Introduction to R for Clinical Psychology

Christopher J Wilson

2024-12-12

Table of contents

Welcome

The purpose of this site/book is to introduce you to R and R Studio for use in Clinical Psychology Research. Before you begin, there are some important things to know:

1. This site is not a comprehensive guide to everything that can be done in R. There are many resources available for learning R, and this site is just a starting point. It will cover the basics of R and R Studio, and will provide examples of how to use R for common tasks in clinical psychology research.
2. This site is not a comprehensive guide to statistics. It will cover some key statistical concepts and how to perform them in R, but it is not a compete substitute for a statistics textbook or course. The goal of this content is to cover some key statistical concepts that are pertinent to clinical psychology research. It is expected that you have some background in statistics, and that you are familiar with basic concepts such as hypothesis testing, p-values, and confidence intervals.
3. It is important to note that R is a powerful tool, but it can be complex and challenging to learn. It is normal to feel overwhelmed at times, and it is important to be patient with yourself as you learn. The more you practice and use R, the more comfortable you will become with it. Set your expectations accordingly, and remember that learning R is a process that takes a long time and should continue throughout your career. This is just your starting point.
4. This site is a work in progress. I am constantly updating, adding new content and removing some less useful elements, so please check back regularly for updates. If you have any feedback or suggestions for content, please let me know.

The tidyverse

> **!** This site/book uses the tidyverse set of packages
>
> The tidyverse is a collection of R packages designed for to make data manipulation and visualization. easier in R. The tidyverse is a powerful set of tools for data analysis, and it is widely used in the R community. It is assumed that you will have the tidyverse installed and loaded for the examples in this site/book. If you do not have the tidyverse

installed, you can install it by running the following code in the R console:

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

You only need to install the package on to your machine once. Once you have installed the tidyverse, you can load it by running the following code in the R console:

```
library(tidyverse)
```

The `library()` function is used to load packages in R. You will need to load the tidyverse package at the beginning of each R session in which you want to use it.

Textbooks that can be accessed online

e-books can be accessed from the library website: https://www.tees.ac.uk/depts/lis/

Research Methods and Statistics

Coolican, 2019. Research Methods and Statistics in Psychology. Taylor & Francis Group

Barker, C., Pistrang, N., & Elliott, R. (2015). Research methods in clinical psychology: An introduction for students and practitioners (3rd ed.). Chichester, West Sussex: Wiley Blackwell.

Weiner, I. B., Schinka, J. A., & Velicer, W. F. (2012). Handbook of psychology, research methods in psychology (2. Aufl. ed.). Somerset: Wiley.

Working with R and RStudio to do analysis

Navarro, D. (2017) Learning statistics with R.

Phillips N. D. (2018) YaRrr! The Pirate's Guide to R

Horton, Pruim and Kaplan (2015) A Student's Guide to R

Mather, M. (2019) R for Academics

Wickham and Grolemund (2019). R For Data Science

Allaire and Grolemund (2019). R Markdown: The Definitive Guide

Basics of RStudio

Data Import

Data Transformation

Data Visualisation with GGPlot

1 Introduction to R and R Studio

1.1 What are R and R Studio?

At the end of this section, you will be able to:

- Download and install R and R Studio
- Understand the basic layout of R Studio
- Describe some of the differences between SPSS and R

1.1.1 Downloading and installing R and R Studio

To get started with R Studio, you need to download and install two pieces of software:

1. **R**: The base software that you will use to write and run code.
2. **R Studio**: An integrated development environment (IDE) that makes it easier to write and run code in R.

Click on these links to download:

- R project
- RStudio

1.1.2 The R Studio layout

When you open R Studio, you will see a screen that looks like this:

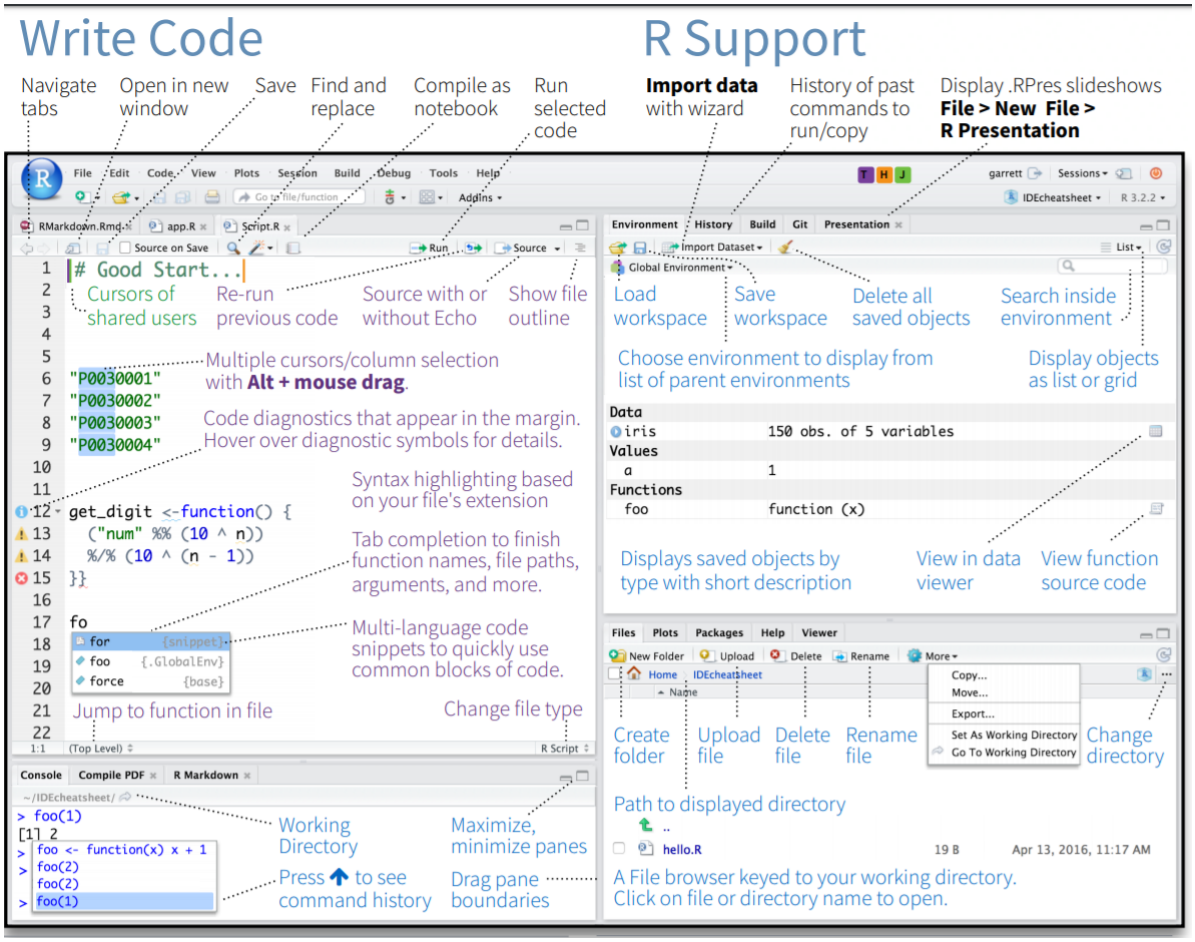


Figure 1.1: R Studio IDE

Briefly, the different panes in R Studio are:

- **Console**: You can write and run code in this pane. However, it is best practice to write code in a script. You will see output from your code in the console.
- **Environment/History**: This pane shows you the objects that you have created in R, and the history of the commands that you have run.
- **Files/Plots/Packages/Help**: These panes allow you to navigate your files, view plots, manage packages, and access help documentation.

You will learn more about these panes as you work through the course.

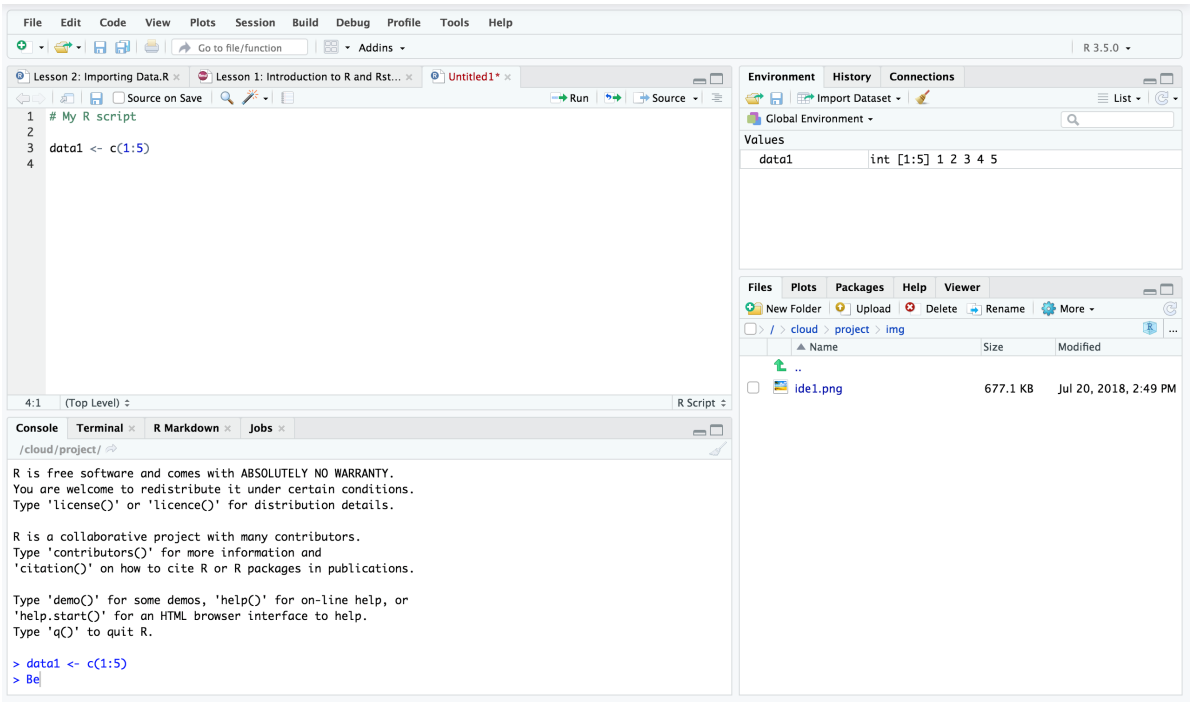


Figure 1.2: R Studio IDE

1.1.3 Differences between SPSS and R

R is a statistical programming language, while SPSS is a point-and-click software package. This means that in R, you write code to perform tasks, while in SPSS, you click buttons and select options from menus.

This can take some getting used to, but there are many advantages to using R:

- **Reproducibility**: You can save your code and rerun it at any time, ensuring that your analysis is reproducible.
- **Flexibility**: You can write code to perform any task you like, rather than being limited to the options available in a menu.
- **Community**: R has a large and active community of users who share code and help each other to solve problems.

With R, you won't manipulate your source data files. Instead, you load the data into R and manipulate it in R. This means that you can always go back to your original data and start again if you need to.

1.2 No more "point and click"! - the R workflow.

At the end of this section, you will be able to:

- Open a new script in R Studio
- Write and run code in a script
- Save a script for later use

1.2.1 Using scripts in R Studio

When you work in R, you will write code in a script. This is a text file that contains the code that you want to run. You can write and run code in the console, but it is best practice to write code in a script. This allows you to save your code and run it again later IT also makes it easier to see what you have done.

To open a new script in R Studio, click on `File > New File > R Script`. This will open a new script in the top-left pane of R Studio.

You can write code in the script, and then run it by selecting the code that you want to run and clicking the `Run` button at the top of the script pane. You can also run code by pressing `Ctrl + Enter` on your keyboard.

To save your script, click on `File > Save As...` and save the file with a `.R` extension.

Organising your work

It is good practice to keep your work organised by putting your scripts, data, and other files in a folder on your computer, for each project that you work on.
RStudio also allows the creation of projects. You can create a new project in R Studio by clicking on `File > New Project...`. This will create a new folder on your computer where you can save your scripts, data, and other files. If you save your script in the project folder, you can easily access it by opening the project in R Studio. If you use projects, be aware that R Studio will load the last project you worked on when you open the software.

1.3 Objects, functions and packages in R

At the end of this section, you will be able to:

- Create objects in R
- Use functions in R to perform tasks
- Install and load packages in R

1.3.1 What are objects?

In R, you can create objects to store data. For example, you can create an object called `numbers` that contains a set of numbers like this:

```
numbers <- c(1, 2, 3, 4, 5)
```

Breaking this code down:

- `numbers` is the name of the object that you are creating.
- `<-` is the assignment operator. It assigns the value on the right-hand side of the operator to the object on the left-hand side.
- `c(1, 2, 3, 4, 5)` is the data that you are assigning to the object. In this case, it is a set of numbers.

When you run this code, R will create an object called `numbers` that contains the numbers 1, 2, 3, 4, and 5. You will be ab‘ le to see the object in the Environment pane in R Studio.

You can then use the object in your code, instead of typing out the data each time (see Section 1.3.2 for example).

1.3.2 What are functions?

Functions are code that have been written to perform a specific task. You can use functions in R to perform tasks like reading data into R, summarising data, and creating plots.

For example, the `mean()` function calculates the mean of a set of numbers. You can use the `mean()` function like this:

```
numbers <- c(1, 2, 3, 4, 5)

mean(numbers)
```

Functions in R have a name, followed by parentheses. You can pass arguments to the function inside the parentheses. In this case, the `mean()` function takes a set of numbers as an argument, and returns the mean of those numbers.

To learn more about a function, you can use the `help()` function. For example, to learn more about the `mean()` function, you can run the following code:

```
help(mean)
```

You can also use the `?` operator to get help on a function. For example, to get help on the `mean()` function, you can run the following code:

```
?mean
```

1.3.3 What are packages?

R has many built-in functions that you can use to perform tasks. However, there are also many packages available that contain additional functions. You can install these packages onto your conputer and then load them into your R session whenever you want to use them.

To install a package, you can use the `install.packages()` function. For example, to install the `tidyverse` package, you would run the following code:

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

To load a package into your R session, you can use the `library()` function. For example, to load the `tidyverse` package, you would run the following code:

```
library(tidyverse)
```

Once you have loaded a package, you can use the functions in that package in your code. For example, the `tidyverse` package contains functions for data manipulation and visualisation.

2 Working with data in R Studio

Working with your data in RStudio is a little bit different from working with data in a spreadsheet, for example. One crucial difference is that you need to be explicit about what you want to do with your data. In a spreadsheet, you can simply click on a cell and start typing. In R, you need to tell the software what you want to do with your data. This can be a bit intimidating at first, but it is also one of the most powerful features of R. It allows you to automate repetitive tasks and perform complex analyses with just a few lines of code.

2.1 Importing data into R

At the end of this section, you will be able to:

- Load data into R from different file types
- Understand the structure of data in R

There are a few different ways to load data into R. You can load data from a file on your computer, from a URL, or from a package. You can load data in different file types, such as CSV, Excel, and SPSS files.

Using RStudio, you can load data by clicking on `File > Import Dataset`. This will open a window where you can select the file that you want to load.

However, you can also load data using code. For example, you can use the `read_csv()` function from the `readr` package to load a CSV file into R. You can use the `readxl` package to load an Excel file, and the `haven` package to load an SPSS file.

```
# Load the readr package

library(readr)

# Load a CSV file into R

data <- read_csv("data.csv")
```

Let's break this code down:

- `library(readr)` loads the `readr` package into your R session. This package contains the `read_csv()` function, which you can use to load a CSV file into R.
- `read_csv("data.csv")` reads the CSV file called `data.csv` into R. The data will be stored in an object called `data`.

When you load data into R, it will be stored as a data frame. A data frame is a type of object in R that is used to store tabular data. It is similar to a spreadsheet in Excel, with rows and columns.

2.2 How are data stored in R?

If you worked through the previous section, you should already have some idea how to load data into R. But how are data stored in R? In R, data are stored in objects. An object is a container that holds data. There are several types of objects in R, but the most common ones are:

- Vectors (e.g., a sequence of numbers)
- Matrices (e.g., a table of rows and columnns, all of the same data type)
- Data frames (e.g., a table of data where each column represents a variable and each row represents an observation)
- Lists (e.g., a collection of objects)

In this section, we will focus on data frames, which are the most common way to store data in R. A data frame is a table of data where each column represents a variable and each row represents an observation. You can think of a data frame as having a structure similar to a spreadsheet.

	participant	happiness	intervention
1	1	15	1
2	2	16	1
3	3	23	1
4	4	22	1
5	5	24	1
6	6	16	1
7	7	12	1

Figure 2.1: Data frame with 3 variables/columns

2.3 How do we use data frames in R?

To view the data in a data frame, you can simply type the name of the data frame in the console and press Enter. For example, if you have a data frame called `my_data`, you can view the data in the data frame by typing `my_data` in the console and pressing Enter.

```
## load the tidyverse package

library(tidyverse)

# Create a data frame
my_data <- data.frame(
  name = c("Alice", "Bob", "Charlie", "David", "Eve", "Frank"),
  age = c(25, 30, 35, 40, 45, 50),
  height = c(160, 175, 180, 165, 170, 190),
  car = c("Electric", "Petrol", "Electric", "Petrol", "Petrol", "Electric")
)

# View or refer to the data in the data frame

my_data
```

```
     name age height      car
1   Alice  25    160 Electric
2     Bob  30    175   Petrol
3 Charlie  35    180 Electric
4   David  40    165   Petrol
5     Eve  45    170   Petrol
6   Frank  50    190 Electric
```

In the code above, we created a data frame called `my_data` with four variables: `name`, `age`, `height`, and `car`. We then used the `my_data` object to view the data in the data frame.

2.4 View or refer to a specific variable in a data frame

To view or refer to a specific variable in a data frame, you can use the `$` operator. For example, if you want to view the `age` variable in the `my_data` data frame, you can type `my_data$age` in the console and press Enter.

```r
# View or refer to a specific variable in a data frame

my_data$age
```

```
[1] 25 30 35 40 45 50
```

2.5 Data types in R

In R, each variable in a data frame has a data type. The most common data types in R are:

- Numeric: for continuous variables (e.g., age, height)
- Factor: for categorical variables
- Logical: for binary variables (TRUE or FALSE)

You can use the `str()` function to view the structure of a data frame, including the data types of each variable.

```r
# View the structure of a data frame

str(my_data)
```

```
'data.frame':   6 obs. of  4 variables:
 $ name  : chr  "Alice" "Bob" "Charlie" "David" ...
 $ age   : num  25 30 35 40 45 50
 $ height: num  160 175 180 165 170 190
 $ car   : chr  "Electric" "Petrol" "Electric" "Petrol" ...
```

In the code above, we used the `str()` function to view the structure of the `my_data` data frame. The output shows the data types of each variable in the data frame.

2.6 Convert data types in R

You can convert the data type of a variable in R using the `as.` functions. For example, you can convert a character variable to a factor variable using the `as.factor()` function.

```r
# Convert a character variable to a factor variable

my_data$name <- as.factor(my_data$name)

my_data$car <- as.factor(my_data$car)
```

In the code above, we converted the `name` variable in the `my_data` data frame from a character variable to a factor variable using the `as.factor()` function.

2.7 Subsetting data in R

> At the end of this section, you will be able to:
>
> - Filter data in R
> - Create subsets of data in R

Subsetting data in R means selecting a subset of the data based on certain criteria. For example, you might want to select only the rows where a certain variable is greater than a certain value, or only the columns that contain certain variables.

If we use the `my_data` data frame from the previous section, we can subset the data to select only the rows where the `age` variable is greater than 30.

```
# Filter the data frame to select only the rows where the age variable is greater than 30

# this method uses the dplyr package, which is a part of the tidyverse. Be sure to load the t

my_data %>% filter(age > 30)
```

```
     name age height      car
1 Charlie  35    180 Electric
2   David  40    165   Petrol
3     Eve  45    170   Petrol
4   Frank  50    190 Electric
```

Let's break this code down:

- `my_data` is the data frame that we want to subset.
- `%>%` is the pipe operator, which is used to pass the data frame to the next function. This allows us to link multiple steps together in a single line of code.
- `filter(age > 30)` is the function that filters the data frame to select only the rows where the `age` variable is greater than 30.

The output of this code will be a new data frame that contains only the rows where the `age` variable is greater than 30. However, this new data frame will not be saved anywhere, so if you want to save it, you need to assign it to a new object. Tp do this, you can use the assignment operator `<-`.

```r
# Filter the data frame to select only the rows where the age variable is greater than 30 and

new_data <- my_data %>% filter(age > 30)
```

In this code, on the left side of the assignment operator `<-`, we have `new_data`, which is the name of the new data frame that will contain only the filtered subset of the data (i.e., the values where the `age` variable is greater than 30). The difference between this code and the previous code is that we are now saving the result to a new data frame called `new_data`, instead of just printing it to the console.

We can also combine multiple conditions when subsetting data. For example, we can select only the rows where the `age` variable is greater than 25 and the `height` variable is greater than 175.

```r
# Filter the data frame to select only the rows where the age variable is greater than 25 and

my_data %>% filter(age > 25 & height > 175)
```

```
     name age height      car
1 Charlie  35    180 Electric
2   Frank  50    190 Electric
```

There are many other ways to subset data in R, depending on the criteria you want to use. For example, you can use the `select()` function to select specific columns, the `arrange()` function to sort the data, and the `mutate()` function to create new variables. We will cover some of these functions in later sections.

2.8 Grouping and summarising data in R

At the end of this section, you will be able to:

- Group data in R
- Summarise data in R

Grouping and summarising data in R means grouping the data by one or more variables and then calculating summary statistics for each group. For example, you might want to calculate the mean age for each group of people based on their height.

If we use the `my_data` data frame from the previous section, we can group the data by the `car` variable and then calculate the mean age for each group.

```r
# Group the data frame by the car variable and calculate the mean age for each group

my_data %>% group_by(car) %>%
  summarise(mean_age = mean(age)) %>%
  ungroup()
```

```
# A tibble: 2 x 2
  car      mean_age
  <fct>       <dbl>
1 Electric     36.7
2 Petrol       38.3
```

Let's break this code down:

- `my_data` is the data frame that we want to group and summarise.
- `%>%` is the pipe operator, which is used to pass the data frame to the next function. This allows us to link multiple steps together in a single line of code.
- `group_by(car)` is the function that groups the data frame by the `car` variable.
- `summarise(mean_age = mean(age))` is the function that calculates the mean age for each group of cars. The `mean_age` variable is the name of the new variable that will contain the mean age for each group.
- `ungroup()` is the function that removes the grouping from the data frame. This is optional, but it is good practice to ungroup the data frame after you have finished summarising it.

The output of this code will be a new data frame that contains the mean age for each group of cars. The `car` variable is the grouping variable, and the `mean_age` variable is the summary statistic that we calculated for each group.

You can also calculate other summary statistics, such as the median, standard deviation, minimum, and maximum, using the `summarise()` function. You can also calculate multiple summary statistics at the same time by specifying multiple variables inside the `summarise()` function. For example, you can calculate the mean and standard deviation of the age variable for each group of cars.

```r
# Group the data frame by the car variable and calculate the mean and standard deviation of t

my_data %>% group_by(car) %>%
  summarise(mean_age = mean(age), sd_age = sd(age)) %>%
  ungroup()
```

```
# A tibble: 2 x 3
  car      mean_age sd_age
  <fct>       <dbl>  <dbl>
1 Electric     36.7  12.6
2 Petrol       38.3   7.64
```

In this code, we calculated the mean and standard deviation of the `age` variable for each group of cars. The `mean_age` and `sd_age` variables are the names of the new variables that will contain the mean and standard deviation for each group.

3 Exploratory and descriptive analysis

In this section, we will cover the basics of exploratory and descriptive analysis in R. We will learn how to conduct some of the most common descriptive statistics, such as mean, median, mode, standard deviation, and variance. We will also look at basic distribution plots, such as histograms and box plots, to visualize the data.

3.1 Mean, median, and mode

At the end of this section, you will be able to:

- Calculate the mean, median, and mode of a dataset

For this section we will use the `album_sales` dataset, which we have already loaded in some of the videos. Let's start by loading the dataset and displaying the first few rows:

```
library(tidyverse)

# Load the album_sales dataset. The location of the dataset will be different based on where

album_sales <- read.csv("Datasets/album_sales.csv")

# Display the first few rows of the dataset

head(album_sales)
```

```
   Adverts Sales Airplay Attract   Genre
1   10.256   330      43      10 Country
2  985.685   120      28       7     Pop
3 1445.563   360      35       7  HipHop
4 1188.193   270      33       7  HipHop
5  574.513   220      44       5   Metal
6  568.954   170      19       5 Country
```

The dataset contains 5 variables: Adverts, Sales, Airplay, Attract and Genre. We will focus on the `Sales` variable for this section.

```
# Calculate the mean of the Sales variable

mean_sales <- mean(album_sales$Sales)

mean_sales
```

```
[1] 193.2
```

Let's break down the code above:

- We used the `mean()` function to calculate the mean of the `Sales` variable in the `album_sales` dataset. To do this, we specified the dataset `album_sales` and the variable `Sales` using the `$` operator.
- The mean sales value is stored in the `mean_sales` variable.

Next, let's calculate the median of the `Sales` variable:

```
# Calculate the median of the Sales variable

median_sales <- median(album_sales$Sales)

median_sales
```

```
[1] 200
```

The code above calculates the median of the `Sales` variable in the `album_sales` dataset. The median sales value is stored in the `median_sales` variable.

Finally, let's calculate the mode of the `Sales` variable. Unfortunately, R does not have a built-in function to calculate the mode. However, we do this in the following way:

```
# Calculate the mode of the Sales variable

album_sales$Sales %>%
 table()
```

.

```
 10  30  40  50  60  70  80  90 100 110 120 130 140 150 160 170 180 190 200 210
  1   1   3   1   5   6   3   4   8   5  10   3  11  12   5   4   8   8   7  13
220 230 240 250 260 270 280 290 300 310 320 330 340 360
  6  17   7  10   3   3   5   8   6   2   4   2   3   6
```

We can see from the output that the mode of the `Sales` variable is 210, since that value appears most frequently in the dataset (13 times).

3.2 Standard deviation and variance

At the end of this section, you will be able to:

- Calculate the standard deviation
- Calculate the variance
- Calculate the range of a dataset
- Calculate the interquartile range (IQR)

Next, let's calculate the standard deviation and variance of the `Sales` variable in the `album_sales` dataset:

```
# Calculate the standard deviation of the Sales variable

sd_sales <- sd(album_sales$Sales)

sd_sales
```

```
[1] 80.69896
```

The code above calculates the standard deviation of the `Sales` variable in the `album_sales` dataset. The standard deviation value is stored in the `sd_sales` variable.

Next, let's calculate the variance of the `Sales` variable:

```
# Calculate the variance of the Sales variable

var_sales <- var(album_sales$Sales)

var_sales
```

```
[1] 6512.322
```

The code above calculates the variance of the `Sales` variable in the `album_sales` dataset. The variance value is stored in the `var_sales` variable.

3.3 Range and interquartile range

The range of a dataset is the difference between the maximum and minimum values. Let's calculate the range of the `Sales` variable in the `album_sales` dataset:

```
# Calculate the range of the Sales variable

range_sales <- range(album_sales$Sales)

range_sales
```

```
[1]  10 360
```

The code above calculates the range of the `Sales` variable in the `album_sales` dataset. The range of the sales values is stored in the `range_sales` variable.

The interquartile range (IQR) is the difference between the 75th percentile (Q3) and the 25th percentile (Q1) of a dataset. Let's calculate the IQR of the `Sales` variable in the `album_sales` dataset:

```
# Calculate the interquartile range of the Sales variable

IQR_sales <- IQR(album_sales$Sales)

IQR_sales
```

```
[1] 112.5
```

The code above calculates the interquartile range (IQR) of the `Sales` variable in the `album_sales` dataset. The IQR value is stored in the `IQR_sales` variable.

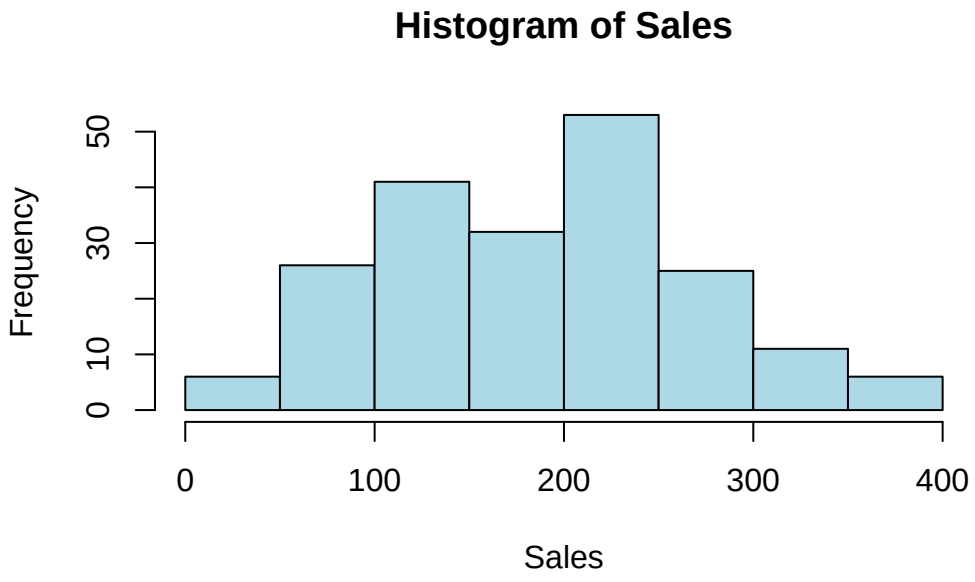
3.4 Distribution plots

At the end of this section, you will be able to:

- Create a histogram to visualize the distribution of a dataset
- Create a box plot to visualize the distribution of a dataset

Next, let's create a histogram to visualize the distribution of the `Sales` variable in the `album_sales` dataset:

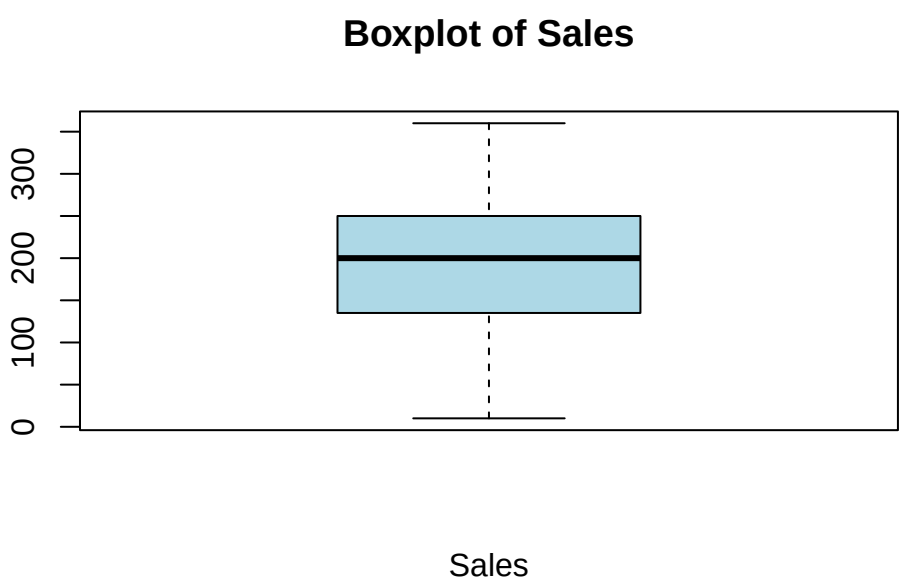
```
# Create a histogram of the Sales variable

hist(album_sales$Sales, main = "Histogram of Sales", xlab = "Sales", ylab = "Frequency", col
```



The code above creates a histogram of the `Sales` variable in the `album_sales` dataset. The histogram displays the frequency of sales values in the dataset. The only required argument for the `hist()` function is the variable you want to plot. The `main`, `xlab`, `ylab`, and `col` arguments are optional and allow you to customize the appearance of the histogram.

Finally, let's create a box plot to visualize the distribution of the `Sales` variable in the `album_sales` dataset:

```
# Create a box plot of the Sales variable

boxplot(album_sales$Sales, main = "Boxplot of Sales", xlab = "Sales", col = "lightblue")
```



The code above creates a box plot of the `Sales` variable in the `album_sales` dataset. The box plot displays the distribution of sales values, including the median, quartiles, and outliers. The only required argument for the `boxplot()` function is the variable you want to plot. The `main`, `xlab`, and `col` arguments are optional and allow you to customize the appearance of the box plot.

We will learn more about plotting data with ggplot2 in another section. However, for now, we have used the base R functions `hist()` and `boxplot()` to create simple distribution plots.

3.5 Assessing the normality of data

At the end of this section, you will be able to:

- Assess the skewness and kurtosis of a dataset
- Test the normality of a dataset using the Shapiro-Wilk test

Many statistical tests assume that the data is normally distributed. When your data sample size is small, violation of the normaility assumption could be an issue. Let's assess the normality of the `Sales` variable in the `album_sales` dataset by calculating the skewness and kurtosis. In order to do this, we will use the `psych` package, which provides functions for calculating skewness and kurtosis. If you haven't installed the `psych` package yet, you can do so by running the following code:

```
# Install the psych package if you haven't already

install.packages("psych")
```

Now, let's calculate the skewness and kurtosis of the `Sales` variable in the `album_sales` dataset:

```
# Load the psych package

library(psych)
```

```
Attaching package: 'psych'

The following objects are masked from 'package:ggplot2':

    %+%, alpha
```

```
# Calculate the skewness of the Sales variable

skew_sales <- skew(album_sales$Sales)

skew_sales
```

```
[1] 0.0432729
```

```
# Calculate the kurtosis of the Sales variable

kurt_sales <- kurtosi(album_sales$Sales)

kurt_sales
```

```
[1] -0.7157339
```

When interpreting the skewness and kurtosis values, remember that values of 0 indicate a normal distribution.

We can also test the normality of the `Sales` variable using the Shapiro-Wilk test. The null hypothesis of the Shapiro-Wilk test is that the data is normally distributed. Let's perform the Shapiro-Wilk test on the `Sales` variable in the `album_sales` dataset:

```r
# Perform the Shapiro-Wilk test on the Sales variable

shapiro.test(album_sales$Sales)
```

```
    Shapiro-Wilk normality test

data:  album_sales$Sales
W = 0.98479, p-value = 0.02965
```

The output of the Shapiro-Wilk test includes the test statistic and the p-value. If the p-value is less than 0.05, we reject the null hypothesis and conclude that the data is not normally distributed. If the p-value is greater than 0.05, we fail to reject the null hypothesis and conclude that the data is normally distributed.

> **Assessing normality**
>
> The shapiro-wilk test is sensitive to sample size. For small sample sizes, the test may be too conservative and reject the null hypothesis too often. For large sample sizes, the test may be too lenient and fail to reject the null hypothesis too often. Therefore, you should not rely solely on the Shapiro-Wilk test to assess the normality of your data. Visual inspection of the data using histograms and Q-Q plots is also recommended. Also remember that the central limit theorem states that the sampling distribution of the mean will be approximately normally distributed for large sample sizes, regardless of the distribution of the original data.
> We could also use non-parametric bootstrapping methods to deal with non-normal data. We will cover this in a later section.

4 Sampling, power and effect size

In this chapter, you will learn how to conduct some analysis on the related concepts of sampling, power, and effect size.

Effect size is a measure of the strength of the relationship between two variables in a statistical population. It is used to quantify the size of the difference between two groups or the strength of an association between two variables. In this section, we will learn how to calculate the effect size for different types of data.

The size of an effect is important when planning a study and trying to determine the sample size required. If an effect is small, we need a larger sample size to detect it. If an effect is large, we can detect it with a smaller sample size.

The ability of a study to detect an effect, using its sample, is called statistical power. Power is the probability that a study will correctly reject a false null hypothesis. In other words, power is the probability that a study will find a true effect when there is one (avoiding a Type 2 error).

> Tip
>
> The power of a study is influenced by the sample size, the effect size, and the significance level. A larger sample size, a larger effect size, and a higher significance level all increase the power of a study.
> However, since our significance level is usually set at 0.05 and the effect size is determined by the data, the sample size is the only factor that we can control to increase the power of a study.

4.1 Different measures of effect size

There are different ways to calculate effect size depending on the type of data and the statistical test used. Here are some common effect size measures:

- **Cohen's d**: This is a measure of the difference between two means in standard deviation units. It is commonly used in t-tests and ANOVA tests.

- **Eta-squared** (η^2): This is a measure of the proportion of variance in the dependent variable that is explained by the independent variable. It is commonly used in ANOVA tests.
- **Phi coefficient** (ϕ): This is a measure of the association between two binary variables. It is commonly used in chi-square tests.
- **Correlation coefficient** (r): This is a measure of the strength and direction of the relationship between two continuous variables. It is commonly used in correlation tests.

In the following sections, we will calculate the effect size for different types of data using some of these measures.

4.2 Calculating Cohen's d

Cohen's d is a measure of the difference between two means in standard deviation units. It is calculated as the difference between the means divided by the pooled standard deviation. The formula for Cohen's d is:

$$d = \frac{\bar{X}_1 - \bar{X}_2}{s_p}$$

where:

- $\bar{X}_1$ and $\bar{X}_2$ are the means of the two groups.
- s_p is the pooled standard deviation, calculated as:

$$s_p = \sqrt{\frac{(n_1 - 1)s_1^2 + (n_2 - 1)s_2^2}{n_1 + n_2 - 2}}$$

where:

- n_1 and n_2 are the sample sizes of the two groups.
- s_1 and s_2 are the standard deviations of the two groups.

Let's calculate Cohen's d for a hypothetical dataset with two groups. The dataset contains the following information:

- Group 1: Mean = 10, Standard deviation = 2, Sample size = 30
- Group 2: Mean = 12, Standard deviation = 3, Sample size = 30

To calculate Cohen's d, we first need to calculate the pooled standard deviation (s_p) using the formula above. Then, we can calculate Cohen's d using the formula for Cohen's d.

Let's calculate Cohen's d for this dataset using R:

```
# Calculate Cohen's d

# Group 1

mean1 <- 10

sd1 <- 2

n1 <- 30

# Group 2

mean2 <- 12

sd2 <- 3

n2 <- 30

# Calculate pooled standard deviation

sp <- sqrt(((n1 - 1) * sd1^2 + (n2 - 1) * sd2^2) / (n1 + n2 - 2))

# Calculate Cohen's d

d <- (mean1 - mean2) / sp


d
```

```
[1] -0.7844645
```

The calculated value of Cohen's d is -0.7844645. This negative value indicates that the mean of Group 1 is smaller than the mean of Group 2 by approximately 0.78 standard deviations.

4.3 Calculating Eta-squared and other effect size measures

It is possible to calculate other effect size measures such as Eta-squared, Phi coefficient. However, these measures are most commonly calculated using the output of statistical tests such as ANOVA and chi-square tests etc. To obtain these measures for the purpose of sample size calculation, you would usually look at previous studies or meta analyses to determine the expected effect size. For clinical research, you may also use the minimal clinically important difference (MCID) as a guide to determine the effect size.

4.4 Power analysis

Power analysis is a method used to determine the sample size required to detect an effect of a given size with a certain level of confidence. It is important to conduct a power analysis before conducting a study to ensure that the sample size is adequate to detect the effect of interest.

To conduct a power analysis, you need to specify the following parameters:

- The effect size: The size of the effect you want to detect. This is usually determined based on previous studies, meta-analyses or MCID.
- The significance level (α): The probability of rejecting the null hypothesis when it is true (Type 1 error rate). This is commonly set at 0.05.
- The power ($1 - \beta$): The probability of correctly rejecting the null hypothesis when it is false (1 - Type 2 error rate). This is commonly set at 0.80 or 0.90.
- The number of groups or conditions: The number of groups or conditions in the study.

The sample size required to achieve a desired power level can be calculated using power analysis functions in R. There are many packages in r for power analysis. The package `pwr` is one such package that provides functions to calculate the sample size required for different types of statistical tests.

The functions for some basic research designs are:

- `pwr.t.test()`: For t-tests
- `pwr.anova.test()`: For ANOVA tests
- `pwr.chisq.test()`: For chi-square tests
- `pwr.f2.test()`: For regression models

Let's calculate the sample size required to achieve a power of 0.80 for a t-test with the example data we used earlier. We will use the `pwr.t.test()` function from the `pwr` package to calculate the sample size required to achieve a power of 0.80 for a t-test with the following parameters:

- Effect size (Cohen's d) = -0.7844645
- Significance level (α) = 0.05

```
# Load the pwr package


library(pwr)

# Calculate the sample size required for a t-test
# using the d value calculated earlier

pwr.t.test(d = d, sig.level = 0.05, power = 0.80)
```

```
     Two-sample t test power calculation

              n = 26.50429
              d = 0.7844645
      sig.level = 0.05
          power = 0.8
    alternative = two.sided

NOTE: n is number in *each* group
```

The output of the `pwr.t.test()` function provides the sample size required to achieve a power of 0.80 for a t-test with the specified effect size and significance level. The output includes the following information:

- n = The sample size required for each group to achieve a power of 0.80.
- d = The effect size (Cohen's d) used in the power analysis.
- sig.level = The significance level used in the power analysis.
- power = The power level achieved with the specified sample size.

The sample size required to achieve a power of 0.80 for a t-test with the specified effect size and significance level is 26.5 for each group. Since the sample size must be a whole number, we would need to round up to the nearest whole number. Therefore, the sample size required for each group is 27.

4.5 More complex power analysis

For more complex designs, different approaches to power analysis might be necessary, such as using simulation. This is possible to do in R, but is beyond the scope of this chapter.

5 Basic statistical tests

In this chapter, we will learn how to perform basic statistical tests in R. These are all tests that you should be familiar with already from your statistics courses. We will cover the following tests:

- Independent t-test
- Paired t-test
- Wilcoxon signed-rank test
- Mann-Whitney U test
- Chi-squared test
- Correlation
- ANOVA

These examples will not be exhaustive, but they should give you a good starting point for performing these tests in R. For theoretical background, you can refer to any standard statistics textbook.

> At the end of this chapter, you will be able to:
>
> - Conduct several basic inferential tests with R

These examples will all use built-in datasets in R. Each example will include the necessary code to load the dataset and perform the test. However, you can also use your own datasets by loading them into R yourself and replacing the dataset name in the examples.

5.1 Independent t-test

The independent t-test is used to compare the means of two independent groups. In R, you can use the `t.test()` function to perform an independent t-test. Here is an example:

```
# This example uses the mtcars dataset, which is a built-in dataset in R that contains data c

# Load the mtcars dataset

data(mtcars)
```

```r
# Independent t-test example: Is there a difference in fuel efficiency between automatic and
# the variable am is a binary variable indicating the type of transmission (0 = automatic, 1

# Perform the independent t-test

t_test_result <- t.test(mpg ~ am, data = mtcars)

# Print the result

t_test_result
```

```
    Welch Two Sample t-test

data:  mpg by am
t = -3.7671, df = 18.332, p-value = 0.001374
alternative hypothesis: true difference in means between group 0 and group 1 is not equal to
95 percent confidence interval:
 -11.280194  -3.209684
sample estimates:
mean in group 0 mean in group 1
       17.14737        24.39231
```

In this example, we are comparing the fuel efficiency (mpg) of automatic and manual cars in the `mtcars` dataset. The `mpg` variable is the dependent variable, and the `am` variable is the independent variable. The `t.test()` function is used to perform the independent t-test, and the result is stored in the `t_test_result` variable.

If we look at the output, we can see the following:

- The t-test result is presented.
- The alternative hypothesis is that the means are not equal.
- The 95% confidence interval for the difference in means is presented.
- The 2 sample means are presented.

5.2 Paired t-test

The paired t-test is used to compare the means of two related groups. In R, you can use the `t.test()` function with the `paired = TRUE` argument to perform a paired t-test. Here is an example:

```r
# This example uses the sleep dataset, which is a built-in dataset in R that contains data on

# Load the sleep dataset

data(sleep)

# Paired t-test example: Is there a difference in sleep duration between the two drugs?

# Perform the paired t-test

paired_t_test_result <- t.test(sleep$extra ~ sleep$group, paired = TRUE)

# Print the result

paired_t_test_result
```

```
    Paired t-test

data:  sleep$extra by sleep$group
t = -4.0621, df = 9, p-value = 0.002833
alternative hypothesis: true mean difference is not equal to 0
95 percent confidence interval:
 -2.4598858 -0.7001142
sample estimates:
mean difference
          -1.58
```

In this example, we are comparing the sleep duration (`extra`) between patients who were given different drugs in the `sleep` dataset (note: 10 people were each measured twice). The `extra` variable is the dependent variable, and the `group` variable is the independent variable. The `t.test()` function is used to perform the paired t-test, and the result is stored in the `paired_t_test_result` variable.

If we look at the output, we can see the following:

- The t-test result is presented.
- The alternative hypothesis is that the means are not equal.
- The 95% confidence interval for the difference in means is presented.
- The mean difference is presented.

5.3 Wilcoxon signed-rank test

The Wilcoxon signed-rank test is a non-parametric test used to compare two related groups. In R, you can use the `wilcox.test()` function to perform a Wilcoxon signed-rank test. Here is an example:

```
# This example uses the sleep dataset, which is a built-in dataset in R that contains data or

# Load the sleep dataset

data(sleep)

# Wilcoxon signed-rank test example: Is there a difference in sleep duration between the two

# Perform the Wilcoxon signed-rank test

wilcoxon_test_result <- wilcox.test(sleep$extra ~ sleep$group, paired = TRUE)
```

```
Warning in wilcox.test.default(x = DATA[[1L]], y = DATA[[2L]], ...): cannot
compute exact p-value with ties
```

```
Warning in wilcox.test.default(x = DATA[[1L]], y = DATA[[2L]], ...): cannot
compute exact p-value with zeroes
```

```
# Print the result

wilcoxon_test_result
```

```

    Wilcoxon signed rank test with continuity correction

data:  sleep$extra by sleep$group
V = 0, p-value = 0.009091
alternative hypothesis: true location shift is not equal to 0
```

In this example, we are comparing the sleep duration (`extra`) between patients who were given different drugs in the `sleep` dataset. The `extra` variable is the dependent variable, and the `group` variable is the independent variable. The `wilcox.test()` function is used to perform the Wilcoxon signed-rank test, and the result is stored in the `wilcoxon_test_result` variable.

If we look at the output, we can see the following:

- The Wilcoxon signed-rank test result is presented (V is the sum of the ranks of the differences between the pairs of observations).
- The alternative hypothesis (true location shift is not equal to 0) is presented. This means that the medians of the two groups are not equal.

Note: if there are ties in the data, the exact p-value is not calculated. Instead, an approximate p-value is presented.

5.4 Mann-Whitney U test

The Mann-Whitney U test is a non-parametric test used to compare the means of two independent groups. In R, you can use the `wilcox.test()` function to perform a Mann-Whitney U test. Note: we will not include the `paired = TRUE` argument that we did in the previous example. Here is an example:

```
# This example uses the mtcars dataset, which is a built-in dataset in R that contains data c

# Load the mtcars dataset

data(mtcars)

# Mann-Whitney U test example: Is there a difference in fuel efficiency between automatic and

# Perform the Mann-Whitney U test

mann_whitney_test_result <- wilcox.test(mpg ~ am, data = mtcars)
```

```
Warning in wilcox.test.default(x = DATA[[1L]], y = DATA[[2L]], ...): cannot
compute exact p-value with ties
```

```
# Print the result

mann_whitney_test_result
```

```
    Wilcoxon rank sum test with continuity correction

data:  mpg by am
W = 42, p-value = 0.001871
alternative hypothesis: true location shift is not equal to 0
```

The output is interpreted in the same way as the Wilcoxon signed-rank test output.

5.5 Chi-squared test

The chi-squared test is used to test the association between two categorical variables. In R, you can use the `chisq.test()` function to perform a chi-squared test. Here is an example:

```
# This example uses the mtcars dataset, which is a built-in dataset in R that contains data c

# Load the mtcars dataset

data(mtcars)

# Chi-squared test example: Is there an association between the number of cylinders and the t

# Create a contingency table

contingency_table <- table(mtcars$cyl, mtcars$am)

# Perform the chi-squared test

chi_squared_test_result <- chisq.test(contingency_table)
```

```
Warning in chisq.test(contingency_table): Chi-squared approximation may be
incorrect
```

```
# Print the result

chi_squared_test_result
```

```

    Pearson's Chi-squared test

data:  contingency_table
X-squared = 8.7407, df = 2, p-value = 0.01265
```

In this example, we are testing the association between the number of cylinders (`cyl`) and the type of transmission (`am`) in the `mtcars` dataset. The `chisq.test()` function is used to perform the chi-squared test, and the result is stored in the `chi_squared_test_result` variable.

If we look at the output, we can see the following:

- The chi-squared test result is presented.

5.6 Correlation

Correlation is used to test the relationship between two continuous variables. In R, you can use the `cor.test()` function to calculate the correlation coefficient with significance test. Here is an example:

```
# This example uses the mtcars dataset, which is a built-in dataset in R that contains data c

# Load the mtcars dataset

data(mtcars)

# Correlation example: Is there a relationship between fuel efficiency and horsepower?

# Calculate the correlation coefficient

correlation_result <- cor.test(mtcars$mpg, mtcars$hp)

# Print the result

correlation_result
```

```
    Pearson's product-moment correlation

data:  mtcars$mpg and mtcars$hp
t = -6.7424, df = 30, p-value = 1.788e-07
alternative hypothesis: true correlation is not equal to 0
95 percent confidence interval:
 -0.8852686 -0.5860994
sample estimates:
       cor
-0.7761684
```

In this example, we are testing the relationship between fuel efficiency (`mpg`) and horsepower (`hp`) in the `mtcars` dataset. The `cor.test()` function is used to calculate the correlation coefficient, and the result is stored in the `correlation_result` variable.

If we look at the output, we can see the following:

- The correlation coefficient is presented in the last line of the output.

- The significance level of the correlation is tested using a t-test, which is also presented in the output.
- The confidence interval for the correlation coefficient is presented.
- The alternative hypothesis is that the correlation is not equal to 0.

5.7 One-way ANOVA

One-way ANOVA (Analysis of Variance) is used to test the differences between the means of three or more groups. In R, you can use the `aov()` function to perform an ANOVA. Here is an example:

```
# This example uses the mtcars dataset, which is a built-in dataset in R that contains data c

# Load the mtcars dataset

data(mtcars)

# One-way ANOVA example: Is there a difference in fuel efficiency between cars with differen

# cyl should be a factor

mtcars$cyl <- as.factor(mtcars$cyl)

# Perform the ANOVA

anova_result <- aov(mpg ~ cyl, data = mtcars)

# Print the result

summary(anova_result)
```

```
            Df Sum Sq Mean Sq F value   Pr(>F)
cyl          2  824.8   412.4    39.7 4.98e-09 ***
Residuals   29  301.3    10.4
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

In this example, we are testing the differences in fuel efficiency (`mpg`) between cars with different numbers of cylinders (`cyl`) in the `mtcars` dataset. The `aov()` function is used to perform the ANOVA, and the result is stored in the `anova_result` variable.

If we look at the output, we can see the following:

- The ANOVA result is within the `summary()` function. The summary includes the F-statistic, the p-value, and the significance level of the ANOVA test.

5.8 Factorial ANOVA

Factorial ANOVA is used to test the effects of two or more independent variables on a dependent variable. In R, you can use the `aov()` function with interaction terms to perform a factorial ANOVA. Here is an example:

```
# This example uses the mtcars dataset, which is a built-in dataset in R that contains data c

# Load the mtcars dataset

data(mtcars)

# Factorial ANOVA example: Is there an interaction effect between the number of cylinders and

# cyl and am should be factors

mtcars$cyl <- as.factor(mtcars$cyl)
mtcars$am <- as.factor(mtcars$am)

# Perform the factorial ANOVA

factorial_anova_result <- aov(mpg ~ cyl * am, data = mtcars)

# Print the result

summary(factorial_anova_result)
```

```
            Df Sum Sq Mean Sq F value   Pr(>F)
cyl          2  824.8   412.4  44.852 3.73e-09 ***
am           1   36.8    36.8   3.999   0.0561 .
cyl:am       2   25.4    12.7   1.383   0.2686
Residuals   26  239.1     9.2
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

In this example, we are testing the interaction effect between the number of cylinders (`cyl`) and the type of transmission (`am`) on fuel efficiency (`mpg`) in the `mtcars` dataset. The

`aov()` function is used to perform the factorial ANOVA, and the result is stored in the `factorial_anova_result` variable.

The interaction term is specified using the `*` operator in the formula.

If we look at the output, we can see the following:

- The ANOVA result is within the `summary()` function. The summary includes the F-statistic, the p-value, and the significance level of each term tested in the ANOVA.
- Each independent variable is tested separately (main effects), and the interaction effect is also tested.
- The interaction effect is denoted by the `cyl:am` term in the output.

5.9 Conclusion

In this chapter, we have covered several basic statistical tests that you can perform in R. These are included for reference and to help you get started with performing statistical tests in R. However, we will be focusing on modelling our data, using different types of regression models, rather than re-learning these tests.

6 Correlation in R

At the end of this chapter, you will be able to:

- Conduct and interpret a correlation analysis in R

6.1 What is Correlation?

Correlation is a measure of the strength and direction of a relationship between two variables. It is most commonly used when we want to see if there is a relationship between two *continuous* variables. However, it is possible to run correlations between a continuous and a categorical variable (this is known as point-biserial correlation) or between two categorical variables (this is known as phi coefficient).

Sticking to the more conventional form of correlation; when we calculate correlation, we get an r value of between -1 and 1. This tells us two things:

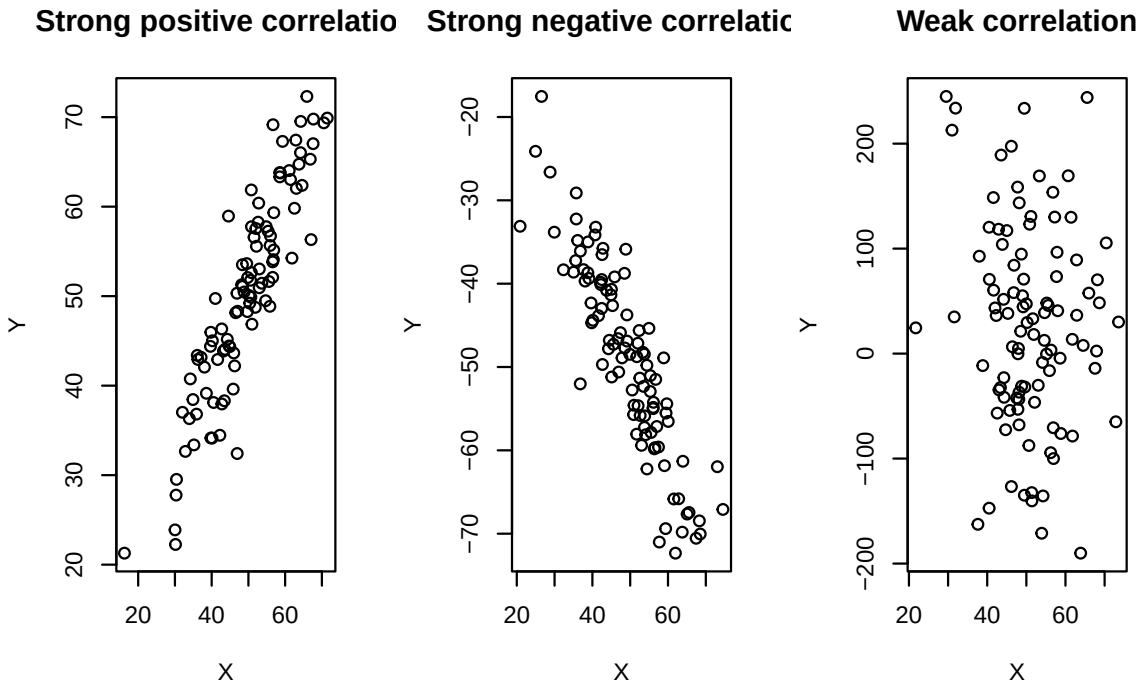
1. The closer the value is to 1, the stronger the relationship. The closer the value is to 0, the weaker the relationship.
2. The sign of the value tells us the direction of the relationship. Positive values indicate a positive relationship (i.e. as one variable increases, so does the other). Negative values indicate a negative relationship (i.e. as one variable increases, the other decreases).

We also often calculate the significance of the correlation, which tests against a null hypothesis that the correlation is 0. This is not part of the correlation per se, but it is often part of correlational research questions.

6.2 Visualising correlation

We can visualise correlation using a scatterplot. This is a graph where each data point is plotted on a graph, with one variable on the x axis and the other on the y axis. If the data points form something resembling a straight line, with all of the data points in a consistent pattern, then we have a strong correlation. If the data points are scattered, then we have a

weaker correlation. However, if the data points are inconsistent or more diffuse, the correlation is weaker. The direction of the line tells us the direction of the correlation.



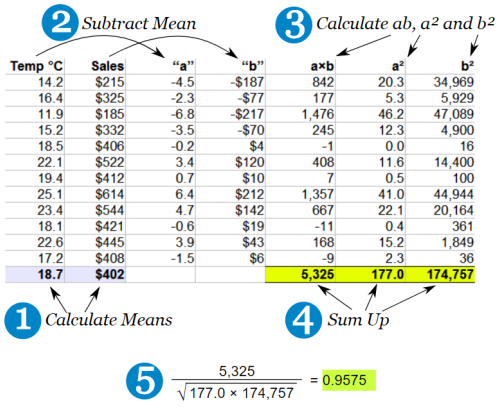
Visualising the data in this way can give us a good idea of the strength and direction of the correlation. However, it is not a substitute for running the correlation itself. At the same time, correlation coefficients can be misleading on their own. It is always a good idea to visualise the data as well.

6.3 How is correlation calculated?

Correlation can be thought of as covariance divided by individual variance. Covariance is a measure of how much two variables change together. Variance is a measure of how much a variable changes on its own. When we divide covariance by variance, we get a value that is standardised and can be compared across different data sets.

If the changes are consistent with both variables (i.e. the covariance is higher and the individual variance is lower), then the final correlation value will be higher. However, if the changes are inconsistent (i.e. the covariance is lower and the individual variance is higher), then the final correlation value will be lower.

$$r_{xy} = \frac{\sum(x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum(x_i - \bar{x})^2 \sum(y_i - \bar{y})^2}}$$



Temp °C	Sales	"a"	"b"	a×b	a²	b²
14.2	$215	-4.5	-$187	842	20.3	34,969
16.4	$325	-2.3	-$77	177	5.3	5,929
11.9	$185	-6.8	-$217	1,476	46.2	47,089
15.2	$332	-3.5	-$70	245	12.3	4,900
18.5	$406	-0.2	$4	-1	0.0	16
22.1	$522	3.4	$120	408	11.6	14,400
19.4	$412	0.7	$10	7	0.5	100
25.1	$614	6.4	$212	1,357	41.0	44,944
23.4	$544	4.7	$142	667	22.1	20,164
18.1	$421	-0.6	$19	-11	0.4	361
22.6	$445	3.9	$43	168	15.2	1,849
17.2	$408	-1.5	$6	-9	2.3	36
18.7	**$402**			**5,325**	**177.0**	**174,757**

6.4 Which correlation to use?

When we run correlation in R, we use the *cor.test()* command. This command will give us the correlation value, the p value and the confidence intervals.

We can specify a Pearson correlation (the default) or a Spearman correlation (for non-parametric data).

6.4.1 Running correlation in R

- R can run correlations using the *cor.test()* command

```
cor.test(regression_data$treatment_duration,regression_data$aggression_level)
```

```
    Pearson's product-moment correlation

data:  regression_data$treatment_duration and regression_data$aggression_level
```

```
t = -9.5503, df = 98, p-value = 1.146e-15
alternative hypothesis: true correlation is not equal to 0
95 percent confidence interval:
 -0.7838251 -0.5765006
sample estimates:
       cor
-0.6942996
```

In the above example, we are testing the correlation between treatment duration and aggression level. Each variable is separated by a comma.

6.4.2 Interpreting the output

- The r value tells us the strength and direction of the relationship
- In the output it is labelled as "cor" (short for correlation)

Correlation values can range from -1 to 1. The closer the value is to 1, the stronger the relationship. The closer the value is to 0, the weaker the relationship. Positive values indicate a positive relationship (i.e. as one variable increases, so does the other). Negative values indicate a negative relationship (i.e. as one variable increases, the other decreases).

```
cor.test(regression_data$treatment_duration,regression_data$aggression_level)
```

```
    Pearson's product-moment correlation

data:  regression_data$treatment_duration and regression_data$aggression_level
t = -9.5503, df = 98, p-value = 1.146e-15
alternative hypothesis: true correlation is not equal to 0
95 percent confidence interval:
 -0.7838251 -0.5765006
sample estimates:
       cor
-0.6942996
```

6.4.3 Check the significance of the correlation

- We can see that the significance by looking at the p value
  - The significance is 1.146^-15
  - This means: 0.0000000000000001146

- Therefore p value < 0.05

```
cor.test(regression_data$treatment_duration,regression_data$aggression_level)
```

```
    Pearson's product-moment correlation

data:  regression_data$treatment_duration and regression_data$aggression_level
t = -9.5503, df = 98, p-value = 1.146e-15
alternative hypothesis: true correlation is not equal to 0
95 percent confidence interval:
 -0.7838251 -0.5765006
sample estimates:
       cor
-0.6942996
```

> **Exponent values**
>
> When we see a value like 1.146e-15, this is a shorthand way of writing a very small number. The e-15 means that we move the decimal point 15 places to the left. So 1.146e-15 is the same as 0.000000000000001146

7 Simple Regression in R

At the end of this chapter, you will be able to:

- Conduct and interpret a regression analysis in R

8 What is regression analysis?

Regression analysis is a statistical technique used to model the relationship between an outcome (dependent) variable and one or more predictor (criterion/independent) variables. The goal of regression analysis is to understand how the outcome variable changes as the predictor variable changes.

When we say simple regression, we mean that there is only one predictor variable. This is to distinguish from multiple regression, where there are two or more predictor variables.

> **!** Regression does not imply causation
>
> Regression analysis can show that two variables are related, but it does not prove that one variable causes the other. To establish causation, you need a research design that can rule out alternative explanations, i.e., an experiment.

9 Simple regression in R

In R, we can conduct a simple regression analysis using the `lm()` function. The `lm()` function stands for linear model and is used to fit linear models. The syntax for the `lm()` function is:

```
lm(y ~ x, data)
```

Where `y` is the outcome variable, `x` is the predictor variable, and `data` is the dataset containing the variables.

Let's conduct a simple regression analysis using the `lm()` function in R. We will use the `regression_data` dataset, which contains `trust_score` as the outcome variable and `treatment_duration` as the predictor variable.

```
# Fit a simple regression model

regression_model <- lm(trust_score ~ treatment_duration, data = regression_data) ①
```

① We fit a simple regression model using the `lm()` function. The outcome variable is `trust_score`, and the predictor variable is `treatment_duration`. The data are specified using the `data` argument.

> What is the model?
>
> The terms **model** or **fit**, are commonly used in regression analysis.
> Model refers to the variables that you included in the model as well as the relationship between them. For example, our model above includes the predictor variable **depression** and the outcome variable **avoidance**. The model is the relationship between these variables. In this case, we are saying that **avoidance** is predicted by **depression**.
> A different model would be one that included different predictor variables. For example, we could have a model that included the predictor variable **treatment** and the outcome variable **avoidance**, or model that includes both **depression** and **treatment** as predictor variables and **avoidance** as the outcome variable.
> Fit refers to the idea that we want to know if this model is a good fit for the data. In other words, does the model explain the data well?
> We might find that our first model is not a good fit for the data. In this case, we might try a different model. We will learn next week how to compare different models to see

which one is the best fit for the data.

9.1 Checking model assumptions

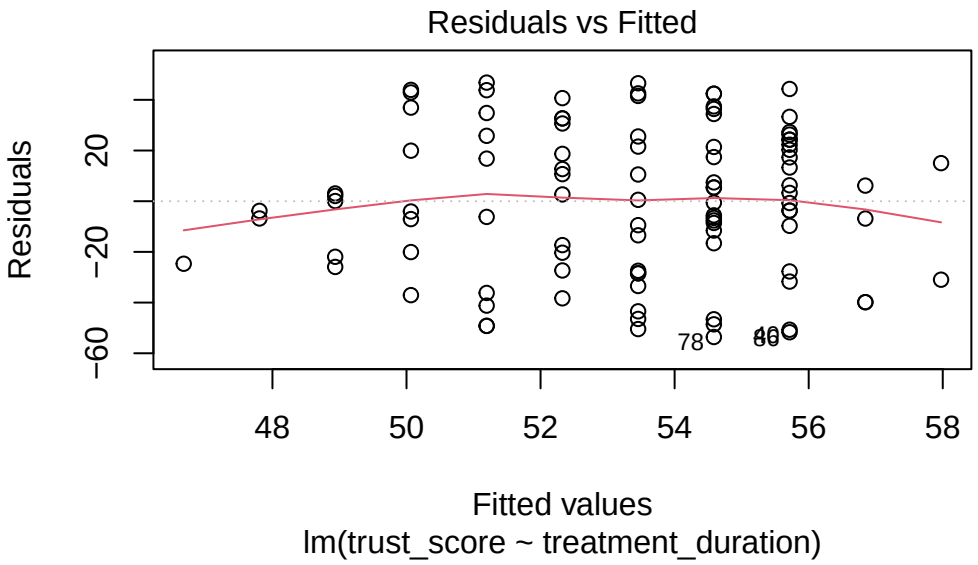
Before interpreting the results of a regression analysis, it is essential to check the assumptions of the model. The key assumptions of linear regression are:

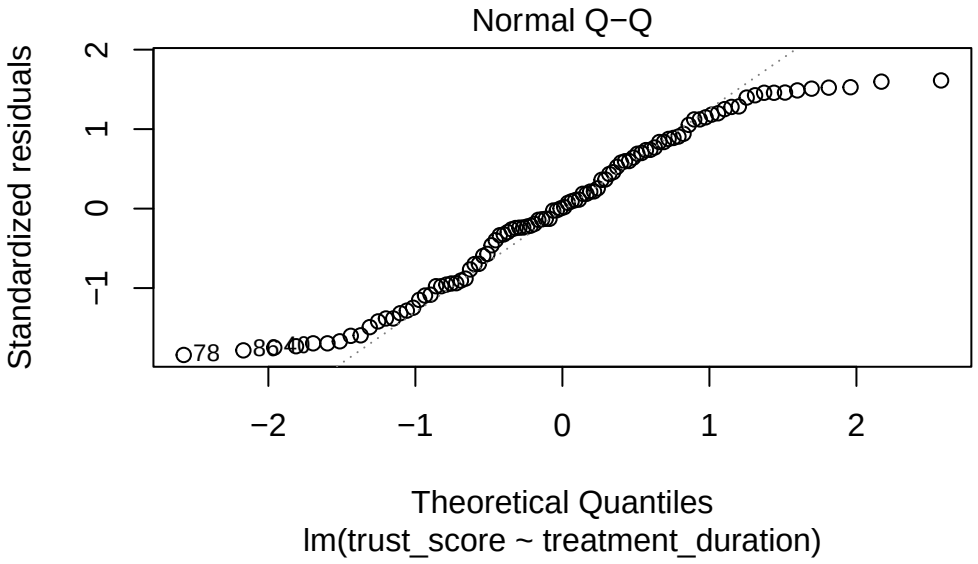
1. **Linearity**: The relationship between the predictor and outcome variables is linear.
2. **Independence**: The residuals (errors) are independent of each other.
3. **Homoscedasticity**: The residuals have constant variance.
4. **Normality**: The residuals are normally distributed.

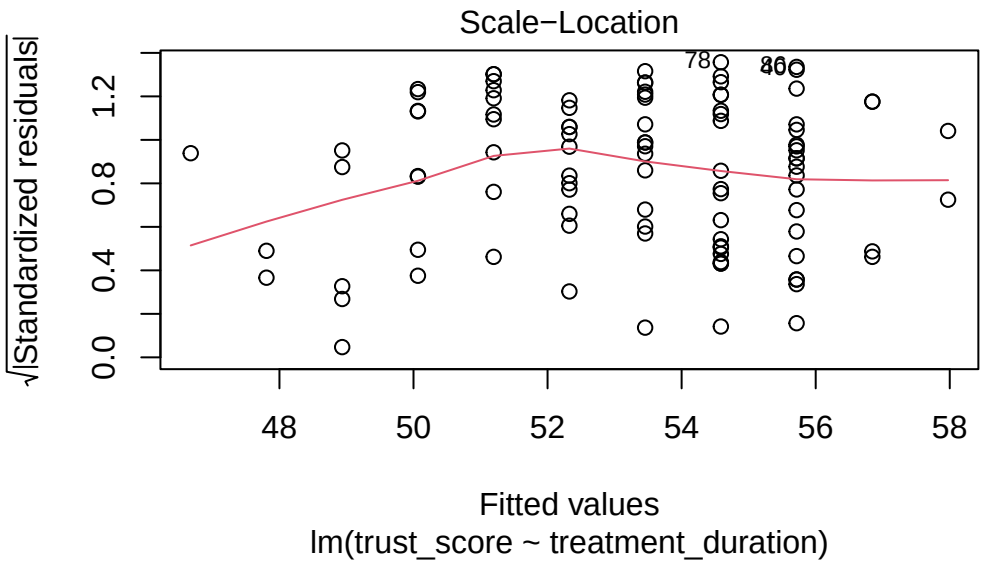
We can check these assumptions using diagnostic plots. The `plot()` function in R can be used to create diagnostic plots for the regression model.

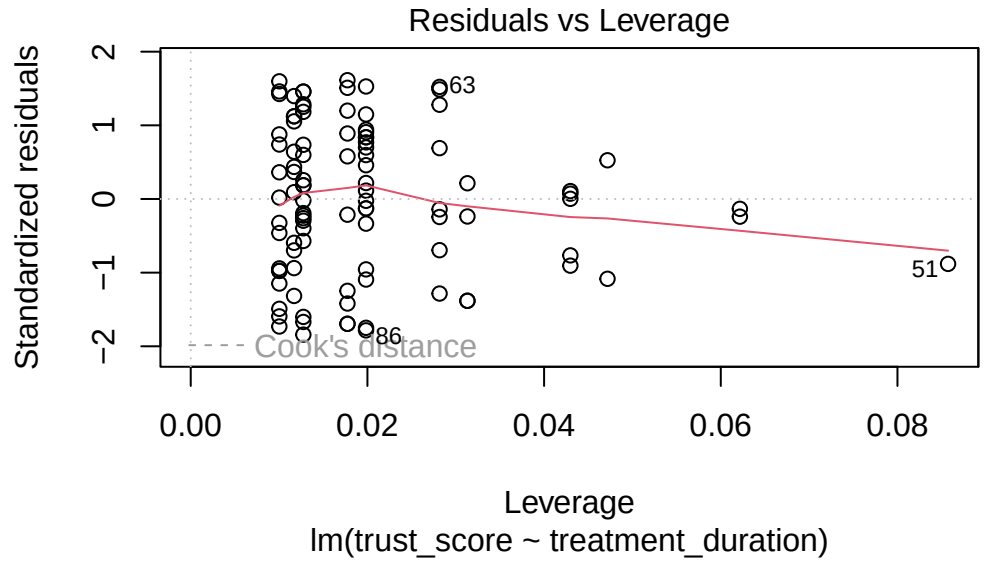
```
# Create diagnostic plots

plot(regression_model)                                                    ①
```

① We create diagnostic plots for the regression model using the `plot()` function. The diagnostic plots include a scatterplot of the residuals against the fitted values, a Q-Q plot of the residuals, a scale-location plot, and a plot of the residuals against the predictor variable.


Residuals vs Fitted
Residuals
20
−20
−60
48
50
52
54
56
58
Fitted values
lm(trust_score ~ treatment_duration)


Normal Q−Q
Standardized residuals
2
1
0
−1
−2
Theoretical Quantiles
lm(trust_score ~ treatment_duration)


Scale−Location
√|Standardized residuals|
0.0
0.4
0.8
1.2
48
50
52
54
56
58
Fitted values
lm(trust_score ~ treatment_duration)


Residuals vs Leverage
Standardized residuals
−2
−1
0
1
2
63
51
86
Cook's distance
0.00
0.02
0.04
0.06
0.08
Leverage
lm(trust_score ~ treatment_duration)

9.2 Check the assumptions

We can check the assumptions of linear regression using the plot() function.

```
# check the assumptions

plot(model)                                                    ①
```

① The **plot()** function is used to create a plot of the model. It actually shows several plots in sequence.

When you run the plot function, there will be a message in the console that says "Hit to see next plot:". You need to press enter to move through the plots.

9.2.1 The assumption of independence

This assumption is the idea that each observation is independent of the others. In other words, the value of one observation (e.g. a participant's score) should not be related to the value of another participant's score. This is not interpreted from the plots, but is an assumption that should be considered when collecting data. For example, you would not use this approach to analyse data from a repeated measures design, because the observations are not independent.

9.2.2 The assumption of linearity

The first plot is a plot of the residuals against the fitted values. This plot is used to check the assumption of linearity. The assumption of linearity states that the relationship between the predictor variable and the outcome variable is linear. In other words, the relationship between the predictor variable and the outcome variable can be described by a straight line. If the relationship is not linear, then the model is not a good fit for the data.

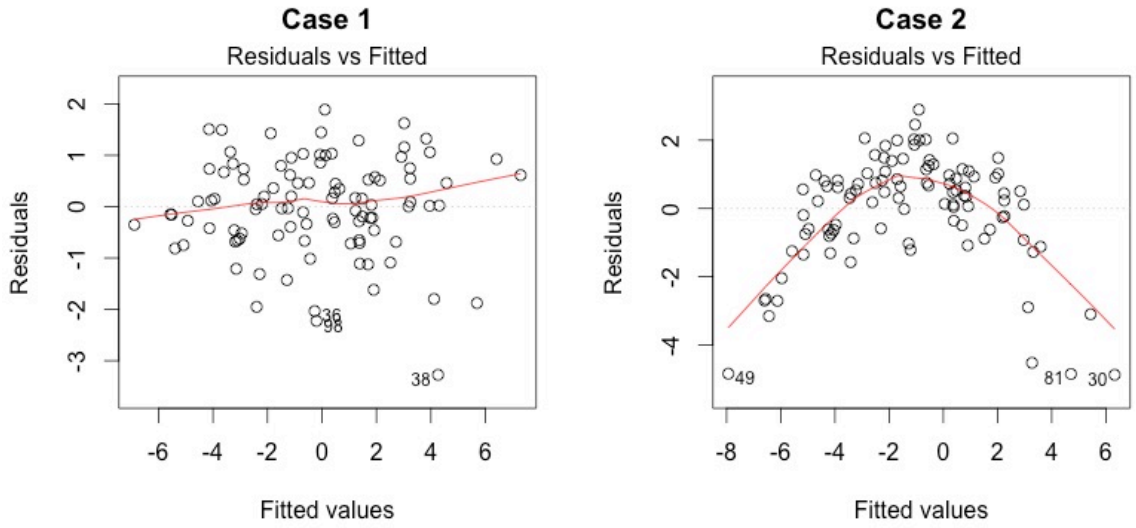


Figure 9.1: Linear and non-linear data

Linear and non-linear residuals

Good: Random scatter around the line. The line is straight.
Bad: Non-random scatter around the line. The line is not straight.
What do we do if this assumption is violated?: Linearity is a very important assumption. If the assumption is violated, then the linear model is likely not a good fit for the data. In this case we should probably try a different model. Data are never perfect, but we shouldn't ignore a clear violation of linearity.

9.2.3 The assumption of normality

The second plot is a normal Q-Q plot of the residuals. This plot is used to check the assumption of normality. The assumption of normality states that the residuals are normally distributed. If the residuals are not normally distributed, then the model is not a good fit for the data.

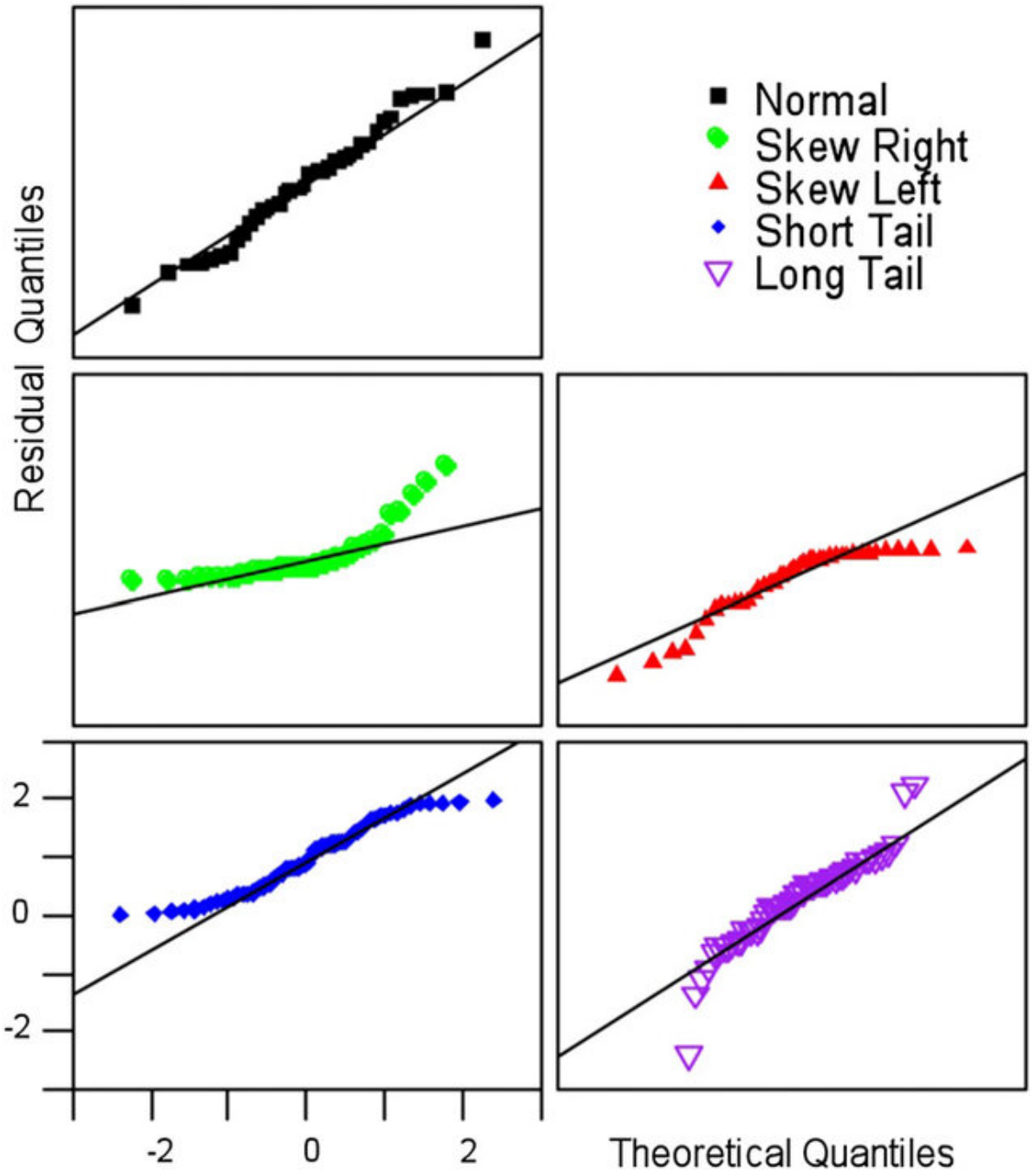


Figure 9.2: qqplot of residuals

Normal and non-normal residuals

Good: The points follow the line.
Bad: The points do not follow the line.
What do we do if this assumption is violated?: Normality affects the accuracy of beta values, significance tests and confidence intervals. However, it is most important with small sample sizes. As sample size increases, the assumption of normality becomes less important.

9.2.4 The assumption of homoscedasticity

The third plot is a scale-location plot of the residuals against the fitted values. This plot is used to check the assumption of homoscedasticity. The assumption of homoscedasticity states that the residuals have equal variance. If the residuals do not have equal variance, then the model is not a good fit for the data.

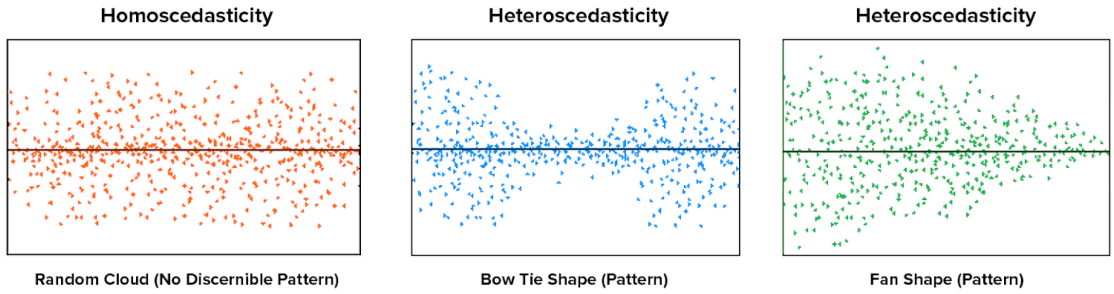


Figure 9.3: homoscedasticity

Homoscedastic and heteroscedastic residuals

Good: The points are randomly scattered around the line. and the line is horizontal.
Bad: The points are not randomly scattered around the line. and the line is not horizontal.
What do we do if this assumption is violated?: This will affect the accuracy of the beta values, significance tests and confidence intervals. Essentially, it means that conclusions we draw from the model are less accurate. What we do depends on the situation. Transformation of the DV (e.g. log transformation) might help. If not, there are weighted regression models that can be used.

9.2.5 Checking for outliers or influential cases

The fourth plot is a plot of Cook's distance. This plot is used to check for outliers. An outlier is a data point that is very different from the rest of the data. If there are outliers, then they could be affecting the regression model. The threshold for Cook's distance is 1. If a data point has a Cook's distance greater than 1, then it is considered an outlier.

The fifth plot is a plot of the residuals against the leverage. The sixth plot is a plot of the Cook's distance against the leverage. They are pretty much the same plots as the plot 4.

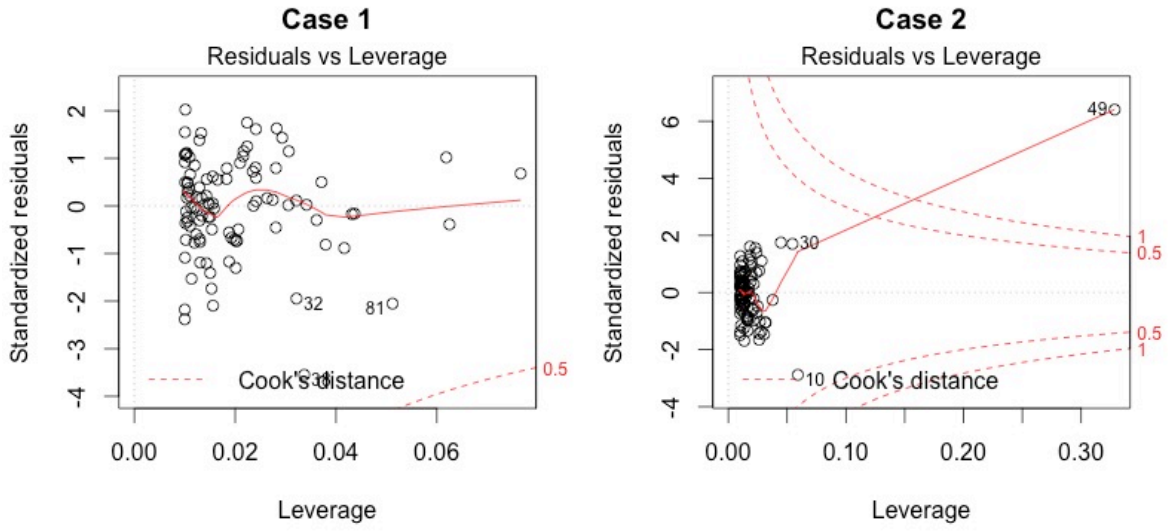


Figure 9.4: Influential Cases and Leverage

On the plot above, Cook's Distance is indicated by a red line. If a data point is outside the red line, then it is considered an outlier.

Leverage is the idea that a particular outlier might have a lot of influence on the regression model. Look for data points that are outside the red line on the top right or bottom right of the plot. These are data points that have a lot of leverage and might be influencing the regression model.

Outliers and influential cases

Good: No data points outside the red lines.
Bad: Data points outside the red lines. Outliers with high leverage.
What do we do if this assumption is violated?: Outliers will affect the calculation of variances (e.g. sum of squares or standard deviation) that are used in many calculations

related to the regression model. If there are influentual cases, we should consider removing them from the analysis. When doing so, it is important to explain why you removed them, and be transparent about how this affected the model results.

9.3 Interpret the regression model results

To view the results of the regression model, we use the summary() function.

```
# View the results

summary(regression_model)                                    ①
```

① We view the results of the regression model using the `summary()` function. The output includes the coefficients, standard errors, t-values, p-values, and R-squared value of the model.

```
Call:
lm(formula = trust_score ~ treatment_duration, data = regression_data)

Residuals:
    Min      1Q  Median      3Q     Max
-53.585 -24.991   0.304  24.285  46.804

Coefficients:
                  Estimate Std. Error t value Pr(>|t|)
(Intercept)         64.754     14.176   4.568 1.44e-05 ***
treatment_duration  -1.130      1.371  -0.824    0.412
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 29.29 on 98 degrees of freedom
Multiple R-squared:  0.006886,  Adjusted R-squared:  -0.003248
F-statistic: 0.6795 on 1 and 98 DF,  p-value: 0.4118
```

9.3.1 Call: The regression formula

The first line of the output is the regression formula. This is the formula that was used to create the model.

9.3.2 Residuals: The residuals

The second line of the output is the residuals. The residuals are the difference between the actual values of the outcome variable and the predicted values of the outcome variable. This section is giving us some summary statistics about the residuals. However, we usually check the assumptions using the plots.

9.3.3 Coefficients: The beta values

The third section of the output is the coefficients. You will see a line of values for the **intercept** and another line for each of the predictor variables in the model. **Estimate** is the beta value. **Std. Error** is the standard error of the beta value. **Pr(>|t|)** is the p value for the beta value.

- Intercept: We are not usually interested in this line by itself. It is the value of the outcome variable when all of the predictor variables are equal to zero. In this case, it is the value of avoidance when depression is equal to zero. However, it might be the case that depression cannot be equal to zero. In this case, the intercept would not be meaningful. *If the predictor were a categorical variable, then the intercept would be the value of the outcome variable when the predictor variable is equal to the reference category (i.e. The mean of the outcome for that group).*
- Depression: This is the beta value for depression. It is the amount that avoidance changes when depression increases by one unit. What *unit* means, depends on how the variables were measured, so it is likely to mean one point in the scale used to measure depression, for example.

The final section in the output shows:

- **Residual standard error**. This is the standard deviation of the residuals. It is the average amount that the actual values of the outcome variable differ from the predicted values of the outcome variable.
- **Multiple R-squared**. This is the R-squared value. It is the amount of variance in the outcome variable that is explained by the model. We usually talk about this as a percentage value.
- **Adjusted R-squared**. This is the adjusted R-squared value. It is the amount of variance in the outcome variable that is explained by the model, adjusted for the number of predictor variables in the model. This is to account for the fact that having more predictors in the model will always increase the R-squared value, even if the predictors are not related to the outcome variable. It is relevant when we have more than one predictor variable in the model.

- **F Statistic**. The F value comes from the ANOVA that is used to test the significance of the model. It tests the null hypothesis that all of the beta values are equal to zero.
- **p-value**. This is the p value for the F statistic (the significance of the overall regression model, with all of the predictors).

9.4 Regression with a categorical predictor variable

So far, we have conducted a regression analysis with a continuous predictor variable. Now, let's consider a regression analysis with a categorical predictor variable.

Our next hypothesis is that the level of depression is different for each treatment group. We can test this hypothesis using a regression model with a categorical predictor variable. Let's conduct a regression analysis with the `treatment` variable as the predictor variable. If there are 2 levels of the predictor variable, then the model will compare the two levels. If there are more than 2 levels, then the model will compare each level to the reference level. By default, R uses the first level of the predictor variable as the reference level. Howeverm you can specify a different reference level using the `relevel()` function.

```
# specify the reference level - this is more useful when we have more than 2 levels

regression_data$treatment_group <- as.factor(regression_data$treatment_group) ⓪
regression_data$treatment <- relevel(regression_data$treatment_group, ref = "therapy1") ①

# Fit a regression model with a categorical predictor variable

regression_model2 <- lm(aggression_level ~ treatment_group, data = regression_data) ②

# Summary of the model

summary(regression_model2)                                                    ③
```

⓪ We convert the `treatment` variable to a factor variable using the `as.factor()` function. This is necessary for R to recognize the variable as a categorical variable.

① We specify the reference level for the `treatment` variable using the `relevel()` function. In this case, we set the reference level to "therapy1". This is likely the default level, but we are specifying it here for clarity. If we had more than 2 levels, we could specify a different reference level using the `ref` argument. For example, 'ref = "control"

② We fit a regression model with the `treatment_group` variable as the predictor variable. The outcome variable is `aggression_level`. The data are specified using the `data` argument.

③ We view the results of the regression model using the `summary()` function. The output includes the coefficients, standard errors, t-values, p-values, and R-squared value of the model.

```
Call:
lm(formula = aggression_level ~ treatment_group, data = regression_data)

Residuals:
    Min      1Q  Median      3Q     Max
-4.6800 -1.4882 -0.0185  1.3460  4.4131

Coefficients:
                         Estimate Std. Error t value Pr(>|t|)
(Intercept)                4.6800     0.2843  16.464  < 2e-16 ***
treatment_grouptherapy2    1.3201     0.4103   3.217  0.00175 **
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 2.05 on 98 degrees of freedom
Multiple R-squared:  0.09554,   Adjusted R-squared:  0.08631
F-statistic: 10.35 on 1 and 98 DF,  p-value: 0.001753
```

The output tells us that the model is significant, and that the beta values for the treatment_group therapy2 is different from the reference group (therapy1) which is the intercept of this model. The beta values are the difference in the outcome variable between the reference group and the other groups. The beta value for the reference group is the mean of the outcome variable (aggression_level) for that group. The beta values for therapy2 is the difference in the mean outcome variable for that group and the reference group.

Interpreting the results of a regression model with a categorical predictor variable

When we have a categorical predictor variable, the interpretation of the beta values changes. The beta values are the difference in the outcome variable between the reference category and the other categories. The reference category is the category that is not included in the model. The beta value for the reference category is the mean of the outcome variable for that category. The beta values for the other categories are the difference in the outcome variable between that category and the reference category.
If you want to see this done another way, if your predictor has 2 levels, you can conduct a t-test and compare the means of the two groups. The t-test will give you the same results as the regression model. The t-test is a simpler way to compare the means of two groups, but the regression model is more flexible and can handle more complex models (by adding more predictor variables).

⚠ Testing assumptions with a categorical predictor variable

When we have a categorical predictor variable, we need to be careful about how we interpret the assumptions of linear regression. The assumptions of linearity, normality and homoscedasticity are still relevant. However, because we have a categorical predictor, the plots will look different. We need to check the assumptions for each level of the categorical predictor variable. Some of the plots will change to reflect this.

9.5 Comparing multiple levels of predictor variables (estimated marginal means)

If we have more than 2 levels of the predictor variable, we can compare each level to the reference level. We can also compare all levels to each other using the `emmeans` package. The `emmeans` package provides estimated marginal means for the levels of a predictor variable. We can use the `emmeans()` function to calculate the estimated marginal means for the levels of the predictor variable.

For this example, let's use the mtcar dataset, which contains information about cars. We will conduct a regression analysis with the `cyl` variable as the predictor variable and the `mpg` variable as the outcome variable. The `cyl` variable has 3 levels (4, 6, and 8 cylinders). We will compare the mean `mpg` for each level of the `cyl` variable.

```
# Load the mtcars dataset

data(mtcars)

# Convert the cyl variable to a factor variable

mtcars$cyl <- as.factor(mtcars$cyl)

# Fit a regression model with the cyl variable as the predictor variable

regression_model3 <- lm(mpg ~ cyl, data = mtcars)                    ①

# Summary of the model

summary(regression_model3)                                           ②

# Compare the levels of the predictor variable with pairwise comparisons

library(emmeans)
```

③ We use the `emmeans()` function from the `emmeans` package to calculate the estimated marginal means for the levels of the `cyl` variable. The `pairwise ~ cyl` argument specifies that we want to compare the levels of the `cyl` variable with pairwise comparisons.

```
Welcome to emmeans.
Caution: You lose important information if you filter this package's results.
See '? untidy'
```

```
emmeans(regression_model3, pairwise ~ cyl)                    ③
```

```
Call:
lm(formula = mpg ~ cyl, data = mtcars)

Residuals:
    Min      1Q  Median      3Q     Max
-5.2636 -1.8357  0.0286  1.3893  7.2364

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)  26.6636     0.9718  27.437  < 2e-16 ***
cyl6         -6.9208     1.5583  -4.441 0.000119 ***
cyl8        -11.5636     1.2986  -8.905 8.57e-10 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 3.223 on 29 degrees of freedom
Multiple R-squared:  0.7325,    Adjusted R-squared:  0.714
F-statistic:  39.7 on 2 and 29 DF,  p-value: 4.979e-09

$emmeans
 cyl emmean    SE df lower.CL upper.CL
 4     26.7 0.972 29     24.7     28.7
 6     19.7 1.220 29     17.3     22.2
 8     15.1 0.861 29     13.3     16.9

Confidence level used: 0.95

$contrasts
 contrast    estimate   SE df t.ratio p.value
 cyl4 - cyl6     6.92 1.56 29   4.441  0.0003
```

```
 cyl4 - cyl8    11.56 1.30 29   8.905  <.0001
 cyl6 - cyl8     4.64 1.49 29   3.112  0.0112

P value adjustment: tukey method for comparing a family of 3 estimates
```

The output tells us that the mean `mpg` for the 4-cylinder cars is significantly different from the mean `mpg` for the 6-cylinder cars and the 8-cylinder cars. The mean `mpg` for the 6-cylinder cars is also significantly different from the mean `mpg` for the 8-cylinder cars. The estimated marginal means provide a way to compare the levels of the predictor variable and determine if there are significant differences between them.

What are estimated marginal means?

Estimated marginal means are the predicted means of the outcome variable for each level of the predictor variable. They are calculated by averaging the predicted values of the outcome variable for each level of the predictor variable, while holding all other variables constant. Estimated marginal means provide a way to compare the levels of the predictor variable and determine if there are significant differences between them.

9.5.1 What is the difference between estimated marginal means and actual means - how should I report them?

Estimated marginal means are the predicted means of the outcome variable for each level of the predictor variable. They are calculated by averaging the predicted values of the outcome variable for each level of the predictor variable, while holding all other variables constant. Actual means are the observed means of the outcome variable for each level of the predictor variable. The difference between estimated marginal means and actual means is that estimated marginal means are based on the regression model, while actual means are based on the observed data. This allows the estimated marginal means to take into account other variables in the model, for example.
When reporting the results of a regression analysis, it is common to report the estimated marginal means rather than the actual means. This is because the estimated marginal means take into account the effects of other variables in the model, while the actual means do not. Reporting the estimated marginal means can give a more accurate representation of the relationship between the predictor variable and the outcome variable.

9.5.2 What are Tukey corrected p-values?

Tukey corrected p-values are used to adjust for multiple comparisons in a pairwise comparison analysis. When conducting multiple comparisons between the levels of a predictor variable, there is an increased risk of making a Type I error (false positive) due to the number of comparisons being made. Tukey corrected p-values adjust for this increased risk. This means that the p-values are adjusted to account for the number of comparisons

being made, reducing the likelihood of making a false positive conclusion.

10 Multiple Regression and Heirarchical Regression

Tip

By the end of this session, you will be able to:

- Compare multiple regression to simple regression
- Describe the assumptions of multiple regression
- Consider sample size in regression
- Interpret the output of Multiple regression
- Conduct and interpret hierarchical multiple regression

10.1 What is multiple regression?

Multiple regression ia an extension of simple regression that allows us to predict an outcome variable (Y) based on multiple predictors (X1, X2 …).

$$Y = b_1X_1 + b_2X_2 + b_0$$

(The constant can be referred to in the equation as **c** or **b0**)

10.2 What are the assumptions of Multiple Regression?

They are primarily the same as simple regression, so look back at the assumptions of simple regression in that section. The additional assumption of no **multicollinearity** applies, due to having multiple predictors. The multicollinearity assumption means that predictors should not be highly correlated. If they were, it would be difficult to determine the unique contribution of each predictor to the model.

10.2.1 What is multicollinearity?

Multicollinearity means that predictors correlated highly with each other. This is not good because:

- It makes it difficult to determine the role of individual predictors
- Increases the error of the model (higher standard errors)
- Difficult to identify significant predictors
- Wider confidence interval

10.2.2 Testing multicollinearity

To test for multicollinearity, we can use the `mctest()` function in R. This function is part of the `mctest` package. It performs several tests for multicollinearity, including the Variance Inflation Factor (VIF) and the Condition Index (CI).

To run the test, you pass the regression model to the `mctest()` function. The function will then return the results of the tests.

```
## use the mctest package
# install.packages('mctest')

library(mctest)

m1 <- lm(aggression_level ~ treatment_group + treatment_duration + trust_score, data=regressi

mctest(m1)                                                                    ②
```

① In this code, we are running a regression model with three predictors (treatment group, treatment duration, and trust score) and the outcome variable aggression level.
② We then pass the regression model to the `mctest()` function to test for multicollinearity.

```
Call:
omcdiag(mod = mod, Inter = TRUE, detr = detr, red = red, conf = conf,
    theil = theil, cn = cn)


Overall Multicollinearity Diagnostics

                       MC Results detection
Determinant |X'X|:         0.9229         0
```

```
Farrar Chi-Square:         7.7960         0
Red Indicator:             0.1547         0
Sum of Lambda Inverse:     3.1728         0
Theil's Method:           -0.8800         0
Condition Number:         13.6549         0

1 --> COLLINEARITY is detected by the test
0 --> COLLINEARITY is not detected by the test
```

The `mctest()` function also takes an additional argument, `type`, which specifies whether you want, the main tests, each individual predictor, or both. The default is `type = "main"`, which will run the main tests.

10.3 Sample size for multiple regression

As the number of predictors increases, the sample size needed to run a multiple regression analysis also increases. A common rule of thumb is to have at least 10-15 participants per predictor. However, this is a loose rule, and the actual number of participants needed will depend on the complexity of the model and the effect size. Always run a power analysis to determine the sample size needed for your study.

10.4 Approaches to multiple regression: All predictors at once

There are different ways we could run a multiple regression analysis depending on our research question and how we conceptualise the relationship between the predictors and the outcome. One way is to include all predictors at the same time. This is useful when we want to know the combined predictive power of all the predictors at the same time.

> Research question: Do a client's treatment duration and treatment group predict aggression level?

```
model1 <- lm(data = regression_data, aggression_level ~ treatment_duration + treatment_group)
```

① Here we are including all of the predictors at the same time. Note that we are using a plus sign + between each predictor - this means that no interactions will be tested.

10.4.1 Using categorical predictors in R

- Treatment group is a categorical (also called "nominal" or "factor") variable
- No special "dummy coding" is required in R to use categorical predictors in regression
- R will use the first group as the reference category and test whether being in another group shows a significant difference
- R chooses the reference group based on numerical value or alphabetical order
- If you want you can change the reference category or "force" it using the relevel function:

```
regression_data$treatment_group <- relevel(regression_data$treatment_group, ref = "therapy1")
```

More information in categorical predictors in section @ref(catreg)

10.4.2 Reviewing the output

The output from this approach will look the same as simple regression, except there will be an additional row for each predictor in the model in the coefficients table.

```
summary(model1)
```

```
Call:
lm(formula = aggression_level ~ treatment_duration + treatment_group,
    data = regression_data)

Residuals:
    Min      1Q  Median      3Q     Max
-2.9468 -1.1104  0.0205  0.9621  3.4481

Coefficients:
                        Estimate Std. Error t value Pr(>|t|)
(Intercept)             11.58713    0.77331  14.984  < 2e-16 ***
treatment_duration      -0.66024    0.07119  -9.274 4.96e-15 ***
treatment_grouptherapy2  0.85032    0.30449   2.793   0.0063 **
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 1.5 on 97 degrees of freedom
Multiple R-squared:  0.5206,    Adjusted R-squared:  0.5107
F-statistic: 52.67 on 2 and 97 DF,  p-value: 3.267e-16
```

As a reminder of what the output tells us:

- Multiple R^2 = Total variance in outcome that is explained by the model
- p-value = Statistical significance of the model
- Coefficients = Contribution of each predictor to the model
  - Pr = Significance of the individual predictor
  - Estimate = Change in the outcome level that occurs when the predictor increases by 1 unit of measurement (for continuous predictors) or the difference in the mean outcome level between the predictor and the reference category (for categorical predictors)

10.4.3 All predictors at once (testing interactions)

It might be the case that we are interested in whether the predictors interact with each other to predict the outcome. For example, we might want to know if the effect of treatment duration on aggression level depends on the treatment group.

Research questions:

- Do a client's treatment duration and treatment group predict aggression level
- Do the predictors interact?

```
model2 <- lm(data = regression_data, aggression_level ~ treatment_duration * treatment_group
```

① Here we are including all of the predictors at the same time. Note that we are using an asterisk * between each predictor. This means that interactions will be tested.

Reviewing the output

```
summary(model2) %>% coefficients
```

```
                                            Estimate Std. Error    t value
(Intercept)                                12.3529190  1.1006127 11.2236751
treatment_duration                         -0.7334435  0.1033086 -7.0995381
treatment_grouptherapy2                    -0.5615517  1.4753596 -0.3806202
treatment_duration:treatment_grouptherapy2  0.1394649  0.1425977  0.9780305
                                               Pr(>|t|)
(Intercept)                                3.599000e-19
treatment_duration                         2.166226e-10
treatment_grouptherapy2                    7.043260e-01
treatment_duration:treatment_grouptherapy2 3.305175e-01
```

In the output, we get additional information in the coefficients table about the interaction between variables. We can see from the output that none of the interactions are significant. This being the case, we would not interpret the main effects of the predictors in the model, and instead, we would interpret the results based on the interaction between the predictors.

There is more information on interactions in regression in the section on moderation.

10.4.4 Hierarchical multiple regression: Theory driven "blocks" of variables

When we have multiple predictors, it might be the case that we have previous research or theory to guide how we run the analysis. For example, we might know that treatment duration and therapy group are likely to predict the outcome, based on the results of previous studies. However, we might also want to check whether client's level of trust in the clinician has any **additional** impact on our ability to predict the outcome (aggression level).

This being the case, we could run a hierarchical multiple regression analysis. This is where we run the regression in "blocks" (groups) of variables. We start with a baseline model (Model 0) and then add additional predictors in each subsequent model.

How we group our predictors will depend on our research question and the theory behind the relationship between the predictors and the outcome. As such, we should have a clear rationale for how we group our predictors.

To do this, we run three regression models:

- Model 0: the constant (intercept)
- Model 1: treatment duration and therapy group
- Model 2: treatment duration and therapy group and trust score

We then compare the two regression models to see if:

- Model 1 is better than Model 0 (the intercept)
- Model 2 is better than Model 1

The intercept (null) model is the simplest model we can run. It is a model that only includes the constant (intercept) and no predictors. This model is used as a baseline to compare the other models to. Therefore, we don't really interpret the null model in isolation, rather, we examine whether another model (with predictors) is more useful in predicting the outcome than the null model.

Hierarchical multiple regression: Running and comparing 2 models

```r
## run regression using the same method as above
model0 <- lm(data = regression_data, aggression_level ~ 1)                    ①
model1 <- lm(data = regression_data, aggression_level ~ treatment_duration + treatment_group)
model2 <- lm(data = regression_data, aggression_level ~ treatment_duration + treatment_group


## use the aov() command to compare the models
anova(model0,model1,model2)                                                   ④
```

① Here we are running the null model (Model 0) with only the intercept
② Here we are running Model 1 with treatment duration and treatment group
③ Here we are running Model 2 with treatment duration, treatment group, and trust score
④ We then use the `anova()` command to compare the models

```
Analysis of Variance Table

Model 1: aggression_level ~ 1
Model 2: aggression_level ~ treatment_duration + treatment_group
Model 3: aggression_level ~ treatment_duration + treatment_group + trust_score
  Res.Df    RSS Df Sum of Sq       F    Pr(>F)
1     99 455.27
2     97 218.26  2   237.013 52.2195 4.507e-16 ***
3     96 217.86  1     0.399  0.1757     0.676
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

When we run the `anova()` command, we get an output that tells us whether the additional predictors in each model significantly change the R^2 value of the model (reducing the residual variance of the model). We can see from the output that Model 1 is significantly better than the null model (Model 0), and Model 2 is not significantly better than Model 1.

10.4.5 Model performance metrics

When we run a hierarchical multiple regression analysis, we are interested in whether the additional predictors in each model significantly change the R^2 value of the model. However, adding additional predictors to the model will always increase the R^2 value of the model somewhat, even if the predictors are not useful This is because the R^2 value is based on the amount of variance in the outcome that is explained by the predictors in the model.

Because of this, we should also consider other metrics of model performance, such as the AIC (Akaike Information Criterion) and BIC (Bayesian Information Criterion). These metrics take

into account the number of predictors in the model and penalise the model for having too many predictors. A lower AIC or BIC value indicates a better model. The `AIC()` and `BIC()` functions in R can be used to calculate these metrics.

```
AIC(model0,model1,model2)                                    ①

BIC(model0,model1,model2)                                    ②
```

① Here we are using the `AIC()` function to calculate the AIC value for each model
② Here we are using the `BIC()` function to calculate the BIC value for each model

```
       df      AIC
model0  2 439.3603
model1  4 369.8394
model2  5 371.6566
       df      BIC
model0  2 444.5707
model1  4 380.2601
model2  5 384.6825
```

We can see from the output that Model 1 has the lowest AIC and BIC values, indicating that it is the best model of the three. We phrase this has having the best fit to the data, given the number of predictors in the model. Although, in this example, model 2 is not significantly better than model 1, it could be the case that you have a different research question or theory where the `anova()` test would show that the model with the additional predictors is significantly better. You could then use the AIC and BIC values to determine which model is actually the best fit to the data.

Remember that the accuracy of your models will depend on the quality of your data and the assumptions of regression being met. How you interpret the results will depend on your research question and the theory behind the relationship between the predictors and the outcome. The analysis cannot prove causation, only association. It is up to you to design your study and analysis to best answer your research question.

11 Creating plots with ggplot2 in R

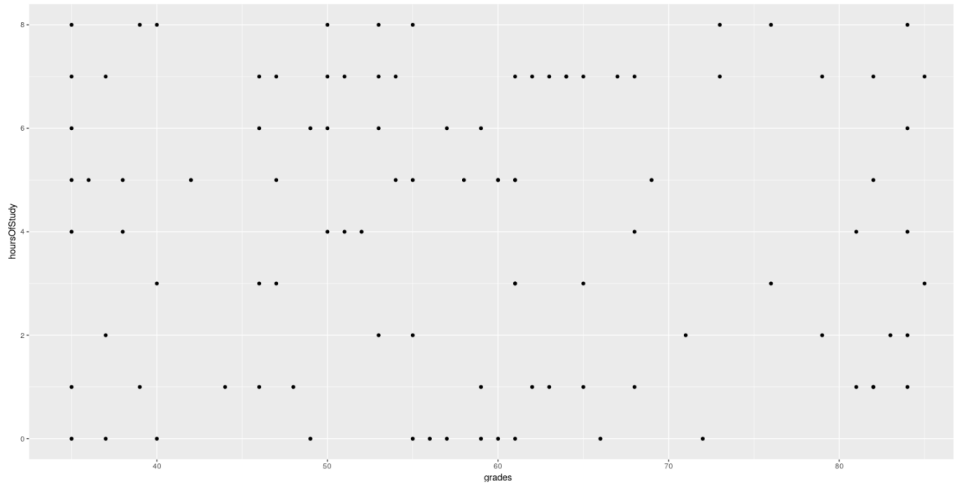
At the end of this chapter, you will be able to:

- Describe the ggplot "grammar of visualisation": coordinates and geoms
- Write a graph function to display multiple variables on a plot
- Amend the titles and legends of a plot
- Save plots in PDF or image formats

11.1 The "grammar of visualisation" with ggplot

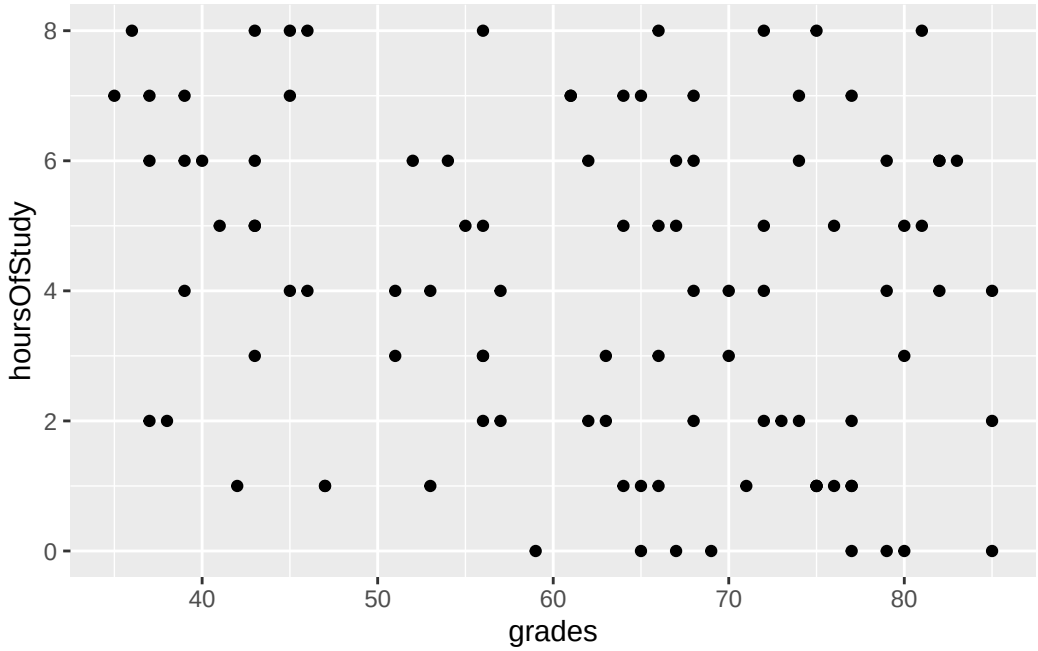
With ggplot, graphs are made up of 3 components: - A dataset - A coordinate system (e.g., the X and Y axes) - Visual marks to represent data **(geoms)**

grades	hoursOfStudy
35	6
35	8
82	7
56	0
61	5
72	0
38	4
82	5
39	1



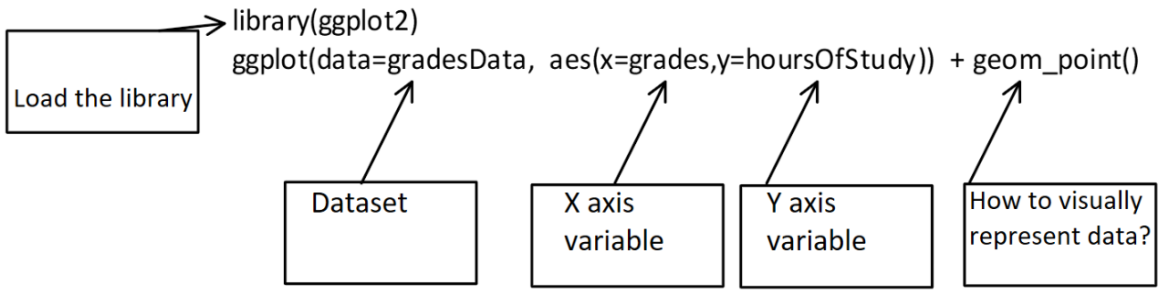
In the above example, the dataset is the *studentData* that we used previously. The *grades* variable is mapped to the X axis. The *hoursOfStudy* variable is mapped to the Y axis. The graph is created using the following code:

```
library(ggplot2)

ggplot(data=studentData, aes(x=grades,y=hoursOfStudy)) +                    ①
  geom_point()                                                              ②
```



In this code:

1. We specify the dataset and the variables for the X and Y axes.
2. We specify the **geom** that will represent the data points visually (in this case, each datum is a point).



> **!** Remember the + operator
>
> - The layers of a ggplot are separated using the + operator. That is, we first specify the dataset and the variables for the X and Y axes, and then we add a +, then the `geom` that will represent the data points visually.

11.2 Adding more variables to a plot

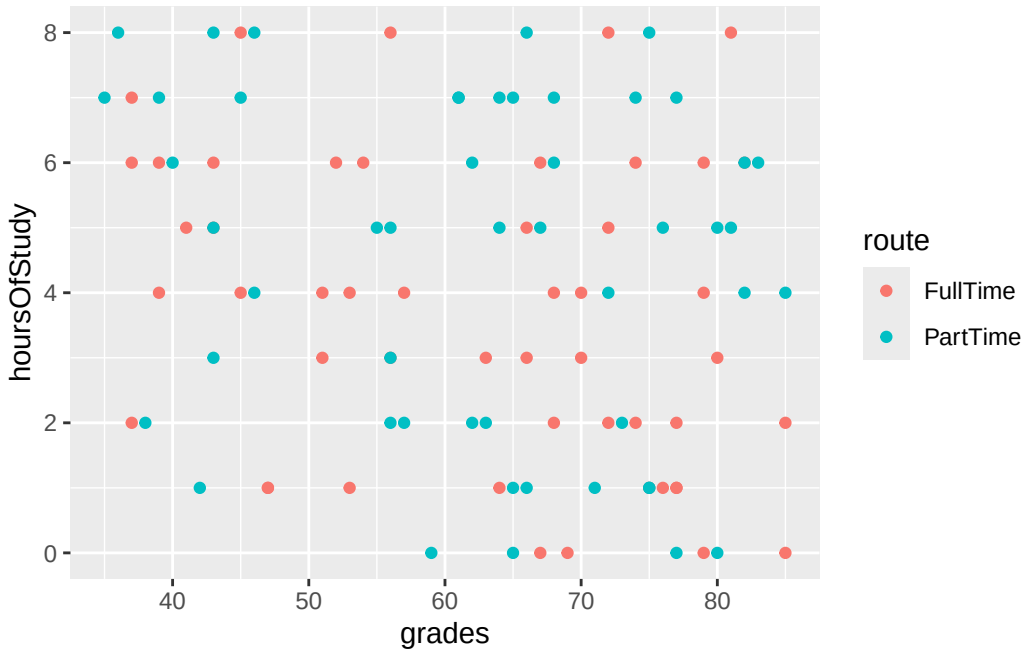
One of the things that makes ggplot so powerful is that you can add more variables to a plot. You can map these variables to different aesthetic attributes of the plot, such as colour, shape, or size.

This can be done either in the `ggplot()` function or in the `geom_point()` function.

11.2.1 Example: assigning a variable to colour (2 different approaches)

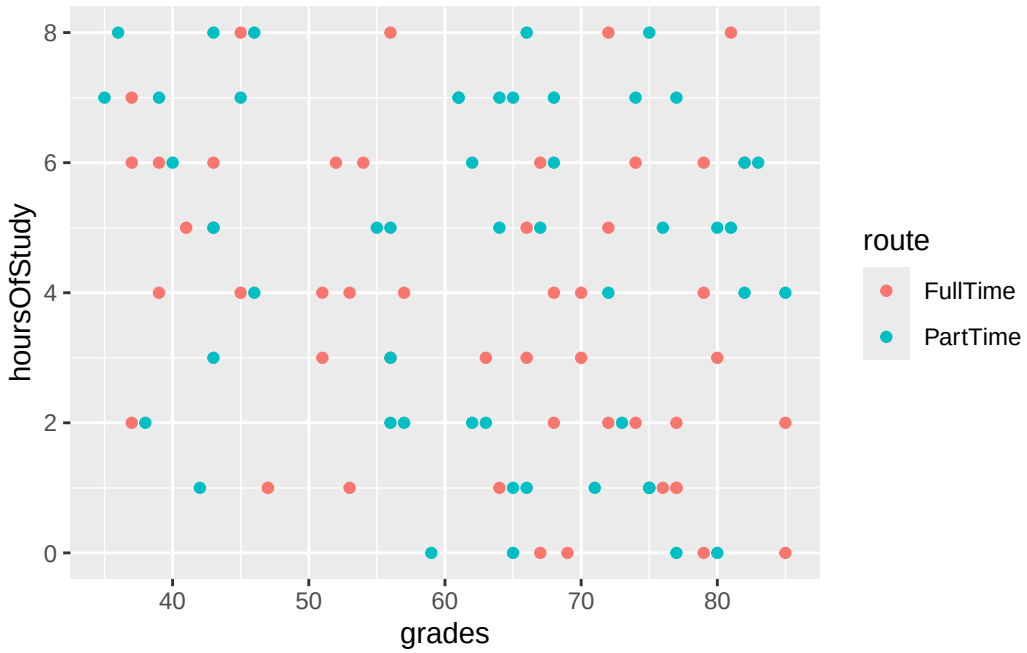
```
ggplot(data=studentData, aes(x=grades,y=hoursOfStudy, color = route)) +     ①
  geom_point()                                                              ②
```

① The `color` aesthetic is mapped to the `route` variable in the main `ggplot()` function.
② The `geom_point()` function is used to represent the data points visually and it takes its colour from the main `ggplot()` function.




```
ggplot(data=studentData, aes(x=grades,y=hoursOfStudy)) +                    ①
  geom_point(aes(color = route))                                            ②
```

① The `color` aesthetic is not mentioned in the main `ggplot()` function.
② The `color` aesthetic is mapped to the `route` variable in the `geom_point()` function.

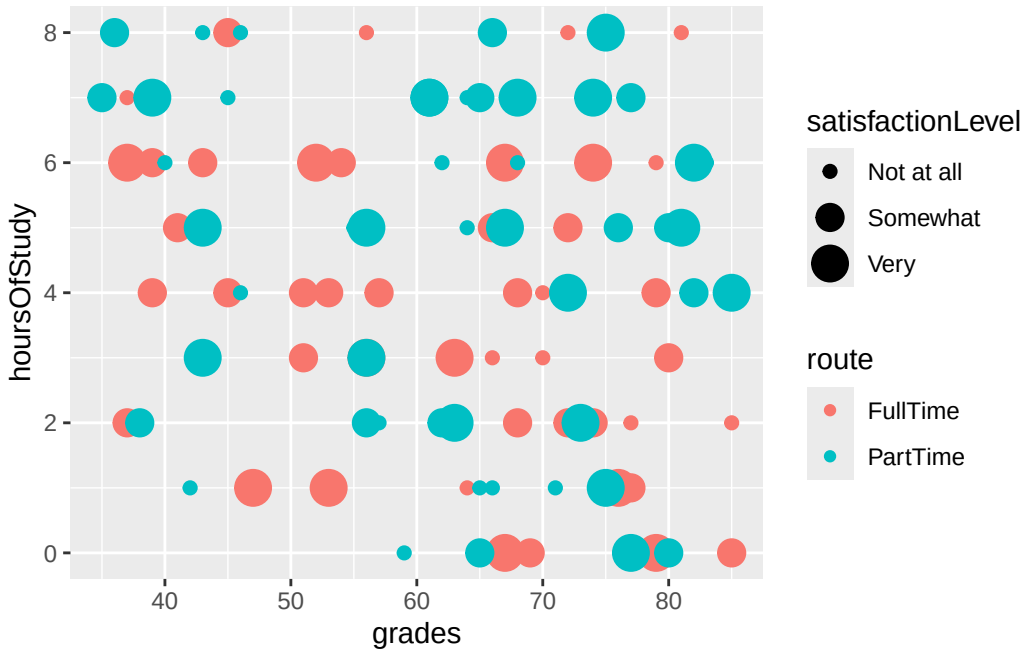


11.2.2 Example: assigning a variable to size

```
ggplot(data=studentData, aes(x=grades,y=hoursOfStudy, size = satisfactionLevel, colour = rou
  geom_point()                                                                          ②
```

① The `size` aesthetic is mapped to the `satisfactionLevel` variable in the main `ggplot()` function.

② The `geom_point()` function is used to represent the data points visually and it takes its size from the main `ggplot()` function.

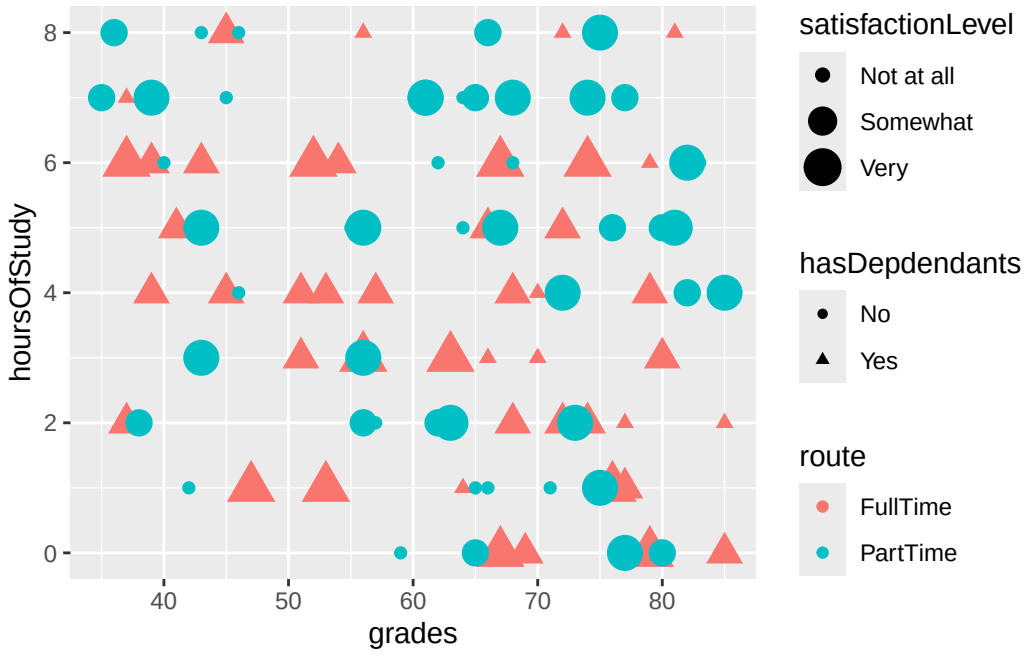


11.2.3 Example: assigning a variable to shape

```
ggplot(data=studentData, aes(x=grades,y=hoursOfStudy, size = satisfactionLevel, colour = rou
  geom_point()                                                                          ②
```

① The `shape` aesthetic is mapped to the `hasDepdendants` variable in the main `ggplot()` function.

② The `geom_point()` function is used to represent the data points visually and it takes its shape from the main `ggplot()` function.



11.3 Different types of geoms: bar charts

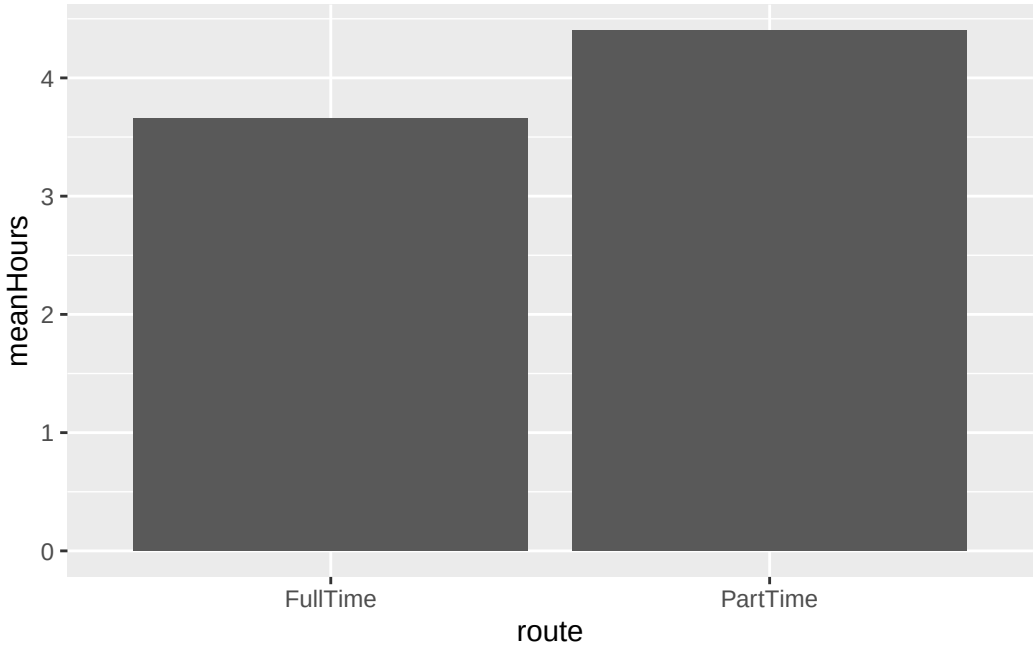
The `geom_point()` function is just one of many geoms that you can use in ggplot. Another common geom is `geom_col()`, which is used to create bar charts.

When creating a bar chart there are 2 approaches:

1. summarise the data before plotting
2. let ggplot do the summarising for you

11.3.1 Example: summarising the data before plotting

```
library(dplyr)

studentData %>%
  group_by(route) %>%
  summarise(meanHours = mean(hoursOfStudy)) %>%
  ggplot(aes(x = route, y = meanHours)) +
  geom_col()
```

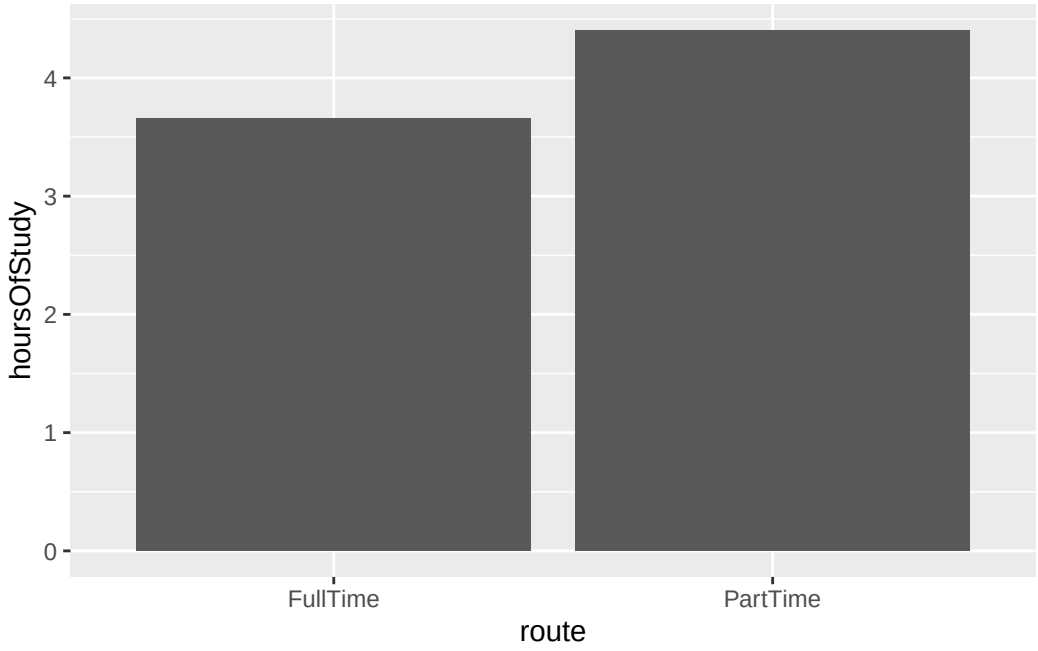


In the above code, we first use the `group_by()` and `summarise()` functions from the `dplyr` package to calculate the mean number of hours of study for each route (see previous lessons for this). We then use the `ggplot()` function to create a plot with the `route` variable on the X axis and the `meanHours` variable on the Y axis. Finally, we use the `geom_col()` function to create the bar chart.

> **!** Remember to refer to the summarised variable in the `ggplot()` function
>
> When you summarise the data, remember to refer to the summarised variable in the `ggplot()` function, not the original variable. In the above example, we use `meanHours` in the `ggplot()` function, not `hoursOfStudy`.
> If you find it easier, you can save the summarised data as a new object and then use this in the `ggplot()` function.

11.3.2 Example: letting ggplot do the summarising for you

```
ggplot(data=studentData, aes(x = route, y = hoursOfStudy)) +
  stat_summary(fun = mean, geom = "col")
```

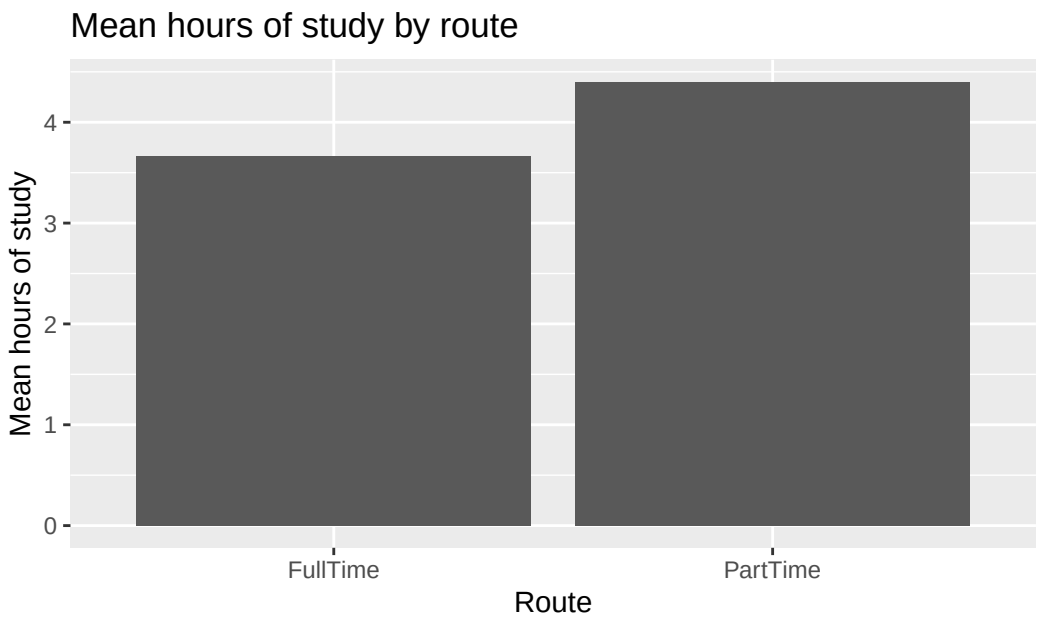


In the above code, we use the `stat_summary()` function to calculate the mean number of hours of study for each route. We specify that we want to calculate the mean using the `fun = mean` argument, and we specify that we want to create a bar chart using the `geom = "col"` argument.

11.4 Changing the titles and legends of a plot

You can change the titles and legends of a plot using the `labs()` function. This function allows you to change the title of the plot, the titles of the X and Y axes, and the legend title.

```
studentData %>%
  group_by(route) %>%
  summarise(meanHours = mean(hoursOfStudy)) %>%
  ggplot(aes(x = route, y = meanHours)) +
  geom_col() +
  labs(title = "Mean hours of study by route",
       x = "Route",
       y = "Mean hours of study",
       caption = "Source: Student data")
```

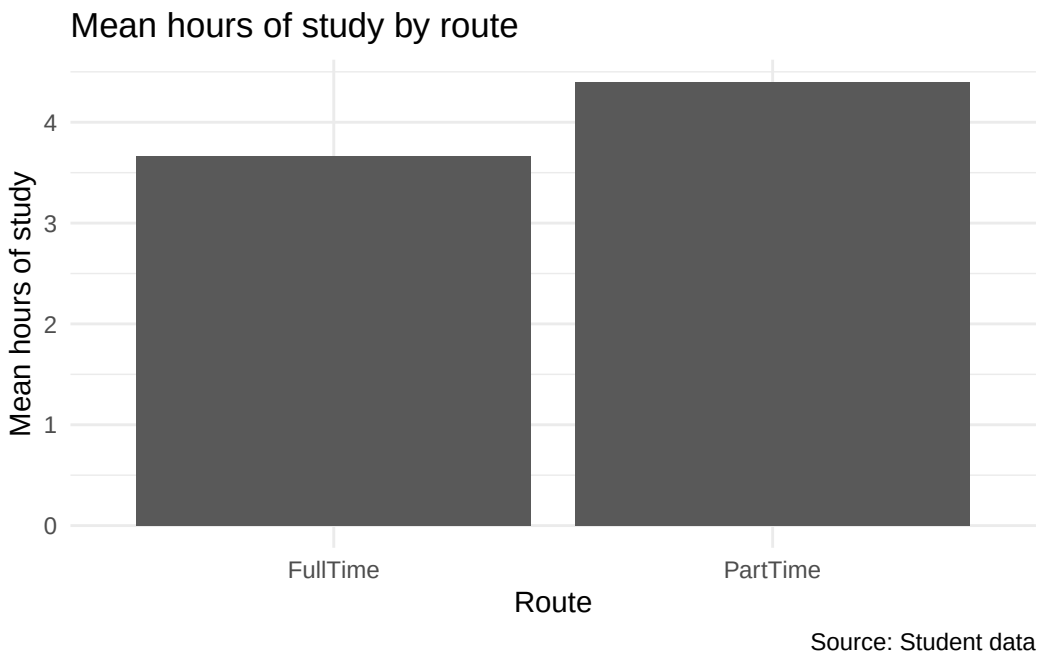


In the above code, we use the `labs()` function to change the title of the plot to "Mean hours of study by route", the title of the X axis to "Route", the title of the Y axis to "Mean hours of study", and the caption to "Source: Student data".

11.5 Themes in ggplot

You can change all of the colours and other appearance elements in ggplot. However, it can be easier to change the appearance of a plot using themes. Themes allow you to change the background colour, the colour of the text, the size of the text, and other aspects of the appearance of the plot.

```
studentData %>%
  group_by(route) %>%
  summarise(meanHours = mean(hoursOfStudy)) %>%
  ggplot(aes(x = route, y = meanHours)) +
  geom_col() +
  labs(title = "Mean hours of study by route",
       x = "Route",
       y = "Mean hours of study",
       caption = "Source: Student data") +
  theme_minimal()
```



In the above code, we use the `theme_minimal()` function to change the appearance of the plot to a minimal theme. There are many other themes that you can use, such as `theme_light()`, `theme_dark()`, and `theme_bw()`.

11.6 Saving plots in ggplot

The best way to save plots (and get reliable results) is to save them using the `ggsave()` function. This function allows you to save plots in a variety of formats, such as PDF, PNG, and JPEG. For publication-quality plots, it is best to save them in PDF format, because this format is scalable and resolution independent.

```
studentData %>%
  group_by(route) %>%
  summarise(meanHours = mean(hoursOfStudy)) %>%
  ggplot(aes(x = route, y = meanHours)) +
  geom_col() +
  labs(title = "Mean hours of study by route",
       x = "Route",
       y = "Mean hours of study",
       caption = "Source: Student data") +
  theme_minimal()
```

```r
ggsave("studentData_plot.pdf", width = 6, height = 4, dpi = 300)
```

In the above code, we use the `ggsave()` function to save the plot as a PDF file called "studentData_plot.pdf". We specify the width of the plot as 6 inches, the height of the plot as 4 inches, and the resolution as 300 dots per inch (dpi). dpi is not actually used for PDF files, but it is useful for other file formats, such as PNG and eps.

In terms of specifying the width and height of the plot, it can take some fiddling around to get the right dimensions. You can specify the width and height in pixels, but it is generally better to specify them in inches, because this is a more standard unit of measurement for plots. I recommend thinking in terms of the aspect ratio of the plot, rather than the exact dimensions. For example, if you want a wide plot, you might specify the width as 6 inches and the height as 4 inches. If you want a square plot, you might specify the width and height as 4 inches each. If you want a tall plot, you might specify the width as 4 inches and the height as 6 inches.

Once the aspect ratio feels right, you can adjust the width and height to get the exact right dimensions for your plot.

> **!** Plots will save into the working directory
>
> When you save a plot using the `ggsave()` function, the plot will be saved into the working directory. You can specify a different directory by specifying the full path to the file, such as "C:/Users/username/Documents/studentData_plot.pdf".
> It's easier just to set the working directory to the folder where you want to save the plots!

11.7 Summary

In this chapter, you have learned how to create plots with ggplot in R. You have learned about the "grammar of visualisation" with ggplot, including the dataset, the coordinate system, and the visual marks that represent the data. You have learned how to write a graph function to display multiple variables on a plot, how to amend the titles and legends of a plot, and how to save plots in PDF or image formats.