Get your dataset ready

Using R and GIS

Rosa Félix Gabriel Valença Rafael Pereira

2024-09-07

Table of contents

1 Introduction **5**
- 1.1 Mobility data . . . 5
  - Why R and GIS . . . 7
- 1.2 Course objectives . . . 7
  - Introduce R Programming Basics . . . 7
  - Teach Data Manipulation Techniques . . . 7
  - Spatial Data Visualization . . . 8
  - Perform Basic Spatial Analysis . . . 8
- 1.3 Target audience . . . 8
- 1.4 Recommended readings . . . 8

2 Course Structure **9**
- 2.1 Day 1 . . . 9
  - Morning . . . 9
  - Afternoon . . . 9
- 2.2 Day 2 . . . 9
  - Morning . . . 9
  - Afternoon . . . 10

3 Detailed schedule (TBC) **11**

4 Location **13**

5 Resources **14**

I Day 1 **15**

6 Software **16**
- 6.1 R . . . 16
  - 6.1.1 Windows . . . 16
  - 6.1.2 Mac . . . 16
  - 6.1.3 Ubuntu . . . 17
- 6.2 RStudio . . . 17
  - 6.2.1 Windows 10/11 . . . 17
  - 6.2.2 MacOS . . . 18

6.2.3 Ubuntu . . . 18
6.3 R packages . . . 18
6.4 r5r . . . 19
6.4.1 Java Development Kit . . . 19
6.4.2 Windows and MacOS . . . 20
6.4.3 Ubuntu . . . 20
6.5 Open Route Service . . . 20

7 R basics **21**
7.1 Simple operations . . . 21
7.1.1 Math operations . . . 21
7.1.2 Basic shortpaths . . . 22
7.2 Practical exercise . . . 23

8 Data manipulation **28**
8.1 Select variables . . . 28
8.1.1 Using pipes! . . . 29
8.2 Filter observations . . . 29
8.3 Create new variables . . . 30
8.4 Change data type . . . 30
8.4.1 Factors . . . 31
8.5 Join data tables . . . 32
8.6 group_by and summarize . . . 34
8.7 Arrange data . . . 35
8.8 All together now! . . . 36
8.9 Other dplyr functions . . . 37

9 Introduction to spatial data **39**

10 Interactive maps **40**
10.1 Mapview . . . 41
10.1.1 Export . . . 43
10.2 tmap . . . 44
10.2.1 Export . . . 45
10.3 Rmarkdown . . . 45

II Day 2 **47**

11 OD pairs and desire lines **48**

12 Georeferenced coordinates **49**

13 Euclidean and routing distances **50**
13.1 Euclidean distances 50

14 Open transportation data **51**

15 Urban Accessibility with R **52**
About the instructor 52

16 Introduction **54**

References **55**

1 Introduction

Materials for the course delivered at the **EIT Doctoral Training Network Annual Forum**, in Gent (Belgium), 19th and 20th September 2024.

This course aims to provide tools to deal with exploring and treating transportation datasets using R programming, an open-source and widely used tool for data analytics in urban mobility.

Additionally, this course provides guidance towards the use of reproducible methods to deal with large datasets that require manipulation and/or spatial analysis.

The course has a **hands-on** approach, where participants will learn the basics of **coding**, **data manipulation**, and **spatial analysis** for urban mobility and transportation.

1.1 Mobility data

There is an emerging increase in mobility data, through new forms of technology, which result in very large and diverse datasets.

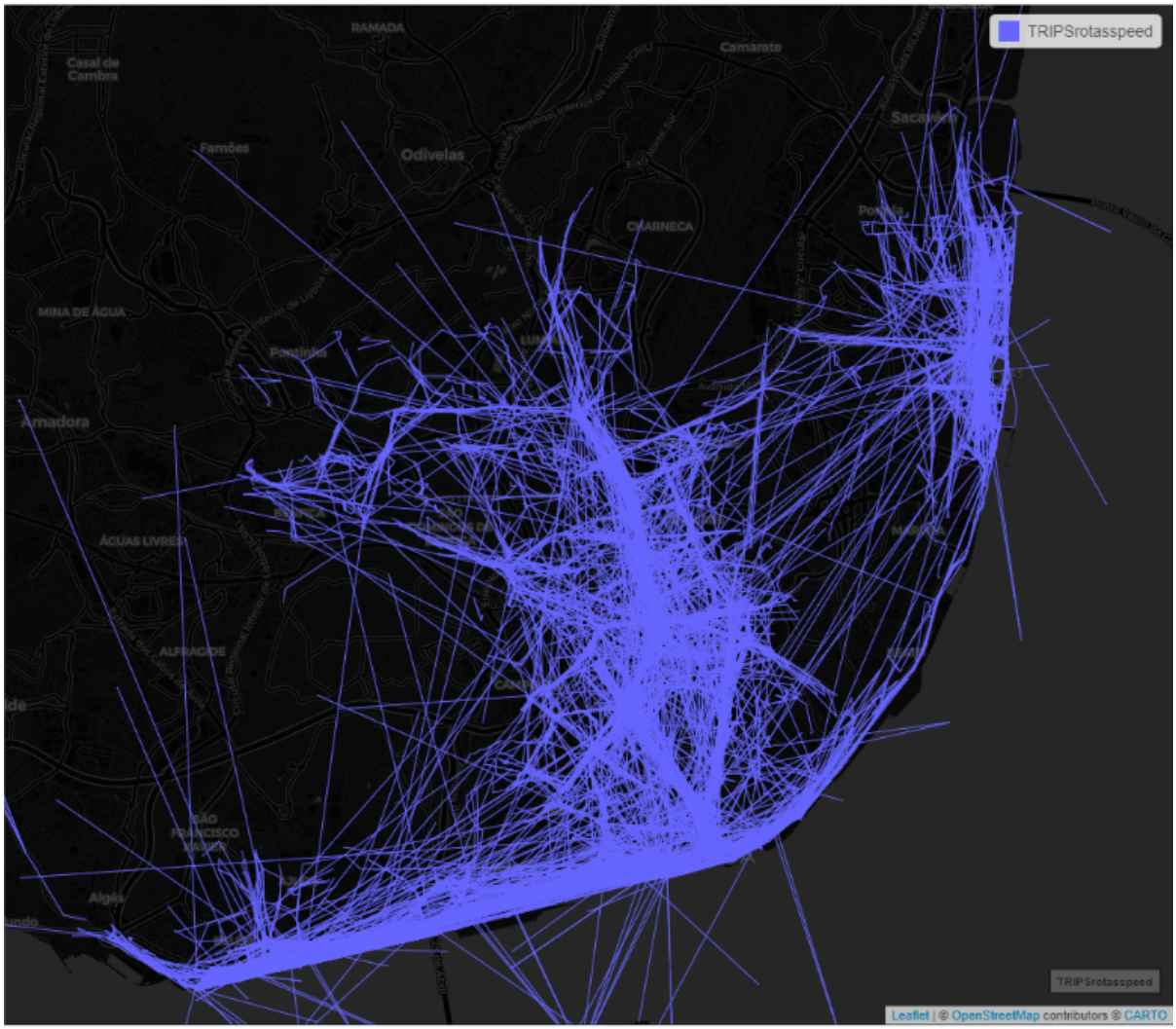


Figure 1.1: E-Scooter trip data in Lisbon. How to deal with it?

Knowing how to get, treat and analyze complex datasets with the up-to-date technologies is extremely relevant for academia, policy makers and start-ups, since it allows them to:

1. acquire critical view on urban mobility based on data;
2. spatially identify locations in the city that require policy priorities;
3. and improve the efficiency of data analysis processes.

Why R and GIS

Most academic programs focus on teaching modelling and deep analysis of data. However, there is a need to learn how to explore and prepare a dataset for modelling. The use of **programming and GIS** techniques have enormous advantages, including their flexibility; reproducibility; and transparency and understanding the step-by-step process.

The use of GIS techniques in transportation is, traditionally, not considered in transportation learning programs, despite being of enormous relevance when doing accessibility analysis or reeling with georreferenced transportation data, such as bike sharing route trips' datasets, origin-destination flows datasets, home/work locations, GTFS public transit data, and so on. There is a need to learn how to locate these open datasets, how to explore them and how to integrate them into transportation and urban analysis. Additionally, the use of open source software and datasets allows researchers to perform methods that are reproducible and transparent.

TLDR

- Open-source tools widely used in data analytics and spatial analysis
- Flexibility and reproducibility in data manipulation and visualization
- Critical for urban mobility and transportation research, with spatial relevance
- Large transportation datasets are becoming increasingly common

1.2 Course objectives

Introduce R Programming Basics

- Equip participants with foundational skills in R programming
- Emphasize reproducible research practices to ensure transparency and replicability in analyses

Teach Data Manipulation Techniques

- Use key R packages for data cleaning, manipulation, and summarization of datasets
- Enable participants to efficiently handle large and complex transportation datasets

Spatial Data Visualization

- Introduce methods for quick and effective spatial data visualization using R and GIS tools
- Provide hands-on experience with creating interactive maps and visualizations

Perform Basic Spatial Analysis

- Teach participants how to perform spatial analysis of transportation datasets using GIS techniques with R
- Cover practical applications such as georeferencing data, accessibility analysis, and routing ODs
- Utilize real-world transportation data for practical, hands-on learning

1.3 Target audience

- Ph.D. candidates from DTN and other researchers
- Policy makers and practitioners in urban mobility
- Beginners to intermediate R users, no prior experience needed

1.4 Recommended readings

- Engel, Claudia A (2023) Introduction to R
- Lovelace, Robin, Nowosad, Jakub & Muenchow, Johannes (2023) Geocomputation with R
- Pereira, Rafael H. M. & Herszenhut, Daniel (2023) Introduction to urban accessibility: a practical guide with R. Ipea - Institute of Applied Economic Research

2 Course Structure

The course consists of an in-person 2-day course, taking place during the EIT DTN Annual Meeting on the **19th and 20th September 2024**.

The first day will focus on learning the basics of R programming and how to treat and explore datasets. The second day will focus on analyzing spatial datasets, and routing origins to destinations.

2.1 Day 1

Morning

- Introduction to **programming** techniques and **data structures**
- Introduction to R, and RStudio: **software installation** and main packages
- **R base and basics**: examples and exercises

Afternoon

- **Data manipulation**: using the dplyr package to select, filter, left-join, group and summarize
- Introduction to **GIS** and **spatial data**: import and visualize vector data
- R markdown and **interactive maps**

2.2 Day 2

Morning

- **Desire lines** from OD and transport zones
- **Georeference** coordinates: examples from surveys
- **Routing**: from euclidean distances to road networks

Afternoon

- **Open Transportation data**: where to find it
- **Accessibility analysis with R**: *r5r demo* (Rafael Pereira)
- Group **exercise**

3 Detailed schedule (TBC)

Day 1	
9.30	Introductions and Presentation of the course contents
10.00	Introduction to programming techniques and data structures
10.30	Introduction to R and RStudio: hands-on to install software and main packages
11.00	*Coffee break*
11.15	(cont.)
11.30	R basics: examples and exercises
12.30	*Lunch break*
13.30	Data manipulation: examples and exercises (select, filter, left-join, group and summarize, using dplyr package)
15.00	Introduction to GIS and spatial data: import and visualize vector data
15.30	*Coffee break*
15.45	(cont.)
16.15	View and export interactive maps
17.00	*End of day 1*

Day 2	
9.30	Desire-lines from OD pairs and transport zones: examples and exercises
10.30	Georeferenced coordinates from survey responses: example and exercises
11.00	*Coffee break*
11.15	(cont.)
11.30	Euclidean and routing distances with sf and r5r
12.30	*Lunch break*
13.30	Open Transportation data: where to find it (OSM and GTFS)

Day 2	
14.00	Accessibility analysis with r5r
16.00	*Coffee break*
16.15	Using you data: manipulation and spatial analysis methods and further applications
16.45	Survey and feedback from participants
17.00	*End of day 2*

4 Location

The course will take place at Campus Sterre, Building S8, room 2.4.

```
Campus_S8_coord = c(3.7105372, 51.0241258)
Campus_S8 = sf::st_sfc(sf::st_point(Campus_S8_coord)) # create point
Campus_S8 = sf::st_as_sf(Campus_S8, crs = 4326) # assign crs

mapview::mapview(Campus_S8, map.types = "OpenStreetMap") # quick map view
```

5 Resources

- You laptop, with any OS
- Github repository with all the materials (data, code and guidelines)
- Survey datasets, school locations and public transport operator datasets

Part I

Day 1

6 Software

In this chapter we will guide you through the installation of R, RStudio and the packages you will need for this course.

R and **RStudio**[1] are separate downloads.

6.1 R

You will need **R** installed on your computer. **R stats** (how it is also known) is a programming language and free software environment for statistical computing and graphics supported by the R Foundation for Statistical Computing.

The download links live at The Comprehensive R Archive Network (aka CRAN). The most recent version is `4.4.1`, but you can use `>= 4.1.x` if you already have it installed.

6.1.1 Windows

Download R-4.4.1 for Windows and run the executable file.

You will also need to install Rtools, which is a collection of tools necessary to build R packages in Windows.

6.1.2 Mac

Download R-4.4.1 for MacOX. You will have to choose between the arm64 or the x86-64 version.

Download the `.pkg` file and install it as usual.

[1] We will use RStudio, although if you already use other studio such as VScode, that's also fine.

6.1.3 Ubuntu

> These are instructions for Ubuntu. If you use other linux distribution, please follow the instructions on The Comprehensive R Archive Network - CRAN.

You can look for R in the Ubuntu **Software Center** or install it via the terminal:

```
# sudo apt update && sudo apt upgrade -y
sudo apt install r-base
```

Or, if you prefer, you can install the latest version of R from CRAN:

```
# update indices
sudo apt update -qq
# install two helper packages we need
sudo apt install --no-install-recommends software-properties-common dirmngr
# add the signing key (by Michael Rutter) for these repos
wget -qO- https://cloud.r-project.org/bin/linux/ubuntu/marutter_pubkey.asc | sudo tee -a /et
# add the R 4.0 repo from CRAN -- adjust 'focal' to 'groovy' or 'bionic' as needed
sudo add-apt-repository "deb https://cloud.r-project.org/bin/linux/ubuntu $(lsb_release -cs)
```

Then run:

```
sudo apt install r-base r-base-core r-recommended r-base-dev
```

[Optional] To keep up-to-date r version and packages, you can follow the instructions at r2u

After this installation, you don't need to open R base. Please proceed to install RStudio.

6.2 RStudio

RStudio Desktop is an integrated development environment (IDE) for R. It includes a console, syntax-highlighting editor that supports direct code execution, as well as tools for plotting, history, debugging and workspace management.

RStudio is available for free download from Posit RStudio.

6.2.1 Windows 10/11

Download RStudio 2024.04 and run the executable file.

6.2.2 MacOS

Download RStudio 2024.04 and install it as usual.

6.2.3 Ubuntu

> These are instructions for Ubuntu **22** / Debian 12. If you use other linux distribution, please follow the instructions on Posit RStudio.

Install it via the terminal:

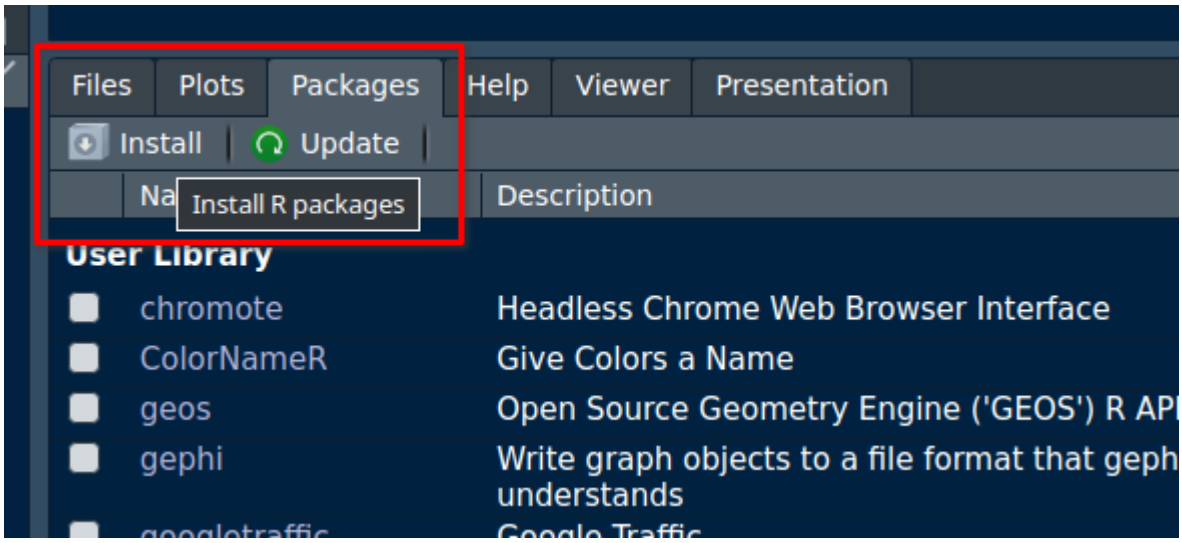
```
sudo apt install libssl-dev libclang-dev
wget https://download1.rstudio.org/electron/jammy/amd64/rstudio-2024.04.2-764-amd64.deb
sudo dpkg -i rstudio*
rm -v rstudio*
```

If you already use Ubuntu **24**, please check and replace the correct url from RStudio Dailies

6.3 R packages

You will need to install some packages to work with the data and scripts in this course.

You can install them in RStudio by searching for them in the **Packages** tab:



or by running the following code in the console:

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

install.packages(c("remotes", "devtools", "usethis")) # optional
install.packages("sf")
install.packages("mapview")
install.packages("rmarkdown")
```

6.4 r5r

The workshop **"A crash course on urban accessibility with R"** uses a few R packages that need to be installed on your machine. The simplest way to do this is running the code below. This might take a few minutes if this is the first time you install these packages.

```
pkgs = c("r5r", "accessibility", "rJavaEnv", "h3jsr")

install.packages(pkgs)
```

6.4.1 Java Development Kit

To use the `{r5r}` package (version v2.0 or higher), you will need to have *Java Development Kit (JDK) 21* installed on your computer. There are numerous open-source JDK implementations. The easiest way to install JDK is using the new {rJavaEnv} package in R.

```
# check version of Java currently installed (if any)
rJavaEnv::java_check_version_rjava()

## if this is the first time you use {rJavaEnv}, you might need to run this code
## below to consent the installation of Java.
# rJavaEnv::rje_consent(provided = TRUE)

# install Java 21
rJavaEnv::java_quick_install(
  version = 21,
  distribution = 'Corretto')

# check if Java was successfully installed
rJavaEnv::java_check_version_rjava()
```

Alternatively, you can manually download and install JDK 21.

6.4.2 Windows and MacOS

Go to Java Development Kit 21, download the latest 21 build corresponding to your operating system and run the executable file.

6.4.3 Ubuntu

Install it via the terminal:

```
sudo apt install -y openjdk-21-jdk openjdk-21-jre
java -version
```

6.5 Open Route Service

Sign up for an account and create a token. Copy your API.

In RStudio console, run:

```
# install.packages("openrouteservice")
openrouteservice::ors_api_key("YOUR-API-KEY")
```

This will store your key on your `.Renviron` file, meaning that every time you open RStudio, you won't need to run this command again.

This is useful also to write your `openrouteservice` scripts without sharing your key with others.

7 R basics

In this chapter we will introduce to the R basics and some exercises to get familiar to how R works.

7.1 Simple operations

7.1.1 Math operations

7.1.1.1 Sum

```
1+1
```

```
[1] 2
```

7.1.1.2 Subtraction

```
5-2
```

```
[1] 3
```

7.1.1.3 Multiplication

```
2*2
```

```
[1] 4
```

7.1.1.4 Division

```r
8/2
```

```
[1] 4
```

7.1.1.5 Round the number

```r
round(3.14)
```

```
[1] 3
```

```r
round(3.14, 1) # The "1" indicates to round it up to 1 decimal digit.
```

```
[1] 3.1
```

You can use help `?round` in the console to see the description of the function, and the default arguments.

7.1.2 Basic shortpaths

7.1.2.1 Perform Combinations

```r
c(1, 2, 3)
```

```
[1] 1 2 3
```

```r
c(1:3) # The ":" indicates a range between the first and second numbers.
```

```
[1] 1 2 3
```

7.1.2.2 Create a comment with `ctrl + shift + m`

```
# Comments help you organize your code. The software will not run the comment.
```

7.1.2.3 Create a table

A simple table with the number of trips by car, PT, walking, and cycling in a hypothetical street segment at a certain period.

Define variables

```
modes <- c("car", "PT", "walking", "cycling") # you can use "=" or "<-"
Trips = c(200, 50, 300, 150) # uppercase letters modify
```

Join the variables to create a table

```
table_example = data.frame(modes, Trips)
```

Take a look at the table

Visualize the table by clicking on the "Data" in the "Environment" page or use :

```
View(table_example)
```

7.2 Practical exercise

Dataset: the number of trips between all municipalities in the Lisbon Metropolitan Area, Portugal (Instituto National de Estatística 2018).

7.2.0.1 Import dataset

You can click directly in the file under the "Files" pan, or:

```
data = readRDS("data/TRIPSmode_mun.Rds")
```

> Note that after you type `"` you can use `tab` to navigate between folders and files and `enter` to autocomplete.

7.2.0.2 Take a first look at the data

Summary statistics

```
summary(data)
```

```
  Origin_mun        Destination_mun        Total             Walk
 Length:315         Length:315         Min.   :     7   Min.   :     0
 Class :character   Class :character   1st Qu.:   330   1st Qu.:     0
 Mode  :character   Mode  :character   Median :  1090   Median :     0
                                       Mean   : 16825   Mean   :  4033
                                       3rd Qu.:  5374   3rd Qu.:     0
                                       Max.   :875144   Max.   :306289
      Bike              Car             PTransit            Other
 Min.   :   0.00   Min.   :     0   Min.   :     0.0   Min.   :    0.0
 1st Qu.:   0.00   1st Qu.:   263   1st Qu.:     5.0   1st Qu.:    0.0
 Median :   0.00   Median :   913   Median :   134.0   Median :    0.0
 Mean   :  80.19   Mean   :  9956   Mean   :  2602.6   Mean   :  152.4
 3rd Qu.:   0.00   3rd Qu.:  4408   3rd Qu.:   975.5   3rd Qu.:   62.5
 Max.   :5362.00   Max.   :349815   Max.   :202428.0   Max.   :11647.0
```

Check the structure of the data

```
str(data)
```

```
'data.frame':   315 obs. of  8 variables:
 $ Origin_mun     : chr  "Alcochete" "Alcochete" "Alcochete" "Alcochete" ...
 $ Destination_mun: chr  "Alcochete" "Almada" "Amadora" "Barreiro" ...
 $ Total          : num  20478 567 188 867 114 ...
 $ Walk           : num  6833 0 0 0 0 ...
 $ Bike           : num  320 0 0 0 0 0 0 0 91 0 ...
 $ Car            : num  12484 353 107 861 114 ...
 $ PTransit       : num  833 0 81 5 0 ...
 $ Other          : num  7 214 0 0 0 0 0 0 0 0 ...
```

Check the first values of each variable

```
data
```

```r
head(data, 3) # first 3 values
```

```
  Origin_mun Destination_mun Total Walk Bike   Car PTransit Other
1  Alcochete       Alcochete 20478 6833  320 12484      833     7
2  Alcochete          Almada   567    0    0   353        0   214
3  Alcochete         Amadora   188    0    0   107       81     0
```

Check the number of rows (observations) and columns (variables)

```r
nrow(data)
```

```
[1] 315
```

```r
ncol(data)
```

```
[1] 8
```

Open the dataset

```r
View(data)
```

7.2.0.3 Explore the data

Check the total number of trips

Use $ to select a variable of the data

```r
sum(data$Total)
```

```
[1] 5299853
```

Percentage of car trips related to the total

```r
sum(data$Car)/sum(data$Total) * 100
```

```
[1] 59.17638
```

Percentage of active trips related to the total

```
(sum(data$Walk) + sum(data$Bike)) / sum(data$Total) * 100
```

```
[1] 24.44883
```

7.2.0.4 Modify original data

Create a column with the sum of the number of trips for active modes

```
data$Active = data$Walk + data$Bike
```

Filter by condition (create new tables)

Filter trips only with origin from Lisbon

```
data_Lisbon = data[data$Origin_mun == "Lisboa",]
```

Filter trips with origin **different** from Lisbon

```
data_out_Lisbon = data[data$Origin_mun != "Lisboa",]
```

Filter trips with origin **and** destination in Lisbon

```
data_in_Out_Lisbon = data[data$Origin_mun == "Lisboa" & data$Destination_mun == "Lisboa",]
```

Remove a column

Look at the first row

```
data[1,] #rows and columns start from 1
```

```
  Origin_mun Destination_mun Total Walk Bike   Car PTransit Other Active
1  Alcochete       Alcochete 20478 6833  320 12484      833     7   7153
```

Look at first row and column

```
data[1,1]
```

```
[1] "Alcochete"
```

Remove the first column

```
data = data[ ,-1] #first column
```

Create a table only with origin, destination and walking trips

There are many ways to do the same operation.

```
names(data)
```

```
[1] "Destination_mun" "Total"           "Walk"            "Bike"
[5] "Car"             "PTransit"        "Other"           "Active"
```

```
data_walk2 = data[ ,c(1,2,4)]
```

```
data_walk3 = data[ ,-c(3,5:9)]
```

7.2.0.5 Export data

Save data in **.csv** and **.Rds**

```
write.csv(data, 'data/dataset.csv', row.names = FALSE)
saveRDS(data, 'data/dataset.Rds') #Choose a different file.
```

7.2.0.6 Import data

```
csv_file = read.csv("data/dataset.csv")
rds_file = readRDS("data/dataset.Rds")
```

8 Data manipulation

In this chapter we will use some very useful `dplyr` functions to handle and manipulate data.

You can load the `dplyr` package directly, or load the entire tidy universe (`tidyverse`).

```
# library(tidyverse)
library(dplyr)
```

Using the same dataset as in R basics but with slightly differences[1].

We will do the same operations but in a simplified way.

```
TRIPS = readRDS("data/TRIPSorigin.Rds")
```

> Note that it is very important to understand the R basics, that's why we started from there, even if the following functions will provide the same results.

You don't need to know everything! And you don't need to know by heart. The following functions are the ones you will probably use most of the time to handle data.

8.1 Select variables

Have a look at your dataset. You can open using `View()`, look at the information at the "Environment" panel, or even print the same information using `glimpse()`

```
glimpse(TRIPS)
```

We will create a new dataset with *Origin*, *Walk, Bike* and *Total*. This time we will use the `select()` function.

```
TRIPS_new = select(TRIPS, Origin, Walk, Bike, Total) # the first argument is the dataset
```

[1] This dataset includes the number of trips with origin in each neighborhood, divided by mode of transport, and inter or intra municipal trips.

The first argument, as usually in R, is the dataset, and the remaining ones are the columns to select.

With most of the `dplyr` functions you don't need to refer to `data$...` you can simply type the variable names (and even without the `"..."`!). This makes coding in R simpler :)

You can also remove columns that you don't need.

```
TRIPS_new = select(TRIPS_new, -Total) # dropping the Total column
```

8.1.1 Using pipes!

Now, let's introduce pipes. Pipes are a rule as: "**For this, do this.**"

This is useful to skip the first argument of the functions (usually the dataset to apply the function).

Applying a pipe to the `select()` function, we can write as:

```
TRIPS_new = TRIPS |> select(Origin, Walk, Bike, Total)
```

Two things to **note**:

1. The pipe symbol can be written as `|>` or `%>%`. [2] To write it you may also use the `ctrl+shift+m` shortcut.
2. After typing `select(` you can press `tab` and the list of available variables of that dataset will show up! `Enter` to select. With this you prevent typo errors.

8.2 Filter observations

You can filter observations based on a condition using the `filter()` function.

```
TRIPS2 = TRIPS[TRIPS$Total > 25000,] # using r-base, you cant forget the comma
TRIPS2 = TRIPS2 |> filter(Total > 25000) # using dplyr, it's easier
```

You can have other conditions inside the condition.

```
summary(TRIPS$Total)
```

[2] You can change this in RStudio > Tools > Global Options > Code.

```
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
    361    5918   17474   22457   33378  112186
```

```
TRIPS3 = TRIPS |> filter(Total > median(Total))
```

Other filter conditions:

- `==` equal, `!=` different
- `<` smaller, `>` greater, `<=` smaller or equal, `>=` greater or equal
- `&` and, `|` or
- `is.na`, `!is.na` is not NA
- `%in%`, `!%in%` not in

8.3 Create new variables

You can also try again to create a variable of Car percentage using pipes! To create a new variable or change an existing one (overwriting), you can use the `mutate()` function.

```
TRIPS$Car_perc = TRIPS$Car/TRIPS$Total * 100 # using r-base

TRIPS = TRIPS |> mutate(Car_perc = Car/Total * 100) # using dplyr
```

8.4 Change data type

Data can be in different formats. For example, the variable *Origin* is a character, but we can convert it to a numeric variable.

```
class(TRIPS$Origin)
```

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

```
TRIPS = TRIPS |>
  mutate(Origin_num = as.integer(Origin)) # you can use as.numeric() as well
class(TRIPS$Origin_num)
```

```
[1] "integer"
```

Most used data types are:

- integer (`int`)
- numeric (`num`)
- character (`chr`)
- logical (`logical`)
- date (`Date`)
- factor (`factor`)

8.4.1 Factors

Factors are useful to deal with categorical data. You can convert a character to a factor using `as.factor()`, and also use labels and levels for categorical ordinal data.

We can change the `Lisbon` variable to a factor, and `Internal` too.

```
TRIPS = TRIPS |>
  mutate(Lisbon_factor = factor(Lisbon, labels = c("No", "Yes")),
         Internal_factor = factor(Internal, labels = c("Inter", "Intra")))
```

But how do we know which levels come first? A simple way is to use `table()` or `unique()` functions.

```
unique(TRIPS$Lisbon) # this will show all the different values
```

```
[1] 0 1
```

```
table(TRIPS$Lisbon) # this will show the frequency of each value
```

```
  0   1
188  48
```

```
table(TRIPS$Lisbon_factor)
```

```
 No Yes
188  48
```

The first number to appear is the first level, and so on.

8.5 Join data tables

When having relational tables - *i.e.* with a common identifier - it is useful to be able to join them in a very efficient way.

`left_join` is a function that joins two tables **by a common column**. The **first table is the one that will be kept**, and the **second one will be joined to** it. How `left_join` works:

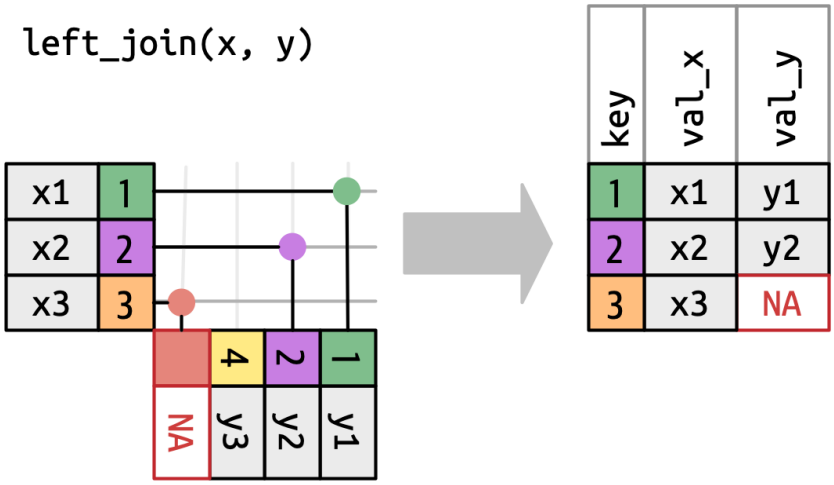


Figure 8.1: A visual representation of the left join where every row in x appears in the output.Source: R for Data Science.

Let's **join the municipalities** to this table with a supporting table that includes all the **relation** between neighbourhoods and municipalities, and the respective names and codes.

```
Municipalities = readRDS("data/Municipalities_names.Rds")
```

```
head(TRIPS)
```

```
# A tibble: 6 x 13
  Origin Total  Walk  Bike   Car PTransit Other Internal Lisbon Car_perc
  <chr>  <dbl> <dbl> <dbl> <dbl>    <dbl> <dbl>    <dbl>  <dbl>    <dbl>
1 110501 35539 11325  1309 21446     1460     0        0      0     60.3
2 110501 47602  3502   416 37727     5519   437        1      0     79.3
3 110506 37183 12645    40 22379     2057    63        0      0     60.2
4 110506 42313  1418   163 37337     3285   106        1      0     88.2
5 110507 30725  9389  1481 19654      201     0        0      0     64.0
```

```
6 110507 54586  2630   168 44611     6963   215        1      0     81.7
# i 3 more variables: Origin_num <int>, Lisbon_factor <fct>,
#   Internal_factor <fct>
```

```
tail(Municipalities)
```

```
    Mun_code Neighborhood_code        Municipality
113     1109            110913               Mafra
114     1114            111409 Vila Franca de Xira
115     1109            110918               Mafra
116     1109            110904               Mafra
117     1502            150202           Alcochete
118     1109            110911               Mafra
                                          Neighborhood
113                                     Santo Isidoro
114                                  Vila Franca de Xira
115 União das freguesias de Azueira e Sobral da Abelheira
116                                        Encarnação
117                                           Samouco
118                                          Milharado
```

We can see that we have a common variable: `Origin` in `TRIPS` and `Neighborhood_code` in `Municipalities`.

To join these two tables we need to specify the common variable in each table, using the `by` argument.

```
TRIPSjoin = TRIPS |> left_join(Municipalities, by = c("Origin" = "Neighborhood_code"))
```

If you prefer, you can mutate or rename a variable so both tables have the same name. When **both tables have the same name**, you don't need to specify the `by` argument.

```
Municipalities = Municipalities |> rename(Origin = "Neighborhood_code") # change name
TRIPSjoin = TRIPS |> left_join(Municipalities) # automatic detects common variable
```

As you can see, both tables don't need to be the same length. The `left_join` function will keep all the observations from the first table, and join the second table to it. If there is no match, the variables from the second table will be filled with `NA`.

8.6 group_by and summarize

We have a very large table with all the neighbourhoods and their respective municipalities. We want to know the total number of trips with origin in each municipality.

To make it easier to understand, let's keep only the variables we need.

```
TRIPSredux = TRIPSjoin |> select(Origin, Municipality, Internal, Car, Total)
head(TRIPSredux)
```

```
# A tibble: 6 x 5
  Origin Municipality Internal   Car Total
  <chr>  <chr>           <dbl> <dbl> <dbl>
1 110501 Cascais             0 21446 35539
2 110501 Cascais             1 37727 47602
3 110506 Cascais             0 22379 37183
4 110506 Cascais             1 37337 42313
5 110507 Cascais             0 19654 30725
6 110507 Cascais             1 44611 54586
```

We can group this table by the `Municipality` variable and summarize the number of trips with origin in each municipality.

```
TRIPSsum = TRIPSredux |>
  group_by(Municipality) |> # you won't notice any chagne with only this
  summarize(Total = sum(Total))
head(TRIPSsum)
```

```
# A tibble: 6 x 2
  Municipality   Total
  <chr>          <dbl>
1 Alcochete      36789
2 Almada        289834
3 Amadora       344552
4 Barreiro      133658
5 Cascais       373579
6 Lisboa       1365111
```

We summed the total number of trips in each municipality.

If we want to group by more than one variable, we can add more `group_by()` functions.

```
TRIPSsum2 = TRIPSredux |>
  group_by(Municipality, Internal) |>
  summarize(Total = sum(Total),
            Car = sum(Car))
head(TRIPSsum2)
```

```
# A tibble: 6 x 4
# Groups:   Municipality [3]
  Municipality Internal  Total    Car
  <chr>           <dbl>  <dbl>  <dbl>
1 Alcochete           0  16954   9839
2 Alcochete           1  19835  15632
3 Almada              0 105841  49012
4 Almada              1 183993 125091
5 Amadora             0 117727  33818
6 Amadora             1 226825 142386
```

We summed the total number of trips and car trips in each municipality, **separated by** inter and intra municipal trips.

> It is a good practice to use the `ungroup()` function after the `group_by()` function. This will remove the grouping. If you don't do this, the grouping will be kept and you may have unexpected results in the next time you use that dataset.

8.7 Arrange data

You can **sort** a dataset by one or more variables.

For instance, `arrange()` by Total trips, ascending or descending order.

```
TRIPS2 = TRIPSsum2 |> arrange(Total)
TRIPS2 = TRIPSsum2 |> arrange(-Total) # descending

TRIPS2 = TRIPSsum2 |> arrange(Municipality) # alphabetic

TRIPS4 = TRIPS |> arrange(Lisbon_factor, Total) # more than one variable
```

This is not the same as opening the view table and click on the arrows. When you do that, the order is not saved in the dataset. If you want to save the order, you need to use the `arrange()` function.

8.8 All together now!

This is the pipes magic. It takes the last result and applies the next function to it. "With this, do this.". You can chain as many functions as you want.

```
TRIPS_pipes = TRIPS |>
  select(Origin, Internal, Car, Total) |>

  mutate(Origin_num = as.integer(Origin)) |>
  mutate(Internal_factor = factor(Internal, labels = c("Inter", "Intra"))) |>

  filter(Internal_factor == "Inter")|>

  left_join(Municipalities) |>

  group_by(Municipality) |>
  summarize(Total = sum(Total),
            Car = sum(Car),
            Car_perc = Car/Total * 100) |>

  arrange(desc(Car_perc))
```

With this code we will have a table with the total number of intercity trips, by municipality, with their names instead of codes, arranged by the percentage of car trips.

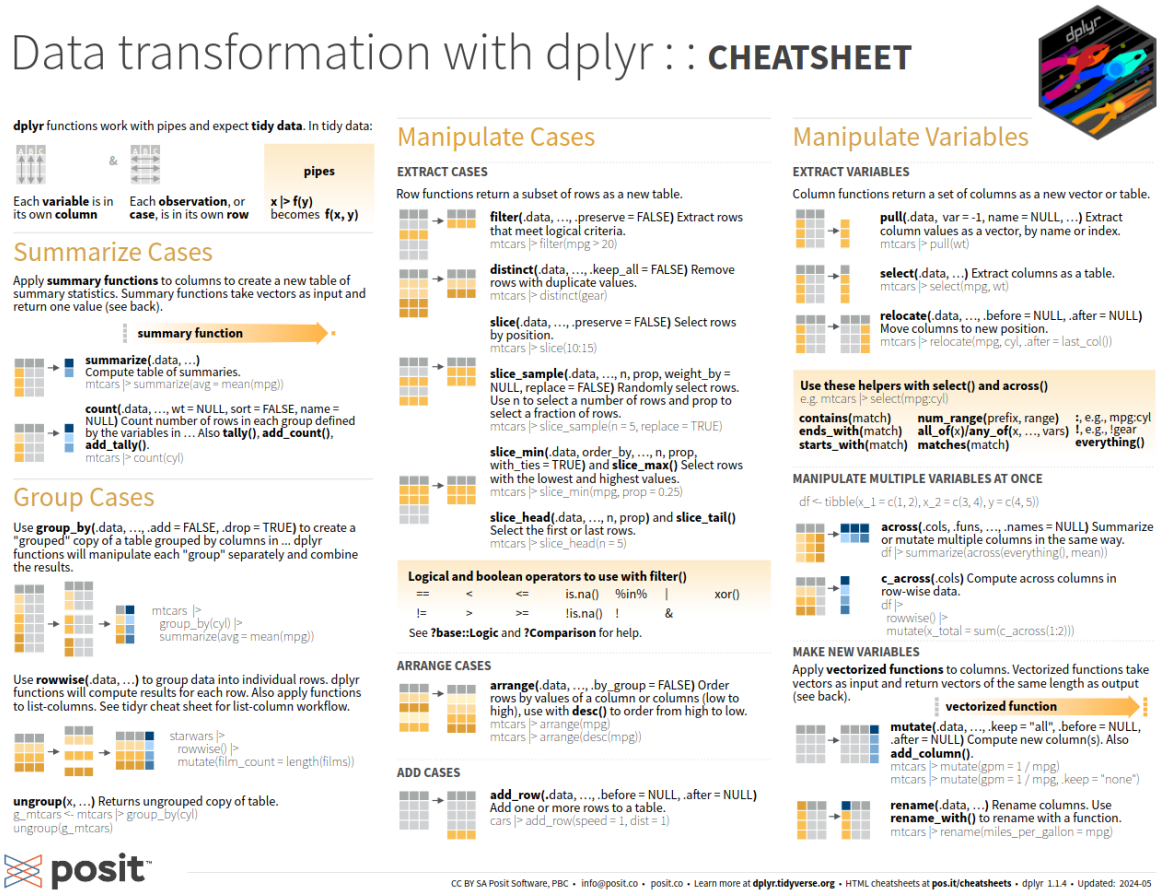
```
TRIPS_pipes
```

```
# A tibble: 18 x 4
   Municipality         Total    Car Car_perc
   <chr>                <dbl>  <dbl>    <dbl>
 1 Mafra                65811  46329     70.4
 2 Sesimbra             49370  31975     64.8
 3 Cascais             161194  96523     59.9
 4 Palmela              66428  39688     59.7
 5 Alcochete            16954   9839     58.0
 6 Setúbal             129059  70318     54.5
 7 Montijo              57164  30900     54.1
 8 Seixal              120747  63070     52.2
 9 Sintra              237445 123408     52.0
10 Oeiras              134862  66972     49.7
11 Almada              105841  49012     46.3
```

```
12 Loures                 132310  60478     45.7
13 Barreiro                52962  24160     45.6
14 Odivelas                93709  39151     41.8
15 Vila Franca de Xira    115152  47201     41.0
16 Moita                   51040  17394     34.1
17 Amadora                117727  33818     28.7
18 Lisboa                 280079  69038     24.6
```

8.9 Other dplyr functions

You can explore other `dplyr` functions and variations to manipulate data in the **dplyr cheat sheet**:



Take a particular attention to `pivot_wider` and `pivot_longer` (`tidyr` package) to transform **OD matrices** in **wide** and **long** formats.

Table 8.1: OD matrix in long format

Origins	Destinations	Trips
A	B	20
A	C	45
B	A	10
C	C	5
C	A	30

Table 8.2: OD matrix in wide format

Trips	A	B	C
A	NA	20	45
B	10	NA	NA
C	30	NA	5

9 Introduction to spatial data

Spatial data is **data that is associated with a geometry**. This geometry can be a point, a line, a polygon, or a grid.

Spatial data can be represented in many ways, such as vector data and raster data. In this tutorial, we will learn how to work with spatial data in R.

We will use the `sf` package to work with vector data, and the `dplyr` package to manipulate data.

```
library(sf)
```

```
Linking to GEOS 3.12.1, GDAL 3.8.4, PROJ 9.3.1; sf_use_s2() is TRUE
```

```
library(dplyr)
```

```
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
```

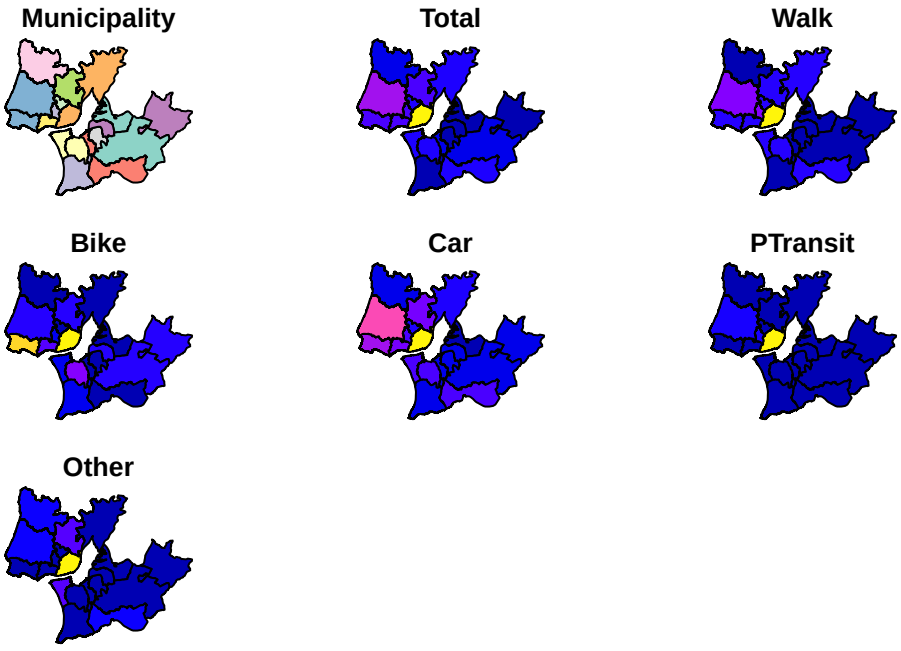
10 Interactive maps

You can plot a static map using `plof(sf)`, but you can also create interactive maps.

```
library(sf)
TRIPSgeo = st_read("data/TRIPSgeo.gpkg")
```

```
Reading layer `TRIPSgeo' from data source `D:\GIS\EITcourse\data\TRIPSgeo.gpkg' using driver
Simple feature collection with 18 features and 7 fields
Geometry type: MULTIPOLYGON
Dimension:     XY
Bounding box:  xmin: -9.500527 ymin: 38.40907 xmax: -8.490972 ymax: 39.06472
Geodetic CRS:  WGS 84
```

```
plot(TRIPSgeo)
```



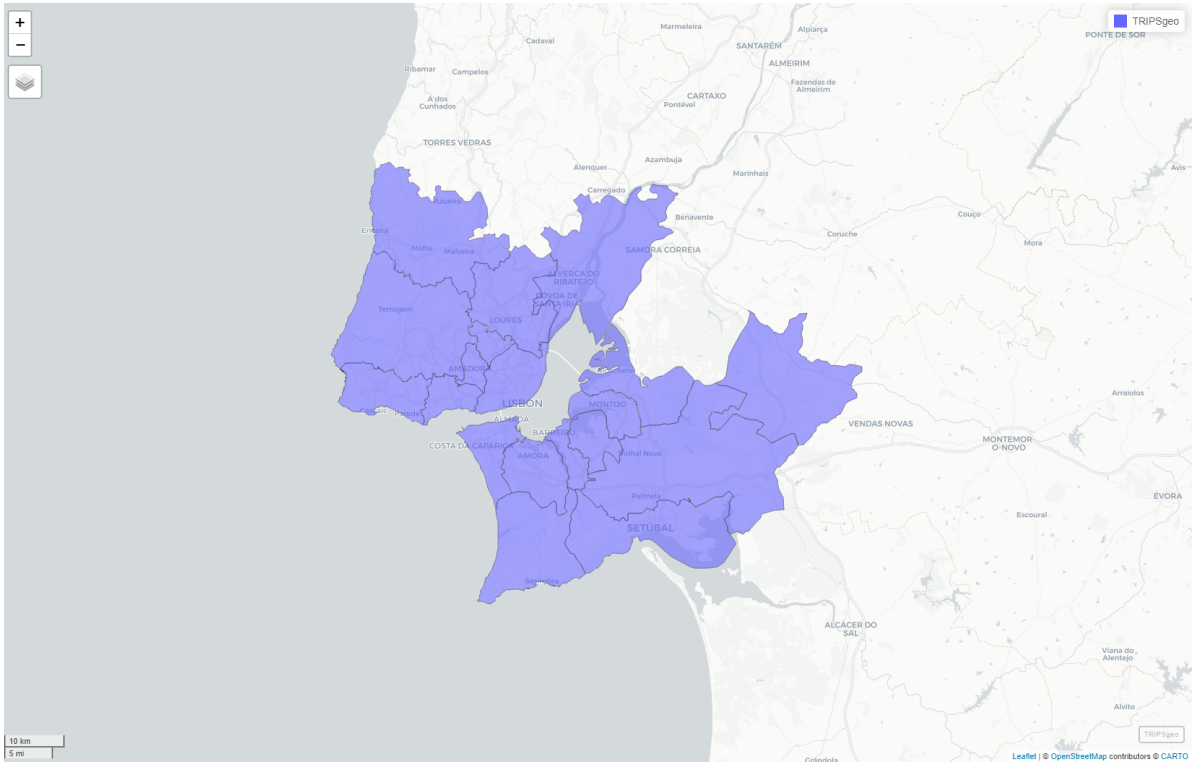
Interactive maps are useful to explore the data, as you can zoom in and out, and click on the points to see the data associated with them.

There are several R packages to create interactive maps. For instance, the `tmap` package, the `leaflet` package, and the `mapview` package.

10.1 Mapview

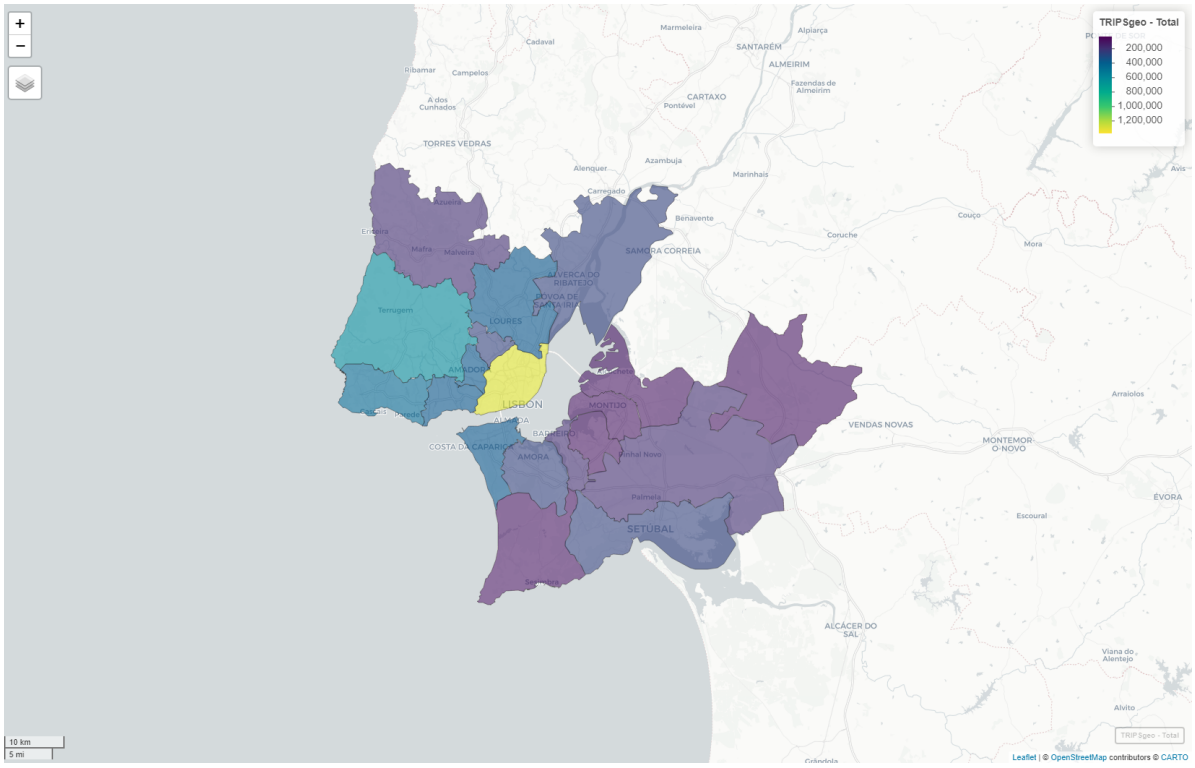
Mapview allows to create quick interactive maps, only by declaring the function `mapview()`.

```
library(mapview)
mapview(TRIPSgeo)
```



To color the points by a variable, you can use the `zcol` argument.

```
mapview(TRIPSgeo, zcol = "Total")
```

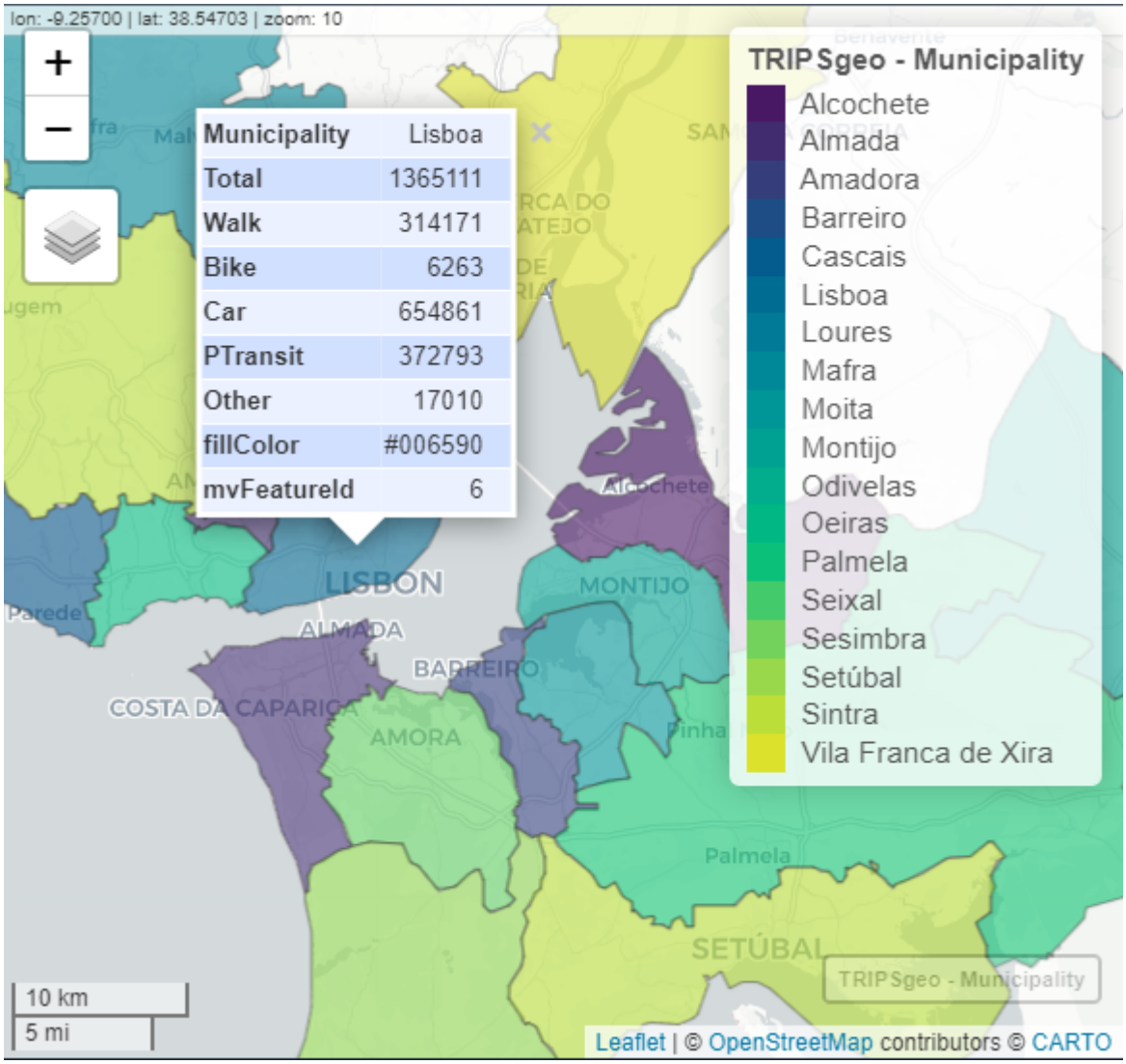


As you can see, a color palette is automatically assigned to the **continuous variable**.

Try to use a **categorical variable**.

```
mapview(TRIPSgeo, zcol = "Municipality")
```

Note that you can change the basemap, and click on the geometries to see the data associated with them.



You can go crazy with all the options that `mapview` offers. Please refer to the documentation to see all the options.

10.1.1 Export

You can export a map as an html file or image.

```
# install.packages("webshot2") # you will need this

map = mapview(TRIPSgeo, zcol = "Total") # fisrt create a objet with the desited map
```

```r
mapshot2(map, "data/map.html") # as webpage
mapshot2(map, file = "data/map.png") # as image
```

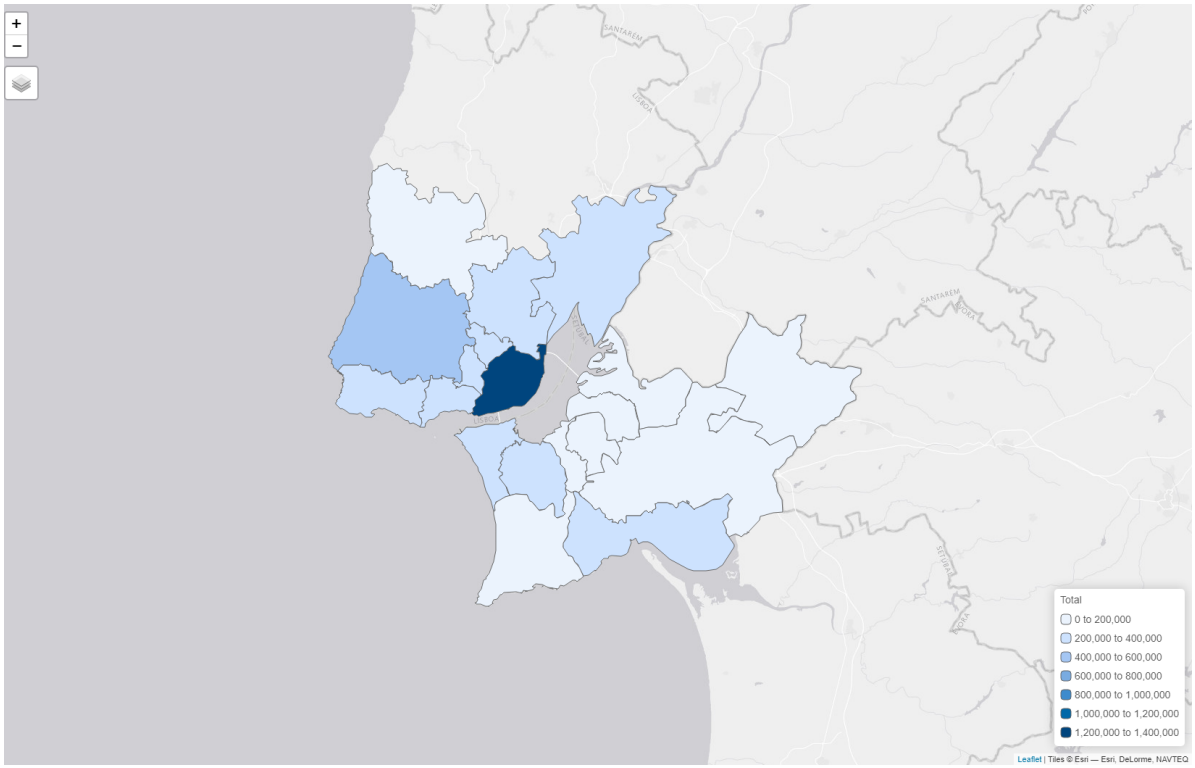
10.2 tmap

The `tmap` package is another package to create interactive maps.

To create a map, you need to declare a `tm_shape()` for the dataset you are using and a `tm_polygons()`. (or lines or other) to declare the representation you want.

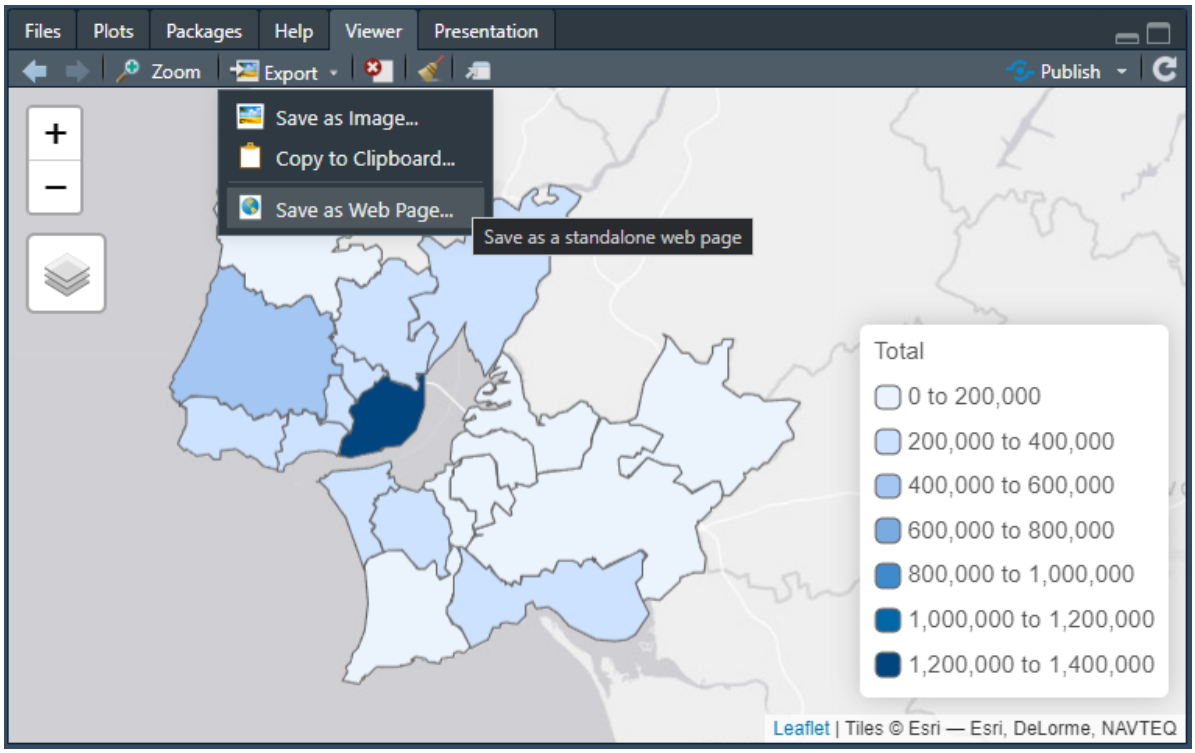
```r
# install.packages("tmap")
library(tmap)
tmap_mode("view") # by default tmap_mode is "plot" - a static map

tm_shape(TRIPSgeo) + tm_polygons("Total")
```



10.2.1 Export

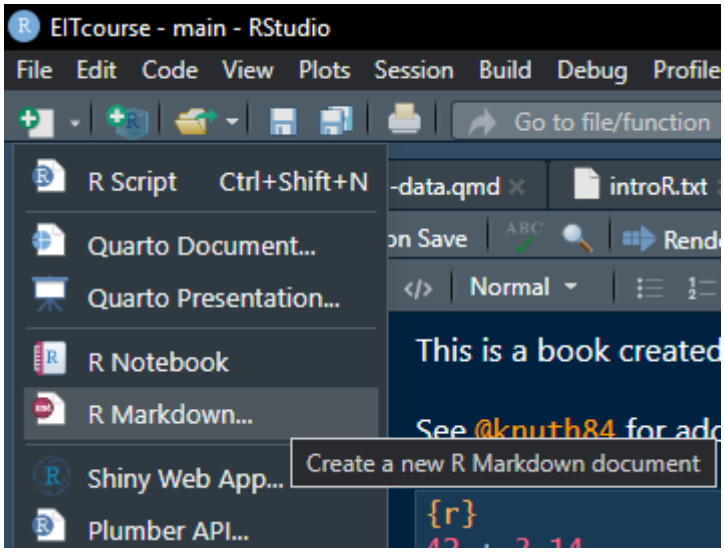
With `tmap` you can directly export the map as an `html` file or image, using the Viewer panel.



This is a most straightforward solution.

10.3 Rmarkdown

To include a map on a report, website, paper (any type), you can create an Rmarkdown file.



And include a R code chunk with a map. If the output is html, you will get an interactive map on your document!

Part II

Day 2

11 OD pairs and desire lines

This is a book created from markdown and executable code.

See Knuth (1984) for additional discussion of literate programming.

```
42 + 3.14
```

```
[1] 45.14
```

12 Georeferenced coordinates

This is a book created from markdown and executable code.

See Knuth (1984) for additional discussion of literate programming.

```
42 + 3.14
```

```
[1] 45.14
```

13 Euclidean and routing distances

We will show how to estimate euclidean distances (*as crown flights*) using `sf` package, and the distances using a road network using `r5r` package (demonstrative).

13.1 Euclidean distances

Taking the municipality centroids, or survey respondents' location

See Knuth (1984) for additional discussion of literate programming.

```
42 + 3.14
```

```
[1] 45.14
```

14 Open transportation data

This is a book created from markdown and executable code.

See Knuth (1984) for additional discussion of literate programming.

```
42 + 3.14
```

```
[1] 45.14
```

15 Urban Accessibility with R

The module "**A crash course on urban accessibility with R**", lectured by Rafael H. M. Pereira, has it's own website with materials.

> Please access it here: https://ipeagit.github.io/access_workshop_eit_2024/

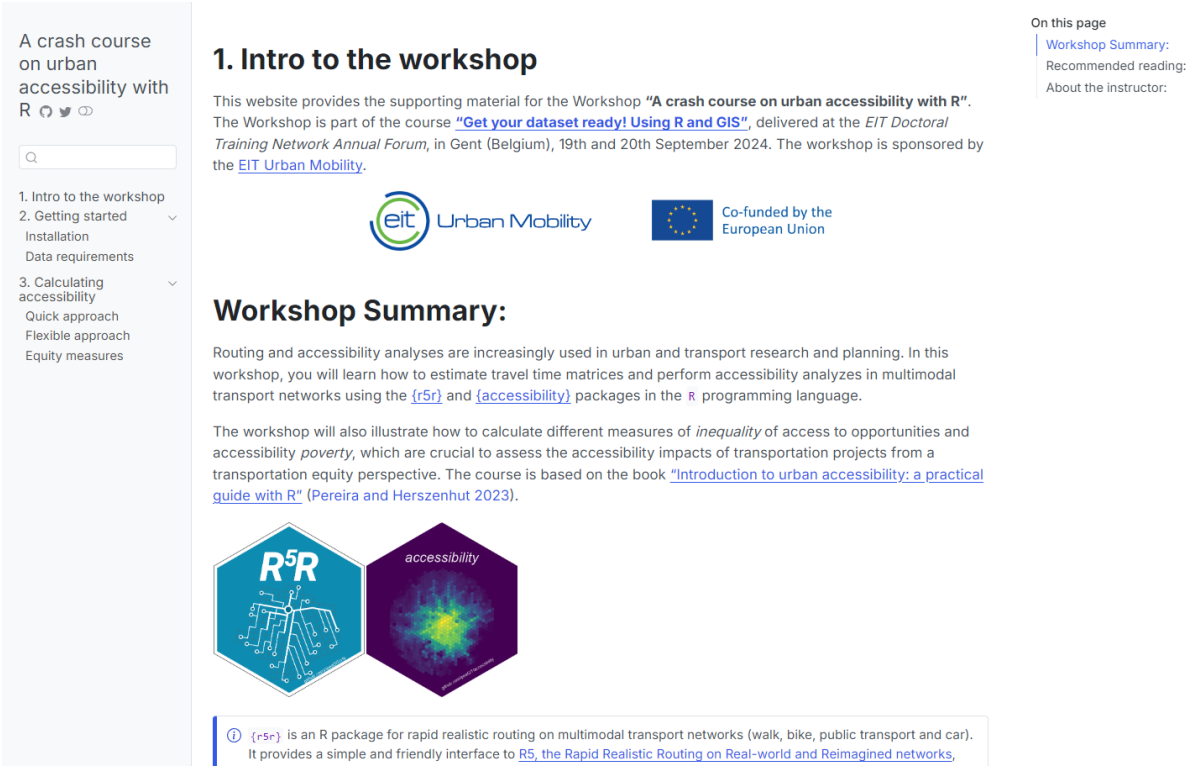


Figure 15.1: Screenshot of the website for this learning module.

About the instructor

Rafael H. M. Pereira
Head of Data Science

Institute for Applied Economic Research (Ipea), Brazil
Website | Google Scholar | Twitter | Linkedin |

Short bio

Rafael H. M. Pereira is a senior researcher in the fields of urban analytics, spatial data science and transport studies at the Institute for Applied Economic Research (Ipea), Brazil. His research looks broadly at how urban policies and technologies shape the spatial organization of cities, human mobility as well as their impacts on social and health inequalities. Some of his key contributions to the fields of urban analytics and planning involve the development of new methods and open-source computational tools to the study of urban systems and transportation networks. These contributions emerge from substantive interests around social justice and sustainability issues in urban development, with particular focus on transportation equity and inequalities in access to opportunities, and the environmental impacts of built environments and mobility patterns. With a background in Sociology and Demography, Dr. Pereira obtained his PhD in Geography from the Transport Studies Unit at Oxford University.

16 Introduction

This is a book created from markdown and executable code.

See Knuth (1984) for additional discussion of literate programming.

```
data = readRDS("data/TRIPSmode_mun.Rds")
data_walk = data[,c("Origin_mun", "Destination_mun", "Walk")]
head(data_walk)
```

```
  Origin_mun Destination_mun Walk
1  Alcochete       Alcochete 6833
2  Alcochete          Almada    0
3  Alcochete         Amadora    0
4  Alcochete        Barreiro    0
5  Alcochete         Cascais    0
6  Alcochete          Lisboa   69
```

References

Instituto National de Estatística. 2018. "Mobilidade e Funcionalidade Do Território Nas Áreas Metropolitanas Do Porto e de Lisboa: 2017." Lisboa. https://www.ine.pt/xportal/xmain?xpid=INE&xpgid=ine_publicacoes&PUBLICACOESpub_boui=349495406&PUBLICACOESmodo=2&xlang=pt.

Knuth, Donald E. 1984. "Literate Programming." *Comput. J.* 27 (2): 97–111. https://doi.org/10.1093/comjnl/27.2.97.