R-workshops

Mehtap Hisarciklilar

2025-01-20

Table of contents

Welcome!

This workbook is created for the seminar sessions of

6036ECN Further Econometrics module.

It is written using Quarto on RStudio by

Mehtap Hisarciklilar

Part I

Seminar 1 (21 January 2025)

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 attend the seminar sessions and practice regularly. There will be a lot of struggle, but the result is worth it.

R, as a programming language, is like any other language: the more you use it, the better you will get. Therefore, make sure to attend the lectures & seminars and engage with the module material. Otherwise, you will struggle to catch-up.

- I list below three apps that you will need to work with this module's material. I recommend installing these on your computers. Alternatively, you may use Coventry University's Appsanywhere platform to get access.
- We will be using **R** as the statistical analysis tool in this module. 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, I 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 I 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. I will be using Quarto to produce R worksheets for this module. 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 module, but the most important ones will be the console and the editor pane.

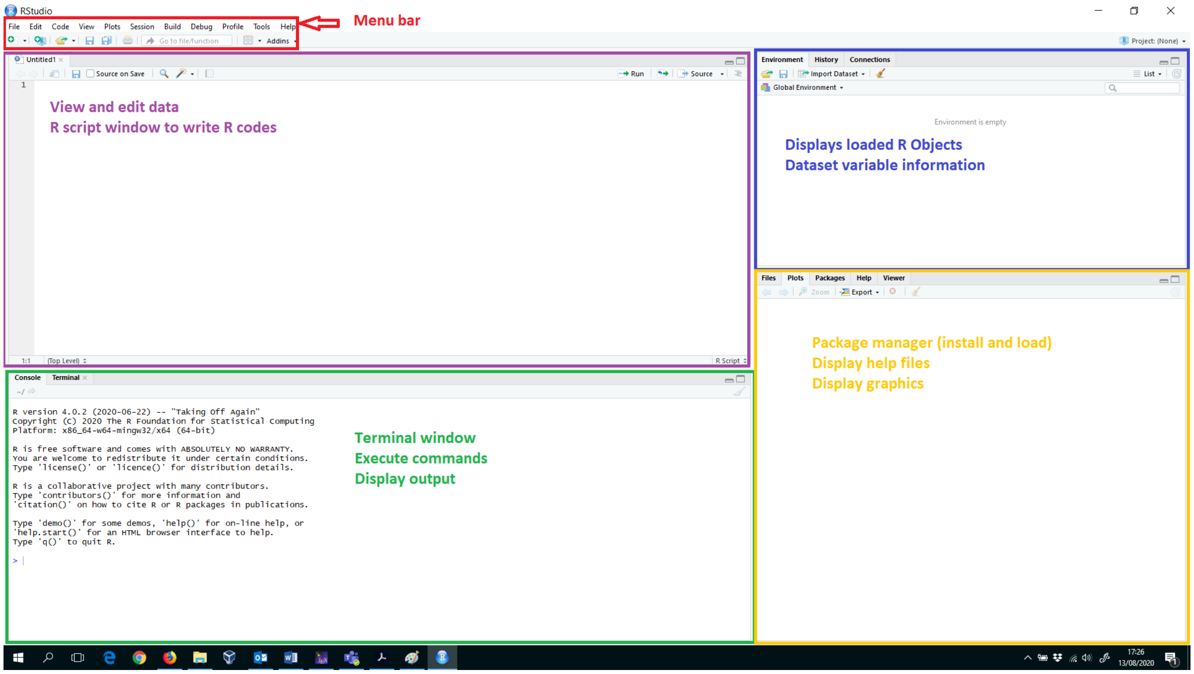


Figure 1.1: RStudio windows and their functions

1.2 File Organisation

- Create a folder for this module. This folder should include all module material you download from Aula or other platforms. Group files in sub-folders in a way that you can locate them easily. So for example, `6036ECN-Further-Econometrics` may be the name of the folder and then you may have sub-folders such as `Lecture-Slides`, `R-workshops`, etc.
- You should have one folder for R-workshops. I recommend naming this folder as `R-workshops` and within that folder, create sub-folders as we progress in the module.
- Please note that my `R-workshops` folder is located on my `desktop`. Hence, I will refer to the folder as `~/Desktop/R-workshops`. You will need to modify this depending on where you locate your files.
- 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`.

```
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`:

  ```
  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)
```

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%.

```r
# 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)

```r
decimal_number <- 2.2
```

2.3.1.2 Integer

```r
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.

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

```r
t
```

```
[1] TRUE
```

```r
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).

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

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

```r
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:

```r
class(decimal_number)
```

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

```r
class(i)
```

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

```r
class(t)
```

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

```r
class(f)
```

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

```r
class(message)
```

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

Part II

Seminar 2 (28 January 2025)

3 Data Management in R

3.1 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. Using packages is key to the successful use of R." (Wickham, Cetinkaya-Rundel, and Grolemund, n.d.)

We will start by installing the `tidyverse` package

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

To install `tidyverse`, type the above line of code in the console, 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(tidyverse)
```

You see above a list of packages that come with `tidyverse`.

You may update `tidyverse` by running

```
#tidyverse_update()
```

3.2 Functions

You may identify functions with the `()` after the function name. For example, `ls()` that we used above.

Functions may also take *arguments*. The data that we pass into the function is called the function's *argument*. The argument can be raw data, an R object, or even the results of another R function.

```
# round a number
round(4.5218)
```

```
[1] 5
```

```
## 5

# calculate the factorial
factorial(3)
```

```
[1] 6
```

```
## 6

# calculate the mean of values from 1 to 6:
mean(1:6)
```

```
[1] 3.5
```

```
## 3.5

round(mean(1:6))
```

```
[1] 4
```

```
## 4
```

Many R functions take multiple arguments that help them do their job. You can give a function as many arguments as you like as long as you separate each argument with a comma.

To see which arguments a function can take, you may type **`args`** in parenthesis after function name:

```
args(round)
```

```
function (x, digits = 0, ...)
NULL
```

```
## function (x, digits = 0)
## NULL

round(3.1415, digits = 2)
```

```
[1] 3.14
```

```
## 3.14
```

3.2.1 Basic Functions

Function	Description
?() or **help()**	Access the documentation and help file for a particular function
install.packages()	Download and install an R package
library()	Loads an R package into the working environment
setwd()	Set the working directory
getwd()	Get the working directory
c()	Create a vector
as.numeric()	Converts an object to a numeric vector
as.logical()	Converts an object to a logical vector
as.character()	Converts and object to a character vector
mode()	Returns the type of the object
sum()	Returns the sum of all input values
length()	Returns the lenght of the obejct
mean()	Returns the arithmetic mean of the vector
median()	Returns the median of the vector
sample()	Returns a specificed size of elements from the object
replicate()	Repeats an expression a specific number of times
hist()	Creates a histogram of given data values

3.3 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.

4 Importing Data into R

4.1 Example 1: Crime data

The example and instructions provided in this section is taken from (Riegler 2022).

The following exercise gives you a hands-on introduction to basic operations in R using a real-world data set. It begins with importing a MS-Excel data set into R and asks you to perform some basic operations to familiarise yourself with some of the commands that will be relevant for the coursework and in subsequent computer classes.

Please download the Excel data set called `crime.xls` from Aula and save it into a drive of your choice. This is a data set that contains crime levels and other socio-economic information on 46 cities across the US for the year 1982. The full version of the data set can be accessed at http://fmwww.bc.edu/ec-p/data/wooldridge/datasets.list.html. The variables are defined as follows:

Variable	Definition
pop	actual population in number
crimes	total number of crimes
unem	unemployment rate (%)
officers	number of police officers
pcinc	per capita income, $
area	land area, square miles
lawexpc	law enforcement expenditure per capita, $

From here on, you need to open a R script to save all your commands to be able to replicate your results:

4.1.1 Task 1

4.1.1.1 Task

Import the Excel data set into R.

4.1.1.2 Guidance

The native data format of R is .Rdata, however, you can also open other formats, such as .xlsx, .csv, etc. Non-native data formats have to be imported rather than just opened. Before we can we can import Excel spreadsheets directly into R, we have to activate a R-library first.

You can either use the package manager window (in the right bottom corner of the R screen) and tick the box next to the package name or you type the following into the terminal window (in the left bottom of the R screen)

```r
library(readxl)
```

This line loads the necessary `readxl` library. But you will probably receive an error message when you run the above line. This is because we first need to install the `read_excel` package. (Note that you will need to type the below line without the pound (hashtag) sign at the beginning of the line).

```r
# install.packages("readxl")
library(readxl)
```

There are two ways to import:

1. Through command line:

   ```r
   crime <- read_excel("./assets/data/crime.xls")
   ```

   In the above line, we import the dataset with the `read_excel` function and store it under the name `crime`. Notice how the new `crime` data is added as an object in the R environment.

2. Through menu:

   File –> Import Dataset –> From Excel

Don't forget to tick the "First Row as Names" box if it is not ticked!

4.1.2 Task 2

4.1.2.1 Task

View the dataset in R's data viewer.

4.1.2.2 Guidance

To open the data viewer, use the `View` function.

```
# View(crime)
```

Note that the first letter of `View` is capitalised.

4.1.3 Task 3

4.1.3.1 Task

View the first few (six) entries of the crime data to get a feeling of what the values look like.

4.1.3.2 Guidance

Use the `head` function

```
head(crime)
```

```
# A tibble: 6 x 7
     pop crimes  unem officers pcinc  area lawexpc
   <dbl>  <dbl> <dbl>    <dbl> <dbl> <dbl>   <dbl>
1 229528  17136  8.20      326  8532  44.6    851.
2 814054  75654  8.10     1621  7551 375      875.
3 374974  31352  9         633  8343  49.8   1122.
4 176496  15698 12.6       245  7592  74      744.
5 288446  31202 12.6       504  7558  97.3    974.
6 122768  16806 13.9       186  6411  55.3    762.
```

4.1.4 Task 4

4.1.4.1 Task

Label the variables using the definitions given above.

4.1.4.2 Guidance

You have to attach a variable label to each variable. There is already a library available which facilitates the allocation of labels to variables. First, we need to install the package!

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

```
Loading required package: maditr
```

```
To drop variable use NULL: let(mtcars, am = NULL) %>% head()
```

```
Use 'expss_output_rnotebook()' to display tables inside R Notebooks.
 To return to the console output, use 'expss_output_default()'.
```

```
crime <- apply_labels(crime,
                      pop = "actual population in number",
                      crimes = "total number of crimes",
                      unem = "unemployment rate (%)",
                      officers = "number of police officers",
                      pcinc = "per capita income, $",
                      area = "land area, square miles",
                      lawexpc = "law enforcement expenditure per capita, $"
                      )
```

4.1.5 Task 5

4.1.5.1 Task

Create a new variable which measures the population density for each city.

4.1.5.2 Guidance

To generate a new variable and add it to the existing crime dataset, we use the following command:

```r
crime$popdens <- crime$pop / crime$area
```

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

Note that the newly created population density variable is now labelled as the original population variable (`pop`). Let's update the label with the method we introduced in Task 4. Note that we do not need to call the library again, as it is already called.

```r
crime <- apply_labels(crime,
                      popdens = "population density: pop / area")
```

4.1.6 Task 6

4.1.6.1 Task

Sort the data with respect to the population density of each city.

4.1.6.2 Guidance

Sorting data is a useful action to get a general feeling for the data, e.g. are there any outliers in the dataset? Are there any unusual patterns?

To change the order of the rows in a data frame, we will apply the `order` function. We first rank all observations with respect to the population density and store this information in a vector called `rank`. The rank vector contains indices that we can use to sort the crime data frame. Below, we save the sorted data under a new name, `crime.popdens1`

```r
rank <- order(crime$popdens)
crime.popdens1 <- crime[rank,]
```

Let's see the result (note. how the population density variable is now sorted from the smallest to the largest):

```r
head(crime.popdens1)
```

```
# A tibble: 6 x 8
  pop        crimes     unem       officers   pcinc      area  lawexpc popdens
  <labelled> <labelled> <labelled> <labelled> <labelled> <lab> <label> <labelle>
1 425093     38195       4.7        767       7991       604.0  570.00  703.7964
2 268887     14537       5.5        400       7704       255.9  570.63 1050.7503
3 462657     34736      10.4        937       7585       352.0  582.56 1314.3665
4 451397     45503      10.4       1145       7480       316.4 1054.17 1426.6656
5 412661     47128       8.3        719       7336       258.5  554.70 1596.3675
6 173630     18915       8.7        366       7409       100.5  827.16 1727.6617
```

```r
# you may alternatively use
# View(crime.popdens1)
```

This procedure sorts the data from the smallest to the largest value. To sort the data from the largest to the smallest number, we set the order argument decreasing to TRUE.

```r
crime.popdens2 <- crime[order(crime$popdens, decreasing = TRUE),]
head(crime.popdens2)
```

```
# A tibble: 6 x 8
  pop        crimes     unem       officers   pcinc      area  lawexpc popdens
  <labelled> <labelled> <labelled> <labelled> <labelled> <lab> <label> <labelle>
1  708287     68598      8.4       1971       9265        46.4 1050.00 15264.806
2  334414     36172     15.4       1166       4525        24.1 1139.32 13876.099
3  365506     52901     12.3        979       6084        34.3  714.00 10656.152
4 1181868    152962     20.3       4092       6251       135.6 1483.52  8715.840
5  360493     28592     16.9       1034       5929        41.8  749.44  8624.235
6  158533     15233     11.3        408       6169        18.9  661.50  8387.990
```

Have you observed a slight difference in the way we sorted the data? We can save some time and space by merging the two steps into one line, however, it is sometimes easier to understand a command if it is split into separate stages.

4.1.7 Task 7

4.1.7.1 Task

What is the minimum and maximum value for population density in the crime data?

4.1.7.2 Guidance

The minimum and maximum values can be produced by generating standard descriptive statistics of the variables.

```r
summary(crime$popdens)
```

```
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max. 
  703.8  2797.1  4236.8  4967.5  7052.2 15264.8 
```

Before you finish, save the dataset under a new name. Never overwrite your original data!

```r
save(crime, file = "./assets/data/crime_v2.Rdata")
```

The above command tells R to use the crime dataset and save it as `crime_v2.Rdata`. Rdata is an R specific format. R can also save data in .csv format, that can be opened with any text editor or spreadsheet software:

```r
write.csv(crime, file = "./assets/data/crime_v2.csv", row.names = TRUE)
```

Now you are ready to answer the following questions on your own:

4.1.8 Further Exercises

1. Find the minimum and maximum number of police officers in the data set.
2. Create a new variable which measures the crime rate per 1,000 of population.
3. Is the city with the highest number of police officers also the city with the highest crime density?
4. How many crimes occurred in the richest city?
5. Is the richest city also the one with the highest number of police officers?
6. What is the average unemployment rate across these 46 U.S. cities?
7. Does the city with the highest unemployment rate also have the highest crime level?

5 Introduction to Regression Analysis

5.1 Example 1: Crime data

The example and instructions provided in this section is taken from (Riegler 2022).

Suppose you are examining the relationship between number of crimes and number of police officers. Below, we will generate descriptive statistics, create a scatter plot and see how we estimate OLS regression.

We will use the crime data set, which is already saved in `Rdata` format.

5.1.1 Task 1

Open `crime_v2.Rdata` (if it not already open). You may so this trough the menu or the command line using the `load()` function:

```
load("~/Desktop/R-workshops/assets/data/crime_v2.Rdata")
```

5.1.2 Task 2

Check the summary statistics for `crimes` and `officers` variables.

```
summary(crime[, c("crimes", "officers")])
```

```
     crimes          officers     
 Min.   :  5276   Min.   : 109.0  
 1st Qu.: 19658   1st Qu.: 402.8  
 Median : 32518   Median : 694.5  
 Mean   : 38123   Mean   : 902.1  
 3rd Qu.: 49434   3rd Qu.:1212.0  
 Max.   :152962   Max.   :4092.0  
```

```
# Standard deviation for the 'crimes' variable
sd_crimes <- sd(crime$crimes, na.rm = TRUE)

# Standard deviation for the 'officers' variable
sd_officers <- sd(crime$officers, na.rm = TRUE)

# Print the results
sd_crimes
```

```
[1] 27660.3
```

```
sd_officers
```

```
[1] 721.7255
```

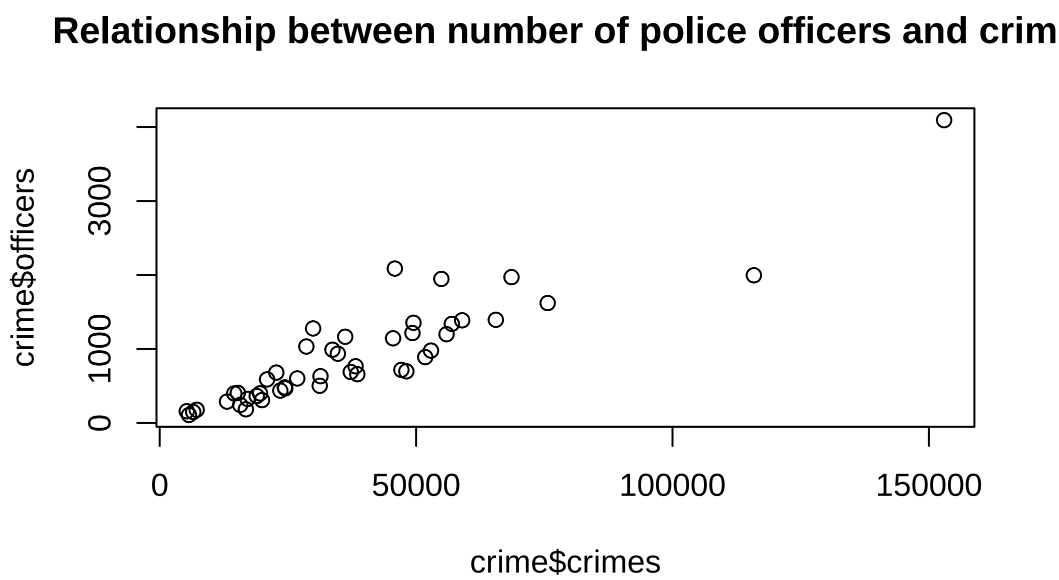
Note the `na.rm = TRUE` above. This argument ensures that nay `NA` values are removed before the calculation.

5.1.3 Task 3

In addition to checking summary statistics, it is always wise to visualise your data before getting into more complicated modelling.

For this task, generate a scatter plot with number of crimes on the y-axis and the number of police officers on the x-axis.

```
plot(crime$officers~crime$crimes,
     main = "Relationship between number of police officers and crime")
```



5.1.4 Task 4

Calculate the Covariance and the Correlation Coefficient between number of crimes and number of police officers. Comment on their values.

5.1.4.1 Guidance

A scatter plot is a good start for identifying relationships between two variables, but it is not sufficient to identify accurately how strong the relationship is between `crimes` and `officers`. There are two numerical statistics, that provide information about the relationship between two variables: The Covariance and the Correlation Coefficient.

To produce a Covariance matrix, use the following command:

```r
cov(crime$officers, crime$crimes)
```

```
[1] 18212436
```

The result is: 18,212,436! This number may appear to be too large but the value we obtain as covariance depends on the measuremetn levels of the variables. This measure (i.e. the covariance) does not provide any information on how strong this relationship between `crimes` and `officers` is. It only reveals that there is a positive relationship between the number of police officers and the number of crimes committed.

Instead of using the covariance, we can use a *standardised covariance* - the *correlation coefficient*. To calculate the correlation matrix, we only have to adjust slightly the covariance command.

```
cor(crime$officers, crime$crimes)
```

```
[1] 0.9123032
```

The correlation coefficient between number of officers and crimes is 0.91. We conclude that there is a strong positive relationship between our two variables.

5.1.5 Task 5

Regress the number of police officers on crimes and comment on:

- the sign and magnitude of the regression coefficients
- the goodness of fit of the estimated model.

```
lm(officers ~ crimes, data = crime)
```

```
Call:
lm(formula = officers ~ crimes, data = crime)

Coefficients:
(Intercept)       crimes
    -5.4183       0.0238
```

We can save this estimation under an object (please note that we use `model_1` below, but you may give any name as long as it satisfies the naming conventions):

```
model_1 <- lm(officers ~ crimes, data = crime)
# display the model
model_1
```

```
Call:
lm(formula = officers ~ crimes, data = crime)

Coefficients:
(Intercept)       crimes
    -5.4183       0.0238
```

Although we see the estimated coefficients in the above output we do not have information about the other statistics that we need to proceed. We use the `summary()` function below:

```r
summary(model_1)
```

```
Call:
lm(formula = officers ~ crimes, data = crime)

Residuals:
    Min      1Q  Median      3Q     Max
-756.64 -153.71  -25.75   89.64 1000.97

Coefficients:
             Estimate Std. Error t value Pr(>|t|)
(Intercept) -5.418291  75.587257  -0.072    0.943
crimes       0.023804   0.001611  14.777   <2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 298.9 on 44 degrees of freedom
Multiple R-squared:  0.8323,    Adjusted R-squared:  0.8285
F-statistic: 218.4 on 1 and 44 DF,  p-value: < 2.2e-16
```

The intercept term is not statistically significant.

The `crimes` variable is statistically significant at 0.1%. (Note the significance codes in the output).

The slope coefficient states that for every additional crime, we observe on average of 0.024 more police officers. Using more reader-friendly numbers, we can also infer that for every 1,000 additional crimes committed within a city, 24 more police officers are employed. Note how the latter way of phrasing makes more sense.

R^2 is the measure that provides information on the overall goodness of fit of the model. In this case it is 0.83. This means that 83% of the variation in police officers can be explained with the variation in number of crimes committed. Our estimated model has a good degree of explanatory power.

Looking at the F-statistic (218.4 with a p-value of almost zero), we can conclude that the model, overall, is statistically significant.

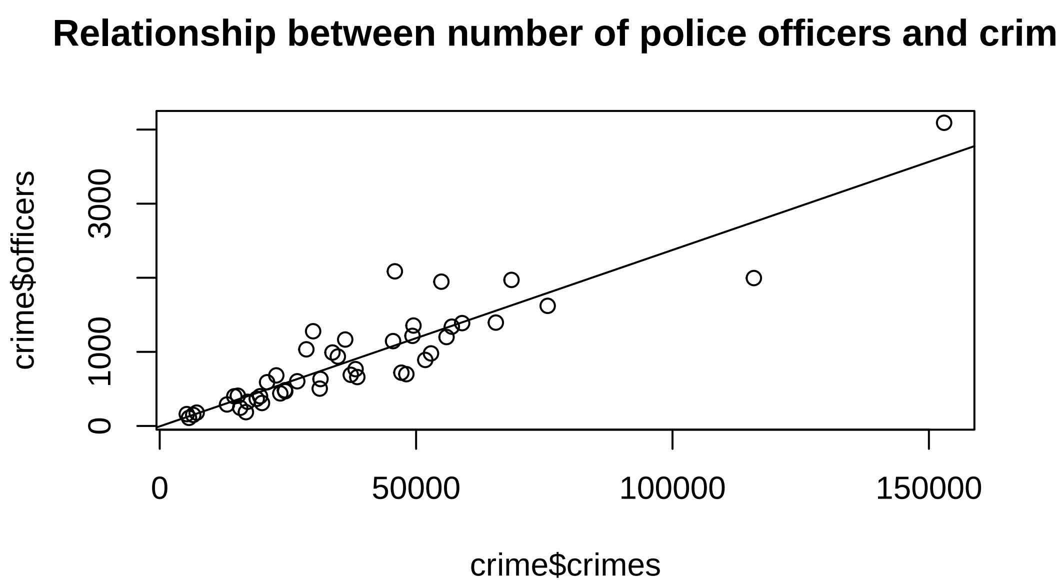
5.1.6 Task 6

Add a regression line to the scatter plot you created in Task 3.

5.1.6.1 Guidance

To add a regression line to the plot, we have to use the previously saved regression object `model_1` and add it to the previous scatter plot.

```
plot(crime$officers~crime$crimes,
     main = "Relationship between number of police officers and crime")
abline(model_1)
```



5.2 Example 2: Wage data

5.2.1 Task 1

5.2.1.1 Task

Import `wage.xls` data into R and view the first few rows of the data to have an idea about the contents of the variables, and then save the data in R format.

5.2.1.2 Guidance

Use `read_excel()` and `head()` functions.

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

# Import Excel data
wage2 <- read_excel("./assets/data/wage2.xls", sheet = "wage2")
```

```
head(wage2)
```

```
# A tibble: 6 x 15
   wage hours    IQ   KWW  educ exper tenure   age married south urban  sibs
  <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>  <dbl> <dbl>   <dbl> <dbl> <dbl> <dbl>
1   769    40    93    35    12    11      2    31       1     0     1     1
2   808    50   119    41    18    11     16    37       1     0     1     1
3   825    40   108    46    14    11      9    33       1     0     1     1
4   650    40    96    32    12    13      7    32       1     0     1     4
5   562    40    74    27    11    14      5    34       1     0     1    10
6  1400    40   116    43    16    14      2    35       1     0     1     1
# i 3 more variables: brthord <dbl>, meduc <dbl>, feduc <dbl>
```

```
# Save data in R format
save(wage2, file = "./assets/data/wage2.Rdata")
```

5.2.2 Task 2

5.2.2.1 Task

Label variable `educ` as “years of schooling” and `exper` as “years of experience”.

5.2.2.2 Guidance

We will need the **expss** package to label the variables. The installation and calling of the package is deactivated below since we already have done these steps above. After running the below command check the changes in the data from the Environment window on the top-right.

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

```
Loading required package: maditr
```

```
To drop variable use NULL: let(mtcars, am = NULL) %>% head()
```

```
wage2 <- apply_labels(wage2,
                      educ = "years of schooling",
                      exper = "years of experince")
```

5.2.3 Task 3

5.2.3.1 Task

Generate two new variables: hourly wage and logarithmic wage.

5.2.3.2 Guidance

```
# Generate new variables
wage2$hourly_wage <- wage2$wage / wage2$hours
wage2$ln_wage <- log(wage2$wage)
```

5.2.4 Task 4

5.2.4.1 Task

Check the summary statistics for (i) the `wage` variable, (ii) for all variables.

5.2.4.2 Guidance

We will use the `summary()` function.

```
# Summary statistics for the wage variable only
summary(wage2$wage)
```

```
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max. 
  115.0   669.0   905.0   957.9  1160.0  3078.0 
```

```
# Summary statistics for all variables in wage2 data
summary(wage2)
```

```
      wage            hours             IQ             KWW       
 Min.   : 115.0   Min.   :20.00   Min.   : 50.0   Min.   :12.00  
 1st Qu.: 669.0   1st Qu.:40.00   1st Qu.: 92.0   1st Qu.:31.00  
 Median : 905.0   Median :40.00   Median :102.0   Median :37.00  
 Mean   : 957.9   Mean   :43.93   Mean   :101.3   Mean   :35.74  
 3rd Qu.:1160.0   3rd Qu.:48.00   3rd Qu.:112.0   3rd Qu.:41.00  
 Max.   :3078.0   Max.   :80.00   Max.   :145.0   Max.   :56.00  
                                                                 
      educ           exper           tenure            age       
 Min.   : 9.00   Min.   : 1.00   Min.   : 0.000   Min.   :28.00  
 1st Qu.:12.00   1st Qu.: 8.00   1st Qu.: 3.000   1st Qu.:30.00  
 Median :12.00   Median :11.00   Median : 7.000   Median :33.00  
 Mean   :13.47   Mean   :11.56   Mean   : 7.234   Mean   :33.08  
 3rd Qu.:16.00   3rd Qu.:15.00   3rd Qu.:11.000   3rd Qu.:36.00  
 Max.   :18.00   Max.   :23.00   Max.   :22.000   Max.   :38.00  
                                                                 
    married          south            urban             sibs       
 Min.   :0.000   Min.   :0.0000   Min.   :0.0000   Min.   : 0.000  
 1st Qu.:1.000   1st Qu.:0.0000   1st Qu.:0.0000   1st Qu.: 1.000  
 Median :1.000   Median :0.0000   Median :1.0000   Median : 2.000  
 Mean   :0.893   Mean   :0.3412   Mean   :0.7176   Mean   : 2.941  
 3rd Qu.:1.000   3rd Qu.:1.0000   3rd Qu.:1.0000   3rd Qu.: 4.000  
 Max.   :1.000   Max.   :1.0000   Max.   :1.0000   Max.   :14.000  
                                                                   
    brthord           meduc           feduc        hourly_wage   
 Min.   : 1.000   Min.   : 0.00   Min.   : 0.00   Min.   :  2.30  
 1st Qu.: 1.000   1st Qu.: 8.00   1st Qu.: 8.00   1st Qu.: 15.07  
 Median : 2.000   Median :12.00   Median :10.00   Median : 21.02  
```

```
 Mean   : 2.277   Mean   :10.68   Mean   :10.22   Mean   : 22.32
 3rd Qu.: 3.000   3rd Qu.:12.00   3rd Qu.:12.00   3rd Qu.: 27.70
 Max.   :10.000   Max.   :18.00   Max.   :18.00   Max.   :102.60
 NA's   :83       NA's   :78      NA's   :194
    ln_wage
 Min.   :4.745
 1st Qu.:6.506
 Median :6.808
 Mean   :6.779
 3rd Qu.:7.056
 Max.   :8.032
```

5.2.5 Task 5

5.2.5.1 Task

Calculate the correlation coefficient between wage and education.

5.2.5.2 Guidance

We can calculate the correlation coefficients using the `cor()` function. In the first example below, the correlation coefficient is reported as a single number, while in the second example, we get a correlation matrix.

In most of empirical work, we are usually interested with pairwise correlations among all variables. Hence, we may use the correlation matrix to check the binary correlations among all variables in our sample. This is provided in the third example below.

The `"use = complete.obs"` added to the commands below asks R to handle missing values by casewise deletion.

```r
# Correlation
cor(wage2$wage, wage2$educ)
```

```
[1] 0.3271087
```

```r
# Correlation
cor(wage2[, c("wage", "educ")], use = "complete.obs")
```

```
          wage      educ
wage 1.0000000 0.3271087
educ 0.3271087 1.0000000
```

```r
cor(wage2, use = "pairwise.complete.obs")
```

```
                    wage        hours          IQ         KWW        educ
wage         1.000000000 -0.009504302  0.30908783  0.32613058  0.32710869
hours       -0.009504302  1.000000000  0.07383930  0.11388938  0.09100889
IQ           0.309087827  0.073839301  1.00000000  0.41351552  0.51569701
KWW          0.326130577  0.113889381  0.41351552  1.00000000  0.38813424
educ         0.327108690  0.091008888  0.51569701  0.38813424  1.00000000
exper        0.002189702 -0.062126227 -0.22491253  0.01745245 -0.45557312
tenure       0.128266391 -0.055528006  0.04215883  0.14139800 -0.03616655
age          0.156701761  0.024811636 -0.04374091  0.39305297 -0.01225396
married      0.136582670  0.032563350 -0.01466753  0.08994782 -0.05856602
south       -0.159387287 -0.029519177 -0.20978466 -0.09439242 -0.09703298
urban        0.198406472  0.016573046  0.03893553  0.09819025  0.07215091
sibs        -0.159203728 -0.049602555 -0.28477277 -0.28497534 -0.23928810
brthord     -0.145485385 -0.043129582 -0.17943947 -0.15358472 -0.20499246
meduc        0.214831839  0.076619806  0.33180383  0.24079168  0.36423913
feduc        0.237586922  0.063172297  0.34390758  0.23488927  0.42692545
hourly_wage  0.931240501 -0.317645466  0.26502635  0.26059936  0.27167136
ln_wage      0.953141156 -0.047219079  0.31478770  0.30627128  0.31211665
                   exper      tenure          age      married       south
wage         0.002189702  0.12826639  0.156701761  0.136582670 -0.15938729
hours       -0.062126227 -0.05552801  0.024811636  0.032563350 -0.02951918
IQ          -0.224912532  0.04215883 -0.043740911 -0.014667528 -0.20978466
KWW          0.017452446  0.14139800  0.393052967  0.089947816 -0.09439242
educ        -0.455573115 -0.03616655 -0.012253956 -0.058566019 -0.09703298
exper        1.000000000  0.24365440  0.495329763  0.106349115  0.02125724
tenure       0.243654402  1.00000000  0.270601647  0.072605374 -0.06169141
age          0.495329763  0.27060165  1.000000000  0.106980249 -0.02947768
married      0.106349115  0.07260537  0.106980249  1.000000000  0.02275672
south        0.021257241 -0.06169141 -0.029477681  0.022756718  1.00000000
urban       -0.047385845 -0.03848582 -0.006749288 -0.040248179 -0.10989797
sibs         0.064310470 -0.03916116 -0.040719238 -0.004327422  0.06631979
brthord      0.088300019 -0.02847775  0.005435916 -0.014737189  0.09370679
meduc       -0.186317286 -0.01496769 -0.029319099 -0.022763437 -0.15787359
feduc       -0.256792630 -0.05924123 -0.071303285 -0.020324390 -0.17236334
hourly_wage  0.017757793  0.13541822  0.126683019  0.115115701 -0.14716118
ln_wage      0.020601158  0.18585262  0.161822314  0.149975894 -0.19481092
```

```
                 urban         sibs      brthord       meduc       feduc
wage         0.198406472 -0.159203728 -0.145485385  0.21483184  0.23758692
hours        0.016573046 -0.049602555 -0.043129582  0.07661981  0.06317230
IQ           0.038935525 -0.284772765 -0.179439471  0.33180383  0.34390758
KWW          0.098190247 -0.284975345 -0.153584717  0.24079168  0.23488927
educ         0.072150908 -0.239288104 -0.204992462  0.36423913  0.42692545
exper       -0.047385845  0.064310470  0.088300019 -0.18631729 -0.25679263
tenure      -0.038485824 -0.039161158 -0.028477749 -0.01496769 -0.05924123
age         -0.006749288 -0.040719238  0.005435916 -0.02931910 -0.07130328
married     -0.040248179 -0.004327422 -0.014737189 -0.02276344 -0.02032439
south       -0.109897970  0.066319792  0.093706790 -0.15787359 -0.17236334
urban        1.000000000 -0.031468824  0.002419787  0.03402366  0.11223944
sibs        -0.031468824  1.000000000  0.593913799 -0.28715120 -0.23202649
brthord      0.002419787  0.593913799  1.000000000 -0.27593376 -0.23037060
meduc        0.034023660 -0.287151198 -0.275933760  1.00000000  0.57649476
feduc        0.112239438 -0.232026494 -0.230370600  0.57649476  1.00000000
hourly_wage  0.189240304 -0.131364072 -0.120293460  0.18348733  0.20469678
ln_wage      0.203797585 -0.152809172 -0.141852712  0.21357476  0.22338514
            hourly_wage     ln_wage
wage         0.93124050  0.95314116
hours       -0.31764547 -0.04721908
IQ           0.26502635  0.31478770
KWW          0.26059936  0.30627128
educ         0.27167136  0.31211665
exper        0.01775779  0.02060116
tenure       0.13541822  0.18585262
age          0.12668302  0.16182231
married      0.11511570  0.14997589
south       -0.14716118 -0.19481092
urban        0.18924030  0.20379758
sibs        -0.13136407 -0.15280917
brthord     -0.12029346 -0.14185271
meduc        0.18348733  0.21357476
feduc        0.20469678  0.22338514
hourly_wage  1.00000000  0.89974921
ln_wage      0.89974921  1.00000000
```

The above table is informative but the reported numbers have far too many decimals. It is distracting our focus. Below, we round these in two decimal points, which is enough to have an idea about the strength of the correlation between our variables

```r
# Calculate pairwise correlations and store them under name cor_matrix
cor_matrix <- cor(wage2, use = "pairwise.complete.obs")

# Round the correlation values to 2 decimal places and save them under the name rounded_cor_m
rounded_cor_matrix <- round(cor_matrix, 2)

# Display the rounded correlation matrix
print(rounded_cor_matrix)
```

```
             wage hours    IQ   KWW  educ exper tenure   age married south
wage         1.00 -0.01  0.31  0.33  0.33  0.00   0.13  0.16    0.14 -0.16
hours       -0.01  1.00  0.07  0.11  0.09 -0.06  -0.06  0.02    0.03 -0.03
IQ           0.31  0.07  1.00  0.41  0.52 -0.22   0.04 -0.04   -0.01 -0.21
KWW          0.33  0.11  0.41  1.00  0.39  0.02   0.14  0.39    0.09 -0.09
educ         0.33  0.09  0.52  0.39  1.00 -0.46  -0.04 -0.01   -0.06 -0.10
exper        0.00 -0.06 -0.22  0.02 -0.46  1.00   0.24  0.50    0.11  0.02
tenure       0.13 -0.06  0.04  0.14 -0.04  0.24   1.00  0.27    0.07 -0.06
age          0.16  0.02 -0.04  0.39 -0.01  0.50   0.27  1.00    0.11 -0.03
married      0.14  0.03 -0.01  0.09 -0.06  0.11   0.07  0.11    1.00  0.02
south       -0.16 -0.03 -0.21 -0.09 -0.10  0.02  -0.06 -0.03    0.02  1.00
urban        0.20  0.02  0.04  0.10  0.07 -0.05  -0.04 -0.01   -0.04 -0.11
sibs        -0.16 -0.05 -0.28 -0.28 -0.24  0.06  -0.04 -0.04    0.00  0.07
brthord     -0.15 -0.04 -0.18 -0.15 -0.20  0.09  -0.03  0.01   -0.01  0.09
meduc        0.21  0.08  0.33  0.24  0.36 -0.19  -0.01 -0.03   -0.02 -0.16
feduc        0.24  0.06  0.34  0.23  0.43 -0.26  -0.06 -0.07   -0.02 -0.17
hourly_wage  0.93 -0.32  0.27  0.26  0.27  0.02   0.14  0.13    0.12 -0.15
ln_wage      0.95 -0.05  0.31  0.31  0.31  0.02   0.19  0.16    0.15 -0.19
            urban  sibs brthord meduc feduc hourly_wage ln_wage
wage         0.20 -0.16   -0.15  0.21  0.24        0.93    0.95
hours        0.02 -0.05   -0.04  0.08  0.06       -0.32   -0.05
IQ           0.04 -0.28   -0.18  0.33  0.34        0.27    0.31
KWW          0.10 -0.28   -0.15  0.24  0.23        0.26    0.31
educ         0.07 -0.24   -0.20  0.36  0.43        0.27    0.31
exper       -0.05  0.06    0.09 -0.19 -0.26        0.02    0.02
tenure      -0.04 -0.04   -0.03 -0.01 -0.06        0.14    0.19
age         -0.01 -0.04    0.01 -0.03 -0.07        0.13    0.16
married     -0.04  0.00   -0.01 -0.02 -0.02        0.12    0.15
south       -0.11  0.07    0.09 -0.16 -0.17       -0.15   -0.19
urban        1.00 -0.03    0.00  0.03  0.11        0.19    0.20
sibs        -0.03  1.00    0.59 -0.29 -0.23       -0.13   -0.15
brthord      0.00  0.59    1.00 -0.28 -0.23       -0.12   -0.14
meduc        0.03 -0.29   -0.28  1.00  0.58        0.18    0.21
```

```
feduc         0.11 -0.23   -0.23  0.58  1.00         0.20    0.22
hourly_wage   0.19 -0.13   -0.12  0.18  0.20         1.00    0.90
ln_wage       0.20 -0.15   -0.14  0.21  0.22         0.90    1.00
```

5.2.6 Task 6

5.2.6.1 Task

Examine the relationship between education and wage using a scatter plot.

5.2.6.2 Guidance

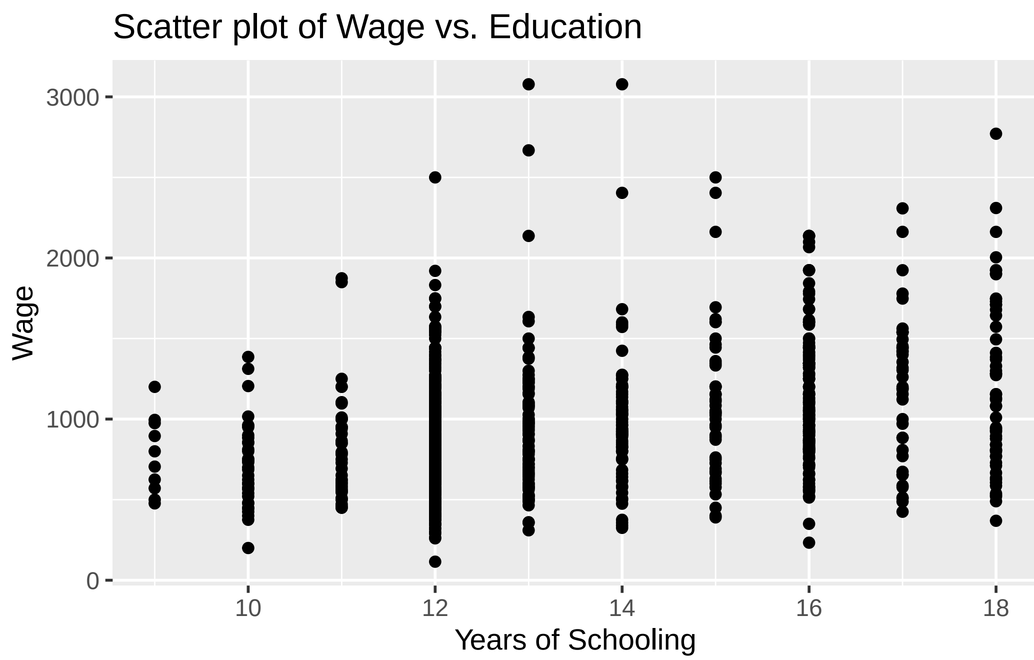
We use the `ggplot2` package to draw plots. First install the package and call the library.

```
# install.packages("ggplot2")
 library(ggplot2)
```

```
Attaching package: 'ggplot2'

The following object is masked from 'package:expss':

    vars
```

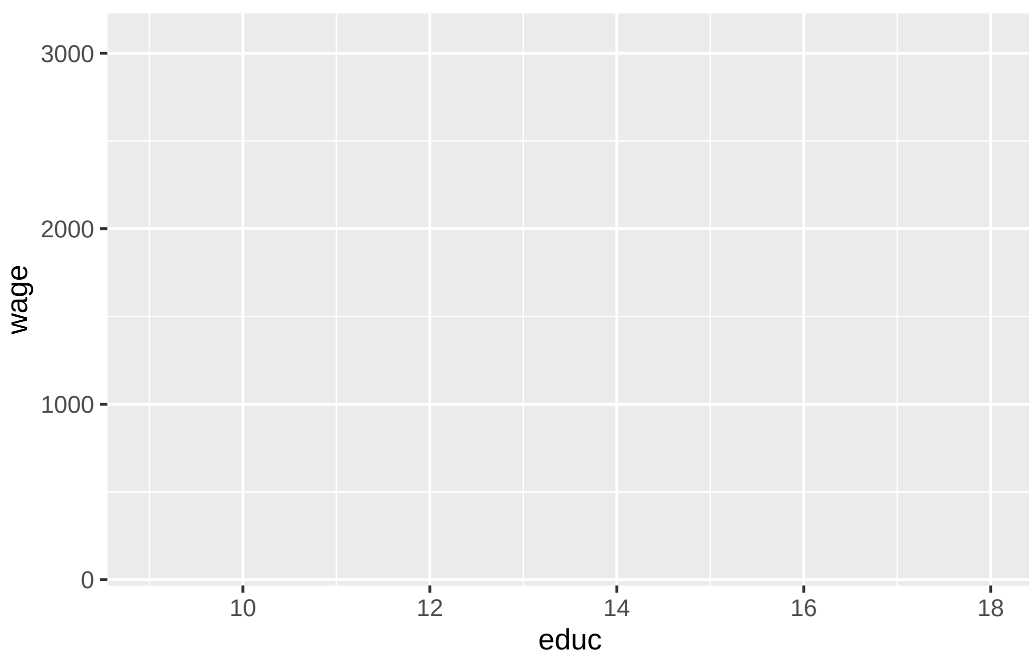
Education is expected to have a positive impact on wage. In our scatter plot, `educ` will be on the horizontal-axis while `wage` will be on the vertical-axis.

```
# Scatter plot
ggplot(wage2, aes(x = educ, y = wage)) +
  geom_point() +
  labs(title = "Scatter plot of Wage vs. Education", x = "Years of Schooling", y = "Wage")
```



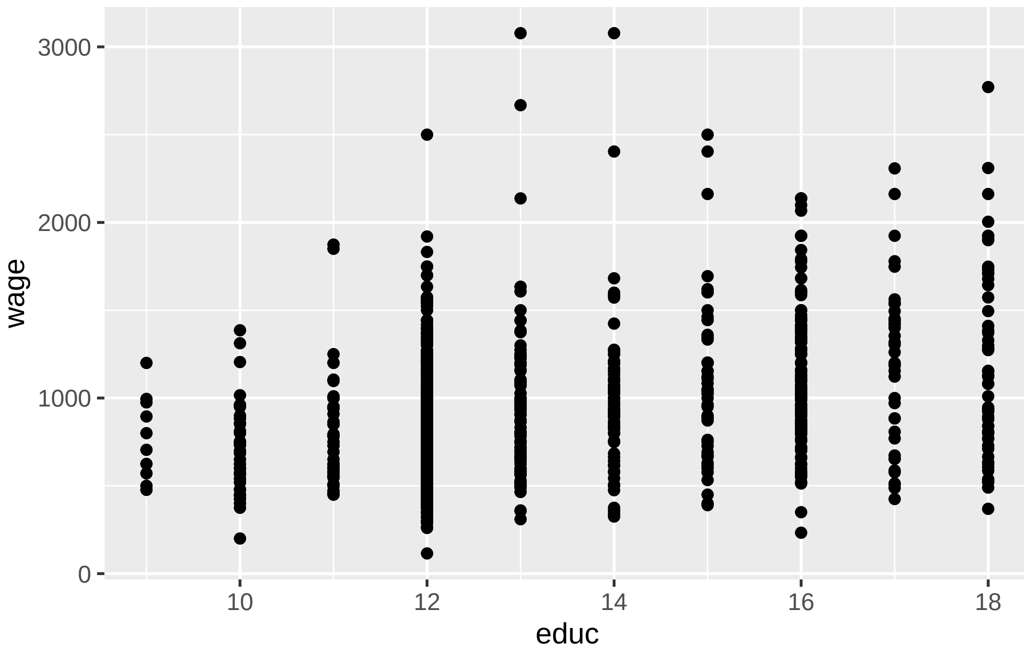
You see above the full set of lines to create this plot. But let us do this step by step to have a better understanding. First, we bring the `educ` and `wage` variables from the `wage2` data and position these on our plot.

```
ggplot(wage2, aes(x = educ, y = wage))
```



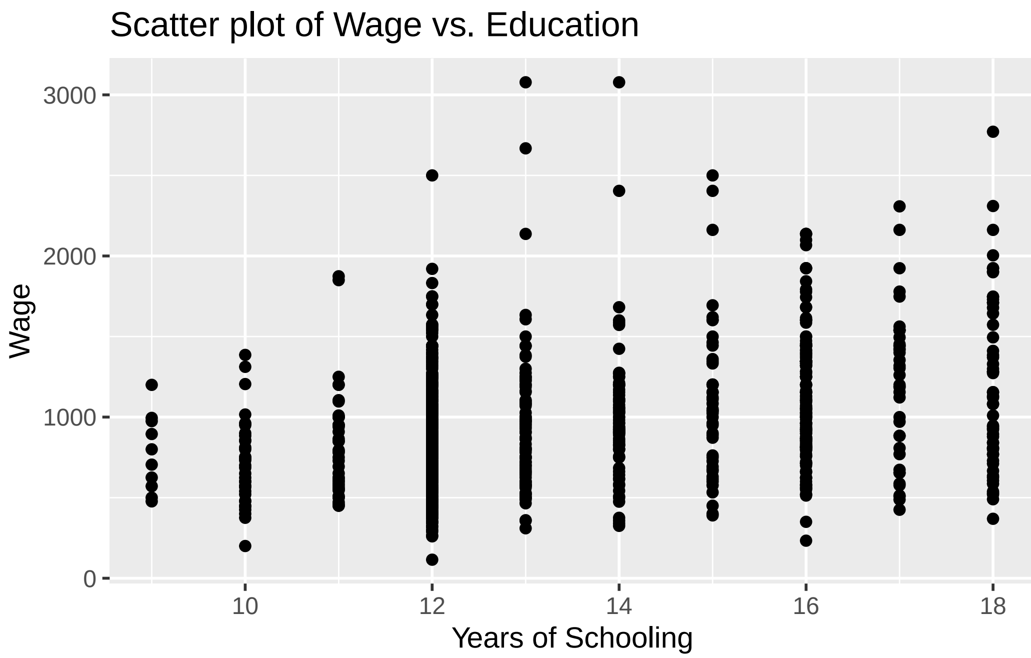
We then add (using the + sign), the observations in our data, represented by dots.

```
ggplot(wage2, aes(x = educ, y = wage)) +
  geom_point()
```



It is always good practice to give a title for your plot. Notice also that the horizontal and vertical axes above are labelled by the variable names. We may also replace these with proper definitions of the variables. This is to make it easier for the readers to understand your plots:

```
ggplot(wage2, aes(x = educ, y = wage)) +
  geom_point() +
  labs(title = "Scatter plot of Wage vs. Education", x = "Years of Schooling", y = "Wage")
```



5.2.7 Task 7

5.2.7.1 Task

Tabulate the `urban` variable to see the distribution of observations in rural and urban areas

5.2.7.2 Guidance

We use the `table()` function for that purpose.

```
table(wage2$urban)
```

```

  0   1 
264 671 
```

5.2.8 Task 8

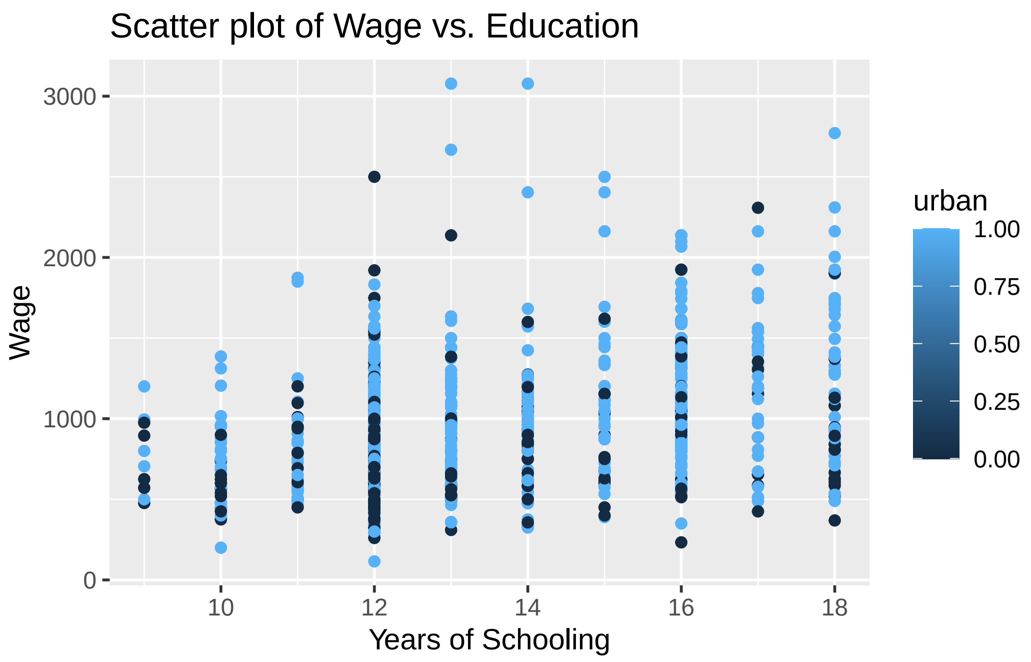
5.2.8.1 Task

Let's say we are interested to plot the education-wage relationship differentiating between people in rural and urban areas. Replicate the scatter plot above, but this time, using different colors for rural and urban.

5.2.8.2 Guidance

Notice how we add the `color = urban` option below. We do the same for the label too.

```
# Scatter plot - colored by urban
ggplot(wage2, aes(x = educ, y = wage, color = urban)) +
  geom_point() +
  labs(title = "Scatter plot of Wage vs. Education", x = "Years of Schooling", y = "Wage", co
```



The labelling of the above plot looks as if we have a range of values for the urban variable, changing from zero to one. The urban variable, in fact, is a dummy, taking two values only: zero for rural and one for urban residence. If you look into this variable entry in more detail, you will see that it is stored as `num`. We can change this using the `factor()` function. Instead of overriding the urban variable, let's create a new variable `urban_residence` to see a comparison.

```
wage2$urban_residence <- factor(wage2$urban, levels = c(0,1), labels = c("rural", "urban"))
```

Below, we view the two variables using R's `dplyr` package.

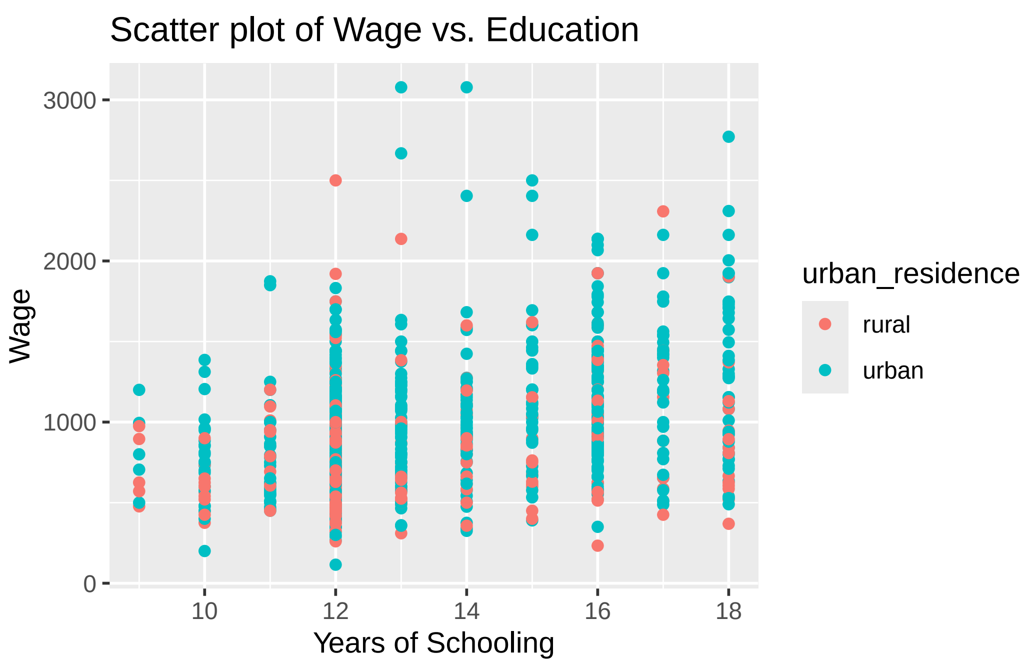
```
# install.packages("dplyr")
library(dplyr)
```

```
Attaching package: 'dplyr'

The following objects are masked from 'package:expss':

    compute, contains, na_if, recode, vars, where

The following objects are masked from 'package:maditr':

    between, coalesce, first, last

The following objects are masked from 'package:stats':

    filter, lag

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

    intersect, setdiff, setequal, union
```

```
View(select(wage2, urban, urban_residence))
```

Let's re-run our scatter plot code again (but replacing `urban` with `urban_residence`:

```
# Scatter plot - colored by urban
ggplot(wage2, aes(x = educ, y = wage, color = urban_residence)) +
  geom_point() +
  labs(title = "Scatter plot of Wage vs. Education", x = "Years of Schooling", y = "Wage", co
```



5.2.9 Task 9

5.2.9.1 Task

Estimate a regression model where wage is regressed on education. Interpret the results.

5.2.9.2 Guidance

We use the `lm()` function to estimate linear regression models. You may read `~` in `wage ~ educ` below as "approximately modelled as" James et al. (2023). We may also say "wage is regressed on education".

```
# Linear regression
model_1 <- lm(wage ~ educ, data = wage2)
summary(model_1)
```

```
Call:
lm(formula = wage ~ educ, data = wage2)

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

```
-877.38 -268.63  -38.38  207.05 2148.26

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)  146.952     77.715   1.891   0.0589 .
educ          60.214      5.695  10.573   <2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 382.3 on 933 degrees of freedom
Multiple R-squared:  0.107, Adjusted R-squared:  0.106
F-statistic: 111.8 on 1 and 933 DF,  p-value: < 2.2e-16
```

In the above regression output, we see that education has a statistically significant impact on wages. Each year of schooling increases wage by around £60, on average. The F test tells us that the regression model has an explanatory power, even though the R-squared value is low.

5.2.10 Task 10

5.2.10.1 Task

Using the regression model above, predict what the wage would be for given values of education (how much do we expect the wage would be for given years of schooling).

5.2.10.2 Guidance

Below, we recall `model_1` to calculate predicted values; save the predictions under name `wage_hat` under `wage2` data.

```
# Save predicted values under name wage_hat
wage2$wage_hat <- predict(model_1)
```

5.2.11 Task 11

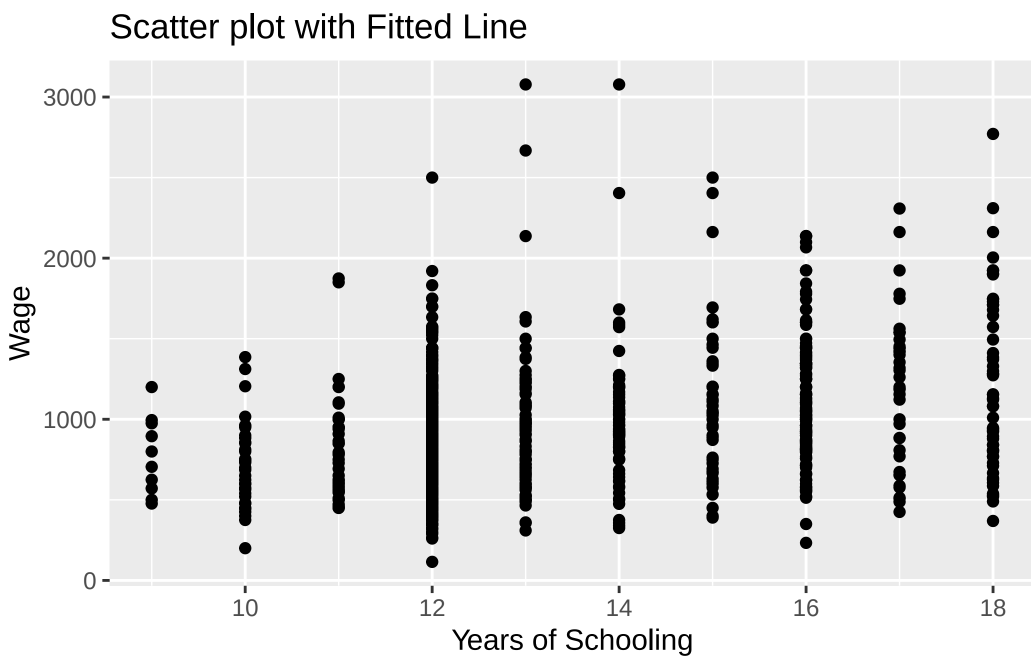
5.2.11.1 Task

Add the estimated regression line to the wage-education scatter plot.

5.2.11.2 Guidance

We will be adding the regression line to the scatter plot we produced above. We use `geom_smooth` for this purpose. Let's first remember what we did before:

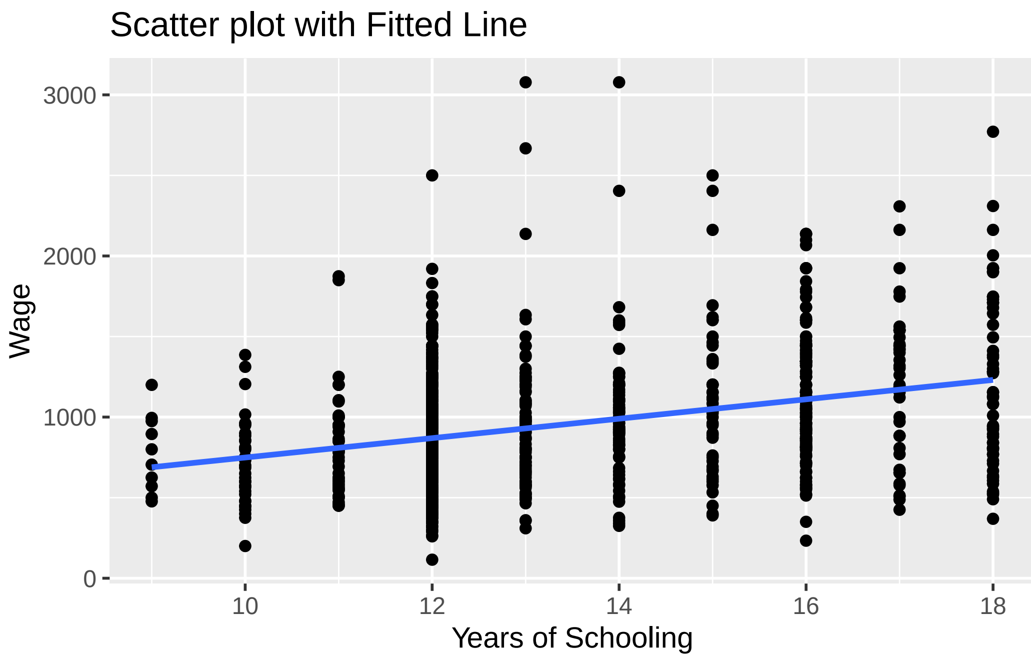
```
# Scatter plot of education and wage
ggplot(wage2, aes(x = educ, y = wage)) +
  geom_point() +
  labs(title = "Scatter plot with Fitted Line", x = "Years of Schooling", y = "Wage")
```



Now, let's add the regression line:

```
# Scatter plot with fitted line
ggplot(wage2, aes(x = educ, y = wage)) +
  geom_point() +
  geom_smooth(method = "lm", se = FALSE) +
  labs(title = "Scatter plot with Fitted Line", x = "Years of Schooling", y = "Wage")
```

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



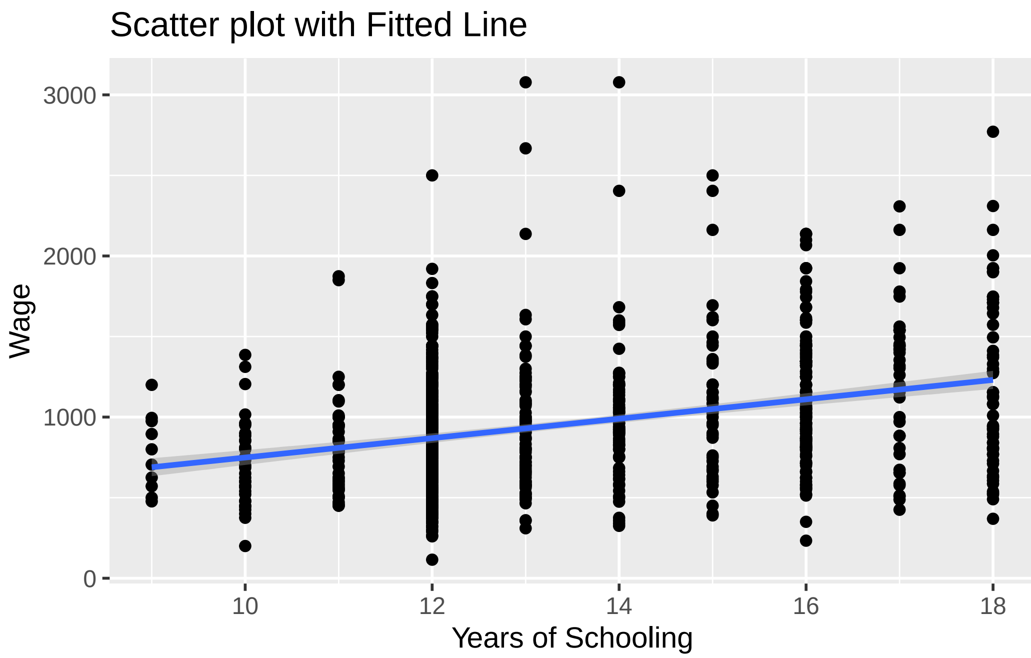
the `geom_smooth(method = "lm")` asks R to add a line estimating a "linear model" (i.e. a regression) of wage on educ.

Note that we could save this plot as an object by assigning it a name on the left hand side of the command. We will do that below and name the plot as `scatter_wage_educ`.

Can you guess what the plot would look if we changed `se = FALSE` to `se = TRUE` above? We can also try that below:

```
# Scatter plot with fitted line
scatter_wage_educ <- ggplot(wage2, aes(x = educ, y = wage)) +
                        geom_point() +
                        geom_smooth(method = "lm", se = TRUE) +
                        labs(title = "Scatter plot with Fitted Line", x = "Years of Schooling
print(scatter_wage_educ)
```

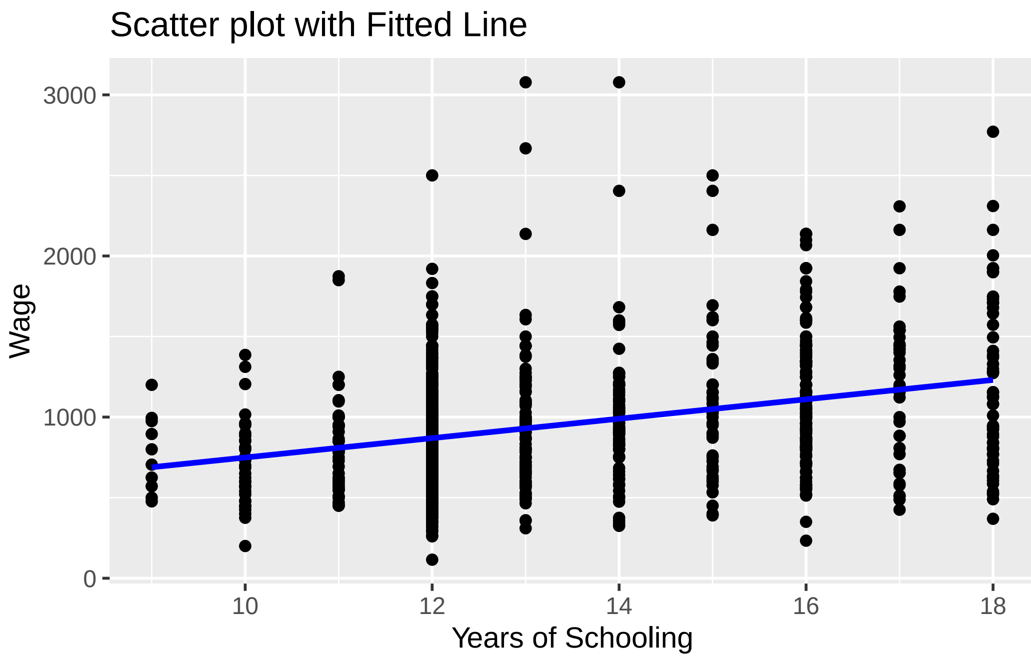
```
`geom_smooth()` using formula = 'y ~ x'
```



We could also add this sample regression line by using the `wage_hat` variable. `wage_hat` shows the predicted value of wage given observed values of education.

```
# Scatter plot with fitted line
# we add the wage_hat variable
ggplot(wage2, aes(x = educ, y = wage)) +
  geom_point() +
  geom_line(aes(y = wage_hat), color = "blue", size = 1) +
  labs(title = "Scatter plot with Fitted Line", x = "Years of Schooling", y = "Wage")
```

```
Warning: Using `size` aesthetic for lines was deprecated in ggplot2 3.4.0.
i Please use `linewidth` instead.
```



Note that we used `geom_line()` this time to add a line plot of an already existing variable in the data set.

- `ggplot(wage2, aes(x = educ, y = wage))` creates a canvas, a plot area with educ at the horizontal and wage at the vertical axis
- **`geom_point()`** adds a scatterplot of `wage` against `educ`.
- **`geom_line(aes(y = wage_hat))`** adds the line for the predicted `wage_hat` values. The `aes(y = wage_hat)` ensures the line graph uses `wage_hat` on the y-axis while sharing the x-axis (`educ`).
- **`color`** and **`size`** are optional for styling the line. Try experimenting with these and observe the changes.

5.2.12 Task 12

5.2.12.1 Task

Estimate a multiple regression model by adding experience and urban residence into the above regression. Save it under name `model_2`

5.2.12.2 Guidance

We will add `exper` and `urban` variables into the regression model using `+` sign.

```r
# Linear regression
model_2 <- lm(wage ~ educ + exper + urban, data = wage2)
summary(model_2)
```

```

Call:
lm(formula = wage ~ educ + exper + urban, data = wage2)

Residuals:
    Min      1Q  Median      3Q     Max 
-799.67 -234.04  -34.26  197.89 2119.62 

Coefficients:
            Estimate Std. Error t value Pr(>|t|)    
(Intercept) -362.821    106.419  -3.409 0.000679 ***
educ          74.119      6.193  11.968  < 2e-16 ***
exper         17.940      3.105   5.777 1.03e-08 ***
urban        160.306     26.920   5.955 3.69e-09 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 369.5 on 931 degrees of freedom
Multiple R-squared:  0.1676,	Adjusted R-squared:  0.1649 
F-statistic: 62.47 on 3 and 931 DF,  p-value: < 2.2e-16
```

How does `model_2` compare to `model_1`?

5.2.13 Task 13

5.2.13.1 Task

Save your data to keep the newly created `hourly_wage` and `ln_wage` variables.

5.2.13.2 Guidance

```r
# Save data in R format
save(wage2, file = "./assets/data/wage2.Rdata")
```

5.2.14 A Gentle Introduction to dplyr library

The `dplyr` library comes with R's `tidyverse` package. The `ggplot2` library we used above to produce plots is also a part of the `tidyverse` package.

I will replicate below a few of the tasks that we performed above using the `dplyr` library

5.2.14.1 Viewing data

We have seen before to use `View` to see the contents of data in a spreadsheet format:

```
View(wage2)
```

We may use `dplyr` to select variables for viewing. Using `select` allows us to "keep or drop columns using their names and types".

```
View(select(wage2, wage, educ, exper, urban, urban_residence))
```

5.2.14.2 Generating new variables

We used the following lines to create `hourly_wage` and `ln_wage` variables:

```
# Generate new variables
wage2$hourly_wage <- wage2$wage / wage2$hours
wage2$ln_wage <- log(wage2$wage)
```

`dplyr` 's `mutate` us used to "create, modify, and delete columns". Let us create a new data frame, `wage2_new` to see what it does:

```
wage2_new <- wage2 %>%
  mutate (
    hourly_wage_n = wage / hours,
    ln_wage_n = log(wage)
  )
```

In the above lines, we create a new data frame based on `wage2` . Note the `%>%` above. This is a part of the command and is called the `pipe operator`. It helps us to simply the code and do the operations one step after another. We first call `wage2` and create the new variables, `hourly_wage_n` and `ln_wage_n` .

Note how we avoided the use of `wage2$` every time we referred to a variable in wage2 data.

Another example we used to create a new variable was when we predicted values of wage for given levels of education after estimating model_1.

Below is the code we used:

```
wage2$wage_hat <- predict(model_1)
```

We can do this as follows using `dplyr`

```
wage2 <- wage2 %>%
  mutate(
    wage_hat_n = predict(model_1)
  )
```

5.2.14.3 Tabulating Variables

We used the code below to tabulate values of `urban` variable

```
table(wage2$urban)
```

```

  0   1 
264 671 
```

we may use `count` in `dplyr` for this purpose

```
wage2 %>%
  count(urban)
```

```
# A tibble: 2 x 2
  urban     n
  <dbl> <int>
1     0   264
2     1   671
```

Remember that we could save this as a new object:

```
urban_table <- wage2 %>%
  count(urban)
print(urban_table)
```

```
# A tibble: 2 x 2
  urban     n
  <dbl> <int>
1     0   264
2     1   671
```

Which output do you prefer?

5.3 Further Exercises

Download the data set called EAWE21.Rdata from the module page on Aula and save it. This is a subset of the Educational Attainment and Wage Equations data set used in Dougherty (2016) available from https://global.oup.com/uk/orc/busecon/economics/dougherty5e/student/datasets/eawe/. For this exercise we are interested in two variables:

- EXP : Total out-of-school work experience (years) as of the 2002 interview
- EARNINGS : Current hourly earnings in $ reported at the 2002 interview

5.3.1 Tasks

1. Calculate summary statistics (mean, median, minimum, maximum) for the variables EXP and EARNINGS
2. Draw scatter plot of EARNINGS on EXP.
3. Calculate the covariance and correlation between earnings and exp and comment on the values
4. Regress EARNINGS on EXP and comment on
    1. the sign and size of the regression coefficients
    2. the goodness of fits of the estimated model.
5. Add a regression line to the scatter plot.

Part III

Seminar 3 (4 February 2025)

6 Multiple Regression and Diagnostic Checks

6.1 Example: wage data

We will use the wage2 data set, which is already saved in `Rdata` format.

6.1.1 Task 1

Open `wage2.Rdata` (if it is not already open). You may so this through the menu or the command line using the `load()` function:

```
load("~/Desktop/R-workshops/assets/data/wage2.Rdata")
```

6.1.2 Task 2

Estimate a multiple regression model by regressing `wage` on `IQ`, `educ`, `exper`, `urban, married` and save it under name `model_3` . Display the estimation results.

6.1.2.1 Guidance

```
# Linear regression
model_3 <- lm(wage ~ IQ + educ + exper + urban, data = wage2)
summary(model_3)
```

```
Call:
lm(formula = wage ~ IQ + educ + exper + urban, data = wage2)

Residuals:
    Min      1Q  Median      3Q     Max
-797.64 -229.84  -38.35  185.10 2082.22
```

```
Coefficients:
             Estimate Std. Error t value Pr(>|t|)
(Intercept) -628.8654   115.5135  -5.444 6.66e-08 ***
IQ             5.0564     0.9234   5.476 5.60e-08 ***
educ          56.0554     6.9340   8.084 1.94e-15 ***
exper         17.7194     3.0583   5.794 9.41e-09 ***
urban        159.9813    26.5107   6.035 2.30e-09 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 363.9 on 930 degrees of freedom
Multiple R-squared:  0.1936,	Adjusted R-squared:  0.1901 
F-statistic: 55.81 on 4 and 930 DF,  p-value: < 2.2e-16
```

6.1.3 Task 3

6.1.3.1 Task

Test for the normality of the residuals

6.1.3.2 Guidance

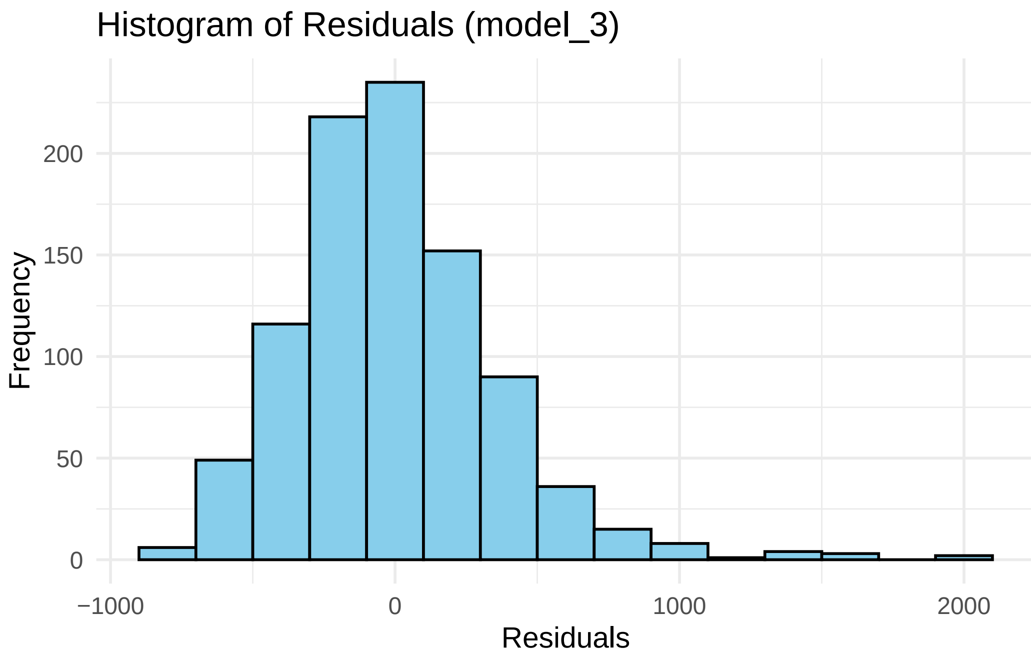
We will be using the Jarque-Bera test for this purpose.

We first save the residuals from `model_3` .

```r
wage2$resid_m3 <- residuals(model_3)
```

Plot the residuals to see the distribution. Please note that is not a part of the test but visualisation helps us to understand the data better.

```r
library(ggplot2)
ggplot(wage2, aes(x = resid_m3)) +
  geom_histogram(binwidth = 200, fill = "skyblue", color = "black") +
  labs(title = "Histogram of Residuals (model_3)", x = "Residuals", y = "Frequency") +
  theme_minimal()
```

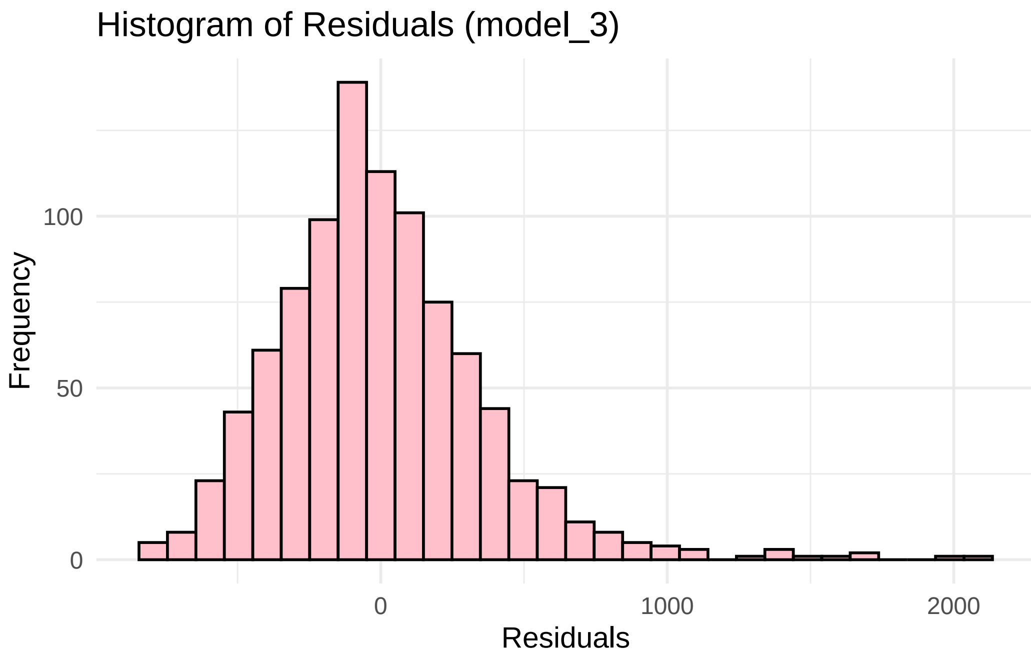


- `aes(x = resid)` specifies the residuals as the variable for the x-axis.
- `geom_histogram()` is used to create the histogram:
  - `binwidth = 200` controls the width of the bins. You can adjust this depending on how detailed you want the histogram to be.
  - `fill` sets the color inside the bars, and `color` adds a border around them for better visibility.
- `labs()` adds labels for the title and axes.
- `theme_minimal()` gives a clean, simple look to the plot - try the plot with and without this.

You may also let `ggplot` choose the number of bins automatically:

```
ggplot(wage2, aes(x = resid_m3)) +
  geom_histogram(fill = "pink", color = "black") +
  labs(title = "Histogram of Residuals (model_3)", x = "Residuals", y = "Frequency") +
  theme_minimal()
```

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



We may use `jarque.bera.test` for the normality test. It is in the `tseries` package.

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

```
Registered S3 method overwritten by 'quantmod':
  method            from
  as.zoo.data.frame zoo
```

```
jarque.bera.test(wage2$resid_m3)
```

```

    Jarque Bera Test

data:  wage2$resid_m3
X-squared = 699.59, df = 2, p-value < 2.2e-16
```

The p-value of the test is almost zero. We reject the null hypothesis of normal distribution. The residuals from model_3 are **not** normally distributed.

6.1.4 Task 4

6.1.4.1 Task

Test for the functional form.

6.1.4.2 Guidance

We may use this to check whether there are any omitted variables or non-linearity in the model. The test is Ramsey RESET.

```
library(lmtest)
```

```
Loading required package: zoo


Attaching package: 'zoo'


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

    as.Date, as.Date.numeric
```

```
resettest(model_3)
```

```

    RESET test

data:  model_3
RESET = 3.8665, df1 = 2, df2 = 928, p-value = 0.02127
```

The default `resettest` includes second and third powers of the fitted values in the test regression. You may change this using the `power` option. Below we include from second to the fourth power of fitted values.

```
resettest(model_3, power = 2:4)
```

```
	RESET test

data:  model_3
RESET = 2.8504, df1 = 3, df2 = 927, p-value = 0.03646
```

The decision depends on the chosen significance level. We reject the null hypothesis of correct functional form if we choose a 5% significance level.

6.1.5 Task 5

6.1.5.1 Task

Test for heteroscedasticity.

6.1.5.2 Guidance

We apply Breusch-Pagan heteroscedasticity test.

```r
bptest(model_3)
```

```
	studentized Breusch-Pagan test

data:  model_3
BP = 16.355, df = 4, p-value = 0.002578
```

The p-value is smaller than 0.05. Hence, we reject the null of no heteroscedasticity at 5% significance level. There is heteroscedasticity.

6.1.6 Task 6

6.1.6.1 Task

Test for autocorrelation in the model

6.1.6.2 Guidance

This is a trick question! Autocorrelation problem is related to time series data whereas we have cross-section data here. Autocorrelation problem is irrelevant here.

6.1.7 Task 7

6.1.7.1 Task

Replicate the above using logarithmic wages. Has there been a change in model diagnostics? Which form do you prefer to use for inference?

6.1.7.2 Guidance

You may use the script file to copy-paste all the code and make the minor changes (i.e. replacement of `wage` with `ln_wage`).

References

James, Gareth, Daniela Witten, Trevor Hastie, and Rob Tibshirani. 2023. *An Introduction to Statistical Learning*. 2nd edition. Springer. https://www.statlearning.com.

Kleiber, C., and A. Zeileis. 2008. *Applied Econometrics with r*. Springer.

Riegler, Robert. 2022. "R Workbook - Guidance for Worksheets." Aston University.

Wickham, Hadley, Mine Cetinkaya-Rundel, and Garrett Grolemund. n.d. *R for Data Science*. 2nd edition. O'Reilly. https://r4ds.hadley.nz/preface-2e.

Wilson, J. H., and B. Keating. 2007. *Business Forecasting*. 5th edition. McGraw-Hill.

Part IV

Seminar 4 (11 February 2025)

7 Introduction to Time Series Analysis

7.1 Example: GAP Sales data

We start by loading 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 becon
```

```
library(ggplot2)
library(dplyr)
library(lmtest)
```

```
Loading required package: zoo

Attaching package: 'zoo'

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

    as.Date, as.Date.numeric
```

```
library(tsseries)
```

```
Registered S3 method overwritten by 'quantmod':
  method            from
  as.zoo.data.frame zoo
```

The GAP_Sales data that we will be using in this session is obtained from (Wilson and Keating 2007). It shows the sales figures of GAP.

7.1.1 Task 1

Start a new project in R and name it as GAP-sales-analysis. Import GAP_Sales.csv data into this project. GAP_Sales is a quarterly time series data covering time period 1985:Q1 to 2004:Q4. In this example, we would like to estimate a regression model explaining sales of GAP.

7.1.1.1 Guidance

Use the menu import GAP_Sales.csv file into R. You need to choose `From Text` (base) because `csv` is a text format. The GAP_Sales data we have is comma separated, but you may encounter a different form of separation, for example, tab or semi-column. In the opening window, give a name for your data frame under the `Name` field and remember to check the `Heading` as `Yes` because we have variable names in the first row of the csv file. Also, note the `strings as factors` option, which asks R to import text-based content (variables) as categorial (`factor` is the terminology R uses).

You could alternatively run the code below

```
df <- read.csv("~/Desktop/R-workshops/assets/data/GAP_Sales.csv", stringsAsFactors=TRUE)
View(df)
```

For ease of typing, I called this data as `df`. In the code below, `df` will refer to the GAP_Sales data we imported.

7.1.2 Task 2

Browse the data and see the contents of the variables.

7.1.2.1 Guidance

We have done this above, using

```
View(df)
```

You may also use `head()` function to see the first 6 rows of data

```
head(df)
```

```
  Year quarter   Yqrt  Sales Time T.squared Q2 Q3 Q4 D911      ICS
1 1985      q1 1985q1 105715    1         1  0  0  0    0 94.46667
2 1985      q2 1985q2 120136    2         4  1  0  0    0 94.30000
3 1985      q3 1985q3 181669    3         9  0  1  0    0 92.83333
4 1985      q4 1985q4 239813    4        16  0  0  1    0 91.06667
5 1986      q1 1986q1 159980    5        25  0  0  0    0 95.53333
6 1986      q2 1986q2 164760    6        36  1  0  0    0 96.76667
```

7.1.3 Task 3

Provide a time series plot of the `Sales` variable.

7.1.3.1 Guidance

GAP_Sales data is a quarterly data. However, R would not recognise this until we tell it that is a quarterly time series. R has a built-in time series class, `ts` for basic data manipulation. Some other popular packages (more advanced than the `ts` in base R) include `tseries` and `zoo`.

As (Kleiber and Zeileis 2008) explains, `ts` is aimed at regular series observed in annual, quarterly, and monthly intervals. Time series objects can be created by supplying the data along with the arguments `start`, `end`, and `frequency`. The data can be:

- a numeric vector (a single variable), or
- a matrix (including a set of variables).

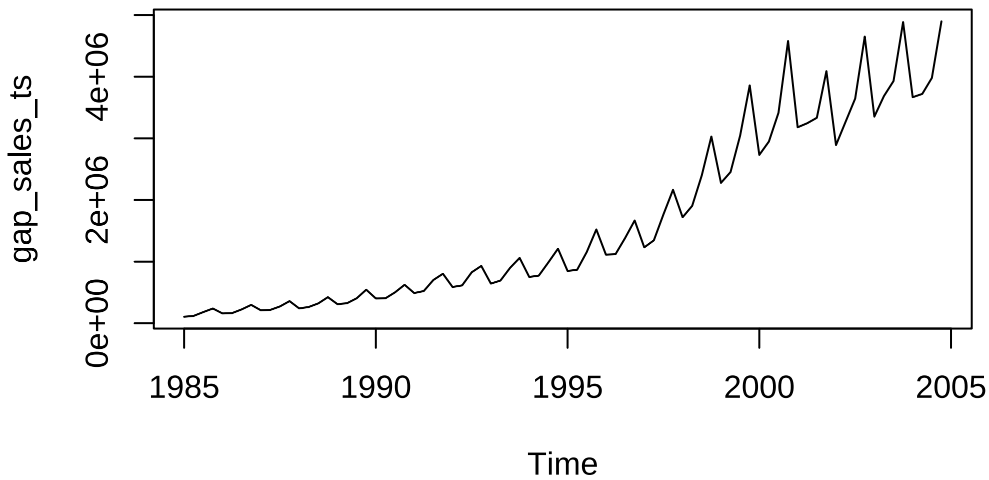
It includes time-series specific methods such as `lag()` (for the lagged values of the variables) and `diff()` (for time differencing the variable).

`Sales` is the variable we are interested in our data. So, let us start by introducing a time dimension to that series. In the code below, we create a single numeric vector, `gap_sales_ts` by defining the start date and the frequency of the `Sales` variable. Our variable starts from the first quarter of 1985 with a frequency of 4 (it is a quarterly data, repeating every 3 months).

```
gap_sales_ts <- ts(df$Sales, start = c(1985, 1), frequency =4)
```

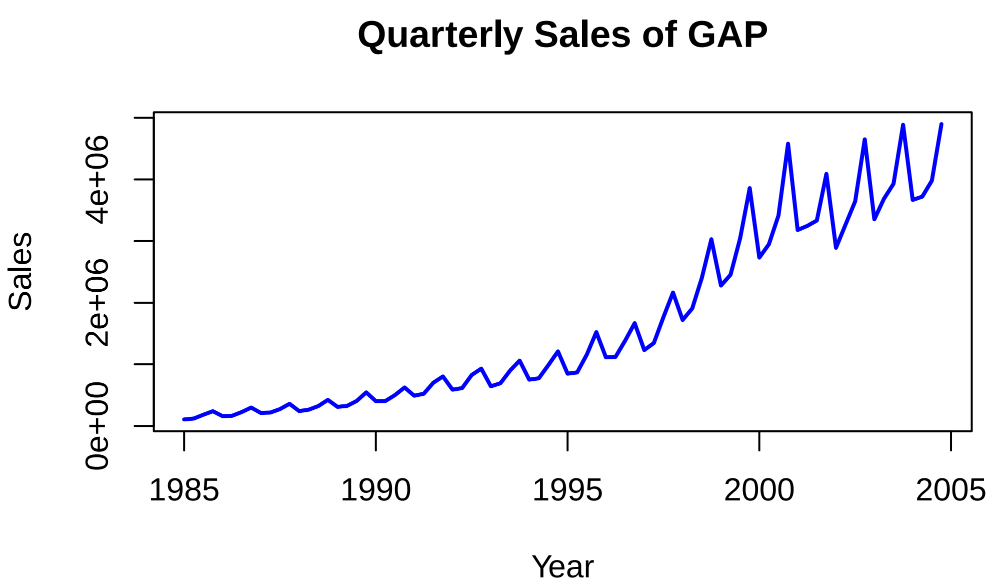
R's basic `plot` function will give us the following:

```
plot(gap_sales_ts)
```



You may add labels and color with some additional options:

```
plot(gap_sales_ts, col = "blue", lwd = 2, xlab = "Year", ylab = "Sales", main = "Quarterly Sa
```

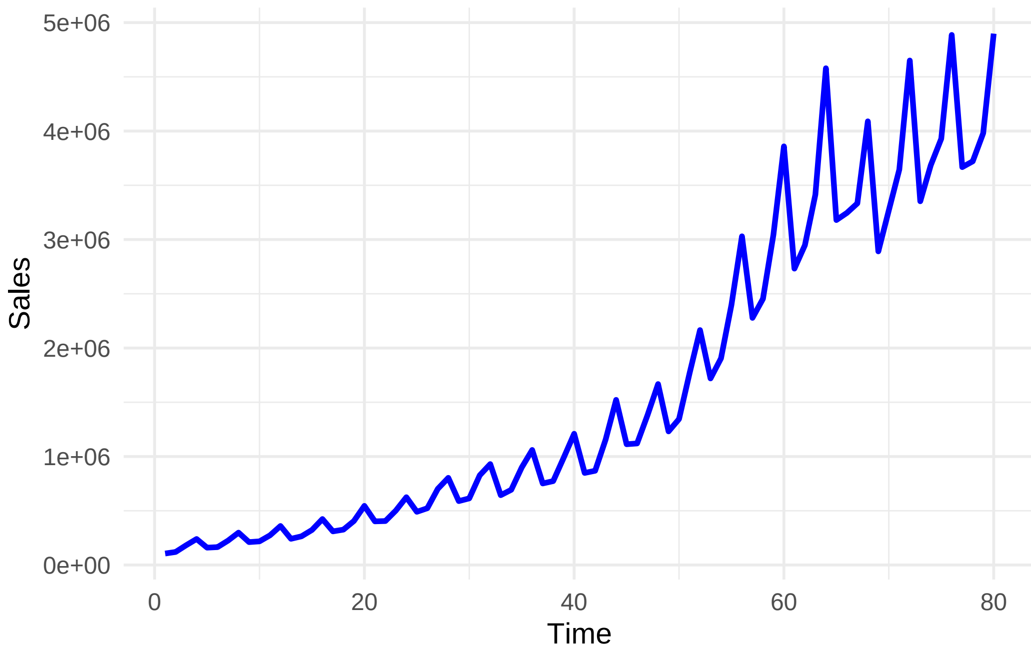


Looking at this plot, what can you say about the sales figures over time? What kind of time-series characteristics it reveals?

You may also use ggplot to plot the Sales data:

```
ggplot(df, aes(x = Time, y = Sales)) +
  geom_line(color = "blue", size = 1) +
  theme_minimal()
```

```
Warning: Using `size` aesthetic for lines was deprecated in ggplot2 3.4.0.
i Please use `linewidth` instead.
```



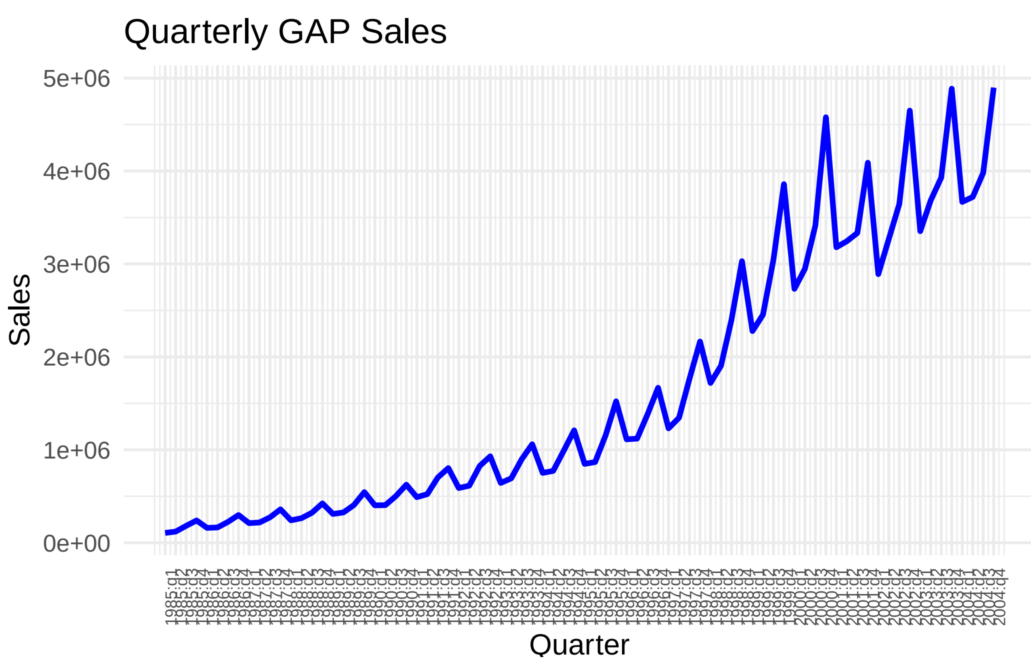
In the above plot, although we can see the pattern of the Sales variable quite clearly, the Time variable labels fail to show us the respective quarter values. We may change these labels by using the following lines of code.

We first define labels to correspond to each data point

```
# First, create a new column for formatted quarter labels
df$Quarter_label <- paste0(df$Year, ":", df$quarter)
# You can achieve the same as above using the code below:
# (note that you do not need this once you create Quarter_Label above )
df$Quarter_label_v2 <- with(df, paste(Year, quarter, sep = ":"))
```

Check the values of `Quarter_label` in the `df`. You will see that it goes on like 1985:q1, 1985:q2, and so on. We may now use these labels instead of the values of the `Time` variable.

```
ggplot(df, aes(x = Time, y = Sales)) +
  geom_line(color = "blue", size = 1) +
  scale_x_continuous(
    breaks = df$Time,  # Position the breaks at each quarter, i.e. at each value of Time
    labels = df$Quarter_label  # Label each point using Quarter_label variable created above
  ) + # provide a title and axes labels below
  labs(title = "Quarterly GAP Sales", x = "Quarter", y = "Sales") +
  theme_minimal() +
  theme(axis.text.x = element_text(angle = 90, size=6))  # Rotate labels for better readabili
```



7.1.4 Task 4

Fit a linear trend line to the `Sales` variable.

a. Provide an interpretation of the slope coefficient.

b. Check how well this model fits the data by plotting the predictions of the model and the observed values against time.

c. Plot the residuals of this model and explain whether or not you see a pattern.

7.1.4.1 Guidance

The `Time` variable will be used to fit a linear trend to `Sales`. The `Time` variables takes values from 1 to 80, increasing by 1 in each data point (quarter).

```
# Fit a linear trend line to Sales data
model_1 <- lm(Sales ~ Time, data = df)
summary(model_1)
```

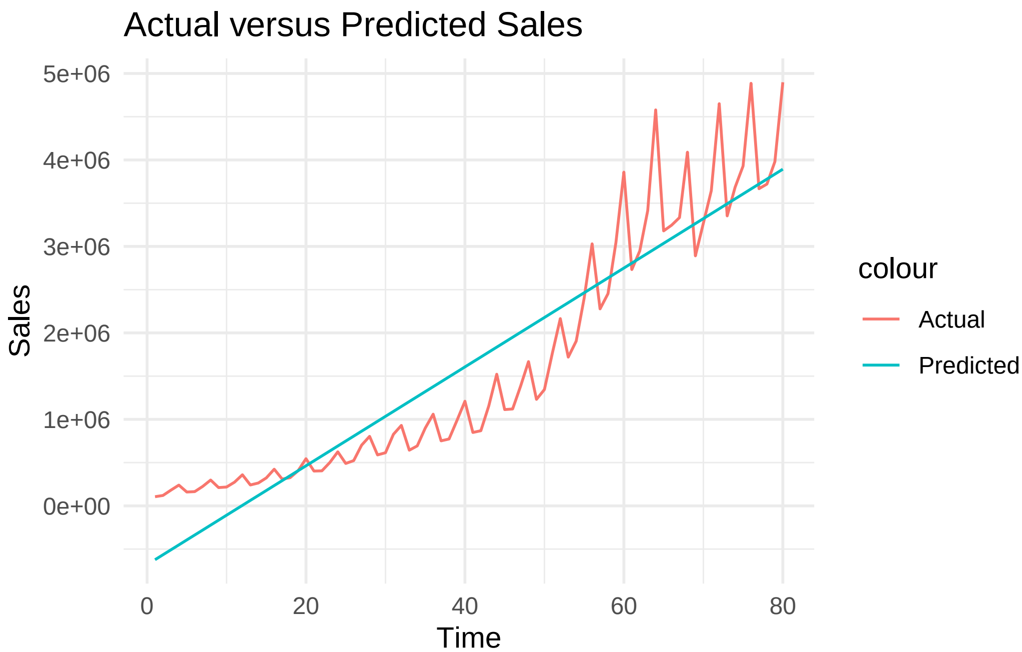
```

Call:
lm(formula = Sales ~ Time, data = df)

Residuals:
    Min      1Q  Median      3Q     Max 
-889709 -390551  -60886  325202 1600763 

Coefficients:
            Estimate Std. Error t value Pr(>|t|)    
(Intercept)  -680044     121435   -5.60 3.08e-07 ***
Time           57162       2605   21.95  < 2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 538000 on 78 degrees of freedom
Multiple R-squared:  0.8606,	Adjusted R-squared:  0.8588 
F-statistic: 481.6 on 1 and 78 DF,  p-value: < 2.2e-16
```

Obtain predictions using `predict()` function.

```
# Obtain predictions
df$sales_hat_m1 <- predict(model_1)
```

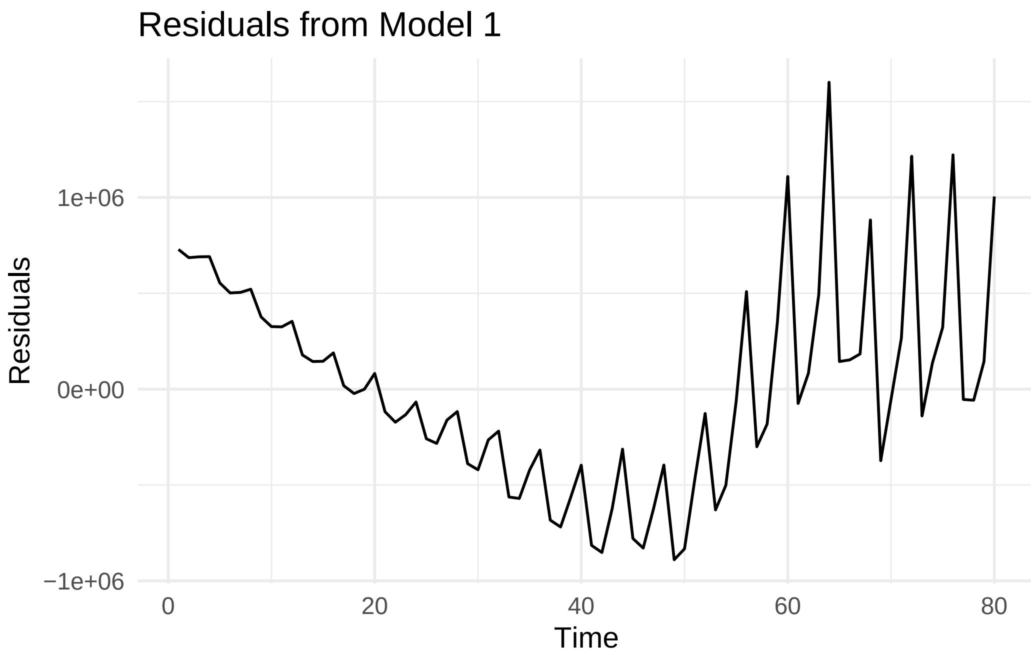
We can use R's base time series plot but we will need to convert the predictions into a time series. Alternatively, ggplot is easier to use.

```
# Plot actual versus predicted Sales
ggplot(df, aes(x = Time)) +
  geom_line(aes(y = Sales, color = "Actual")) +
  geom_line(aes(y= sales_hat_m1, color = "Predicted")) +
  theme_minimal() +
  labs(title = "Actual versus Predicted Sales", x = "Time", y = "Sales")
```



Below, we save and plot the residuals from model_1

```
# Save residuals from model_1
df$residuals_m1 <- residuals(model_1)

# Residual plot
ggplot(df, aes(x = Time, y = residuals_m1)) +
  geom_line() +
  theme_minimal() +
  labs(title = "Residuals from Model 1", x = "Time", y = "Residuals")
```



7.1.5 Task 5

Replicate the same analysis using logarithm of `Sales`

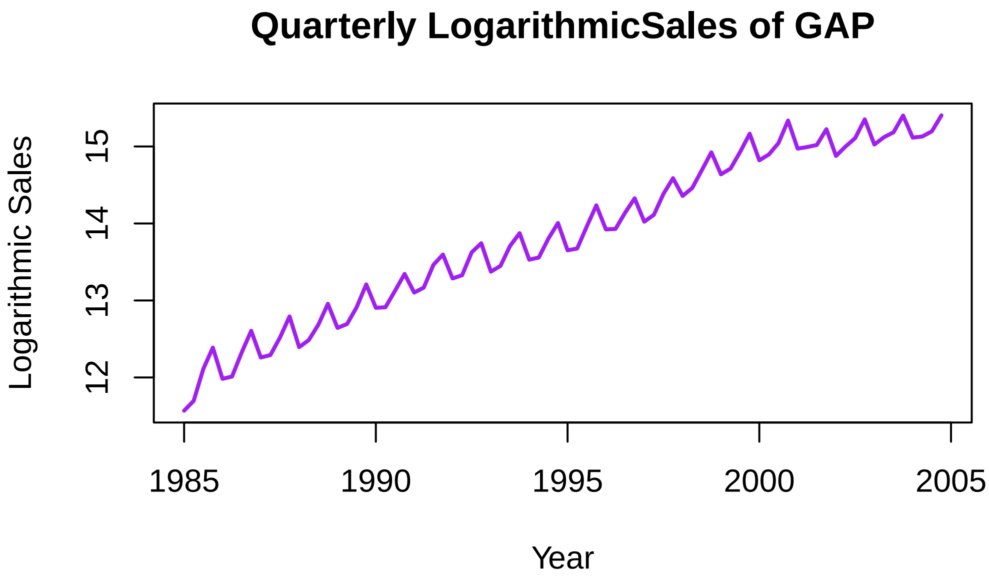
7.1.5.1 Guidance

We start by taking the logarithm of `Sales` variable.

```
# Logarithmic Sales data
df$ln_sales <- log(df$Sales)
```

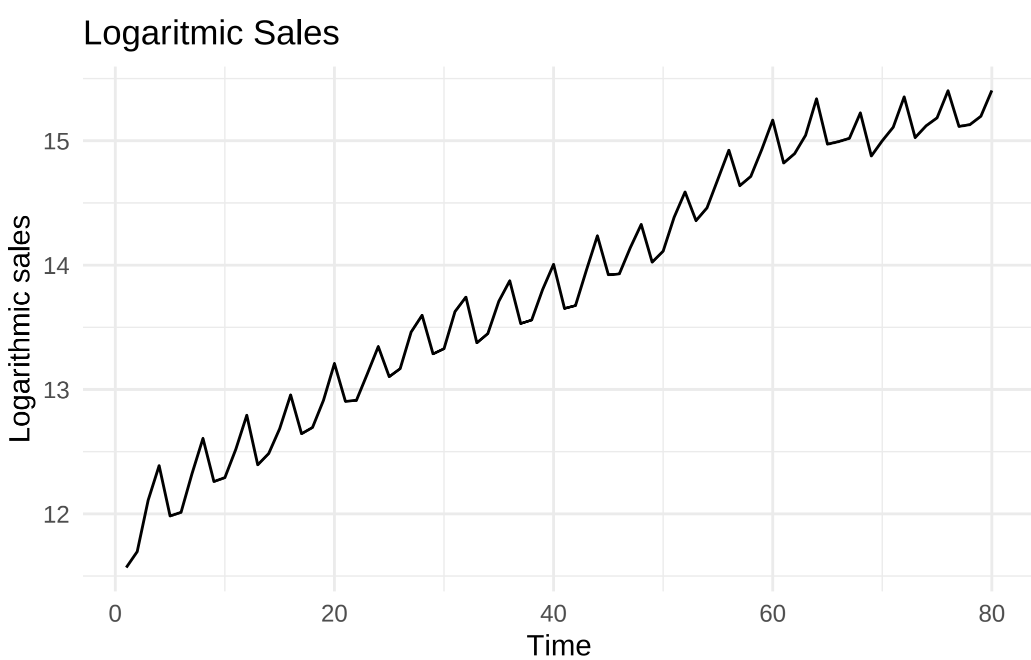
Plot logarithmic sales. Let's first do this base R's time series plot. We start by converting our `ln_sales` into quarterly time series, and then use the `plot()` function. `lwd` option below sets the line width of the plot. Change the color and the `lwd` values and see what you get.

```
# Plot of logarithmic sales (first approach - convert to time series)
ln_sales_ts <- ts(df$ln_sales, start = c(1985, 1), frequency =4) # covenrt the ln_sales into
plot(ln_sales_ts, col = "purple", lwd = 2, xlab = "Year", ylab = "Logarithmic Sales", main =
```



We may also use `ggplot` for the same purpose

```
# Plot of logarithmic sales (second approach - use ggplot)
ggplot(df, aes(x = Time, y = ln_sales)) +
  geom_line() +
  theme_minimal() +
  labs(title = "Logaritmic Sales", x = "Time", y = "Logarithmic sales")
```



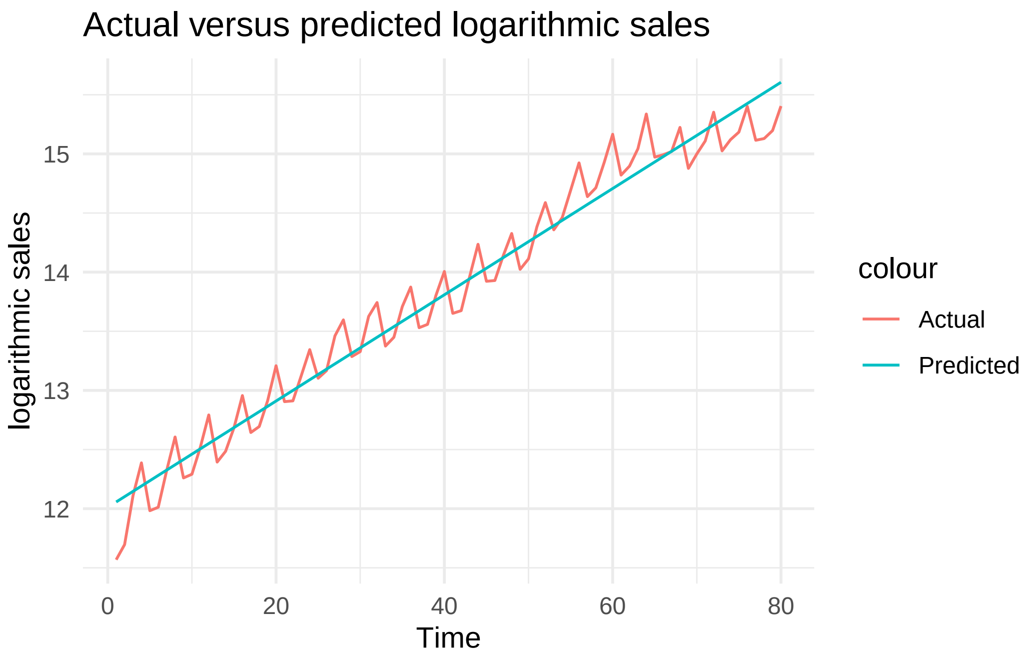
Fit a trend line to logarithmic sales

```r
# Fit a trend line to logarithmic sales
model_2 <- lm(ln_sales ~ Time, data = df)
summary(model_2)
```

```
Call:
lm(formula = ln_sales ~ Time, data = df)

Residuals:
    Min      1Q  Median      3Q     Max 
-0.4883 -0.1559 -0.0026  0.1684  0.4589 

Coefficients:
             Estimate Std. Error t value Pr(>|t|)    
(Intercept) 12.011834   0.046964  255.76   <2e-16 ***
Time         0.044919   0.001007   44.59   <2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 0.2081 on 78 degrees of freedom
Multiple R-squared:  0.9623,    Adjusted R-squared:  0.9618 
F-statistic:  1988 on 1 and 78 DF,  p-value: < 2.2e-16
```

Let's now plot the predictions from this model with the actual `ln_sales` figures

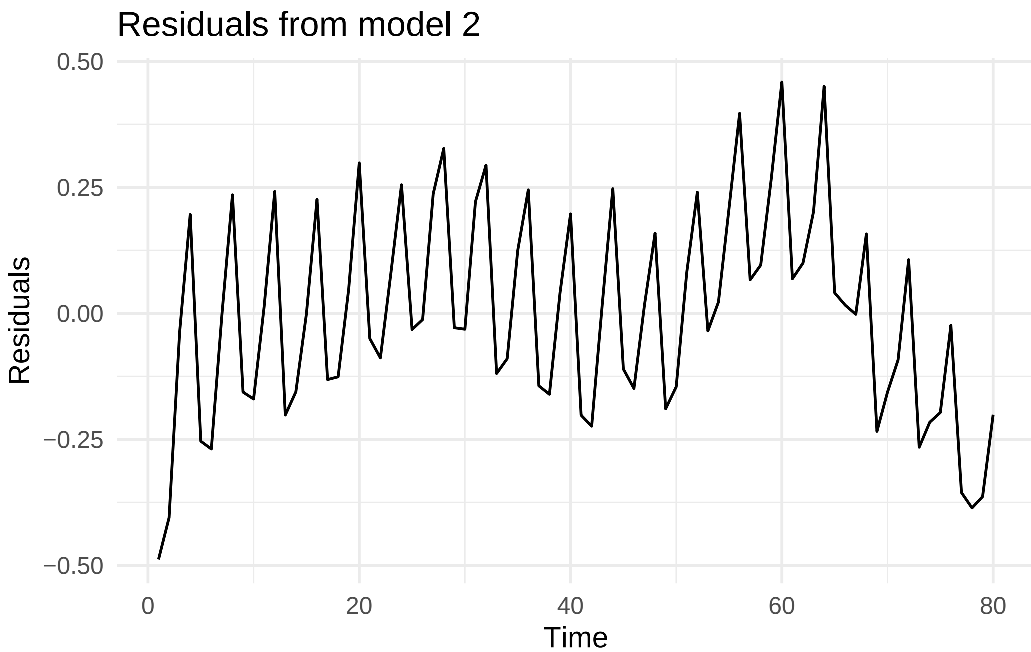
```r
# Obtain predictions from the logarithmic model
df$ln_sales_hat_m2 <- predict(model_2)

# Plot actual versus predicted log sales
ggplot(df, aes(x = Time)) +
  geom_line(aes(y = ln_sales, color = "Actual")) +
  geom_line(aes(y = ln_sales_hat_m2, color = "Predicted")) +
  theme_minimal() +
  labs(title = "Actual versus predicted logarithmic sales", x = "Time", y = "logarithmic sale
```



What do you think about this fit?

Let's check what the residuals from the above estimation look like

```
# Residuals from model_2
df$residuals_m2 <- residuals(model_2)

# Plot residuals from model_2
ggplot(df, aes(x = Time, y= residuals_m2))+
  geom_line() +
  theme_minimal() +
  labs(title = "Residuals from model 2", x = "Time", y = "Residuals")
```



7.1.6 Task 6

Do you have any suggestions to improve the fit of this model?

7.1.6.1 Guidance

Check the residual plot above. Do you see a specific pattern? What can we do to capture the fluctuations that you see?

7.1.7 Task 7

Add quarter dummies to the model you estimated above.

a. Interpret the coefficients in this model.

b. Check how well this model fits the data by plotting the predictions of the model and the observed values against time.

c. Plot the residuals of this model and explain whether or not you see a pattern.

d. Does the inclusion of the quarter dummies improve the fit of the model? Test for the joint significance of the quarter dummies.

e. If you were to choose one the models that you have estimated using the GAP sales data, which one would you choose? Why?

7.1.7.1 Guidance

The quarter dummies that we need for this model are already in the data: `Q2`, `Q3`, `Q4`. If these were not in the data, we could create them using the `dplyr` package. This is provided below.

```
df <- df %>%
  mutate(
    quarter1 = ifelse(quarter == "q1", 1, 0),
    quarter2 = ifelse(quarter == "q2", 1, 0),
    quarter3 = ifelse(quarter == "q3", 1, 0),
    quarter4 = ifelse(quarter == "q4", 1, 0)
  )
```

Check the values of these newly created dummies (`quarter1`, `quarter2`, `quarter3`, and `quarter4`) in the data.

We can now estimate the model including these quarter dummies together with a linear trend

```
# Estimate the model using trend and quarter dummies
model_3 <- lm(ln_sales ~ Time + quarter2 + quarter3 + quarter4, data = df)
summary(model_3)
```

```

Call:
lm(formula = ln_sales ~ Time + quarter2 + quarter3 + quarter4,
    data = df)

Residuals:
     Min       1Q   Median       3Q      Max
-0.41512 -0.06014 -0.00347  0.09422  0.23885

Coefficients:
             Estimate Std. Error t value Pr(>|t|)
(Intercept) 11.882418   0.042709 278.219  < 2e-16 ***
Time         0.044620   0.000707  63.110  < 2e-16 ***
quarter2     0.013792   0.046130   0.299 0.765783
quarter3     0.184808   0.046146   4.005 0.000145 ***
quarter4     0.367414   0.046173   7.957 1.44e-11 ***
---
```

```
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 0.1459 on 75 degrees of freedom
Multiple R-squared:  0.9822,    Adjusted R-squared:  0.9812
F-statistic:  1032 on 4 and 75 DF,  p-value: < 2.2e-16
```

Are these quarterly dummies contributing to the explanatory power of the model? In other words, are they jointly statistically significant? We can check this using an F-test for restrictions. This could be done using the `anova` function in R.

Below are the steps we follow to test for the restrictions:

1. Estimate the full (**unrestricted**) model. We have done that above. It is saved under `model_3`.
2. Estimate the **restricted** model where quarter dummy coefficients take value zero. This is in fact, our `model_2` above.
3. Perform an F-test to compare the restricted and unrestricted models using `anova()` :

   `anova(restricted_model, unrestricted_model)`

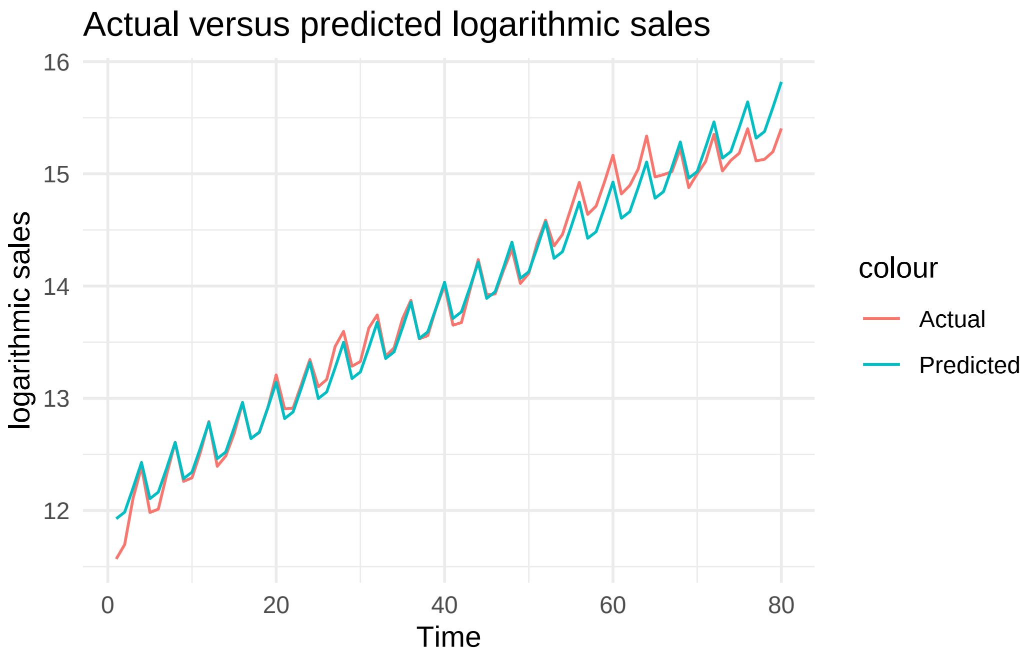
```
# Perform an F-test
anova(model_2, model_3)
```

```
Analysis of Variance Table

Model 1: ln_sales ~ Time
Model 2: ln_sales ~ Time + quarter2 + quarter3 + quarter4
  Res.Df    RSS Df Sum of Sq      F   Pr(>F)
1     78 3.3767
2     75 1.5956  3    1.7811 27.906 3.17e-12 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

The null hypothesis in the above test is that the coefficients of quarter dummies are jointly equal to zero versus the alternative that at least one is different than zero. We have a very small p-value. Hence we reject the null hypothesis and conclude that the quarter dummies are jointly statistically significant.

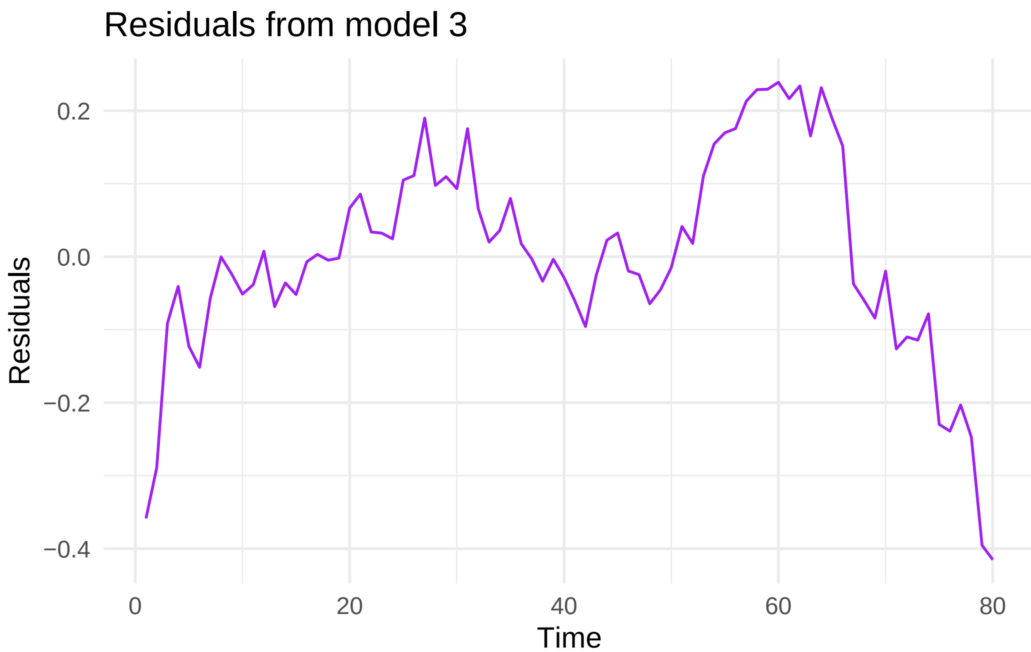
Let's plot the actual values against predictions to see the improvement by the inclusion of the quarter

```r
# Obtain predictions from model_3
df$ln_sales_hat_m3 <- predict(model_3)

# Plot actual versus predicted log sales
ggplot(df, aes(x = Time)) +
  geom_line(aes(y = ln_sales, color = "Actual")) +
  geom_line(aes(y = ln_sales_hat_m3, color = "Predicted")) +
  theme_minimal() +
  labs(title = "Actual versus predicted logarithmic sales", x = "Time", y = "logarithmic sale
```



And finally, let's have a look at the residuals

```r
# Residuals from model_2
df$residuals_m3 <- residuals(model_3)

# Plot residuals from model_2
ggplot(df, aes(x = Time, y= residuals_m3))+
  geom_line(color = "purple") +
  theme_minimal() +
  labs(title = "Residuals from model 3", x = "Time", y = "Residuals")
```



7.1.8 Task 8

Conduct the conventional misspecification tests on the last model estimated.

7.1.8.1 Guidance

We may start with the **normality of the residuals**. For this test, we will be using the `jarque.bera.test()` from the `tseries` package.

```
# Normality of residuals
jarque.bera.test(df$residuals_m3)
```

```

	Jarque Bera Test

data:  df$residuals_m3
X-squared = 6.4227, df = 2, p-value = 0.0403
```

The null hypothesis of normal distribution is rejected at 5% significance level.

For the tests that follow, we will using the `lmtest` package.

Autocorrelation Test

We use the `bgtest()` function below. It performs the Breusch-Godfrey Test. We first test for the first order autocorrelation and then, because we have quarterly data, the existence of autocorrelation up to order 4.

```
# Autocorrelation
bgtest(model_3)
```

```

	Breusch-Godfrey test for serial correlation of order up to 1

data:  model_3
LM test = 60.429, df = 1, p-value = 7.628e-15
```

```
bgtest(model_3, order = 4)
```

```

	Breusch-Godfrey test for serial correlation of order up to 4

data:  model_3
LM test = 62.487, df = 4, p-value = 8.703e-13
```

There is autocorrelation problem in our model.

Heteroscedasticity

We will use `bptest()` function for heteroscedasticity. It performs the Breusch-Pagan Test.

```
# Heteroscedasticity
bptest(model_3)
```

```

	studentized Breusch-Pagan test

data:  model_3
BP = 9.4108, df = 4, p-value = 0.05161
```

The null of no heteroscedasticity cannot be rejected at 5% significance level.

Functional Form

We will use `resettest()` for Ramsey's RESET.

```
# Ramsey RESET
resettest(model_3)
```

```

	RESET test

data:  model_3
RESET = 36.442, df1 = 2, df2 = 73, p-value = 1.06e-11
```

The null of correct functional form is rejected.

7.1.9 Task 9

Using the last model, forecast the sales value for each quarter of 2005.

7.1.9.1 Guidance

There are more advanced ways of producing forecasts in R. But we need at this stage is explained below.

Define a `forecast_2005` function using the coefficients of `model_3`. Let us see what `model_3` coefficients are

```
summary(model_3)
```

```

Call:
lm(formula = ln_sales ~ Time + quarter2 + quarter3 + quarter4,
    data = df)

Residuals:
     Min       1Q   Median       3Q      Max
-0.41512 -0.06014 -0.00347  0.09422  0.23885

Coefficients:
             Estimate Std. Error t value Pr(>|t|)
(Intercept) 11.882418   0.042709 278.219  < 2e-16 ***
Time         0.044620   0.000707  63.110  < 2e-16 ***
quarter2     0.013792   0.046130   0.299 0.765783
quarter3     0.184808   0.046146   4.005 0.000145 ***
```

```
quarter4     0.367414   0.046173   7.957 1.44e-11 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 0.1459 on 75 degrees of freedom
Multiple R-squared:  0.9822,    Adjusted R-squared:  0.9812
F-statistic:  1032 on 4 and 75 DF,  p-value: < 2.2e-16
```

Define a function to forecast future values:

```r
forecast_2005 <- function(Time, quarter2, quarter3, quarter4) {
  exp(11.882418 + 0.044620 * Time + 0.013792 * quarter2 + 0.184808 * quarter3 + 0.367414 * qu
}
```

Use the above function for forecasts.

```r
# Forecasts from 2005
# quarter 1
y2005_q1 <- forecast_2005(81,0,0,0)
print(y2005_q1)
```

```
[1] 5371609
```

```r
# quarter 2
y2005_q2 <- forecast_2005(81,1,0,0)
print(y2005_q2)
```

```
[1] 5446207
```

```r
# quarter 3
y2005_q3 <- forecast_2005(81,0,1,0)
print(y2005_q3)
```

```
[1] 6461978
```

```r
# quarter 4
y2005_q4 <- forecast_2005(81,0,0,1)
print(y2005_q4)
```

```
[1] 7756579
```

Part V

Seminar 5 (18 February 2025)

8 Unit Root and Cointegration

8.1 Unit Root (Non-stationary Time Series)

8.1.1 Example: Pepper Price

The Pepper Price example provided in this section is taken from (Kleiber and Zeileis 2008). We start by loading the required libraries.

```
library(AER) # Applied Econometrics with R, Kleiber and Zeileis, 2008
```

```
Loading required package: car

Loading required package: carData

Loading required package: lmtest

Loading required package: zoo


Attaching package: 'zoo'

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

    as.Date, as.Date.numeric

Loading required package: sandwich

Loading required package: survival
```

```
library(tseries) #Required for the adf test
```

```
Registered S3 method overwritten by 'quantmod':
  method            from
  as.zoo.data.frame zoo
```

Load Pepper Price time series data and check the first 6 rows of observations.

```r
data("PepperPrice")
head(PepperPrice)
```

```
        black   white
[1,]  884.050 1419.78
[2,]  919.329 1503.55
[3,]  930.350 1536.62
[4,] 1102.310 1629.22
[5,] 1150.810 1737.24
[6,] 1093.490 1629.22
```

There are two series here: black pepper and white pepper. Let's understand the time series components better:

```r
# tsp stands for "Time Series Properties"
tsp(PepperPrice)
```

```
[1] 1973.75 1996.25   12.00
```

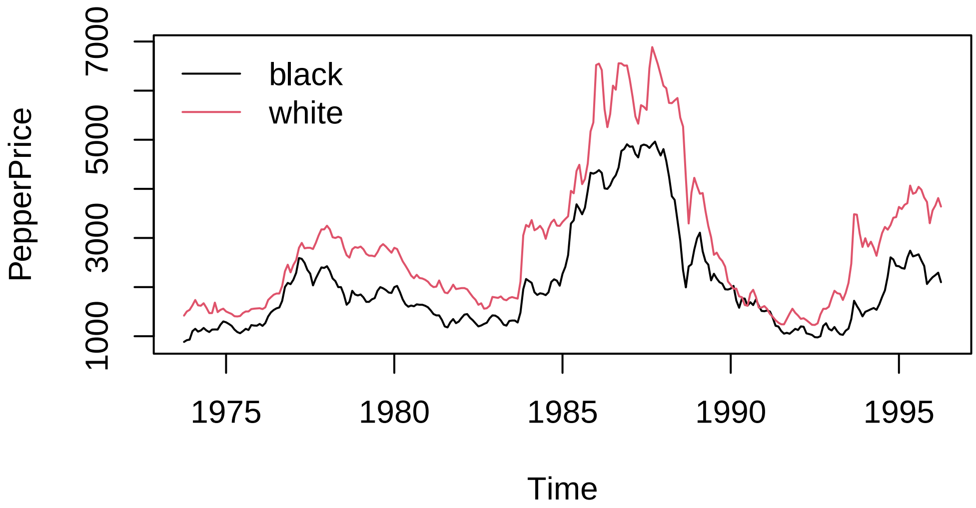
`tsp` above stands for time series properties. It seems like we have monthly data (frequency of 12), starting in year 1973 and ending in year 1996.

```r
window(PepperPrice, end = c(1974, 6))
```

```
            black   white
Oct 1973  884.050 1419.78
Nov 1973  919.329 1503.55
Dec 1973  930.350 1536.62
Jan 1974 1102.310 1629.22
Feb 1974 1150.810 1737.24
Mar 1974 1093.490 1629.22
Apr 1974 1117.740 1620.40
May 1974 1168.450 1671.11
Jun 1974 1117.740 1578.51
```

Let us start by plotting the data:

```
plot(PepperPrice, plot.type = "single", col = 1:2)
legend("topleft", c("black", "white"), bty = "n", col = 1:2, lty = rep(1,2))
```



- `plot(PepperPrice, ...)` is the base R plot function for time series objects.
- `plot.type = "single"` ensures that multiple time series within `PepperPrice` are plotted on the same graph (rather than separate subplots).
- `col = 1:2` assigns different colors to the time series (we use R defaults above)

The second line after `plot` is about legend.

- `legend("topleft", ...)` places the legend in the top-left corner of the plot.
- `c("black", "white")` are the legend labels for the two time series.
- `bty = "n"` removes the legend box (makes it look cleaner).
- `col = 1:2` matches the line colors (black and red).
- `lty = rep(1,2)` sets line type to solid (`lty = 1`) for both series.

to be continued