Statistical methods for assessing agreement between two methods of clinical measurement

Solutions

Ernest Guevarra

28 November 2024

Contents

List of Figures

List of Tables

This document provides detailed solutions to the tasks set in the **Statistical methods for assessing agreement between two methods of clinical measurement** 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

The following tasks have been setup to help students get familiar with functional programming 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 `ba.R` and saved in the project's root directory.

This exercise is based on:

> Bland, J. M. & Altman, DouglasG. Statistical Methods For Assessing Agreement Between Two Methods Of Clinical Measurement. Lancet 327, 307–310 (1986).

The dataset used in the paper can be accessed from the teaching_datasets repository. The URL to the `.dat` file is https://raw.githubusercontent.com/OxfordIHTM/teaching_datasets/main/ba.dat.

The `ba.dat` dataset contains peak expiratory flow rate (PEFR) measurements (in litres per minute) taken with a Wright peak flow metre (`Wright` variable) and a Mini-Wright peak flow metre (`Mini` variable). This is the same data that is presented in the referenced Lancet article above.

2 Task 1: Read the dataset

This task is asking for the learner to create a function that would do the following:

1. Download the `ba.dat` file from the provided download link/URL; and,
2. Read that data into R.

To create this function, we can use already existing functions that does each step separately. For downloading files from the internet, there is a function called `download.file()`. For reading a text dataset, we can look at the `read.table()` family of functions.

2.1 Downloading files from the internet

The `download.file()` is the most straightforward function available in base R for downloading files from the internet. The basic syntax for the function is shown below:

```
download.file(
  url = "https://raw.githubusercontent.com/OxfordIHTM/teaching_datasets/main/ba.dat", ①
  destfile = "data/ba.dat"                                                            ②
)
```

① Provide the download link/URL for your file. For this task, it would be https://raw.githubusercontent.com/OxfordIHTM/teaching_datasets/main/ba.dat. Note that download link/URL should be enclosed in quotes.
② Provide the file path to where the file should be downloaded. The instructions specifically said that this should be downloaded into the `data` directory of the project and for the name, we just use the same name of the data file.

The arguments `url` and `destfile` are the minimum required arguments to specify. When you run this line of code, you should expect the `ba.dat` file to be downloaded into the `data` directory. You can check this by issuing the following command on the R console:

```
list.files("data")
```

```
[1] "ba.dat"
```

which shows that a file called `ba.dat` can be found inside the `data` directory of this project.

2.2 Reading text data

2.2.1 Reading text data after downloading

The `read.table()` function can be used to read the `ba.dat` file. After the data has been downloaded, it should be available in the `data` directory of our project. With this, we can use the `read.table()` function as follows:

```
read.table(
  file = "data/ba.dat",  ①
  header = TRUE,         ②
  sep = " "              ③
)
```

① Specify the file path to the dataset. In our case, the dataset has been downloaded into the `data` directory so the relative file path to the data is `data/ba.dat`.
② Specify whether the dataset being read has variable names or column names in the first row. In our case, the `ba.dat` dataset has variable names included (as described in the instructions). So, we specify `header = TRUE`.

③ Specify the value separators used in the dataset being read. The `ba.dat` dataset uses whitespace (`" "`) to separate values. So we specifcy `sep = " "`.

When we run this line of code, we get:

```
   Wright Mini
1     494  512
2     395  430
3     516  520
4     434  428
5     476  500
6     557  600
7     413  364
8     442  380
9     650  658
10    433  445
11    417  432
12    656  626
13    267  260
14    478  477
15    178  259
16    423  350
17    427  451
```

We get a `data.frame` with 17 rows and 2 columns of data.

2.2.2 Reading text data without downloading

The `read.table()` family of functions allow for reading of text data direct from a download link/URL without having to download the data first. The `file` argument in the `read.table()` family of functions accepts URL of a text data file. The function then reads the data directly from that URL. This functionality is useful for when downloading of data is not needed or if there are rules with regard to downloading and/or keeping of the required dataset. However, it should be noted that reproducibility using a link to read data directly into R depends on whether the URL link is a permanent link that will not change over time.

To read the `ba.dat` dataset directly into R without downloading first, we use the following code:

```
read.table(
  file = "https://raw.githubusercontent.com/OxfordIHTM/teaching_datasets/main/ba.dat",
  header = TRUE,
  sep = " "
)
```

which gives:

```
   Wright Mini
1     494  512
2     395  430
3     516  520
4     434  428
5     476  500
6     557  600
7     413  364
8     442  380
9     650  658
10    433  445
11    417  432
12    656  626
13    267  260
14    478  477
15    178  259
16    423  350
17    427  451
```

This is the same output as the earlier approach.

2.2.3 Function to download and then read the dataset

Now that we know how to use the `download.file()` and the `read.table()` functions, we can now use them into creating a function that downloads the data and then reads it. A basic implementation of this function will look like this:

```
read_ba_data <- function(url, destfile) {                          ①
  download.file(url = url, destfile = destfile)                    ②

  read.table(file = destfile, header = TRUE, sep = " ")            ③
}
```

① We set a function name of `read_ba_data()` and use arguments for `url` and `destfile` which are the two required arguments for `download.file()`. The `destfile` argument will then be used as the specification for the `file` argument for the path to the file to read in the `read.table()` function.

② Use the `url` and `destfile` argument specification to supply corresponding input values to the `download.file()` function.

③ Use the `destfile` argument specification to supply input value to the `read.table()` function. Specifications for `header` and `sep` arguments are hardcoded as it assumes that the `ba.dat` file has a header and uses `" "` as value separator.

We can then try this function to see if it gives us the expected outputs.

```
read_ba_data(
  url = "https://raw.githubusercontent.com/OxfordIHTM/teaching_datasets/main/ba.dat",
  destfile = "data/ba.dat"
)
```

which produces the following result:

```
   Wright Mini
1     494  512
2     395  430
3     516  520
4     434  428
5     476  500
6     557  600
7     413  364
8     442  380
9     650  658
10    433  445
11    417  432
12    656  626
13    267  260
14    478  477
15    178  259
16    423  350
17    427  451
```

The same output is produced as earlier when the dataset was read into R. To check that the download component worked, we check if the `ba.dat` dataset is in the `data` directory of the project.

```
list.files("data")
```

```
[1] "ba.dat"
```

The `ba.dat` file is in the `data` directory. The `read_ba_data()` function works as expected.

2.2.4 Function to conditionally download dataset and then read the dataset

The `read_ba_data()` function is working the way we expect and need it to.

However, we can still consider adding some other features/functionalities that can make the function work more efficiently. For example, depending on our context and our requirements, we might want to just read the `ba.dat` dataset directly from the URL without downloading

it. So, we probably would like to give the user of our function the ability to decide whether the file should be downloaded and then read or should just be read into R directly. For, this, we can refactor the earlier function as follows:

```
read_ba_data <- function(url,
                         download = TRUE,                    ①
                         destfile) {
  ## If download = TRUE, download dataset
  if (download) {                                            ②
    download.file(url = url, destfile = destfile)
  } else {
  ## If download = FALSE, set destfile as url link
    destfile <- url                                          ③
  }

  ## Read dataset
  read.table(file = destfile, header = TRUE, sep = " ")
}
```

① Use an argument called `download` which takes in logical values (TRUE or FALSE). If TRUE, then the function should download the dataset in the specified `destfile`. If FALSE, then the dataset is read directly from the `url`.

② Create a conditional `if()` statement based on whether `download` argument is TRUE or FALSE. If TRUE, then the function should download the dataset in the specified `destfile`.

③ If the `download` conditions is FALSE, then the `destfile` argument is specified with the `url` value so that the dataset is read directly from the `url`.

To test that this updated `read_ba_data()` function works as expected, we first remove the `ba.dat` file that we have already downloaded earlier by using the following commands in the R console:

```
file.remove("data/ba.dat")
```

```
[1] TRUE
```

Then, we check whether it has indeed been removed:

```
file.exists("data/ba.dat")
```

```
[1] FALSE
```

The `ba.dat` file doesn't exist in the `data` directory.

We then test the updated `read_ba_data()` function first with `download = FALSE` as follows:

```
read_ba_data(
  url = "https://raw.githubusercontent.com/OxfordIHTM/teaching_datasets/main/ba.dat",
  download = FALSE                                                                  ①
)
```

① Set `download = FALSE` so that `ba.dat` is read directly into R without downloading. Since no download of data will be performed, there is no need to specify the `destfile` argument.

This code outputs the following:

```
   Wright Mini
1     494  512
2     395  430
3     516  520
4     434  428
5     476  500
6     557  600
7     413  364
8     442  380
9     650  658
10    433  445
11    417  432
12    656  626
13    267  260
14    478  477
15    178  259
16    423  350
17    427  451
```

which is the same output as the earlier function.

We then test whether or not the `ba.dat` file is present in the `data` directory. Our expectation is that the `ba.dat` file is not to be found there. We test as follows:

```
file.exists("data/ba.dat")
```

```
[1] FALSE
```

The `ba.dat` file is not found in the `data` directory. The updated `read_ba_data()` function works as expected.

2.2.5 Function to conditionally download dataset and then read the dataset - overwrite

The updated `read_ba_data()` function is working the way we expect and need it to.

However, we can still consider adding some conditionalities that can make the function work more efficiently. For example, once we have used the current `read_ba_data()` function with `download = TRUE` specification, the `ba.dat` file should already be in our `data` directory. Since this dataset is not expected to change anymore[1], we might want to update the current `read_ba_data()` function such that it will not overwrite an existing download of `ba.dat` if `download = TRUE`. This is useful as we don't have to repeat a download operation if the file is already present in our `data` directory. To implement this functionality, the `read_ba_data()` function can be updated as follows:

```
read_ba_data <- function(url,
                         download = TRUE,
                         destfile,
                         overwrite = FALSE) {                    ①
  ## If download = TRUE, download dataset
  if (download) {
    ## If overwrite = TRUE, download dataset
    if (overwrite) {                                             ②
      download.file(url = url, destfile = destfile)
    } else {
    ## If overwrite = FALSE, check if destfile exists before downloading
      if (!file.exists(destfile)) {                              ③
        download.file(url = url, destfile = destfile)
      }
    }
  } else {
  ## If download = FALSE, set destfile as url link
    destfile <- url
  }

  ## Read dataset
  read.table(file = destfile, header = TRUE, sep = " ")
}
```

① Use `overwrite` argument that takes on logical (TRUE or FALSE) value so that user can specify whether they want the function to overwrite an existing file specified by `destfile`.

② If `download = TRUE` and `overwrite = TRUE`, download of the dataset is performed regardless of whether it is already present.

[1]This dataset is a teaching dataset and is provided by Bland and Altman in their paper. It is very reasonable to expect that no changes will happen to this dataset in the future.

③ If `download = TRUE` and `overwrite = FALSE` (default), a check is performed whether the file path specified in `destfile` exists. If it doesn't exist, then download of the dataset is performed. If the file exists, then dataset is not downloaded.

We can now test whether the function works as expected. From the earlier example, we have the previous download of the `ba.dat` file and then ran the previous function with `download = FALSE`. So, no `ba.dat` file is currently found in the `data` directory. To test whether `overwrite = FALSE` works, we use the updated function as follows

```
read_ba_data(
  url = "https://raw.githubusercontent.com/OxfordIHTM/teaching_datasets/main/ba.dat",
  download = TRUE,                                                                    ①
  destfile = "data/ba.dat",
  overwrite = FALSE
)
```

and then we get:

```
   Wright Mini
1     494  512
2     395  430
3     516  520
4     434  428
5     476  500
6     557  600
7     413  364
8     442  380
9     650  658
10    433  445
11    417  432
12    656  626
13    267  260
14    478  477
15    178  259
16    423  350
17    427  451
```

We get the data read into R and then we expect that the download process happened because `download = TRUE` even though `overwrite = FALSE` because the `ba.dat` file is not present in `data` directory. We check this by

```
file.exists("data/ba.dat")
```

```
[1] TRUE
```

The `ba.dat` file was found in the `data` directory of the project.

Now that the `ba.dat` file is back in the `data` directory, we can test the `overwrite = FALSE` argument again. We expect that the `read_ba_data()` will not re-download the `ba.dat` dataset. Instead, it will just read what is already available in the `data` directory.

```
read_ba_data(
  url = "https://raw.githubusercontent.com/OxfordIHTM/teaching_datasets/main/ba.dat",
  download = TRUE,                                                                  ①
  destfile = "data/ba.dat",
  overwrite = FALSE
)
```

We get the data read into R and during the running of the code, we notice that no console message relating to download were showing with the whole process taking relatively quicker than earlier.

```
   Wright Mini
1     494  512
2     395  430
3     516  520
4     434  428
5     476  500
6     557  600
7     413  364
8     442  380
9     650  658
10    433  445
11    417  432
12    656  626
13    267  260
14    478  477
15    178  259
16    423  350
17    427  451
```

Then we can test the `overwrite = TRUE` argument. We expect that the `read_ba_data()` will download the `ba.dat` dataset regardless of whether the `ba.dat` is found in the `data` directory.

```
read_ba_data(
  url = "https://raw.githubusercontent.com/OxfordIHTM/teaching_datasets/main/ba.dat",
  download = TRUE,                                                                  ①
  destfile = "data/ba.dat",
  overwrite = TRUE
)
```

We get the data read into R and during the running of the code, we notice that on the console we see messages relating to the download process were showing (see console output 1) with the whole process taking relatively longer than earlier.

Listing 1 Console output showing downloading process ongoing

```
trying URL 'https://raw.githubusercontent.com/OxfordIHTM/teaching_datasets/main/ba.dat'
Content type 'text/plain; charset=utf-8' length 166 bytes
==================================================
downloaded 166 bytes
```

```
   Wright Mini
1     494  512
2     395  430
3     516  520
4     434  428
5     476  500
6     557  600
7     413  364
8     442  380
9     650  658
10    433  445
11    417  432
12    656  626
13    267  260
14    478  477
15    178  259
16    423  350
17    427  451
```

The updated `read_ba_data()` function is working as expected. We use it to create a data object called `ba_data`.

```
ba_data <- read_ba_data(
  url = "https://raw.githubusercontent.com/OxfordIHTM/teaching_datasets/main/ba.dat",
  destfile = "data/ba.dat"
)
```

which results in:

```
ba_data
```

```
   Wright Mini
1     494  512
```

```
2      395   430
3      516   520
4      434   428
5      476   500
6      557   600
7      413   364
8      442   380
9      650   658
10     433   445
11     417   432
12     656   626
13     267   260
14     478   477
15     178   259
16     423   350
17     427   451
```

3 Task 2: Calculate the metrics needed for a Bland and Altman plot

The different metrics used to consruct a Bland and Altman plot are:

- Mean of the per subject measurements made by the Wright and the Mini-Wright;
- Difference between the per subject measurements made by the Wright and the Mini-Wright;
- Mean of the difference between the per subject measurements made by the Wright and the Mini-Wright; and,
- Lower and upper limits of agreement between the per subject measurements made by the Wright and the Mini-Wright.

We can approach this task in five different ways.

3.1 Function to calculate Bland and Altman metrics - vectorised approach

The vectorised approach calculates each metric as vectors[2] and then concatenates them into a list[3]. The following function shows this approach:

[2]A vector is essentially a collection of elements of the same data type, arranged in a one-dimensional array.
[3]A list is a data structure that can store elements of different types. Unlike vectors, which are homogenous (all elements must be of the same type), lists are heterogeneous and can hold elements such as numbers, strings, vectors, other lists, and even functions.

```r
calculate_ba_metrics <- function(ba_data) {                          ①
  ## Get per row mean of measurements
  mean_values <- (ba_data$Wright + ba_data$Mini) / 2                 ②

  ## Get per row difference of measurements
  differences <- ba_data$Wright - ba_data$Mini                       ③

  ## Mean of the differences of measurements
  mean_differences <- mean(differences)                              ④

  ## Upper and lower limits of agreement
  upper_limit <- mean_differences + 1.96 * sd(differences)           ⑤
  lower_limit <- mean_differences - 1.96 * sd(differences)

  ## Concatenate metrics into a named list
  list(                                                              ⑥
    mean_values = mean_values,
    differences = differences,
    mean_differences = mean_differences,
    upper_limit = upper_limit,
    lower_limit = lower_limit
  )
}
```

① Name the function `calculate_ba_metrics()` and use an argument `ba_data` which is the data.frame of the dataset we downloaded and read in the first task.
② Calculate the per row mean of the two measurements and assign it to the `mean_values` object.
③ Calculate the per row difference between the two measurements and assign it to the `differences` object.
④ Calculate the mean of the per row differences and assign it to the `mean differences` object.
⑤ Calculate the upper and lower limits of agreement and assign them to the `upper_limit` and `lower_limit` objects respectively.
⑥ Concatenate all the calculated values into a named list.

We use this function as follows:

```r
calculate_ba_metrics(ba_data)
```

which gives the following output:

```
$mean_values
 [1] 503.0 412.5 518.0 431.0 488.0 578.5 388.5 411.0 654.0 439.0 424.5 641.0
[13] 263.5 477.5 218.5 386.5 439.0
```

```
$differences
 [1] -18 -35  -4   6 -24 -43  49  62  -8 -12 -15  30   7   1 -81  73 -24

$mean_differences
[1] -2.117647

$upper_limit
[1] 73.86201

$lower_limit
[1] -78.0973
```

We see the named list structure of the output with the `mean_values` and the `differences` being vectors whilst the rest of the metrics are single values.

3.2 Function to calculate Bland and Altman metrics - data.frame approach

The data.frame approach outputs each metric as vectors and concatenates them into the input dataset. The function using this approach can look like this:

```
calculate_ba_metrics <- function(ba_data) {
  ## Get per row mean of measurements and add to ba_data
  ba_data$mean_values <- (ba_data$Wright + ba_data$Mini) / 2              ①

  ## Get per row difference of measurements and add to ba_data
  ba_data$differences <- ba_data$Wright - ba_data$Mini                    ②

  ## Mean of the differences of measurements
  ba_data$mean_differences <- mean(ba_data$differences)                  ③

  ## Upper and lower limits of agreement
  ba_data$upper_limit <- ba_data$mean_differences + 1.96 *               ④
    sd(ba_data$differences)
  ba_data$lower_limit <- ba_data$mean_differences - 1.96 *
    sd(ba_data$differences)

  ## Return ba_data
  ba_data
}
```

① The per row mean of measurements are calculated and added as a new column in the input dataset.
② The per row difference in measurements are calculated and added as a new column in the input dataset.

③ The mean of the difference in measurements is calculated and added as a new column in the input dataset. Because this value is a single value, it is repeated per row of the input data.

④ The upper and lower limits of agreement in measurements are calculated and are each added as new columns in the input dataset. Because these values correspond to single values, they are repeated per row of the input data.

We use this function as follows:

```
calculate_ba_metrics(ba_data)
```

which gives the following output:

```
   Wright Mini mean_values differences mean_differences upper_limit lower_limit
1     494  512       503.0         -18        -2.117647    73.86201    -78.0973
2     395  430       412.5         -35        -2.117647    73.86201    -78.0973
3     516  520       518.0          -4        -2.117647    73.86201    -78.0973
4     434  428       431.0           6        -2.117647    73.86201    -78.0973
5     476  500       488.0         -24        -2.117647    73.86201    -78.0973
6     557  600       578.5         -43        -2.117647    73.86201    -78.0973
7     413  364       388.5          49        -2.117647    73.86201    -78.0973
8     442  380       411.0          62        -2.117647    73.86201    -78.0973
9     650  658       654.0          -8        -2.117647    73.86201    -78.0973
10    433  445       439.0         -12        -2.117647    73.86201    -78.0973
11    417  432       424.5         -15        -2.117647    73.86201    -78.0973
12    656  626       641.0          30        -2.117647    73.86201    -78.0973
13    267  260       263.5           7        -2.117647    73.86201    -78.0973
14    478  477       477.5           1        -2.117647    73.86201    -78.0973
15    178  259       218.5         -81        -2.117647    73.86201    -78.0973
16    423  350       386.5          73        -2.117647    73.86201    -78.0973
17    427  451       439.0         -24        -2.117647    73.86201    -78.0973
```

We see the `data.frame` structure of the output with new columns/variables for each of the calculated metric and the repeated per row values for the `mean_differences`, `upper_limit`, and `lower_limit`.

3.3 Function to calculate Bland and Altman metrics - combined approach

The structure of the output (either `list` or `data.frame`) each has their pros and cons. A `list` structure is relatively more flexible and can be transformed in many ways whilst the `data.frame` structure is ideal for use in functions that require a `data.frame` input (i.e., `ggplot2` for plotting). With this in mind, a combined approach that gives users an option between a vectorised or a data.frame approach can be useful and have a broader and more generalised usage.

This function can be implemented as follows:

```r
calculate_ba_metrics <- function(ba_data, type = c("list", "df")) {          ①
  type <- match.arg(type)                                                    ②

  mean_values <- (ba_data$Wright + ba_data$Mini) / 2                         ③
  differences <- ba_data$Wright - ba_data$Mini
  mean_differences <- mean(differences)
  upper_limit <- mean_differences + 1.96 * sd(differences)
  lower_limit <- mean_differences - 1.96 * sd(differences)

  if (type == "list") {                                                      ④
    list(
      mean_values = mean_values,
      differences = differences,
      mean_differences = mean_differences,
      upper_limit = upper_limit,
      lower_limit = lower_limit
    )
  } else {
    data.frame(
      ba_data, mean_values, differences, mean_differences,
      upper_limit, lower_limit
    )
  }
}
```

① Use an argument called `type` which can be specified as either `list` or `df` (short for `data.frame`).
② The function `match.arg()` matches the input value for `type` to the choices allowed and checks whether it is specified correctly. If `type` not specified, `match.arg()` uses the first value (`list`) as the default.
③ Calculate each of the metrics like before.
④ Based on what `type` is, the metrics are concatenated into a `list` or into a `data.frame`.

We use this function as follows to output a `list`:

```r
calculate_ba_metrics(ba_data)
```

which gives the following output:

```
$mean_values
 [1] 503.0 412.5 518.0 431.0 488.0 578.5 388.5 411.0 654.0 439.0 424.5 641.0
[13] 263.5 477.5 218.5 386.5 439.0

$differences
 [1] -18 -35  -4   6 -24 -43  49  62  -8 -12 -15  30   7   1 -81  73 -24
```

```
$mean_differences
[1] -2.117647

$upper_limit
[1] 73.86201

$lower_limit
[1] -78.0973
```

We use this function as follows to output a `data.frame`:

```
calculate_ba_metrics(ba_data, type = "df")
```

which gives the following output:

```
   Wright Mini mean_values differences mean_differences upper_limit lower_limit
1     494  512       503.0         -18        -2.117647    73.86201    -78.0973
2     395  430       412.5         -35        -2.117647    73.86201    -78.0973
3     516  520       518.0          -4        -2.117647    73.86201    -78.0973
4     434  428       431.0           6        -2.117647    73.86201    -78.0973
5     476  500       488.0         -24        -2.117647    73.86201    -78.0973
6     557  600       578.5         -43        -2.117647    73.86201    -78.0973
7     413  364       388.5          49        -2.117647    73.86201    -78.0973
8     442  380       411.0          62        -2.117647    73.86201    -78.0973
9     650  658       654.0          -8        -2.117647    73.86201    -78.0973
10    433  445       439.0         -12        -2.117647    73.86201    -78.0973
11    417  432       424.5         -15        -2.117647    73.86201    -78.0973
12    656  626       641.0          30        -2.117647    73.86201    -78.0973
13    267  260       263.5           7        -2.117647    73.86201    -78.0973
14    478  477       477.5           1        -2.117647    73.86201    -78.0973
15    178  259       218.5         -81        -2.117647    73.86201    -78.0973
16    423  350       386.5          73        -2.117647    73.86201    -78.0973
17    427  451       439.0         -24        -2.117647    73.86201    -78.0973
```

3.4 Function to calculate Bland and Altman metrics - modular approach

The modular approach creates multiple functions that calculates each component metric of the Bland and Altman plot and then assembles them into one overall function. An example of how these functions might look like is shown below:

```
calculate_mean_values <- function(m1, m2) { (m1 + m2) / 2 }

calculate_diff_values <- function(m1, m2) { m1 - m2 }

calculate_mean_diff <- function(m1, m2) {
```

```
  mean(calculate_diff_values(m1 = m1, m2 = m2))
}

calculate_diff_limits <- function(m1, m2) {
  differences <- calculate_diff_values(m1 = m1, m2 = m2)
  mean_differences <- calculate_mean_diff(m1 = m1, m2 = m2)

  upper_limit <- mean_differences + 1.96 * sd(differences)
  lower_limit <- mean_differences - 1.96 * sd(differences)

  c(upper_limit, lower_limit)
}

calculate_ba_metrics <- function(ba_data, type = c("list", "df")) {
  type <- match.arg(type)

  mean_values <- calculate_mean_values(m1 = ba_data$Wright, m2 = ba_data$Mini)
  differences <- calculate_diff_values(m1 = ba_data$Wright, m2 = ba_data$Mini)
  mean_differences <- calculate_mean_diff(m1 = ba_data$Wright, m2 = ba_data$Mini)
  limits <- calculate_diff_limits(m1 = ba_data$Wright, m2 = ba_data$Mini)

  if (type == "list") {
    list(
      mean_values = mean_values,
      differences = differences,
      mean_differences = mean_differences,
      upper_limit = limits[1],
      lower_limit = limits[2]
    )
  } else {
    data.frame(
      ba_data, mean_values, differences, mean_differences,
      upper_limit, lower_limit
    )
  }
}
```

4 Task 3: Create a Bland and Altman plot

Develop a function that creates a Bland and Altman plot as described in the article.