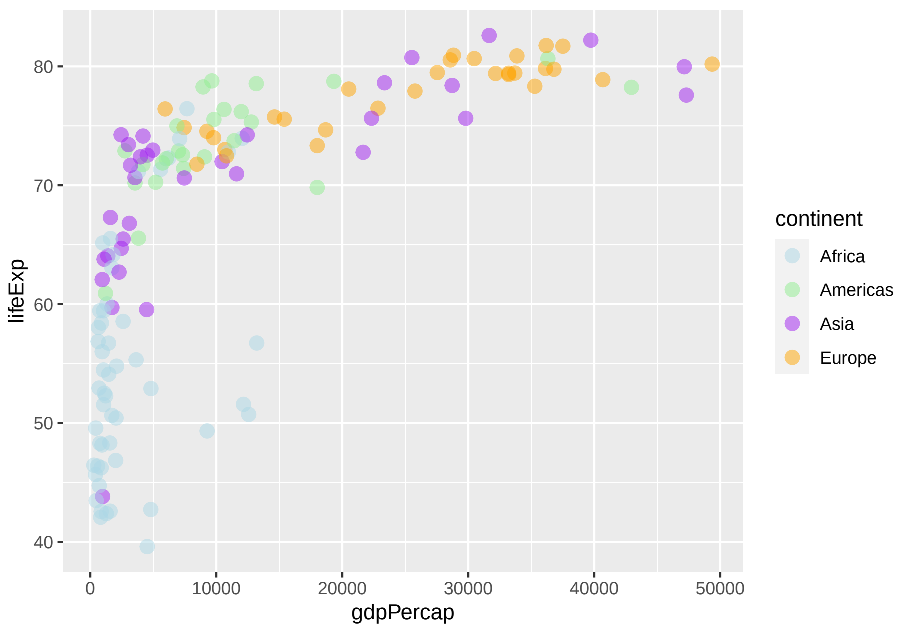
```
ggplot(gapminder_2007) +
  geom_point(aes(y = lifeExp, x = gdpPercap, colour = continent), size = 3,
             shape = 19, alpha = 0.5) +
  scale_colour_grey()
```



6.6.8 Manually specifying colours

The functions `scale_colour_manual()` and `scale_fill_manual()` specify colour and fill, respectively.

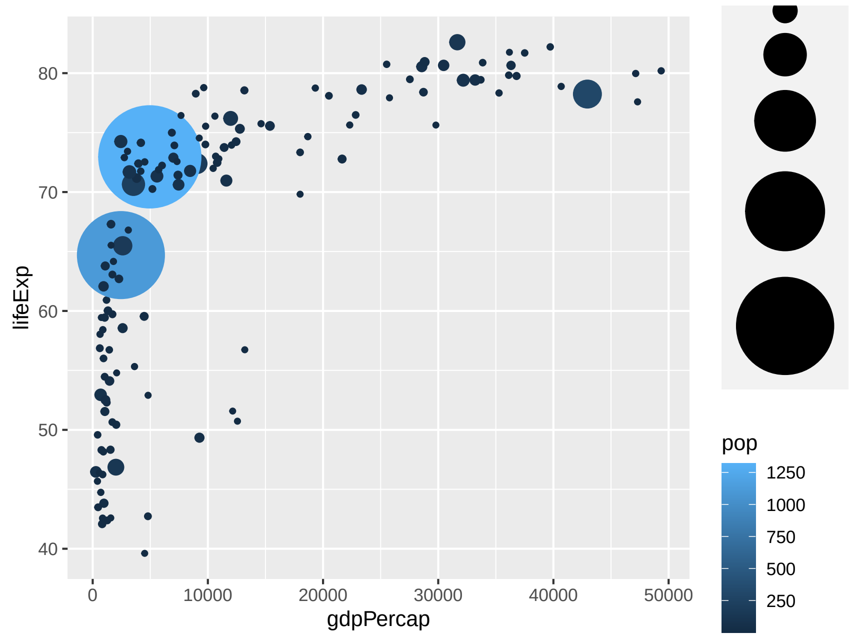
```
ggplot(gapminder_2007) +
  geom_point(aes(y = lifeExp, x = gdpPercap, colour = continent), size = 3,
             shape = 19, alpha = 0.5) +
  scale_colour_manual(values = c('lightblue', 'lightgreen', 'purple', 'orange', 'pink'))
```



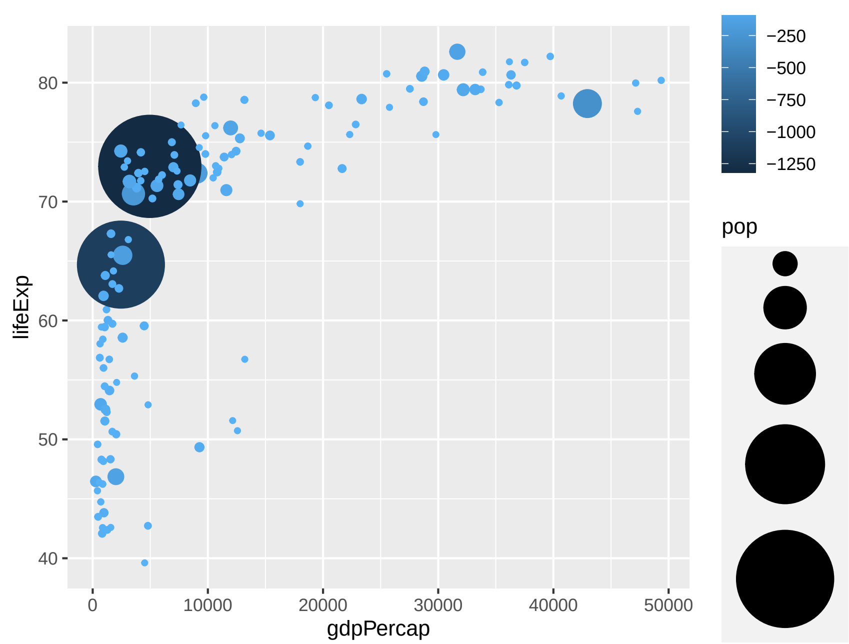
6.6.9 Mapping colours by continuous variables

As with sizes, colours are mapped by assigning a continuous variable to them and placing them within `aes()`.

```
# colour by pop
ggplot(gapminder_2007) +
  geom_point(aes(y = lifeExp, x = gdpPercap, size = pop, col = pop), shape = 19) +
  scale_radius(range = c(1, 24))
```




```
# reversing colour with desc()
ggplot(gapminder_2007) +
  geom_point(aes(y = lifeExp, x = gdpPercap, size = pop, colour = desc(pop)), shape =
  scale_radius(range = c(1, 24))
```

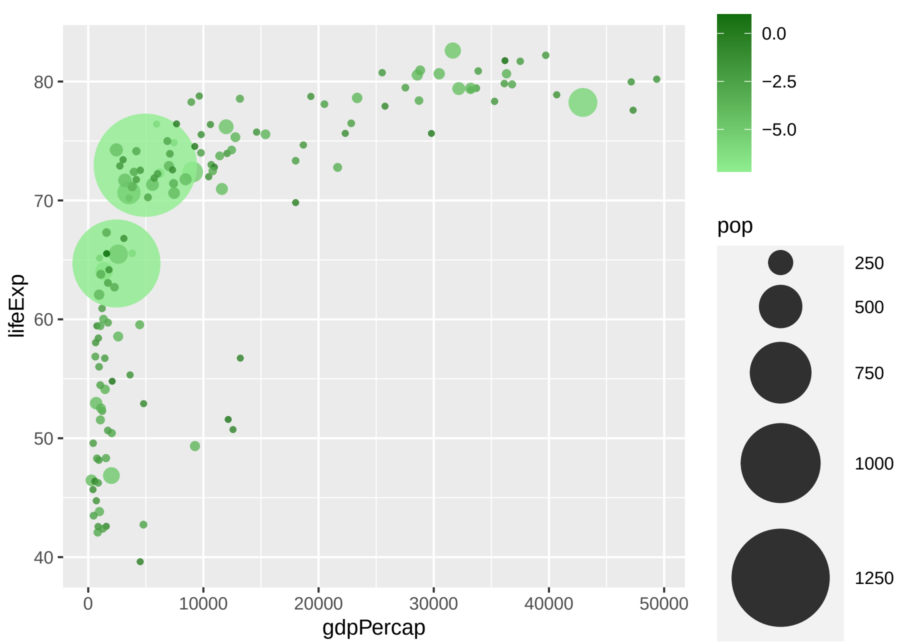


6.6.10 Manually defining colours

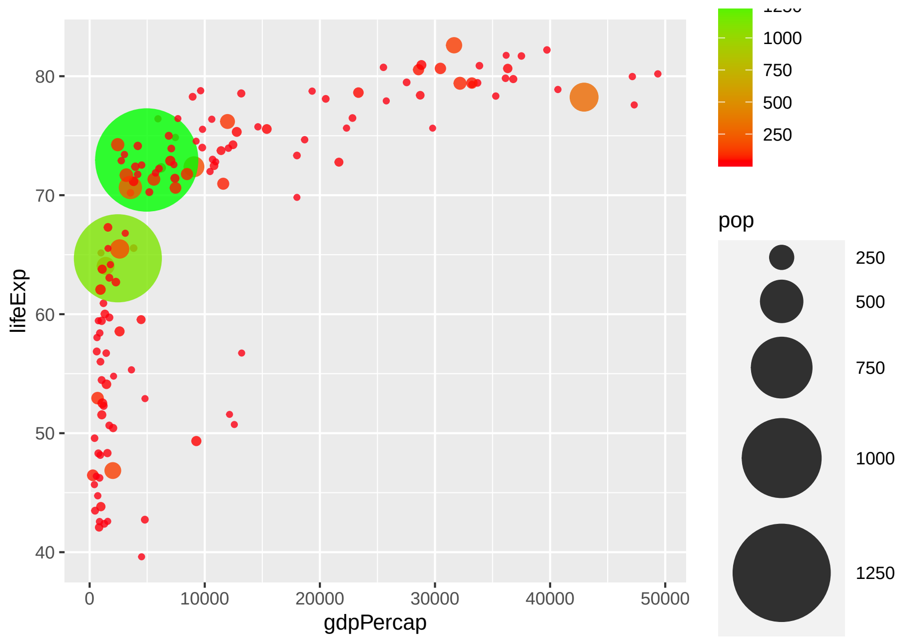
The functions:

scale_colour_gradient() and scale_fill_gradient() defines a two-colour gradient scale_colour_gradient2() and scale_fill_gradient2() defines a three-colour gradient (low-mid-high) scale_colour_gradientn() and scale_fill_gradientn() defines a more then three colour gradient

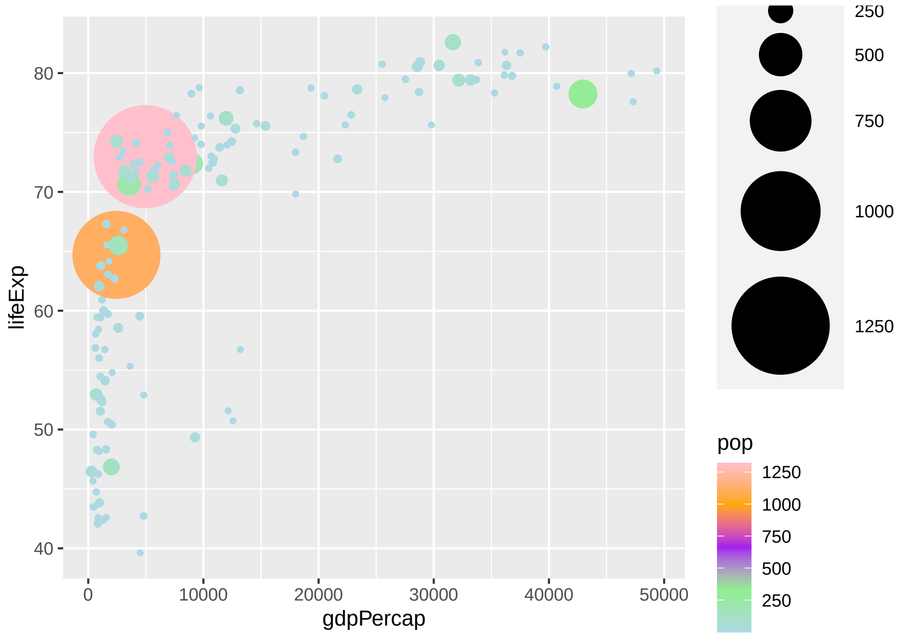
```
# two colour gradient
ggplot(gapminder_2007) +
  geom_point(aes(y = lifeExp, x = gdpPercap, size = pop, col = desc(log(pop))),
             shape = 19, alpha = 0.8) +
  scale_radius(range = c(1, 24)) +
  scale_colour_gradient(low = 'lightgreen', high = 'darkgreen')
```




```
# three colour gradient
ggplot(gapminder_2007) +
  geom_point(aes(y = lifeExp, x = gdpPercap, size = pop, col = pop),
             shape = 19, alpha = 0.8) +
  scale_radius(range = c(1, 24)) +
  scale_colour_gradient2(low = 'blue', mid = 'red', high = 'green',
                         midpoint = mean(gapminder_2007$pop))
```




```
# five colour gradient
ggplot(gapminder_2007) +
  geom_point(aes(y = lifeExp, x = gdpPercap, size = pop, col = pop), shape = 19) +
  scale_radius(range = c(1, 24)) +
  scale_colour_gradientn(colors = c('lightblue', 'lightgreen', 'purple', 'orange', 'pi
```



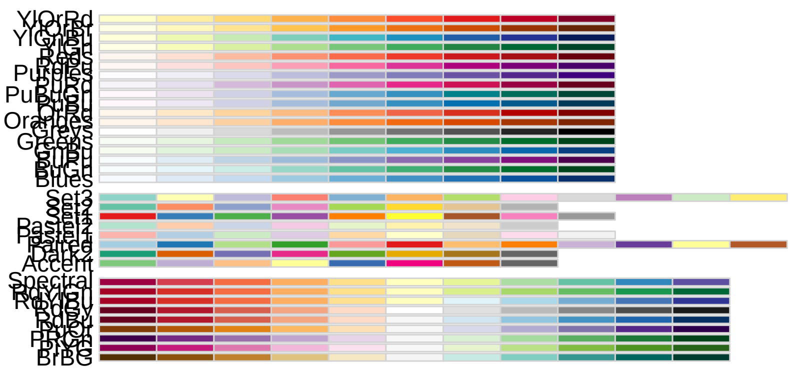
6.7 Colour palettes

6.7.1 rcolorbrewer

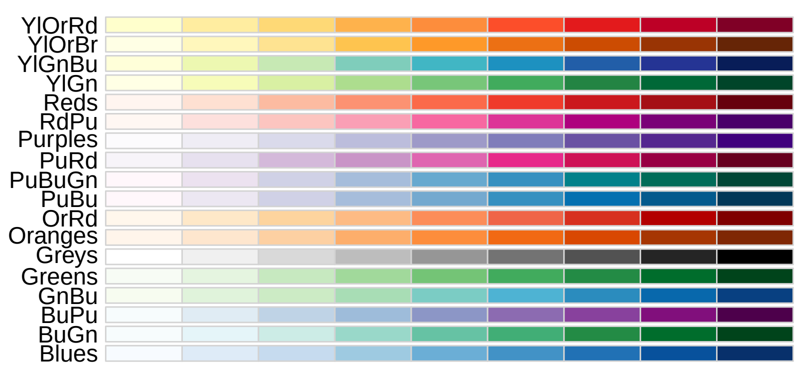
RcolorBrewer is R's implementation of ColorBrewer. It classifies colours into three board classes:

seq (sequential): suited for data which has an order, progressing from low to high div (diverging): suited for data with two extremes, one for positive and the other for negative values qual (qualitative): suited for data which colour bears no meaning. (nominal and categorical data)

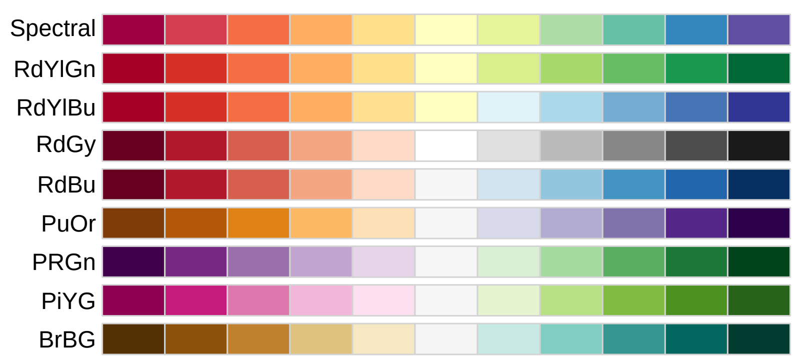
```
library(RColorBrewer)
# displays all the various palettes in RcolorBrewer
display.brewer.all()
```



```
# display sequential colours
display.brewer.all(type = "seq")
```

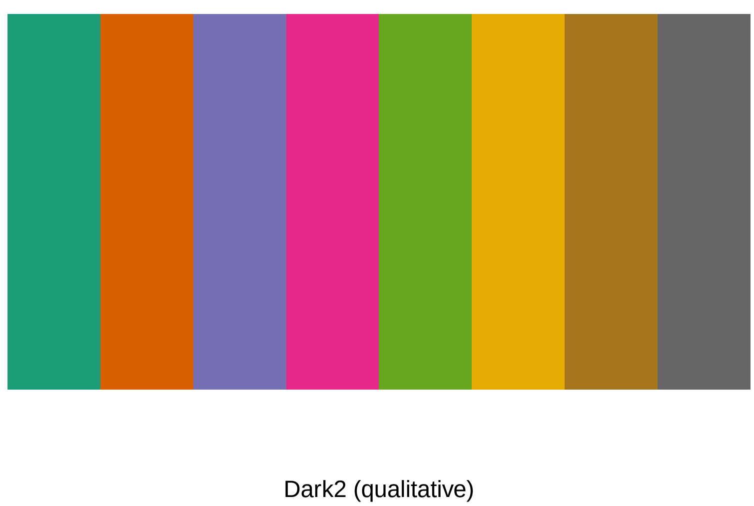


```
# display diverging colours
display.brewer.all(type = "div")
```



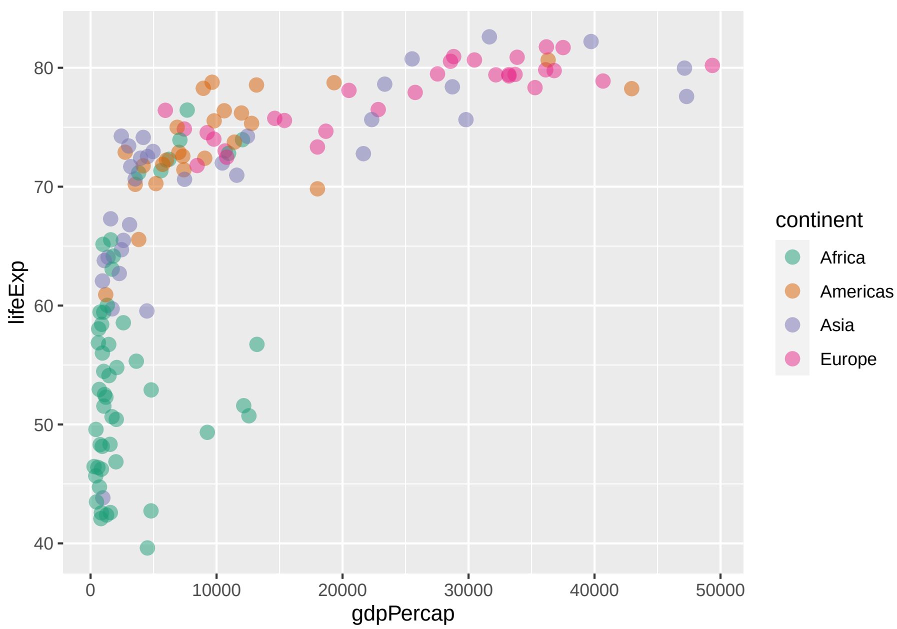

```
# display qualitative colours
display.brewer.all(type = "qual")
```


```
# displaying a particular colour palette
display.brewer.pal(n = 8, name = 'Dark2')
```



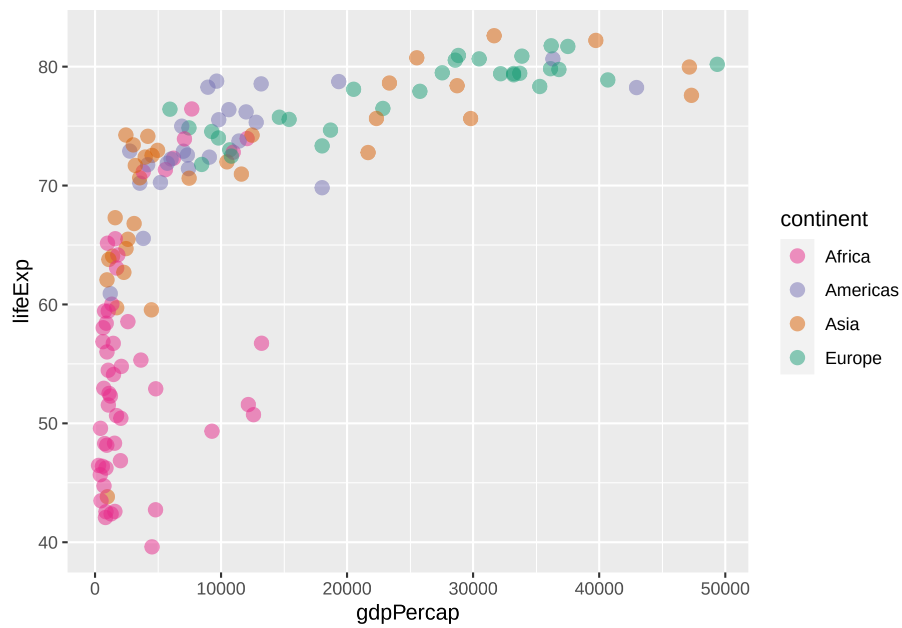
The functions `scale_colour_brewer()` and `scale_fill_brewer()` defines colour scale for discrete variables.

```
# discrete variable
ggplot(gapminder_2007) +
  geom_point(aes(y = lifeExp, x = gdpPercap, colour = continent), size = 3, shape = 19,
             alpha = 0.5) +
  scale_colour_brewer(palette = "Dark2")
```



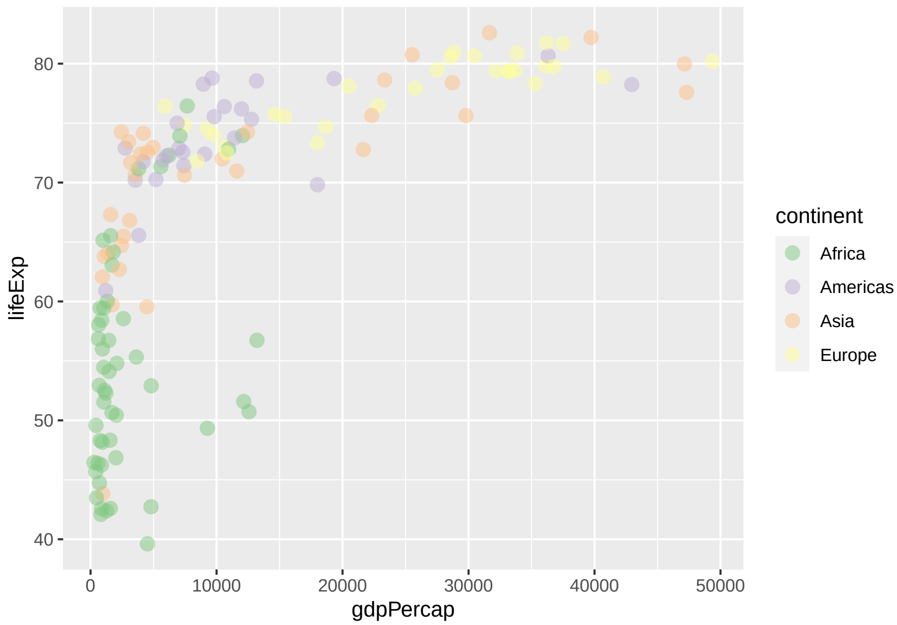
The argument direction reverses the order of the colours.

```
# reversing colours with direction
ggplot(gapminder_2007) +
  geom_point(aes(y = lifeExp, x = gdpPercap, colour = continent), size = 3, shape = 19,
             alpha = 0.5) +
  scale_colour_brewer(palette = "Dark2", direction = -1)
```

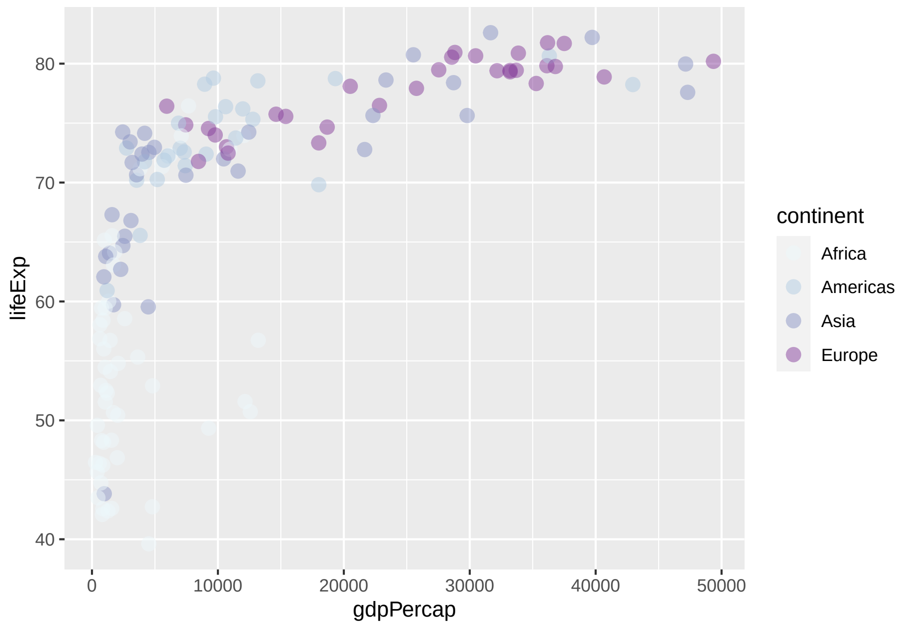


The type of palette is specified by the argument type.

```
# specifying palette class
ggplot(gapminder_2007) +
  geom_point(aes(y = lifeExp, x = gdpPercap, colour = continent), size = 3, shape = 19
             alpha = 0.5) +
  scale_colour_brewer(type = 'qual', palette = 1)
```

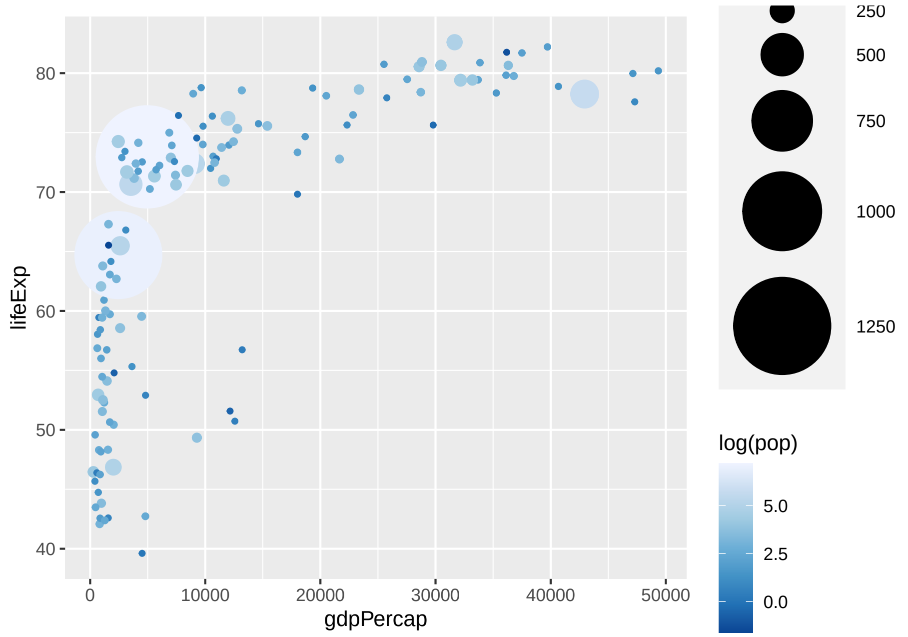



```
# specifying palette class
ggplot(gapminder_2007) +
  geom_point(aes(y = lifeExp, x = gdpPercap, colour = continent), size = 3, shape = 19,
             alpha = 0.5) +
  scale_colour_brewer(type = 'seq', palette = 3)
```

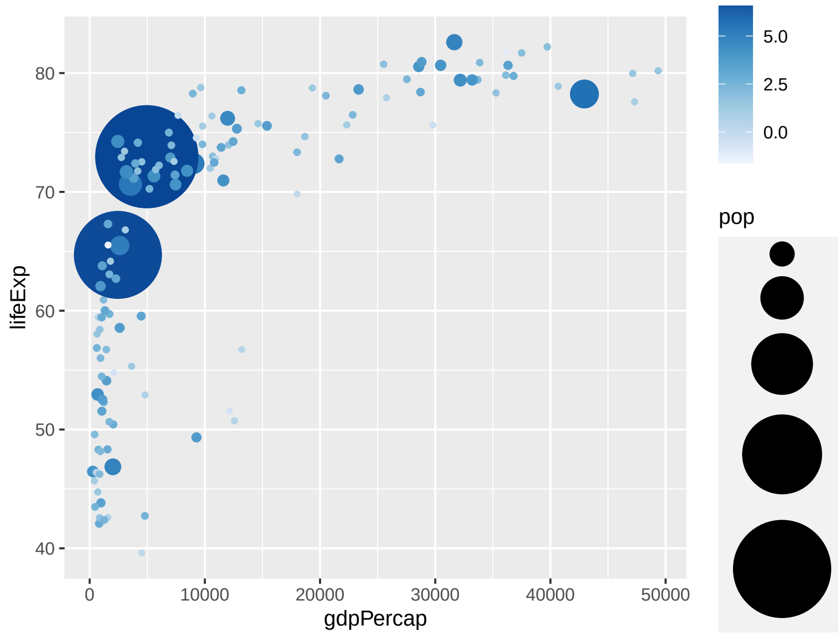


The functions `scale_colour_distiller()` and `scale_fill_distiller()` defines colour scale for continuous variables.

```
# continuous variable
ggplot(gapminder_2007) +
  geom_point(aes(y = lifeExp, x = gdpPercap, size = pop, col = log(pop)), shape = 19) +
  scale_radius(range = c(1, 24)) +
  scale_colour_distiller(palette = 'Blues')
```




```
# continuous variable
ggplot(gapminder_2007) +
  geom_point(aes(y = lifeExp, x = gdpPercap, size = pop, col = log(pop)), shape = 19) +
  scale_radius(range = c(1, 24)) +
  scale_colour_distiller(palette = 1, direction = 1)
```



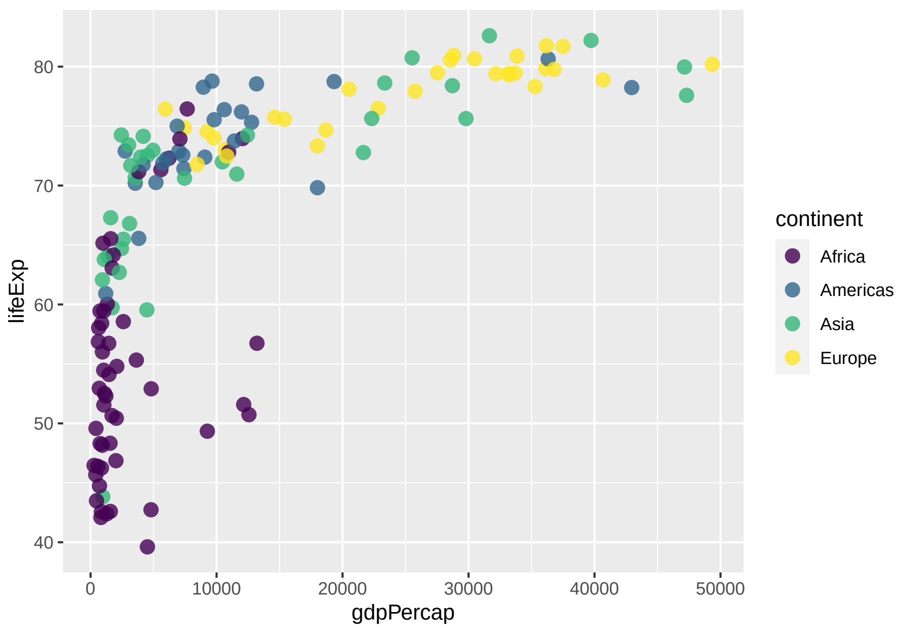
6.7.2 The viridis color palettes

The viridis package brings to R colour scales created by Stéfan van der Walt and Nathaniel Smith for the Python data visualization package matplotlib. viridis comes with the following colour palettes:

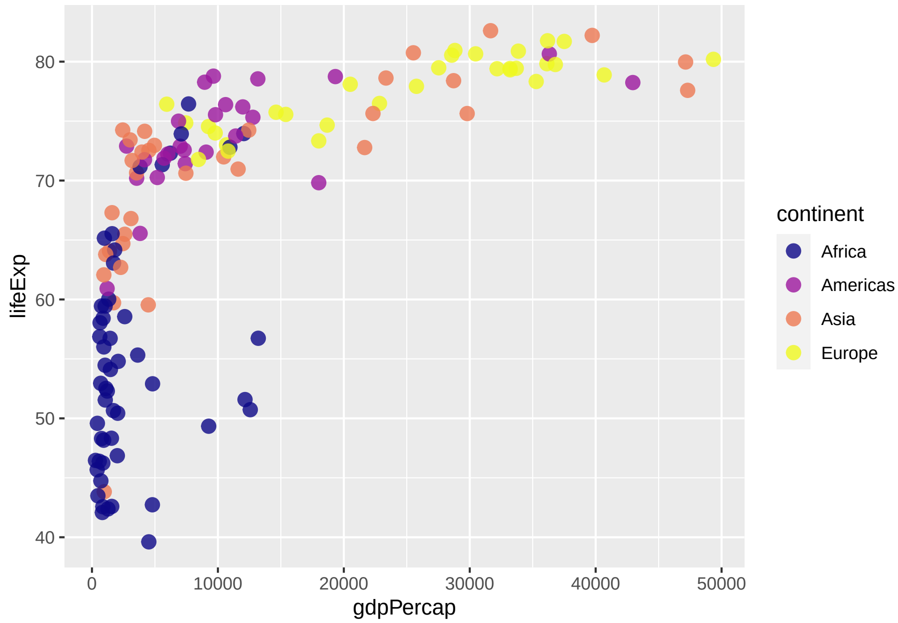
- Viridis (default)
- magma
- plasma
- inferno

The functions `scale_colour_viridis()` and `scale_fill_viridis()` defines colour scale for both discrete and continuous variables, with `discrete = TRUE` indicating discrete while `discrete = FALSE` indicating continuous. To be more specific, use `scale_colour_viridis_d()` for discrete and`scale_colour_viridis_c()` for continuous.

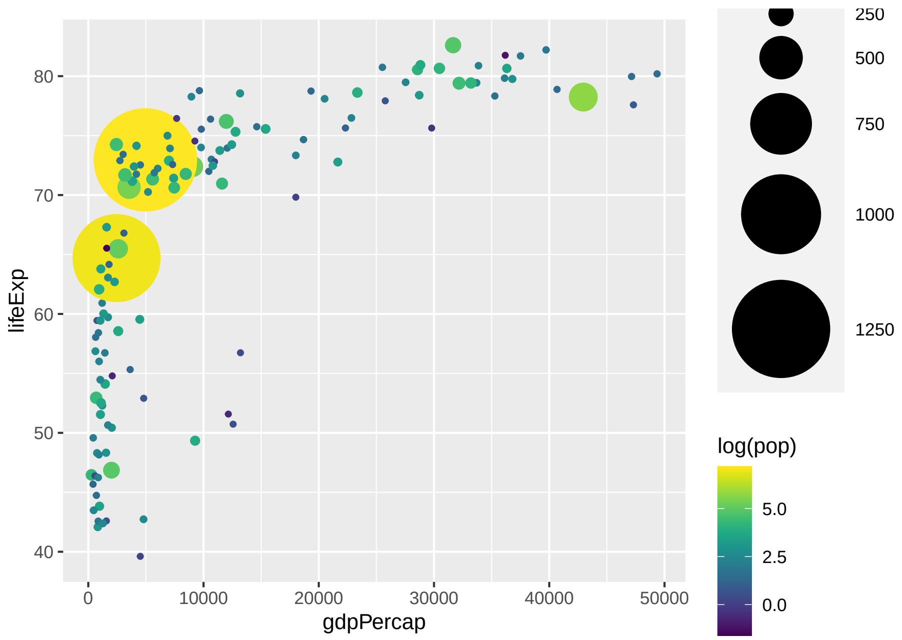
```
library(viridis)
# discrete variable
ggplot(gapminder_2007) +
  geom_point(aes(y = lifeExp, x = gdpPercap, colour = continent), size = 3,
           shape = 19, alpha = 0.8) +
  scale_colour_viridis(discrete = TRUE)
```



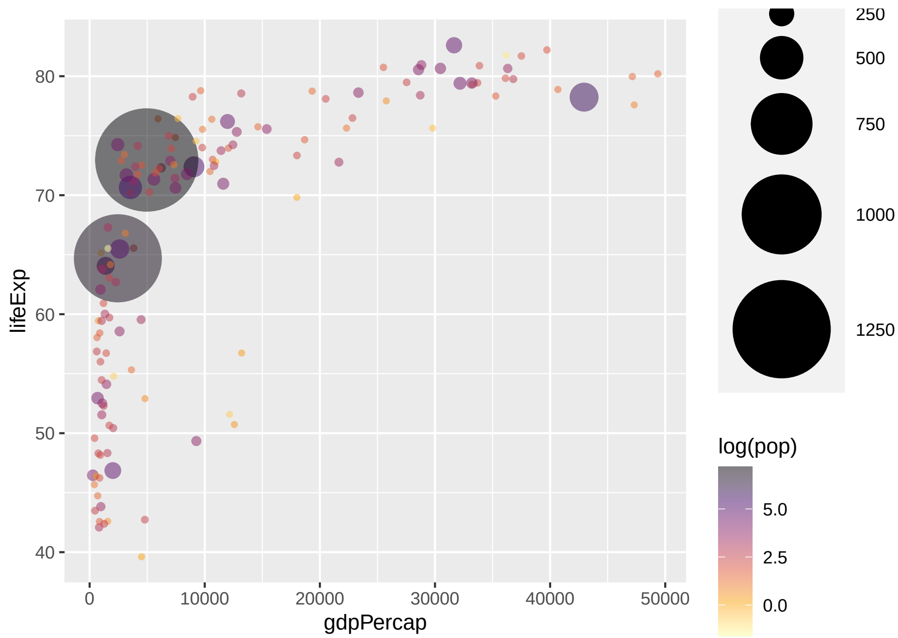

```
# discrete variable
ggplot(gapminder_2007) +
  geom_point(aes(y = lifeExp, x = gdpPercap, colour = continent), size = 3,
             shape = 19, alpha = 0.8) +
  scale_colour_viridis_d(option = 'plasma')
```




```
# continuous variable
ggplot(gapminder_2007) +
  geom_point(aes(y = lifeExp, x = gdpPercap, size = pop, col = log(pop)), shape = 19) +
  scale_radius(range = c(1, 24)) +
  scale_colour_viridis()
```



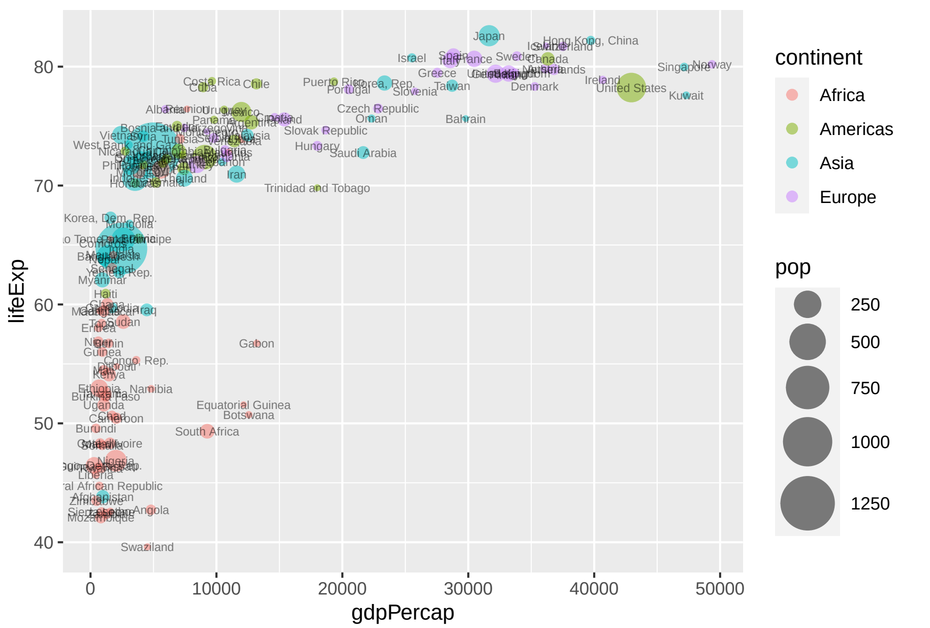

```r
# continuous variable
ggplot(gapminder_2007) +
  geom_point(aes(y = lifeExp, x = gdpPercap, size = pop, col = log(pop)), shape = 19) +
  scale_radius(range = c(1, 24)) +
  scale_colour_viridis_c(option = 'inferno', direction = -1, alpha = 0.5)
```



6.8 Text

The function `geom_text()` adds text to a plot.

```
ggplot(data = gapminder_2007, aes(y = lifeExp, x = gdpPercap)) +
  geom_point(alpha = 0.5, stroke = 1, aes(size = pop, colour = continent)) +
  scale_size_area(max_size = 12) +
  geom_text(aes(label = country), size = 2, alpha = 0.5)
```

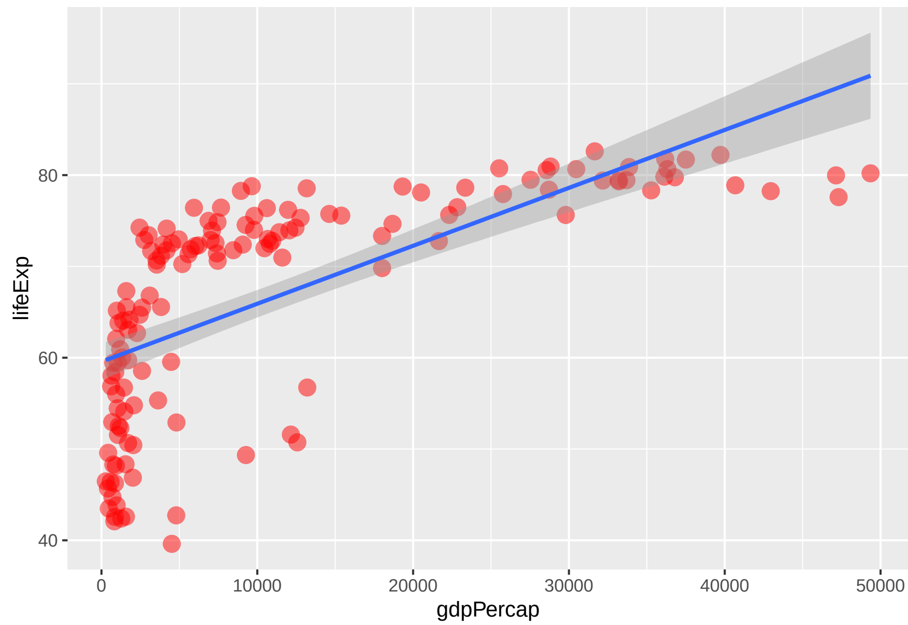


6.9 Fitting a regression line to a plot

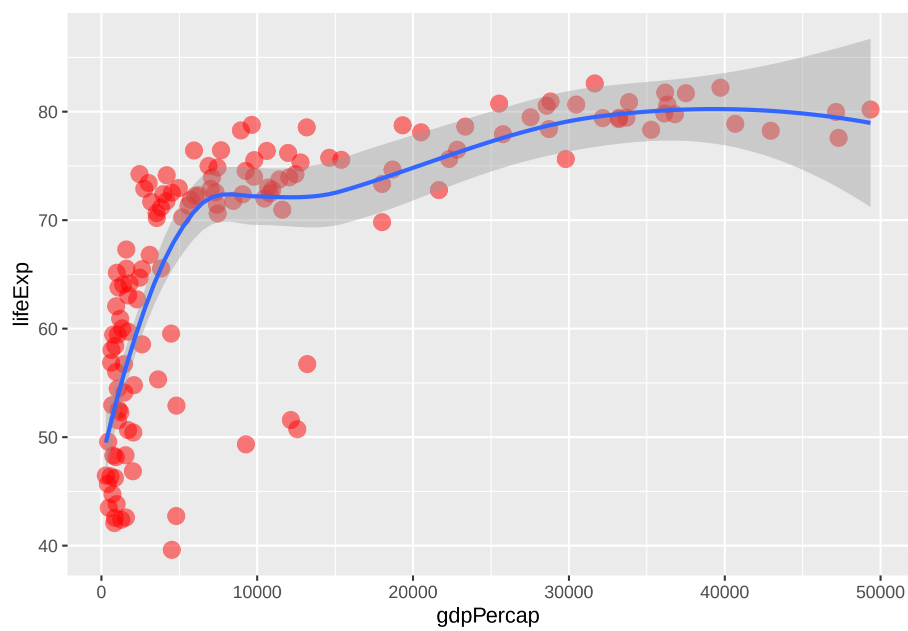
The function `geom_smooth()` adds a regression line to a plot. We use the arguments:

method = lm for linear, method = loess for loess and se = FALSE to remove the confidence intervals.

```
# adding a linear line
ggplot(data = gapminder_2007, aes(y = lifeExp, x = gdpPercap)) +
  geom_point(colour = 'red', size = 3, shape = 19, alpha = 0.5, stroke = 1) +
  geom_smooth(method = lm)
```




```
# changing to loess
ggplot(data = gapminder_2007, aes(y = lifeExp, x = gdpPercap)) +
  geom_point(colour = 'red', size = 3, shape = 19, alpha = 0.5, stroke = 1) +
  geom_smooth(method = loess)
```

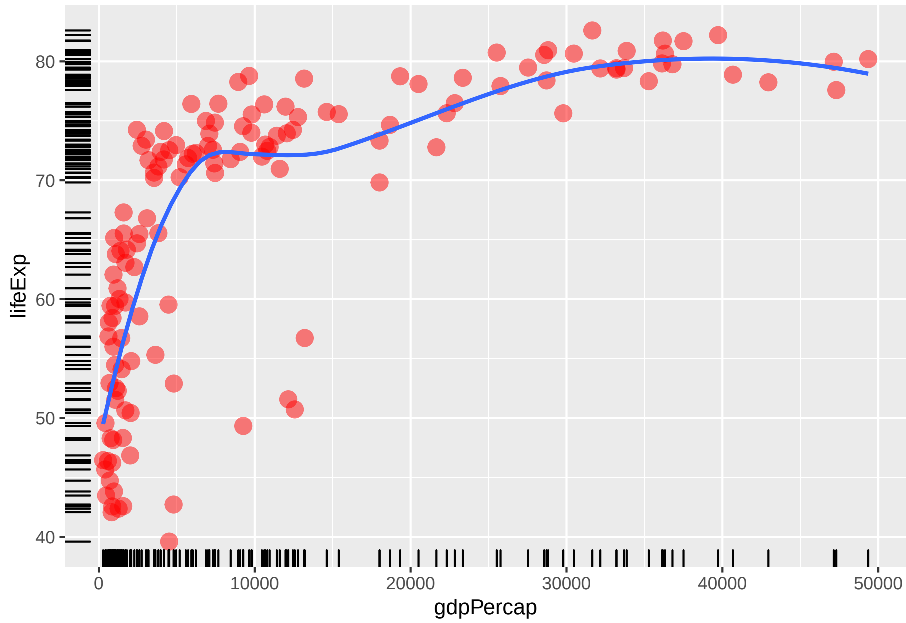


```
# removing the confidence intervals
ggplot(data = gapminder_2007, aes(y = lifeExp, x = gdpPercap)) +
  geom_point(colour = 'red', size = 3, shape = 19, alpha = 0.5, stroke = 1) +
  geom_smooth(method = loess, se = FALSE)
```

6.10 Adding some rug

The function `geom_rug()` adds rug to a plot.

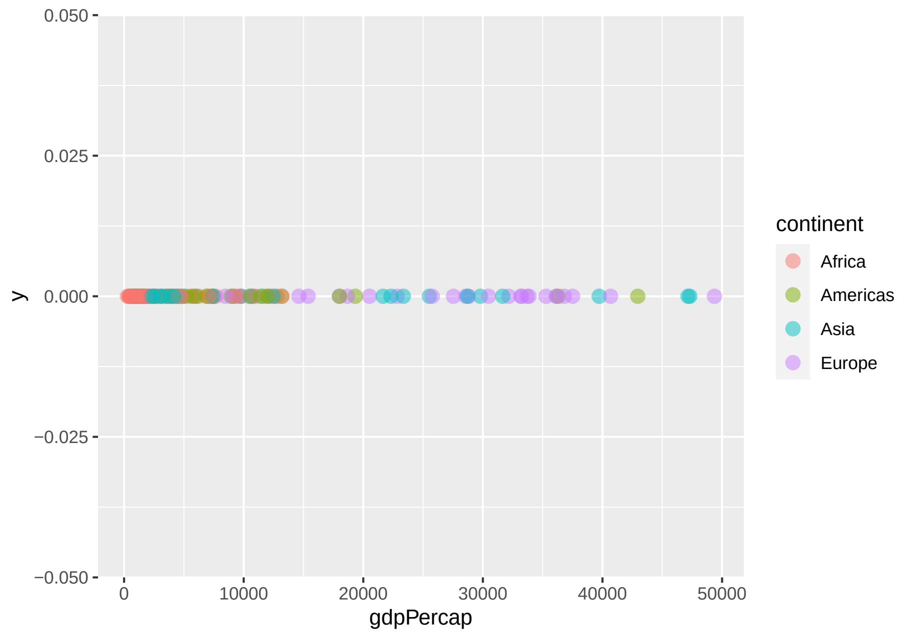
```
ggplot(data = gapminder_2007, aes(y = lifeExp, x = gdpPercap)) +
  geom_point(colour = 'red', size = 3, shape = 19, alpha = 0.5, stroke = 1) +
  geom_smooth(method = loess, se = FALSE) +
  geom_rug()
```



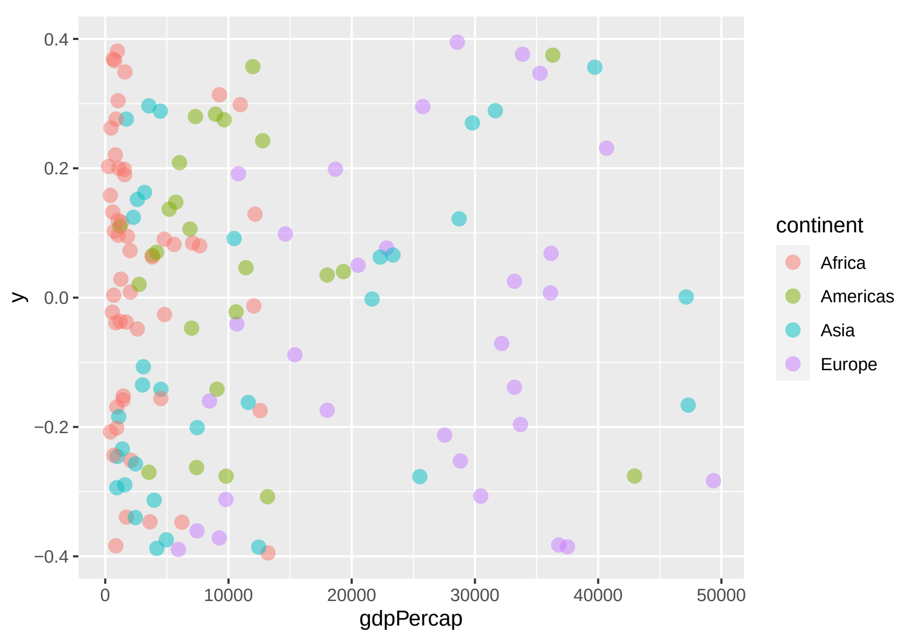
6.11 Position adjustment

Position adjustments determine how to arrange geoms that would otherwise occupy the same space.

```
ggplot() +
  geom_point(data = gapminder_2007, aes(y = 0, x = gdpPercap, colour = continent),
             alpha = 0.5, size = 3)
```



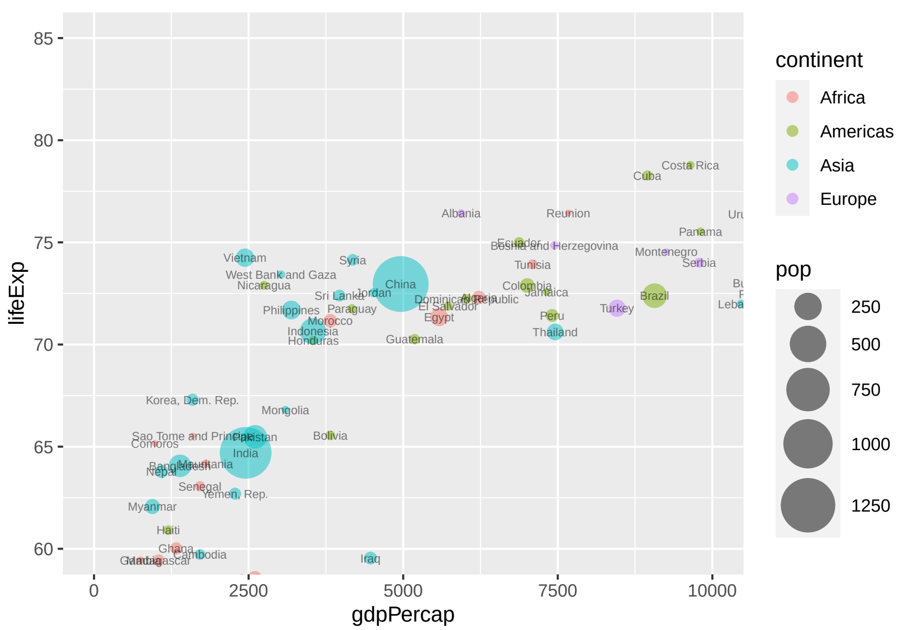

```
# changing the position to jitter
ggplot() +
  geom_point(data = gapminder_2007, aes(y = 0, x = gdpPercap, colour = continent),
             alpha = 0.5, size = 3, position = "jitter")
```



6.12 Coordinate system

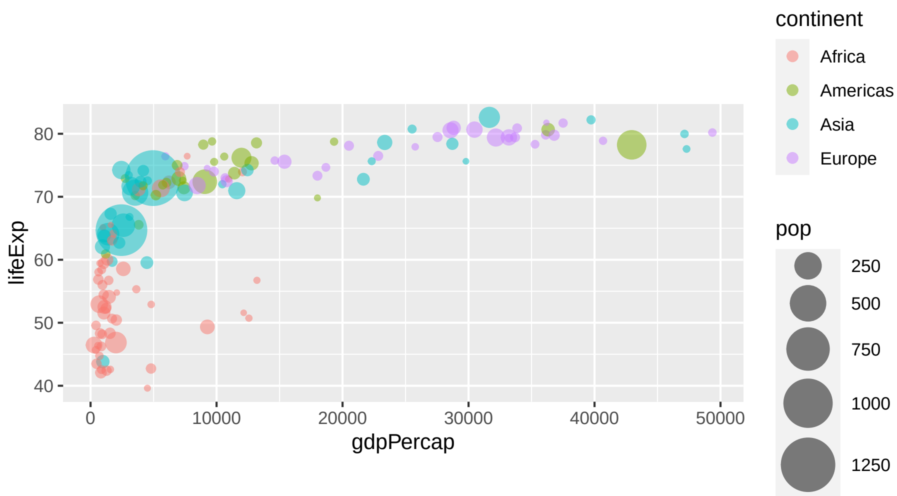
The function `coord_cartesian()` zooms a plot. It expects ylim and/or xlim arguments.

```
ggplot(data = gapminder_2007, aes(y = lifeExp, x = gdpPercap)) +
  geom_point(alpha = 0.5, stroke = 1, aes(size = pop, colour = continent)) +
  scale_size_area(max_size = 12) +
  geom_text(aes(label = country), size = 2, alpha = 0.5) +
  coord_cartesian(ylim = c(60, 85), xlim = c(0, 10000))
```



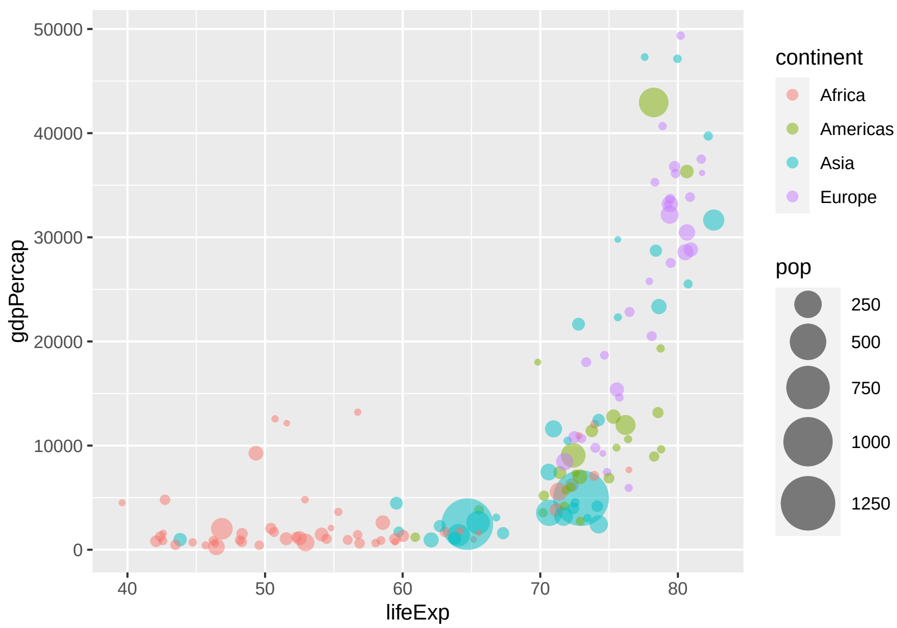
The function `coord_fixed()` controls the aspect ratio. It expects a ratio of y/x.

```
ggplot(data = gapminder_2007, aes(y = lifeExp, x = gdpPercap)) +
  geom_point(alpha = 0.5, stroke = 1, aes(size = pop, colour = continent)) +
  scale_size_area(max_size = 12) +
  coord_fixed(ratio = 500)
```



The function `coord_flip()` flips a plot along its diagonal.

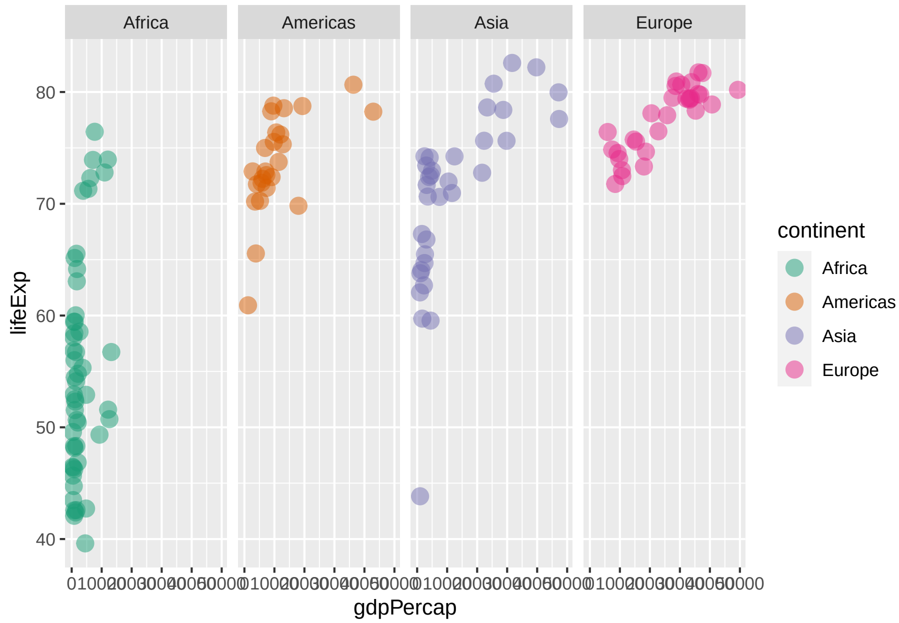
```
ggplot(data = gapminder_2007, aes(y = lifeExp, x = gdpPercap)) +
  geom_point(alpha = 0.5, stroke = 1, aes(size = pop, colour = continent)) +
  scale_size_area(max_size = 12) +
  coord_flip()
```



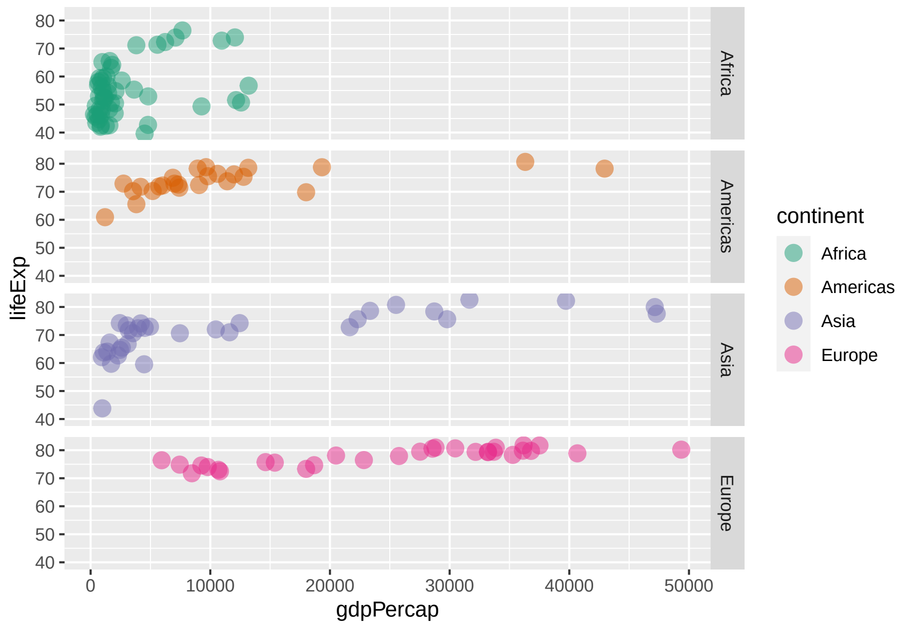
6.13 Faceting layer

The functions `facet_grid()` and `facet_wrap()` controls faceting. The former forms a matrix of panels defined by row and column faceting variables while the later wraps a 1d sequence of panels into 2d.

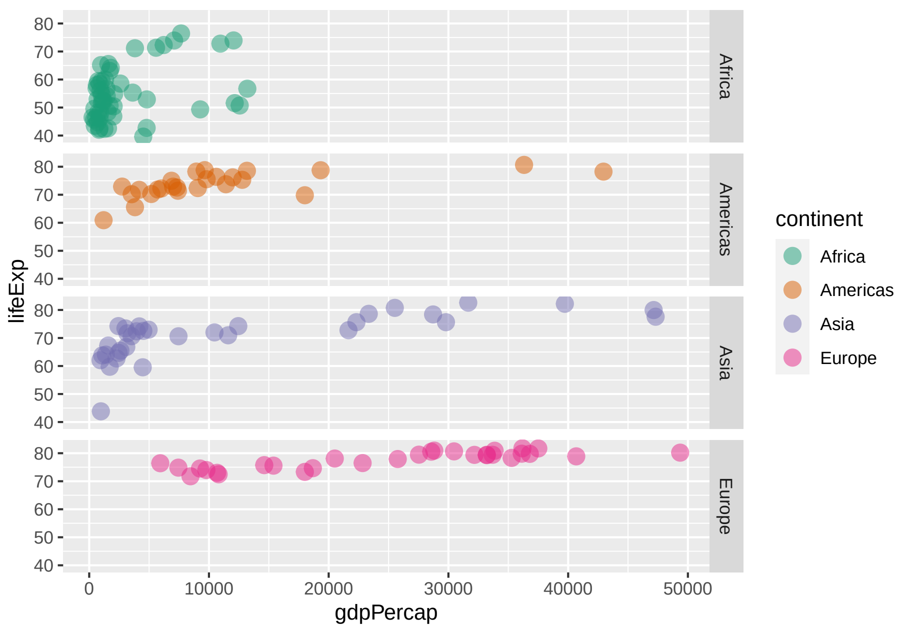
```
ggplot(data = gapminder_2007, aes(y = lifeExp, x = gdpPercap, colour = continent)) +
  geom_point(size = 3, shape = 19, alpha = 0.5, stroke = 1) +
  scale_colour_brewer(palette = "Dark2") +
  facet_grid(.~ continent)
```



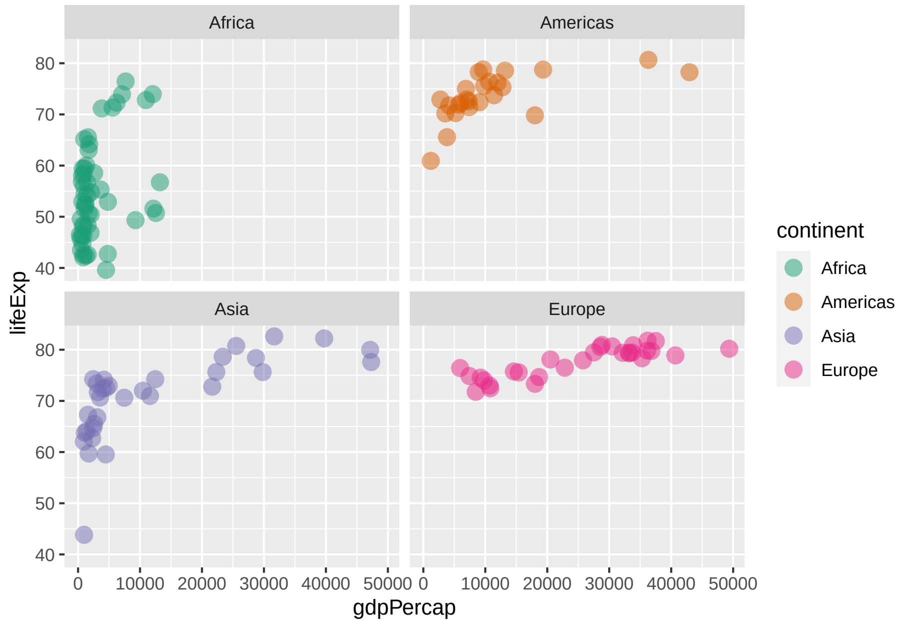

```
ggplot(data = gapminder_2007, aes(y = lifeExp, x = gdpPercap, colour = continent)) +
  geom_point(size = 3, shape = 19, alpha = 0.5, stroke = 1) +
  scale_colour_brewer(palette = "Dark2") +
  facet_grid(continent ~ .)
```




```
ggplot(data = gapminder_2007, aes(y = lifeExp, x = gdpPercap, colour = continent)) +
  geom_point(size = 3, shape = 19, alpha = 0.5, stroke = 1) +
  scale_colour_brewer(palette = "Dark2") +
  facet_grid(continent ~ ., )
```

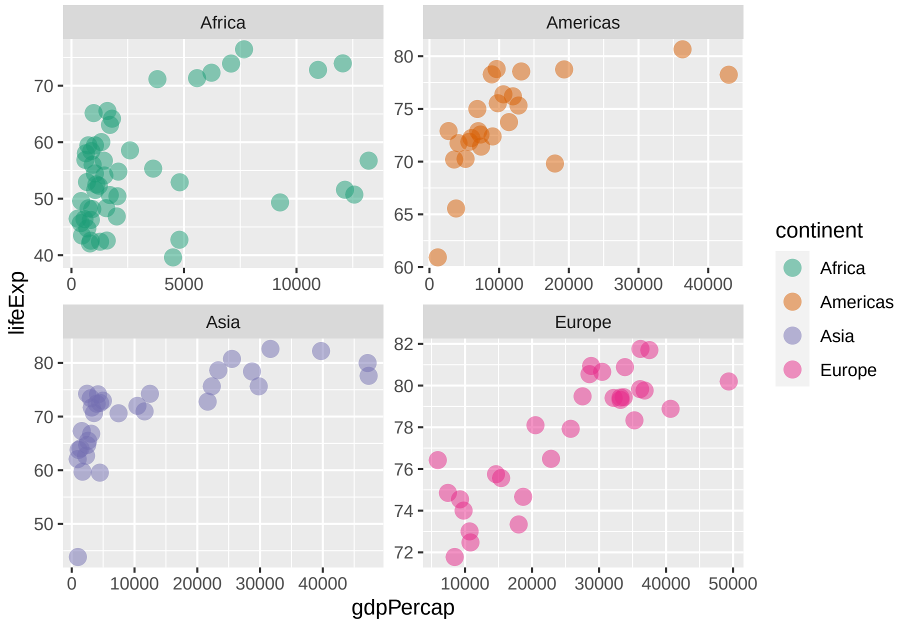



```
ggplot(data = gapminder_2007, aes(y = lifeExp, x = gdpPercap, colour = continent)) +
  geom_point(size = 3, shape = 19, alpha = 0.5, stroke = 1) +
  scale_colour_brewer(palette = "Dark2") +
  facet_wrap(continent ~ ., )
```



By default, all axis have the same scale, using the argument scales = 'free' we can render the scales for each plot independent.

```
# independent axis
ggplot(data = gapminder_2007, aes(y = lifeExp, x = gdpPercap, colour = continent)) +
  geom_point(size = 3, shape = 19, alpha = 0.5, stroke = 1) +
  scale_colour_brewer(palette = "Dark2") +
  facet_wrap(continent ~ ., scales = 'free')
```

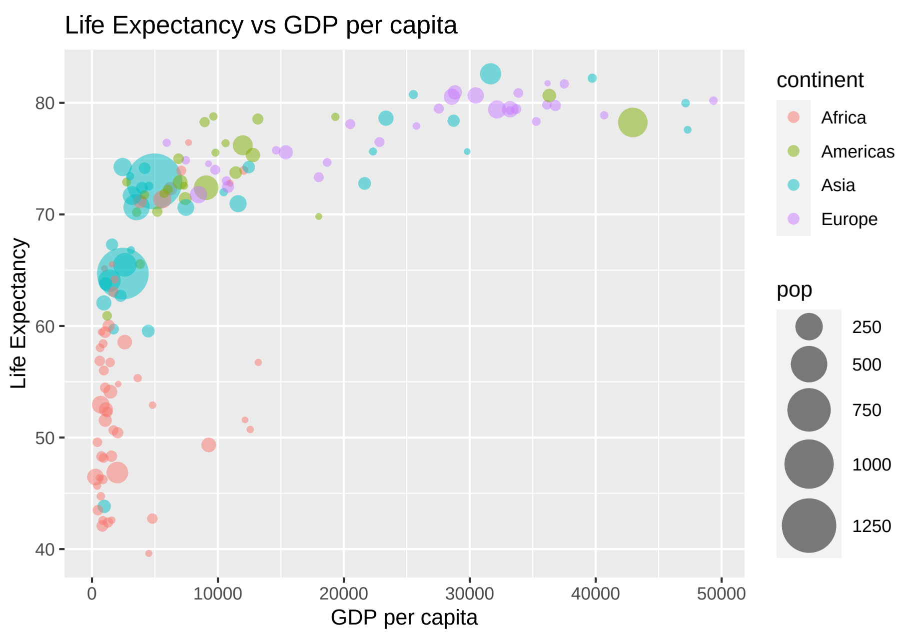


6.14 Plot elements

6.14.1 Title, captions and labels

The function `labs()` is used to add title and labels.

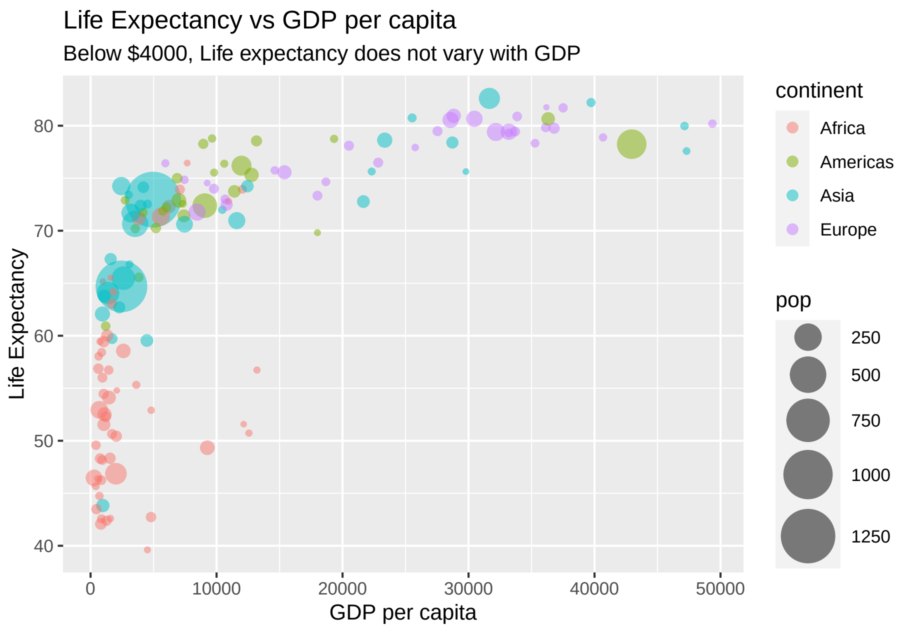
```
ggplot(data = gapminder_2007, aes(y = lifeExp, x = gdpPercap)) +
  geom_point(alpha = 0.5, stroke = 1, aes(size = pop, colour = continent)) +
  scale_size_area(max_size = 12) +
  labs(y = 'Life Expectancy', x = 'GDP per capita', title = 'Life Expectancy vs GDP per capita')
```



The function:

- `ggtitle()` adds title to a plot
- `xlab()` adds x-axis label
- `ylab()` adds y-axis label
- `labs()` adds all of the above

```
ggplot(data = gapminder_2007, aes(y = lifeExp, x = gdpPercap)) +
  geom_point(alpha = 0.5, stroke = 1, aes(size = pop, colour = continent)) +
  scale_size_area(max_size = 12) +
  ggtitle('Life Expectancy vs GDP per capita',
          subtitle = "Below $4000, Life expectancy does not vary with GDP") +
  ylab('Life Expectancy') +
  xlab('GDP per capita')
```

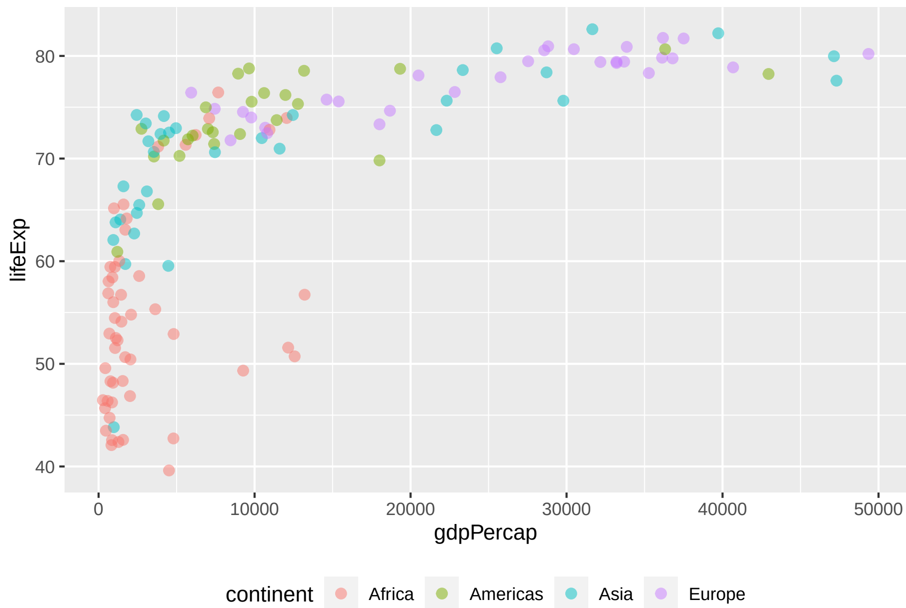


6.14.2 Legend

The function `theme()` is used to customize the non-data components of a plot. We shall use it to customize legends.

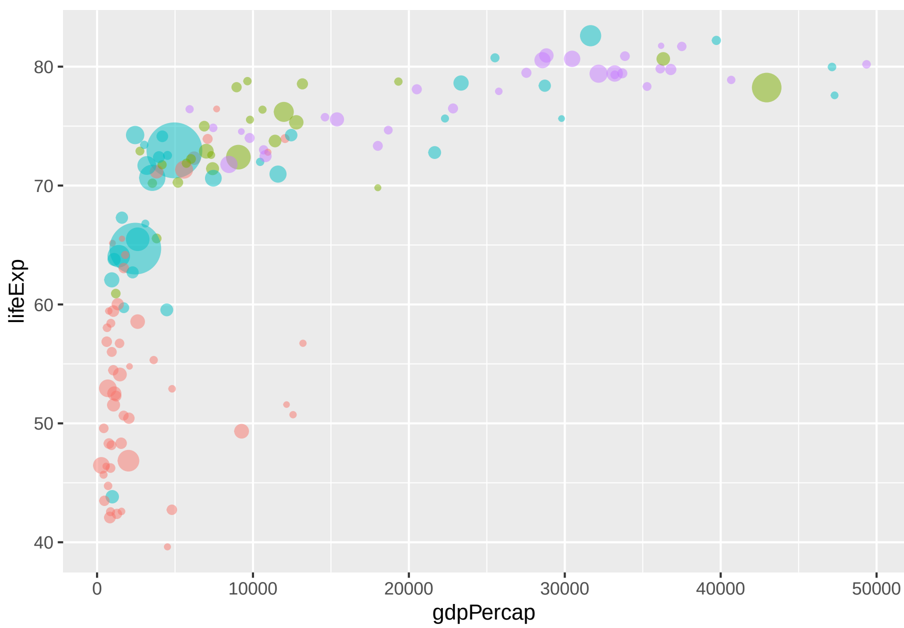
Legend position The argument legend.position determines the position of the legend. It accepts 'bottom', 'left', 'top' and 'right'.

```
# position legend at the bottom
ggplot(data = gapminder_2007, aes(y = lifeExp, x = gdpPercap)) +
  geom_point(alpha = 0.5, stroke = 1, aes(colour = continent)) +
  theme(legend.position = "bottom")
```



Removing legends using theme() The argument legend.position = “none” removes all the legends in a plot.

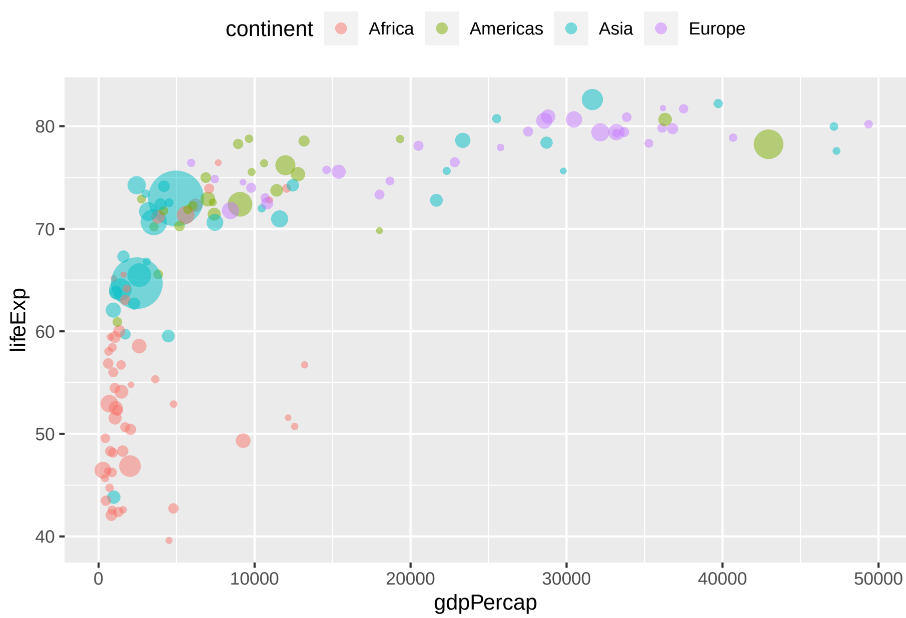
```
# removing legend
ggplot(data = gapminder_2007, aes(y = lifeExp, x = gdpPercap)) +
  geom_point(alpha = 0.5, stroke = 1, aes(size = pop, colour = continent)) +
  scale_size_area(max_size = 12) +
  theme(legend.position = "none")
```



6.14.3 Removing legends using guides()

The function guides() removes legends by a specific scale. The legend of each scale can be removed by passing either 'none' or FALSE to it.

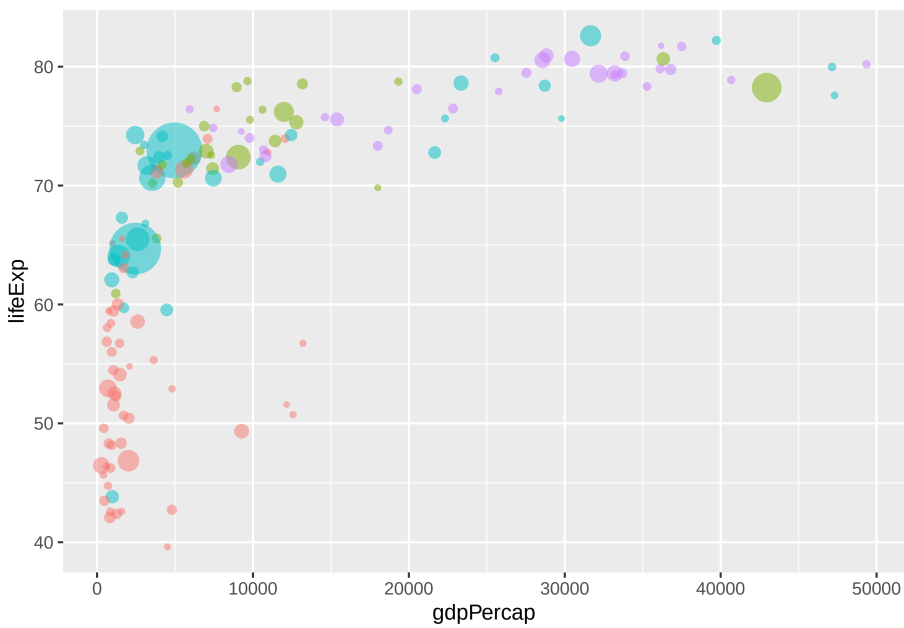
```
# removing the size legend
ggplot(data = gapminder_2007, aes(y = lifeExp, x = gdpPercap)) +
  geom_point(alpha = 0.5, stroke = 1, aes(size = pop, colour = continent)) +
  scale_size_area(max_size = 12) +
  theme(legend.position = "top") +
  guides(size = FALSE)
```



6.14.4 Removing legend using geom

The argument `show.legend = F` within a geom, removes the legend of that geom.

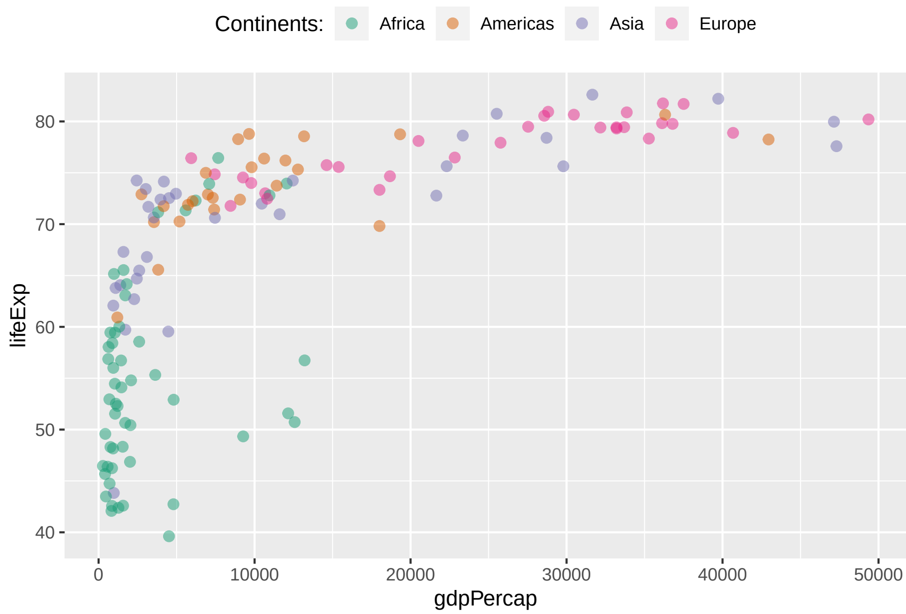
```
ggplot(data = gapminder_2007, aes(y = lifeExp, x = gdpPercap)) +
  geom_point(alpha = 0.5, stroke = 1, aes(size = pop, colour = continent), show.legend
  scale_size_area(max_size = 12) +
  theme(legend.position = "top")
```



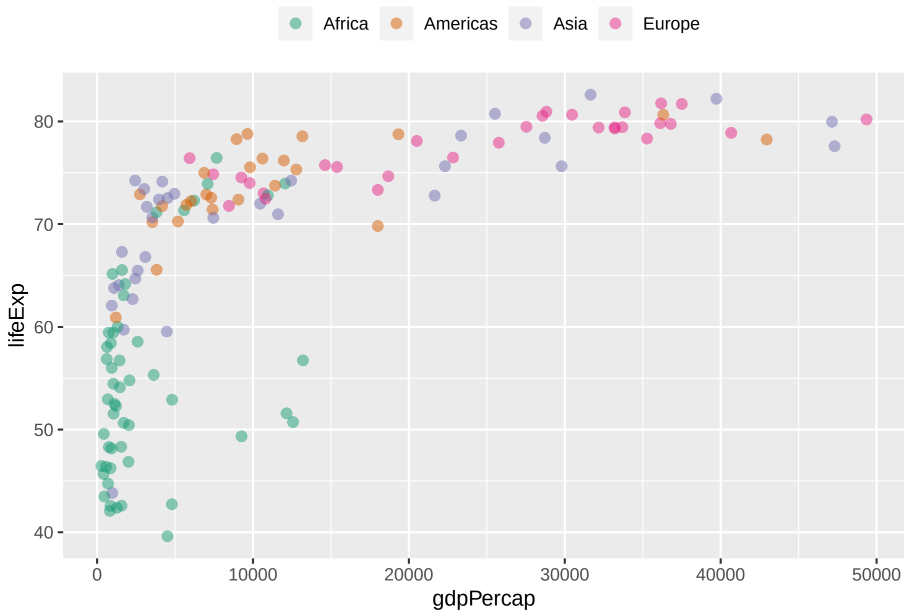
6.14.4.1 Legend title

The argument name within `scale_*` is used to control the legend title.

```
# renaming legend
ggplot(data = gapminder_2007, aes(y = lifeExp, x = gdpPercap)) +
  geom_point(alpha = 0.5, stroke = 1, aes(colour = continent)) +
  scale_colour_brewer(palette = "Dark2", name = 'Continents:') +
  theme(legend.position = "top")
```



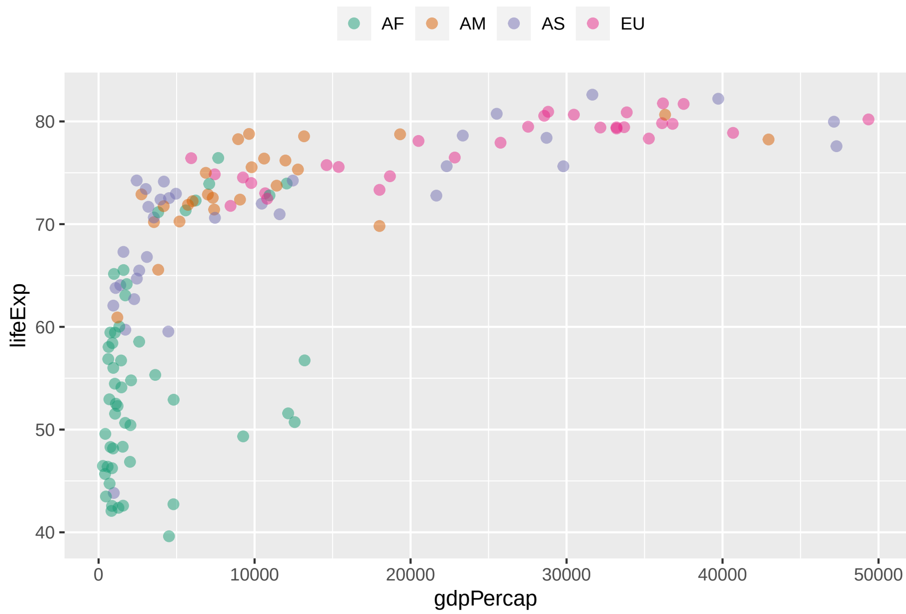

```
# drop legend title
ggplot(data = gapminder_2007, aes(y = lifeExp, x = gdpPercap)) +
  geom_point(alpha = 0.5, stroke = 1, aes(colour = continent)) +
  scale_colour_brewer(palette = "Dark2", name = '') +
  theme(legend.position = "top")
```



####Changing legend labels

The argument label within `scale_*` is used to change legend labels.

```
ggplot(data = gapminder_2007, aes(y = lifeExp, x = gdpPercap)) +
  geom_point(alpha = 0.5, stroke = 1, aes(colour = continent)) +
  scale_colour_brewer(palette = "Dark2", name = '', label = c('AF', 'AM', 'AS', 'EU', 'OC')) +
  theme(legend.position = "top")
```

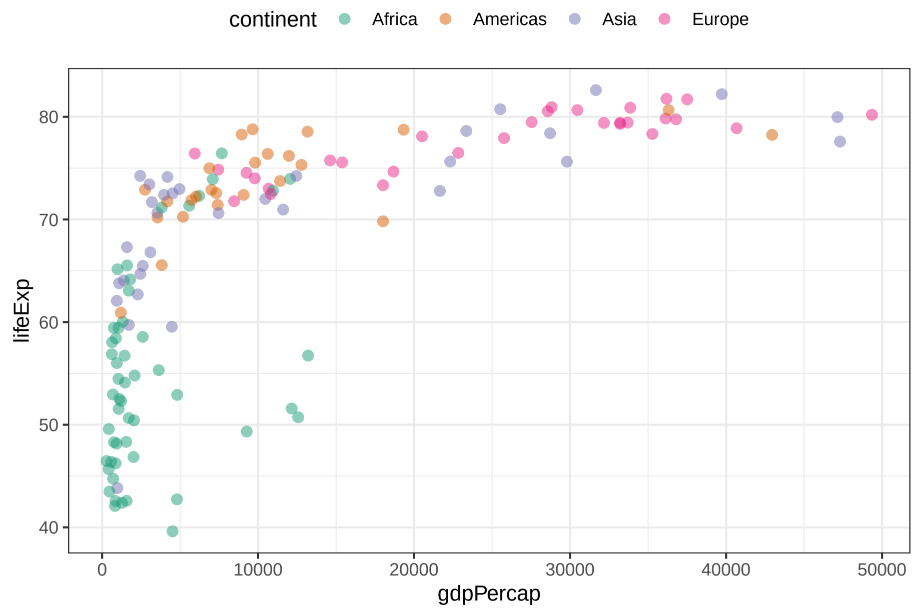


6.14.5 Built-in themes

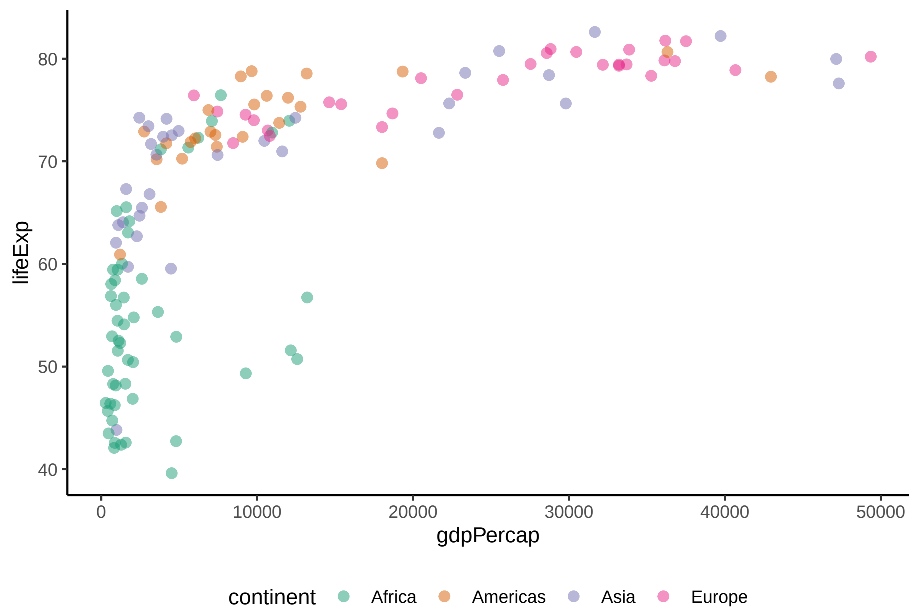
ggplot2 comes with some built-in themes for customizing plots. These includes:

theme_grey() theme_bw() theme_linedraw() theme_light() theme_dark() theme_minimal() theme_classic() theme_void() theme_test()

```
ggplot(data = gapminder_2007, aes(y = lifeExp, x = gdpPercap)) +
  geom_point(alpha = 0.5, stroke = 1, aes(colour = continent)) +
  scale_colour_brewer(palette = "Dark2") +
  theme_bw() +
  theme(legend.position = "top")
```




```
ggplot(data = gapminder_2007, aes(y = lifeExp, x = gdpPercap)) +
  geom_point(alpha = 0.5, stroke = 1, aes(colour = continent)) +
  scale_colour_brewer(palette = "Dark2") +
  theme_bw() +
  theme_classic() +
  theme(legend.position = "bottom")
```



6.15 Saving plots

There are two ways of saving plots in ggplot2 which are using:

- graphic devices
- `ggsave()`

6.15.1 Saving plots using graphic devices

With this method, we must first open the graphic device using any of the following rendering functions:

- `pdf()`
- `svg()`
- `png()`
- `jpeg()`
- `tiff()`
- `bmp()`

Then we produce the plot and finally, we close the device using dev.off().

```
# preparing plot
plt <-
ggplot(data = gapminder_2007, aes(y = lifeExp, x = gdpPercap)) +
  geom_point(alpha = 0.5, stroke = 1, aes(size = pop, colour = continent)) +
  scale_size_area(max_size = 12) +
```

```
  theme(legend.position = "top") +
  guides(size = FALSE)

# initiating device
pdf('world.pdf', width = 8, height = 8)

# saving plot
print(plt)

# closing device
dev.off()
#> pdf
#>   2

# initiating device
png('world.png', width = 800, height = 600)

# saving plot
print(plt)

# closing device
dev.off()
#> pdf
#>   2

# checking files
file.exists(c('world.pdf', 'world.png'))
#> [1] TRUE TRUE

# removing files
file.remove(c('world.pdf', 'world.png'))
#> [1] TRUE TRUE
```

6.15.2 Saving plots using ggsave()

The function ggsave() saves a plot directly to disc.

```
ggsave('world.pdf', plt, width = 16, height = 16, units = 'cm')
ggsave('world.png', plt, width = 8, height = 8, units = 'cm')

# checking files
file.exists(c('world.pdf', 'world.png'))
#> [1] TRUE TRUE

# removing files
```

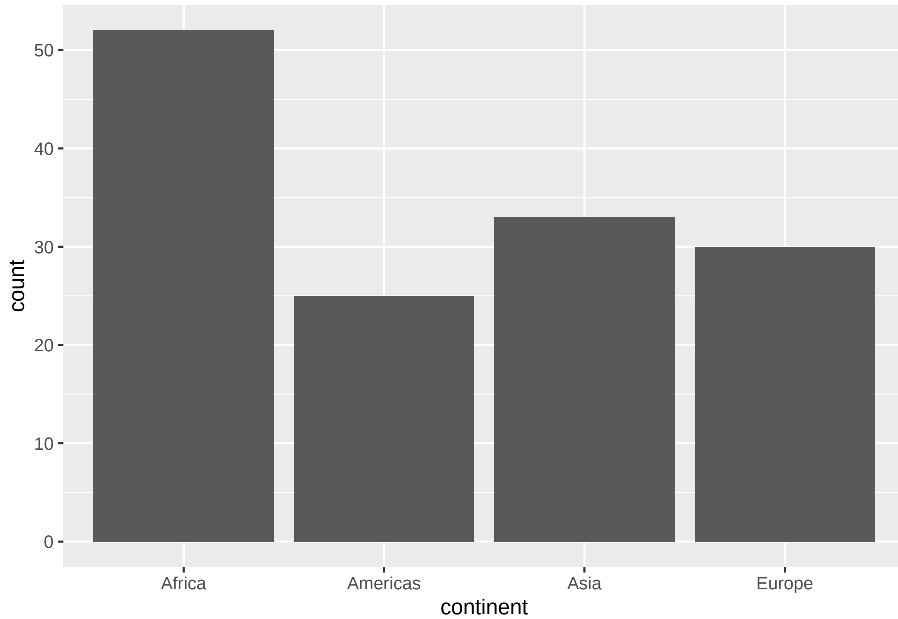
```
file.remove(c('world.pdf', 'world.png'))
#> [1] TRUE TRUE
```

6.16 Statistical plots with ggplot2

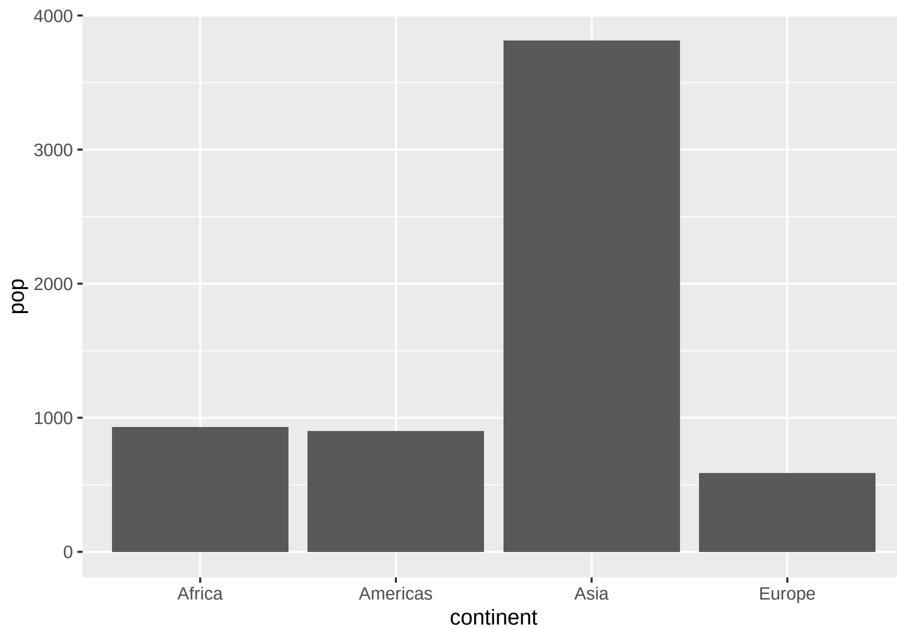
6.16.1 Bar and column chart

The functions `geom_bar()` and `geom_col()` are used to create bar charts. While the former works on a categorical column, returning a bar for the count of each category, the later requires a numeric column for the y-axis and category names for the x-axis.

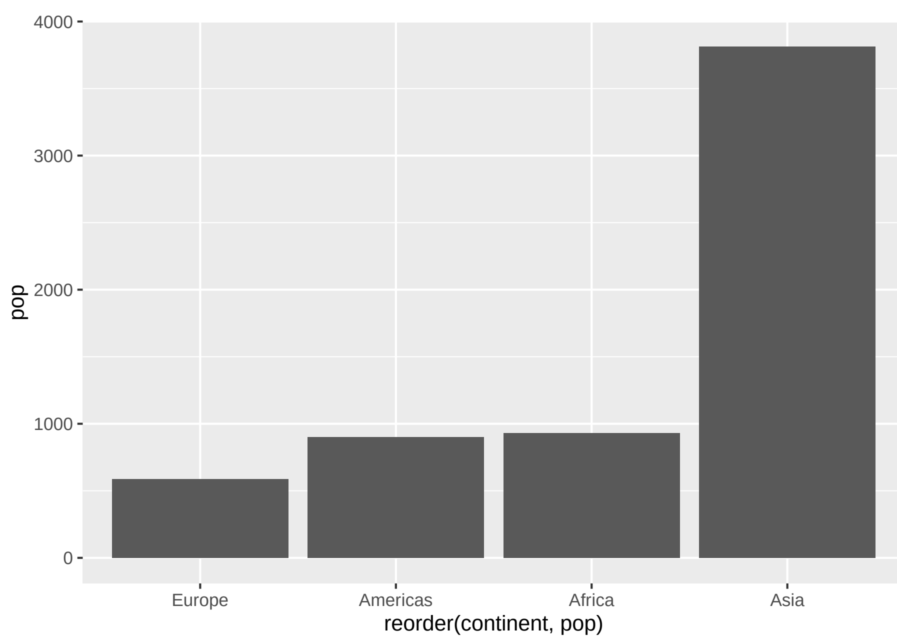
```
library(ggplot2)
library(dplyr)
library(gapminder)
library(RColorBrewer)

gapminder_2007 <-
gapminder %>%
filter(year == '2007' & continent != 'Oceania') %>%
mutate(pop = round(pop/1e6, 1)) %>%
select(-year)
head(gapminder_2007)
#> # A tibble: 6 x 5
#>   country     continent lifeExp   pop gdpPercap
#>   <fct>       <fct>       <dbl> <dbl>     <dbl>
#> 1 Afghanistan Asia         43.8  31.9      975.
#> 2 Albania     Europe       76.4   3.6     5937.
#> 3 Algeria     Africa       72.3  33.3     6223.
#> 4 Angola      Africa       42.7  12.4     4797.
#> 5 Argentina   Americas     75.3  40.3    12779.
#> 6 Austria     Europe       79.8   8.2    36126.

# count of countries by continent
ggplot(gapminder_2007, aes(x = continent)) +
   geom_bar()
```



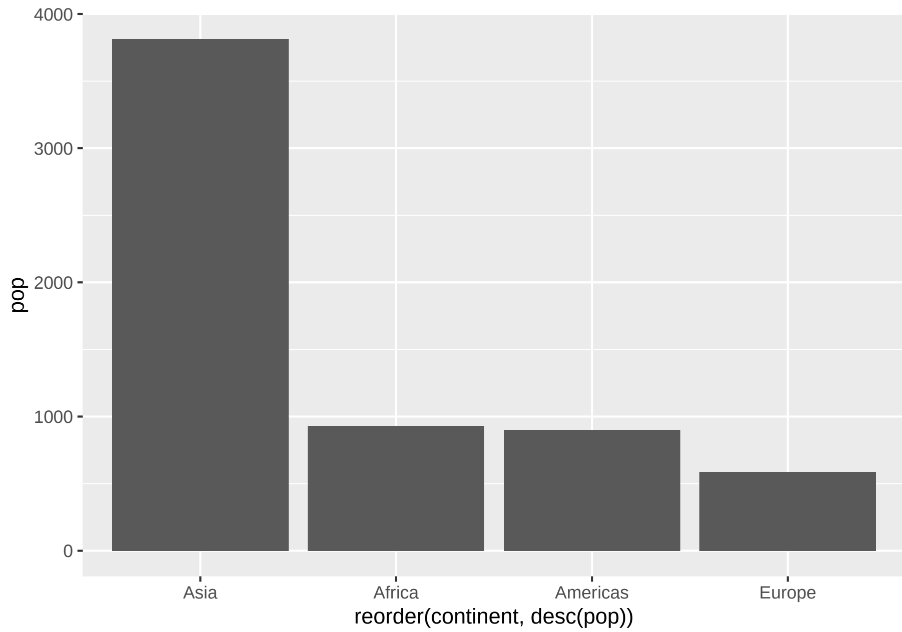

```
# preparing data
pop_2007 <-
gapminder_2007 %>%
group_by(continent) %>%
summarise(pop = sum(pop, na.rm = T))
pop_2007
#> # A tibble: 4 x 2
#>   continent   pop
#>   <fct>     <dbl>
#> 1 Africa     930.
#> 2 Americas   899.
#> 3 Asia      3812.
#> 4 Europe     586.

# population by continent
pop_2007 %>%
ggplot(aes(x = continent, y = pop)) +
   geom_col()
```




```
# sorting columns ascending
ggplot(pop_2007, aes(x = reorder(continent, pop), y = pop)) +
  geom_col()
```



```
# sorting columns descending
ggplot(pop_2007, aes(x = reorder(continent, desc(pop)), y = pop)) +
  geom_col()
```

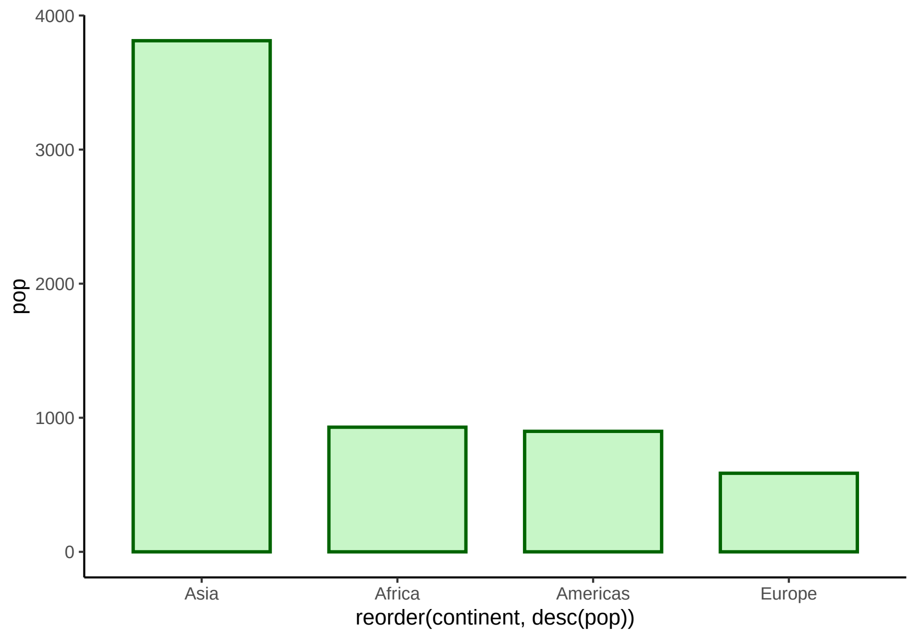


6.16.1.1 Borders and colours

The argument:

- `fill=`: fills bars
- `colour=`: colours borders
- `size=`: controls border size
- `width=`: controls bar width

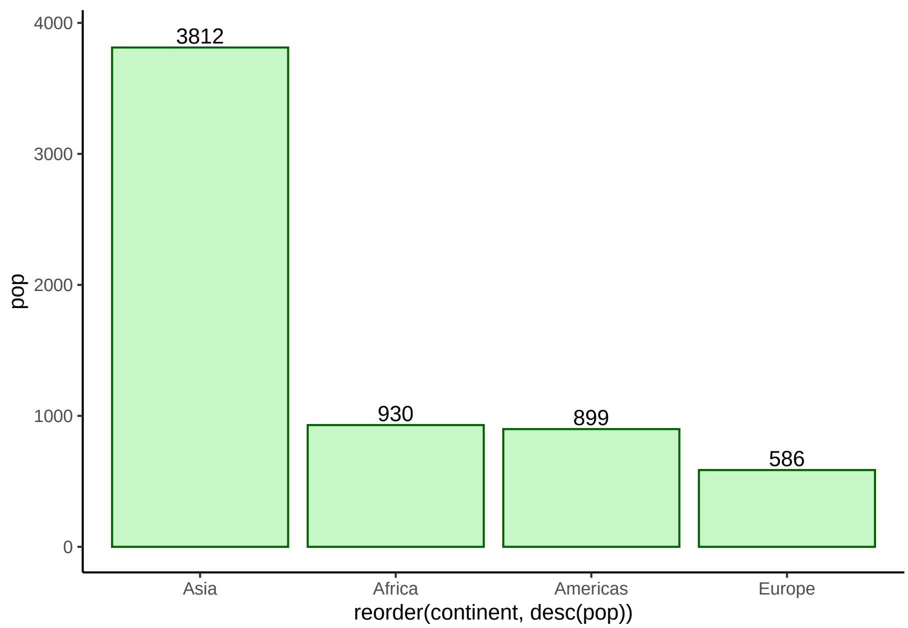
```
ggplot(pop_2007, aes(x = reorder(continent, desc(pop)), y = pop)) +
  geom_col(fill = 'lightgreen', colour = 'darkgreen', alpha = 0.5, size = 0.8, width = 0.7) +
  theme_classic()
```



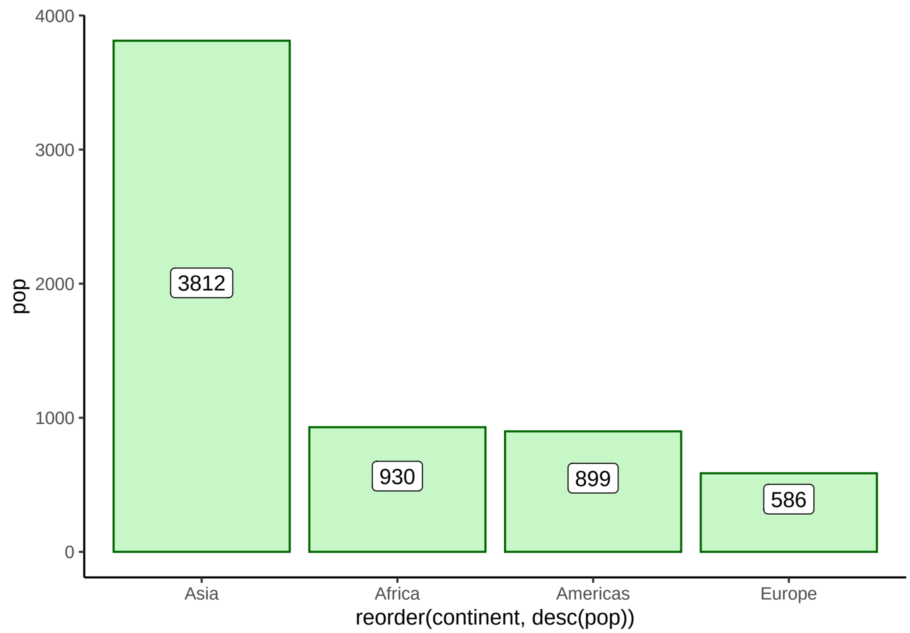
6.16.1.2 Adding labels

The functions `geom_text()` and `geom_label()` are used to add data labels.

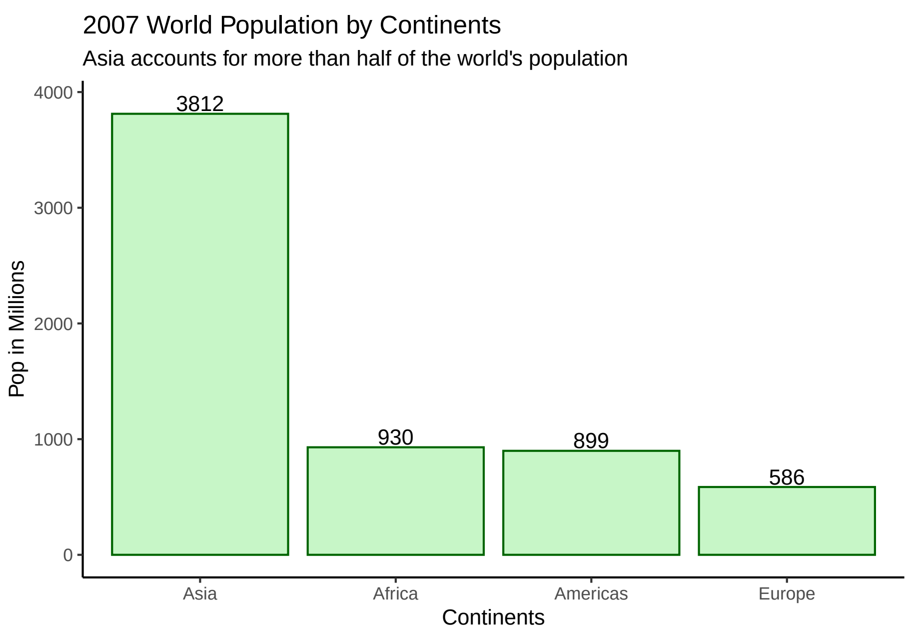
```
ggplot(data = pop_2007, aes(x = reorder(continent, desc(pop)), y = pop)) +
   geom_col(fill = 'lightgreen', colour = 'darkgreen', alpha = 0.5) +
   geom_text(aes(label = round(pop)), nudge_y = 90) +
   theme_classic()
```




```
# placing label at centre of bars
ggplot(data = pop_2007) +
geom_col(aes(x = reorder(continent, desc(pop)), y = pop),
         fill = 'lightgreen', colour = 'darkgreen', alpha = 0.5) +
   geom_label(aes(x = reorder(continent, desc(pop)),
                  y = pop/2, label = round(pop)), nudge_y = 100) +
   theme_classic()
```



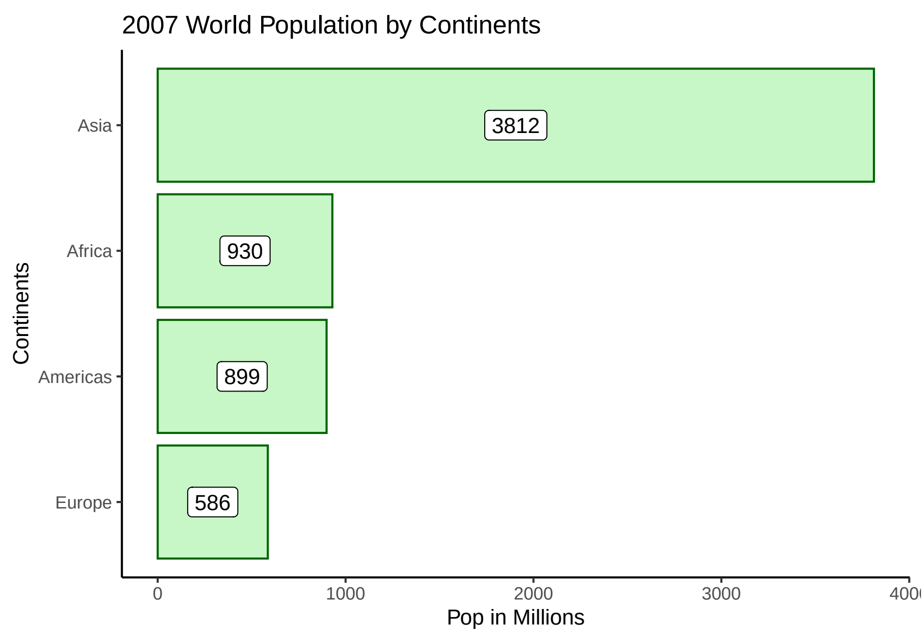

```
ggplot(pop_2007, aes(x = reorder(continent, desc(pop)), y = pop)) +
   geom_col(fill = 'lightgreen', colour = 'darkgreen', alpha = 0.5) +
   geom_text(aes(label = round(pop)), nudge_y = 90) +
   ggtitle('2007 World Population by Continents',
           subtitle = "Asia accounts for more than half of the world's population") +
   xlab('Continents') +
   ylab('Pop in Millions') +
   theme_classic()
```



6.16.1.4 Column chart

Using the function `coord_flip()`, we can flip a bar chart into a column chart.

```
# producing a column chart
ggplot(pop_2007, aes(x = reorder(continent, pop), y = pop)) +
   geom_col(fill = 'lightgreen', colour = 'darkgreen', alpha = 0.5) +
   labs(x = 'Continents',y = 'Pop in Millions',title = '2007 World Population by Continents') +
   geom_label(aes(label = round(pop), y = pop/2)) +
   theme_classic() +
   coord_flip()
```

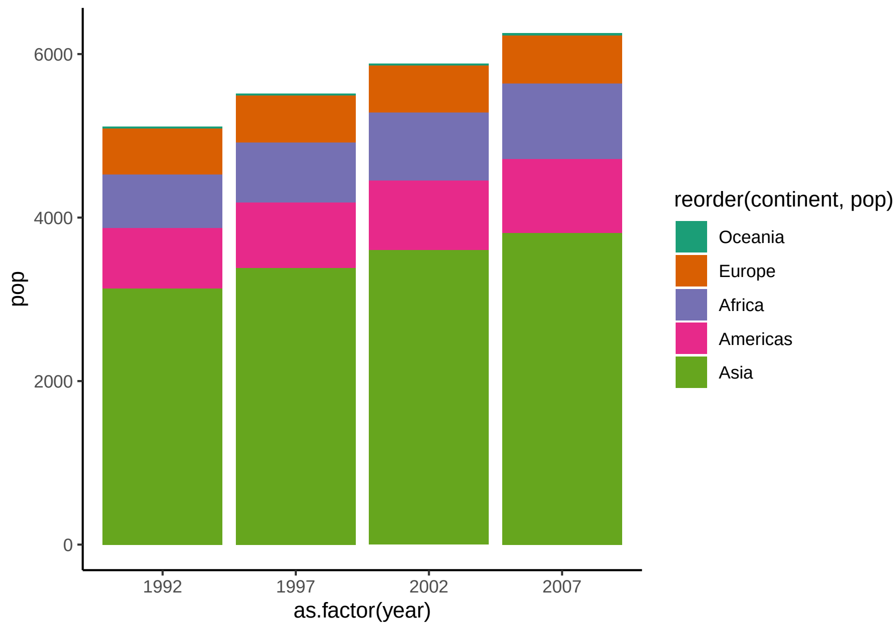


6.16.1.5 Stacked bar chart

To create stacked column bars, we use the fill argument by mapping it to a continuous variable.

```
# preparing data
dt <-
gapminder %>%
filter(year >= 1992) %>%
group_by(year, continent) %>%
summarise(pop = round(sum(pop/1e6, na.rm = T)))
head(dt)
#> # A tibble: 6 x 3
#> # Groups:   year [2]
#>    year continent   pop
#>   <int> <fct>     <dbl>
#> 1  1992 Africa      659
#> 2  1992 Americas    739
#> 3  1992 Asia       3133
#> 4  1992 Europe      558
#> 5  1992 Oceania      21
#> 6  1997 Africa      744

# producing a stacked bar chart
ggplot(dt, aes(x = as.factor(year), y = pop, fill = reorder(continent, pop))) +
```

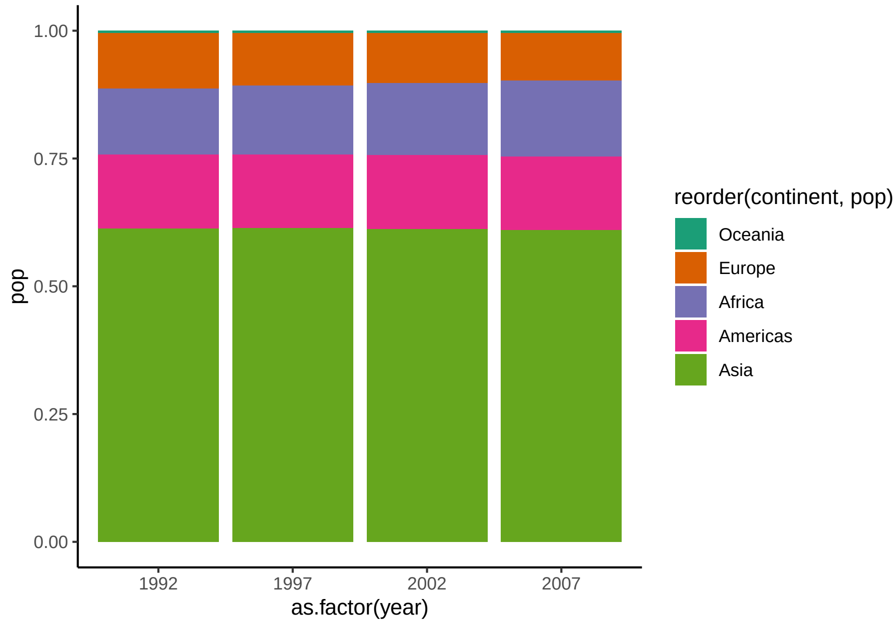
```
  geom_col() +
  theme_classic() +
  scale_fill_brewer(palette = "Dark2")
```



6.16.1.6 The 100% stacked bar chart

To create a 100% stacked bar chart, we set `position = "fill"` inside `geom_col()`.

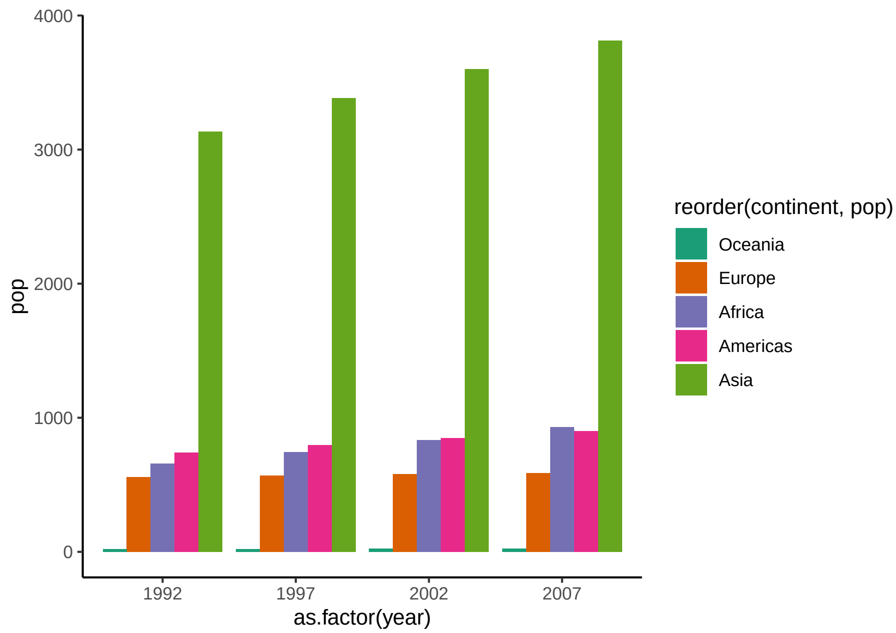
```
ggplot(dt, aes(x = as.factor(year), y = pop, fill = reorder(continent, pop))) +
  geom_col(position = "fill") +
  theme_classic() +
  scale_fill_brewer(palette = "Dark2")
```



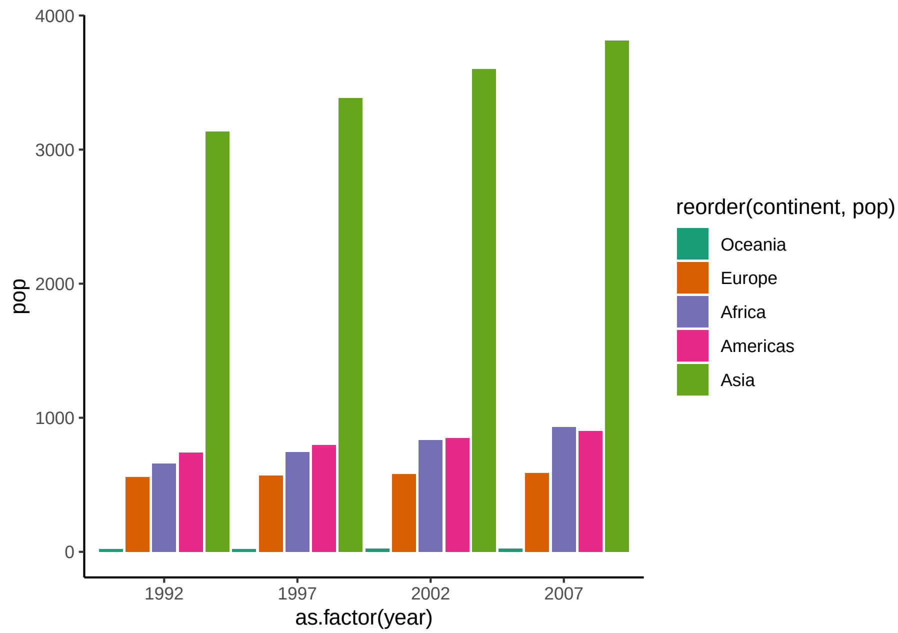
6.16.1.7 Clustered bar chart

To create a clustered bar chart, we set `position = "dodge"` inside `geom_col()`.

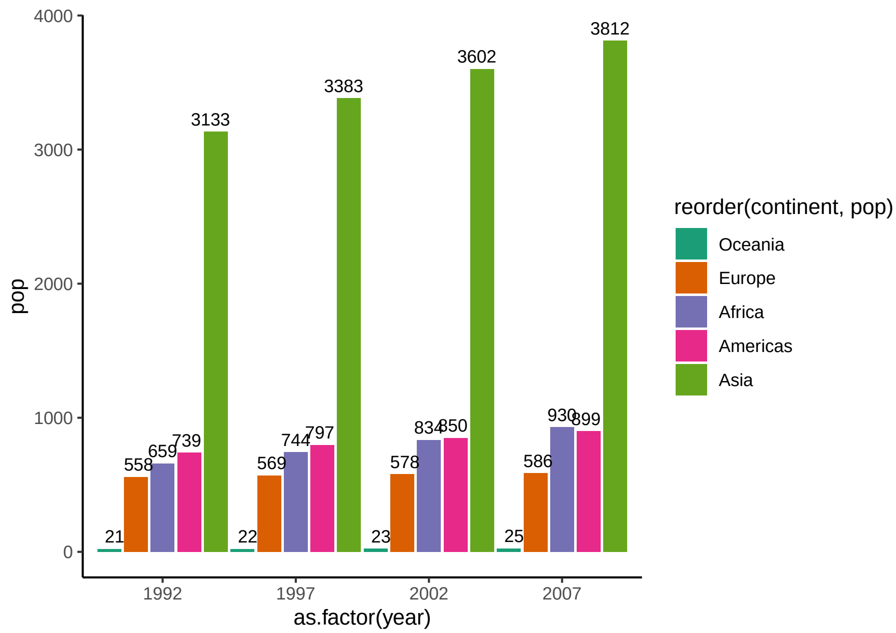
```
ggplot(dt, aes(x = as.factor(year), y = pop, fill = reorder(continent, pop))) +
   geom_col(position = "dodge") +
   theme_classic() +
   scale_fill_brewer(palette = "Dark2")
```




```
# adding space between bars
ggplot(dt, aes(x = as.factor(year), y = pop, fill = reorder(continent, pop))) +
   geom_col(position = position_dodge(width = 1)) +
   theme_classic() +
   scale_fill_brewer(palette = "Dark2")
```



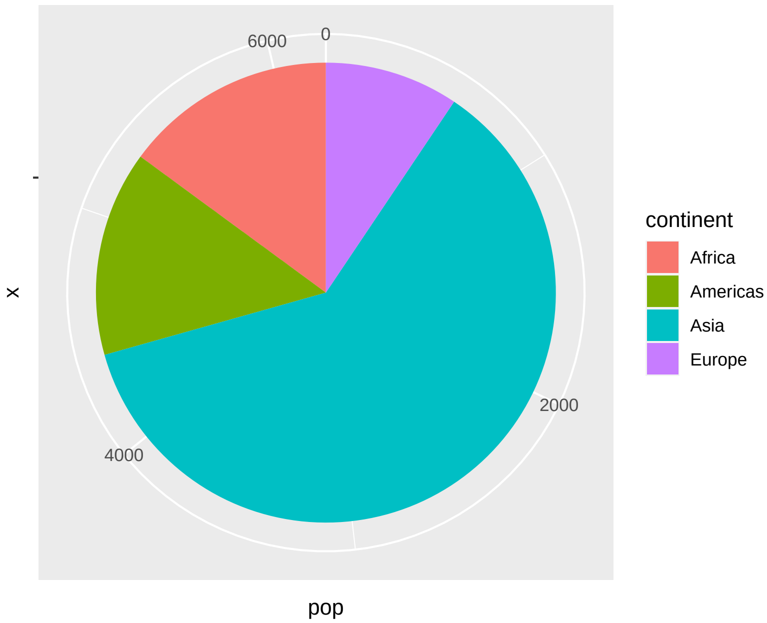

```r
# adding data labels
ggplot(dt, aes(x = as.factor(year), y = pop, fill = reorder(continent, pop))) +
   geom_col(position = position_dodge(width = 1)) +
   theme_classic() +
   scale_fill_brewer(palette = "Dark2") +
   geom_text(aes(label = round(pop), y = pop), position = position_dodge(0.9),
             size = 3, vjust = -0.5, hjust = 0.5)
```



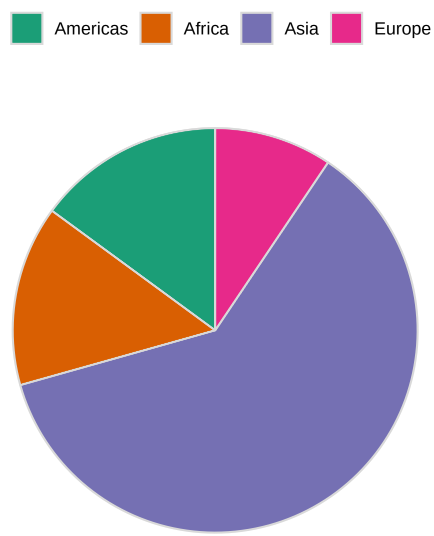
6.16.2 Pie chart

There is no geom for producing pie charts but by using coord_polar(), we can produce pie charts.

```
# data
pop_2007
#> # A tibble: 4 x 2
#>   continent   pop
#>   <fct>     <dbl>
#> 1 Africa     930.
#> 2 Americas   899.
#> 3 Asia      3812.
#> 4 Europe     586.

ggplot(pop_2007, aes(y = pop, x = '', fill = continent)) +
  geom_col() +
  coord_polar("y", start = 0)
```



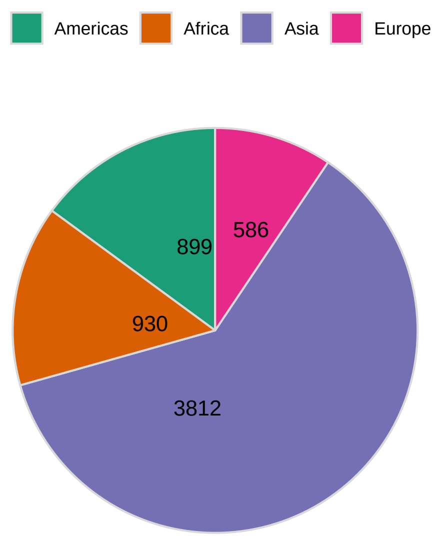

```
ggplot(pop_2007, aes(y = pop, x = '', fill = continent)) +
   geom_col(colour = grey(0.85), size = 0.5) +
   coord_polar("y", start = 0) +
   scale_fill_brewer(palette = "Dark2", label = c('Americas', 'Africa', 'Asia', 'Europe
   theme_minimal() +
   labs(x = '', y = '') +
   theme(legend.position = "top",
         axis.ticks = element_blank(),
         panel.grid=element_blank(),
         axis.text.x=element_blank(),
         legend.title = element_blank()
)
```



6.16.2.2 Adding data labels

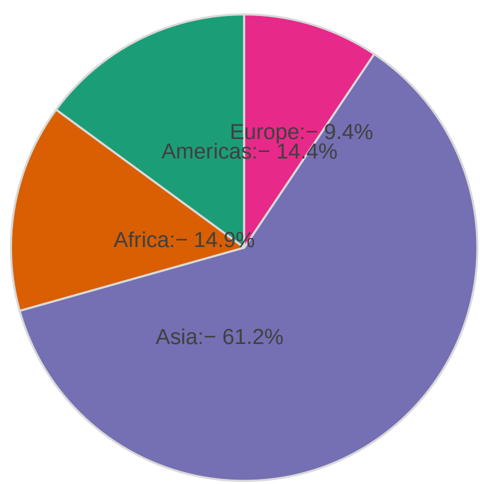
```
# preparing label
pop_2007 %>%
arrange(desc(pop)) %>%
mutate(label_y = cumsum(pop))
#> # A tibble: 4 x 3
#>   continent   pop label_y
#>   <fct>     <dbl>   <dbl>
#> 1 Asia      3812.   3812.
#> 2 Africa     930.   4742.
#> 3 Americas   899.   5640.
#> 4 Europe     586.   6227.

pop_2007 %>%
arrange(desc(pop)) %>%
mutate(label_y = cumsum(pop)) %>%

ggplot(aes(y = pop, x = '', fill = continent)) +
   geom_col(colour = grey(0.85), size = 0.5) +
   coord_polar("y", start = 0) +
   scale_fill_brewer(palette = "Dark2", label = c('Americas', 'Africa', 'Asia', 'Europe')) +
   theme_minimal() +
   labs(x = '', y = '') +
```

```
  theme(legend.position = "top",
        axis.ticks = element_blank(),
        panel.grid=element_blank(),
        axis.text.x=element_blank(),
        legend.title = element_blank()) +
  geom_text(aes(y = label_y, label = round(pop)), hjust = -0.5)
```




```
# preparing data
pop_2007 %>%
arrange(desc(pop)) %>%
mutate(label_y = cumsum(pop)) %>%
mutate(label_per = round(pop/sum(pop),3))
#> # A tibble: 4 x 4
#>   continent   pop label_y label_per
#>   <fct>     <dbl>   <dbl>     <dbl>
#> 1 Asia      3812.   3812.     0.612
#> 2 Africa     930.   4742.     0.149
#> 3 Americas   899.   5640.     0.144
#> 4 Europe     586.   6227.     0.094

pop_2007 %>%
arrange(desc(pop)) %>%
mutate(label_y = cumsum(pop)) %>%
mutate(label_per = round(pop/sum(pop),3)) %>%
```

```
ggplot(aes(y = pop, x = '', fill = continent)) +
   geom_col(colour = grey(0.85), size = 0.5) +
   coord_polar("y", start = 0) +
   scale_fill_brewer(palette = "Dark2") +
   theme_minimal() +
   labs(x = '', y = '') +
   theme(legend.position = "none",
         axis.ticks = element_blank(),
         panel.grid=element_blank(),
         axis.text.x=element_blank(),
         legend.title = element_blank()) +
geom_text(aes(y = label_y, label = paste0(continent,':- ', scales::percent(label_per, 0.1))),
          hjust = 0.1, size = 4, colour = grey(0.25))
```

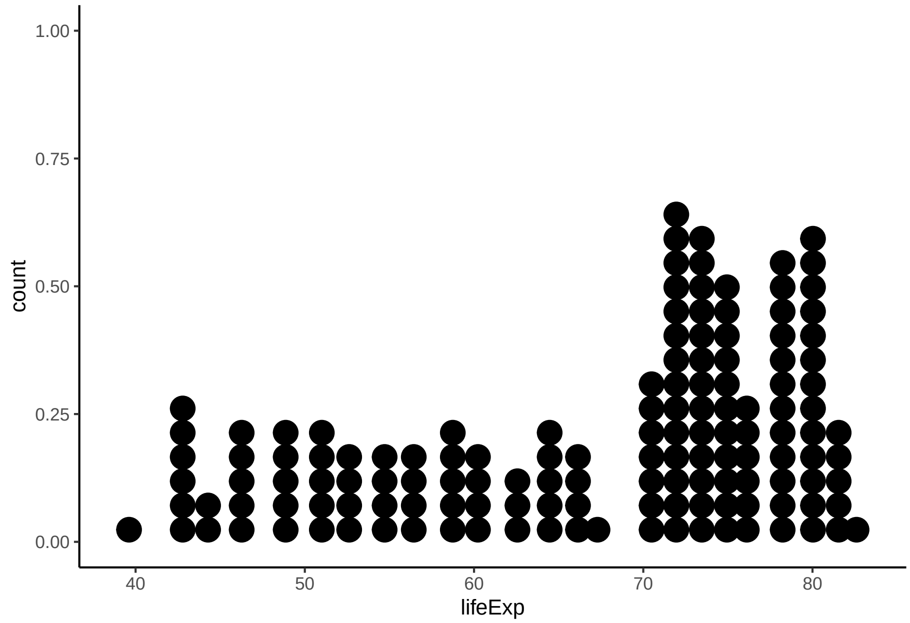


6.16.3 Dot plot

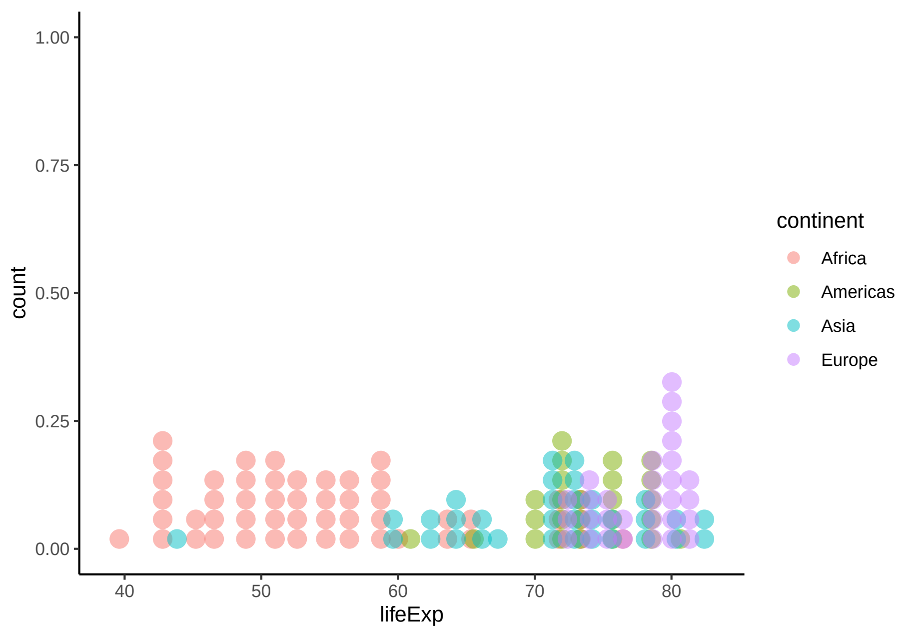
6.16.3.1 Wilkinson dot plot

The function geom_dotplot() is used to create a dot plot.

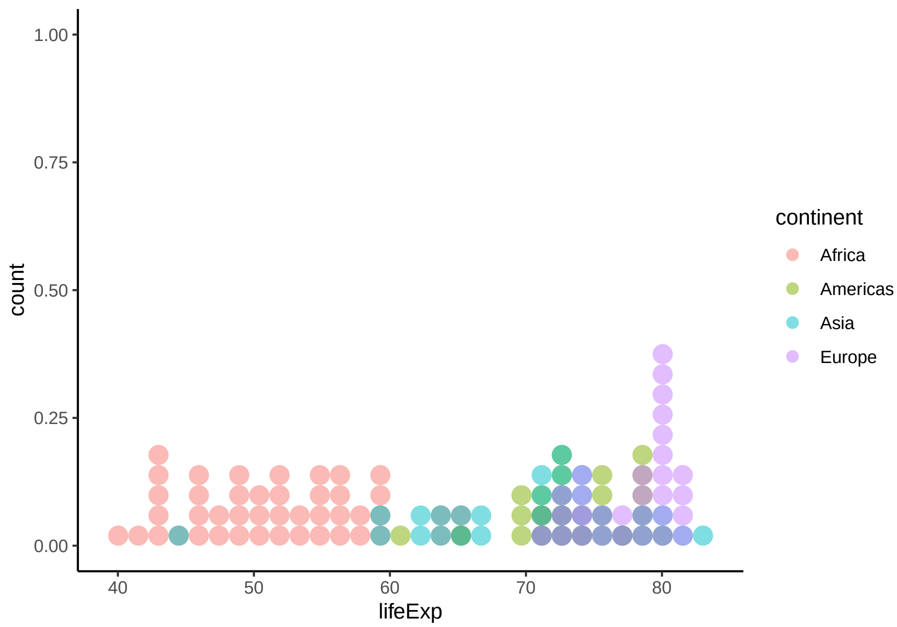
```
ggplot(gapminder_2007, aes(x = lifeExp)) +
   geom_dotplot() +
   theme_classic()
```



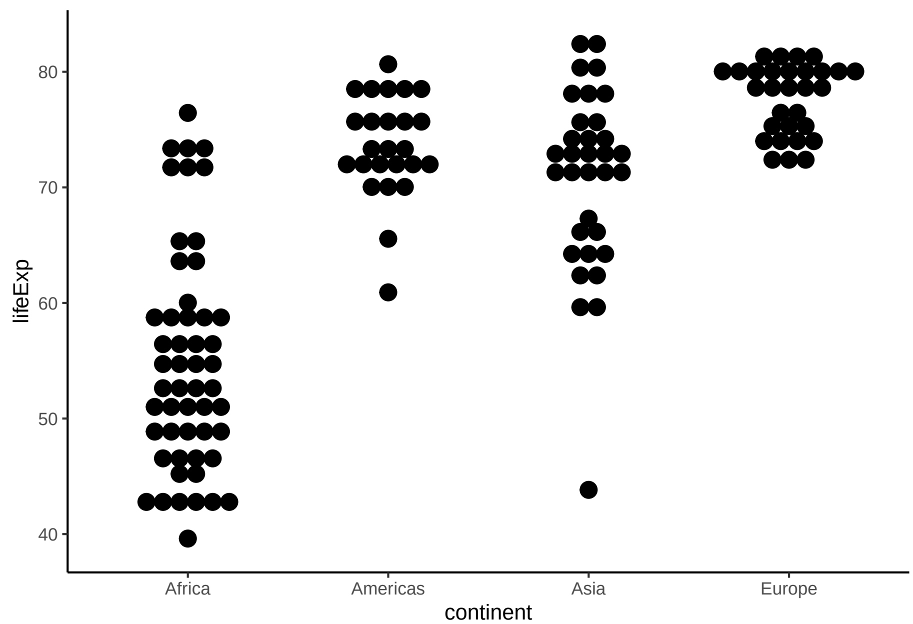

```
ggplot(gapminder_2007, aes(x = lifeExp)) +
   geom_dotplot(aes(fill = continent), alpha = 0.5, colour = NA) +
   theme_classic()
```



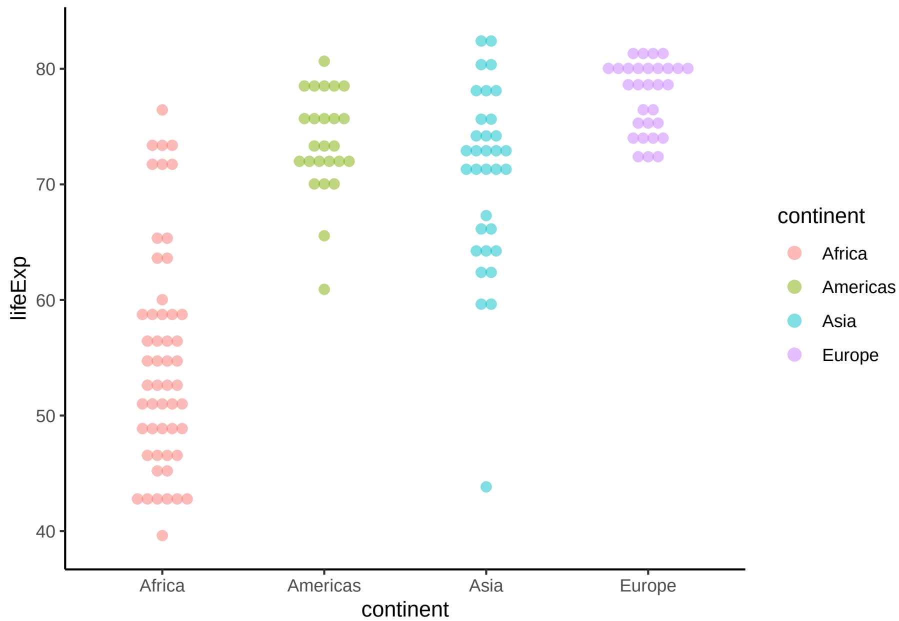
```
ggplot(gapminder_2007, aes(x = lifeExp)) +
  geom_dotplot(aes(fill = continent), alpha = 0.5, colour = NA, method = 'histodot') +
  theme_classic()
```




```
ggplot(data = gapminder_2007, aes(y = lifeExp, x = continent)) +
  geom_dotplot(binaxis = 'y', stackdir = 'center') +
  theme_classic()
```



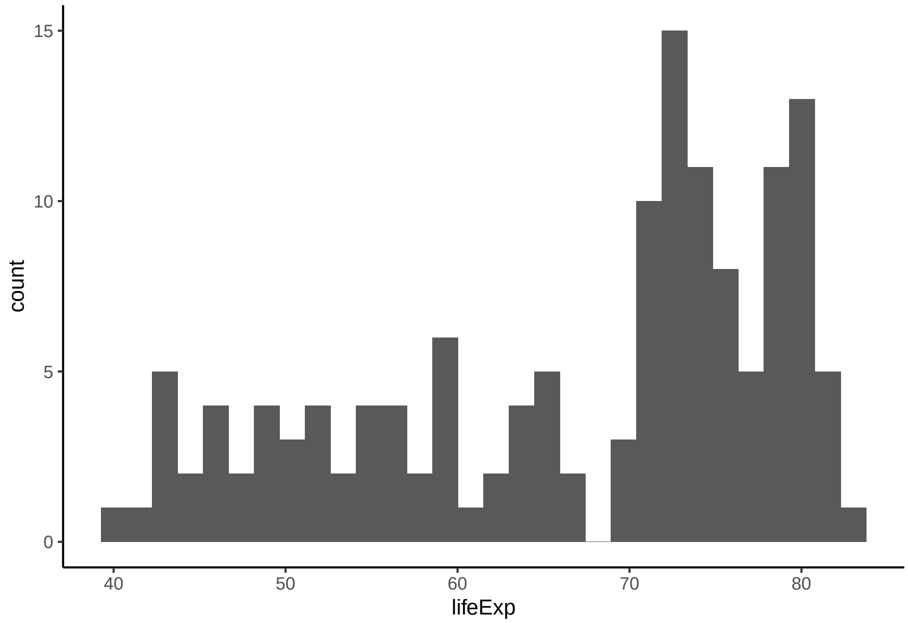

```
ggplot(data = gapminder_2007,
aes(y = lifeExp, x = continent, colour = continent, fill = continent)) +
   geom_dotplot(binaxis = 'y', stackdir = 'center', dotsize = 0.6, alpha = 0.5) +
   theme(legend.position = "none") +
   theme_classic()
```



6.16.4 Histogram

The function geom_histogram() is used to create histograms.

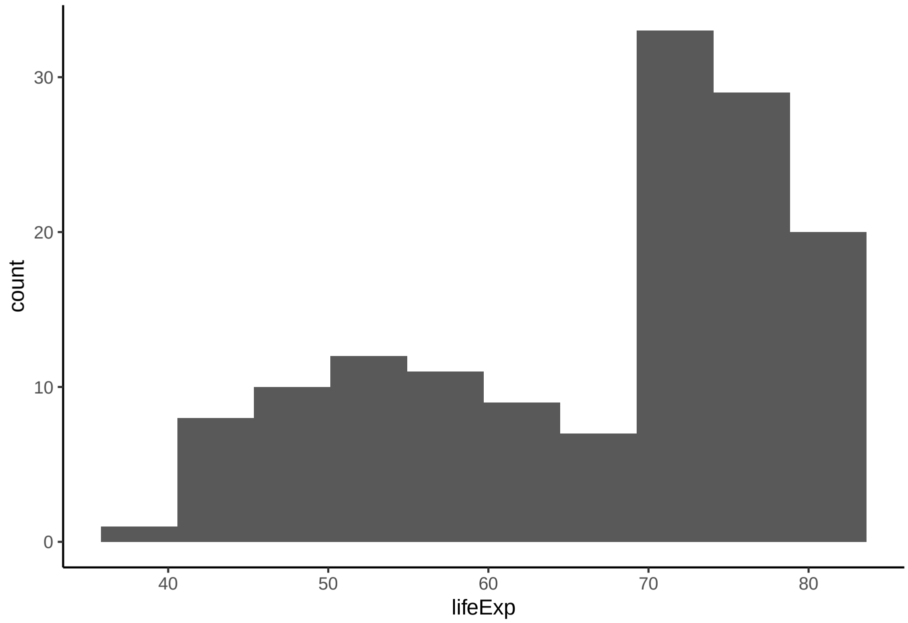
```
ggplot(gapminder_2007) +
  geom_histogram(aes(x = lifeExp)) +
  theme_classic()
```



6.16.4.1 Controlling the number of bins

The argument bins controls the number of bins.

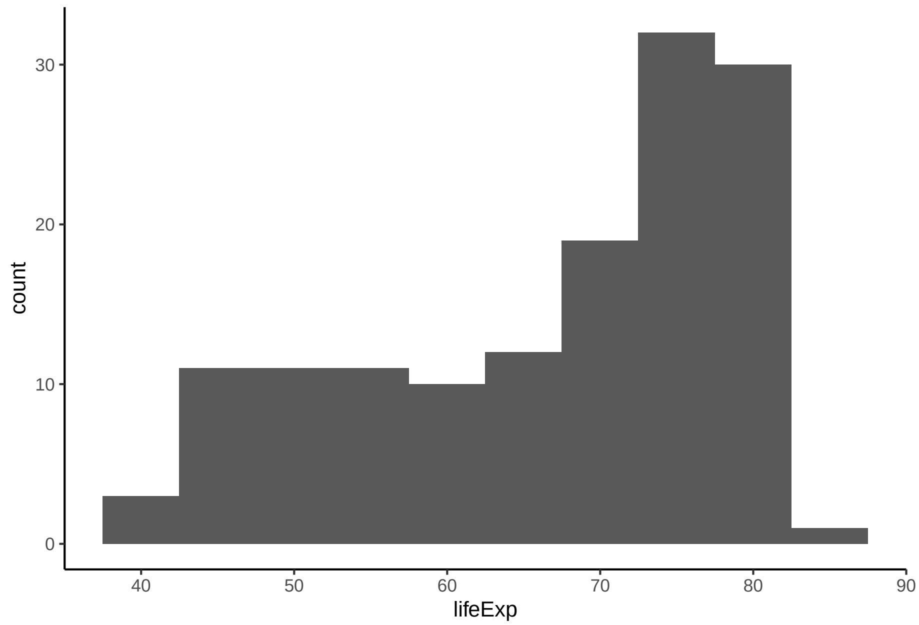
```
ggplot(gapminder_2007) +
   geom_histogram(aes(x = lifeExp), bins = 10) +
   theme_classic()
```



6.16.4.2 Controlling bin size

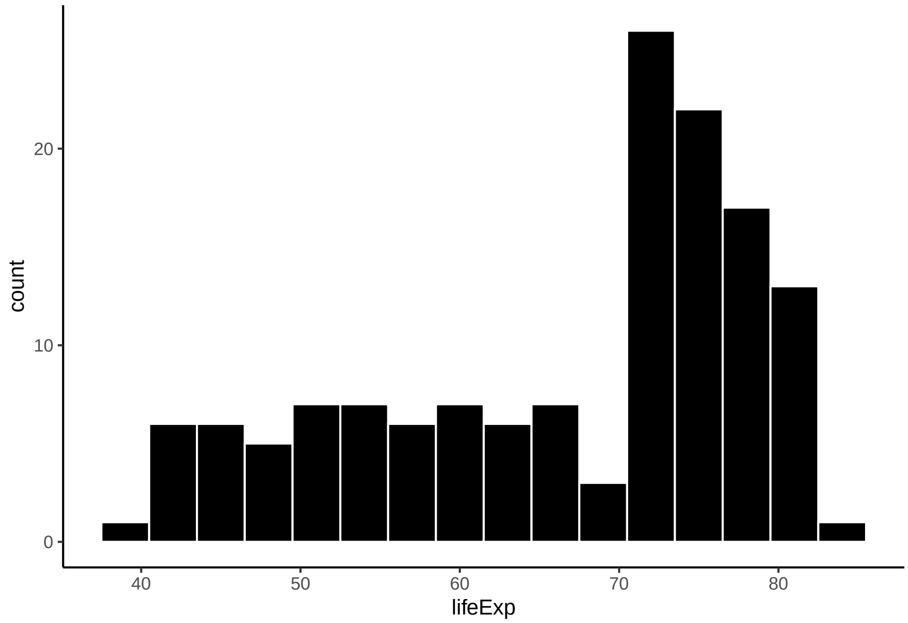
The argument binwidth controls the width of the bins.

```
ggplot(gapminder_2007) +
  geom_histogram(aes(x = lifeExp), binwidth = 5) +
  theme_classic()
```




```
ggplot(gapminder_2007, aes(x = lifeExp)) +
   geom_histogram(binwidth = 3, fill = 'black', colour = 'white') +
   theme_classic()
```

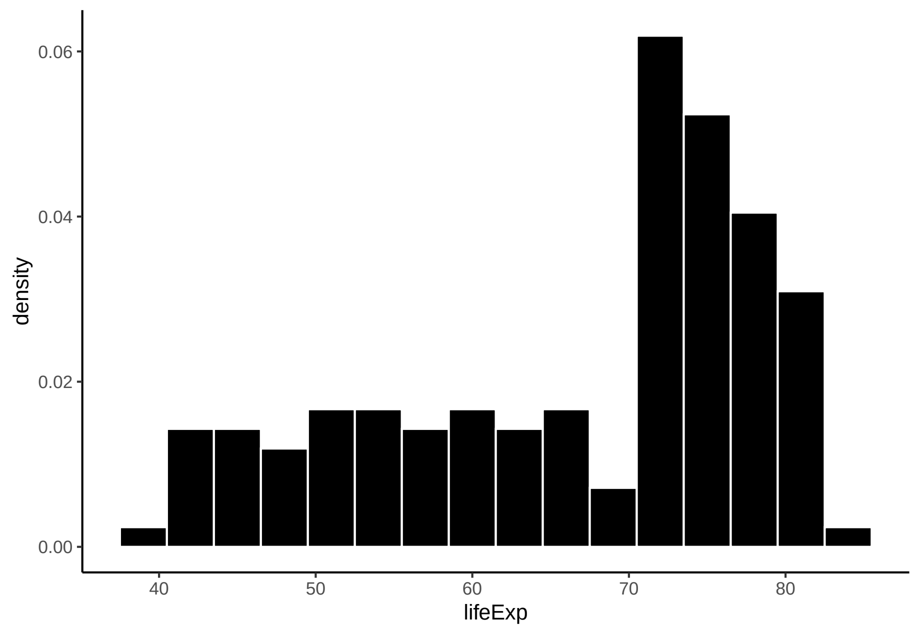
6.16.4.3 Colour and fill



6.16.4.4 Density Histogram

The argument `y = ..density..` is used to create a density histogram. By default, histograms are count but to combine them with density plot, we need to convert them to density histograms.

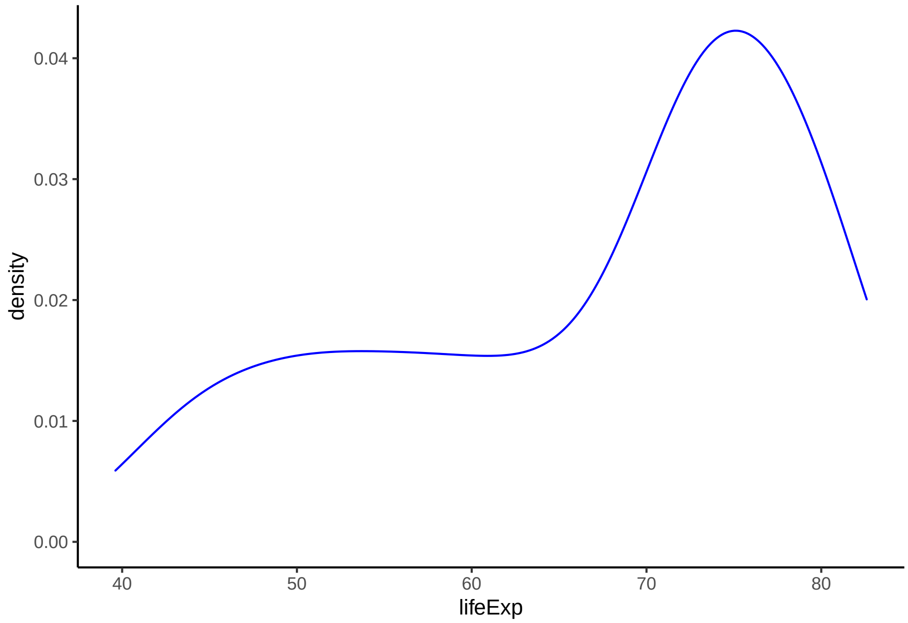
```
ggplot(gapminder_2007, aes(x = lifeExp, y = ..density..)) +
  geom_histogram(fill = 'black', colour = 'white', binwidth = 3) +
  theme_classic()
```



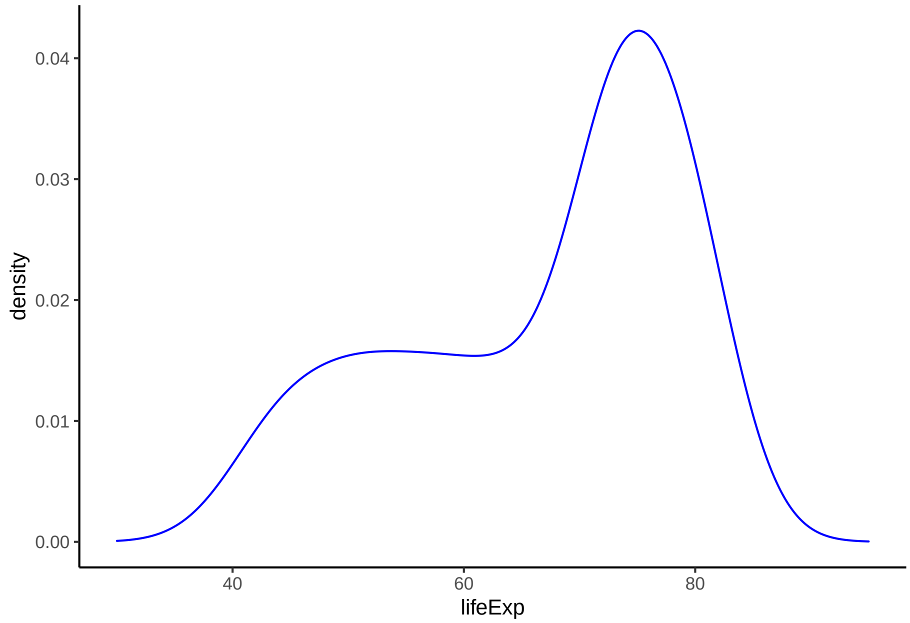
6.16.5 Density plot

The function `geom_density()` creates density plots.

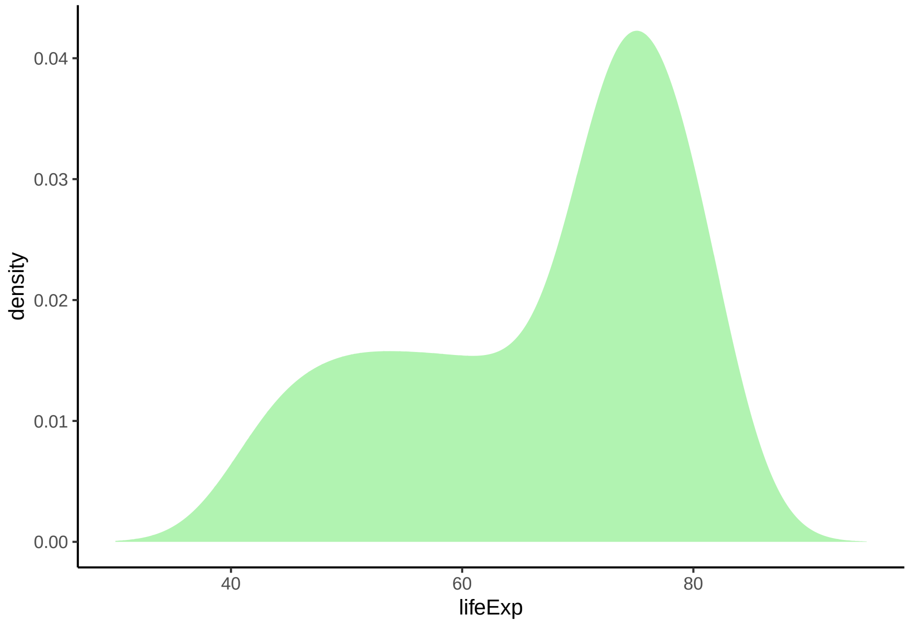
```
ggplot(gapminder_2007, aes(x = lifeExp)) +
   geom_density(colour = 'blue', size = 0.5) +
   theme_classic()
```



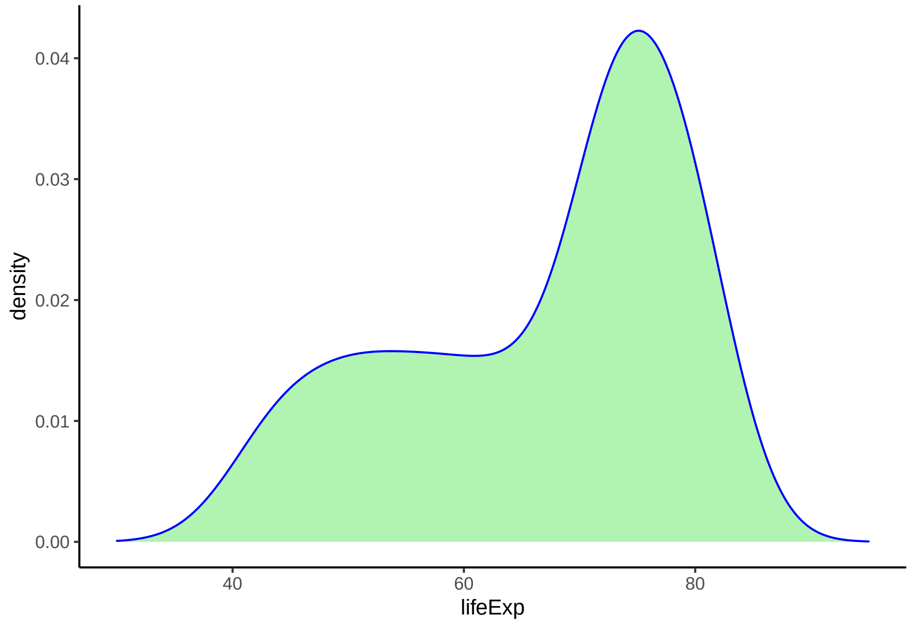

```r
# expanding x-axis
ggplot(gapminder_2007, aes(x = lifeExp)) +
   geom_density(colour = 'blue', size = 0.5) +
   theme_classic() +
xlim(30, 95)
```



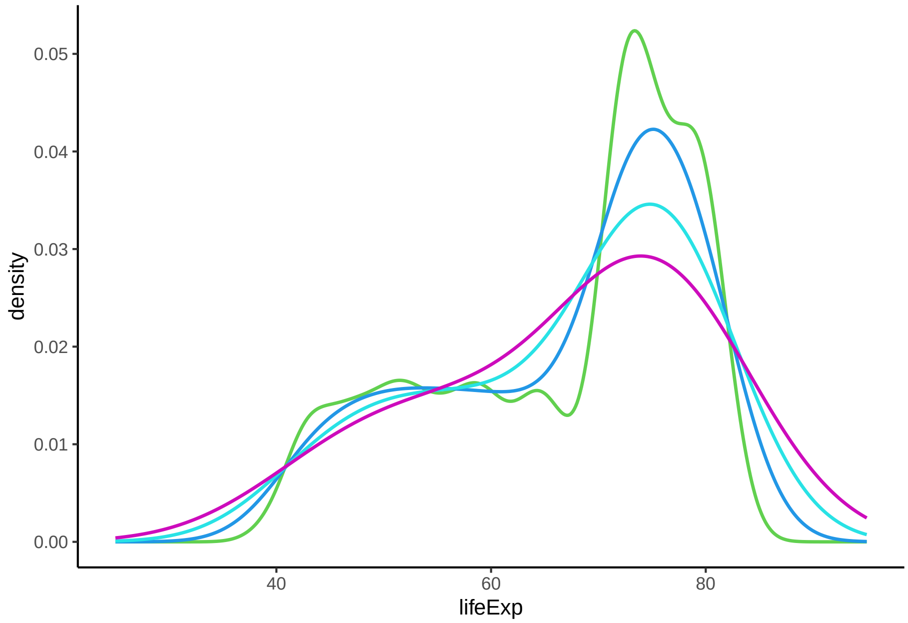

```
# filling area under the curve
ggplot(gapminder_2007, aes(x = lifeExp)) +
   geom_density(colour = NA, fill = 'lightgreen', alpha = 0.7) +
   theme_classic() +
   xlim(30, 95)
```



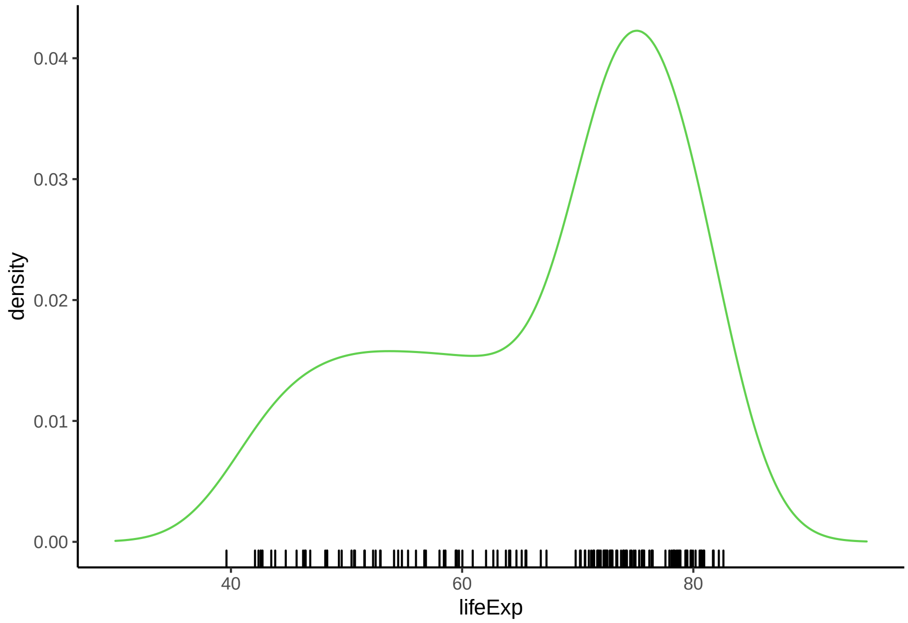

```
# fill and colour
ggplot(gapminder_2007, aes(x = lifeExp)) +
   geom_density(colour = 'blue', fill = 'lightgreen', alpha = 0.7) +
   theme_classic() +
   xlim(30, 95)
```



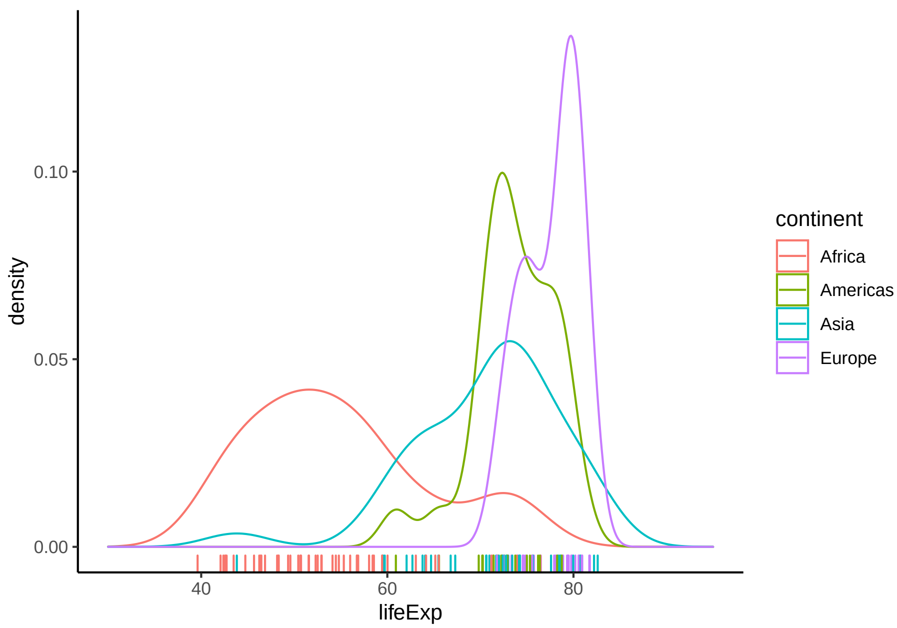

```
# plotting density with geom_line()
ggplot(gapminder_2007, aes(x = lifeExp)) +
   geom_line(colour = 3, stat = 'density', size = 0.8, adjust = 0.5) +
   geom_line(colour = 4, stat = 'density', size = 0.8, adjust = 1) +
   geom_line(colour = 5, stat = 'density', size = 0.8, adjust = 1.5) +
   geom_line(colour = 6, stat = 'density', size = 0.8, adjust = 2) +
   theme_classic() +
   xlim(25, 95)
```



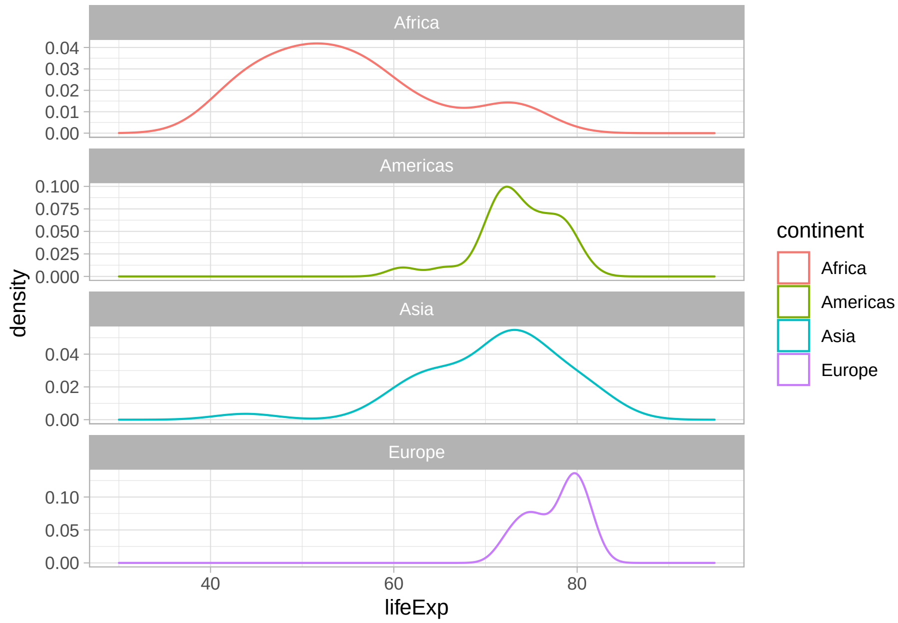

```
# adding rug
ggplot(gapminder_2007, aes(x = lifeExp)) +
   geom_density(colour = 3) +
   xlim(30, 95) +
   theme_classic() +
   geom_rug()
```



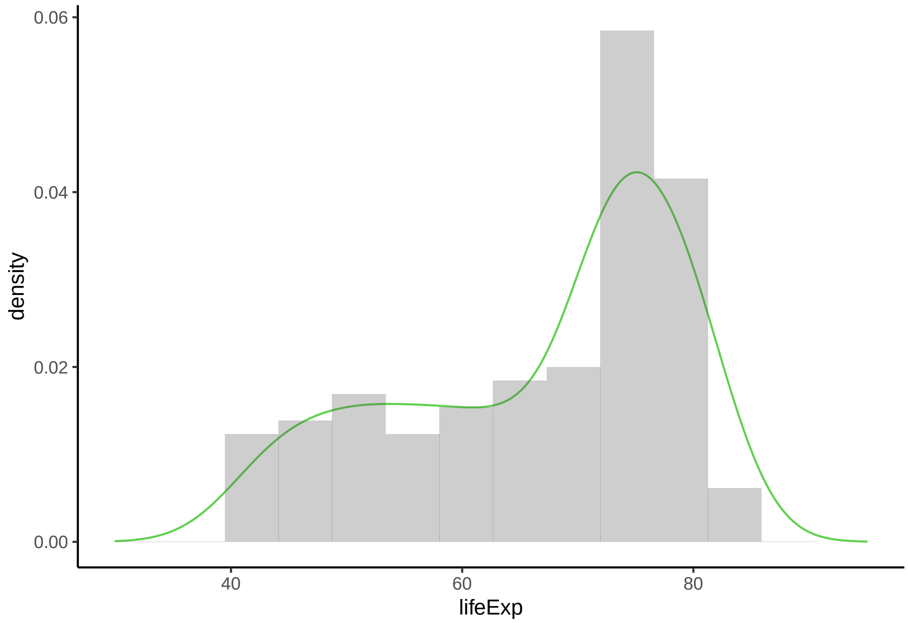

```
# by groups
ggplot(gapminder_2007, aes(x = lifeExp, colour = continent)) +
   geom_density(size = 0.5, alpha = 0.5) +
   xlim(30, 95) +
   geom_rug() +
   theme_classic()
```




```
# subplots
ggplot(gapminder_2007, aes(x = lifeExp, colour = continent)) +
   geom_density(size = 0.5, alpha = 0.5) +
   xlim(30, 95) +
   theme_light() +
   facet_wrap(continent ~ ., nrow = 5, ncol = 1, scales = 'free_y')
```



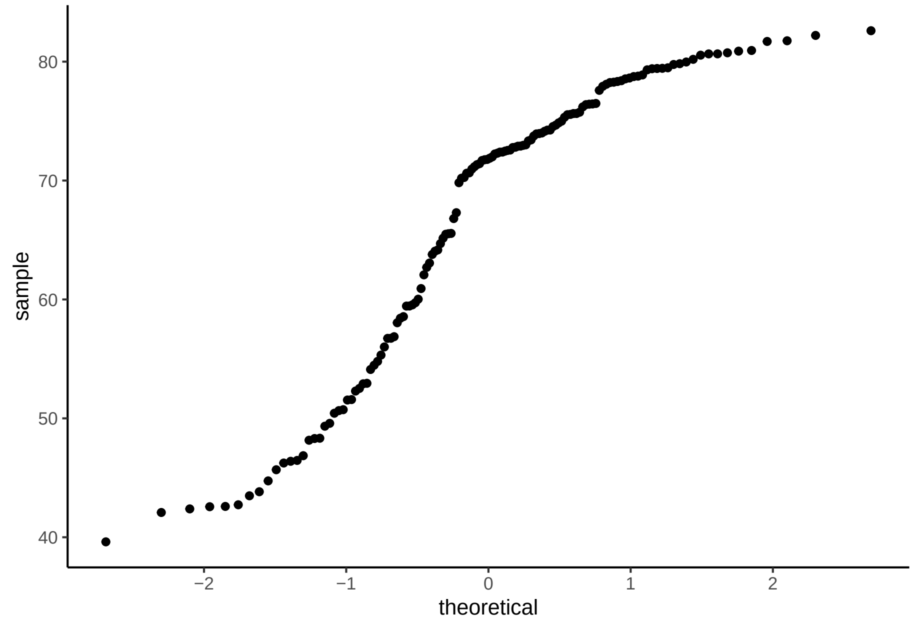

```
# combining density and histogram
ggplot(gapminder_2007, aes(x = lifeExp, y = ..density..)) +
   geom_density(colour = 3, size = 0.5) +
   geom_histogram(alpha = 0.3, bins = 15) +
   theme_classic() +
   xlim(30, 95)
```



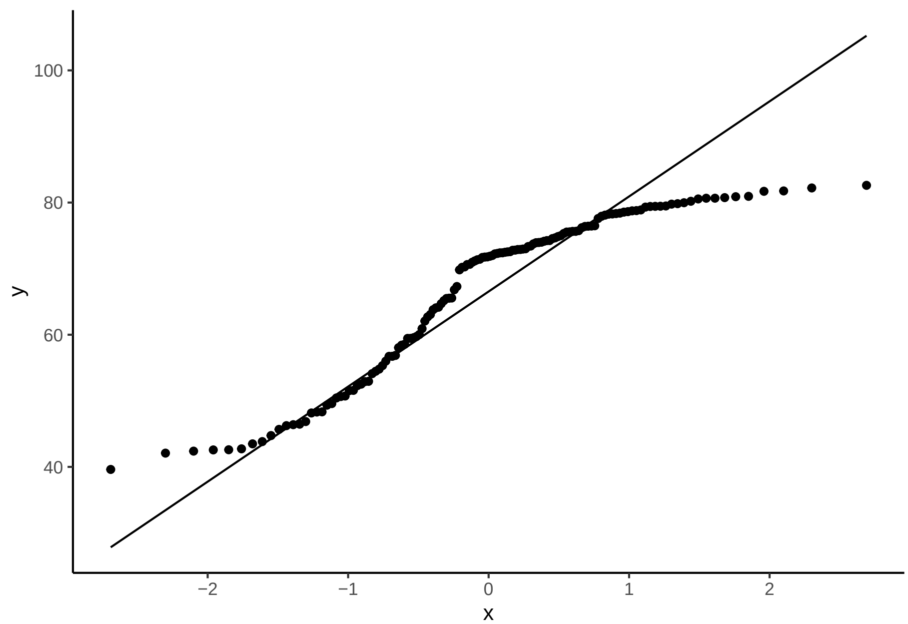
6.16.6 Q-Q plot

The function `geom_qq()` creates a q-q plot.

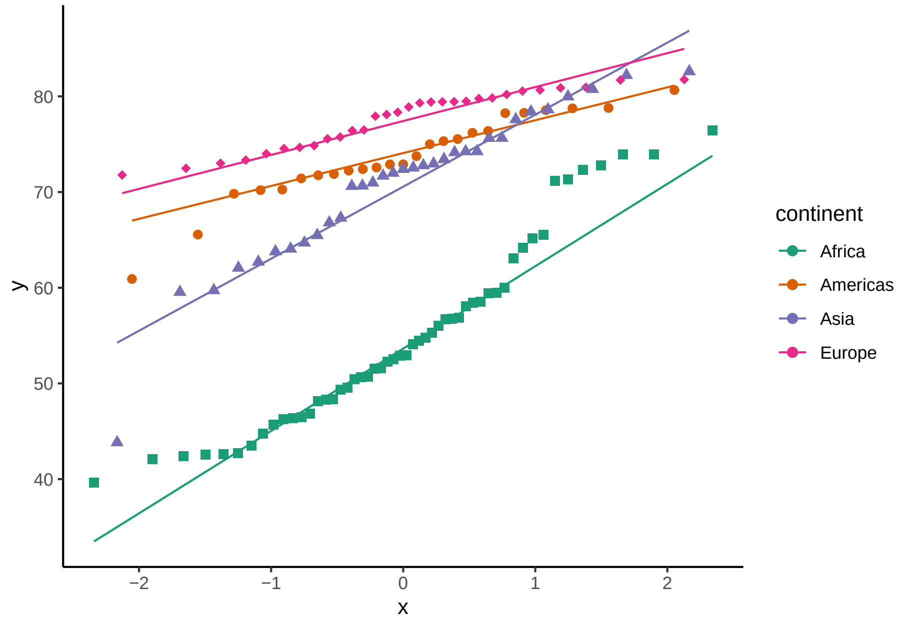
```
ggplot(data = gapminder_2007) +
geom_qq(aes(sample = lifeExp)) +
theme_classic()
```




```
# adding a line
ggplot(data = gapminder_2007, aes(sample = lifeExp)) +
   geom_qq() +
   geom_qq_line() +
   theme_classic()
```



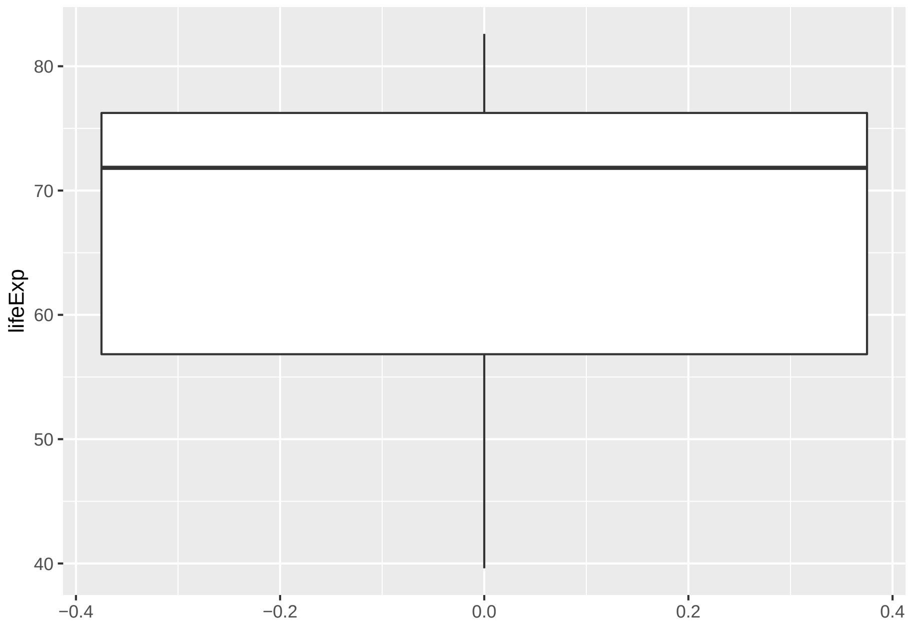

```
# by groups
ggplot(data = gapminder_2007, aes(sample = lifeExp, colour = continent, shape = continent)) +
   geom_qq(size = 2) +
   geom_qq_line() +
   scale_colour_brewer(palette = "Dark2") +
   scale_shape_manual(values = 15:19) +
   guides(shape = 'none') +
   theme_classic()
```



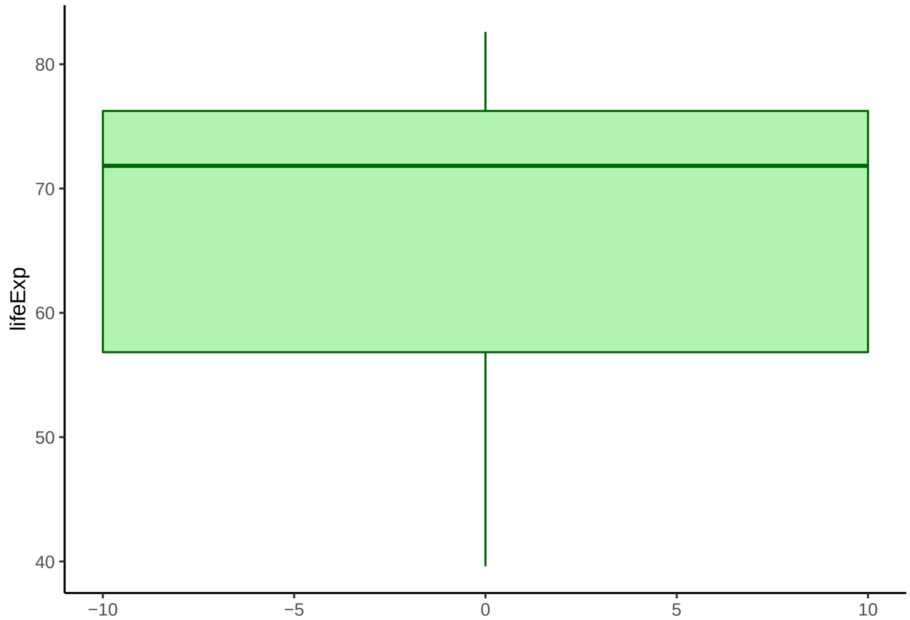
6.16.7 Boxplot

The function `geom_boxplot()` creates a boxplot.

```
ggplot(data = gapminder_2007) +
  geom_boxplot(aes(y = lifeExp))
```



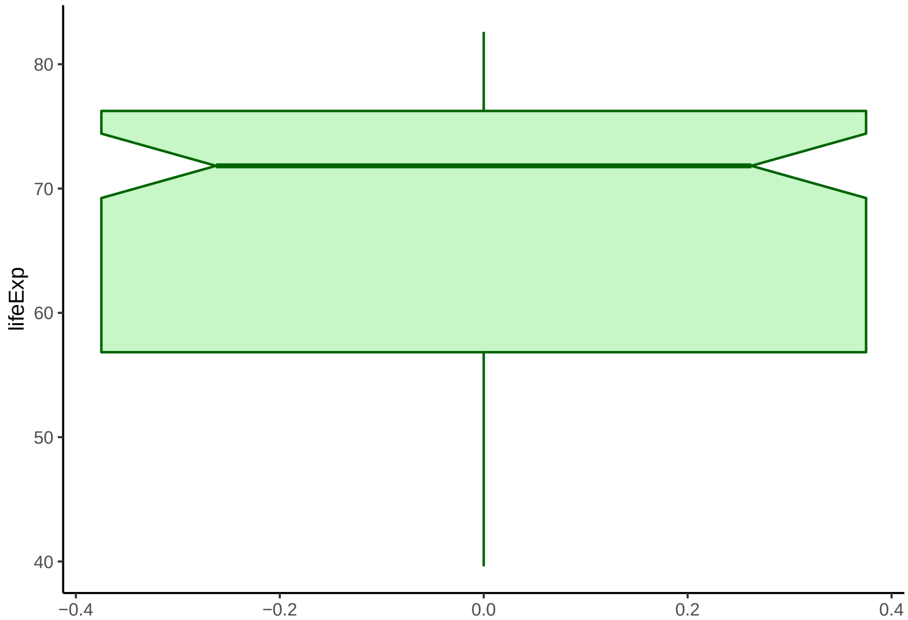

```
ggplot(data = gapminder_2007, aes(y = lifeExp)) +
   geom_boxplot(width = 20,
                fill = 'lightgreen',
                colour = 'darkgreen',
                alpha = 0.7,
                size = 0.5) +
   theme_classic()
```



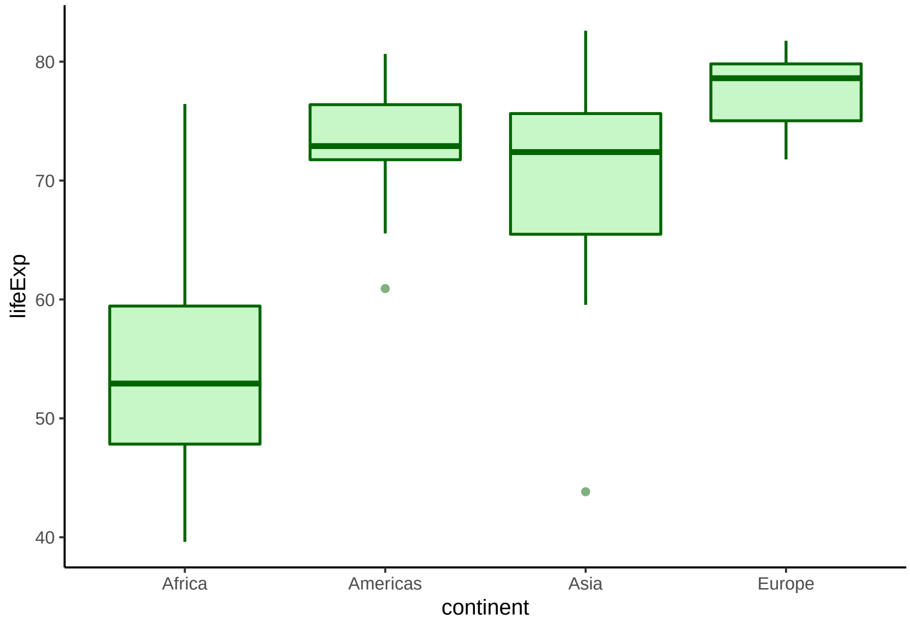
6.16.7.2 Adding notch

The argument notch is used to add notch while notchwidth is used to adjust notch size.

```
ggplot(data = gapminder_2007, aes(y = lifeExp)) +
   geom_boxplot(fill = 'lightgreen',
                colour = 'darkgreen',
                alpha = 0.5,
                size = 0.6,
                notch = TRUE,
                notchwidth = 0.7) +
   theme_classic()
```



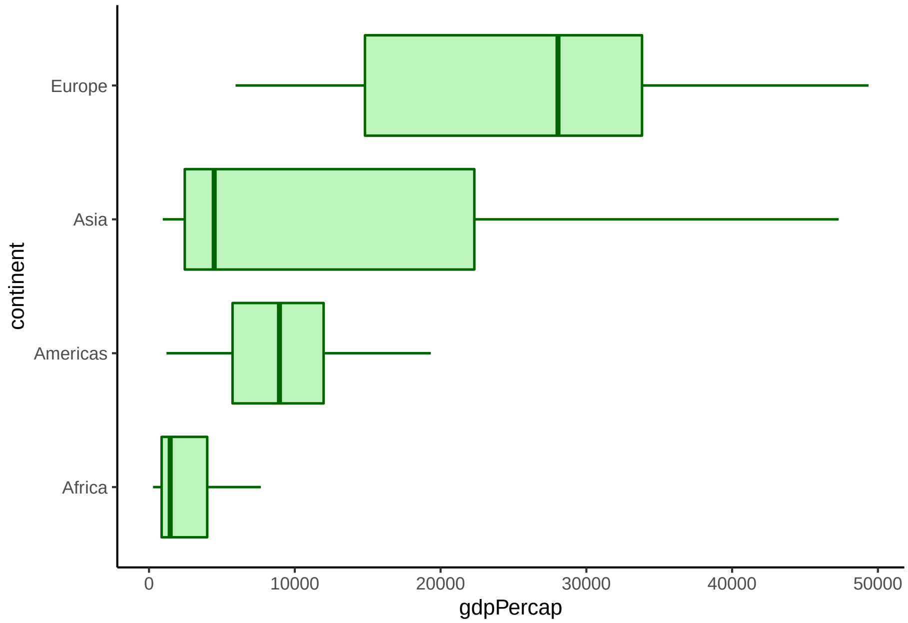

```
ggplot(data = gapminder_2007) +
   geom_boxplot(aes(y = lifeExp, x = continent),
                fill = 'lightgreen',
                colour = 'darkgreen',
                alpha = 0.5,
                size = 0.7) +
   theme_classic()
```



6.16.7.4 Removing outliers

The argument `outlier.shape = NA` is used to remove outliers.

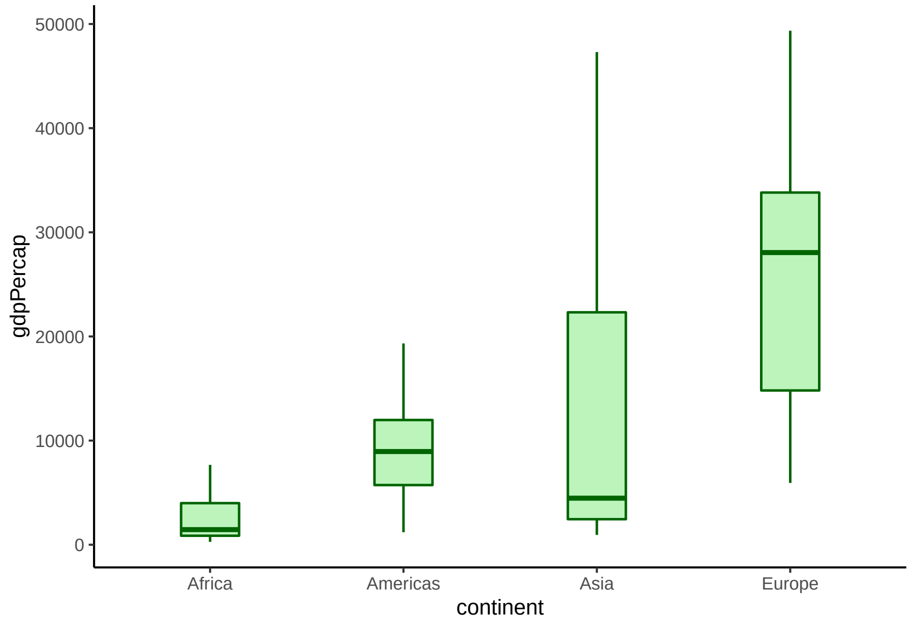
```
# removing outliers
ggplot(data = gapminder_2007) +
   geom_boxplot(aes(y = gdpPercap, x = continent),
                fill = 'lightgreen',
                colour = 'darkgreen',
                size = 0.6,
                alpha = 0.6,
                outlier.shape = NA) +
   coord_flip() +
   theme_classic()
```



6.16.7.5 Box width

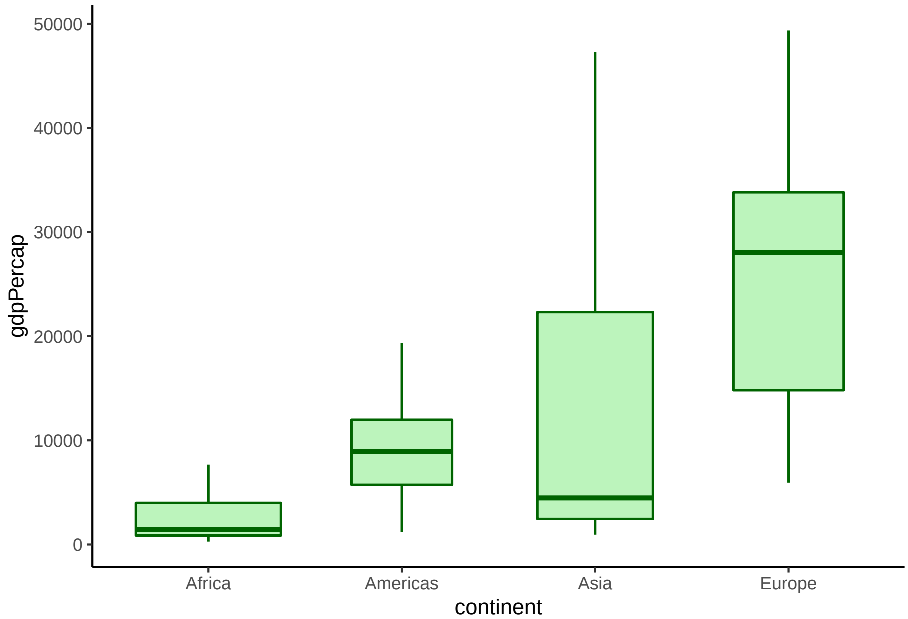
The argument width controls box width.

```
# box width
ggplot(data = gapminder_2007) +
   geom_boxplot(aes(y = gdpPercap, x = continent),
                fill = 'lightgreen',
                colour = 'darkgreen',
                size = 0.6,
                alpha = 0.6,
                outlier.shape = NA,
                width = 0.3) +
   theme_classic()
```



The argument `varwidth = TRUE` enables box width to be proportionate to the square root of the count of values for each group.

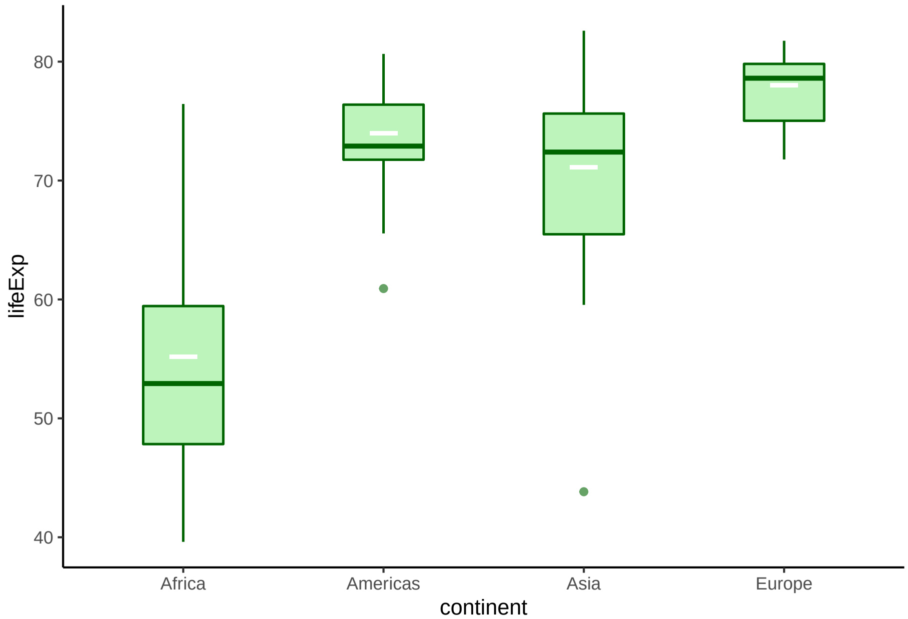
```
# width by the count of values
ggplot(data = gapminder_2007) +
   geom_boxplot(aes(y = gdpPercap, x = continent),
                fill = 'lightgreen',
                colour = 'darkgreen',
                size = 0.6,
                alpha = 0.6,
                outlier.shape = NA,
                varwidth = TRUE) +
   theme_classic()
```



6.16.7.6 Adding mean and median

The function `stat_summary()` can be used to add both mean and median values.

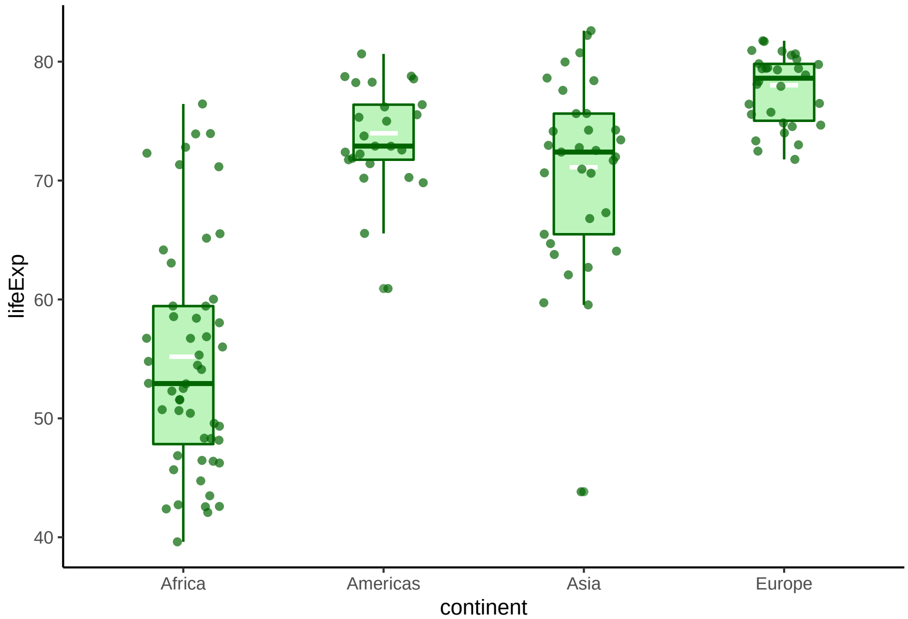
```
# adding mean
ggplot(data = gapminder_2007, aes(y = lifeExp, x = continent), ) +
   geom_boxplot(fill = 'lightgreen',
                colour = 'darkgreen',
                size = 0.6,
                alpha = 0.6,
                width = 0.4) +
   stat_summary(fun.y = mean, geom = 'point', shape = '-', size = 10, colour = 'white') +
   theme_classic()
```



6.16.7.7 Adding jitter

The function `geom_jitter()` is used to add jitter to a plot.

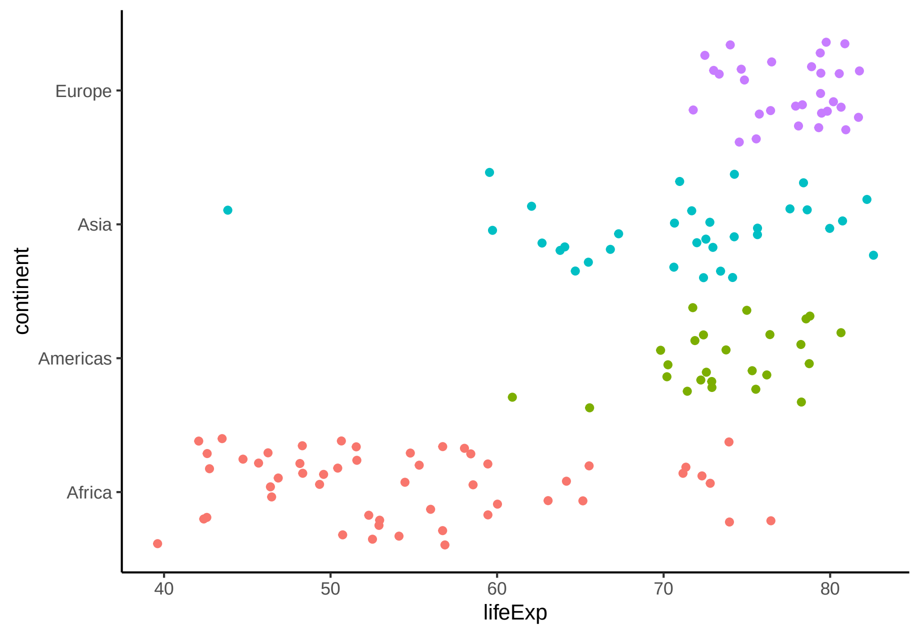
```
ggplot(data = gapminder_2007, aes(y = lifeExp, x = continent)) +
   geom_boxplot(fill = 'lightgreen',
                colour = 'darkgreen',
                size = 0.6,
                alpha = 0.6,
                width = 0.3) +
   stat_summary(fun.y = mean, geom = 'point', shape = '-', size = 10, colour = 'white')
   geom_jitter(width = 0.2, alpha = 0.7, colour = 'darkgreen') +
   theme_classic()
```



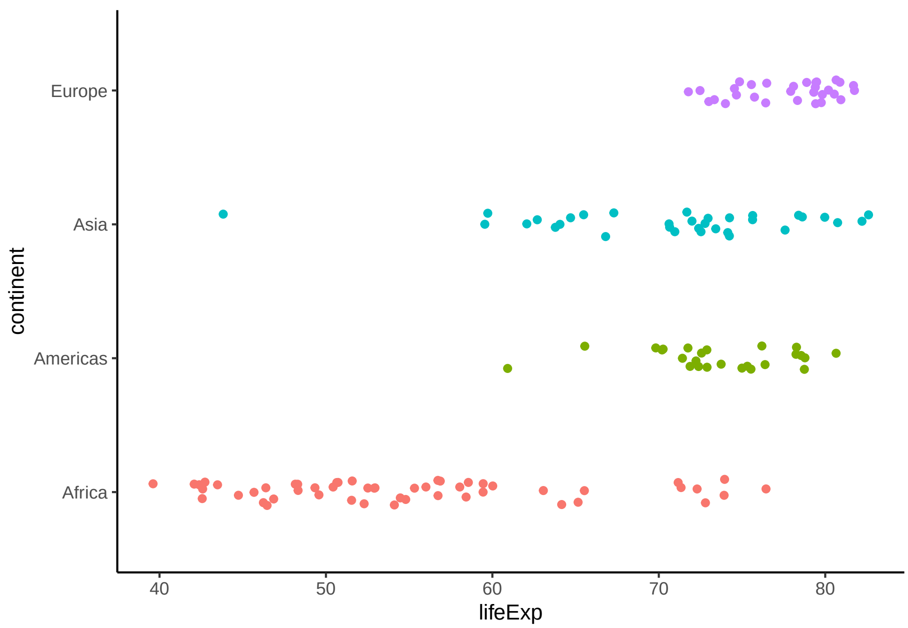
6.16.8 Strip plot

There is no specific geom to create a strip plot but using `geom_jitter()`, we can create a strip plot.

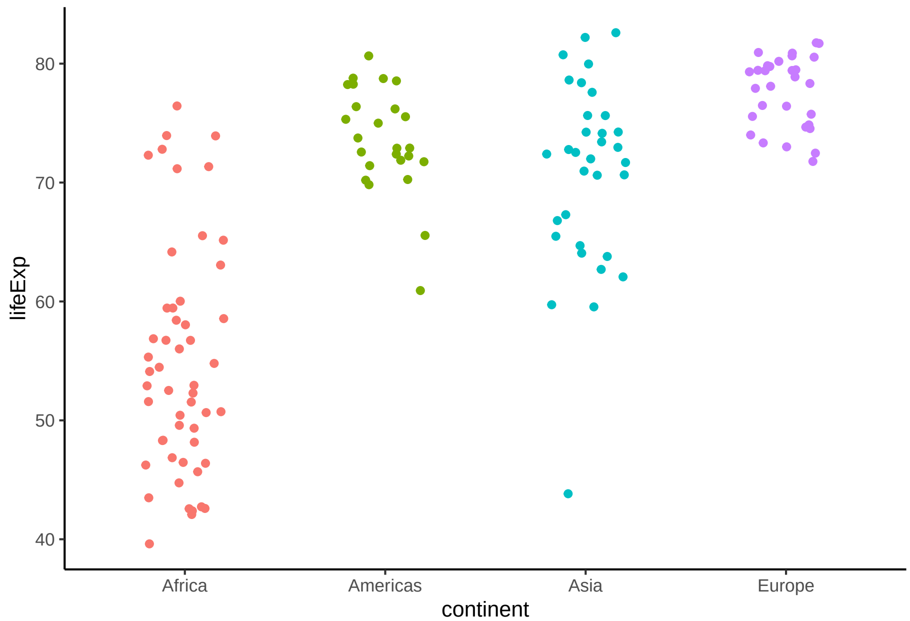
```
ggplot(data = gapminder_2007, aes(x = lifeExp, y = continent, colour = continent)) +
   geom_jitter() +
   theme_classic() +
   theme(legend.position = "none")
```



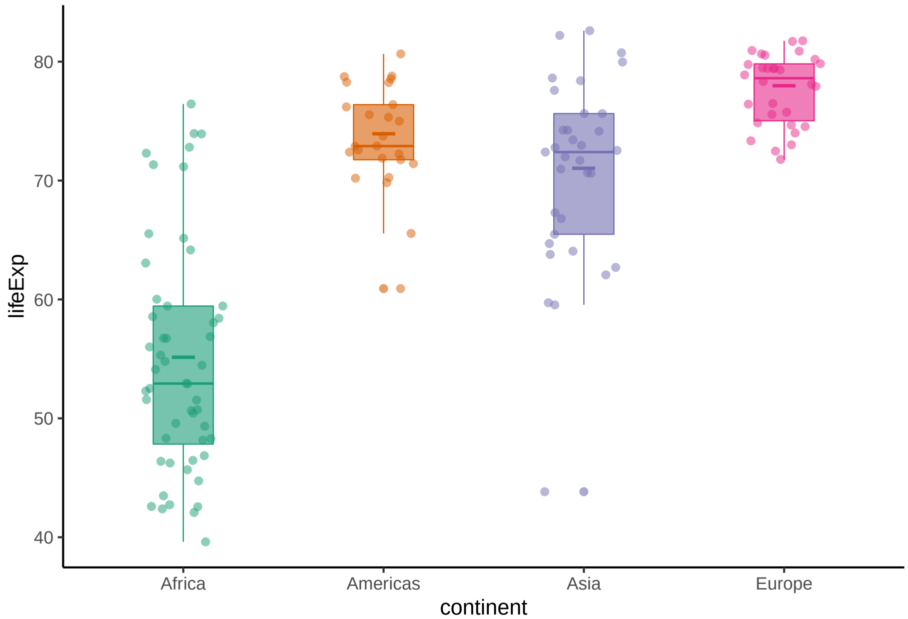

```
ggplot(data = gapminder_2007, aes(x = lifeExp, y = continent, colour = continent)) +
   geom_jitter(position = position_jitter(height = 0.1)) +
   theme_classic() +
   theme(legend.position = "none")
```




```
ggplot(data = gapminder_2007, aes(y = lifeExp, x = continent, colour = continent)) +
   geom_jitter(position = position_jitter(width = 0.2)) +
   theme_classic() +
   theme(legend.position = "none")
```



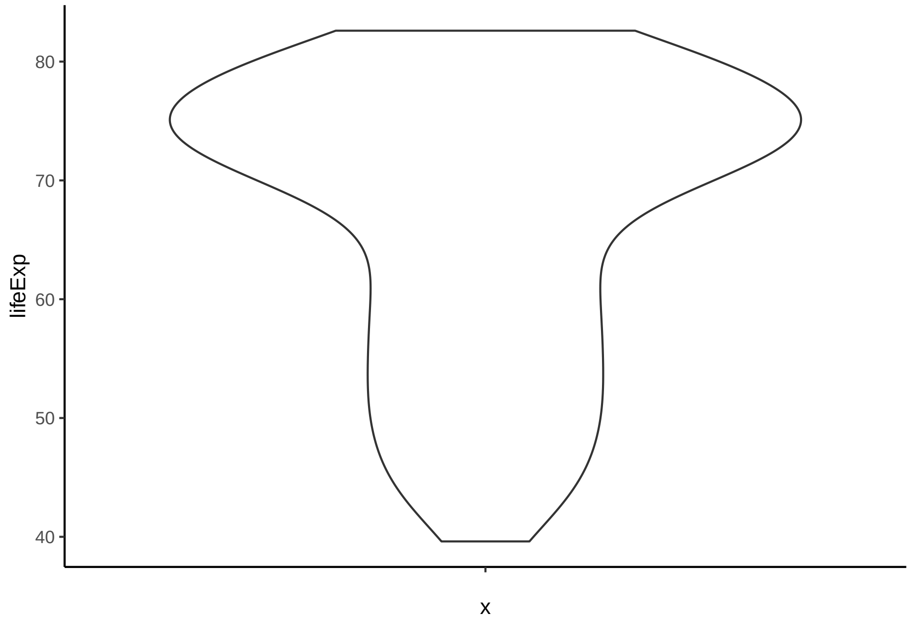

```
ggplot(data = gapminder_2007,
       aes(y = lifeExp, x = continent, colour = continent, fill = continent)) +
  geom_boxplot(size = 0.3, alpha = 0.6, width = 0.3) +
  stat_summary(fun.y = mean, geom = 'point', shape = '-', size = 8) +
  geom_jitter(position = position_jitter(width = 0.2), alpha = 0.5) +
  scale_colour_brewer(palette = "Dark2") +
  scale_fill_brewer(palette = "Dark2") +
  theme_classic() +
  theme(legend.position = "none")
```



6.16.9 Violin plot

The function `geom_violin()` creates a violin plot.

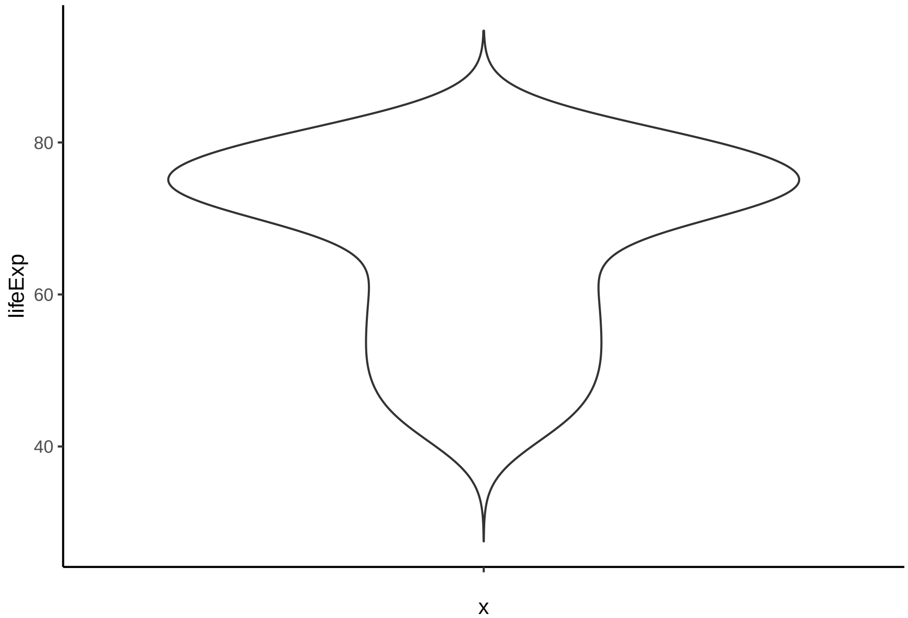
```
ggplot(data = gapminder_2007, aes(y = lifeExp, x = '')) +
  geom_violin() +
  theme_classic()
```



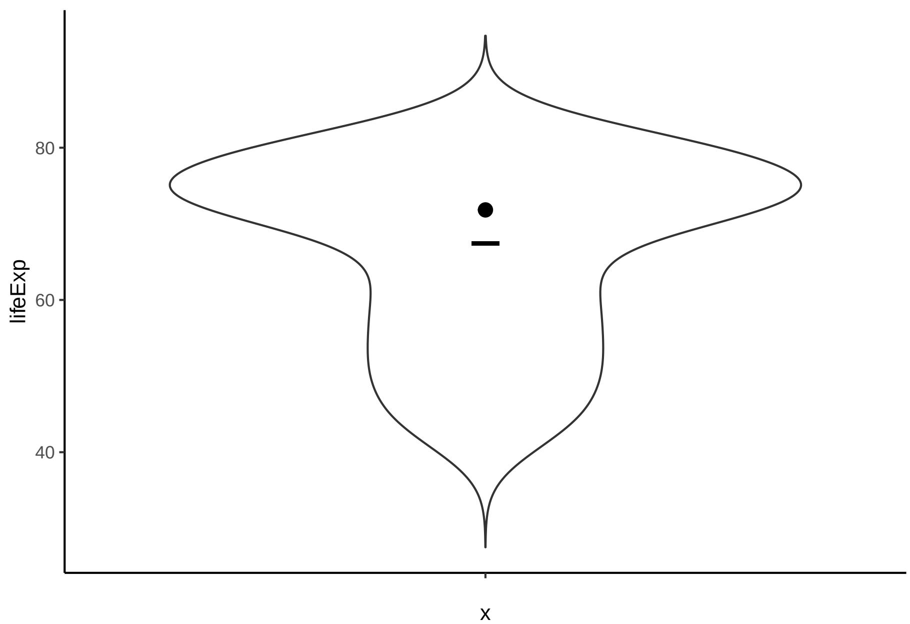
6.16.9.1 Remove trimming

The argument `trim = FALSE` removes trimming.

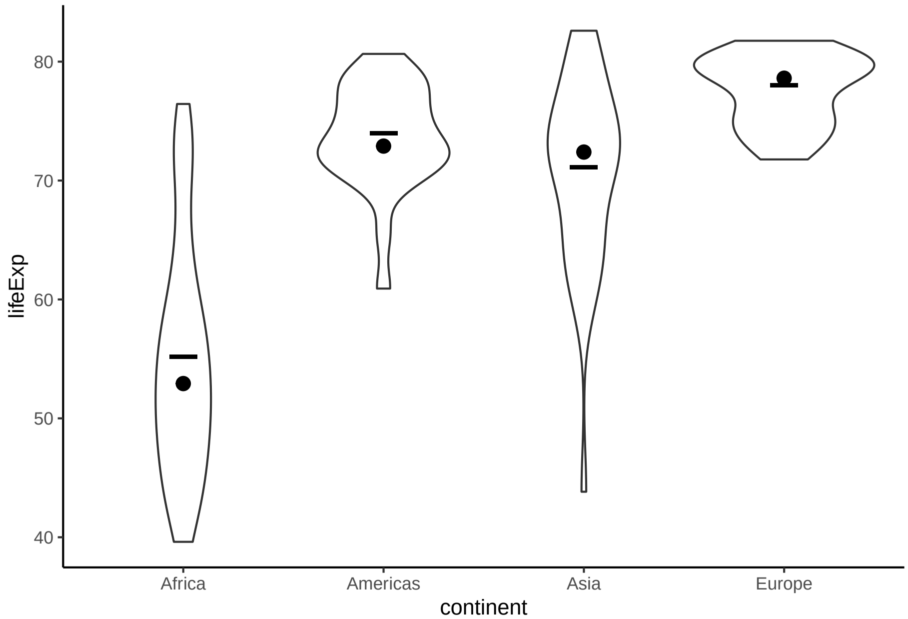
```
# removing trim
ggplot(data = gapminder_2007, aes(y = lifeExp, x = '')) +
   geom_violin(trim = FALSE) +
   theme_classic()
```



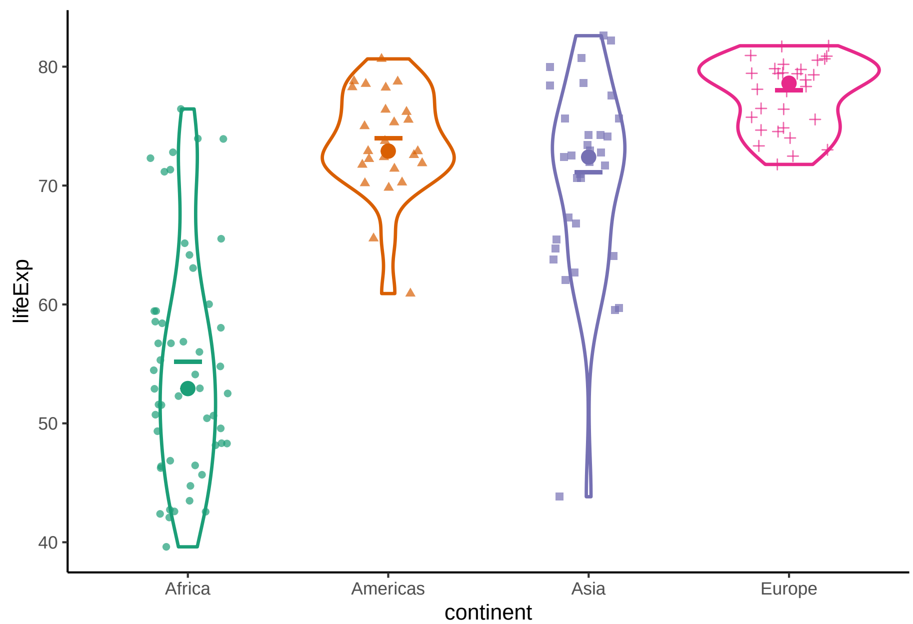

```
# adding mean and median
ggplot(data = gapminder_2007, aes(y = lifeExp, x = '')) +
   geom_violin(trim = FALSE) +
   stat_summary(fun.y = mean, geom = 'point', shape = '-', size = 10) +
   stat_summary(fun.y = median, geom = 'point', shape = 19, size = 3) +
   theme_classic()
```




```
ggplot(data = gapminder_2007, aes(y = lifeExp, x = continent)) +
   geom_violin() +
   stat_summary(fun.y = mean, geom = 'point', shape = '-', size = 10) +
   stat_summary(fun.y = median, geom = 'point', shape = 19, size = 3) +
   theme_classic()
```




```
ggplot(data = gapminder_2007,
       aes(y = lifeExp, x = continent, color = continent, shape = continent)) +
   geom_violin(size = 0.8) +
   stat_summary(fun.y = mean, geom = 'point', shape = '-', size = 10) +
   stat_summary(fun.y = median, geom = 'point', shape = 19, size = 3) +
   geom_jitter(position = position_jitter(width = 0.2), alpha = 0.7) +
   scale_colour_brewer(palette = "Dark2") +
   theme_classic() +
   theme(legend.position = "none")
```

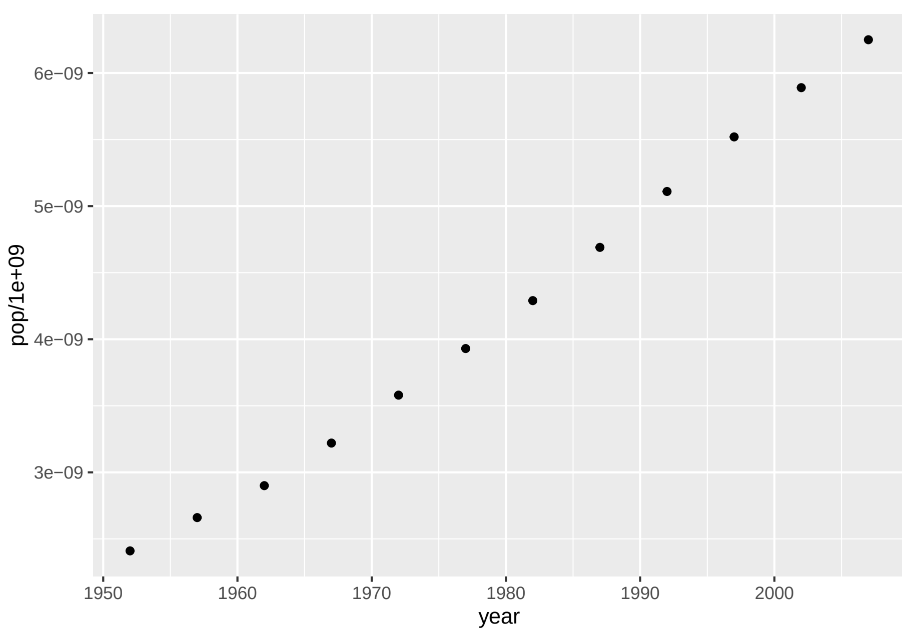


6.16.10 Line graph

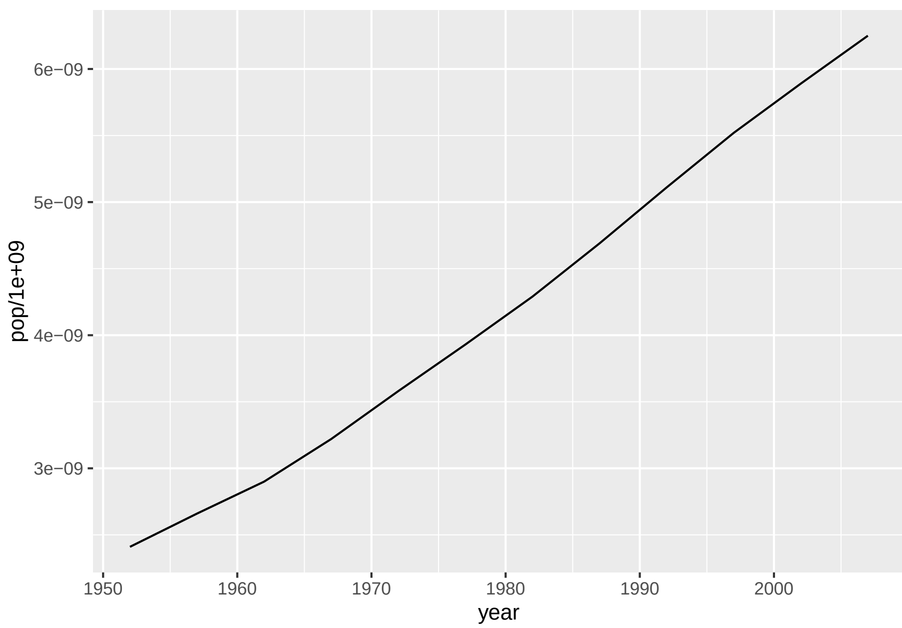
The function `geom_line()` produces a line plot.

```
# preparing plot
pop_growth <-
gapminder %>%
group_by(year) %>%
summarise(pop = round(sum(pop/1e9, na.rm = T), 2))
pop_growth
#> # A tibble: 12 x 2
#>     year   pop
#>    <int> <dbl>
#>  1  1952  2.41
#>  2  1957  2.66
#>  3  1962  2.9
#>  4  1967  3.22
#>  5  1972  3.58
#>  6  1977  3.93
#>  7  1982  4.29
#>  8  1987  4.69
#>  9  1992  5.11
#> 10  1997  5.52
#> 11  2002  5.89
#> 12  2007  6.25
```

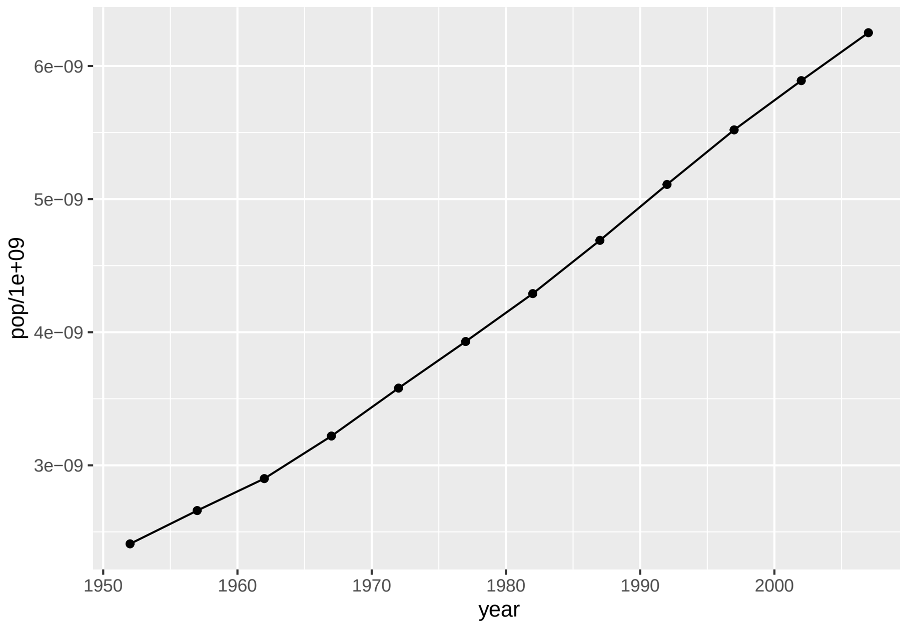
```
ggplot(data = pop_growth, aes(y = pop/1e9, x = year)) +
  geom_point()
```



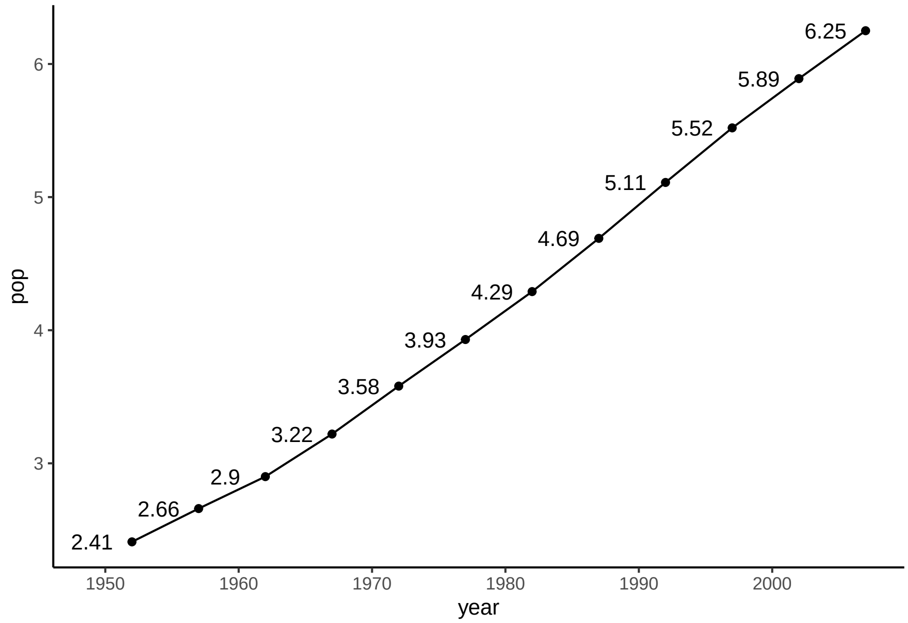

```
# adding line
ggplot(data = pop_growth, aes(y = pop/1e9, x = year)) +
  geom_line()
```



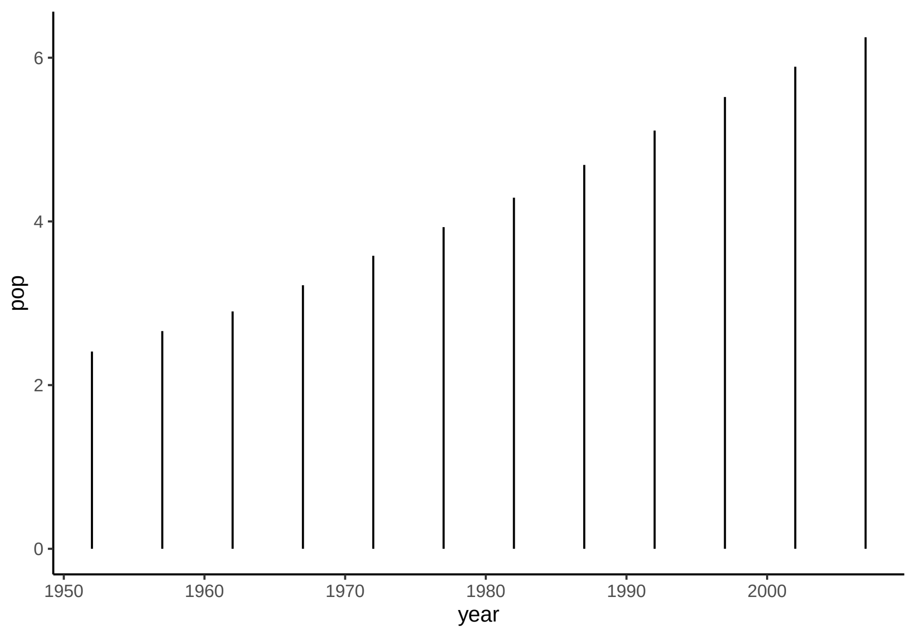

```
# combining line and points
ggplot(data = pop_growth, aes(y = pop/1e9, x = year)) +
  geom_line() +
  geom_point()
```




```
# adding data label
ggplot(data = pop_growth, aes(y = pop, x = year)) +
  geom_line() +
  geom_point() +
  geom_text(aes(label = round(pop, 2)), nudge_x = -3) +
  theme_classic()
```



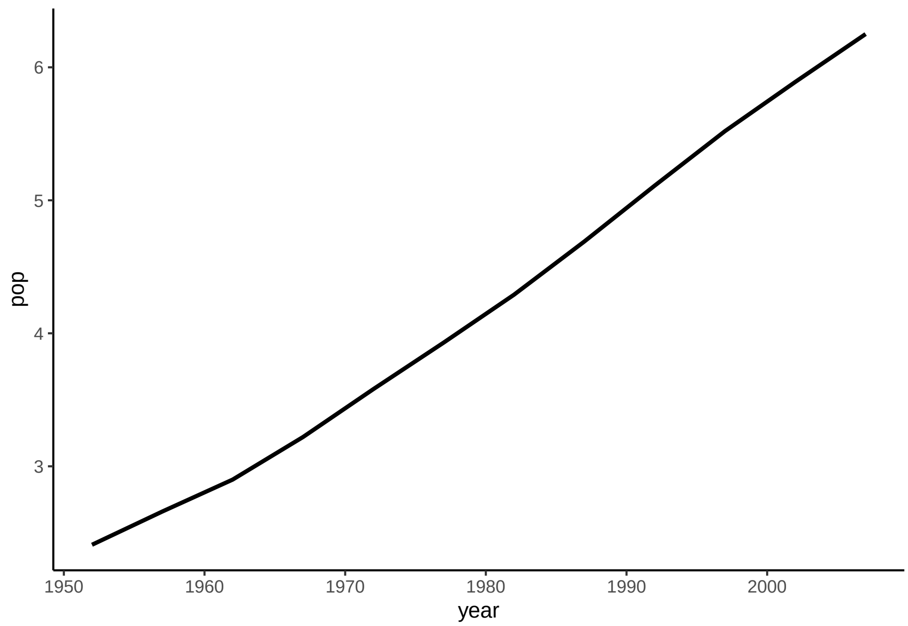

```
ggplot(data = pop_growth, aes(y = pop, x = year)) +
  geom_segment(aes(y = pop, x = year, yend = 0, xend = year)) +
  theme_classic()
```



6.16.10.1 Line width

The argument `size=`, control line width.

```
ggplot(data = pop_growth, aes(y = pop, x = year)) +
  geom_line(size = 1) +
  theme_classic()
```

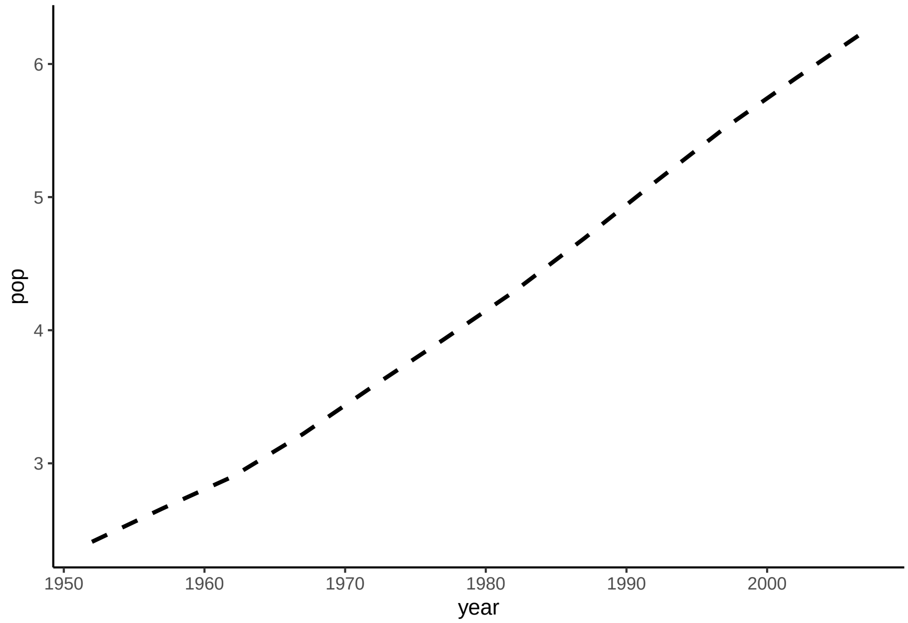


6.16.10.2 Line style

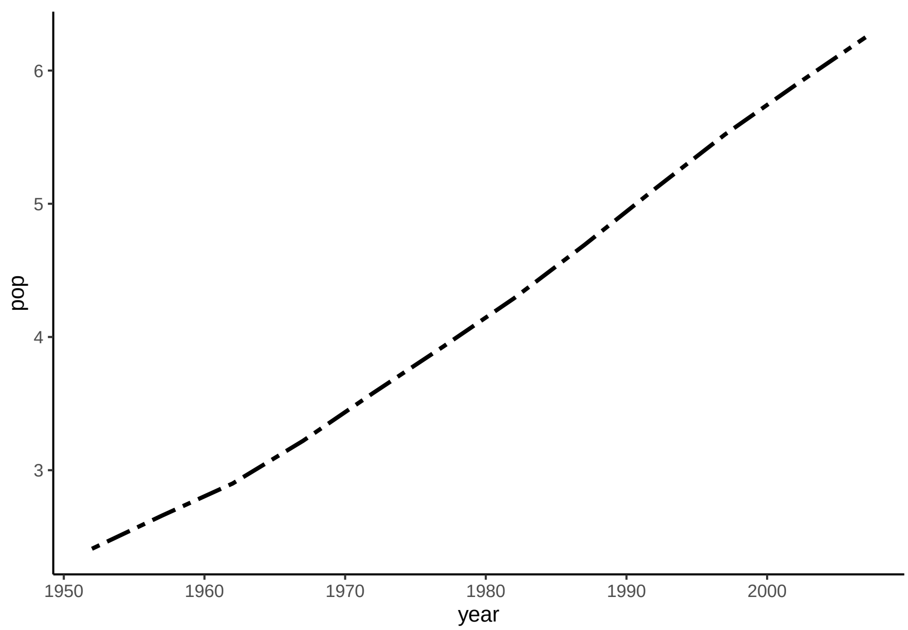
The argument `linetype=` controls line style. It accepts the same values as base graphics that is, integers ranging from 0 to 6 and

- 'blank' = 0,
- 'solid' = 1 (default)
- 'dashed' = 2
- 'dotted' = 3
- 'dotdash' = 4
- 'longdash' = 5
- 'twodash' = 6

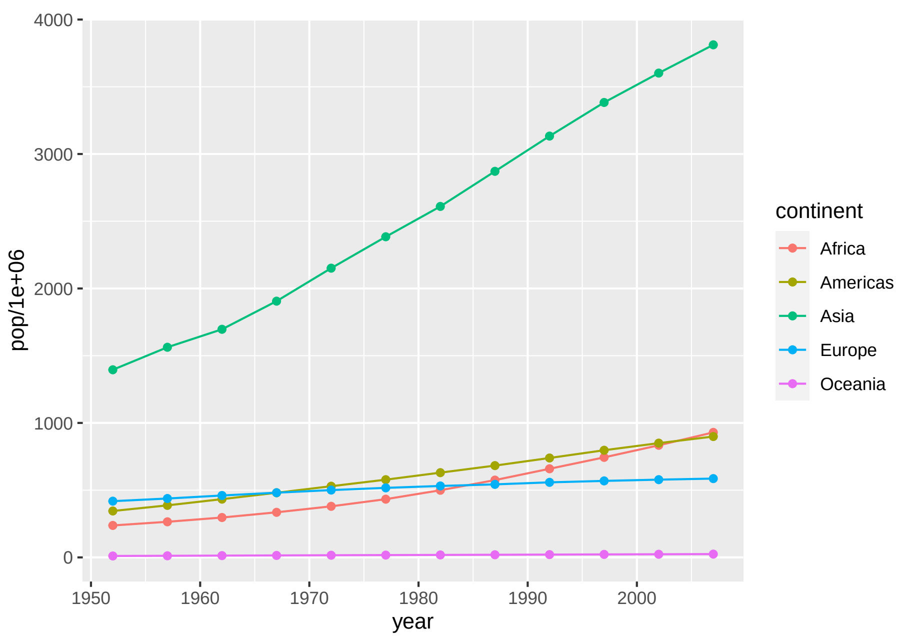
```
ggplot(data = pop_growth, aes(y = pop, x = year)) +
  geom_line(size = 1, linetype = 2) +
  theme_classic()
```



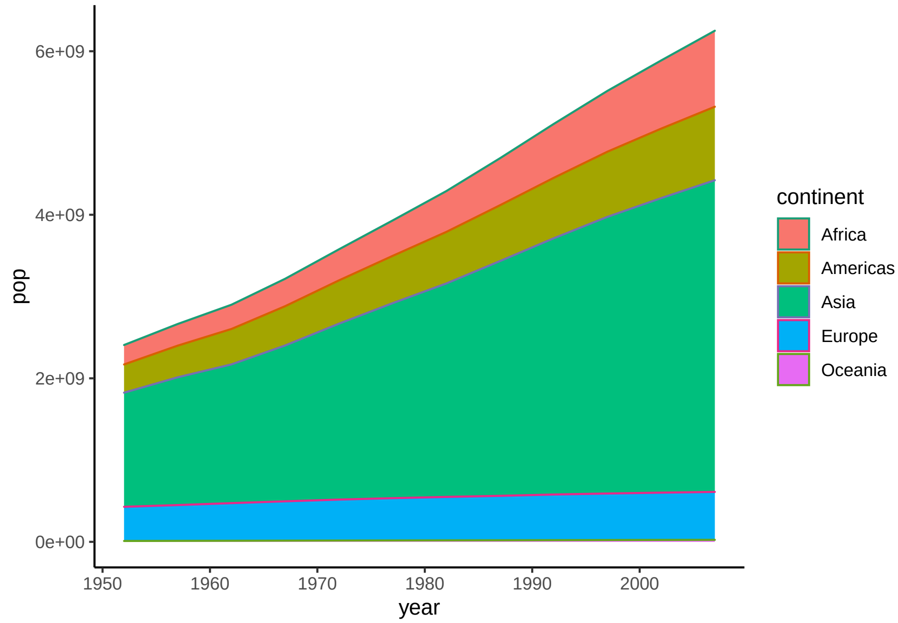

```
ggplot(data = pop_growth, aes(y = pop, x = year)) +
  geom_line(size = 1, linetype = 'twodash') +
  theme_classic()
```



```r
# preparing data
pop_growth_cont <- aggregate(pop ~ year + continent, gapminder, sum)
head(pop_growth_cont)
#>   year continent       pop
#> 1 1952    Africa 237640501
#> 2 1957    Africa 264837738
#> 3 1962    Africa 296516865
#> 4 1967    Africa 335289489
#> 5 1972    Africa 379879541
#> 6 1977    Africa 433061021

ggplot(data = pop_growth_cont,
       aes(y = pop/1e6, x = year, colour = continent, fill = continent)) +
  geom_line() +
  geom_point()
```



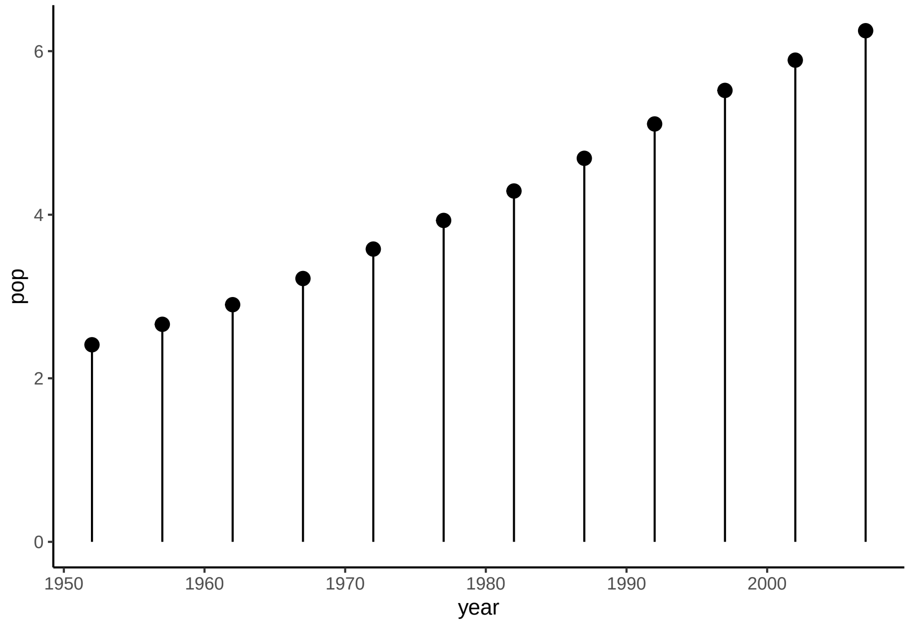

```r
ggplot(data = pop_growth_cont, aes(y = pop, x = year, colour = continent, fill = continent)) +
  geom_area() +
  scale_colour_brewer(palette = "Dark2") +
  theme_classic()
```



6.16.11 Lollipop plot

By combining the functions `geom_segment()` and `geom_point()`, we can produce a lollipop plot.

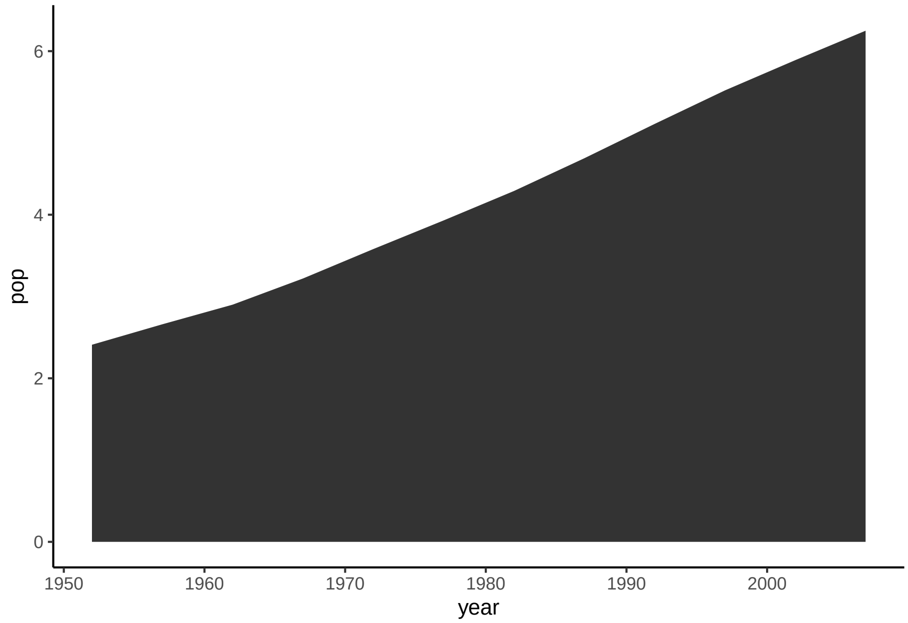
```
ggplot(data = pop_growth, aes(y = pop, x = year)) +
  geom_segment(aes(yend = 0, xend = year)) +
  geom_point(aes(y = pop, x = year), size = 3) +
  theme_classic()
```



6.16.12 Area plot

The function `geom_area()` is used to create an area plot.

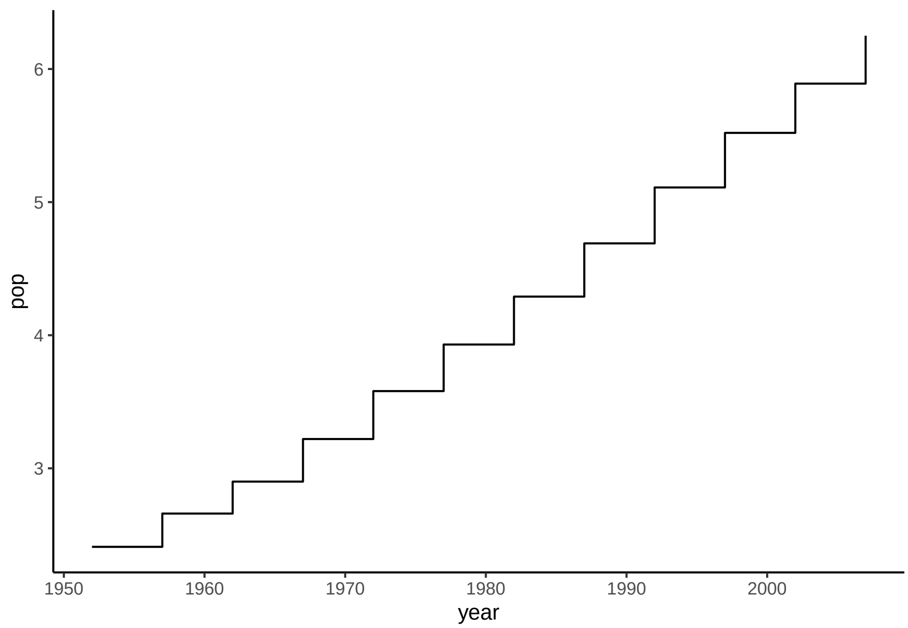
```
ggplot(data = pop_growth, aes(y = pop, x = year)) +
  geom_area() +
  theme_classic()
```



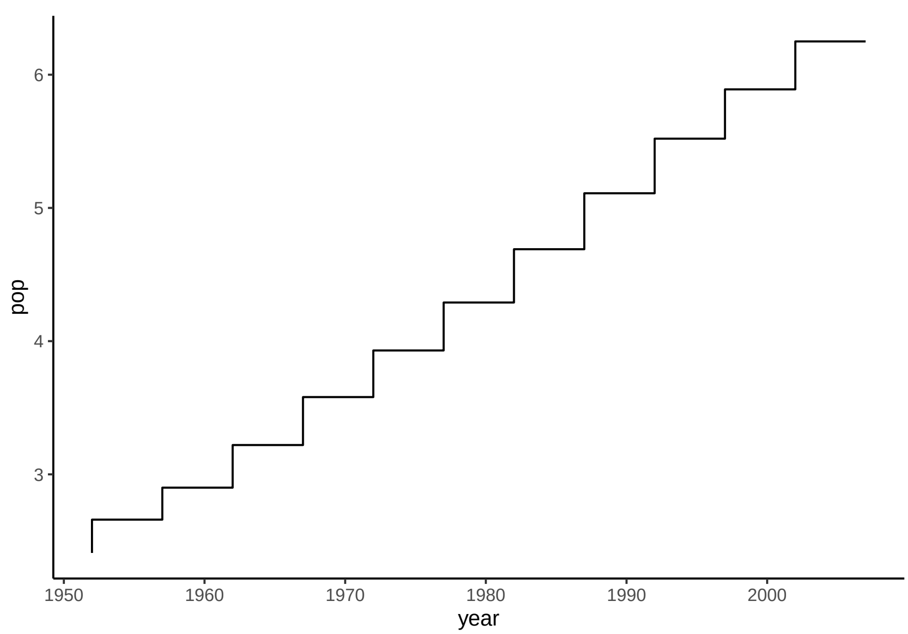
6.16.13 Step plot

The function `geom_step()` is used to create a step plot with the argument direction indicating the direction of the plot.

```
ggplot(data = pop_growth, aes(y = pop, x = year)) +
  geom_step(aes(y = pop, x = year)) +
  theme_classic()
```




```
# vh (vertical then horizontal)
ggplot(data = pop_growth, aes(y = pop, x = year)) +
  geom_step(aes(y = pop, x = year), direction = 'vh') +
  theme_classic()
```



Chapter 7

Bioconductor

Chapter 8

RNA-Seq (an example)

Chapter 9

Summary

9.1 RMarkdown

With R markdown, it is easy to reproduce not only the analysis used, but also the entire report. The advantage of using R markdown (versus a script) is that you can combine computation with explanation. In other words, you can weave the outputs of your R code, like figures and tables, with text to create a report.

	RMarkdown	R script
File extension	.Rmd	.R
File contents	R code + Markdown text + YAML header	R code
Reproducibility	analysis + entire report	only the analysis
Output format	PDF, HTML, Word DOCX	-

9.2 Advanced data manipulation

	Base R	Tidyverse R
Most used function	`[]`	`%>%`
Import	`read.table()` `rio::import()` Link	`readr::read_delim()` Link
Export	`write.table()` `rio::export()` Link	`readr::write_delim()` Link
Inspecting dataset	`str()` Link	`dplyr::glimpse()` Link
Working with factors	`factor()` Link	`forcats::fct_infreq()` Link

	Base R	Tidyverse R
Working with strings	`paste()` Link	`stringr::str_c()` Link
Working with column names	`names()` Link	`dplyr::rename()` Link
Working with row names	`rownames()` Link	`tibble::rowid_to_column()` Link
Filtering columns	`[]` Link	`dplyr::select()` Link
Filtering rows	`[]` Link	`dplyr::filter()` Link
Sorting rows	`[]` Link	`dplyr::arrange()` Link
Changing your data	`cut()` Link	`dplyr::mutate()` Link
Summarising data	`aggregate()` Link	`dplyr::summarise()` Link
Combining datasets	`merge()` Link	`dplyr::left_join()` Link
Reshaping data	`reshape2::melt()` `reshape2::dcast()` Link	`tidyr::pivot_longer()` `tidyr::pivot_wider()` Link

9.3 Modern graphics in R - ggplot2

The grammar of graphics lies at the heart of ggplot2 and also lies at the heart of how we define our data visualizations (Wilkinson, 2005).

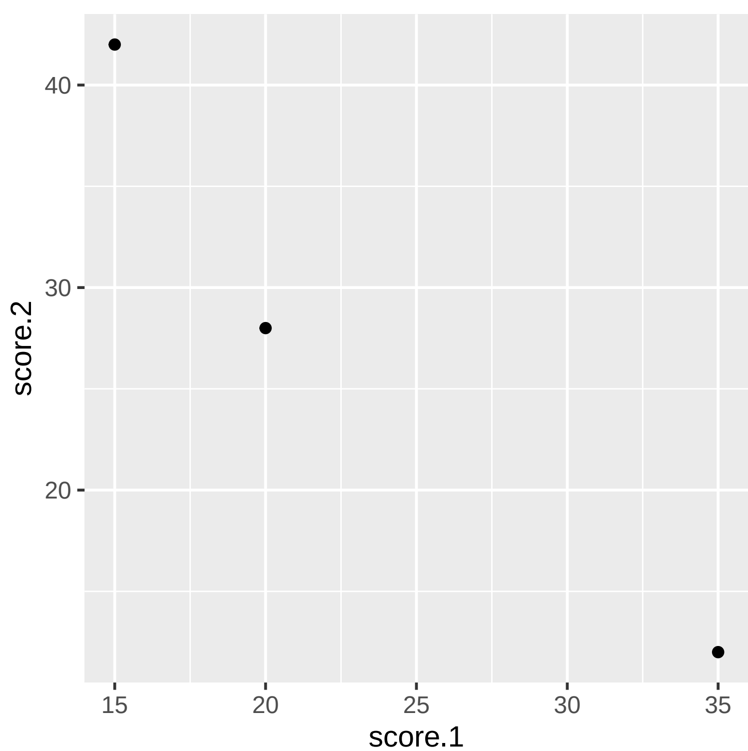
Table 9.3: The Grammar of Graphics

Component	Description
Data	Raw data that we'd like to visualize
Geometries	Shapes that we use to visualize
Aesthetics	Properties of geometries (size, color etc.)
Mapping	Mapping between data and aesthetics

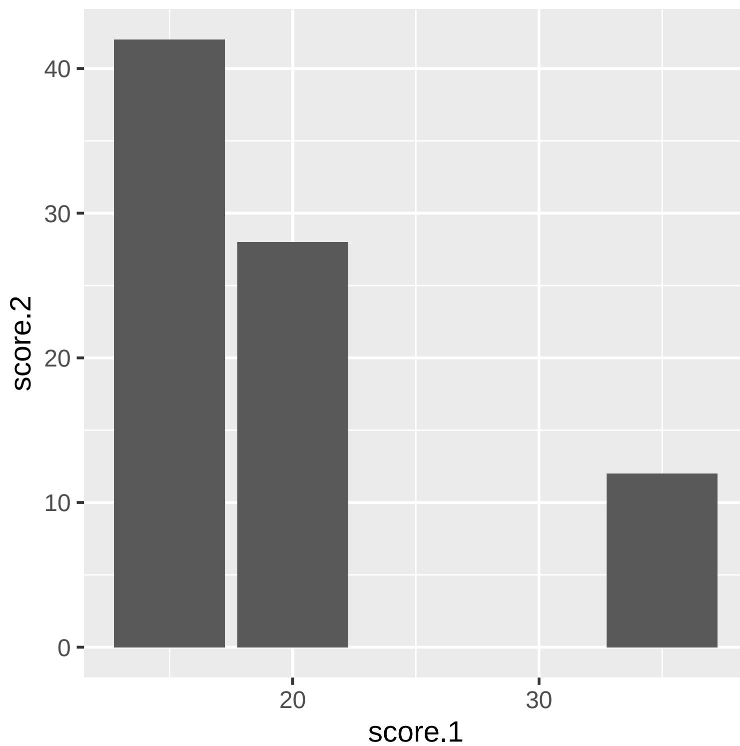
```
library(tidyverse)
# a tibble for data, 3 rows, 4 columns
d.tbl <- tribble(
  ~group, ~score.1, ~score.2, ~score.3,
  "AA", 15, 42, 12,
  "BB", 20, 28, 18,
```

```
  "CC", 35, 12, 21
)
```

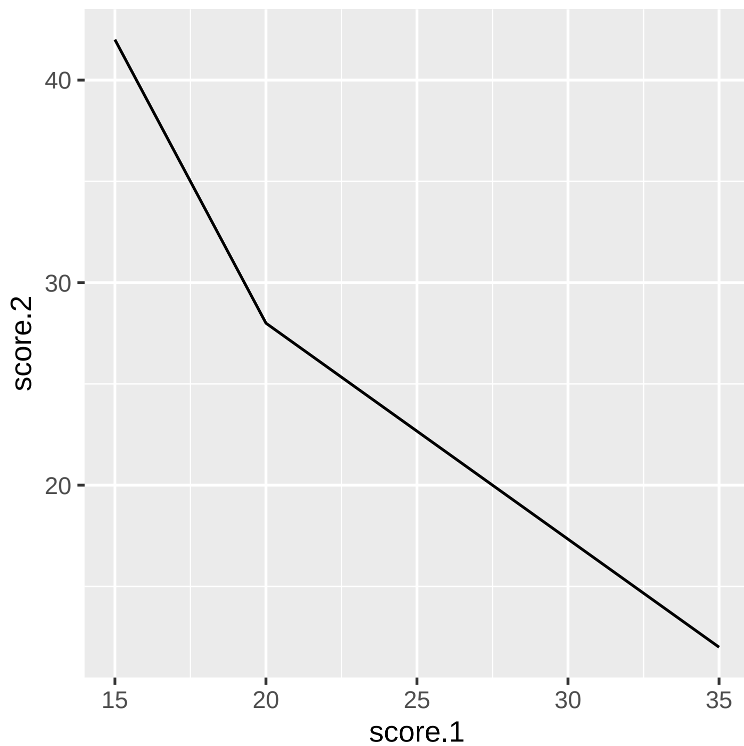
```
# Scatterplot
#   Data: d.tbl
#   Geometry: point
#   Aesthetics: x, y
#   Mapping: x=score.1, y=score.2
ggplot(data=d.tbl, mapping=aes(x=score.1, y=score.2)) + geom_point()
```



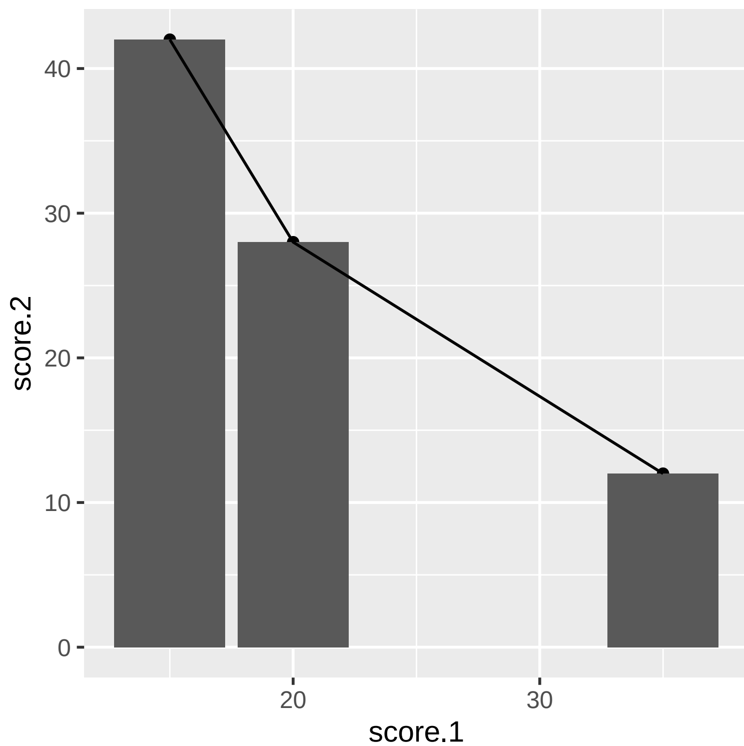

```
# Column Graph
#   Data: d.tbl
#   Geometry: column
#   Aesthetics: x, y
#   Mapping: x=score.1, y=score.2
ggplot(data=d.tbl, mapping=aes(x=score.1, y=score.2)) + geom_col()
```



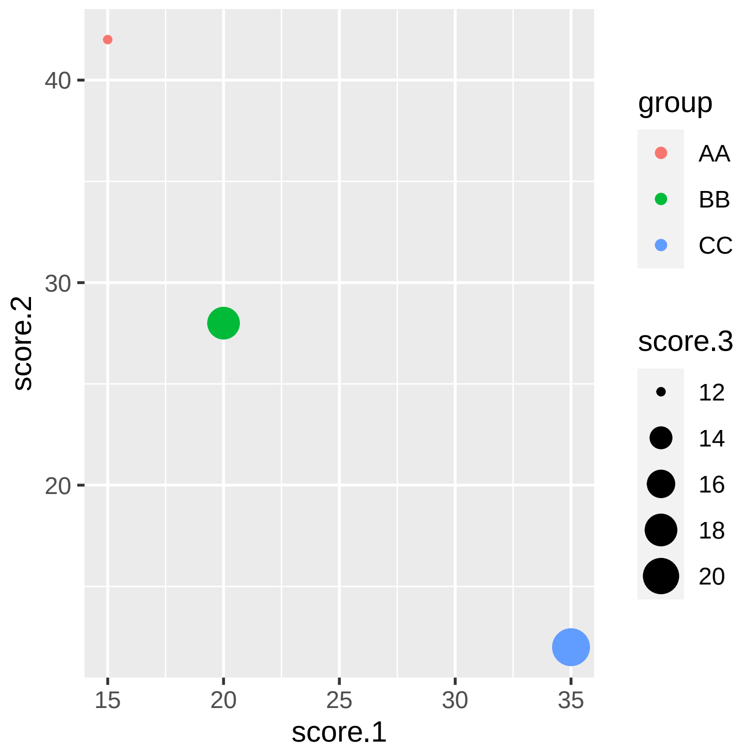

```
# Line Graph
#   Data: d.tbl
#   Geometry: line
#   Aesthetics: x, y
#   Mapping: x=score.1, y=score.2
ggplot(data=d.tbl, mapping=aes(x=score.1, y=score.2)) + geom_line()
```




```
# all in one
ggplot(data=d.tbl, mapping=aes(x=score.1, y=score.2)) +
  geom_point() + geom_col() + geom_line()
```




```
# Scatterplot
#   Data: d.tbl
#   Geometry: point
#   Aesthetics: x, y, size, color
#   Mapping: x=score.1, y=score.2, size=score.3, color=group
ggplot(data=d.tbl,
       mapping=aes(x=score.1, y=score.2, size=score.3, color=group)) +
  geom_point()
```




```
# Column Graph
#   Data: d.tbl
#   Geometry: column
#   Aesthetics: x, y, fill
#   Mapping: x=score.1, y=score.2, fill=score.3
ggplot(data=d.tbl, mapping=aes(x=score.1, y=score.2, fill=group)) +
  geom_col()
```

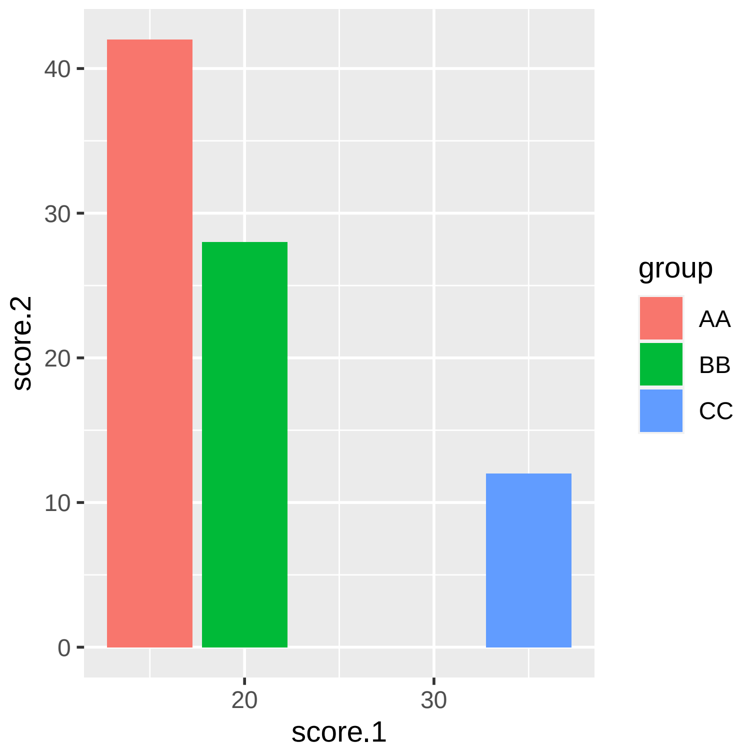


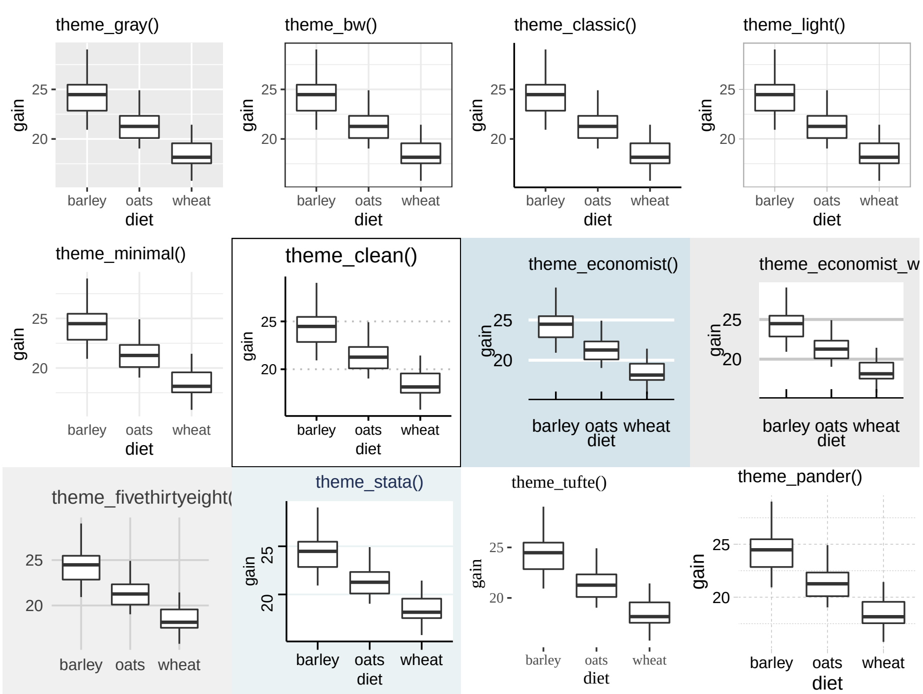
Table 9.4: Geometries with required and optional aesthetics.

Geometry	Required aesthetics	Optional aesthetics
`geom_abline()`	`slope`, `intercept`	`alpha`, `color`, `linetype`, `size`
`geom_hline()`	`yintercept`	`alpha`, `color`, `linetype`, `size`
`geom_vline()`	`xintercept`	`alpha`, `color`, `linetype`, `size`
`geom_area()`	`x`, `ymin`, `ymax`	`alpha`, `colour`, `fill`, `group`, `linetype`, `size`
`geom_col()`	`x`, `y`	`alpha`, `colour`, `fill`, `group`, `linetype`, `size`
`geom_bar()`	`x`, `y`	`alpha`, `colour`, `fill`, `group`, `linetype`, `size`
`geom_boxplot()`	`x`, (`lower`, `middle`, `upper`, `ymax`, `ymin`)	`alpha`, `color`, `fill`, `group`, `linetype`, `shape`, `size`, `weight`
`geom_density()`	`x`, `y`	`alpha`, `color`, `fill`, `group`, `linetype`, `size`, `weight`
`geom_dotplot()`	`x`, `y`	`alpha`, `color`, `fill`, `group`, `linetype`, `stroke`
`geom_histogram()`		`alpha`, `color`, `fill`, `linetype`, `size`, `weight`

Geometry	Required aesthetics	Optional aesthetics
geom_jitter()	x, y	alpha, color, fill, shape, size
geom_line()	x, y	alpha, color, linetype, size
geom_point()	x, y	alpha, color, fill, shape, size
geom_ribbon()	x, ymax, ymin	alpha, color, fill, linetype, size
geom_smooth()	x, y	alpha, color, fill, linetype, size, weight
geom_text()	label, x, y	alpha, angle, color, family, fontface, hjust, lineheight, size, vjust

9.4 Type of plots

9.5 Themes



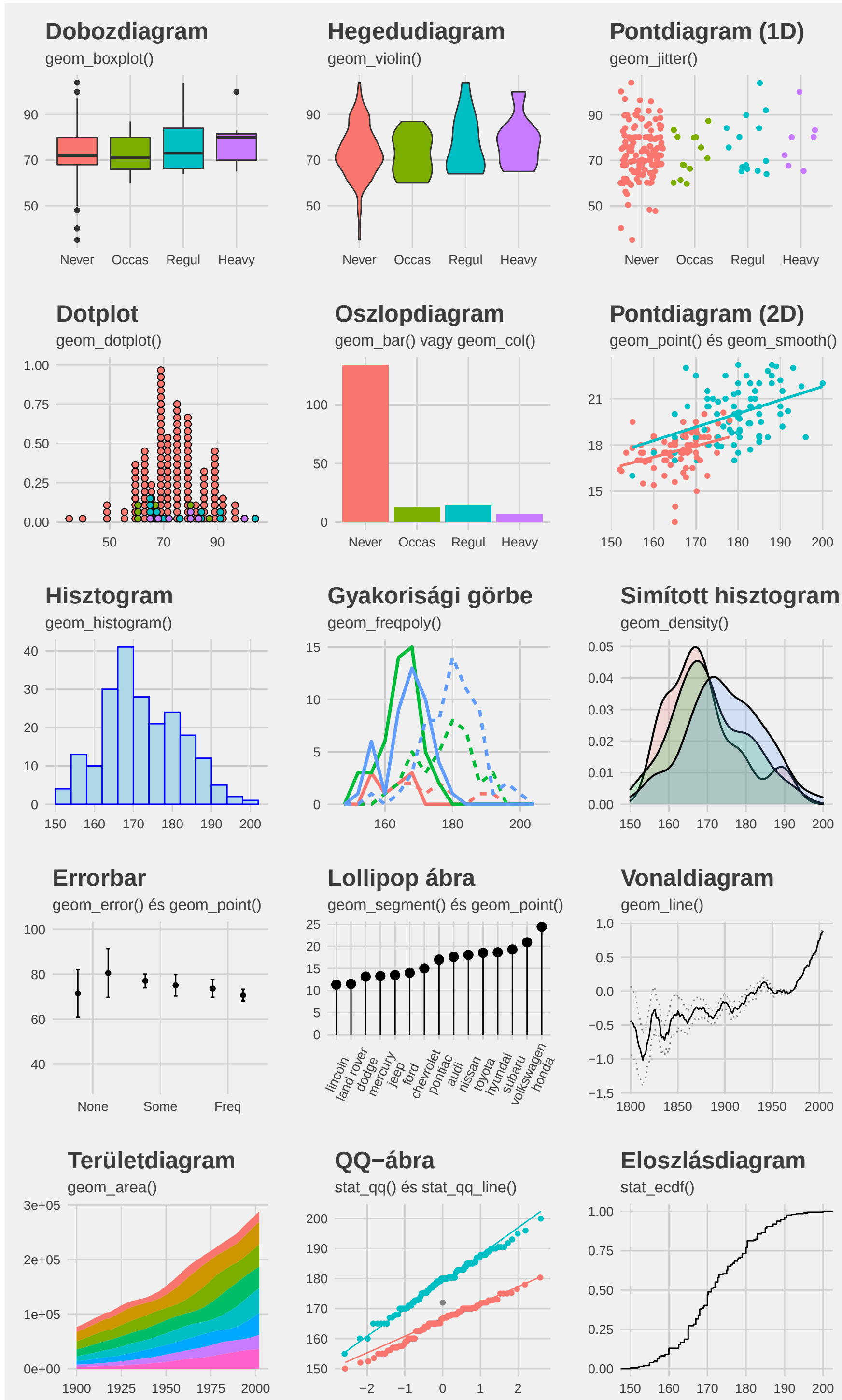


Figure 9.1: Type of plots and geometry

Bibliography

Wilkinson, L. (2005). *The Grammar of Graphics (Statistics and Computing)*. Springer-Verlag, Berlin, Heidelberg.