Introduction to R

Dr Mehtap Hisarciklilar & Dr Robert Riegler

2025-01-20

Table of contents

Welcome!

This workbook is created for the Introduction to R sessions at Coventry University[1].

organised by the School of EFA Economics Lab team in collaboration with the British Council funded project "Strengthening Pathways into Employment: Female Students in Economics and Finance" at Coventry University.

It is written using Quarto on RStudio by

Dr Mehtap Hisarciklilar, Centre for Financial and Corporate Integrity

Dr Robert Riegler, Aston University

[1]The workshop series are organised by the School of EFA Economics Lab team in collaboration with the British Council funded project "Strengthening Pathways into Employment: Female Students in Economics and Finance".

Part I

Session 1 Introduction to Data Analysis

1 Introduction to R

1.1 R, R Studio and Quarto

R is a very powerful statistical software that is becoming increasingly popular. Being able to do data analysis using R will very likely increase your employability.

Warning: R is not like other apps that you have used! It requires coding. You will need to practice regularly. There will be a lot of struggle, but the result is worth it.

We list below the apps that you will need to work with during the sessions. You may install these on your computers. Alternatively, you may use Coventry University's Appsanywhere platform to get access. But you will find working with the app easier if it is locally installed.

- We will be using **R** as the statistical analysis tool. For R documentations, support and download links, visit the R Project for Statistical Computing. R is freely available for Linux, MacOS and Windows. **Please download the version that matches your computer's operating system.**
- To facilitate your work with R, we highly recommend to download and install the integrated development environment (IDE) **RStudio Desktop** from posit. This platform will make it easier for you to write and run R code.
- A final package that we highly recommend you to install is a publishing system, **Quarto**. You may use Quarto to produce documents in various formats (such as HTML, MS Word, PDF, PowerPoint, etc) while integrating your R code and output. You will easily have the option to change the format of your output as you desire. We will be using Quarto to produce documents in the third session of the series. Please visit Quarto for further information and download.
  - Once you download Quarto, you will have access to it through RStudio.

RStudio has four main windows, that often have more than just one purpose. Figure 1.1 provides a brief description of each RStudio window. We will use all of them during the sessions, but the most important ones will be the console and the editor pane.

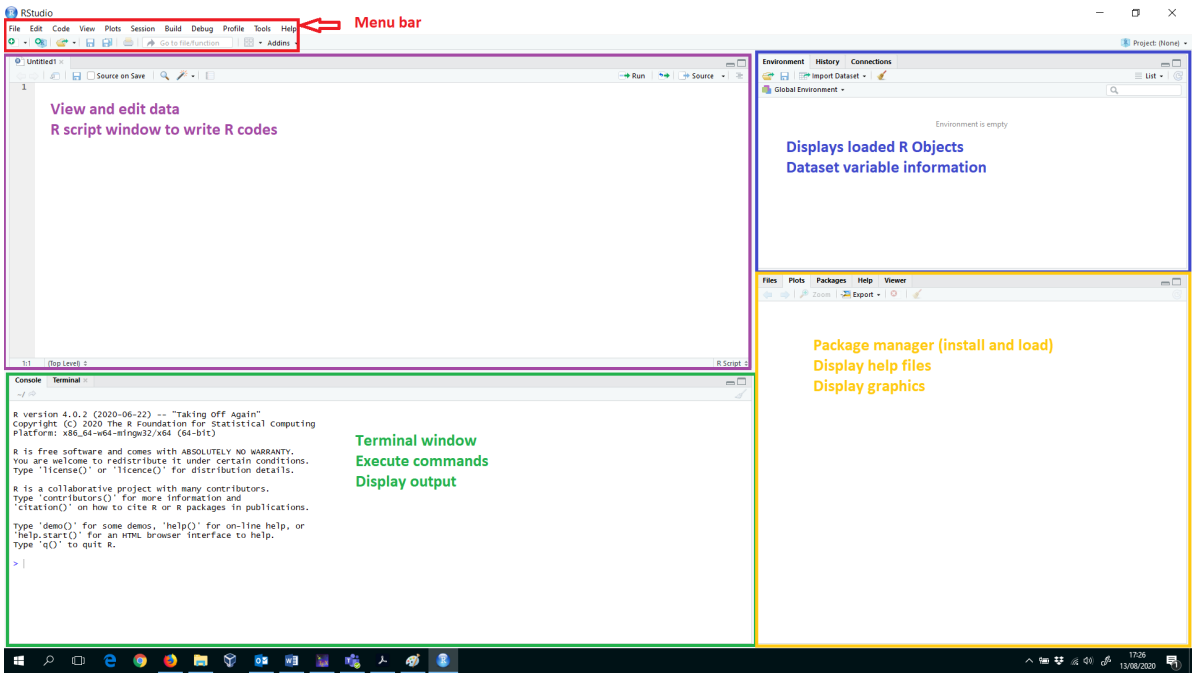


Figure 1.1: RStudio windows and their functions

1.2 File Organisation

- Create a folder for this workshop. This folder should include all material you download from shared OneDrive folder. Group files in sub-folders in a way that you can locate them easily. So for example, `Introduction-to-R` may be the name of the folder and then you may have sub-folders such as `data`, `R-scripts`, etc.
- If you are using the computers in the lab, it may be best if you create a folder on your OneDrive account as you can easily access this at home and on-campus.
- Before working on the data, set your working directory. R will save all files in there and, if you want to open a dataset, R will also look in there first. Select the folder you have created for R workshops.
- Use `setwd(the_address_you_would_like_to_locate_your_work)` in the console to choose your work directory. You may alternatively do this through the menu:

Session –> Set Working Directory –> Choose Directory

You will see the console printing this action, which may help you to remember how to use the console next time.

- If you are unsure of in which folder your work is, type `getwd()` in the console and R will print the current location you are at.

1.3 Getting Help

If you should ever struggle with some of R's commands, a look into R's help-files can be very helpful. To access the help file, you have to type into the console window `?` and then the command name. For example, if you want to know more about the command `getwd()`, type the following:

```
?getwd()
```

2 Basics of R

2.1 Using R as a calculator

You may use R as a calculator. Some examples are given below.

```
# Addition
5 + 4
```

```
[1] 9
```

```
# Subtraction
5 - 4
```

```
[1] 1
```

```
# Multiplication
3 * 6
```

```
[1] 18
```

```
# Division
10 / 2
```

```
[1] 5
```

```
# Exponents
2^3
```

```
[1] 8
```

```
# Modulo
5 %% 2
```

```
[1] 1
```

2.1.1 Basic Operators

Operator	Description
Arithmetic	
+	Addition
-	Subtraction
*	Multiplication
/	Division
^ or **	Exponential
%%	Modulus
% / %	Integer Division
Logic	
<	Less than
<=	Less than or equal to
>	Greater than
>=	Greater than or equal to
==	Exactly equal to
!=	Not equal to
!x	Not x
x | y	x OR y
x & y	x AND y

2.1.2 Order of operators

- Parenthesis
- Multiplication / division
- Addition / subtraction
- Multiplication has the same importance as division. Similarly, addition and subtraction are at the same level. When we need to decide between the two, we apply the operation that shows first from the left to the right.
- Use of parentheses makes it easier to perform the correct operation

- Can you guess the result of the following operation?

  $-$ 8 / 2 * (2 + 2)

```
8 / 2 * ( 2 + 2)
```

```
[1] 16
```

```
8 / 2 * 2 + 2
```

```
[1] 10
```

```
100 * 2 + 50 / 2
```

```
[1] 225
```

```
(100 * 2) + (50 / 2)
```

```
[1] 225
```

2.2 Storing information in objects

R lets you save data by storing it inside an R `object`. An object is a name that you can use to call up stored data.

```
a <- 5

a
```

```
[1] 5
```

```
a + 2
```

```
[1] 7
```

In the example above, we store value of 5 under object `a`. We then call the value stored under `a` and sum it with 2.

Note the use of `<` together with `-` . This representation (`<-`) resembles a backward pointing arrow, and it assigns the value 2 to the object `a`.

```r
b_vector <- 1:6
b_vector
```

```
[1] 1 2 3 4 5 6
```

```
## [1] 1 2 3 4 5 6
```

In the above example, we create a vector, whose elements are numbers from 1 to 6 and store it under b_vector.

When you create an object, the object will appear in the environment pane of RStudio (on the top right-hand-side of the R screen). This pane will show you all of the objects you've created since opening RStudio.

2.2.1 Naming of objects

Note the following;

- An object name cannot start with a number (for example, `2var` or `2_var`)
- A name cannot use some special symbols, like `^`, `!`, `$`, `@`, `+`, `-`, `/`, or `*` . You may use `_`
- R is case-sensitive, so `name` and `Name` will refer to different objects
- R will overwrite any previous information stored in an object without asking your confirmation. So, be careful while making changes.
- You can see which object names you have already used by calling the function `ls`:

```r
ls()
```

```
[1] "a"        "b_vector"
```

```
## [1] "a"        "b_vector"
```

2.2.2 Naming conventions

You may see the following styles for naming of variables:

- Camel case

Camel case variable naming is common in Javascipt. However, it is considered as bad practise in R. Try to avoid this kind of naming.

```
bankAccount = 100
```

- Use of dots

dot is used in variable names by many R users. However, try to avoid this too because base R uses dots in function names (`contrib.url()`) and class names (`data.frame`). Avoiding dot in your variable names will help you avoid confusion, particularly in the initial stages of your learning!

```
bank.account = 100
```

- Snake case

Use of snake case is considered to be good practice. Try to follow this approach.

```
bank_account = 100
```

Note that you may find different users of R having a preference towards different styles. The recommendations above are from the "Tidyverse style guide", which is available from https://style.tidyverse.org.

Start your variable names with a lower case and reserve the capital letter start for function names!

2.2.3 Removing objects

You will see that the `Environment` window can quickly get over-crowded while working interactively. You may remove the objects that you no longer need. by `rm(object_name )`

```
rm(a)
```

If you have too many objects piled up and you would like to remove them all, then you may type `rm(list = ls())`. This will fully clear your environment.

```
rm(list = ls())
```

2.2.4 Example of using variables

Let us calculate the module mark for a student who got 65% from coursework and 53% from exam. The weights for the coursework and exam are, respectively, 25% and 75%.

```
# let's calculate module mark for a student
coursework <- 65
exam <- 53
module_mark <- coursework * 0.25 + exam * 0.75

print(module_mark)
```

```
[1] 56
```

2.3 Datatypes in R

2.3.1 Numeric

Decimal numbers and integers are part of the numeric class in R.

2.3.1.1 Decimal (floating point values)

```
decimal_number <- 2.2
```

2.3.1.2 Integer

```
i <- 5
```

2.3.2 Logical

Boolean values (TRUE and FALSE) are part of the logical class in R. These are written in capital letters.

```
t <- TRUE
f <- FALSE
```

```
t
```

```
[1] TRUE
```

```
f
```

```
[1] FALSE
```

2.3.3 Characters

Text (string) values are known as characters in R. You may use single or double quotation to create a text (string).

```
message <- "hello all!"
print(message)
```

```
[1] "hello all!"
```

```
an_other_message <- 'how are you?'
print(an_other_message)
```

```
[1] "how are you?"
```

2.3.4 Checking data type classes

We can use the `class()` function to check the data type of a variable:

```
class(decimal_number)
```

```
[1] "numeric"
```

```
class(i)
```

```
[1] "numeric"
```

```
class(t)
```

```
[1] "logical"
```

```
class(f)
```

```
[1] "logical"
```

```
class(message)
```

```
[1] "character"
```

2.4 Scripts

You can create a draft of your code as you go by using an R *script*. An R script is just a plain text file that you save R code in. You can open an R script in RStudio using the menu bar:

File –> New File –> R Script

We will write and edit R code in a script. This will help create a reproducible record of your work. When you're finished for the day, you can save your script and then use it to rerun your entire analysis the next day.

To save a script, click the scripts pane, and then go to *File –> Save As* in the menu bar.

- You can automatically execute a line of code in a script by clicking the Run button on the top right of the pane. R will run whichever line of code your cursor is on.
- If you have a whole section highlighted, R will run the highlighted code.
- You can run the entire script by clicking the Source button.
- You can use Control + Return in your keyboard as a shortcut for the Run button. On Macs, that would be Command + Return.

2.5 R Projects

R projects make it easier to save all your work and continue from where you left next time. They are portable. Please check the information here.

3 Importing and Viewing Data

3.1 Example: House Price Data

The data that we will be using in this session come from R's `wooldridge` package, which includes 115 data sets from "Introductory Econometrics: A Modern Approach, 7e" by Jeffrey M. Wooldridge".

We will start by using `hprice2` data, which includes information on 506 communities in the US Boston area. Below table provides a list of variables that we will use:

Variable	Description
price	Median housing price, $
crime	Crimes committed per capita
nox	Nitrogen Oxide in the air, in parts per million
stratio	Average student-teacher ratio of schools in the community
rooms	Average number of rooms in houses in the community
lprice	Logarithm of price

3.2 Packages and libraries

In order to access specialised data analysis tools in R, we will need to install some R packages.

"An R **package** is a collection of functions, data, and documentation that extends the capabilities of base R.

We will start by installing the `wooldridge` package. We require this package to access the Wooldridge data sets, mentioned above.

```
# install.packages("wooldridge")
```

To install `wooldridge` package, type the above line of code in the console (without the `#`), and then press enter to run it. R will download the packages from CRAN and install them on to your computer.

Once installed, you may use this package after loading it with the `library()` function.

```
library(wooldridge)
```

3.3 Open data as an R object

The `hprice2` data that we will be using already comes in R format within the `wooldridge` package. We can call it to our environment by using the `data()` function.

```
data("hprice2")
```

Observe that `hprice2` is now added as an object to your environment. We can use the `View` function to view it (Note the capital V in `View`).Type the below line without the `#` in your console.

```
# View(hprice2)
```

3.4 Import Excel data into R

You may download the `hprice2`data in Excel format here. Rather than left-clicking on the link, right-click and choose *Download Linked File*.

Most data we work with are initially in Excel or in text (such as csv) format. Importing data is one of the rare R commands that you can proceed with using the menu. You may find the `Import Dataset` button on the top right window and also under File menu. The `hprice2` data is provided to you in Excel format. Try to import it using `Import Dataset` but this time, in the opening window, name the dataset as `df` under `Import Options` towards the left bottom.

You will see that R executes the following three lines of code:

```
library(readxl)
df <- read_excel("data/hprice2.xlsx")
# View(df)
```

The first line call the library required; the second line reads the Excel file and saves it under name `df` and the third line views the newly imported data.

You may use `head()` and `tail()` functions to view, respectively, the first few and the last few lines of the data.

```
# View the first few observations of the data
head(hprice2)
```

```
  price crime  nox rooms dist radial proptax stratio lowstat    lprice     lnox
1 24000 0.006 5.38  6.57 4.09      1    29.6    15.3    4.98 10.085809 1.682688
2 21599 0.027 4.69  6.42 4.97      2    24.2    17.8    9.14  9.980402 1.545433
3 34700 0.027 4.69  7.18 4.97      2    24.2    17.8    4.03 10.454495 1.545433
4 33400 0.032 4.58  7.00 6.06      3    22.2    18.7    2.94 10.416311 1.521699
5 36199 0.069 4.58  7.15 6.06      3    22.2    18.7    5.33 10.496787 1.521699
6 28701 0.030 4.58  6.43 6.06      3    22.2    18.7    5.21 10.264688 1.521699
  lproptax
1 5.690360
2 5.488938
3 5.488938
4 5.402678
5 5.402678
6 5.402678
```

```
# View the last few observations of the data
tail(hprice2)
```

```
    price crime  nox rooms dist radial proptax stratio lowstat    lprice
501 16800 0.224 5.85  6.03 2.50      6    39.1    19.2   14.33  9.729135
502 22400 0.063 5.73  6.59 2.48      1    27.3    21.0    9.67 10.016816
503 20600 0.045 5.73  6.12 2.29      1    27.3    21.0    9.08  9.933046
504 23899 0.061 5.73  6.98 2.17      1    27.3    21.0    5.64 10.081592
505 22000 0.110 5.73  6.79 2.39      1    27.3    21.0    6.48  9.998797
506 11900 0.047 5.73  6.03 2.51      1    27.3    21.0    7.88  9.384294
        lnox lproptax
501 1.766442 5.968708
502 1.745715 5.609472
503 1.745715 5.609472
504 1.745715 5.609472
505 1.745715 5.609472
506 1.745715 5.609472
```

4 Calculating Summary Statistics

- We may use the `mean()` function to find the average value of a variable. For example, below, we find the average housing price:

```
# Calculate the average price
mean(hprice2$price)
```

```
[1] 22511.51
```

- Similarly `median()` function could be used to find the median.

```
# Calculate the median price
median(hprice2$price)
```

```
[1] 21200
```

- Use `sd()` for standard deviation

```
# Calculate the standard deviation of price
sd(hprice2$price)
```

```
[1] 9208.856
```

You may wonder why we add `hprice2$` in front of every variable. The reason is that R can store more than one data frame, matrix, list, vector etc., at the same time, so the prefix `hprice2$` is necessary to avoid ambiguity and ensure that we are working with variables in the `hprice2` data. Think of `hprice2$` as an address where e.g. the variable `price` stays. If you have loaded another data frame that contains a `price` variable, R would know that we want to use the variable from the hprice2 data set and not from the other data frame. There are library packages that can facilitate the process, however, these are beyond the scope of this workshop.

In addition to mean and median, we may be interested with the minimum and maximum values. In that case, we can use the `summary()` function to ask for a full set of summary statistics. (Please note that `summary()` function does not report the standard deviation. You will need to use the `sd()` function for that).

```
# Summary statistics for price
summary(hprice2$price)
```

```
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
   5000   16850   21200   22512   24999   50001
```

Again, note the use of `hprice2$` above.

We can provide the name of the data set as an argument in the `summary()` function to get summary statistics for all variables in the data.

```
# summary statistics for all variables
summary(hprice2)
```

```
     price            crime              nox            rooms
 Min.   : 5000   Min.   : 0.0060   Min.   :3.85   Min.   :3.560
 1st Qu.:16850   1st Qu.: 0.0820   1st Qu.:4.49   1st Qu.:5.883
 Median :21200   Median : 0.2565   Median :5.38   Median :6.210
 Mean   :22512   Mean   : 3.6115   Mean   :5.55   Mean   :6.284
 3rd Qu.:24999   3rd Qu.: 3.6770   3rd Qu.:6.24   3rd Qu.:6.620
 Max.   :50001   Max.   :88.9760   Max.   :8.71   Max.   :8.780
      dist            radial          proptax         stratio
 Min.   : 1.130   Min.   : 1.000   Min.   :18.70   Min.   :12.60
 1st Qu.: 2.100   1st Qu.: 4.000   1st Qu.:27.90   1st Qu.:17.40
 Median : 3.210   Median : 5.000   Median :33.00   Median :19.10
 Mean   : 3.796   Mean   : 9.549   Mean   :40.82   Mean   :18.46
 3rd Qu.: 5.188   3rd Qu.:24.000   3rd Qu.:66.60   3rd Qu.:20.20
 Max.   :12.130   Max.   :24.000   Max.   :71.10   Max.   :22.00
    lowstat          lprice            lnox          lproptax
 Min.   : 1.730   Min.   : 8.517   Min.   :1.348   Min.   :5.231
 1st Qu.: 6.923   1st Qu.: 9.732   1st Qu.:1.502   1st Qu.:5.631
 Median :11.360   Median : 9.962   Median :1.683   Median :5.799
 Mean   :12.701   Mean   : 9.941   Mean   :1.693   Mean   :5.931
 3rd Qu.:17.058   3rd Qu.:10.127   3rd Qu.:1.831   3rd Qu.:6.501
 Max.   :39.070   Max.   :10.820   Max.   :2.164   Max.   :6.567
```

And finally, if we may want to have summary statistics only for a selection of variables, for example, `price`, `nox`, and `crime`.

```
# summary statistics for a selection of variables:
summary(hprice2[, c("price", "nox", "crime")])
```

```
     price            nox           crime
 Min.   : 5000   Min.   :3.85   Min.   : 0.0060
 1st Qu.:16850   1st Qu.:4.49   1st Qu.: 0.0820
 Median :21200   Median :5.38   Median : 0.2565
 Mean   :22512   Mean   :5.55   Mean   : 3.6115
 3rd Qu.:24999   3rd Qu.:6.24   3rd Qu.: 3.6770
 Max.   :50001   Max.   :8.71   Max.   :88.9760
```

We may calculate the covariance or correlation between the variables using the `cov()` and `cor()` functions:

```
# Covariance between price and crime
cov(hprice2$price, hprice2$crime)
```

```
[1] -30686.87
```

```
# Correlation between price and crime
cor(hprice2$price, hprice2$crime)
```

```
[1] -0.3879191
```

5 Labeling Variables

There are alternative packages which could be used to label variables. We will be using the `Hmisc` package. Remember to install this package and call the library by running the code below (without the `#`):

```
# install.packages("Hmisc") # to label variables
library(Hmisc)
```

```
Attaching package: 'Hmisc'
```

```
The following objects are masked from 'package:base':
```

```
    format.pval, units
```

We are ready to label our variables:

```
# label variables
label(hprice2$price) <- "Median housing price, $"
label(hprice2$crime) <- "Crimes committed per capita"
label(hprice2$nox) <- "Nitrogen Oxide in the air, in parts per million"
label(hprice2$stratio) <- "Average student-teacher ratio of schools in the community"
label(hprice2$rooms) <- "Average number of rooms in houses in the community"
```

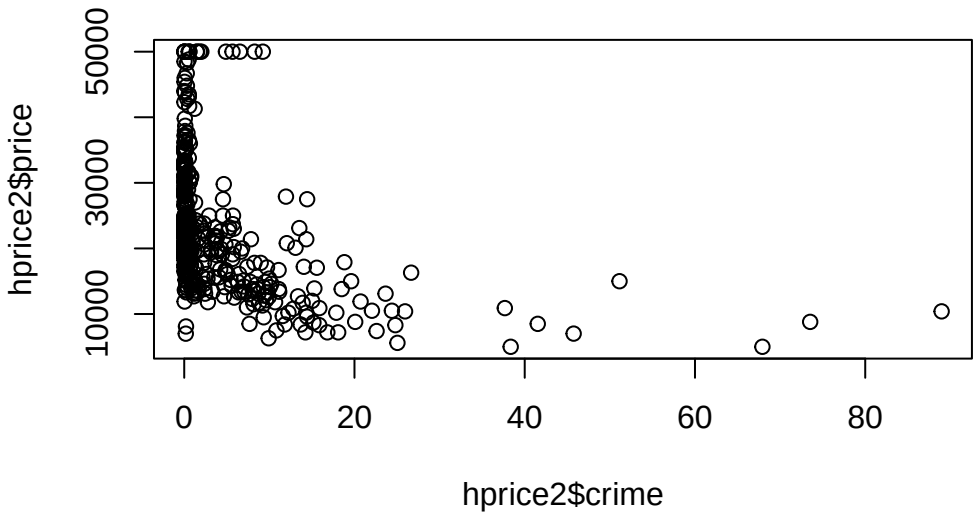
6 Basic Plots of Data

In Session 2 of this workshop, we will be using `ggplot2` for data visualisation in R, which produces better-looking plots. For the moment, below are a few examples of what we can do with base R (with base R, we refer to the operations of R that could be done without installing or calling additional libraries).

Scatter plot

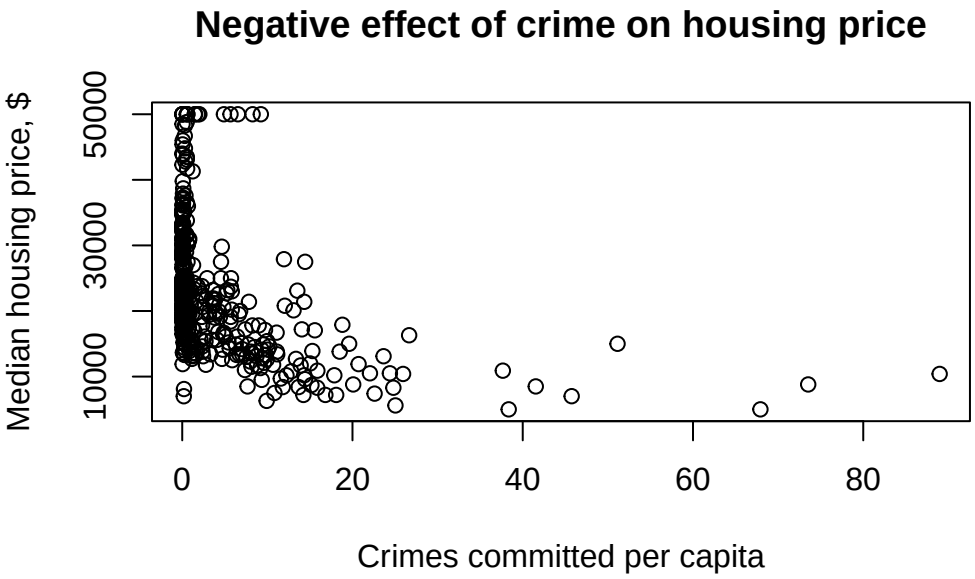
We may use scatter plots to depict the relationship between two variables. In the example below, we are depicting the relationship between housing price and crime. You may read the `~` between the two variables as *approximated by*. So here, price is approximated by crime.

```
# scatterplot showing the relationship between price and crimes
plot(hprice2$price ~ hprice2$crime)
```



We may add more informative titles to our axes and also provide a general title for the plot:

```
# scatterplot showing the relationship between price and crimes
plot(hprice2$price ~ hprice2$crime,
     xlab = "Crimes committed per capita",
     ylab = "Median housing price, $",
     main = "Negative effect of crime on housing price")
```



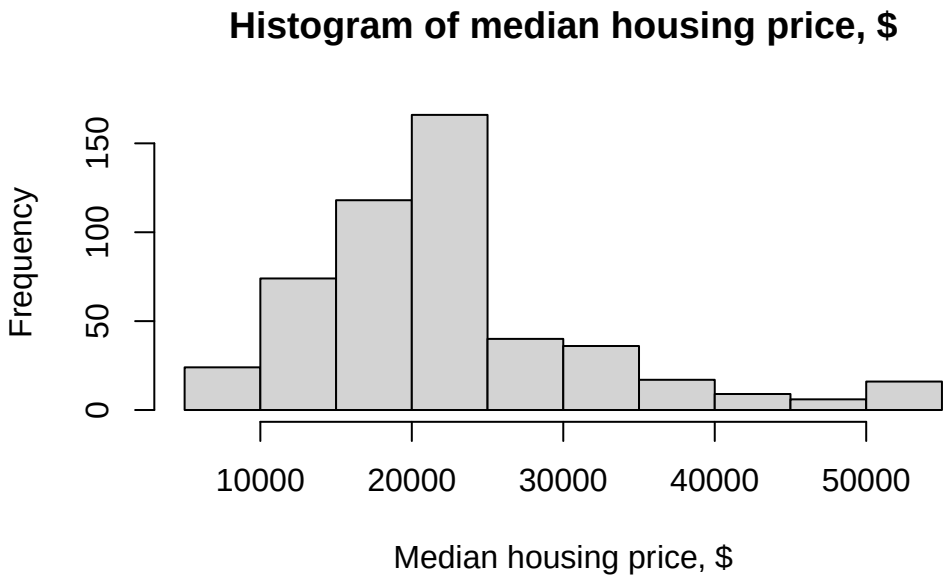
As expected, there is a negative relationship between the crime rates in a location and the median house prices in that location.

In the following pages, we will be estimating regresions to model the houseing price with a set of independent variables. Before moving on to regression analysis, it is good practice to get to know our dependent variable. A histogram will reveal its distribution.

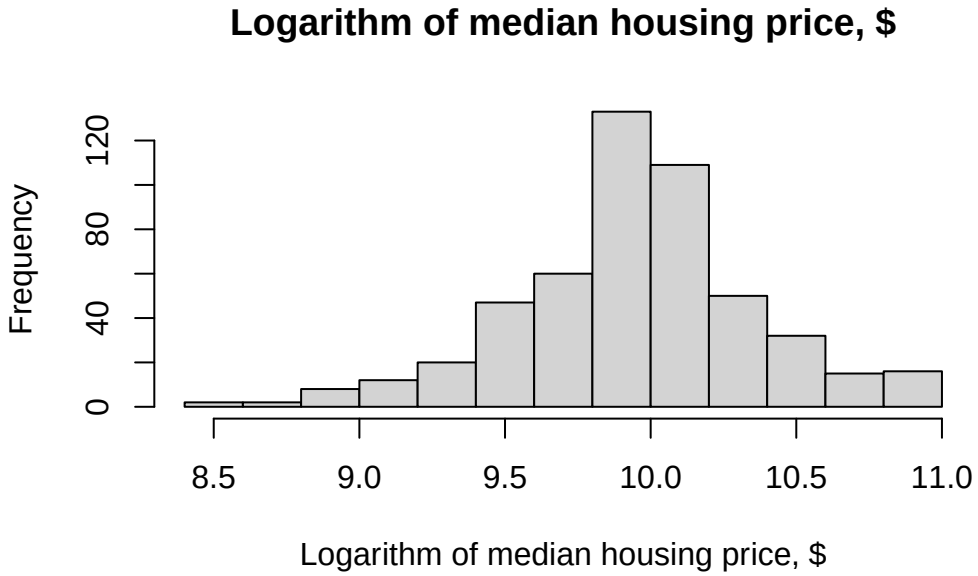
Histogram

Below, we provide histograms of `price` and `lprice` (logarithmic price). Which one should we use in our regression?

```
# Histogram of price and logarithmic price
hist(hprice2$price,
     xlab = "Median housing price, $",
     main = "Histogram of median housing price, $")
```




```
hist(hprice2$lprice,
     xlab = "Logarithm of median housing price, $",
     main = "Logarithm of median housing price, $")
```



7 OLS Regression

We will start with a simple regression relating logarithmic house prices with crime. Below, we will name and store this regression as `model_1`. Note that you may provide any name as you wish, as long as it does not clash with the naming conventions. For example, while `model_1` is OK, `1_model` is not possible.

```
# Regression
model_1 <- lm(lprice ~ crime, data = hprice2)
summary(model_1)
```

```
Call:
lm(formula = lprice ~ crime, data = hprice2)

Residuals:
    Min      1Q  Median      3Q     Max
-1.1736 -0.2041 -0.0301  0.1750  1.4538

Coefficients:
             Estimate Std. Error t value Pr(>|t|)
(Intercept) 10.031818   0.016786  597.63   <2e-16 ***
crime       -0.025131   0.001803  -13.94   <2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 0.348 on 504 degrees of freedom
Multiple R-squared:  0.2783,    Adjusted R-squared:  0.2768
F-statistic: 194.3 on 1 and 504 DF,  p-value: < 2.2e-16
```

- The R function we use for linear regression is `lm`, which stands for *linear model*.
- The regression equation we are estimating is given by `lprice ~ crime`, where `price` is approximated by `crime`.
- `data = hprice2` refers to the hprice2 data we are using

- `summary(model_1)` is a separate line of command, which asks R to provide the summary estimation results.

We may add another independent variable in the model by using a + sign. This time it is saved under name `model_2`.

```
model_2 <- lm(lprice ~ crime + nox, data = hprice2)
summary(model_2)
```

```

Call:
lm(formula = lprice ~ crime + nox, data = hprice2)

Residuals:
     Min       1Q   Median       3Q      Max
-1.08309 -0.18979 -0.05017  0.16518  1.10231

Coefficients:
             Estimate Std. Error t value Pr(>|t|)
(Intercept) 10.689699   0.074846 142.822   <2e-16 ***
crime       -0.018140   0.001847  -9.822   <2e-16 ***
nox         -0.123091   0.013696  -8.987   <2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 0.3234 on 503 degrees of freedom
Multiple R-squared:  0.3781,    Adjusted R-squared:  0.3756
F-statistic: 152.9 on 2 and 503 DF,  p-value: < 2.2e-16
```

Let us add two mode independent variables under name `model_3`:

```
model_3 <- lm(lprice ~ crime + nox + stratio + rooms,
              data = hprice2)
summary(model_3)
```

```

Call:
lm(formula = lprice ~ crime + nox + stratio + rooms, data = hprice2)

Residuals:
     Min       1Q   Median       3Q      Max
```

```
-0.87500 -0.12789  0.00664  0.11087  1.35514

Coefficients:
             Estimate Std. Error t value Pr(>|t|)
(Intercept)  9.653725   0.187981  51.355  < 2e-16 ***
crime       -0.013143   0.001450  -9.065  < 2e-16 ***
nox         -0.078260   0.010751  -7.279 1.31e-12 ***
stratio     -0.042702   0.005568  -7.670 9.04e-14 ***
rooms        0.247830   0.017289  14.334  < 2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 0.2465 on 501 degrees of freedom
Multiple R-squared:   0.64, Adjusted R-squared:  0.6371
F-statistic: 222.6 on 4 and 501 DF,  p-value: < 2.2e-16
```

We may want to save the residuals or predictions after estimating an OLS model.

- Use `predict()` to store the predictions for the dependent variable.

Can you tell the difference between the two lines below?

```
price_hat <- predict(model_3)

hprice2$lprice_hat <- predict(model_3)
```

In the first one predictions are saved under a separate object of its own while the second line saves these as a variable in our `hprice2` data.

- Use `rediduals` to store the residuals of the model.

```
# Saving residuals of model_3
resid_3 <- residuals(model_3)
```

Let us view the newly created `lprice_hat` variable. We can do this by viewing the whole data set:

```
# View the newly created predicted price variable - whole data
#View(hprice2)
```

It is difficult to compare the actual and predicted value columns above. We may ask R to view only a selection of variable rather than the full data.

```
# Create a subset of variables and then view them
subset <- hprice2[, c("lprice", "lprice_hat")]
#View(subset)

# Combining the two lines under one
#View(hprice2[, c("lprice", "lprice_hat")])
```

Finally, we may use a summary table to present results of the three models together. We use the **stargazer** package for that.

```
#install.packages("stargazer")
library(stargazer)
```

```
Please cite as:

 Hlavac, Marek (2022). stargazer: Well-Formatted Regression and Summary Statistics Tables.

 R package version 5.2.3. https://CRAN.R-project.org/package=stargazer
```

```
# Summarising the three model estimates under one table
stargazer(model_1, model_2, model_3, type = "text")
```

```
================================================================================================
                                          Dependent variable:
                    ---------------------------------------------------------------------------
                                                  lprice
                              (1)                   (2)                    (3)
------------------------------------------------------------------------------------------------
crime                      -0.025***             -0.018***              -0.013***
                            (0.002)               (0.002)                (0.001)

nox                                              -0.123***              -0.078***
                                                  (0.014)                (0.011)

stratio                                                                 -0.043***
                                                                        (0.006)

rooms                                                                    0.248***
```

			(0.017)
Constant	10.032***	10.690***	9.654***
	(0.017)	(0.075)	(0.188)
Observations	506	506	506
R2	0.278	0.378	0.640
Adjusted R2	0.277	0.376	0.637
Residual Std. Error	0.348 (df = 504)	0.323 (df = 503)	0.247 (df = 501)
F Statistic	194.303*** (df = 1; 504)	152.911*** (df = 2; 503)	222.623*** (df = 4; 50
Note:			*p<0.1; **p<0.05; ***p<0.0

8 Student Activity

You may download the `bweight.xlsx` data here. Rather than left-clicking on the link, right-click and choose *Download Linked File*.

Work on the following tasks on your own or with a person sitting next to you.

1. Close the project you have been working on. R will ask you whether you would like to save your changes. Your response should be a **yes** if you want to keep all your work!
2. Create a new project called `birth-weight`. You may do this either through the File menu or the button on the top right center of your screen. **Please position this project in a new folder that you create under your R session folder**.
3. Create a new R script file to add your code.
4. Import the `bweight` Excel data into R.
5. Below are the labels of the variables. Assign these to each of the corresponding variable in data.

Variable	Label
bwght	baby's birth weight, in grams
mage	mother's age
cigs	average number of cigarettes smoked per day
male	dummy variable equal to 1 if baby is a male

6. Provide summary statistics for the variables.
7. Calculate the standard deviation for the `bwght` variable.
8. Estimate a regression model which explains the `bwght` with the mother's age, smoking behaviour, and the baby's gender.
9. Using the model you estimated, predict the baby's birth weight for the individuals in the sample.

Part II

Session 2 Data Visualisation with ggplot

9 Data Visualisation with ggplot

We will need the following packages installed for the exercises in this section. Remember to remove the # in front of the `install.packages()` function before running! The first package (`tidyverse` is a must to produce `ggplots`, while the last two are for animated plots.

```
# Install the required packages
# install.packages("tidyverse")
# install.packages("gganimate")
# install.packages("gifski")
```

We will then call the required libraries.

```
# Call the required libraries
library(tidyverse)
```

```
-- Attaching core tidyverse packages ------------------------ tidyverse 2.0.0 --
v dplyr     1.1.4     v readr     2.1.5
v forcats   1.0.0     v stringr   1.5.1
v ggplot2   3.5.1     v tibble    3.2.1
v lubridate 1.9.3     v tidyr     1.3.1
v purrr     1.0.2
-- Conflicts ------------------------------------------ tidyverse_conflicts() --
x dplyr::filter() masks stats::filter()
x dplyr::lag()    masks stats::lag()
i Use the conflicted package (<http://conflicted.r-lib.org/>) to force all conflicts to beco
```

```
library(gganimate)
library(gifski)
```

9.1 Plots with mpg data

`mpg` data comes with `ggplot2` package. We can call it directly without the need to import once we initiate the `ggplot` library. You may view the contents of the data with `View(mpg)` or ask R to print the first few lines with `head(mpg)`.

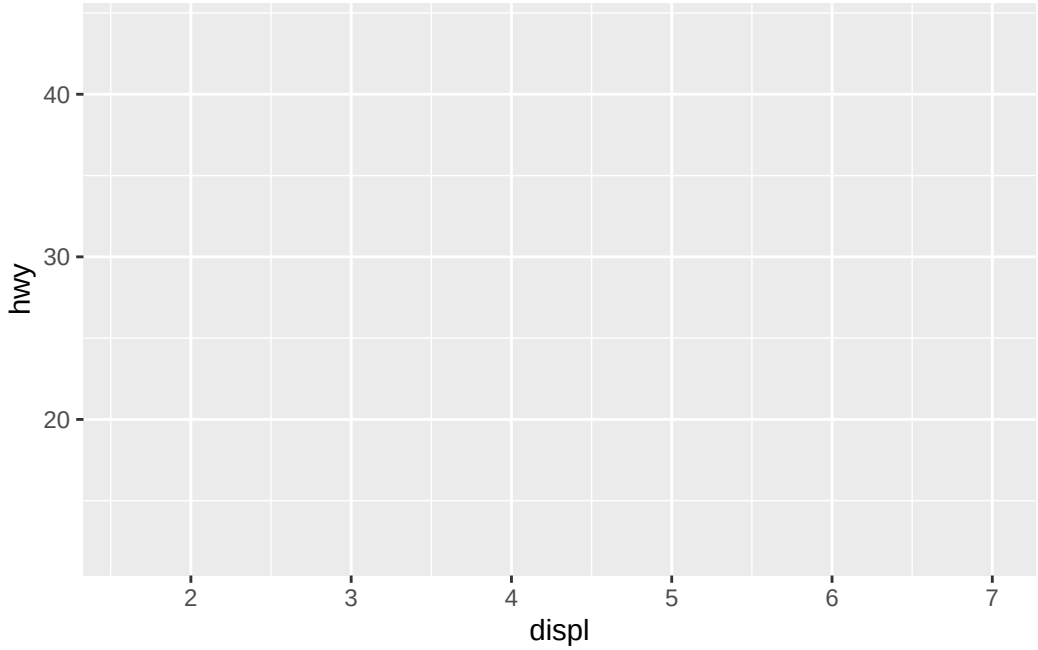
```
# View(mpg)
head(mpg)
```

```
# A tibble: 6 x 11
  manufacturer model displ  year   cyl trans      drv     cty   hwy fl    class
  <chr>        <chr> <dbl> <int> <int> <chr>      <chr> <int> <int> <chr> <chr>
1 audi         a4      1.8  1999     4 auto(l5)   f        18    29 p     compa~
2 audi         a4      1.8  1999     4 manual(m5) f        21    29 p     compa~
3 audi         a4      2    2008     4 manual(m6) f        20    31 p     compa~
4 audi         a4      2    2008     4 auto(av)   f        21    30 p     compa~
5 audi         a4      2.8  1999     6 auto(l5)   f        16    26 p     compa~
6 audi         a4      2.8  1999     6 manual(m5) f        18    26 p     compa~
```

Let's say we are interested in the relationship between `displ` (engine size in liters) and `hwy` (highway miles per gallon).

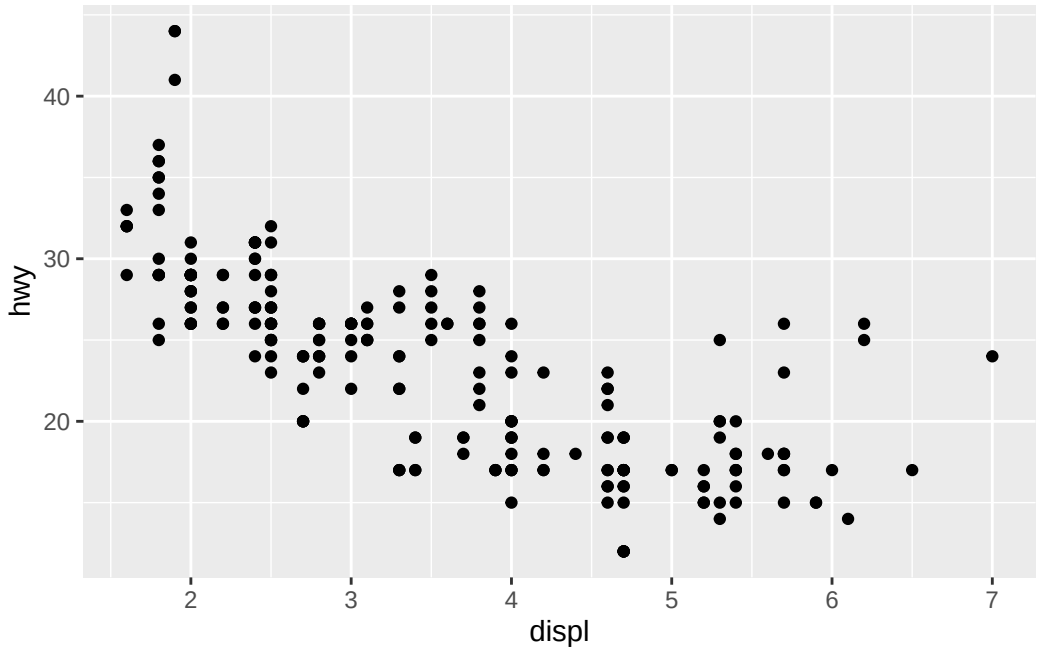
We start by creating a canvas for our plot:

```
# Create a canvas!
ggplot(mpg, aes(x = displ, y = hwy))
```



You see that the above plot is empty; we do not see much other than the axes labelled by our variable names. This is because we have not asked R to plot yet. Let us ask for a scatter plot by adding a new layer to our plot.

```
ggplot(mpg, aes(x = displ, y = hwy)) +
  geom_point()
```

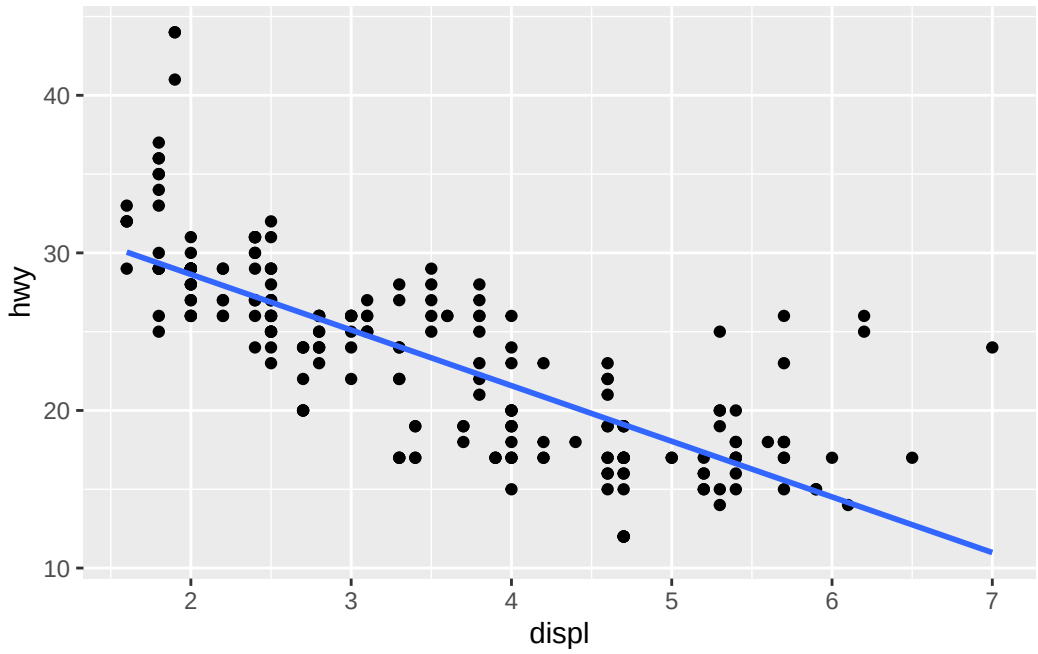


We see the negative relationship between engine size and the highway miles per gallon. The bigger the engine, the less fuel efficient the car is.

We may want to see this relationship represented by a line that fits best to the points in the plot. You may remember the `lm()` function from Session 1. We used it to estimate a linear regression (which depicts the relationship between dependent variable and independent variables). Note that we use this in the `method` attribute below.

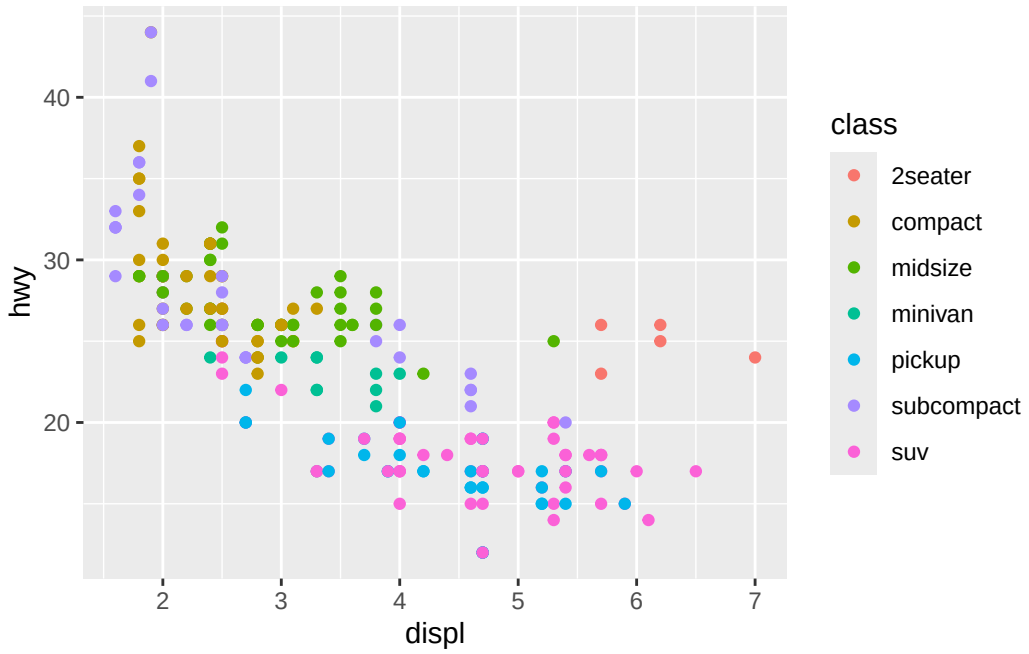
```
# Add a regression line (a line that fits best to the points on plot)
ggplot(mpg, aes(x = displ, y = hwy)) +
  geom_point() +
  geom_smooth(method = "lm", se = FALSE)
```

```
`geom_smooth()` using formula = 'y ~ x'
```



In the above scatter plot, we would like to identify the type of cars. We can do this by changing the shape of the dots and assigning a different shape for each car type, or for a better looking outcome, we may do this through introducing separate colors for each car type. The `class` variable in the data identifies the car type.

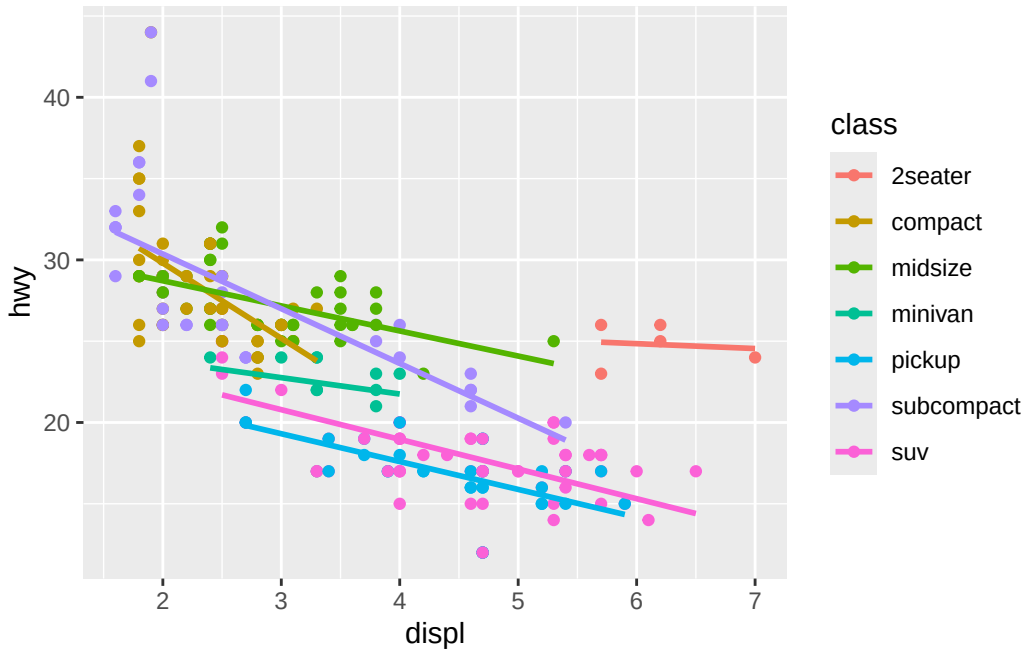
```
# Differentiate data points by class using colors
ggplot(mpg, aes(x = displ, y = hwy, colour = class)) +
  geom_point()
```



Now, let us add out best fit line back:

```
# Adding a best fit line with class coloring
ggplot(mpg, aes(x = displ, y = hwy, colour = class)) +
  geom_point() +
  geom_smooth(method = "lm", se= FALSE)
```

```
`geom_smooth()` using formula = 'y ~ x'
```

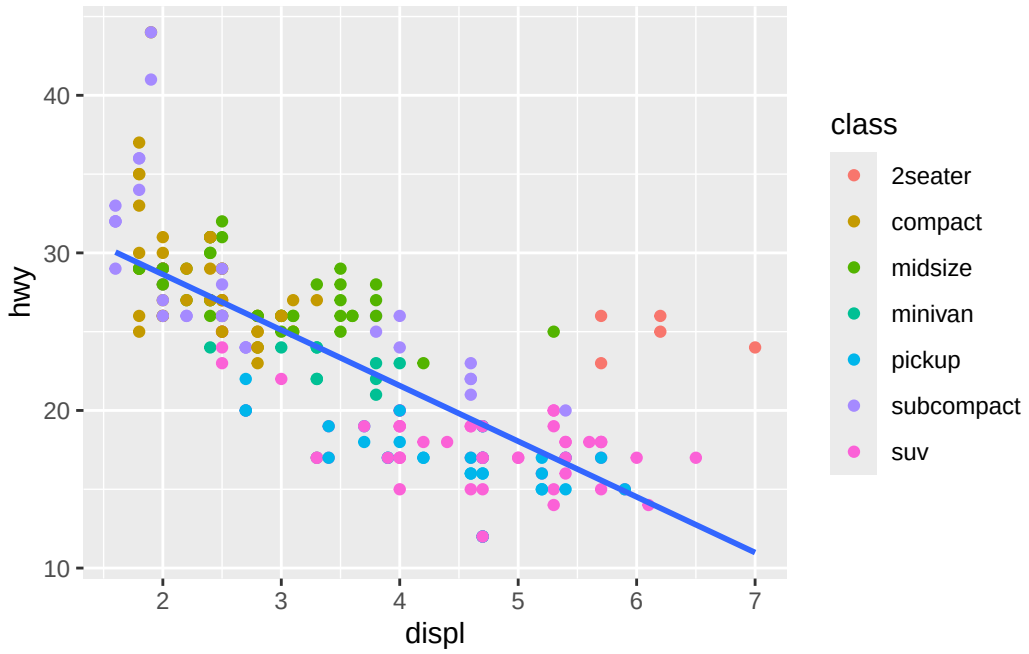


Although we used the same `geom_smooth(method = "lm", se= FALSE)`, this looks very much different than the line we added before! Why could that be?

Can you spot the difference between the above and below lines of code?

```
# Adding a best fit line for the overall group of data points
ggplot(mpg, aes(x = displ, y = hwy)) +
  geom_point(aes(colour = class)) +
  geom_smooth(method = "lm",se = FALSE)
```

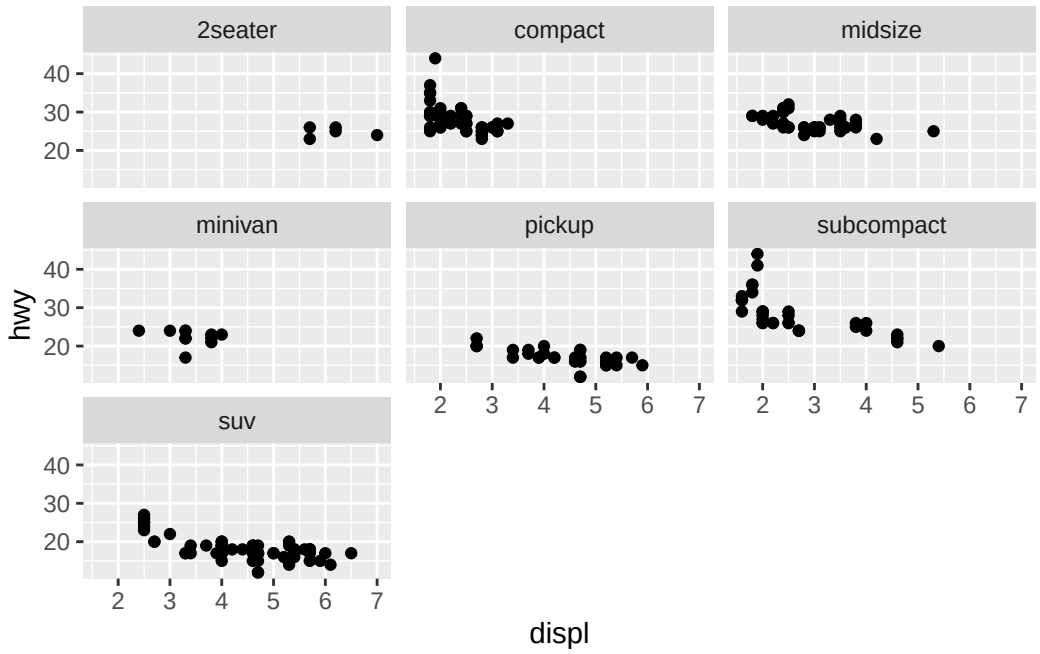
```
`geom_smooth()` using formula = 'y ~ x'
```



In the second version, we added the `aes(colour = class)` option to the layer where we defined the scatter plot. Hence, the color differentiation is applied only to the points of the scatter plot whereas in the first version, the coloring is introduced in the first line, where we define the aesthetics of the whole graph. Hence, the color differentiation is applied to all added layers, including the `geom_smooth()`.

Rather than using color differentiation, you may want to display the relationship for each car type separately. You may achieve this by faceting, where we use `facet_wrap()`.

```
# Faceting
ggplot(mpg, aes(displ,hwy)) +
  geom_point() +
  facet_wrap(~class)
```



9.2 Student task: Plots with region data

Download `region_data` here. Rather than left-clicking on the link, right-click and choose "*Download linked file*" or"*Save link as*".

`region_data` was sourced from https://ons.gov.uk/ and simplified to only contain aggregated regional data on gross disposable household income per capita for the period 1997 to 2022. Note that the data is measured in GDP at current prices.

Complete the following tasks:

1. Import the dataset
2. View the data
3. Create a plot using the `ggplot` function. Add the `year` variable to the x-axis and `GDHI_pc` on the y-axis. Use the `color = Region_name` option to the mapping aesthetics to illustrate regional differences by color. Select a suitable geometric object to display the data. Experiment with different options from the list below and select the one you think is most suitable:

    - `geom_area`
    - `geom_line`
    - `geom_point`

4. Use faceting to create a plot for each region. We do not need a separate legend, so we can suppress the legend by adding the layer `theme` with (`legend.position = "none"`.

5. In which region(s) do you observe the largest increase in gross disposable household income per capita?

9.2.1 Guidance

We will be importing the `region_data.csv`.

```
region_data <- read.csv("data/region_data.csv")
# View(region_data)
```

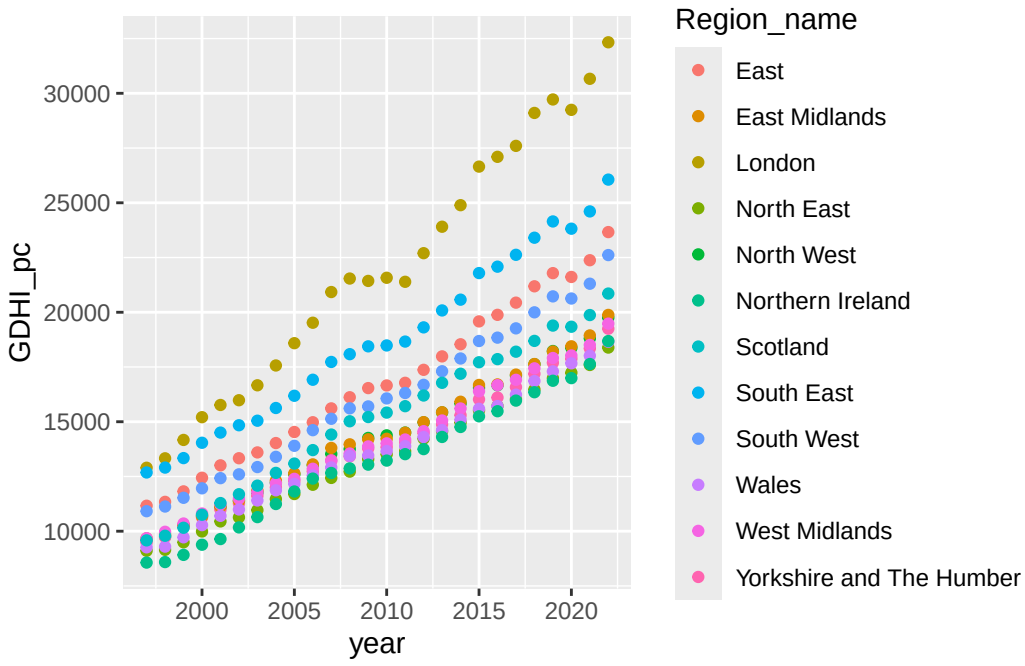
View the first few observations with `head()` function.

```
head(region_data)
```

```
  Region_name year GDHI_pc
1  North East 1997    9107
2  North East 1998    9150
3  North East 1999    9494
4  North East 2000    9987
5  North East 2001   10448
6  North East 2002   10631
```

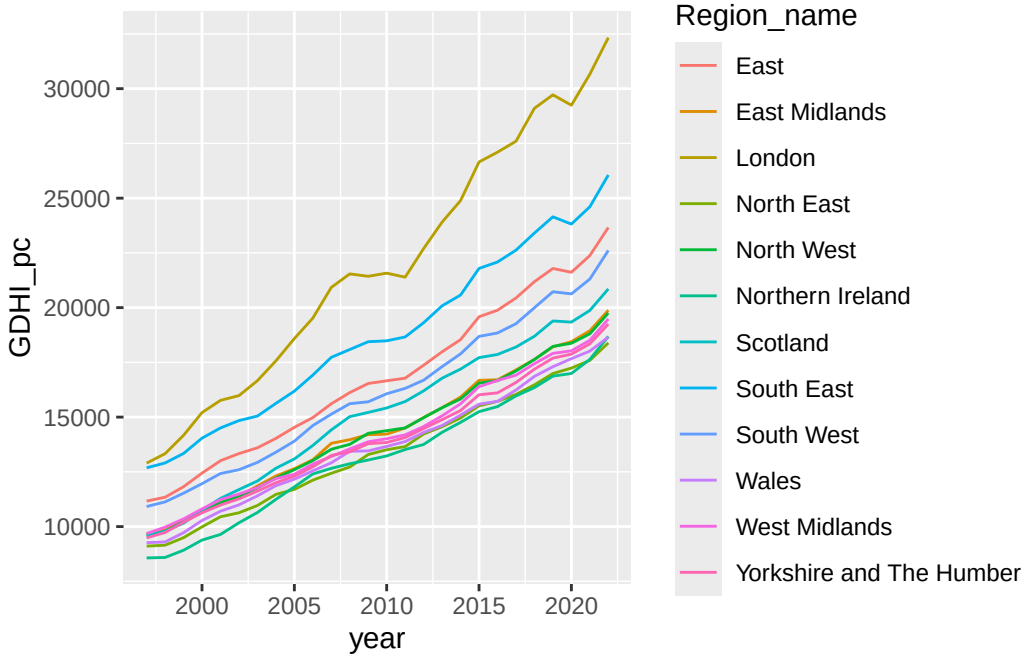
We start with a scatter plot.

```
# Scatter plot
ggplot(region_data, aes(x = year, y = GDHI_pc, colour = Region_name)) +
  geom_point()
```



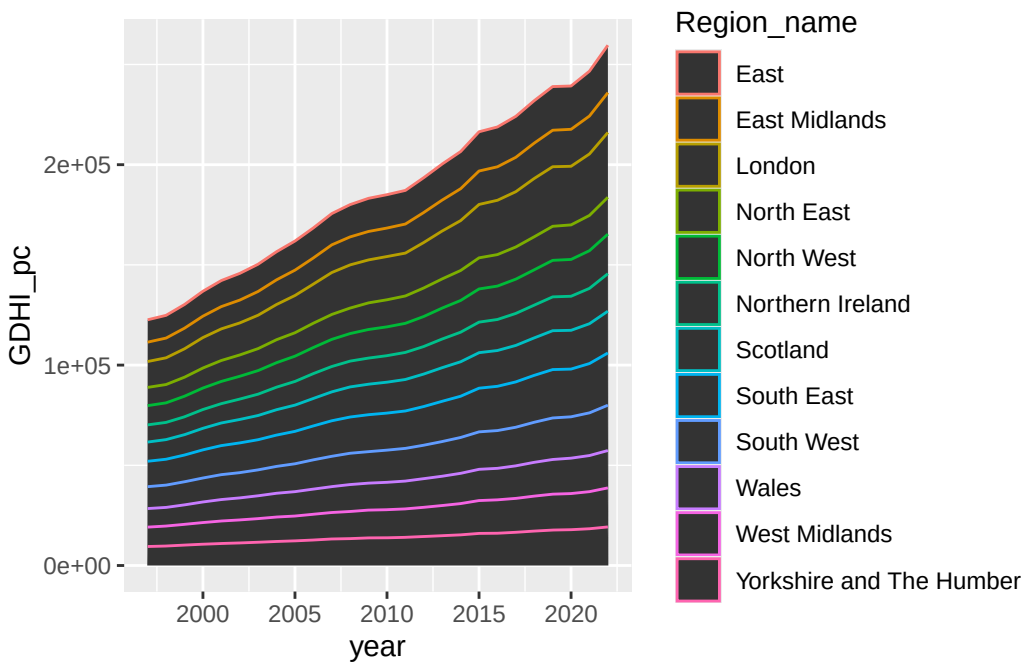
The line plot version is given below

```
# Line plot
ggplot(region_data, aes(x = year, y = GDHI_pc, colour = Region_name)) +
  geom_line()
```



Finally, the area plot

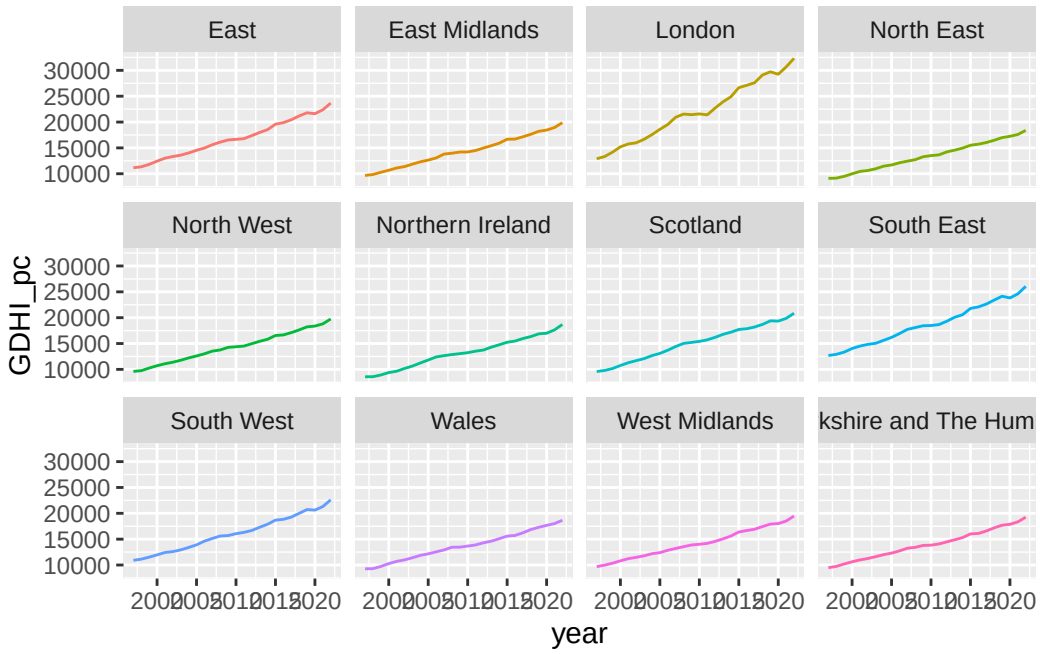
```
# Area plot
ggplot(region_data, aes(x = year, y = GDHI_pc, colour = Region_name)) +
  geom_area()
```



Looking at the three plots above, `geom_line` is the most suitable one for our purpose.

Adding faceting will help is to see the pattern in each region more clearly.

```
# Line plot with faceting
ggplot(region_data, aes(x = year, y = GDHI_pc, colour = Region_name)) +
  geom_line() +
  facet_wrap(~Region_name) +
  theme(legend.position = "none")
```



London has experienced the largest increase in `GDHI_pc`. With the exception of South East, most regions have a significantly lower disposable income.

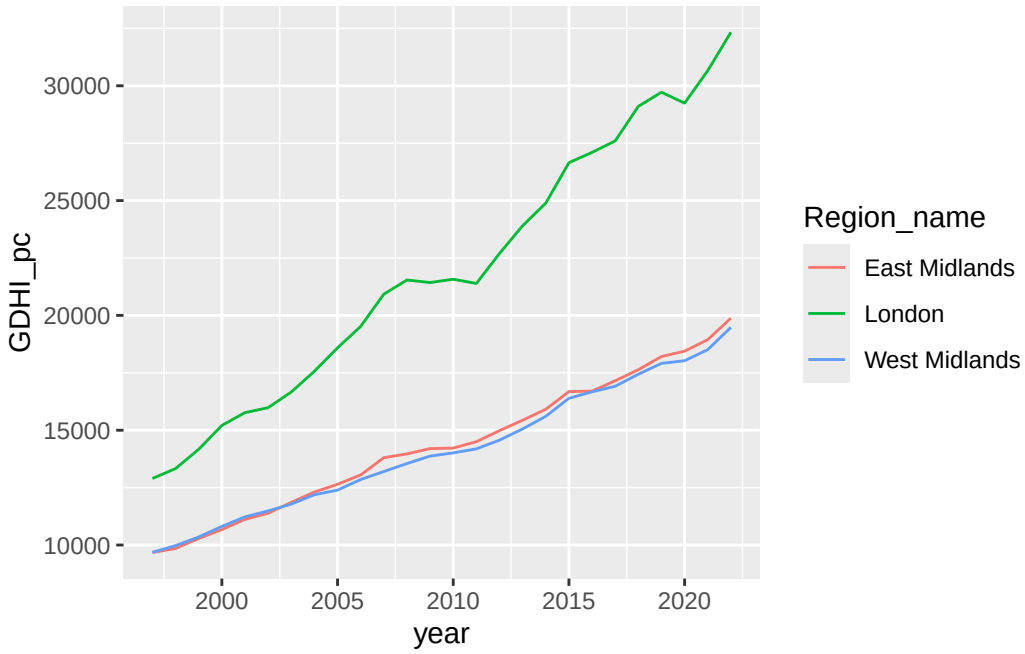
9.2.2 Exploring themes by a selected group of regions

We will be using a subsample of the above region data where there is a selection of regions: East Midlands, West Midlands, and London. Start by importing the data. You may download `region_data_sel` from here. Rather than left-clicking on the link, right-click and choose "*Download linked file*" or"*Save link as*".

```
# Import region_data_sel
region_data_sel <- read.csv("data/region_data_sel.csv")
```

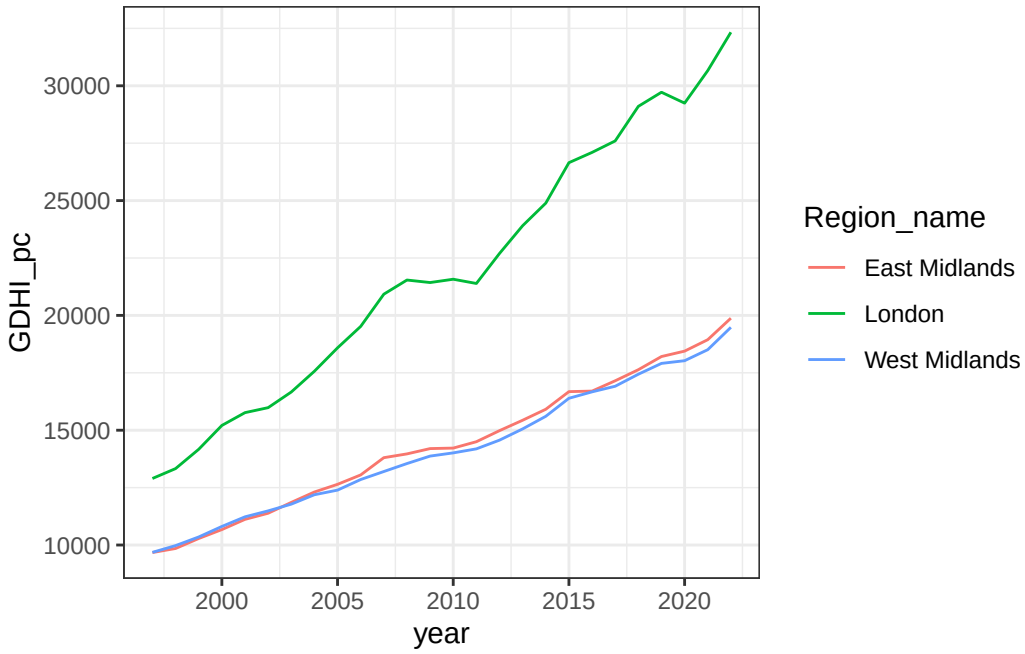
Let us do the line plot again.

```
ggplot(region_data_sel, aes(x = year, y = GDHI_pc, colour = Region_name)) +
  geom_line()
```



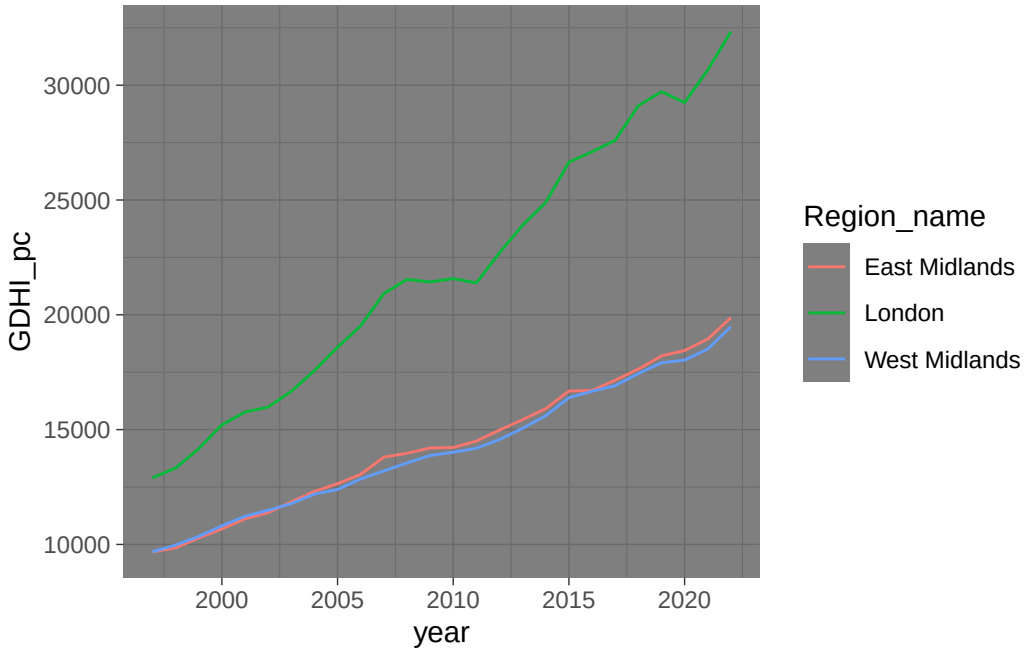
We can use themes in R to change the look of our plot. Some of these themes are already integrated into `ggplot2`, while you may install some other developed by R users or organisations. For example, BBC has its own theme. Below, we re-produce the above plot by adding the `theme_bw()`. This theme has a white background.

```
ggplot(region_data_sel, aes(x = year, y = GDHI_pc, colour = Region_name)) +
  geom_line() +
  theme_bw()
```



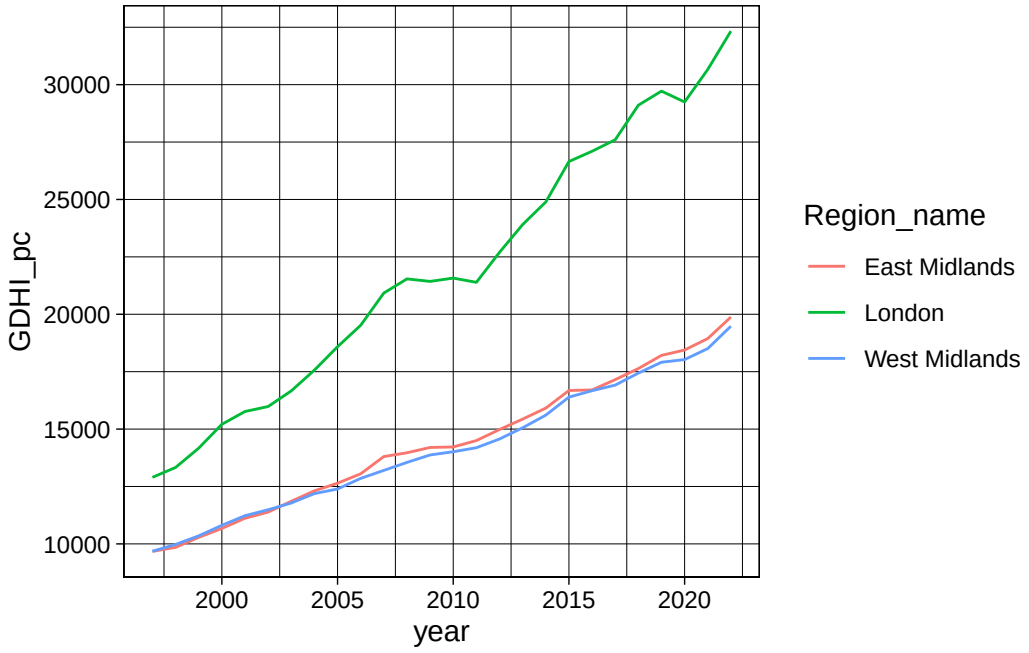
`theme_dark()` has a dark background.

```
ggplot(region_data_sel, aes(x = year, y = GDHI_pc, colour = Region_name)) +
  geom_line() +
  theme_dark()
```



`theme_linedraw()`.

```
ggplot(region_data_sel, aes(x = year, y = GDHI_pc, colour = Region_name)) +
  geom_line() +
  theme_linedraw()
```




```
# BBC-style
# install.packages('devtools')
# devtools::install_github('bbc/bbplot')
#library(bbplot)
#ggplot(region_data_sel, aes(x = year, y = GDHI_pc, colour = Region_name)) +
#  geom_line() +
#  bbc_style()
```

Source: https://www.bbc.co.uk/opensource/projects/project/bbplot

Which one do you prefer?

See https://ggplot2.tidyverse.org/reference/ggtheme.html for more information.

9.3 Bubble Charts

We will now follow the example from https://data.europa.eu/apps/data-visualisation-guide/grammar-of-graphics-in-practice-ggplot2, with more recent data from the World Bank,

World Development Indicators Database.

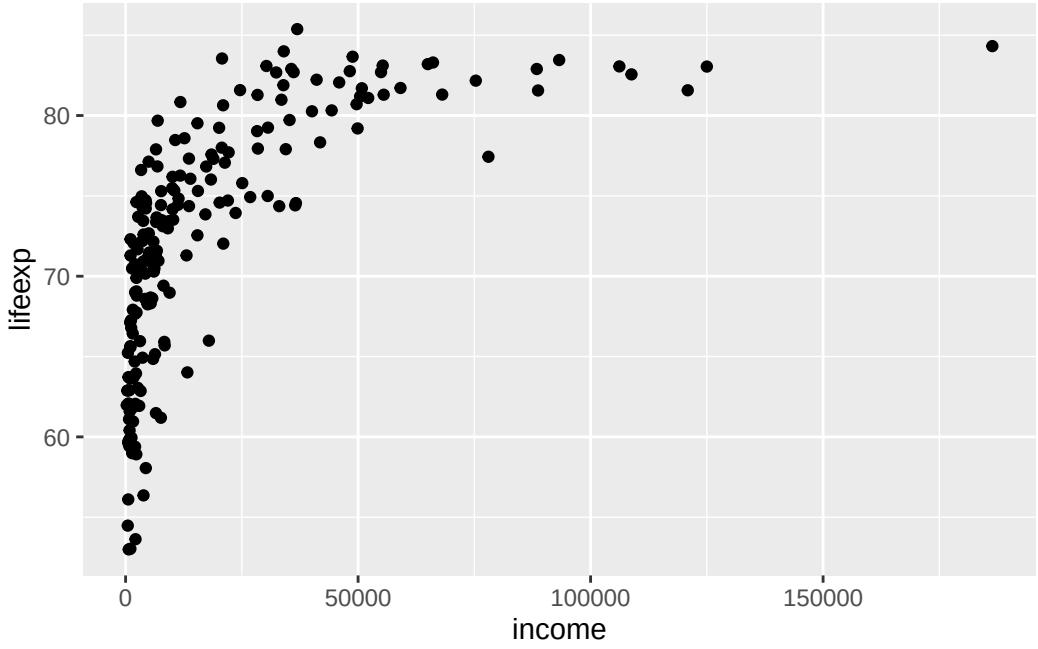
Import the `bubble.chart.data`. This data provides information on life expectancy, income, and population in 197 countries, as well as a variable on the continent they are located.

You may download the `bubble.chart.data` here. Rather than left-clicking on the link, right-click and choose "*Download linked file*" or"*Save link as*".

```
# Import bubble.chart.data
bubble.chart.data <- read.csv("data/bubble.chart.data.csv")
```

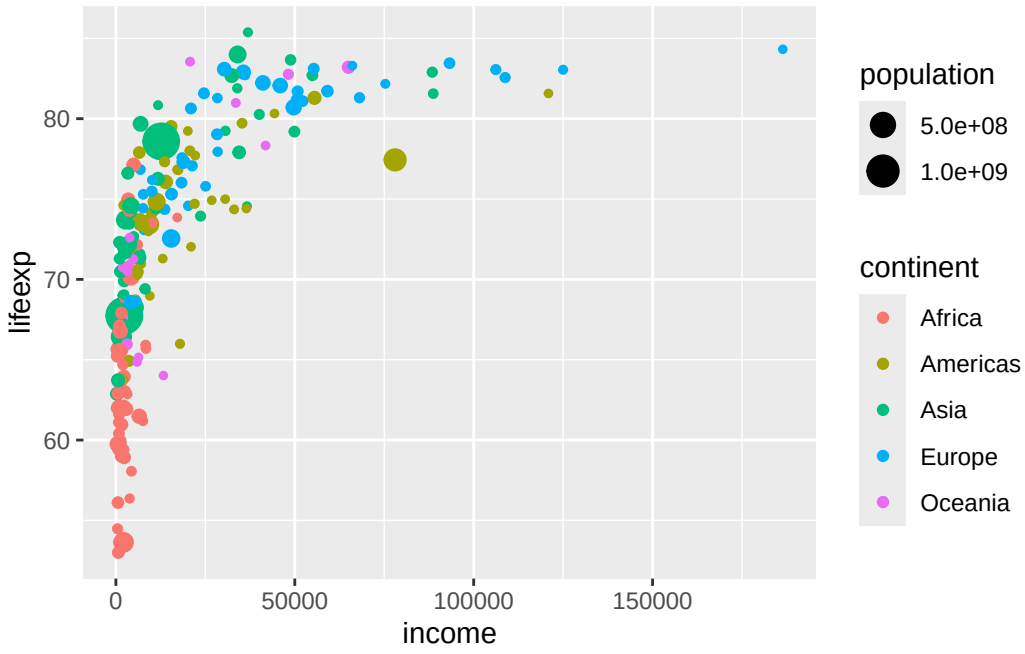
Let's observe the relationship between life expectancy and income.

```
# Scatterplot of life expectancy and income
ggplot(data = bubble.chart.data, aes(x = income, y = lifeexp)) +
  geom_point()
```



We would like to see how countries from different continents are positioned in the above plot. We will do this through color differentiating the data points. But also, we will change the size of the points to reflect the population of each country plotted.

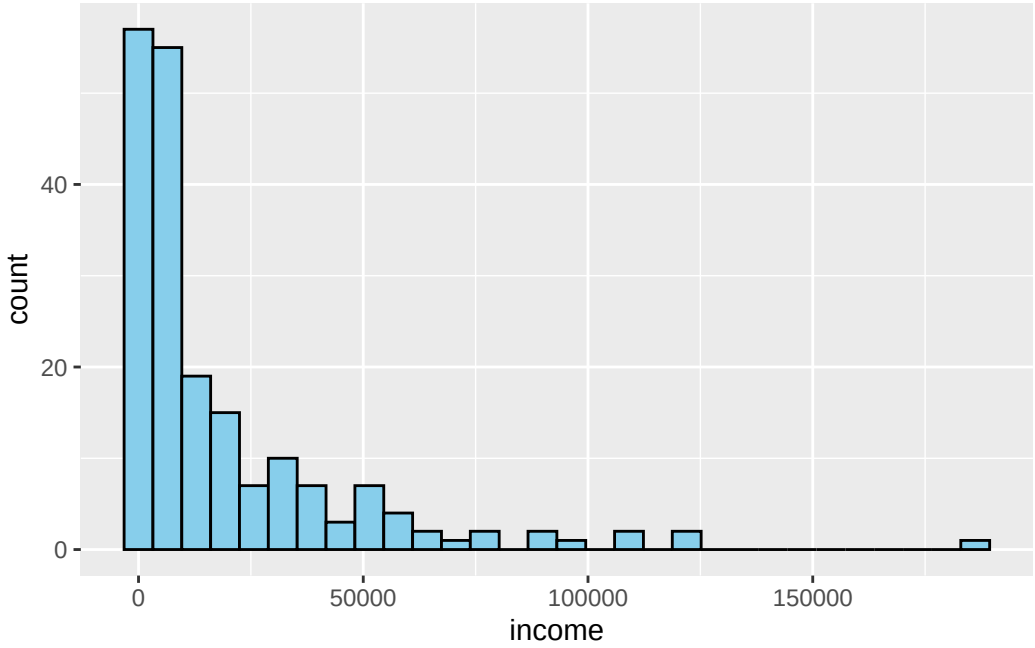
```
# Add coloring by continent and change size of points using population
ggplot(data = bubble.chart.data, aes(x = income, y = lifeexp,
                                     size = population, colour = continent)) +
  geom_point()
```



We observe above that life expectancy in African countries is the lowest while European countries rank among the highest. But it is difficult to get much information from the above plot because of the highly skewed distribution of income. Only a few countries income higher than 50,000 while most countries are much below that. We may also observe this through a histogram of income.

```
ggplot(data = bubble.chart.data, aes(x = income)) +
  geom_histogram(fill = "skyblue", color = "black")
```

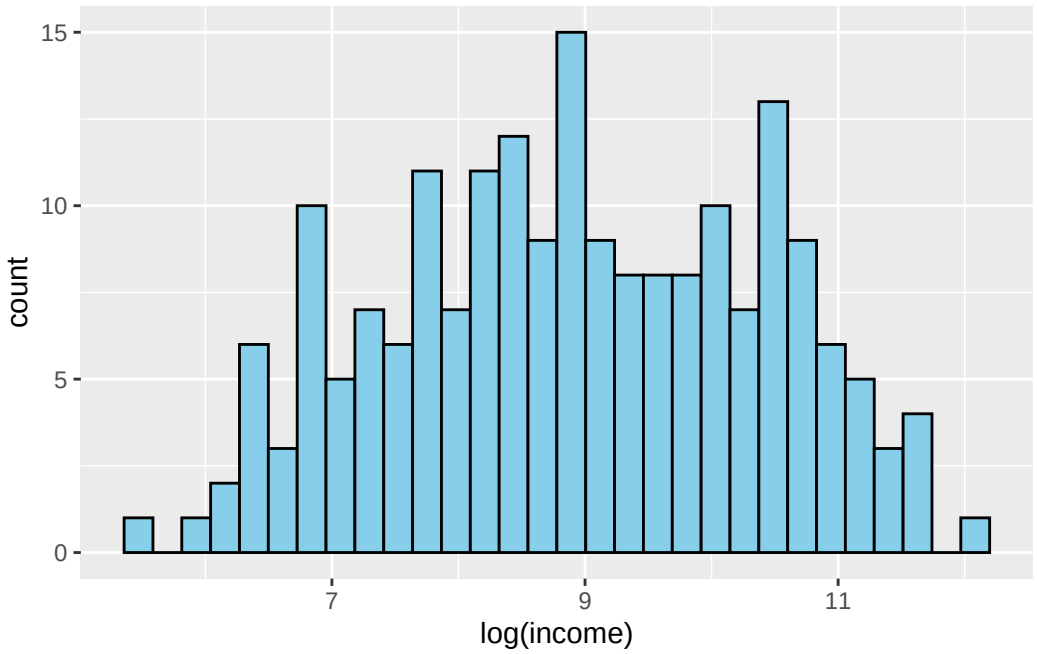
```
`stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```



To make this distribution less skewed, we may take the logarithm of income. This will reduce the wide gaps at the higher ends of the distribution, which will help us to better observe the data points at the lower end.

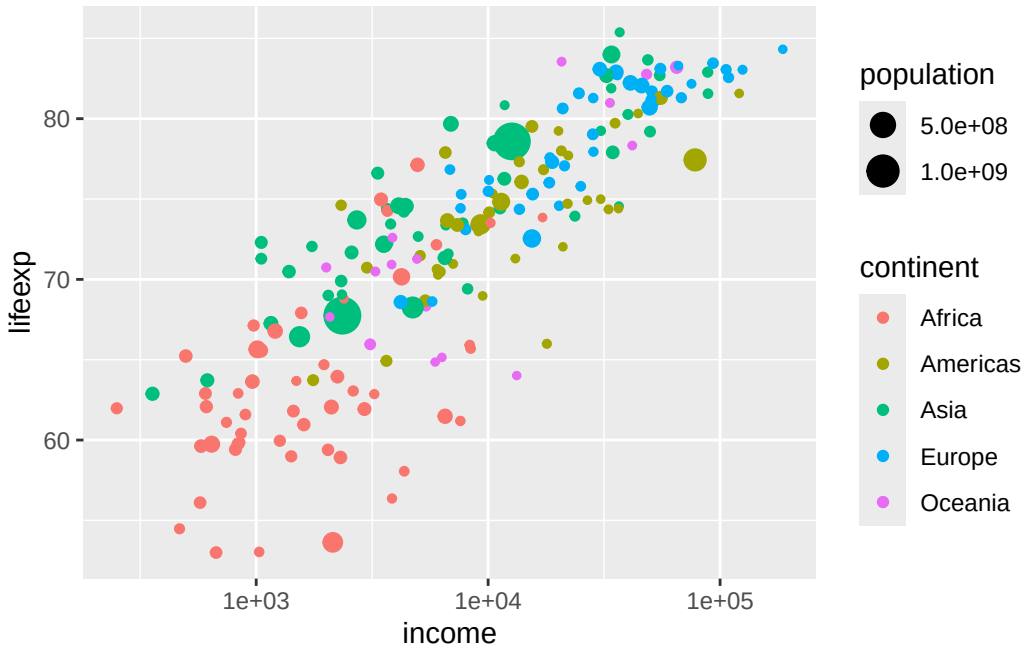
```
ggplot(data = bubble.chart.data, aes(x = log(income))) +
  geom_histogram(fill = "skyblue", color = "black")
```

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



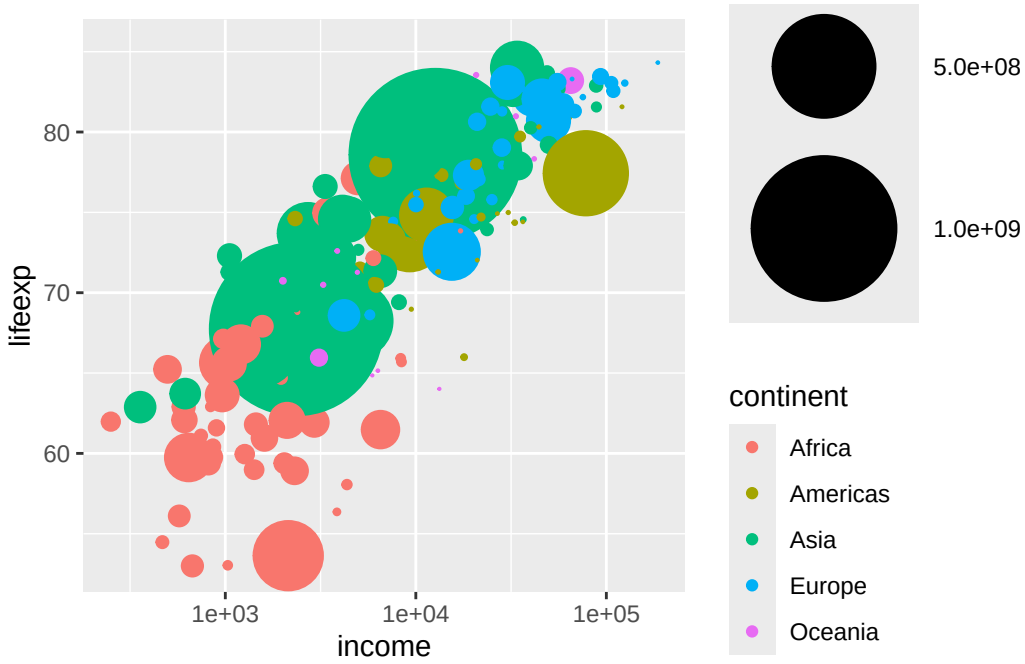
Now, let's apply this to our bubble chart and ask `ggplot` to change the scale of the x-axis. It will give is a logarithmic scale.

```
ggplot(data = bubble.chart.data, aes(x = income, y = lifeexp,
                                     size = population, colour = continent)) +
  geom_point() +
  scale_x_log10()
```



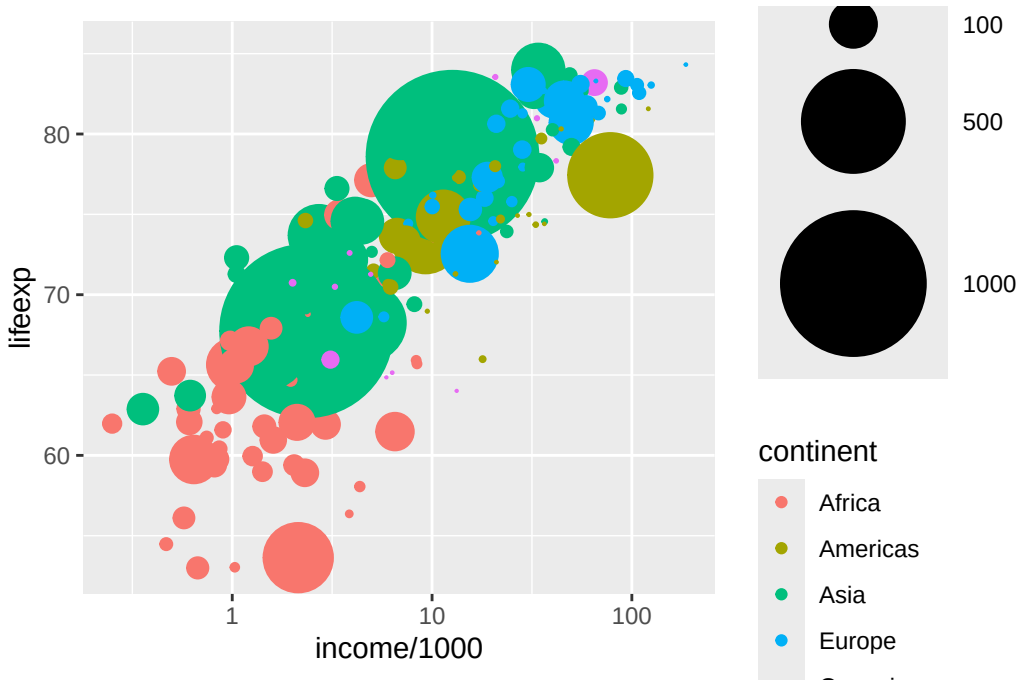
Much better! However, at the moment, the size of each data point is allocated to an ordinal scale, which only shows us which countries are large and which are small; it does not reflect the actual magnitude of the population size differences. We will change this (the size based on population) to be a continuous scale. While doing that, we will also set a maximum size so that we do not lose clarity with too large bubbles.

```
# Add accurate re-sizing of points using population
ggplot(data = bubble.chart.data, aes(x = income, y = lifeexp,
                                     size = population, colour = continent)) +
  geom_point() +
  scale_x_log10() +
  scale_size_area(max_size = 30, name = "Population")
```



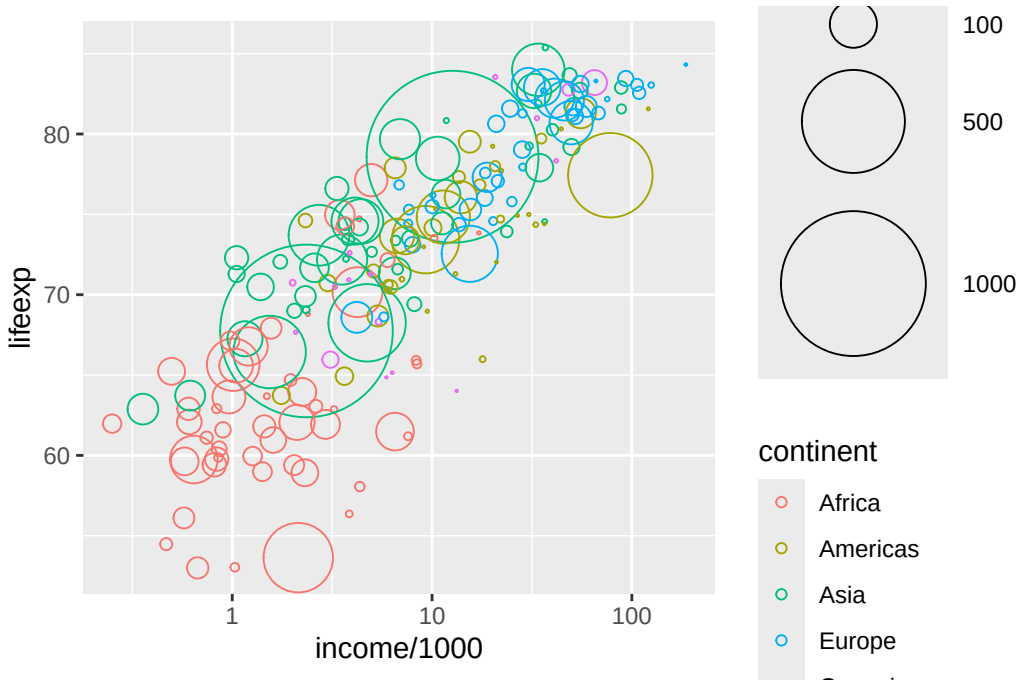
We try to avoid scientific representation of numbers (like the legend for population above) so, we will change the units of measurement: we will measure population in millions of people and income in thousands. Also, we will assign some more intuitave breaks on the x-axis.

```
# Add legends
ggplot(data = bubble.chart.data, mapping = aes(
  x = income/1000,
  y = lifeexp,
  size = population/1000000,
  colour = continent)) +
  geom_point() +
  scale_x_log10() +
  scale_size_area(
    max_size = 30,
    name = "Population (millions)",
    breaks = c(10, 100, 500, 1000))
```



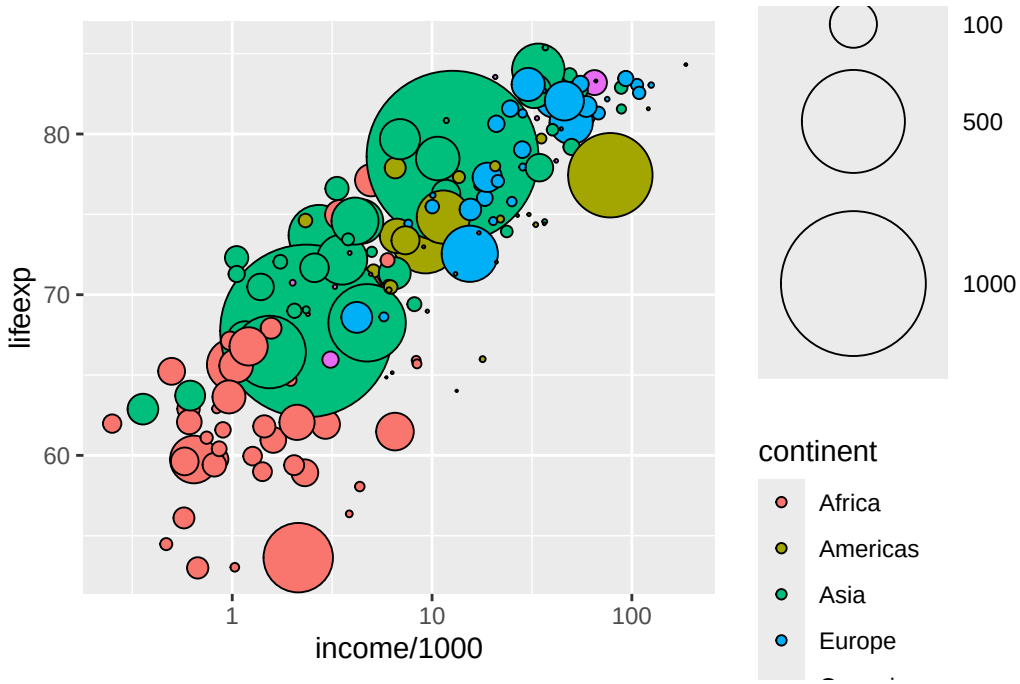
There are a lot of overlaps between circles. We would like be able to better differentiate them. We will use shape 21 for adding a full circle around our data points. See https://ggplot2.tidyverse.org/articles/ggplot2-specs.html#sec:shape-spec for different shape values.

```
# Add border to circles
ggplot(data = bubble.chart.data, mapping = aes(
  x = income/1000,
  y = lifeexp,
  size = population/1000000,
  color = continent)) +
  geom_point(shape = 21) +
  scale_x_log10() +
  scale_size_area(
    max_size = 30,
    name = "Population (millions)",
    breaks = c(10, 100, 500, 1000))
```



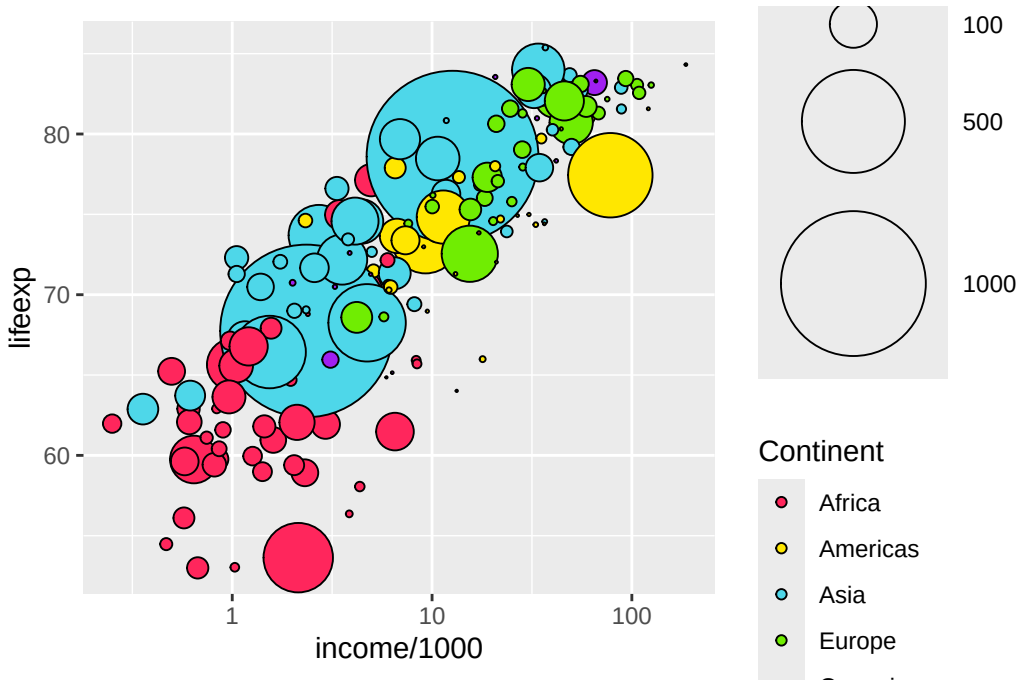
Now, we have the bubbles but we would like them to be colour-filled. We will change the `color = continent` option above to `fill = continent`

```
ggplot(data = bubble.chart.data, mapping = aes(
  x = income/1000,
  y = lifeexp,
  size = population/1000000,
  fill = continent)) +
  geom_point(shape = 21) +
  scale_x_log10() +
  scale_size_area(
    max_size = 30,
    name = "Population (millions)",
    breaks = c(10, 100, 500, 1000))
```



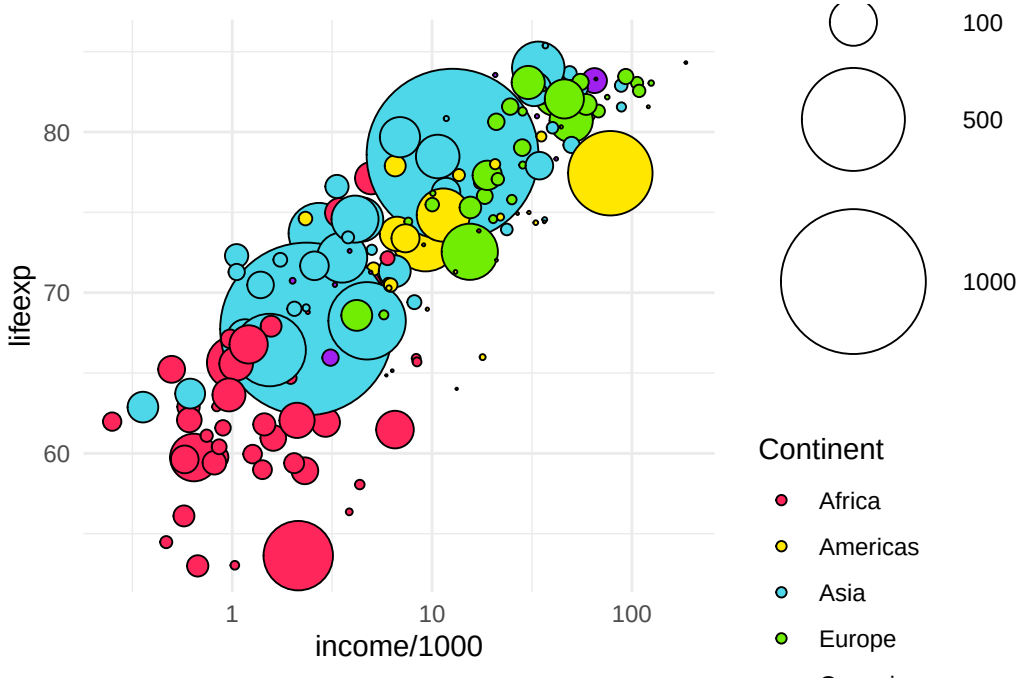
We my add custom colors to our plot

```
# Add custom colors for circles
ggplot(data = bubble.chart.data, mapping = aes(
  x = income/1000,
  y = lifeexp,
  size = population/1000000,
  fill = continent)) +
  geom_point(shape = 21) +
  scale_x_log10() +
  scale_size_area(
    max_size = 30,
    name = "Population (millions)",
    breaks = c(10, 100, 500, 1000)) +
  scale_fill_manual(
    values = c("#FF265C", "#FFE700", "#4ED7E9", "#70ED02", "purple"),
    name = "Continent")
```



Looking much nicer! But it would help to remove the gray color at the background. There is no purpose of coloring at the background.

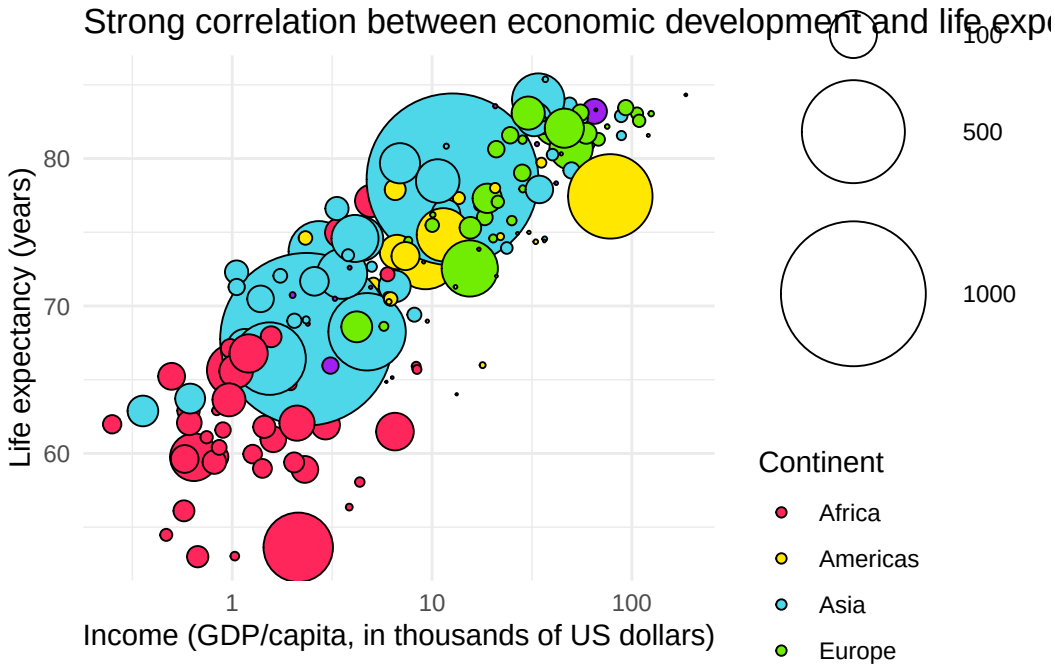
```
# Change plot theme to minimal to remove background color
ggplot(data = bubble.chart.data, mapping = aes(
  x = income/1000,
  y = lifeexp,
  size = population/1000000,
  fill = continent)) +
  geom_point(shape = 21) +
  scale_x_log10() +
  scale_size_area(
    max_size = 30,
    name = "Population (millions)",
    breaks = c(10, 100, 500, 1000)) +
  scale_fill_manual(
    values = c("#FF265C", "#FFE700", "#4ED7E9", "#70ED02", "purple"),
    name = "Continent") +
  theme_minimal()
```



It is now easier on the eye.

Finally, add title and axis labels below

```
# Add title and label axes
ggplot(data = bubble.chart.data, mapping = aes(
  x = income/1000,
  y = lifeexp,
  size = population/1000000,
  fill = continent)) +
  geom_point(shape = 21) +
  scale_x_log10() +
  scale_size_area(
    max_size = 30,
    name = "Population (millions)",
    breaks = c(10, 100, 500, 1000)) +
  scale_fill_manual(
    values = c("#FF265C", "#FFE700", "#4ED7E9", "#70ED02", "purple"),
    name = "Continent") +
  theme_minimal() +
  labs(x ="Income (GDP/capita, in thousands of US dollars)",
       y ="Life expectancy (years)", title = "Strong correlation between economic development
```



9.4 Animated Plots

We need the `gganimate` and `gifski` for the animated plots. We installed and loaded these packages at the very top of this worksheet.

Load the data first. You may download the `animate.csv` data from here. Rather than left-clicking on the link, right-click and choose "*Download linked file*" or"*Save link as*".

```
animate_data <- read.csv("data/animate.csv")
```

We will copy the last command from above and introduce some changes:

- Save the plot object under name `p`
- add `transition_time()` layer to create plots for each year (to be combined in our gif image after)
- add `panel.grid.minor = element_blank()` to `theme()`
- add `title = "Year: {frame_time}"` to `labs`

```
p <- ggplot(data = animate_data, mapping = aes(
  x = income/1000,
  y = lifeexp,
  size = population/1000000,
  fill = continent)) +
  geom_point(shape = 21) +
  scale_x_log10() +
  scale_size_area(
    max_size = 30,
    name = "Population (millions)",
    breaks = c(10, 100, 500, 1000)) +
  scale_fill_manual(
    values = c("#FF265C", "#FFE700", "#4ED7E9", "#70ED02", "purple"),
    name = "Continent") +
  theme_minimal() +
  theme(panel.grid.minor = element_blank(),
        legend.position = "none") +
  labs(title = "Year: {frame_time}",
       x = "Income (GDP/capita, PPP (constant 2021 international $))",
       y = "Life expectancy (years)") +
  transition_time(year)
```

Render the plot object `p`

```
# render plot
#p_anim <- animate(p, renderer = gifski_renderer())
```

Save the animated plot

```
# save animated plot
#anim_save("my_animation.gif", animation = p_anim)
```