WRIC Data Processing Workshop

Nina Ziegenbein

Welcome!

This document serves as a guided tutorial for our workshop on **06.03.2025** at Steno Diabetes Center Aarhus. It is designed to help you get started using the WRIC_preprocessing package, but you can use it afterwards to follow it on your own.

Before we dive into the code-along session, please ensure you have the following installed:

- **R**: The programming language we'll be using for data analysis and visualization.
- **A Programming Environment (RStudio)**: I will demonstrate using RStudio, but feel free to use a different IDE.
- **Quarto**: To render this tutorial, if you choose to go through it by yourself.

Click here for installation links and instructions

- **Install R**
- **Install RStudio**
- **Install Quarto**

FAQ and "Programming Terms"

What is the difference between R and RStudio?

R is a programming language, while **RStudio** is a, so called, integrated development environment (IDE) designed specifically for R, offering a user-friendly interface for coding, plotting, and managing projects. You can think of R like a language, like english, where RStudio is Word - a program you can write english inside. But you could also use other programs for example libre office or latex.

qmd vs R vs Rmd

There are different types of files where you can write R code. A file ending in **.R** is a standard R script for writing and running R code, while in **.qmd (Quarto Document)** or **.Rmd (R Markdown)** you can combine code and text for dynamic reports, or for example this tutorial.

Getting Started

All functions are contained in the `R/preprocessing.R` file. But to also have access to example data, doc_strings, tutorials and HowTo's please download the entire repository for this workshop.

Download the repository from GitHub

1. Clone the Repository using Git (Recommended)

Using Git is the standard and most efficient way to work with this repository, as it allows you to easily pull updates and manage changes in the future. If you don't have Git installed, you can download it from here.

To clone the repository:

- Install Git: If you haven't already, install Git on your computer.
- Clone the Repository: Open your terminal (or Git Bash on Windows), and run the following command to clone the repository to your local machine:

```
git clone https://github.com/hulmanlab/wrictools.git
```

This will create a copy of the repository on your local machine, and you can start working with it immediately.

2. Download the Repository as a ZIP (Alternative)

If you prefer not to use Git, you can download the repository as a ZIP file:

- Go to the repository on GitHub.
- Click the Code button, then click Download ZIP.
- Extract the ZIP file to your desired location on your machine.

Pulling Updates in the Future (Using Git)

Once you've cloned the repository, you can easily stay up to date with the latest features and changes by pulling new updates from the repository.

To get the latest changes from the repository navigate to your local repository. In your terminal, navigate to the directory where you cloned the repository and then pull Updates to get the latest updates (new features, bug fixes, etc.):

::: {.cell}

```
cd Docuents/wrictools #add your own path here
git pull origin main
```

:::

This will download and merge any changes from the GitHub repository to your local copy. By using Git, you can always keep your version up-to-date with the latest improvements without needing to manually download or re-extract the repository.

Set your working directory

- Set your working directory in R to the downloaded WRIC_processing directory. That means your path should end in `".../WRIC_processing"`. You can check the full path, by [add info for both mac and windows]
- **Via RStudio menu:** Session -> Set Working Directory -> Choose Directory
- **Via Terminal:**

```
setwd("path/to/this/folder/WRIC_processing")
getwd() # To check the current directory
```

Why do we change the working directory

The working directory is where RStudio is looking for files. Meaning we can easily type source('R/preprocessing.R') instead of the entire file path. There is nothing wrong with specifiying the entire file path and not changing the working directory. But for this workshop it makes it a lot easier, when we can all use the same code and this ensure that all code runs on your computer without having to change anything.

3. Importing the functions from the "package"

- `source()` runs an R script and loads its functions into your environment

```r
source('R/preprocessing.R')
```

```
RedCap API configuration file 'config.R' not found. Proceeding without.


Attaching package: 'dplyr'

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

    filter, lag

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

    intersect, setdiff, setequal, union
```

Preprocess WRIC data

Now let's preprocess the txt files, that is created by the WRIC. The function `preproces_WRIC_file()` disentangles the meta-data at the top of the file (ID, comment etc) and creates DataFrames and csv-files with the actual data, seperated between both rooms and summarized between the two measurements for each room.

```r
result <- preprocess_WRIC_file("example_data/data.txt")
```

```
Rows: 2601 Columns: 67
-- Column specification --------------------------------------------------------
Delimiter: "\t"
chr   (4): X1, X18, X35, X52
dbl  (56): X3, X4, X5, X6, X7, X8, X9, X10, X11, X12, X13, X14, X15, X16, X2...
lgl   (3): X17, X34, X51
time  (4): X2, X19, X36, X53

i Use `spec()` to retrieve the full column specification for this data.
i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

```r
R1_metadata <- result$R1_metadata
R2_metadata <- result$R2_metadata
df_room1 <- result$df_room1
df_room2 <- result$df_room2
```

The function returns a list with “R1_metadata”, “R2_metadata”, “df_room1” and “df_room2”. Each item of the list is a DataFrame of either the metadata or the preprocessed actual data for either room 1 or 2. If ´save_csv‘ is True, then the DataFrames will be saved as csv files with “id_comment_WRIC_data.csv” or “id_comment_WRIC_metadata.csv”.

Let’s look at the output really quick for room 1:

```
View(R1_metadata)
View(df_room1)
```

But the function can do a lot more and has a lot of extra parameters you can specify. The following is the exact same function call, but mentioning all optional parameters you can call with their default values. Default means, that if you do not specify this parameter, this is the value that the parameter has by default.

```
result <- preprocess_WRIC_file(
    "./example_data/data.txt",
    code="id",
    manual=NULL,
    save_csv=TRUE,
    path_to_save=NULL,
    combine=TRUE,
    method="mean",
    start=NULL,
    end=NULL,
    notefilepath= NULL,
    keywords_dict=NULL
)
```

```
Rows: 2601 Columns: 67
-- Column specification --------------------------------------------------------
Delimiter: "\t"
chr   (4): X1, X18, X35, X52
dbl  (56): X3, X4, X5, X6, X7, X8, X9, X10, X11, X12, X13, X14, X15, X16, X2...
lgl   (3): X17, X34, X51
time  (4): X2, X19, X36, X53

i Use `spec()` to retrieve the full column specification for this data.
i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

Here are explanations and options to all parameters you can specify:

- **filepath:** [String, filepath] Directory path to the WRIC .txt file.
- **code** [String] Method for generating subject IDs. Default is "id", also possible to specify "id+comment", where both ID and comment values are combined or "manual", where you can specify your own.
- **manual** [String] Custom codes for subjects in Room 1 and Room 2 if `code` is "manual".
- **save_csv** [Logical], whether to save extracted metadata and data to CSV files or not. Default is True
- **path_to_save** [String] Directory path for saving CSV files, NULL uses the current directory, NULL is Deafult.
- **combine** [Logical], whether to combine S1 and S2 measurements. Default is True
- **method** [String] Method for combining measurements ("mean", "median", "s1", "s2", "min", "max").
- **start** [character or POSIXct or NULL], rows before this will be removed, if NULL takes first row e.g "2023-11-13 11:43:00"
- **end** [character or POSIXct or NULL], rows after this will be removed, if NULL takes last rows e.g "2023-11-13 11:43:00"
- **notefilepath:** If you specify a path to the corresponding notefile, the code will try to automatically extract the datetime and current protocol specification (sleeping, exercising, eating etc). If possible please read the How To Note File, before you start your study for consistent note taking. If there is a TimeStamp in the note e.g "Participants starts eating at 16:10", the time of the creation of the note will be overwritten with the time specified in the free-text of the note. The "protocol" is extracted by keyword search. You can check currently included keywords and extend them by checking the keywords_dict in the extract_note_info() function of the preprocessing.R file.
- **keywords_dict:** [Nested List] A "dictionary" with keywords for extracting protocol information out of the notefile.

Your Turn

So now it is your turn. Using the `preprocess_WRIC_file()` method create a csv file using "data.txt" in folder example_data.

1) create a csv with the name "XXXX_WRIC_data.csv" combining S1 and S2 measurements by taking the mean between them.
2) create a csv, but cut-off the start to 10:45 on 13/11/2023 and the end to 11:58 on the same day. The csv should be saved as "testing_start_end_parameter_WRIC_data.csv".
3) *Optional:* Try out the notefilepath parameter and see what happens.

Automatic note file extraction - adaptation to your notes

One helpful feature of the `preprocess_WRIC_file()` method is to automatically extract the protocol from the note_file, that is filled in manually during the experiment. With "protocol" I mean coding wether the participant is currently sleeping, eating, exercising etc. This enables quick processing and easy access to extract and compare various e.g. eating periods. Let's try it:

```
result <- preprocess_WRIC_file("./example_data/data.txt",
                               notefilepath="./example_data/note.txt")
```

```
Rows: 2601 Columns: 67
-- Column specification --------------------------------------------------------
Delimiter: "\t"
chr   (4): X1, X18, X35, X52
dbl  (56): X3, X4, X5, X6, X7, X8, X9, X10, X11, X12, X13, X14, X15, X16, X2...
lgl   (3): X17, X34, X51
time  (4): X2, X19, X36, X53

i Use `spec()` to retrieve the full column specification for this data.
i Specify the column types or set `show_col_types = FALSE` to quiet this message.

[1] "Starting time for room 1 is 2023-11-13 21:14:22 and end 2023-11-15 06:35:48 and for roo

Drift: 1.35
```

```
View(result$df_room1)
```

When looking at `df_room1` now, we can see a new column called "protocol". We can see the file starts with 0 and at 22:41:21 changes to 1.

Your Turn

1) Look into the `note.txt` file and find out why there is a change at 22:41:21 and what 0 and 1 might represent. Are there more numbers? What do they represent?
2) When comparing with previous results, notice that the file now starts at a later time and stops at an earlier one. Why might that be? _OBS:_Since we keep reusing variable names (result, df_room1 etc) and use the same data.txt file to create csv_files, we overwrite those files and variables. That is completely fine for this tutorial, where we are focused on how to use it and not the results. But be careful in your own work!

A bit more information about extracting data from the notefile

When specifying a notefilepath, the function will

1) Check wether there is a time in the first row. If there is, this will be used to calculate the drift of the system. This drift will be added to all further datetimes you sepcify within the notefile.

2) Check wether there is information about the partcipant entering or exiting the chamber. If yes, the data is cut to only include times in wich the participant is in the chamber.

   What are the keywords for entering/exiting?

   start = c("ind i kammer", "enter", "ind", "entry") This is only checked in the first three rows. Reasoning behind it, is that the first shows the time drift and then there might be two rows - one for each participant - detailing their entry into the chamber.

   end = c("ud", "exit", "out") This is only checked for the two last rows.

3) Read each row and compare if it contains a keyword that responds to one of the predefined keywords. If yes change the label for that time and all following times until the next match. If you do not specify the `keywords_dict` parameter it will use a default dictionary of keywords and protocol values:

```
keywords_dict <- list(
      sleeping = list(keywords = list(c("seng", "sleeping", "bed", "sove", "soeve", "godnat"
      eating = list(keywords = list(c("start", "begin", "began"), c("maaltid", "måltid", "ea
      stop_sleeping = list(keywords = list(c("vaagen", "vågen", "vaekke", "væk", "wake", "wo
      stop_anything = list(keywords = list(c("faerdig", "færdig", "stop", "end ", "finished"
      activity = list(keywords = list(c("start", "begin", "began"), c("step", "exercise", "pl
      ree_start = list(keywords = list(c("start", "begin", "began"), c("REE", "BEE", "BMR",
    )
```

If there are to lists e.g. for `sleeping`, at least one word of each list need to be present for it to be classified as sleeping. The value at the end of the list is the value used int he protocol column in the created dataframe.

4) Check wether there is a timestamp within the comment. If yes, the time drift is added (if available) and the time in the comment is used instead of the one made in the note file.

Your Turn

1) Look at the `note_new.txt` notefile. Then use notefilepath and specify keywords_dict to automatically process the notefile. Use `data.txt` as the data file. Which comment might be hard to catch with keywords and should be avoided using the `HowToNotefile.pdf`?

Docstrings

Docstrings are text that explain what a function does, what parameter you can specify and what (if any) function returns. For packages in R that you have installed you can read the docstring by typing `?functionname`. Since this is not an official package (yet) this unfortunately does not work yet, but I have created a bit of a workaround. As long as you copy and paste the following piece of code to the top of your R script, you can use `doc("functionname")` to receive the same output - show docstring to see parameter documentation

```
# This part tries to import the documentation of the functions
tryCatch({
  source("R/function_docs.R")
}, error = function(e) {
  cat("Documentation file 'function_docs.R' not found.
      Proceeding without documentation.\n")
})

# This part assigns the documentation imported above to each imported function
if (exists("function_docs", envir = .GlobalEnv)) {
  for (func_name in names(function_docs)) {
    if (exists(func_name, envir = .GlobalEnv)) {
      func <- get(func_name, envir = .GlobalEnv)
      assign(func_name, structure(func, doc = function_docs[[func_name]]),
             envir = .GlobalEnv)    }
  }
}
```

Let's try it and look a bit closer at the output:

```
doc("preprocess_WRIC_file")
```

```
Preprocesses a WRIC data file, extracting metadata, creating DataFrames, and optionally savi
```

```
  @param filepath Path to the WRIC .txt file.

  @param code Method for generating subject IDs ("id", "id+comment", or "manual").

  @param manual Custom codes for subjects in Room 1 and Room 2 if `code` is "manual".

  @param save_csv Logical, whether to save extracted metadata and data to CSV files.

  @param path_to_save Directory path for saving CSV files, NULL uses the current directory.

  @param combine Logical, whether to combine S1 and S2 measurements.

  @param method Method for combining measurements ("mean", "median", "s1", "s2", "min", "max"

  @param start character or POSIXct or NULL, rows before this will be removed, if NULL takes

  @param end character or POSIXct or NULL, rows after this will be removed, if NULL takes las

  @param notefilepath String, Directory path of the corresponding note file (.txt)

  @param keywords_dict Nested List, used to extract protocol values from note file

  @return A list containing the metadata and DataFrames for Room 1 and Room 2.
```

The first sentance is a summary of what the function does. Next there is a list of parameters with the tag @param, followed by the name, the type of variable and a short description. Finally there is an @return statement, stating what the function returns.

Batch Processing

Next lets look at processing multiple files together. You might have all of your wric_data files in one folder and want to process them at the same time. This is an example of just that. In the example_data folder is a folder named my_project with three wric-data files and each containing two participants. That means at the end we have generated 12 files, 6 data files and 6 metadata files.

```r
# Specify the folder with the wric_data
data_folder <- "./example_data/my_project"

# Find all files in the folder that start with "Results_"
data_files <- list.files(data_folder, pattern = "^Results_", full.names = TRUE)
```

```r
# Iterate over all files, call the function and save the csv-files in the same folder
for (data_file in data_files) {
  preprocess_WRIC_file(data_file, path_to_save = data_folder, code = "id+comment")
}
```

```
Rows: 2601 Columns: 67
-- Column specification --------------------------------------------------------
Delimiter: "\t"
chr   (4): X1, X18, X35, X52
dbl  (56): X3, X4, X5, X6, X7, X8, X9, X10, X11, X12, X13, X14, X15, X16, X2...
lgl   (3): X17, X34, X51
time  (4): X2, X19, X36, X53

i Use `spec()` to retrieve the full column specification for this data.
i Specify the column types or set `show_col_types = FALSE` to quiet this message.
Rows: 2601 Columns: 67
-- Column specification --------------------------------------------------------
Delimiter: "\t"
chr   (4): X1, X18, X35, X52
dbl  (56): X3, X4, X5, X6, X7, X8, X9, X10, X11, X12, X13, X14, X15, X16, X2...
lgl   (3): X17, X34, X51
time  (4): X2, X19, X36, X53

i Use `spec()` to retrieve the full column specification for this data.
i Specify the column types or set `show_col_types = FALSE` to quiet this message.
Rows: 2601 Columns: 67
-- Column specification --------------------------------------------------------
Delimiter: "\t"
chr   (4): X1, X18, X35, X52
dbl  (56): X3, X4, X5, X6, X7, X8, X9, X10, X11, X12, X13, X14, X15, X16, X2...
lgl   (3): X17, X34, X51
time  (4): X2, X19, X36, X53

i Use `spec()` to retrieve the full column specification for this data.
i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

When you also want to process note files with it, the code becomes a little bit more complex, since you want to make sure that the correct files are processed together. You can do this based on the shared date in the filename, or a more labour intensive, but maybe easier option is to create a list of filename pairs. Below you can see both options:

Option 1 - Pairs based on shared dates

```
data_folder <- "./example_data/my_project"

data_files <- list.files(data_folder, pattern = "^Results_.*_(\\d{12})\\.txt$",
                         full.names = TRUE)
note_files <- list.files(data_folder, pattern = "^note_(\\d{12})\\.txt$",
                         full.names = TRUE)

# Create a lookup table by extracting the 12-digit date from the filenames
note_lookup <- setNames(note_files, sub("^(note_)(\\d{12})\\.txt$", "\\2",
                         basename(note_files)))

# Loop through the data files and match the date with the note_lookup
for (data_file in data_files) {
  date <- sub(".*_(\\d{12})\\.txt$", "\\1", basename(data_file))
  print(date)
  if (date %in% names(note_lookup)) {
    preprocess_WRIC_file(data_file, notefilepath = note_lookup[date],
                           path_to_save = data_folder, code = "id+comment")
    message("Processed: ", data_file)
  }
}
```

```
[1] "202501130800"

Rows: 2601 Columns: 67
-- Column specification --------------------------------------------------------
Delimiter: "\t"
chr   (4): X1, X18, X35, X52
dbl  (56): X3, X4, X5, X6, X7, X8, X9, X10, X11, X12, X13, X14, X15, X16, X2...
lgl   (3): X17, X34, X51
time  (4): X2, X19, X36, X53

i Use `spec()` to retrieve the full column specification for this data.
i Specify the column types or set `show_col_types = FALSE` to quiet this message.

[1] "Starting time for room 1 is 2023-11-13 21:14:22 and end 2023-11-15 06:35:48 and for roo

Drift: 1.35
Processed: ./example_data/my_project/Results_1m_0101_202501130800.txt
```

```
[1] "202501190800"

Rows: 2601 Columns: 67
-- Column specification --------------------------------------------------------
Delimiter: "\t"
chr   (4): X1, X18, X35, X52
dbl  (56): X3, X4, X5, X6, X7, X8, X9, X10, X11, X12, X13, X14, X15, X16, X2...
lgl   (3): X17, X34, X51
time  (4): X2, X19, X36, X53

i Use `spec()` to retrieve the full column specification for this data.
i Specify the column types or set `show_col_types = FALSE` to quiet this message.

[1] "Starting time for room 1 is 2023-11-13 21:14:22 and end 2023-11-15 06:35:48 and for roo

Drift: 1.35
Processed: ./example_data/my_project/Results_1m_0101_202501190800.txt

[1] "202501250800"

Rows: 2601 Columns: 67
-- Column specification --------------------------------------------------------
Delimiter: "\t"
chr   (4): X1, X18, X35, X52
dbl  (56): X3, X4, X5, X6, X7, X8, X9, X10, X11, X12, X13, X14, X15, X16, X2...
lgl   (3): X17, X34, X51
time  (4): X2, X19, X36, X53

i Use `spec()` to retrieve the full column specification for this data.
i Specify the column types or set `show_col_types = FALSE` to quiet this message.

[1] "Starting time for room 1 is 2023-11-13 21:14:22 and end 2023-11-15 06:35:48 and for roo

Drift: 1.35
Processed: ./example_data/my_project/Results_1m_0101_202501250800.txt
```

Option 2 - Based on File-Pairs

```r
# Manually specify the pairs of data files and note files
filename_pairs <- list(
  list(
    data_file = "./example_data/my_project/Results_1m_0101_202501130800.txt",
    note_file = "./example_data/my_project/note_202501130800.txt"
  ),
  list(
    data_file = "./example_data/my_project/Results_1m_0101_202501190800.txt",
    note_file = "./example_data/my_project/note_202501190800.txt"
  ),
  list(
    data_file = "./example_data/my_project/Results_1m_0101_202501250800.txt",
    note_file = "./example_data/my_project/note_202501250800.txt"
  )
)

# Loop through the filename pairs and process them
for (pair in filename_pairs) {
  preprocess_WRIC_file(pair$data_file, notefilepath = pair$note_file,
                       path_to_save = "./example_data/my_project",
                       code = "id+comment")
  message("Processed: ", pair$data_file, " and ", pair$note_file)
}
```

Your folder structure might look different, so you might have to adjust this code. *(ChatGPT can be very helpful, or feel free to ask me, if your stuck.)*

RedCap

You can also use RedCap's API (Application Programming Interface) to use files directly from RedCap, and also upload the resulting files. To loop over record IDs and process all files within a project on RedCap, use the `preprocess_WRIC_files` function. To find out more about using RedCap, refer to this tutorial: tutorials/RedCap_tutorial.qmd

OBS: During processing, the data-file(s) will be downloaded and afterwards deleted again. If the data is not allowed to be on your personal device at any point, please use this package on a secure server, where you are allowed to (temporarily) store the data.

Should this be in general the end of this "basic" tutorial and be split up into multiple tutorials? -> NO I think makes sense to keep it as this for the workshop

Working with a subset of the data (specific time)

Often you are interested in a certain time period (e.g. after eating or during exercise) and want to perform some calculations based on those time frames. Let's look at how we would do that.

1) Import the preprocessed data (the csv-file) *You can skip this step, if you already have the data.frame, for example right after calling* `preprocess_WRIC_files`.

```
data <- read.csv("./example_data/my_project/AB56_Visit 1_WRIC_data.csv")
# If you followed the tutorial so far, you should have created this csv file
# in this location (otherwise go back to Batch Processing and execute the first
# code chunk or choose a different path
View(data)
```

2) Let's extract the data that we are interested in. Let's start with the first time our participant is eating (breakfast) including one hour afterwards.

```
# we take the first (1) instance where the protocol is 2 (eating)
breakfast_index <- which(data$protocol == 2)[1]
print(breakfast_index)
```

```
[1] 663
```

```
# we create a new data.frame where we take the next 60 rows, including the start_index
data_breakfast <- data[breakfast_index:(breakfast_index + 59),]
View(data_breakfast) #Let's look at the data to check wether it worked correctly
```

Maybe we want to compare RER after breakfast with RER after dinner. So let's extract the dinner time. As participants are eating for some time (e.g 15min) there are 15 rows where protocol is 2. So it would not work to just take the second instance, but we need to identify transitions from another number to 2 and then choose the second transition *(if this is something that is difficult and happens often, this could be a feature for the future?)*

```
# we additionally check wether the row right before (lag) is not 2 and
# then take the second instance (2) to get the dinner time
dinner_index <- which(data$protocol == 2 & lag(data$protocol) != 2)[2]
data_dinner <- data[dinner_index:(dinner_index + 59),]
View(data_dinner) #Let's look at the data to check wether it worked correctly
```

3) Now we can compare the two dataframes. For this example let's use a paired t-test to check wether there are differences in RER between breakfast and dinner. *Please note, this is randomly generated synthetic data and does not represent realistic data. So you can not pull any insights out of this. It's purely to demonstrate how you would use this.*

```r
t.test(data_breakfast$RER, data_dinner$RER, paired = TRUE)
```

```
    Paired t-test

data:  data_breakfast$RER and data_dinner$RER
t = 4.1184, df = 59, p-value = 0.0001205
alternative hypothesis: true mean difference is not equal to 0
95 percent confidence interval:
 0.01718292 0.04965941
sample estimates:
mean difference
     0.03342117
```

Perfect, that was easy enough. Some more helpful functions, you might want to use for/on your sub-dataframes:

- `add_relative_time(dataframe)` - Renumbers *relative_time* column starting from 0. Might be more intuitiv for further use. Example: `data_dinner <- add_relative_time(data_dinner)`
- `cut_rows`- With this function you can easily create a subdataframe (like we did above) based on datetime values (instead of the protocol value). Example: `data_dinner <- cut_rows(data, start="2023-11-14 20:04:00", end="2023-11-14 21:04:00")`
-

Of course you can do these analyes batch-wise as well, the same way as above. Here an example:

```r
files <- list.files(folder_path, pattern = "_data\\.csv$", full.names = TRUE)
for (file in files) {
    data <- read.csv(file)
    breakfast_index <- which(data$protocol == 2)[1]
    data_breakfast <- data[breakfast_index:(breakfast_index + 59),]
    dinner_index <- which(data$protocol == 2 & lag(data$protocol) != 2)[2]
    data_dinner <- data[dinner_index:(dinner_index + 59),]
    message("T-Test Result for : ", file)
    t.test(data_breakfast$RER, data_dinner$RER, paired = TRUE)
}
```

That concludes this tutorial. Now you know all basic functionalities of the package and are ready to use it in your own projects and with real data. Have fun!