Tropical Depressions, Storms, and Typhoons Oh My!

Solutions

Ernest Guevarra

18 November 2024

Contents

List of Figures

List of Tables

This document provides detailed solutions to the tasks set in the **Tropical Depressions, Storms, and Typhoons Oh My!** exercise set of the **Open and Reproducible Science in R** module of the **MSc in International Health and Tropical Medicine**.

1 Introduction to the exercise

1.1 Instructions for the assignment

The following tasks have been setup to help students get familiar with the basics of R and performing basic operations and functions in R.

The students are expected to go through the tasks and appropriately write R code/script to fulfill the tasks and/or to answer the question/s being asked within the tasks. R code/script should be written inside a single R file named `cyclones.R` and saved in the project's root directory.

1.2 The dataset

Oceans and seas significantly impact continental weather, with evaporation from the sea surface driving cloud formation and precipitation. Tropical cyclones, warm-core low-pressure systems, form over warm oceans where temperatures exceed 26°C, precipitated by the release of latent heat from condensation. These cyclones, known by various names depending on the region, have organised circulations and develop primarily in tropical and subtropical waters, except in regions with cooler sea surface temperatures and high vertical wind shears. They reach peak intensity over warm tropical waters and weaken upon landfall, often causing extensive damage before dissipating.

The Philippines frequently experiences tropical cyclones because of its geographical position. These cyclones typically bring heavy rainfall, leading to widespread flooding, as well as strong winds that cause significant damage to human life, crops, and property. Data on cyclones are collected and curated by the Philippine Atmospheric, Geophysical, and Astronomical Services Administration (PAGASA).

A dataset in comma-separated value (CSV) format called `cyclones.csv` can be found inside the `data` folder of this repository. This dataset contains records of every cyclone that entered the Philippine area of responsibility from 2017 to 2021. The dataset has the following variables/fields (see Table 1):

Table 1: Cyclones dataset field names and description

Variable/Field	Description
year	Year
category_code	Tropical cyclone category code
category_name	Tropical cyclone category name
name	Name given to the tropical cyclone by Philippine authorities
rsmc_name	Name given to the tropical cyclone by Philippine authorities
start	Date and time at which cyclone enters Philippine waters
end	Date and time at which cyclone leaves Philippine waters
pressure	Maximum central pressure in hPa

speed	Maximum sustained wind speed in km/h

2 Task 1: Read the cyclones dataset into R

There are many ways to read a dataset into R and the method used will depend on the file type/format of the dataset. The dataset for this exercise is a **comma-separated value** or **CSV** file type/format. A CSV file is a text file that stores data in a table structure, with each value separated by a comma.

The `read.table()` functions are the base R functions that can be used to read CSV files into R. Of the `read.table()` family of functions, the two functions that are most relevant to reading the cyclones CSV file are the `read.table()` and the `read.csv()` function.

2.1 Using the `read.table()` function

The `read.table()` function can be used to read the cyclones CSV file as follows:

```
read.table(
  file = "data/cyclones.csv",
  header = TRUE,
  sep = ","
)
```

which results in (showing first 10 rows of data):

```
   year category_code          category_name    name rsmc_name
1  2017            TD    Tropical Depression  Auring      <NA>
2  2017            TD    Tropical Depression  Bising      <NA>
3  2017            TD    Tropical Depression Crising      <NA>
4  2017            TS         Tropical Storm   Dante     Muifa
5  2017           STS  Severe Tropical Storm   Emong  Nanmadol
6  2017            TD    Tropical Depression  Fabian      Roke
7  2017            TY                Typhoon   Gorio     Nesat
8  2017            TS         Tropical Storm Huaning   Haitang
9  2017           STS  Severe Tropical Storm   Isang      Hato
10 2017            TS         Tropical Storm  Jolina    Pakhar
                 start                 end pressure speed
1  2017-01-07 08:00:00 2017-01-09 08:00:00     1000    55
2  2017-02-03 14:00:00 2017-02-06 14:00:00     1004    45
3  2017-04-14 14:00:00 2017-04-15 20:00:00     1004    45
4  2017-04-26 08:00:00 2017-04-27 20:00:00      998    65
5  2017-07-02 02:00:00 2017-07-03 02:00:00      987    95
6  2017-07-22 02:00:00 2017-07-22 14:00:00     1000    55
7  2017-07-25 14:00:00 2017-07-30 02:00:00      957   145
```

```
8   2017-07-30 02:00:00 2017-07-31 05:00:00       990     85
9   2017-08-20 08:00:00 2017-08-22 14:00:00       977    110
10  2017-08-24 14:00:00 2017-08-26 14:00:00       993     80
```

In the `read.table()` function, we used 3 arguments that ensures that the function is able to read a CSV file properly.

```r
read.table(
  file = "data/cyclones.csv",    # ①
  header = TRUE,                 # ②
  sep = ","                      # ③
)
```

① The `file` argument should be supplied with the file path to the file which the data are to be read from. For this exercise, the dataset is found in the **data** folder within our project so we specify “data/cyclones.csv” to the `file` argument. The specification for this argument should be enclosed in `""`.

② The `header` argument requires a logical value (`TRUE` or `FALSE`). The value supplied to this argument should indicate whether the file to be read contains the names of the variables as its first line. Since the `cyclones.csv` files had variable names as its first line, we set this to `TRUE`.

③ The `sep` argument should be supplied with the field separator character which is the character used to separate every value in each line of the file. Since the cyclones dataset is a CSV file, the `sep` argument should be set as `","`.

When the `read.table()` function is used to read a CSV file, the three arguments described above should always be specified in order for R to read the CSV file properly.

2.2 Using the `read.csv()` function

The `read.csv()` function is a member of the `read.table()` family of functions. The `read.csv()` function is a specialised function built on the `read.table()` function. By default, the `read.csv()` function sets `header` argument to `TRUE` and the `sep` argument to `","`. Hence, we use the `read.csv()` function as follows:

```r
read.csv(file = "data/cyclones.csv")
```

which returns the following output (showing first 10 rows of data):

```
   year category_code         category_name    name rsmc_name
1  2017            TD   Tropical Depression  Auring      <NA>
2  2017            TD   Tropical Depression  Bising      <NA>
3  2017            TD   Tropical Depression Crising      <NA>
4  2017            TS        Tropical Storm   Dante     Muifa
5  2017           STS Severe Tropical Storm   Emong  Nanmadol
6  2017            TD   Tropical Depression  Fabian      Roke
```

```
7  2017              TY                 Typhoon   Gorio     Nesat
8  2017              TS          Tropical Storm Huaning   Haitang
9  2017             STS Severe Tropical Storm   Isang      Hato
10 2017              TS          Tropical Storm  Jolina    Pakhar
                 start                 end pressure speed
1  2017-01-07 08:00:00 2017-01-09 08:00:00     1000    55
2  2017-02-03 14:00:00 2017-02-06 14:00:00     1004    45
3  2017-04-14 14:00:00 2017-04-15 20:00:00     1004    45
4  2017-04-26 08:00:00 2017-04-27 20:00:00      998    65
5  2017-07-02 02:00:00 2017-07-03 02:00:00      987    95
6  2017-07-22 02:00:00 2017-07-22 14:00:00     1000    55
7  2017-07-25 14:00:00 2017-07-30 02:00:00      957   145
8  2017-07-30 02:00:00 2017-07-31 05:00:00      990    85
9  2017-08-20 08:00:00 2017-08-22 14:00:00      977   110
10 2017-08-24 14:00:00 2017-08-26 14:00:00      993    80
```

The output of using the `read.csv()` function is exactly the same as the `read.table()` function.

In general, when dealing with CSV files, the `read.csv()` function is the most convenient and straightforward function to use.

2.3 Creating a `cyclones` object

So that we can use the cyclones data for the next steps of our task, we create an object called `cyclones` and assign the output of either the `read.table()` or the `read.csv()` function to this object as shown below:

```r
cyclones <- read.csv(file = "data/cyclones.csv")
```

When we inspect the `cyclones` object, we get (showing first 10 rows of data):

```r
cyclones
```

```
   year category_code         category_name    name rsmc_name
1  2017            TD    Tropical Depression  Auring      <NA>
2  2017            TD    Tropical Depression  Bising      <NA>
3  2017            TD    Tropical Depression Crising      <NA>
4  2017            TS          Tropical Storm   Dante     Muifa
5  2017           STS Severe Tropical Storm   Emong  Nanmadol
6  2017            TD    Tropical Depression  Fabian      Roke
7  2017            TY                 Typhoon   Gorio     Nesat
8  2017            TS          Tropical Storm Huaning   Haitang
9  2017           STS Severe Tropical Storm   Isang      Hato
10 2017            TS          Tropical Storm  Jolina    Pakhar
                 start                 end pressure speed
```

```
1  2017-01-07 08:00:00 2017-01-09 08:00:00     1000    55
2  2017-02-03 14:00:00 2017-02-06 14:00:00     1004    45
3  2017-04-14 14:00:00 2017-04-15 20:00:00     1004    45
4  2017-04-26 08:00:00 2017-04-27 20:00:00      998    65
5  2017-07-02 02:00:00 2017-07-03 02:00:00      987    95
6  2017-07-22 02:00:00 2017-07-22 14:00:00     1000    55
7  2017-07-25 14:00:00 2017-07-30 02:00:00      957   145
8  2017-07-30 02:00:00 2017-07-31 05:00:00      990    85
9  2017-08-20 08:00:00 2017-08-22 14:00:00      977   110
10 2017-08-24 14:00:00 2017-08-26 14:00:00      993    80
```

3 Task 2: Describing the data structure

3.1 Shape of the data

The shape of the data usually describes the structure. A "rectangular" dataset is probably the most familiar shape/structure for all of us as this is a tabular structure (rows and columns). In R, a `data.frame` is the most basic rectangular data structure. A "linear/line" dataset shape can either be a vector dataset or a list dataset in R. These "shapes" of data provide us with ideas/clues as to how to interact and use them in R.

The `class()` function gives us ideas/clues as to what "shape" a dataset can be. We can apply the `class()` function to the `cyclones` object as follows:

```
class(cyclones)
```

which gives the following output:

```
[1] "data.frame"
```

The `cyclones` dataset is a `data.frame` object which means that it is "rectangular" or tabular in shape.

3.2 Number of records in the data

Often we need to know how big our data is which is basically about how many records or rows of data is in our data. For this, we can use the `nrow()` function to get how many rows of data there are in a dataset as shown below:

```
nrow(cyclones)
```

which gives the following output:

```
[1] 101
```

The `nrow()` function tells us that there are 101 rows of data in the `cyclones` dataset.

3.3 Names of variables of the data

When working with data, it is useful to know the names of the variables of the data. In R, we can use the `names()` function to get the variable names of a dataset as follows:

```r
names(cyclones)
```

which gives the following output:

```
[1] "year"          "category_code" "category_name" "name"
[5] "rsmc_name"     "start"         "end"           "pressure"
[9] "speed"
```

The `names()` function tells us that the `cyclones` dataset has the following variables: year, category_code, category_name, name, rsmc_name, start, end, pressure, speed.

3.4 Number of variables (columns) in the data

We sometimes also want to know how many variables there are in a dataset. We can use the `ncol()` function to now the number of variables (or columns) of a dataset as follows:

```r
ncol(cyclones)
```

which gives the following output:

```
[1] 9
```

Another approach to getting the number of variables of a dataset is by counting the names of the variables. This can be done as follows:

```r
length(names(cyclones))
```

which gives the following output:

```
[1] 9
```

We get the same output from both approaches. There are 9 variables in the `cyclones` dataset.

3.5 Guide to indexing in R

In order to be able to perform various analysis and apply different kinds of statistics on a dataset, we need to be able access specific values within it. There are multiple ways of doing that in R. In this solution set, we show how to use the `$` operator to access the columns of values for variables combined with the use of the indexing operator `[]` to filter or subset specific rows and/or columns of data based on what we need for specific analysis or computation. The approaches we will discuss here are for `data.frame` objects which have a rectangular or tabular shape. This shape of a `data.frame` object is an important idea to have to get a good understanding of how indexing works in R.

A `data.frame` object given its rectangular or tabular shape has rows (values of which go from left to right) and has columns (values of which go from top to bottom). Hence, you can think of a `data.frame` as having some sort of coordinate system with positions of various values in the dataset being defined by its row and column within the rectangle/table. To further illustrate this, let us work with a smaller dataset shown below:

Table 2: Example dataset for indexing methods

student	number	colour
Tumi	1	red
Seiza	1	red
Alaa	2	blue
Ibrahim	3	blue
Simon	3	blue

The dataset is a make believe dataset which has 5 records of 5 IHTM students and their favourite number between 1, 2, and 3 and their favourite colour between red and blue.

Because there are just 5 records in the dataset, it is very easy for us to answer the following questions:

Note 1: Which student/s has number 2 as their favourite number?

To answer this question, we can just look at the column of data labelled as `number` and then go down the values of that variable and look for values that are equal to 2 (Figure 1 step 1). Once we spot a value of 2 in `number` column, we can then look to the left towards the `student` column on the same row as where there is a value of 2 in the `number` column to find the name of the student whose favourite number from 1 to 3 is 2 (Figure 1 step2). We find that the student is named **Alaa**. Then, back to the `number` column at the point where we found a number 2, we continue looking down until the end of the dataset to see if there are other values for `number` that are equal to 2. We find that there are no other values of `number` that are 2. So, the answer to the question is that the student named **Alaa** is the one who has a favourite number of 2.

student	number	colour
Tumi	1	red
Seiza	1	red
Alaa	2	blue
Ibrahim	3	blue
Simon	3	blue

Figure 1: Manual indexing example 1

Note 2: What is the favourite colour between red and blue of the student named Simon

To answer this question, we can look at the column of data labelled `student` and then look from top to bottom at these values until we find a student name that is equal to Simon (Figure 2 step 1). Once we find that, we look to the right towards the column named `colour` on the same row as where the `student` column value is Simon (Figure 2 step 2). We see that the value `colour` for the `student` called Simon is blue. So, the answer to the question is blue.

student	number	colour
Tumi	1	red
Seiza	1	red
Alaa	2	blue
Ibrahim	3	blue
Simon	3	blue

Figure 2: Manual indexing example 2

The manual process described above is similar to how the indexing in R happens. We provide code to R to index the rows and/or columns of a `data.frame` to arrive at the values that we need. The general syntax for this uses the indexing operator `[]` as follows:

```
object[row, column]
```

With this syntax, we can answer **question 1** above as follows:

```
student_data <- data.frame(
  student = c("Tumi", "Seiza", "Alaa", "Ibrahim", "Simon"),
  number = c(1, 1, 2, 3, 3),
  colour = c("red", "red", "blue", "blue", "blue")
)

student_data[student_data$number == 2, "student"]
```

which gives the following output:

```
[1] "Alaa"
```

For **question 2**, we can use the following code:

```
student_data[student_data$student == "Simon", "colour"]
```

which gives the following output:

```
[1] "blue"
```

We can apply the same approach to answer the following questions about the `cyclones` dataset.

Question 1: How many cyclones entered the Philippines in 2017?

```r
nrow(cyclones[cyclones$year == 2017, ])
```

```
[1] 22
```

There were **22** cyclones that entered the Philippines in 2017.

Question 2: What is the mean cyclone speed of the cyclones that hit the Philippines in 2019?

```r
mean(cyclones[cyclones$year == 2019, "speed"])
```

```
[1] 59.04762
```

There mean cyclone speed of the cyclones that hit the Philippines in 2019 was **59.047619 kph**.

Question 3: What is the name of the cyclone with the lowest pressure in 2020?

```r
cyclones2020 <- cyclones[cyclones$year == 2020, ]
cyclones2020[cyclones2020$pressure == min(cyclones2020$pressure), "name"]
```

```
[1] "Rolly"
```

Rolly was the name of the cyclones with the lowest pressure in 2020.

Question 4: How many cyclones have a speed of less than 100 kph and a pressure greater than 1000?

```r
nrow(cyclones[cyclones$speed < 100 & cyclones$pressure > 1000, ])
```

```
[1] 13
```

There were **13** cyclones with speed less than 100 and pressure greater than 1000 in the whole dataset.

3.6 Accessing the different values in a dataset

3.6.1 Using $ operator

A straightforward way to access variables in a dataset object is using the `$` operator. So, to access the `speed` values in the `cyclones` dataset, we use:

```
cyclones$speed
```

which gives the following output:

```
 [1]  55  45  45  65  95  55 145  85 110  80  65 130  85  45 110 185 105  90
[19]  75  65  80 120  35  35 105  60  40  65 105  40  50  30  40  30 110  40
[37] 110 105 115 115  60  80  30  30 105  30  40  25  45  30 105  55  40  35
[55]  95  25  65  75  75  65  70  55  95  80  85  40  25  70  45  65  30  65
[73]  85  95 105  45  55  45  25  65  90 120  50  45  85  30  45 120  35  40
[91]  30  80  50  45  55  65 115  35  55  30 105
```

3.6.2 Using the index [] operator

The other approach to access variables in a dataset object is using the index `[]` operator as earlier described. So, to access the `speed` values in the `cyclones` dataset, we use:

```
cyclones[ , "speed"]
```

which results in the following output:

```
 [1]  55  45  45  65  95  55 145  85 110  80  65 130  85  45 110 185 105  90
[19]  75  65  80 120  35  35 105  60  40  65 105  40  50  30  40  30 110  40
[37] 110 105 115 115  60  80  30  30 105  30  40  25  45  30 105  55  40  35
[55]  95  25  65  75  75  65  70  55  95  80  85  40  25  70  45  65  30  65
[73]  85  95 105  45  55  45  25  65  90 120  50  45  85  30  45 120  35  40
[91]  30  80  50  45  55  65 115  35  55  30 105
```

We can also use a numerical index for the `speed` variable. Since the `speed` variable is the 9th column in the `cyclones` dataset, we can use the following:

```
cyclones[ , 9]
```

which gives the same results as using the variable name for the index:

```
 [1]  55  45  45  65  95  55 145  85 110  80  65 130  85  45 110 185 105  90
[19]  75  65  80 120  35  35 105  60  40  65 105  40  50  30  40  30 110  40
[37] 110 105 115 115  60  80  30  30 105  30  40  25  45  30 105  55  40  35
[55]  95  25  65  75  75  65  70  55  95  80  85  40  25  70  45  65  30  65
[73]  85  95 105  45  55  45  25  65  90 120  50  45  85  30  45 120  35  40
[91]  30  80  50  45  55  65 115  35  55  30 105
```

4 Task 3: Summarise and describe the dataset graphically

4.1 Boxplot of cyclone speed

4.1.1 Basic boxplot of cyclone speed

We use the `boxplot()` function to create a boxplot of the cyclone speed for the entire dataset as follows:

```
boxplot(cyclones$speed)
```

which produces the following plot (Figure 3):

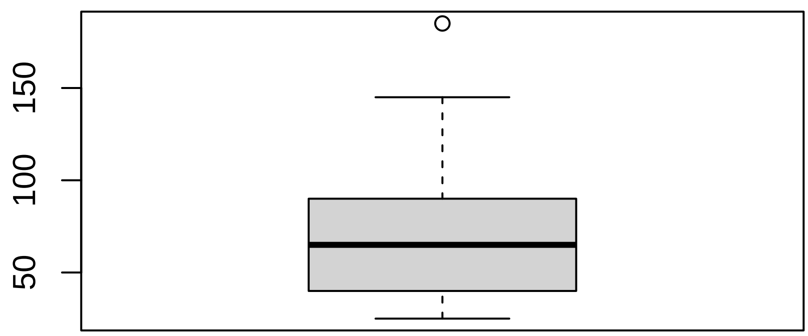

Figure 3: Boxplot of cyclone speed

4.1.2 Basic boxplot of cyclone speed with title and axis labels

We can add a title and axis labels to this plot as follows:

```
boxplot(
  x = cyclones$speed,
  main = "Boxplot of cyclone speed",                    ①
  ylab = "Speed in kph"                                 ②
)
```

① Use the `main` argument of the `boxplot()` function to set a plot title.

② Use the `ylab` argument of the `boxplot()` function to set an y-axis label.

For single boxplots, an x-axis label doesn't make sense so is not specified.

This produces the following plot (Figure 4):

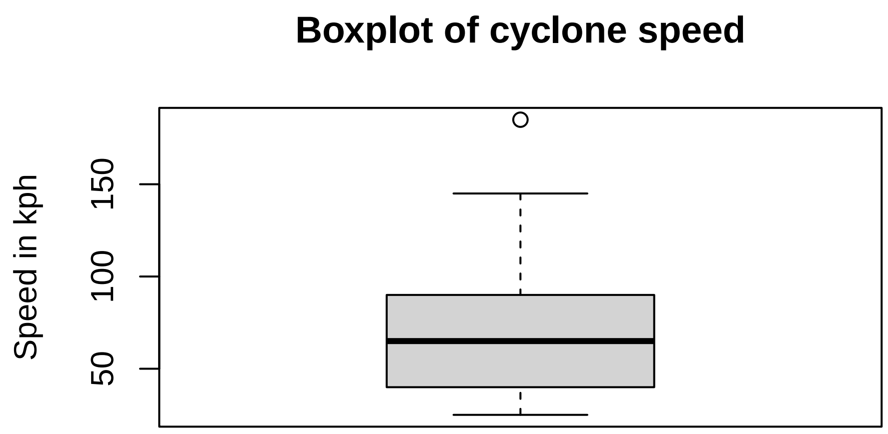

Figure 4: Boxplot of cyclone speed with title and y-axis label

4.1.3 Basic boxplot of cyclone speed with colour

To add colour to a boxplot, we use the following syntax:

```
boxplot(
  x = cyclones$speed,
  main = "Boxplot of cyclone speed",
  ylab = "Speed in kph",
  border = "darkblue",                                    ①
  col = "lightblue"                                       ②
)
```

① Use the `border` argument in `boxplot()` function to specify outline colour for the boxplot.
② Use the `col` argument in `boxplot()` function to specify fill colour the boxplot.

This produces the following plot (Figure 5):

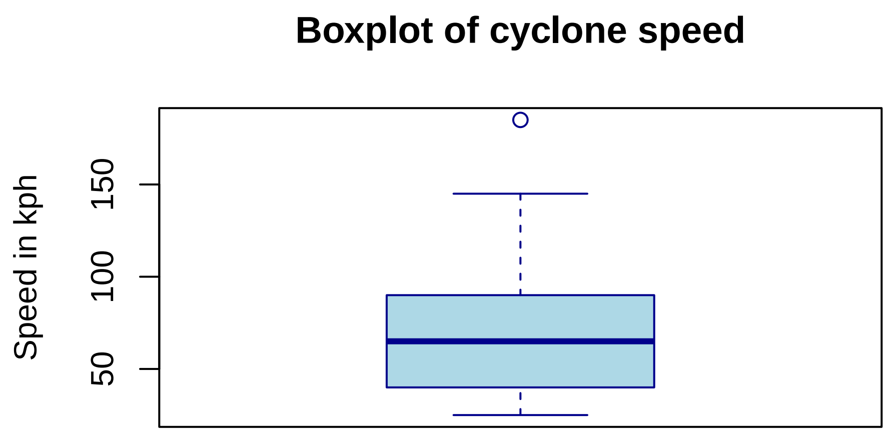

Figure 5: Boxplot of cyclone speed with colours

4.2 Boxplot of cyclone speed by year

4.2.1 Basic boxplot of cyclone speed by year

We use the `boxplot()` function's formula method to create boxplots of cyclone speed by year. The syntax for this is:

```
boxplot(
  speed ~ year,                                   ①
  data = cyclones                                 ②
)
```

① This is the formula method syntax for creating boxplots by a grouping variable.
② Specify the argument for `data` with the data object you are using. This is part of the overall formula method syntax.

This produces the following plot (Figure 6)

Noticeable is that the x and y axis labels have default values based on the names of the variables used for the plot.

4.2.2 Basic boxplot of cyclone speed by year with title and adjusted axis labels

We can add a title and adjust/style the x- and y-axis labels as follows:

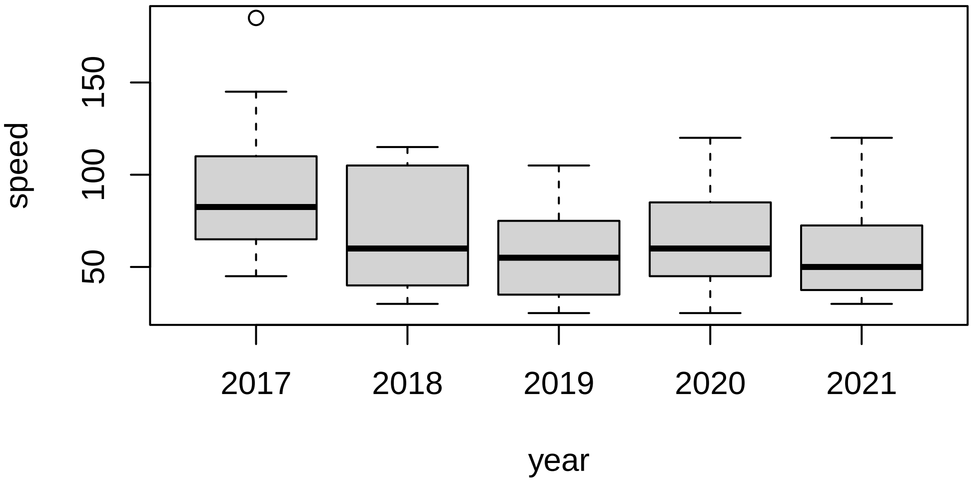

Figure 6: Boxplot of cyclone speed by year

```
boxplot(
  speed ~ year,
  data = cyclones,
  main = "Cyclone speed by year",                                  ①
  xlab = "Year",                                                   ②
  ylab = "Speed in kph"                                            ③
)
```

① Use `main` argument in `boxplot()` function to specify a title for the plot.
② Use the `xlab` argument in `boxplot()` function to edit the x-axis label.
③ Use the `ylab` argument in `boxplot()` function to edit the y-axis label.

This produces the following plot (Figure 7):

4.2.3 Basic boxplot of cyclone speed by year with colour

We can add colour to the boxplots of speed by year as follows:

```
boxplot(
  speed ~ year,
  data = cyclones,
  main = "Cyclone speed by year",
  xlab = "Year",
  ylab = "Speed in kph",
```

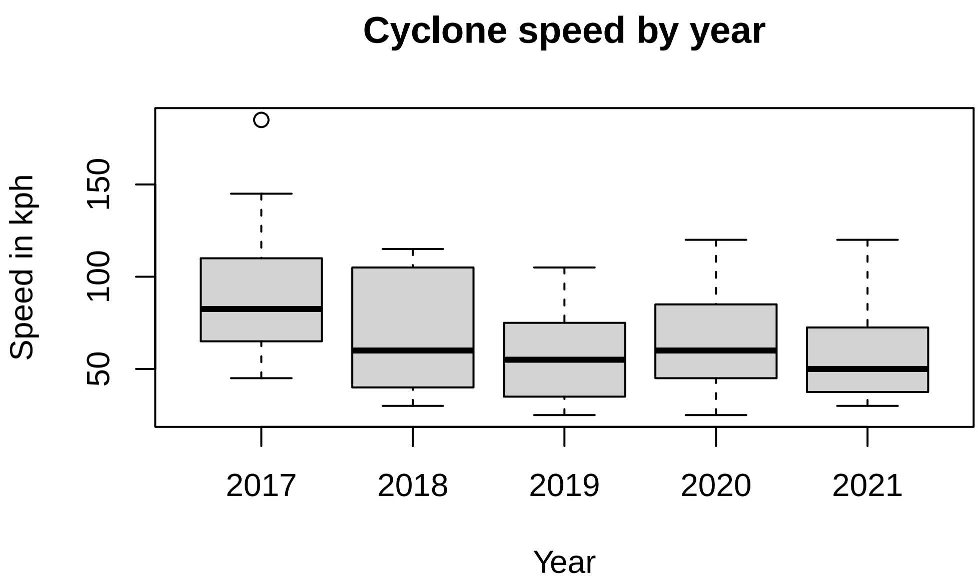

Figure 7: Boxplot of cyclone speed by year

```
  border = "darkblue",                                    ①
  col = "lightblue"                                       ②
)
```

① The `border` argument is used to change the colour of the outline of the boxplots.
② The `col` argument is used to change the fill colour of the boxplots.

This produces the following plot (Figure 8):

We can add colour with each boxplot having its own colour. This can be implemented as follows:

```
boxplot(
  speed ~ year,
  data = cyclones,
  main = "Cyclone speed by year",
  xlab = "Year",
  ylab = "Speed in kph",
  border = rainbow(5),                                    ①
  col = rainbow(5)
)
```

① For the `border` and `col` argument, we supply a vector of five colours using the `rainbow()` function, one for each of the years.

This produces the following plot (Figure 9):

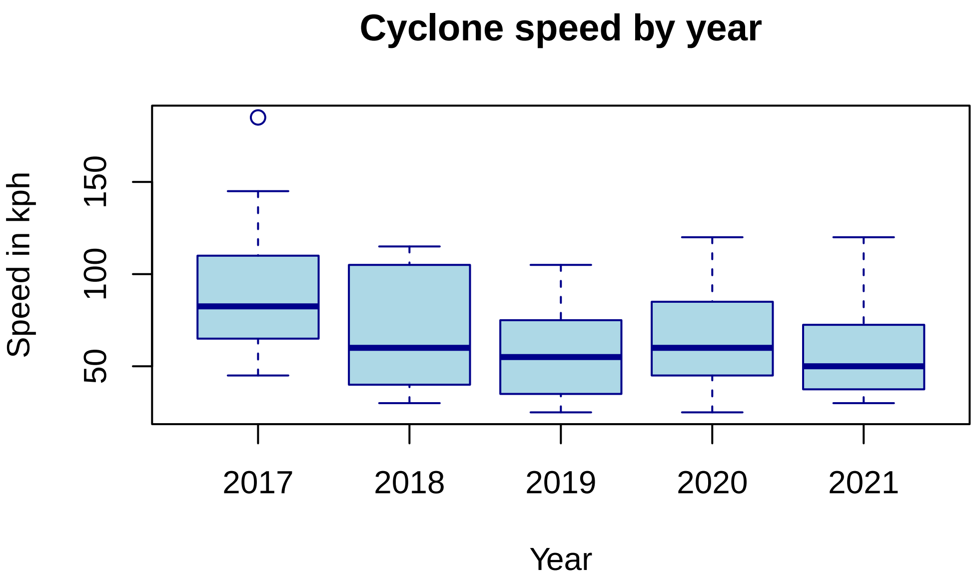

Figure 8: Boxplot of cyclone speed by year

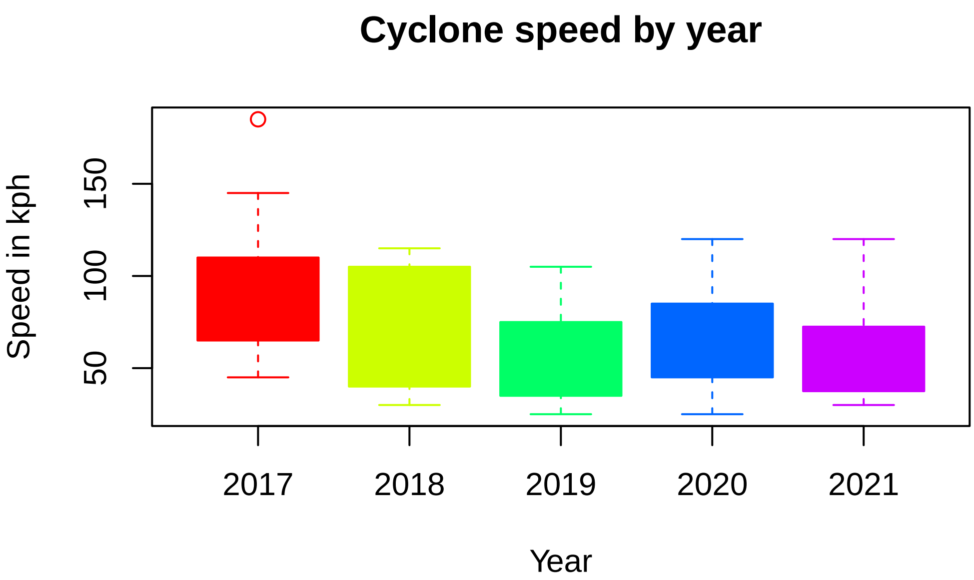

Figure 9: Boxplot of cyclone speed by year

4.3 Histogram of cyclone pressure

4.3.1 Basic histogram of cyclone pressure

We use the `hist()` function to plot a histogram of cyclone pressure as follows:

```r
hist(cyclones$pressure)
```

This produces the following plot (Figure 10):

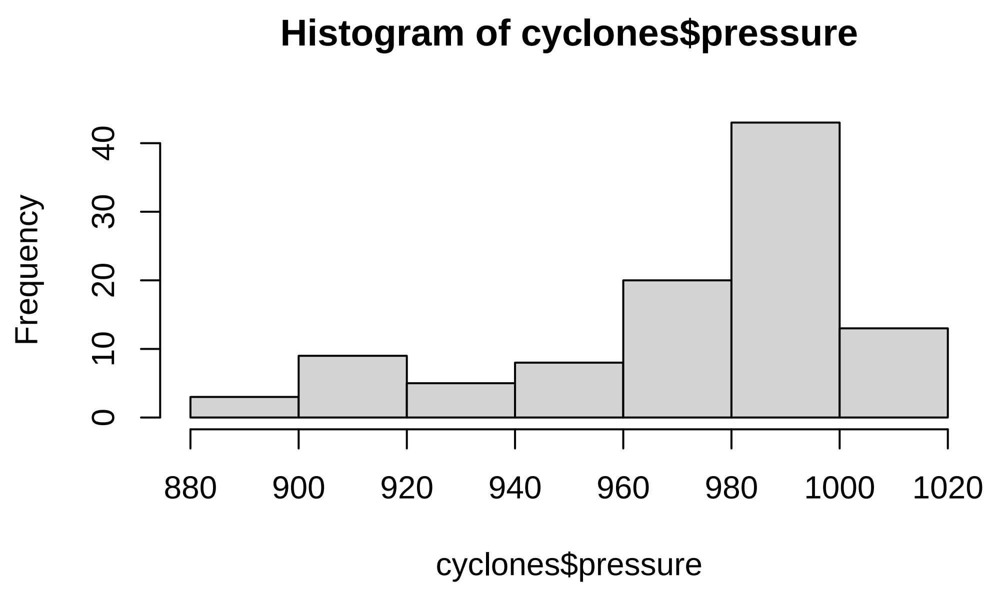

Figure 10: Histogram of cyclone pressure

4.3.2 Basic histogram of cyclone pressure with edited title and axis labels

We can edit the title and the x- and y-axis labels of the histogram as follows:

```r
hist(
  cyclones$pressure,
  main = "Histogram of cyclone pressure",    ①
  xlab = "Pressure (hPa)"                    ②
)
```

① Use `main` argument of `hist()` function to edit the title of the plot.
② Use `xlab` argument of `hist()` function to edit the x-axis label of the plot.

This produces the following plot (Figure 11):

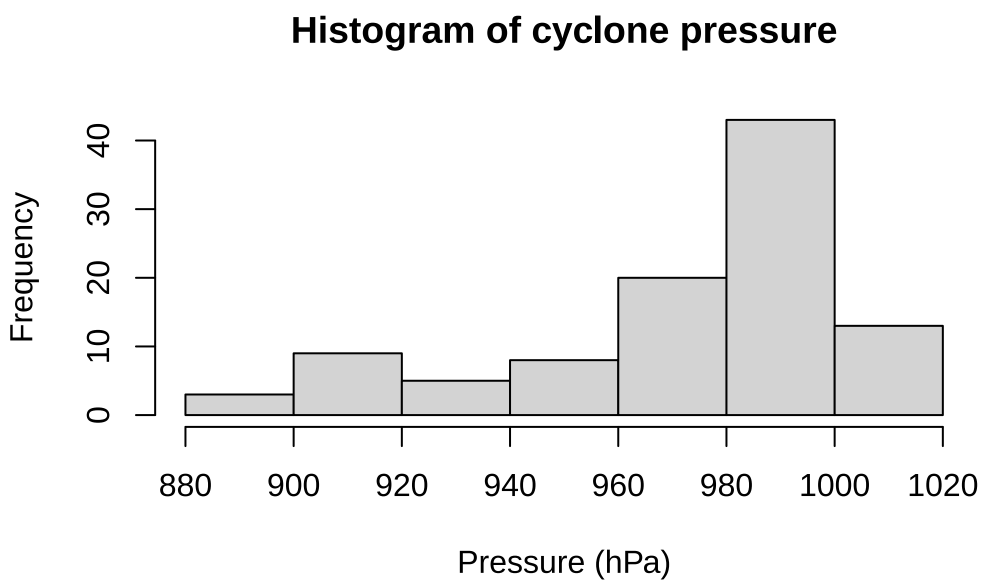

Figure 11: Histogram of cyclone pressure

4.3.3 Basic histogram of cyclone pressure with colour

We can change the colour of a histogram as follows:

```
hist(
  cyclones$pressure,
  main = "Histogram of cyclone pressure",
  xlab = "Pressure (hPa)",
  border = "darkblue",                                    (1)
  col = "lightblue"
)
```

(1) Use `border` and `col` argument of `hist()` function to colour the outline and the fill of the histogram respectively.

This produces the following plot (Figure 12):

4.3.4 Histogram of cyclone pressure for varying cyclone speed

We can plot cyclone pressure by different groups of cyclone speeds. For example, the histogram of cyclone pressure for cyclone speed of less than 100 kph and the histogram of cyclone pressure for cyclone speed of greater than or equal to 100 kph can be plotted as follows:

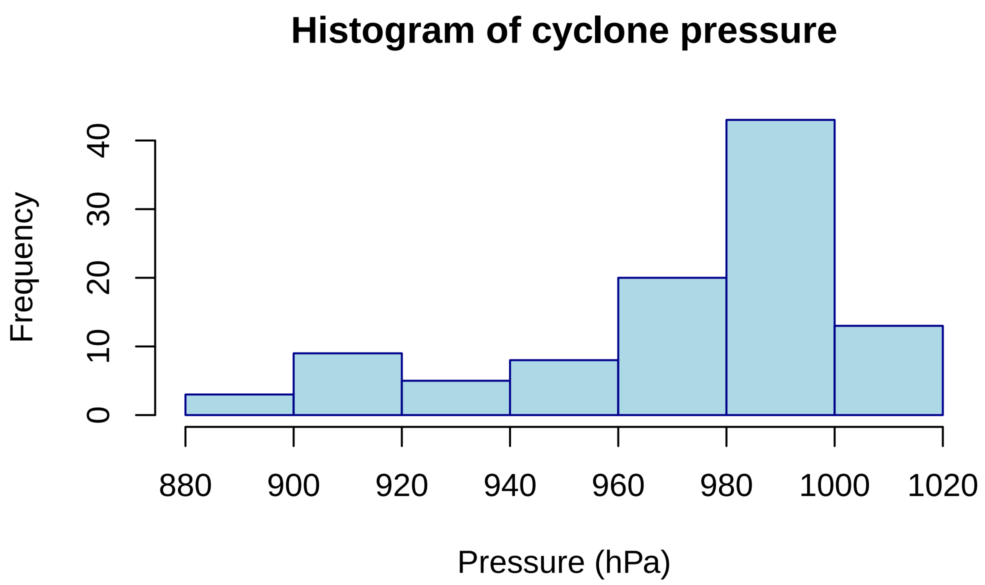

Figure 12: Histogram of cyclone pressure

```
hist(cyclones$pressure[cyclones$speed < 100])
hist(cyclones$pressure[cyclones$speed >= 100])
```

which produces the following plots (Figure 13; Figure 14):

4.3.5 Histogram of cyclone pressure for varying cyclone speed - layered plot

The two plots for different groupings of cyclones by speed can be plotted one plot over the other to facilitate comparison. This can be done as follows:

```
hist(
  cyclones$pressure[cyclones$speed < 100],          ①
  border = "darkgreen",                             ②
  col = "lightgreen",
  main = "Histogram of cyclone pressure",
  xlab = "Pressure in hPa",
  xlim = c(880, 1020)                               ③
)

hist(
  cyclones$pressure[cyclones$speed >= 100],         ④
  border = "darkblue",                              ⑤
  col = "lightblue",
  add = TRUE                                        ⑥
```

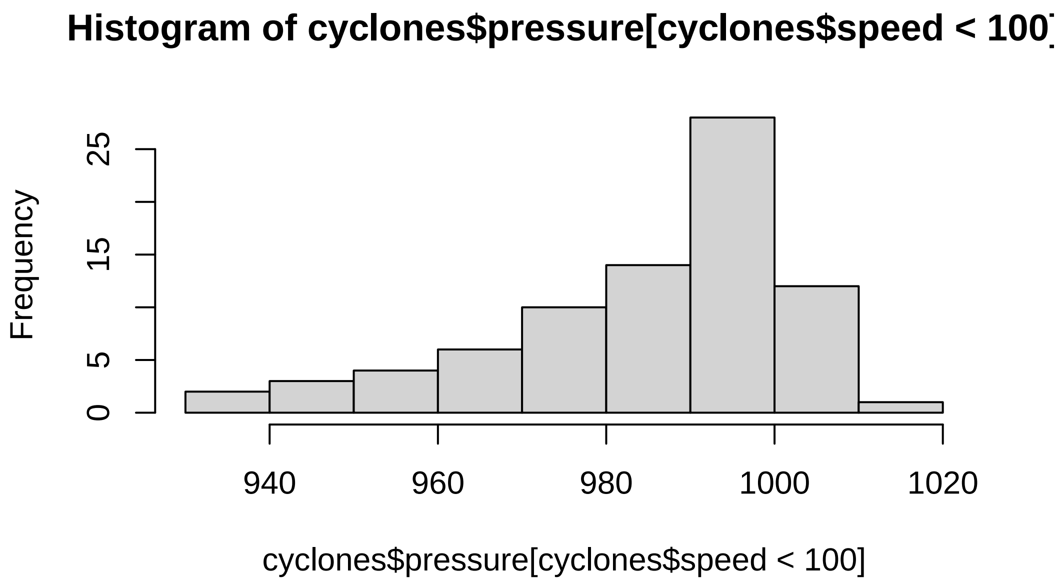

Figure 13: Histogram of cyclone pressure for cyclone speed < 100

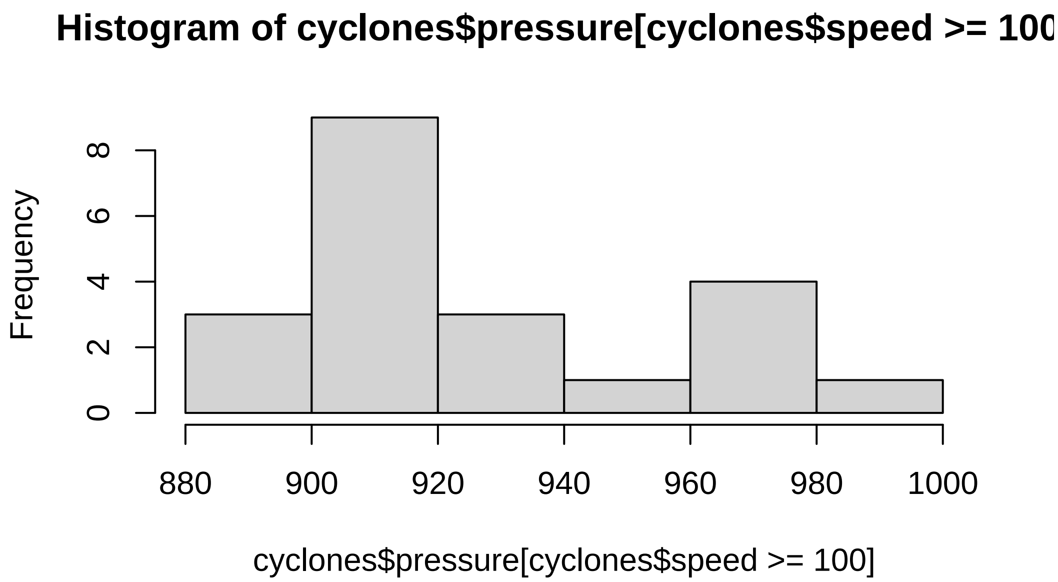

Figure 14: Histogram of cyclone pressure for cyclone speed >= 100

```
)

legend(                                                    ⑦
  x = "topleft",                                           ⑧
  legend = c("Speed < 100", "Speed >= 100"),               ⑨
  fill = c("lightgreen", "lightblue"),                     ⑩
  bty = "n",                                               ⑪
  cex = 0.8,                                               ⑫
  y.intersp = 0.8                                          ⑬
)
```

① Index cyclone pressure by cyclone speed less than 100 kph.
② Set colours to the outline and fill of the histogram for cyclone pressure of cyclones with speed less than 100 kph.
③ Set the x-axis range so that both plots will show appropriately.
④ Index cyclone pressure by cyclone speed greater than or equal to 100 kph.
⑤ Set colours to the outline and fill of the histogram for cyclone pressure of cyclones with speed greater than or equal to 100 kph.
⑥ Use `add` argument of `hist()` function and set to `TRUE` so that current plot is added to the plotting window of previous plot (layered).
⑦ Add a legend using the `legend9()` function to be able to label the plot for cyclone pressure of those cyclones with speed less than 100 kph and the plot for cyclone pressure of those cyclones with speed greater than or equal to 100 kph.
⑧ Set the position of the legend to the top left corner of the plot.
⑨ Add legend labels.
⑩ Set legend colours to match plot colours.
⑪ Remove legend box.
⑫ Set the text size of the legend text.
⑬ Set the amount of space in between lines of text in the legend.

This produces the following plot (Figure 15):

4.3.6 Histogram of cyclone pressure for varying cyclone speed - side-by-side plot

The two plots for different groupings of cyclones by speed can be plotted one plot side-by-side with the other to facilitate comparison. This can be done as follows:

```
par(mfcol = c(1, 2))                                       ①

hist(                                                      ②
  cyclones$pressure[cyclones$speed < 100],
  main = NULL,
  xlab = "Speed < 100 kph",
  ylim = c(0, 30)                                          ③
)
```

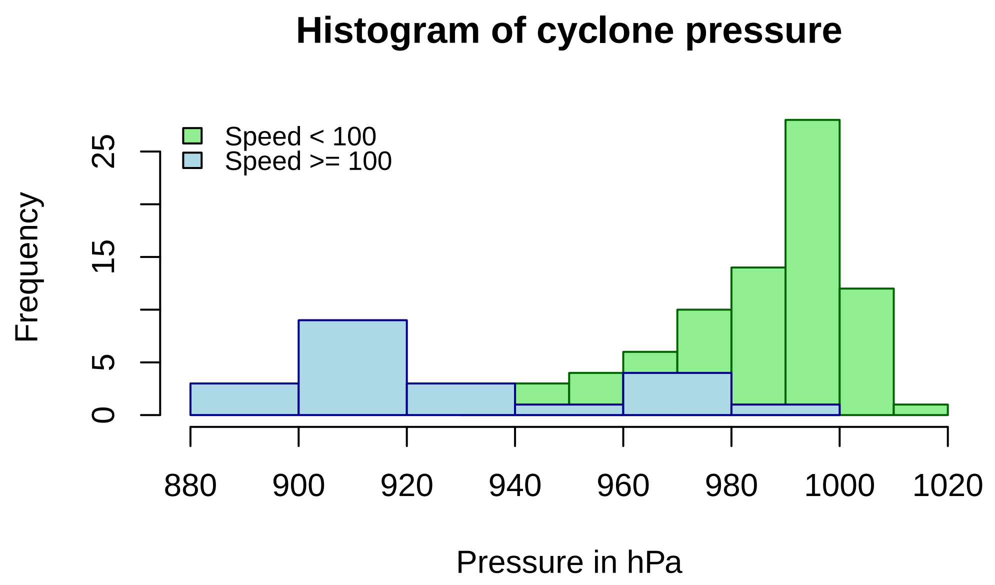

Figure 15: Histogram of cyclone pressure for varying cyclone speed

```
hist(                                                          ④
  cyclones$pressure[cyclones$speed >= 100],
  main = NULL,
  xlab = "Speed >= 100 kph",
  ylim = c(0, 30)                                              ③
)

par(mfcol = c(1, 1))                                           ⑤

title(main = "Histogram of cyclone pressure")                  ⑥
```

① Split plotting window to two - one row and two columns format.
② Plot histogram of cyclones pressure for cyclones with speed less than 100 kph.
③ Set y-axis range of values so that both plots are on the same y-axis scale for comparison.
④ Plot histogram of cyclones pressure for cyclones with speed greater than or equal to 100 kph.
⑤ Set plotting window back to 1 by 1.
⑥ Set title to overall plot.

This produces the following plot (Figure 16):

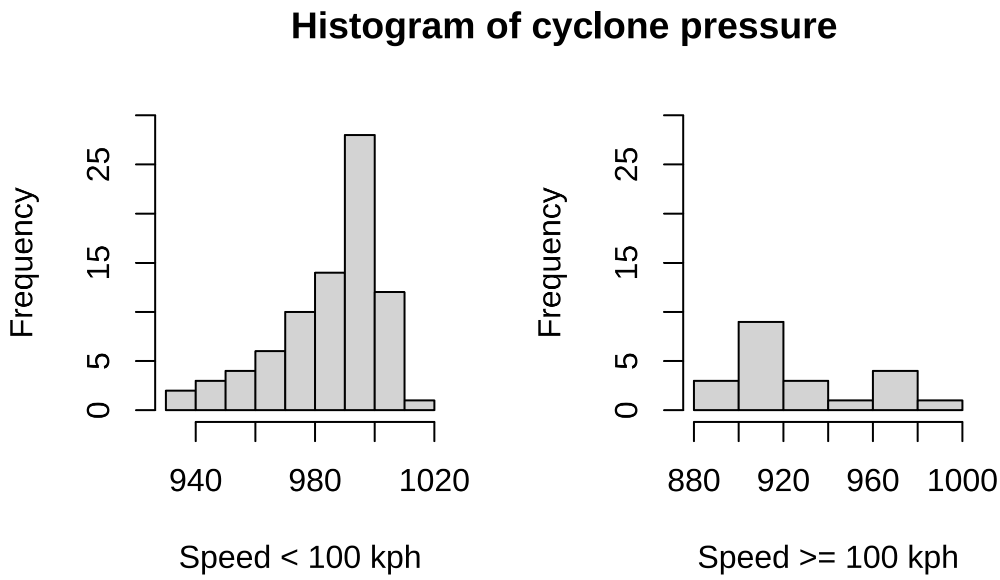

Figure 16: Histogram of cyclone pressure by varying cyclone speed

4.4 Quantile-quantile plots of cyclone pressure and cyclone speed

4.4.1 Quantile-quantile plot of cyclone pressure

A quantile-quantile plot of cyclone pressure can be created as follows:

```
qqnorm(                                                    ①
  cyclones$pressure,
  main = "Quantile-Quantile plot of cyclone pressure"
)
qqline(cyclones$pressure)                                  ②
```

① Produce a QQ plot of cyclone pressure.
② Create a line through a theoretical normal distribution QQ plot that passes through the probability quantities.

This produces the following plot (Figure 17):

4.4.2 Quantile-quantile plot of cyclone speed

A quantile-quantile plot of cyclones speed can be created as follows:

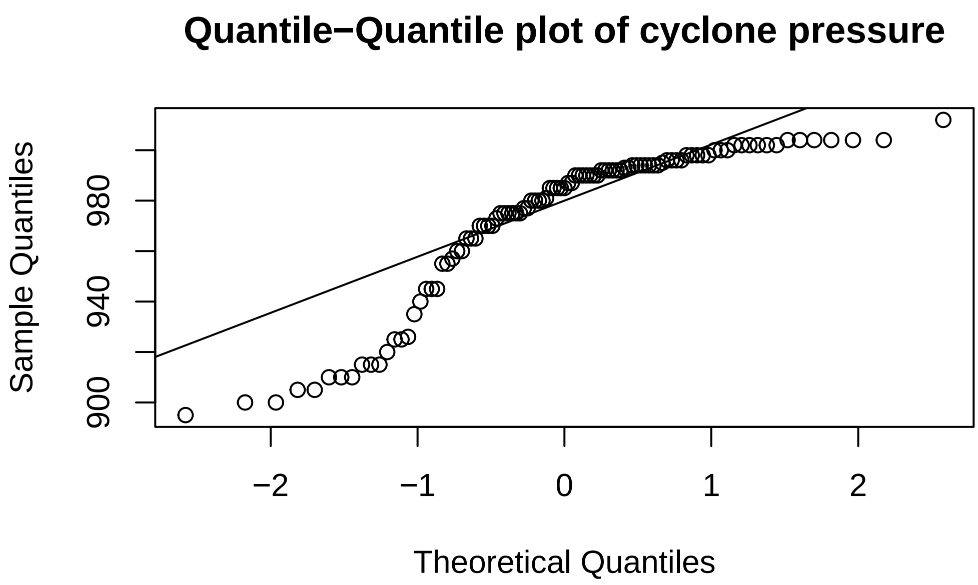

Figure 17: Quantile-Quantile plot of cyclone pressure

```
qqnorm(
  cyclones$speed,                                          ①
  main = "Quantile-Quantile plot of cyclone speed"
)
qqline(cyclones$speed)                                     ②
```

① Produce a QQ plot of cyclone speed.
② Create a line through a theoretical normal distribution QQ plot that passes through the probability quantities.

This produces the following plot (Figure 18):

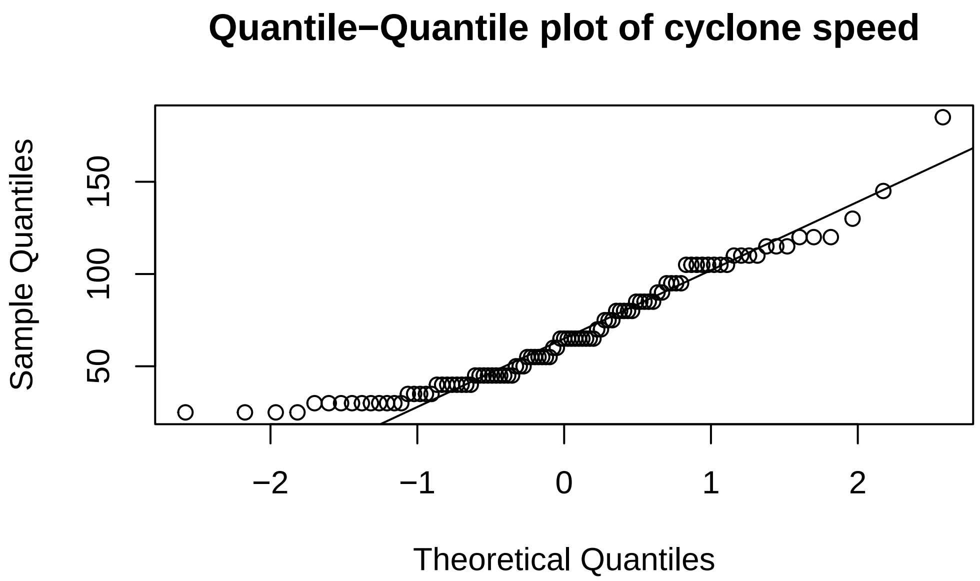

Figure 18: Quantile-Quantile plot of cyclone speed