Get your dataset ready!

Using R and GIS

Rosa Félix Gabriel Valença Rafael Pereira

2024-09-11

Table of contents

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.

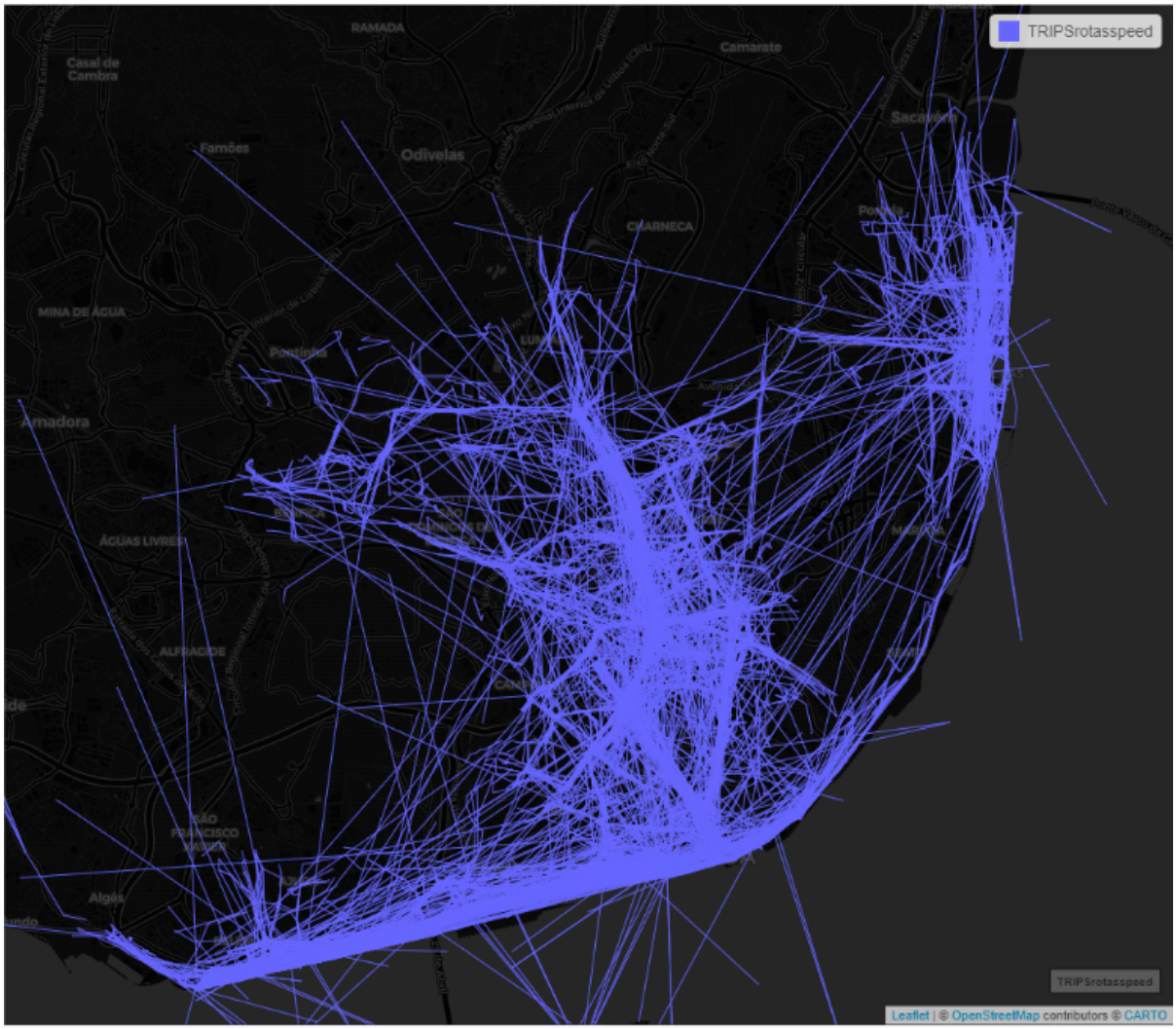


Figure 1.1: E-Scooter trip data in Lisbon. How to deal with it?

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 (2023) Introduction to R.
- Lovelace, Nowosad, and Muenchow (2024) Geocomputation with R.
- R. H. Pereira and Herszenhut (2023) Introduction to urban accessibility: a practical guide with R.

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 Contents

2.1.1 Day 1

Morning

We will start by a brief introduction to this course, followed by an introduction to programming techniques and data structures.

Then, we will install R and RStudio, and and the required R packages for this course, as in Software section.

After having everything setup, we will start with the R basics, with examples and exercises.

Afternoon

In the afternoon, we will focus on data manipulation, using the dplyr package to select, filter, left-join, group and summarize datasets.

Then, we will introduce GIS and spatial data, learning how to importing and visualize vector data.

Finally, we will learn how to create cool interactive maps using mapview and R markdown.

2.1.2 Day 2

Morning

We will start the day by estimating the different types of centroids of transport zones.

After this, the natural next step is to create desire lines from orgins and destinations of the transport zones.

We will then learn how to estimate euclidean and routing distances for the desire-lines, using transport networks.

Afternoon

In the second afternoon, we will briefly learn where to find and extract open transportation data, such as OpenStreetMap and GTFS.

Then, we will learn how to perform accessibility analysis, using the r5r package.

And finally, to wrap up all this topics, we will have a group exercise using other complex datasets, where you will apply the knowledge learned during the course.

2.2 Detailed schedule

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)
15.30	*Coffee break*
15.45	Introduction to GIS and spatial data: import create vector data
16.30	View and export interactive maps
17.00	*End of day 1*

Day 2	
9.30	Centroids of transport zones
10.15	Desire-lines from OD pairs and transport zones
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)
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*

2.3 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
```

2.4 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

3 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.

3.1 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.

3.1.1 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.

3.1.2 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.

3.1.3 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.

3.2 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.

3.2.1 Windows 10/11

Download RStudio 2024.04 and run the executable file.

3.2.2 MacOS

Download RStudio 2024.04 and install it as usual.

3.2.3 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:

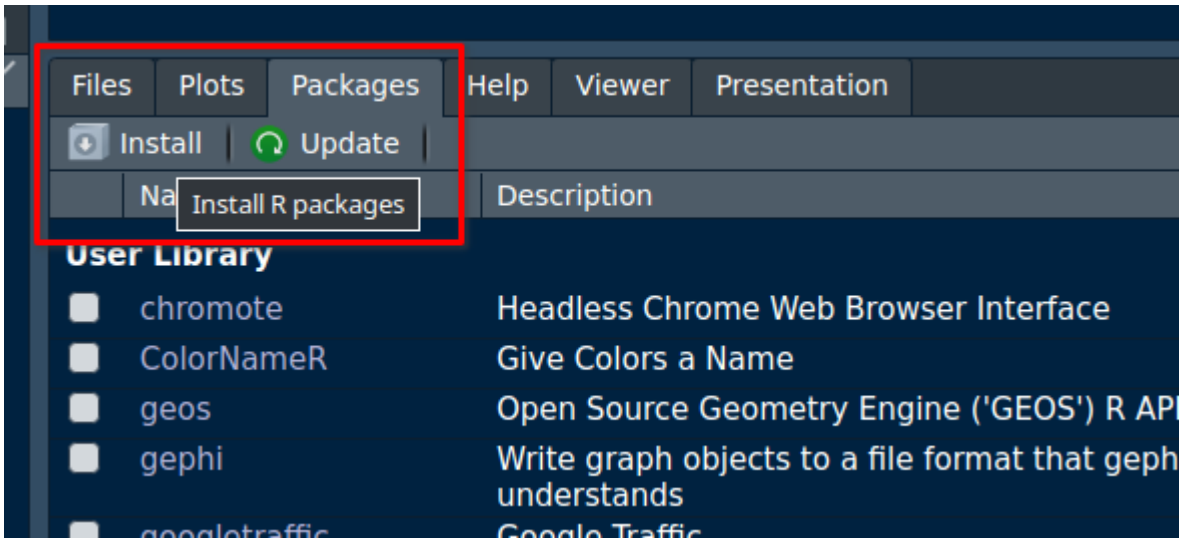
```
sudo apt install libssl-dev libclang-dev
wget https://download1.rstudio.org/electron/jammy/amd64/rstudio-2024.04.2-764-amd64.deb
sudo dpkg -i rstudio*
rm -v rstudio*
```

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

3.3 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("sf")
install.packages("mapview")
install.packages("rmarkdown")
install.packages("centr")
install.packages("od")

install.packages(c("remotes", "devtools", "usethis")) # optional
install.packages("osmextract") # optional
install.packages("stplanr") # optional
```

3.4 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)
```

3.4.1 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.

3.4.2 Windows and MacOS

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

3.4.3 Ubuntu

Install it via the terminal:

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

4 R basics

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

4.1 Math operations

4.1.1 Sum

```
1+1
```

```
[1] 2
```

4.1.2 Subtraction

```
5-2
```

```
[1] 3
```

4.1.3 Multiplication

```
2*2
```

```
[1] 4
```

4.1.4 Division

```
8/2
```

```
[1] 4
```

4.1.5 Round the number

```
round(3.14)
```

```
[1] 3
```

```
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.

4.2 Basic shortpaths

4.2.1 Perform Combinations

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

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

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

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

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

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

4.2.3 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)
```

Look at the first row

```
table_example[1,] #rows and columns start from 1 in R, differently from Python which starts
```

```
  modes Trips
1   car   200
```

Look at first row and column

```
table_example[1,1]
```

```
[1] "car"
```

4.3 Practical exercise

Dataset: the number of trips between all municipalities in the Lisbon Metropolitan Area, Portugal (INE 2018).

4.3.1 Import dataset

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

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

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

4.3.2 Take a first look at the data

Summary statistics

```
summary(data)
```

```
   Origin           Destination            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     : chr  "Alcochete" "Alcochete" "Alcochete" "Alcochete" ...
 $ Destination: 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
```

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

```
     Origin Destination 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)

```
nrow(data)
```

```
[1] 315
```

```
ncol(data)
```

```
[1] 8
```

Open the dataset

```
View(data)
```

4.3.3 Explore the data

Check the total number of trips

Use $ to select a variable of the data

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

```
[1] 5299853
```

Percentage of car trips related to the total

```
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
```

4.3.4 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 == "Lisboa",]
```

Filter trips with origin **different** from Lisbon

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

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

```
data_in_Out_Lisbon = data[data$Origin == "Lisboa" & data$Destination == "Lisboa",]
```

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" "Total"       "Walk"        "Bike"        "Car"
[6] "PTransit"    "Other"       "Active"
```

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

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

4.3.5 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.
```

4.3.6 Import data

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

5 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")
```

> **!** Important
>
> 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.

> There are several ways to reach the same solution. Here we present only one of them.

5.1 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)
```

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

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
```

5.1.1 Using pipes!

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

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

5.2 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)
```

```
   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

5.3 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
```

5.4 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.

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

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

```r
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`)

5.4.1 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.

```r
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.

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

```
[1] 0 1
```

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

```
  0   1
188  48
```

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

```
 No Yes
188  48
```

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

5.5 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:

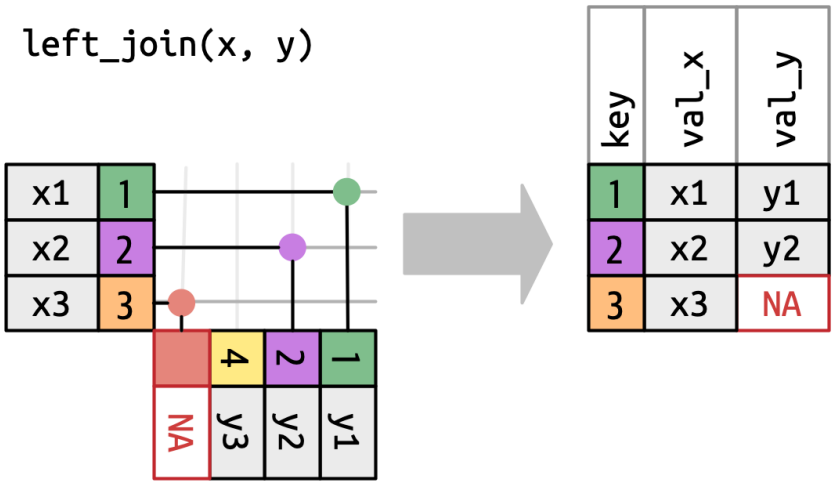


Figure 5.1: A visual representation of the left join where every row in x appears in the output.Source: R for Data Science.

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`.

5.6 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.

5.7 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.

5.8 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) |>
  ungroup() |>

  arrange(desc(Car_perc))
```

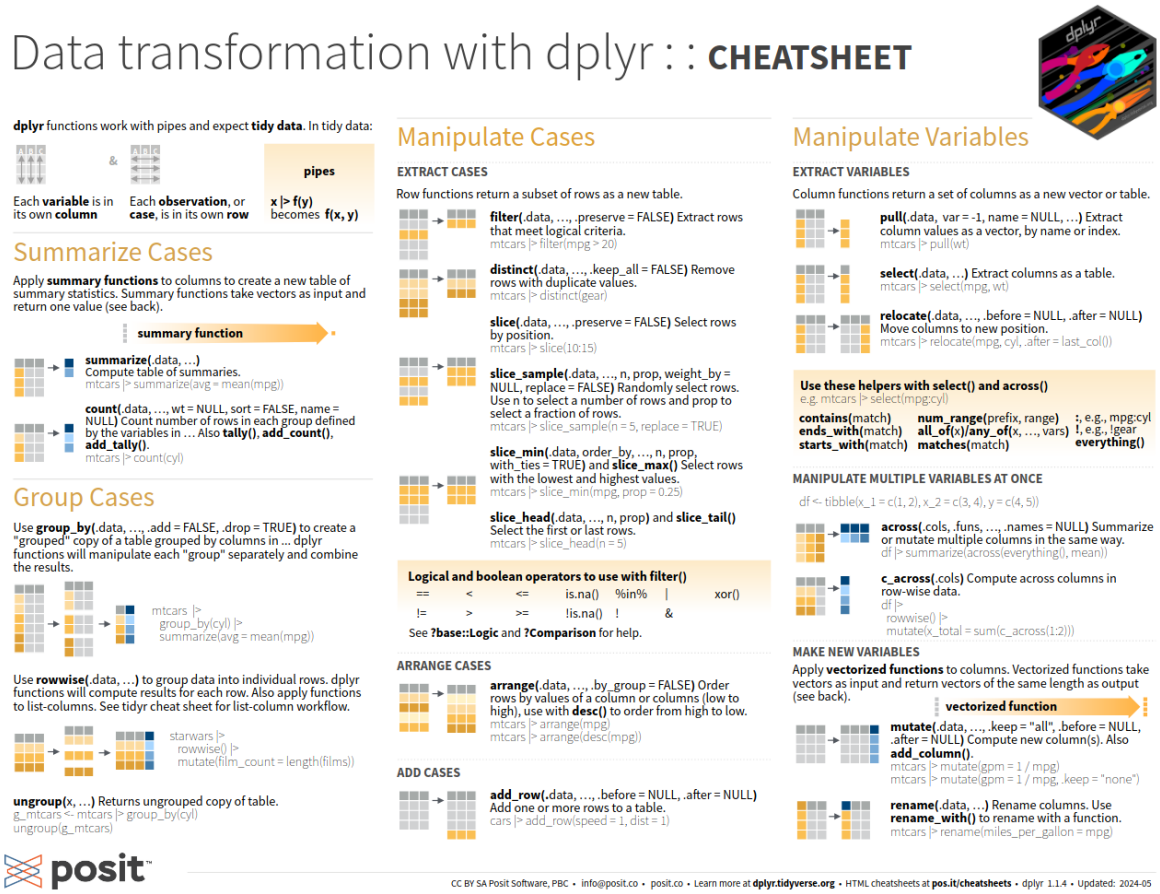
With this code we will have a table with the total number of intercity trips, by municipality, with their names instead of codes, arranged by the percentage of car trips.

```
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
```

5.9 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 5.1: OD matrix in long format

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

Table 5.2: OD matrix in wide format

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

6 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)
library(dplyr)
```

The `sf` package is a powerful package for working with spatial data in R. It includes hundreds of functions to deal with spatial data (Pebesma and Bivand 2023).

6.1 Import vector data

Download and open `Municipalities_geo.gpkg` under EITcourse/data repository.

Within the `sf` package, we use the `st_read()` to read spatial features.

```
Municipalities_geo = st_read("data/Municipalities_geo.gpkg")
```

```
Reading layer `Municipalities_geo' from data source
  `D:\GIS\EITcourse\data\Municipalities_geo.gpkg' using driver `GPKG'
Simple feature collection with 18 features and 1 field
Geometry type: MULTIPOLYGON
Dimension:     XY
Bounding box:  xmin: -9.500527 ymin: 38.40907 xmax: -8.490972 ymax: 39.06472
Geodetic CRS:  WGS 84
```

You can also open directly from url from github. Example:

```
url = "https://github.com/U-Shift/EITcourse/raw/main/data/Municipalities_geo.gpkg"
Municipalities_geo = st_read(url)
```

6.1.1 Projected vs Geographic Coordinate Systems

A **projected coordinate system** is a flat representation of the Earth's surface. A **geographic coordinate system** is a spherical representation of the Earth's surface.

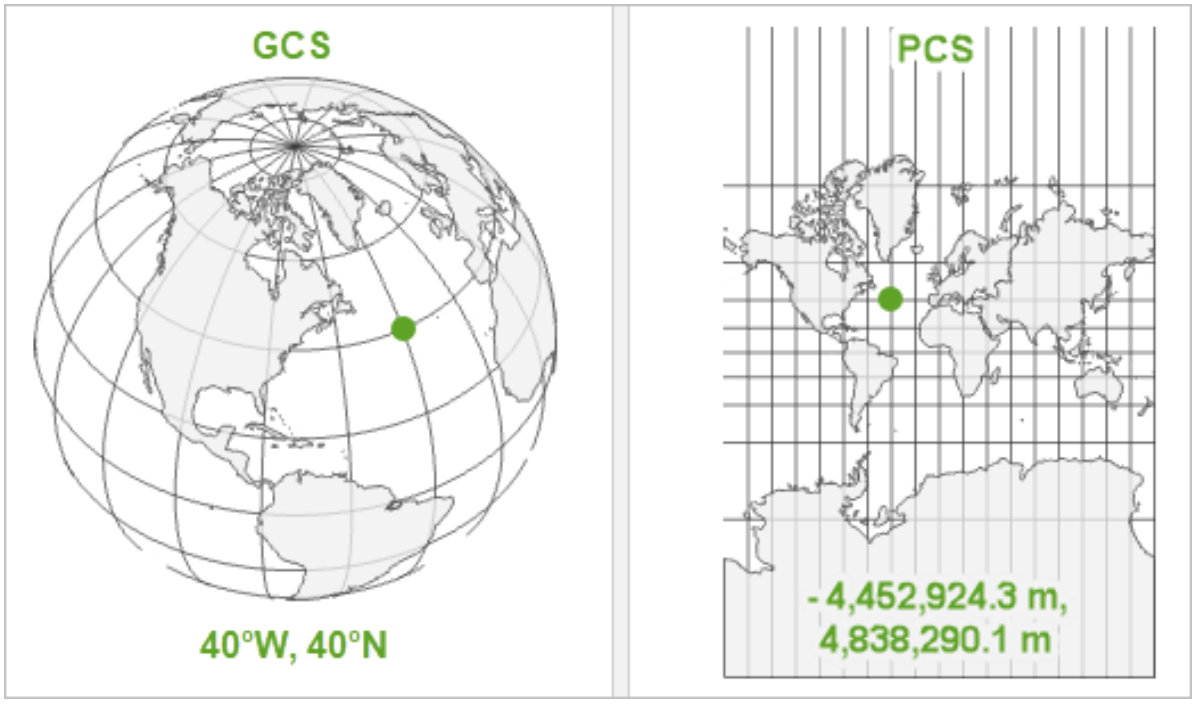


Figure 6.1: Source: ESRI

The `st_crs()` function can be used to check the **coordinate reference system** of a spatial object.

```
st_crs(Municipalities_geo)
```

```
Coordinate Reference System:
  User input: WGS 84
  wkt:
GEOGCRS["WGS 84",
```

```
    ENSEMBLE["World Geodetic System 1984 ensemble",
        MEMBER["World Geodetic System 1984 (Transit)"],
        MEMBER["World Geodetic System 1984 (G730)"],
        MEMBER["World Geodetic System 1984 (G873)"],
        MEMBER["World Geodetic System 1984 (G1150)"],
        MEMBER["World Geodetic System 1984 (G1674)"],
        MEMBER["World Geodetic System 1984 (G1762)"],
        MEMBER["World Geodetic System 1984 (G2139)"],
        ELLIPSOID["WGS 84",6378137,298.257223563,
            LENGTHUNIT["metre",1]],
        ENSEMBLEACCURACY[2.0]],
    PRIMEM["Greenwich",0,
        ANGLEUNIT["degree",0.0174532925199433]],
    CS[ellipsoidal,2],
        AXIS["geodetic latitude (Lat)",north,
            ORDER[1],
            ANGLEUNIT["degree",0.0174532925199433]],
        AXIS["geodetic longitude (Lon)",east,
            ORDER[2],
            ANGLEUNIT["degree",0.0174532925199433]],
    USAGE[
        SCOPE["Horizontal component of 3D system."],
        AREA["World."],
        BBOX[-90,-180,90,180]],
    ID["EPSG",4326]]
```

WGS84 is the most common geographic coordinate system, used in GPS, and EPSG:**4326** is code for it.

If we want to project the data to a projected coordinate system, to use **metric units** instead of degrees, we can use the `st_transform()` function.

In this case, the EPGS:**3857** is the code for the Pseudo-Mercator coordinate system.

```
Municipalities_projected = st_transform(Municipalities_geo, crs = 3857)
```

Now see the differences when calling `Municipalities_geo` and `Municipalities_projected`.

6.2 Join geometries to data frames

Import `TRIPSmun.Rds` file and check data class-

```
TRIPSmun = readRDS("data/TRIPSmun.Rds")
class(TRIPSmun)
```

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

```
class(Municipalities_geo)
```

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

To join the geometries from the `Municipalities_geo` to the data frame, we can use the `left_join()` function from the `dplyr` package.

```
TRIPSgeo =
  TRIPSmun |>
  left_join(Municipalities_geo)

class(TRIPSgeo)
```

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

As you can see, this **does not make the object a spatial feature**. To do this, we need to use the `st_as_sf()` function.

```
TRIPSgeo = TRIPSgeo |> st_as_sf()
class(TRIPSgeo)
```

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

Now we have a spatial feature with the data frame.

6.3 Create spatial data from coordinates

The `st_as_sf()` function can also be used to create a spatial feature from a data frame with coordinates. In that case, we need to specify the columns with the coordinates.

We will use survey data (in `.txt`) with the participants' home latitude/longitude coordinates to create a spatial feature.

```
SURVEY = read.csv("data/SURVEY.txt", sep = "\t") # tab delimiter
class(SURVEY)
```

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

```
SURVEYgeo = st_as_sf(SURVEY, coords = c("lon", "lat"), crs = 4326) # create spatial feature
class(SURVEYgeo)
```

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

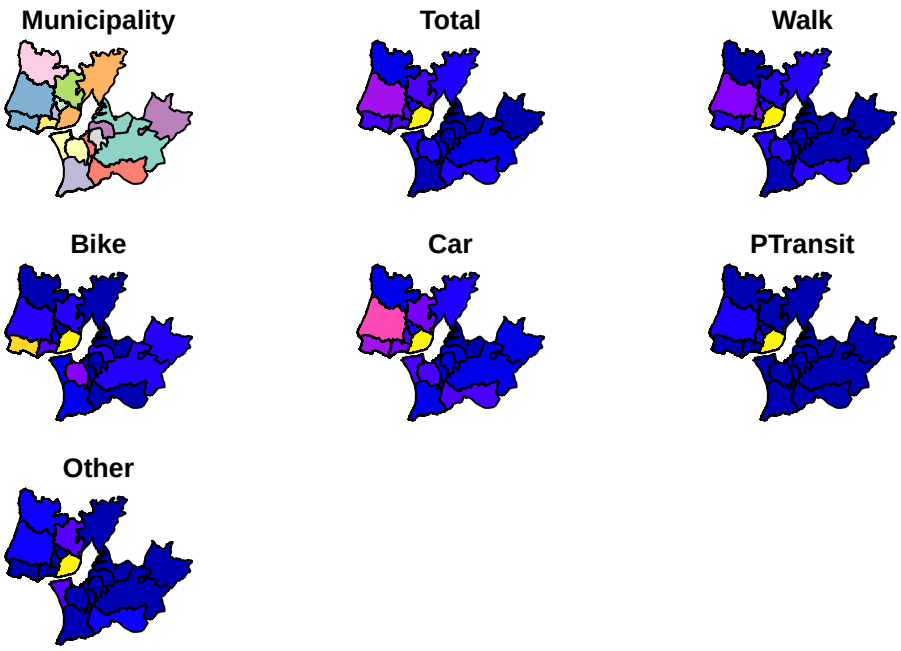
We can also set the **crs** of the spatial feature on the fly.

Check the differences between both data variables.

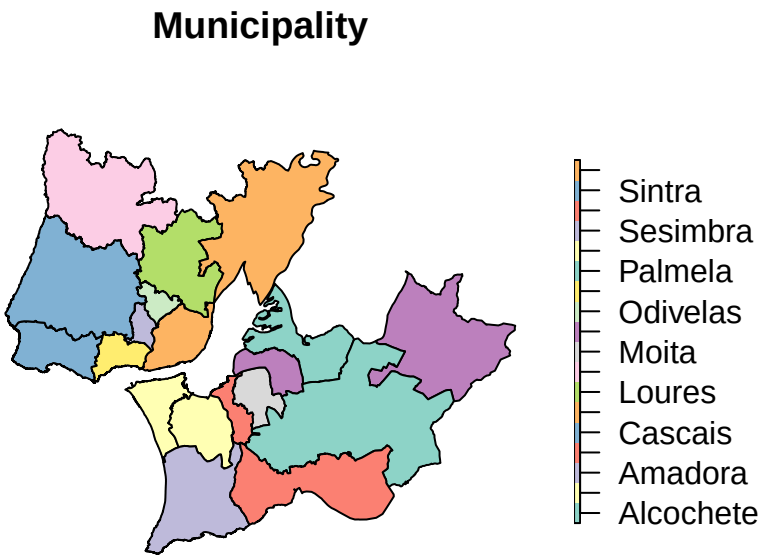
6.4 Visuzlize spatial data

Represent Transport Zones with Total and Car, using `plot()`.

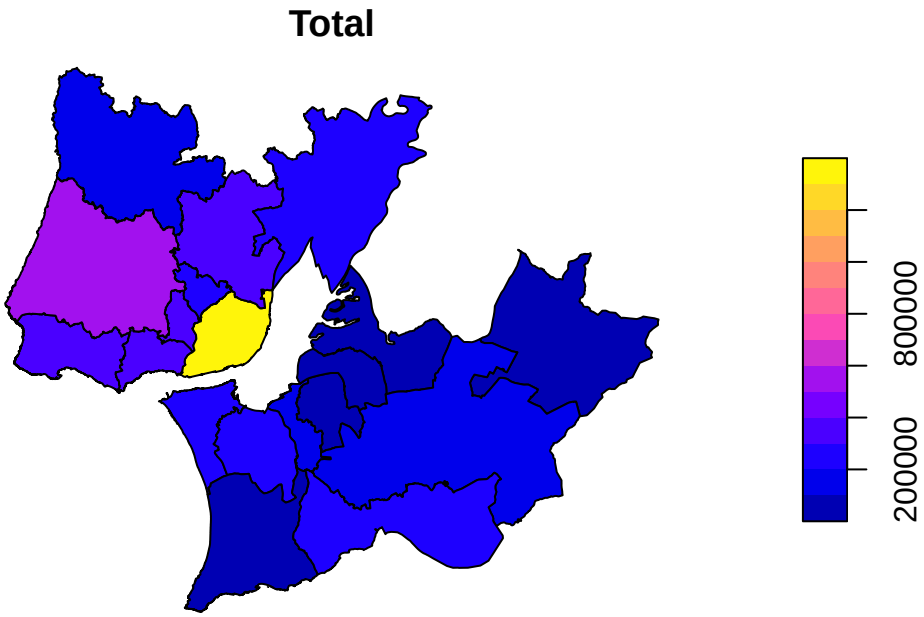
```
plot(TRIPSgeo) # all variables
```



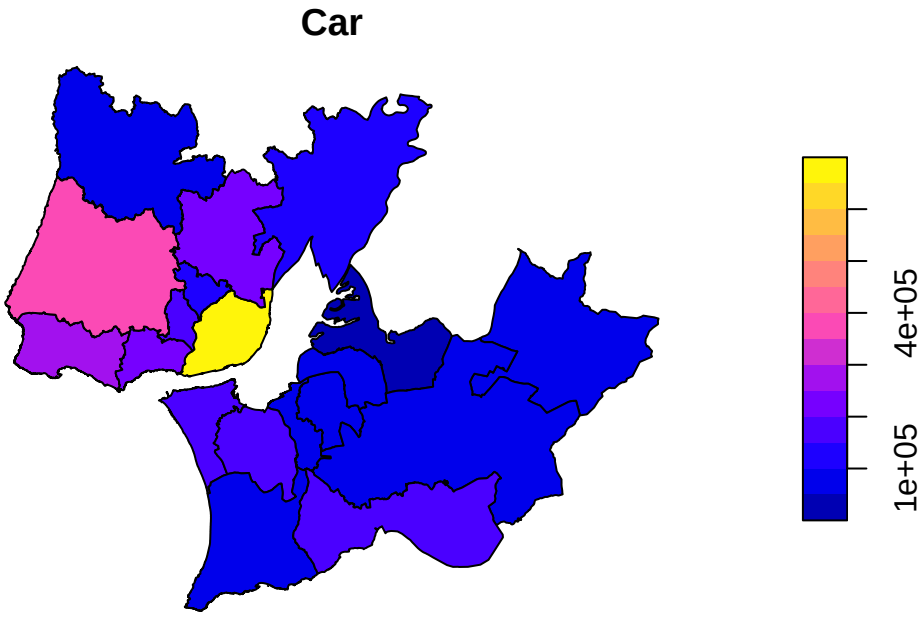
```
plot(TRIPSgeo["Municipality"])
```



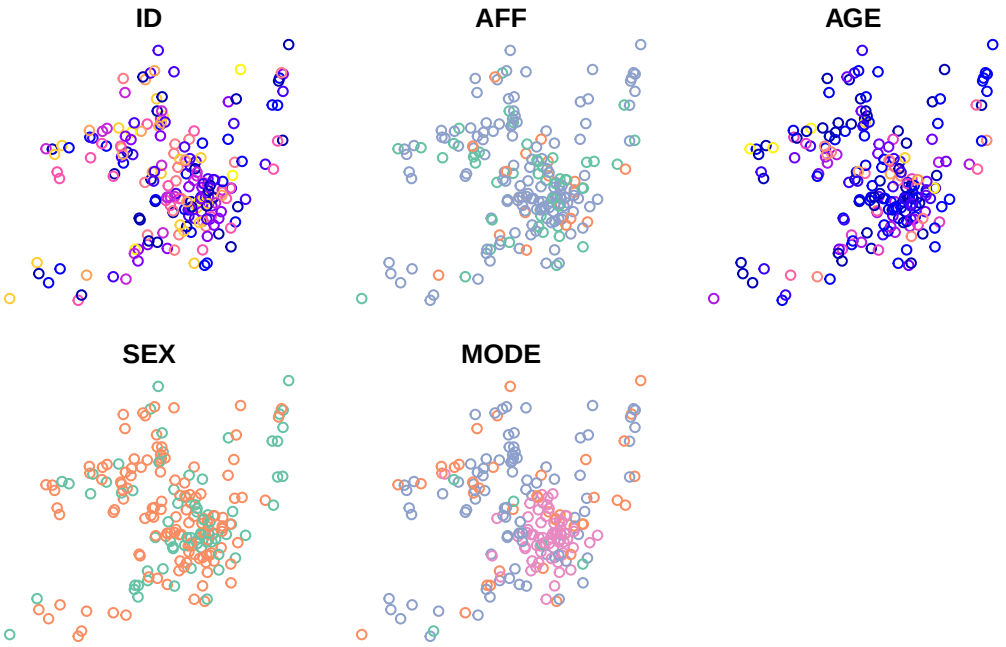

```
plot(TRIPSgeo["Total"])
```

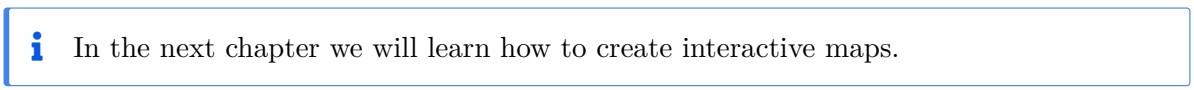



```
plot(TRIPSgeo["Car"])
```




```
# plot pointy data
plot(SURVEYgeo)
```





6.5 Export spatial data

You can save your spatial data in different formats using the function `st_write()`, such as shapefiles (ESRI), GeoJSON, and GeoPackage.

This is also useful to convert spatial data between formats.

```
st_write(TRIPSgeo, "data/TRIPSgeo.gpkg") # as geopackage
st_write(TRIPSgeo, "data/TRIPSgeo.shp") # as shapefile
st_write(TRIPSgeo, "data/TRIPSgeo.geojson") # as geojson
st_write(TRIPSgeo, "data/TRIPSgeo.csv", layer_options = "GEOMETRY=AS_WKT") # as csv, with WK
```

⚠ If you already have a file with the same name, you can use the `delete_dns = TRUE` argument to overwrite it.

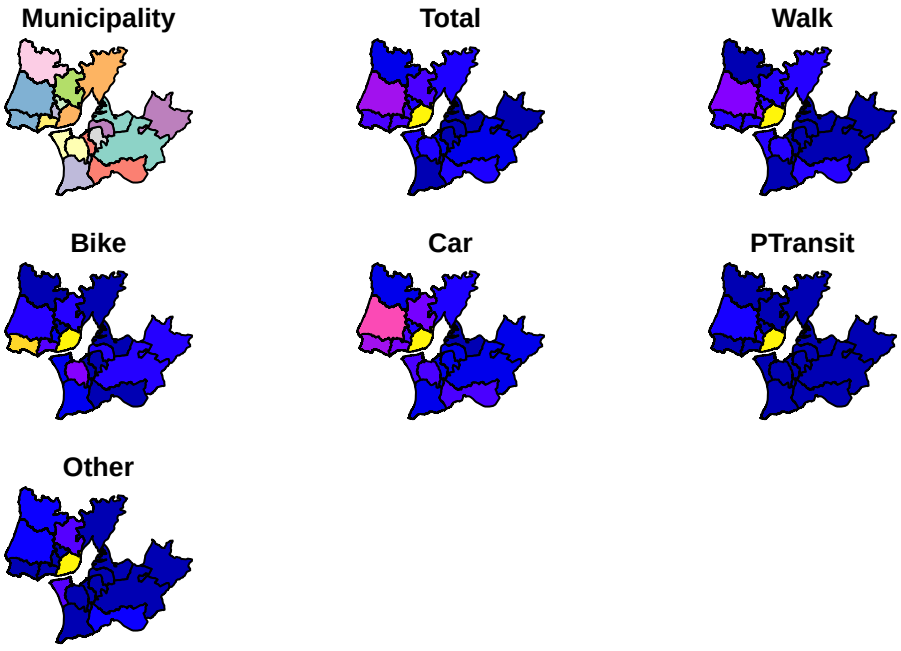
7 Interactive maps

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)
```



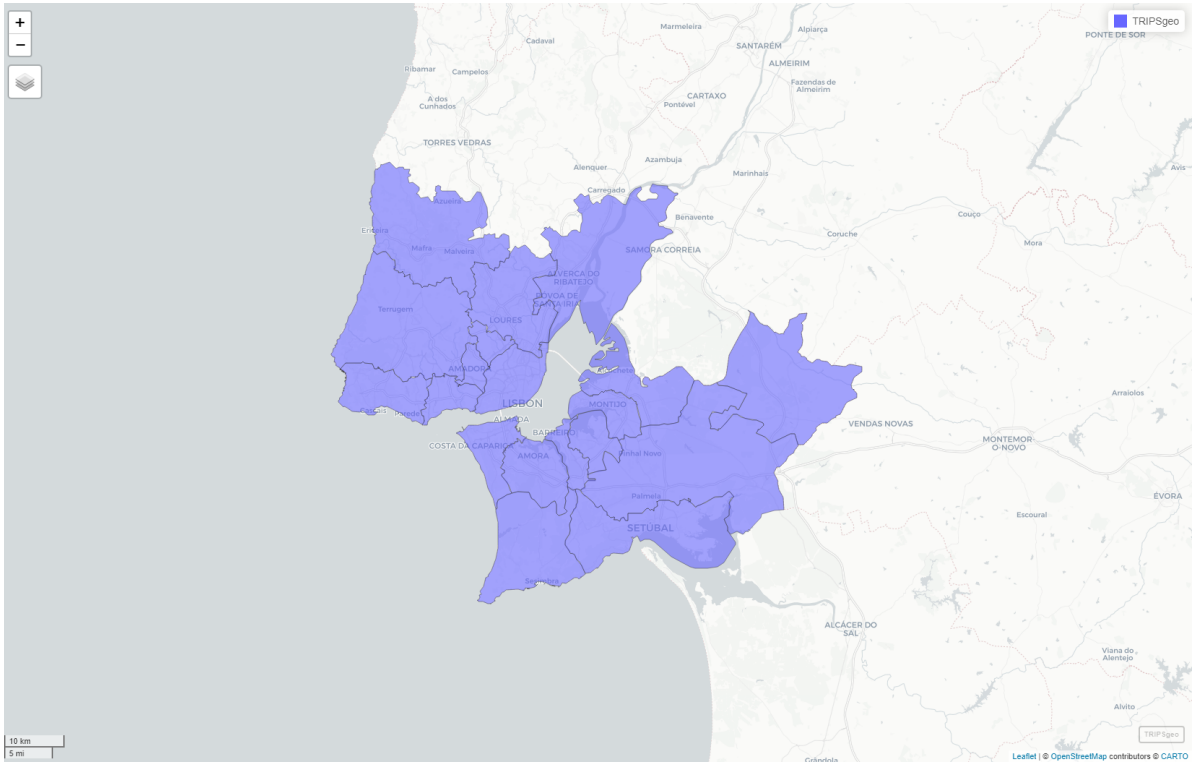
Interactive maps are useful to explore the data, as you can zoom in and out, and click on the points to see the data associated with them.

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

7.1 Mapview

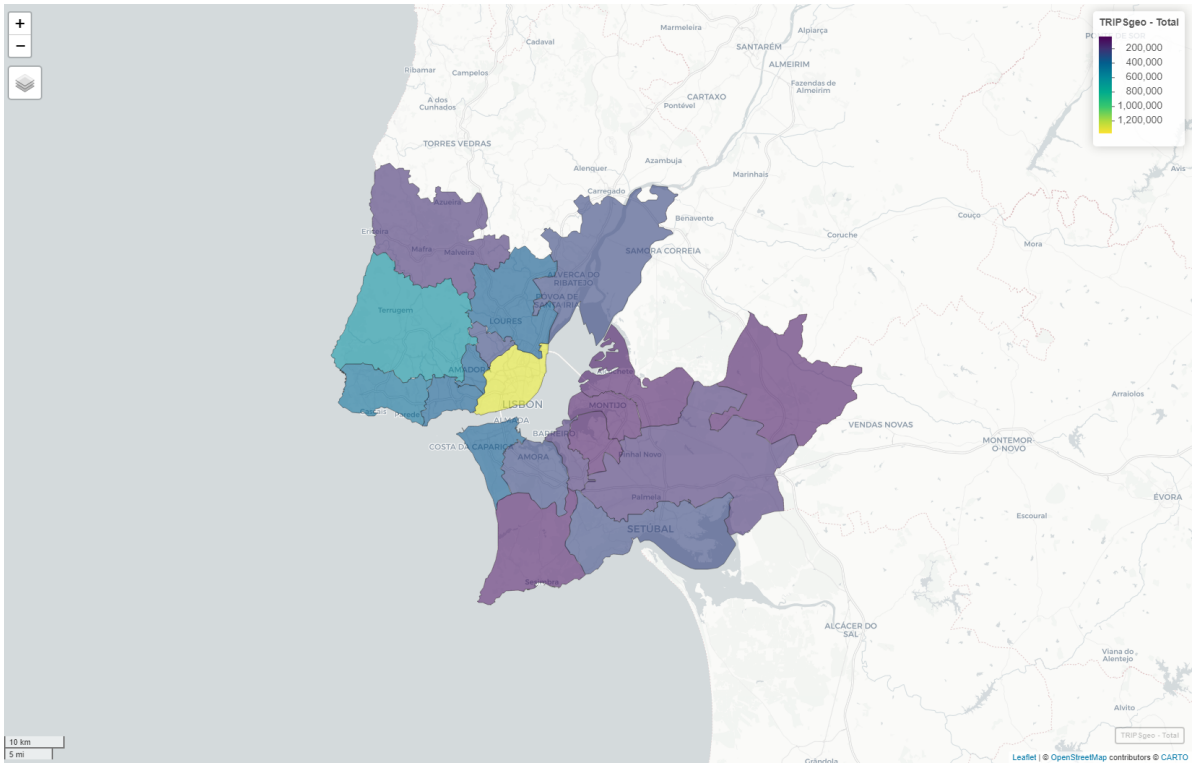
Mapview allows to create quick interactive maps, only by declaring the function `mapview()`.

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



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

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

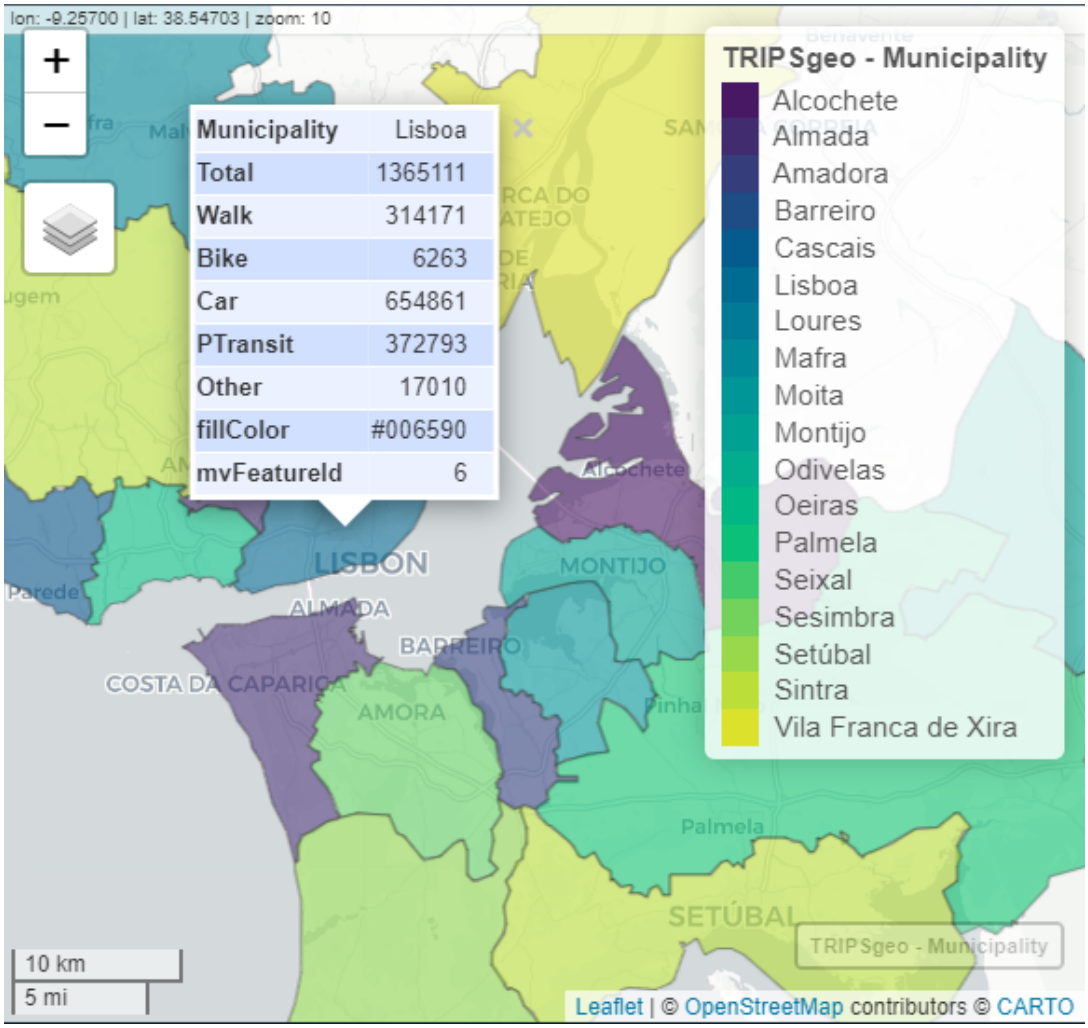


As you can see, a color palette is automatically assigned to the **continuous variable**.

Try to use a **categorical variable**.

```
mapview(TRIPSgeo, zcol = "Municipality", alpha.regions = 0.4) # also add transparency)
```

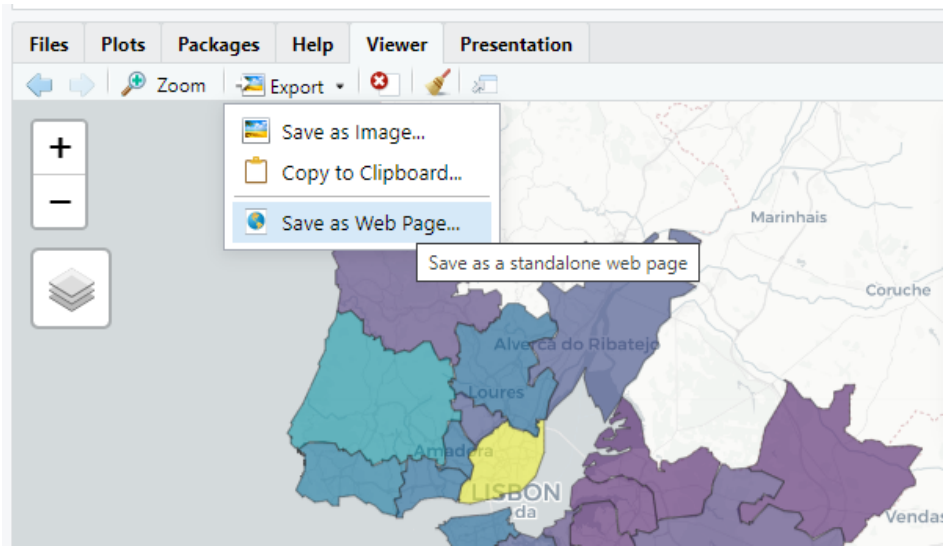
> Note that you can change the **basemap**, and click on the geometries to **see the data** associated with them.



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

7.1.1 Export

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



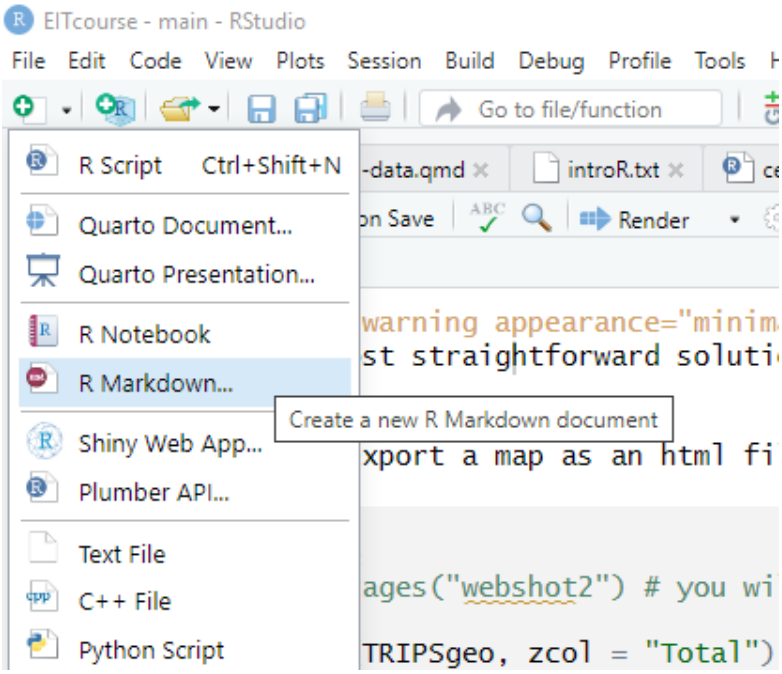
This is the most straightforward solution.

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

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

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

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

7.2 Rmarkdown

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



And include a R code chunk (`ctrl + alt + i`) with a map. If the output is html, you will get an interactive map on your document!

Part II

Day 2

8 Centroids of transport zones

In this section we will calculate the geometric and the weighted centroids of transport zones.

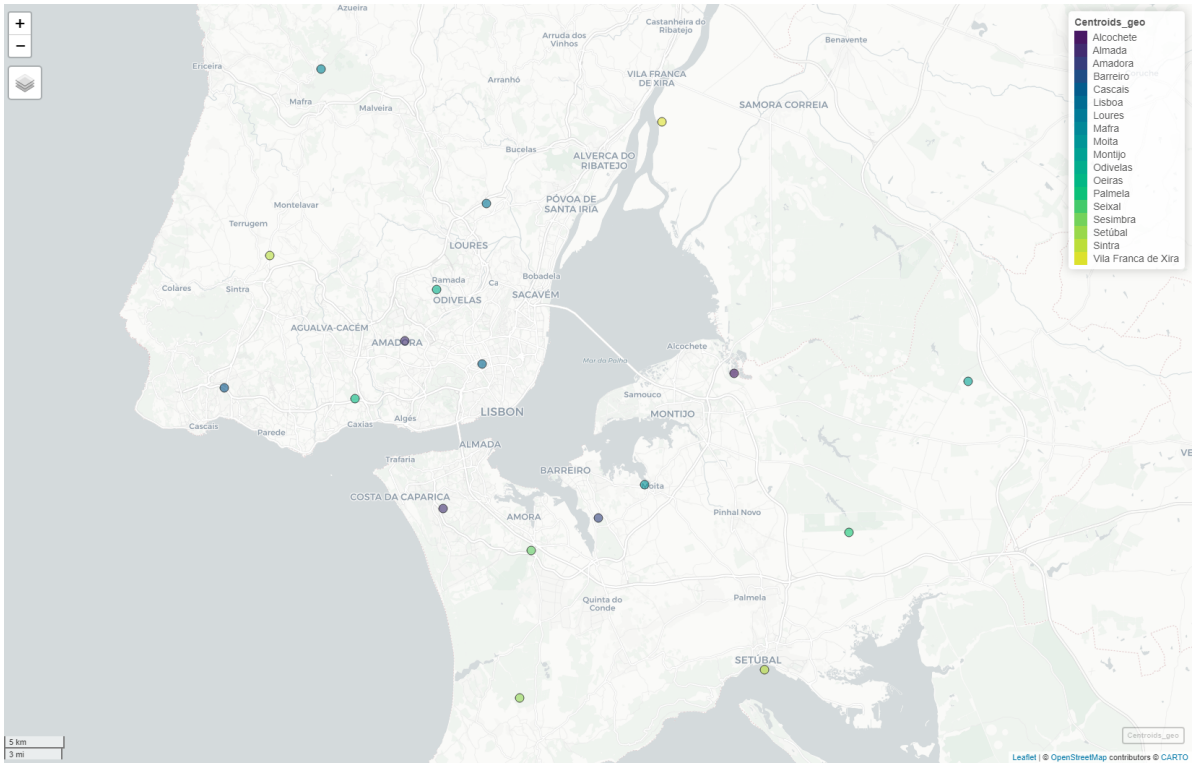
8.1 Geometric centroids

Taking the `Municipalities_geo` data from the previous section, we will calculate the geometric centroids, using the `st_centroid()` function.

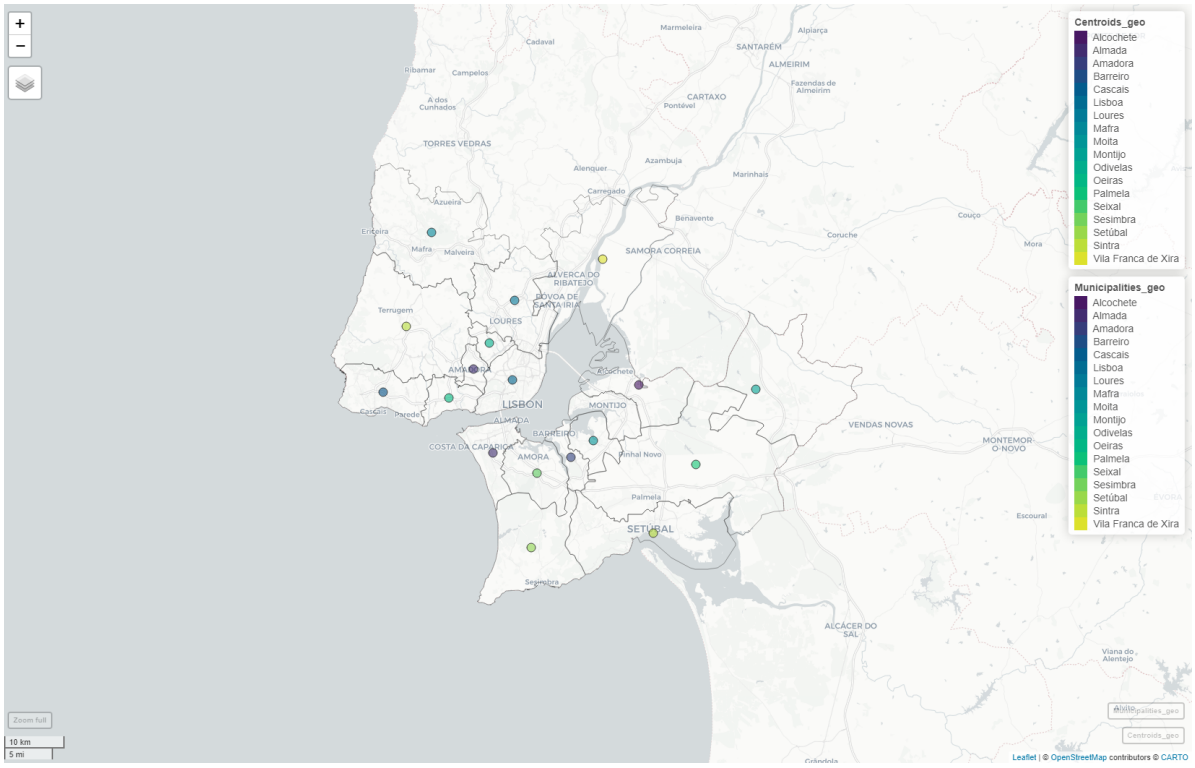
```
library(dplyr)
library(sf)
library(mapview)

Municipalities_geo = st_read("data/Municipalities_geo.gpkg", quiet = TRUE)

Centroids_geo = st_centroid(Municipalities_geo)
```

This creates points at the geometric center of each polygon.

```
mapview(Centroids_geo)
```




```
mapview(Centroids_geo) + mapview(Municipalities_geo, alpha.regions = 0) # both maps, with ful
```



But… is this the best way to represent the center of a transport zone?

These results may be biased by the shape of the polygons, and not represent where activities are. Example: lakes, forests, etc.

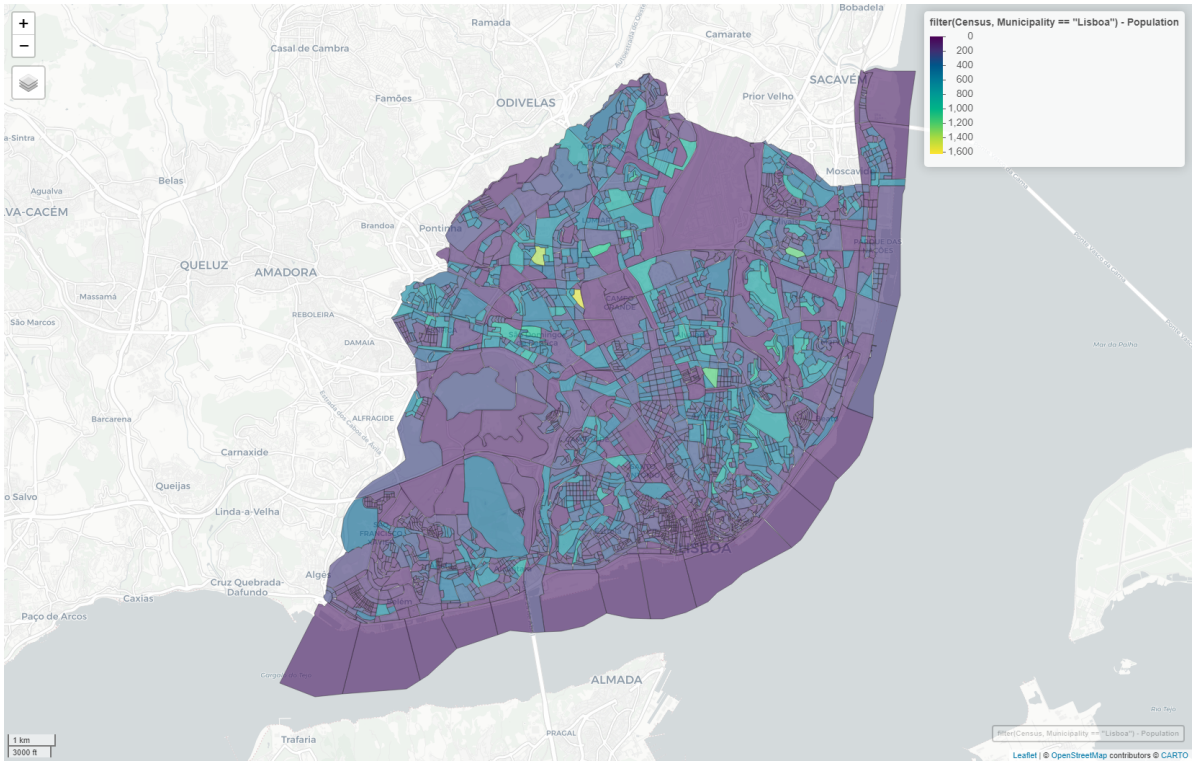
To overcome this, we can use **weighted centroids**.

8.2 Weighted centroids

We will weight centroids of transport zones by population and by number of buildings.

For this, we will need the census data (INE 2022).

```
Census = st_read("data/census.gpkg", quiet = TRUE)

mapview(Census |> filter(Municipality == "Lisboa"), zcol = "Population")
```



It was not that easy to estimate weighted centroids with R, as it is with GIS software. But there is this new package `centr` that can help us (Zomorrodi 2024).

We need to specify the **group** we want to calculate the **mean centroids**, and the **weight variable** we want to use.

```
# test
library(centr)
Centroid_pop = Census |>
  mean_center(group = "Municipality", weight = "Population")
```

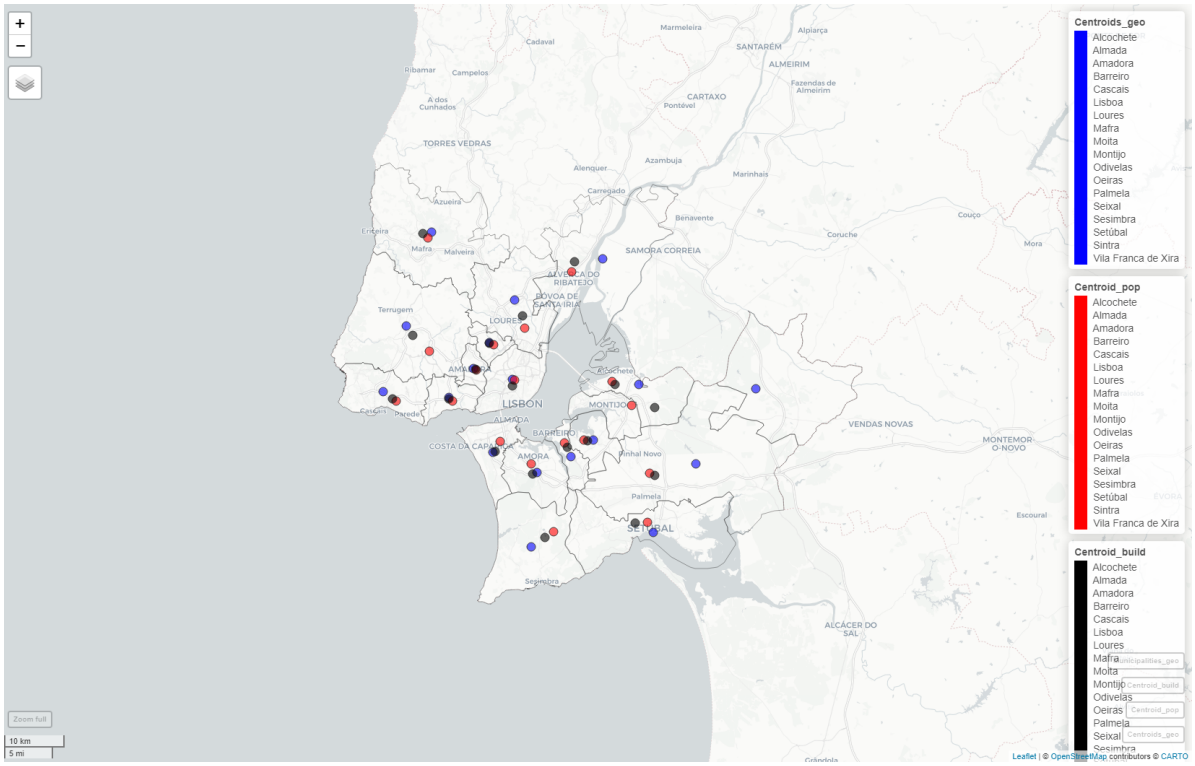
We can do the same for buildings.

```
Centroid_build = Census |>
  mean_center(group = "Municipality", weight = "Buildings")
```

8.3 Compare centroids in a map

8.3.1 Interactive map

```
mapview(Centroids_geo, col.region = "blue") +
  mapview(Centroid_pop, col.region = "red") +
  mapview(Centroid_build, col.region = "black") +
  mapview(Municipalities_geo, alpha.regions = 0) # polygon limits
```



See how the building, population and geometric centroids of Sintra are apart, from closer to Tagus, to the rural area.

8.3.2 Static map

To produce the same map, using only `plot()` and `st_geometry()`, we need to make sure that the geometries have the same crs.

```
st_crs(Centroids_geo) # 4326 WGS84
st_crs(Centroid_pop) # 3763 Portugal TM06
```

So, we need to transform the geometries to the same crs.

```
Centroid_pop = st_transform(Centroid_pop, crs = 4326)
Centroid_build = st_transform(Centroid_build, crs = 4326)
```

Now, to use `plot()` we incrementally add layers to the plot.

```
# Plot the Municipalities_geo polygons first (with no fill)
plot(st_geometry(Municipalities_geo), col = NA, border = "black")

# Add the Centroids_geo points in blue
plot(st_geometry(Centroids_geo), col = "blue", pch = 16, add = TRUE) # add!

# Add the Centroid_pop points in red
plot(st_geometry(Centroid_pop), col = "red", pch = 16, add = TRUE)

# Add the Centroid_build points in black
plot(st_geometry(Centroid_build), col = "black", pch = 16, add = TRUE)
```

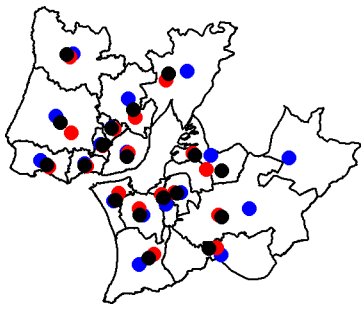


Figure 8.1: Static map of different centroids of Municipalities

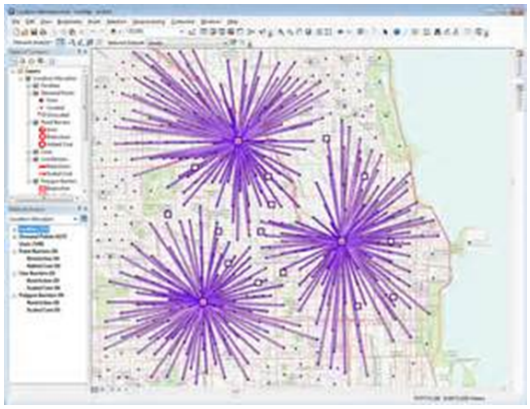
In the next section we will use these centroids to calculate the desire lines between them.

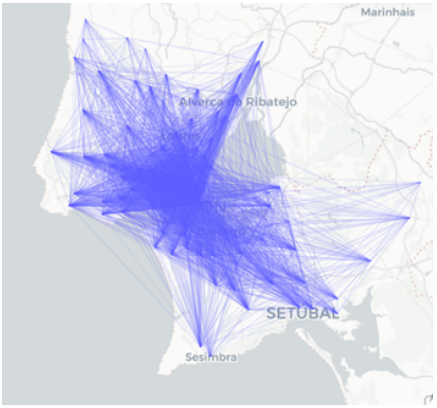
9 OD pairs and desire lines

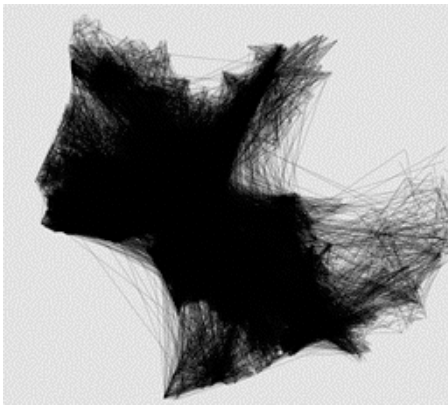
Desire lines are a way to represent the flow of people or goods between two points. They are often used in transport planning to represent the flow of trips between zones.

There are many ways to create desire lines, connecting origins and destinations. For instance:

- 1 origin to 1 destination
  - home work place
- multiple origins to a single destinations
  - students' homes 1 school
- origin zone to destination zone
  - for aggregated trips (macro representation)
- OD jittering[1]







The `stplanr` package is a collection of functions for sustainable transport planning with R, and it is built on top of the `sf` package (Lovelace and Ellison 2018).

In this workshop, we will use the `od` package, a lightweight package with a few functions from `stplanr`, namely the ones to create desire lines from origin-destination (OD) pairs (Lovelace and Morgan 2024).

[1] See (Lovelace, Félix, and Carlino 2022).

9.1 Desire lines with od_2_sf

To create desire lines, we need a dataset with OD pairs **and** other dataset with the corresponding transport zones (spatial data).

The `TRIPSmode.Rds` dataset includes origins, destinations and number of trips between municipalities.

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

The `Municipalities_geo.gpkg` dataset includes the geometry of the transport zones.

```
library(sf)
Municipalities_geo = st_read("data/Municipalities_geo.gpkg", quiet = TRUE) # supress mesage
```

Then, we need to load the `od` package. We will use the `od_to_sf()` function to create desire lines from OD pairs.

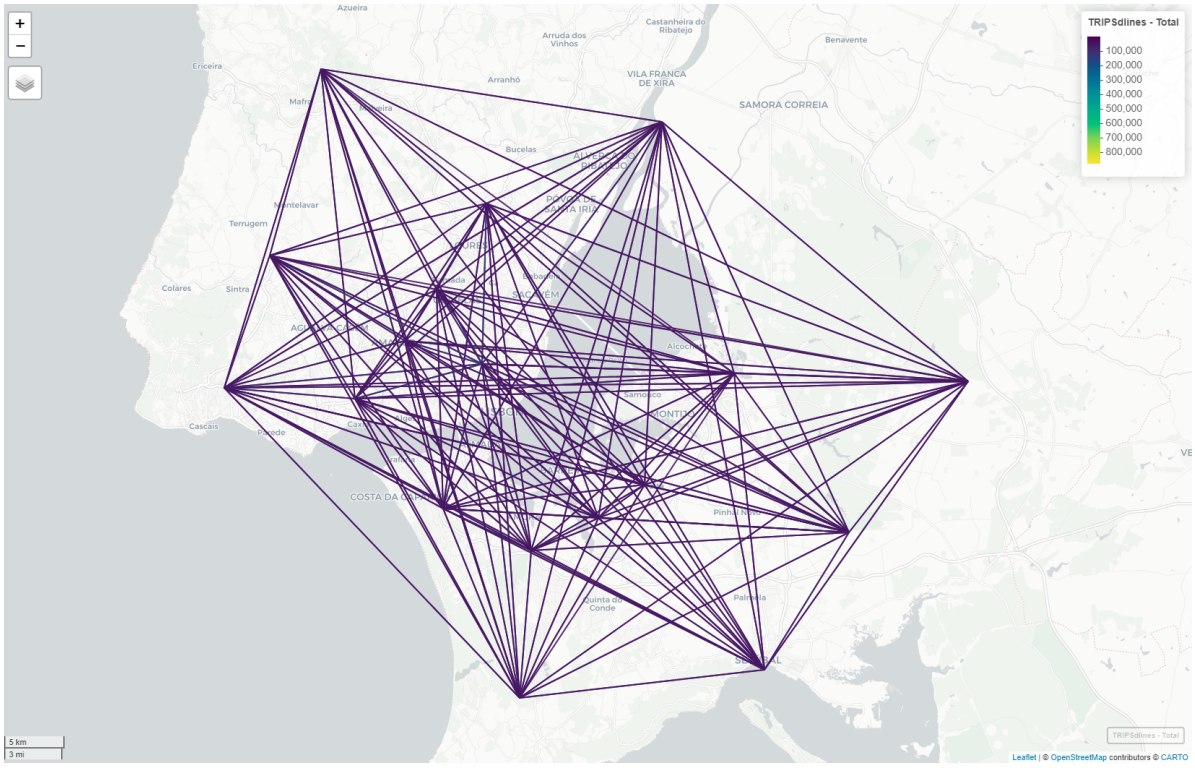
```
# install.packages("od")
library(od)

TRIPSdlines = od_to_sf(TRIPSmode, z = Municipalities_geo) # z for zones
```

For this magic to work smoothly, the first two columns of the `TRIPSmode` dataset must be the **origin** and **destination** zones, and these zones need to correspond to the first column of the `Municipalities_geo` dataset (with an associated geometry).

> See more options with the `?stplanr::od2line` function.

Now we can visualize the desire lines using the `mapview` function.

```
library(mapview)
mapview(TRIPSdlines, zcol = "Total")
```

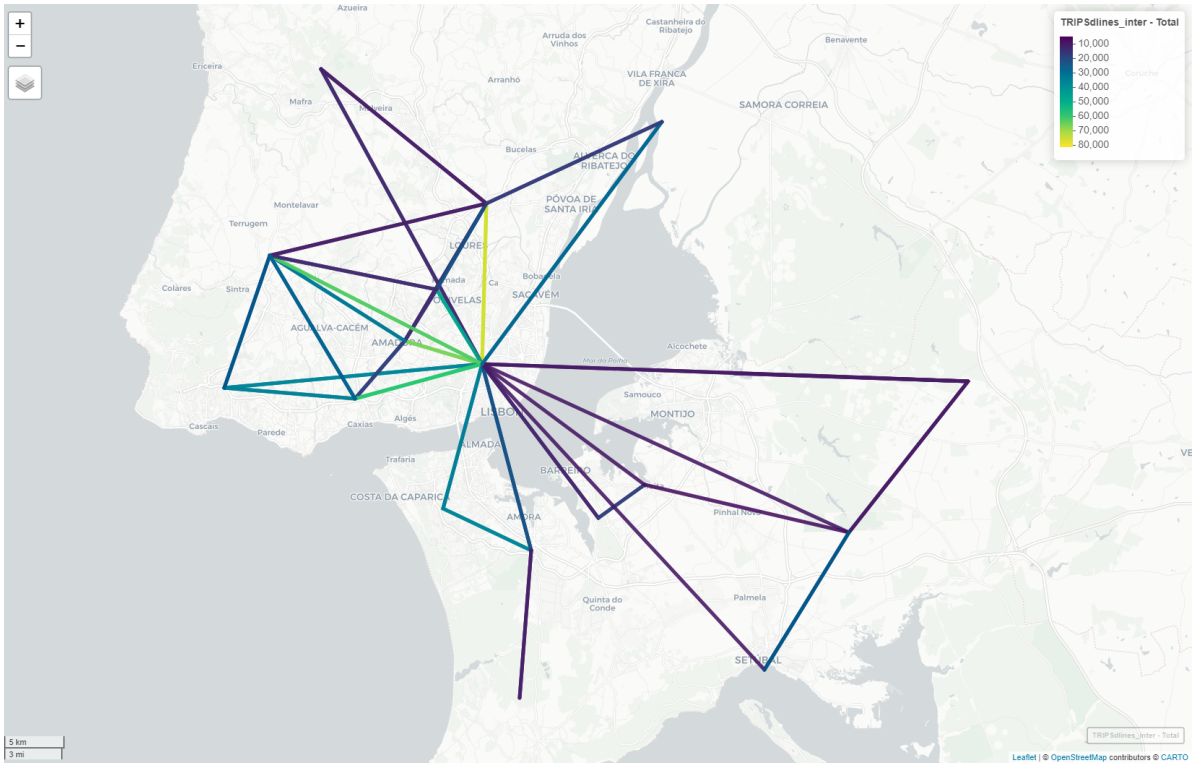


As you can see, this is too much information to be able to understand the flows.

9.1.1 Filtering desire lines

Filter **intrazonal** trips.

```
library(dplyr)

TRIPSdlines_inter = TRIPSdlines |>
  filter(Origin != Destination) |> # remove intrazonal trips
  filter(Total > 5000) # remove noise

mapview(TRIPSdlines_inter, zcol = "Total", lwd = 5)
```



Filter trips with origin or destination **not in** Lisbon.

```
TRIPSdl_noLX = TRIPSdlines_inter |>
  filter(Origin != "Lisboa", Destination != "Lisboa")

mapview(TRIPSdl_noLX, zcol = "Total", lwd = 8) # larger line width
```

Try to replace the `Total` with other variables, such as `Car`, `PTransit`, and see the differences.

9.2 Oneway desire lines

Note that the `od_to_sf()` function creates bidirectional desire lines. This can be not the ideal for visualization / representation purposes, as you will have 2 lines overlaping.

The function `od_oneway()` aggregates oneway lines to produce bidirectional flows.

By default, it returns the sum of each numeric column for each bidirectional origin-destination pair.

```
nrow(TRIPSdlines)
```

```
[1] 315
```

```
TRIPSdlines_oneway = od_oneway(TRIPSdlines)
nrow(TRIPSdlines_oneway)
```

```
[1] 168
```

Note that for the last municipalities you have less combinations now. Nevertheless, all the possible combinations are represented.

```
head(TRIPSdlines_oneway[,c(1,2)]) # just the first 2 columns
```

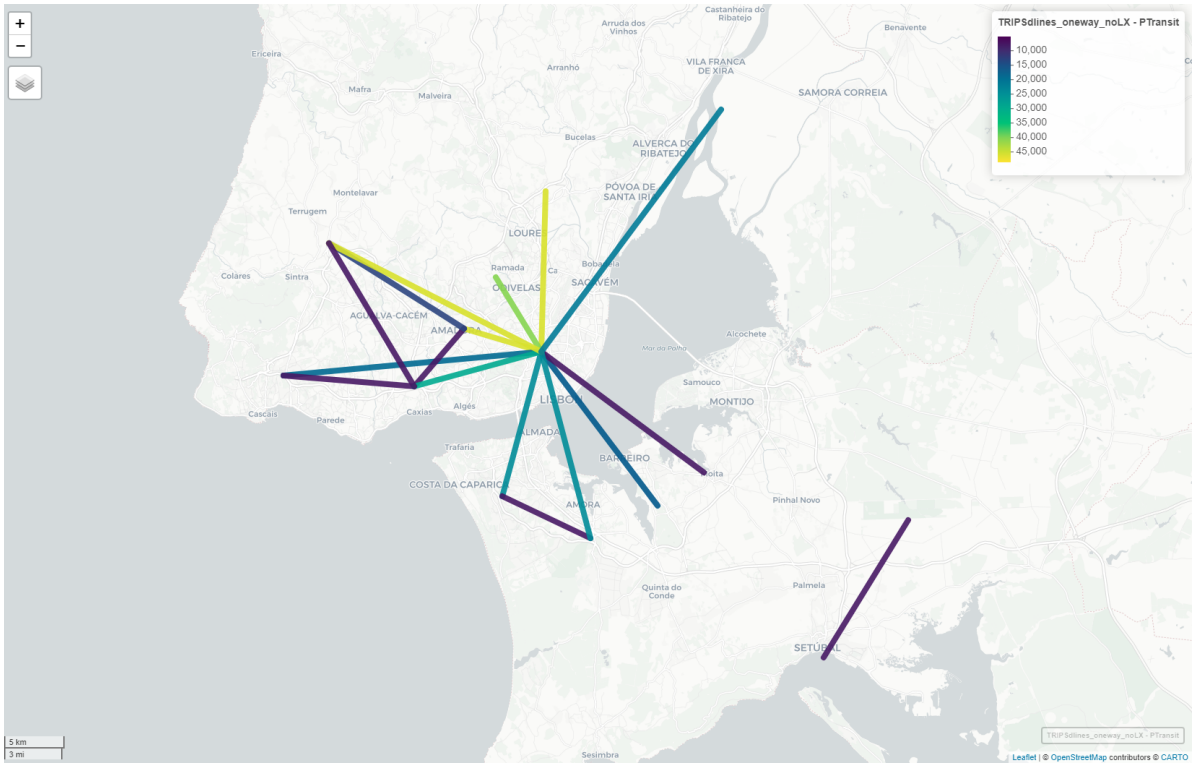
```
Simple feature collection with 6 features and 2 fields
Attribute-geometry relationships: identity (2) (with 3 geometries empty)
Geometry type: LINESTRING
Dimension:     XY
Bounding box:  xmin: -9.229502 ymin: 38.63562 xmax: -8.91588 ymax: 38.75981
Geodetic CRS:  WGS 84
          o         d                       geometry
1 Alcochete Alcochete               LINESTRING EMPTY
2 Alcochete    Almada LINESTRING (-8.91588 38.735...
3    Almada    Almada               LINESTRING EMPTY
4 Alcochete   Amadora LINESTRING (-8.91588 38.735...
5    Almada   Amadora LINESTRING (-9.193069 38.63...
6   Amadora   Amadora               LINESTRING EMPTY
```

```
tail(TRIPSdlines_oneway[,c(1,2)])
```

```
Simple feature collection with 6 features and 2 fields
Attribute-geometry relationships: identity (2) (with 1 geometry empty)
Geometry type: LINESTRING
Dimension:     XY
Bounding box:  xmin: -9.357651 ymin: 38.4949 xmax: -8.806644 ymax: 38.92211
Geodetic CRS:  WGS 84
                     o                   d                       geometry
163            Palmela Vila Franca de Xira LINESTRING (-8.806644 38.61...
164              Seixal Vila Franca de Xira LINESTRING (-9.108801 38.60...
165           Sesimbra Vila Franca de Xira LINESTRING (-9.120124 38.49...
166            Setúbal Vila Franca de Xira LINESTRING (-8.887481 38.51...
167             Sintra Vila Franca de Xira LINESTRING (-9.357651 38.82...
168 Vila Franca de Xira Vila Franca de Xira               LINESTRING EMPTY
```

Example of visualization with Public Transit trips in both ways.

```
TRIPSdlines_oneway_noLX = TRIPSdlines_oneway |>
  filter(o != d) |> # remove intrazonal trips
  filter(PTransit > 5000) # reduce noise

mapview(TRIPSdlines_oneway_noLX, zcol = "PTransit", lwd = 8)
```



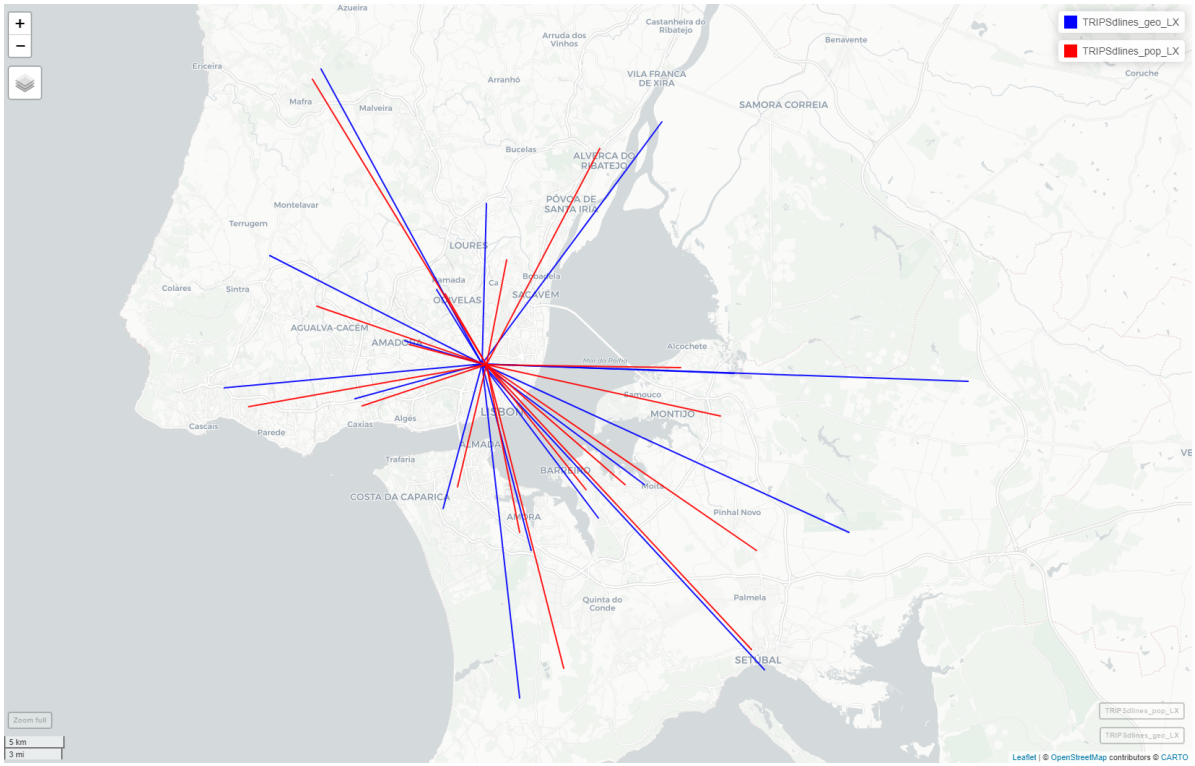
9.3 Using population centroids

The `od_to_sf()` function uses the geometric center of the zones to create the desire lines. But if we replace those zones by the weighted centroids, we can have a more realistic representation of the flows.

```
# Centroid_pop = st_read("data/Centroid_pop.gpkg")

TRIPSdlines_pop = od_to_sf(TRIPSmode, z = Centroid_pop) |>  # works the same way
  od_oneway() # oneway
```

Check differences of lines with trips from/to Lisbon:

```
TRIPSdlines_geo_LX = TRIPSdlines_oneway |>
  filter(o == "Lisboa" | d == "Lisboa") # or condition
TRIPSdlines_pop_LX = TRIPSdlines_pop |>
  filter(o == "Lisboa" | d == "Lisboa")

mapview(TRIPSdlines_geo_LX, color = "blue") + mapview(TRIPSdlines_pop_LX, color = "red")
```



The next step will be estimating the **euclidean distances** between these centroids, and compare them with the **routing distances**.

10 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).

10.1 Euclidean distances

Taking the survey respondents' location, we will estimate the distance to the university (IST) using the `sf` package.

10.1.1 Import survey data frame convert to sf

We will use a survey dataset with 200 observations, with the following variables: ID, Affiliation, Age, Sex, Transport Mode to IST, and latitude and longitude coordinates.

```
library(dplyr)

SURVEY = read.csv("data/SURVEY.txt", sep = "\t") # tab delimiter
names(SURVEY)
```

```
[1] "ID"   "AFF"  "AGE"  "SEX"  "MODE" "lat"  "lon"
```

As we have the coordinates, we can convert this data frame to a spatial feature, as explained in the Introduction to spatial data section.

```
library(sf)

SURVEYgeo = st_as_sf(SURVEY, coords = c("lon", "lat"), crs = 4326) # convert to as sf data
```

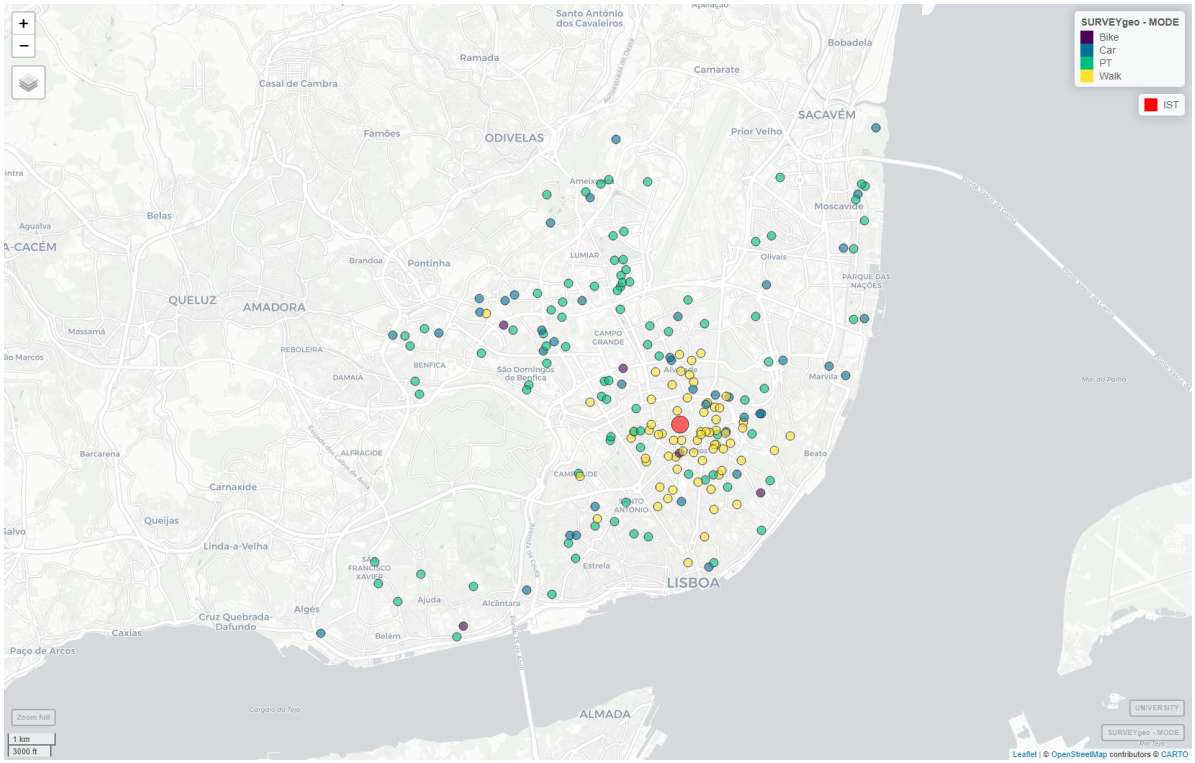
10.1.2 Create new point at the university

Using coordinates from Instituto Superior Técnico, we can directly create a simple feature and assign its crs.

```
UNIVERSITY = data.frame(place = "IST",
                        lon = -9.1397404,
                        lat = 38.7370168) |>  # first a dataframe
  st_as_sf(coords = c("lon", "lat"), # then a spacial feature
           crs = 4326)
```

Visualize in a map:

```
library(mapview)
mapview(SURVEYgeo, zcol = "MODE") + mapview(UNIVERSITY, col.region = "red", cex = 12)
```



10.1.3 Straight lines

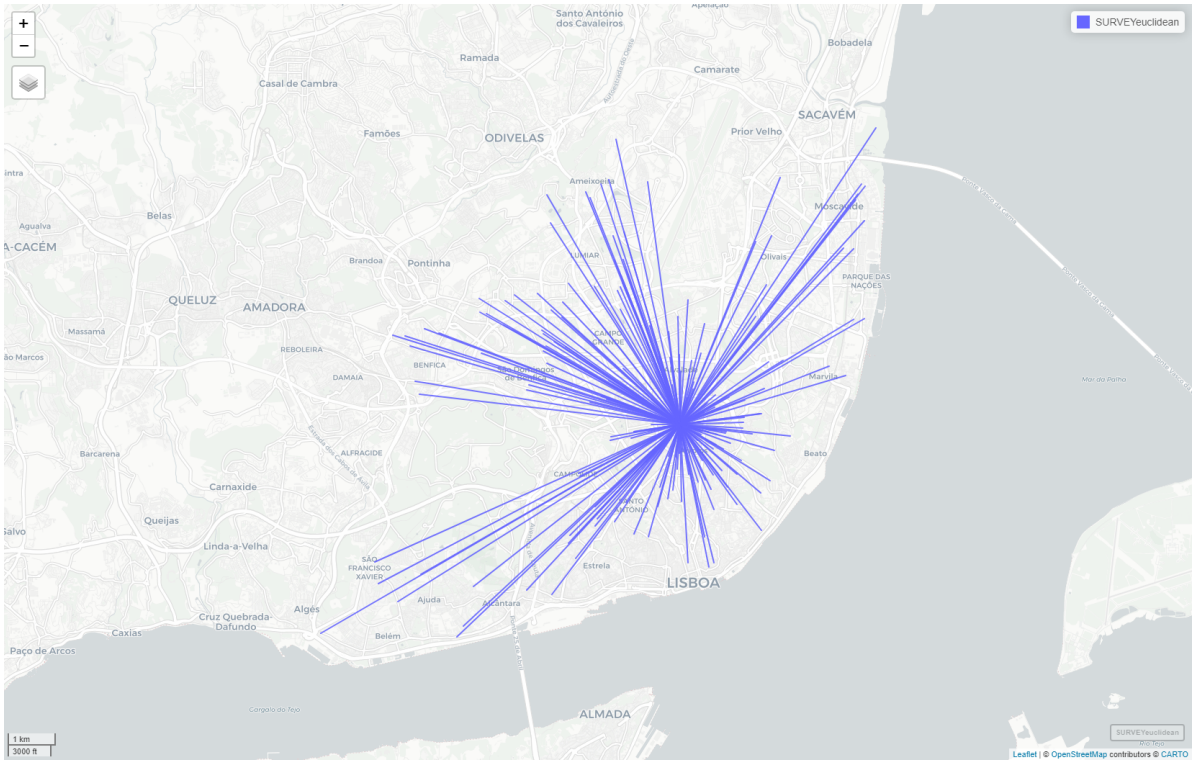
First we will create lines connecting the survey locations to the university, using the `st_nearest_points()` function.

This function finds returns the nearest points between two geometries, and creates a line between them. This can be useful to find the nearest train station to each point, for instance.

As we only have 1 point at UNIVERSITY layer, we will have the same number of lines as number of surveys = 200.

```
SURVEYeuclidean = st_nearest_points(SURVEYgeo, UNIVERSITY, pairwise = TRUE) |>
  st_as_sf() # this creates lines

mapview(SURVEYeuclidean)
```

```
Warning in cbind(`Feature ID` = fid, mat): number of rows of result is not a
multiple of vector length (arg 1)
```



Note that if we have more than one point in the second layer, the `pairwise = TRUE` will create a line for each combination of points. Set to `FALSE` if, for instance, you have the same number of points in both layers and want to create a line between the corresponding points.

10.1.4 Distance

Now we can estimate the distance using the `st_length()` function.

```
# compute the line length and add directly in the first survey layer
SURVEYgeo = SURVEYgeo |>
  mutate(distance = st_length(SURVEYeuclidean))

# remove the units - can be useful
SURVEYgeo$distance = units::drop_units(SURVEYgeo$distance)
```

We could also estimate the distance using the `st_distance()` function **directly**, although we would not get and sf with lines.

```
SURVEYgeo = SURVEYgeo |>
  mutate(distance = st_distance(SURVEYgeo, UNIVERSITY)[,1] |>  # in meters
           units::drop_units()) |>  # remove units
  mutate(distance = round(distance)) # round to integer

summary(SURVEYgeo$distance)
```

```
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
    298    1106    2186    2658    3683    8600
```

`SURVEYgeo` is still a points' sf.

10.2 Routing Engines

There are different types of routing engines, regarding the type of network they use, the type of transportation they consider, and the type of data they need. We can have:

- Uni-modal vs. Multi-modal
  - One mode per trip vs. One trip with multiple legs that can be made with different modes
  - Multi-modal routing may require GTFS data (realistic Public Transit)
- Output level of the results
  - Routes (1 journey = 1 route)
  - Legs

  - Segments

- Routing profiles
  - Type of user
  - fastest / shortest path
  - avoid barriers / tolls, etc

Figure 10.1: Routing options in OpenRouteService

- Local vs. Remote (service request - usually web API)
  - Speed vs. Quota limits / price
  - Hard vs. Easy set up
  - Hardware limitations in local routing
  - Global coverage in remote routing, with frequent updates

Examples: OSRM, Dodgr, r5r, Googleway, CycleStreets, HERE.

10.2.1 Routing distances with r5r

We use the `r5r` package to estimate the distance using a road network (R. H. M. Pereira et al. 2021).

> **i** To properly the setup r5r model for the area you are working on, you need to download the **road network** data from OpenStreetMap and, if needed, add a **GTFS** and **DEM** file, as it will be explained in the next section.

We will use only respondents with a distance to the university less than 2 km.

```
SURVEYsample = SURVEYgeo |> filter(distance <= 2000)
nrow(SURVEYsample)
```

```
[1] 95
```

We need an id (unique identifier) for each survey location, to be used in the routing functions of `r5r`.

```r
# create id columns for both datasets
SURVEYsample = SURVEYsample |>
  mutate(id = c(1:nrow(SURVEYsample))) # from 1 to the number of rows

UNIVERSITY = UNIVERSITY |>
  mutate(id = 1) # only one row
```

10.2.1.1 Distances by car

Estimate the routes with time and distance by car, from survey locations to University.

```r
SURVEYcar = detailed_itineraries(
  r5r_core = r5r_network,
  origins = SURVEYsample,
  destinations = UNIVERSITY,
  mode = "CAR",
  all_to_all = TRUE # if false, only 1-1 would be calculated
)

names(SURVEYcar)
```

```
 [1] "from_id"          "from_lat"         "from_lon"         "to_id"
 [5] "to_lat"           "to_lon"           "option"           "departure_time"
 [9] "total_duration"   "total_distance"   "segment"          "mode"
[13] "segment_duration" "wait"             "distance"         "route"
[17] "geometry"
```

The `detailed_itineraries()` function is super detailed!

> **i** If we want to know only time and distance, and **not the route** itself, we can use the `travel_time_matrix()`.

10.2.1.2 Distances by foot

Repeat the same for `WALK`[1].

[1] For bike you would use `BICYCLE`.

```
SURVEYwalk = detailed_itineraries(
  r5r_core = r5r_network,
  origins = SURVEYsample,
  destinations = UNIVERSITY,
  mode = "WALK",
  all_to_all = TRUE # if false, only 1-1 would be calculated
)
```

10.2.1.3 Distances by PT

For Public Transit (`TRANSIT`) you may specify the egress mode, the departure time, and the maximum number of transfers.

```
SURVEYtransit = detailed_itineraries(
  r5r_core = r5r_network,
  origins = SURVEYsample,
  destinations = UNIVERSITY,
  mode = "TRANSIT",
  mode_egress = "WALK",
  max_rides = 1, # The maximum PT rides allowed in the same trip
  departure_datetime =  as.POSIXct("20-09-2023 08:00:00",
                                   format = "%d-%m-%Y %H:%M:%S"),
  all_to_all = TRUE # if false, only 1-1 would be calculated
)
```

10.3 Compare distances

We can now compare the euclidean and routing distances that we estimated for the survey locations under 2 km.

```
summary(SURVEYsample$distance) # Euclidean
```

```
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
    298     790    1046    1112    1470    1963
```

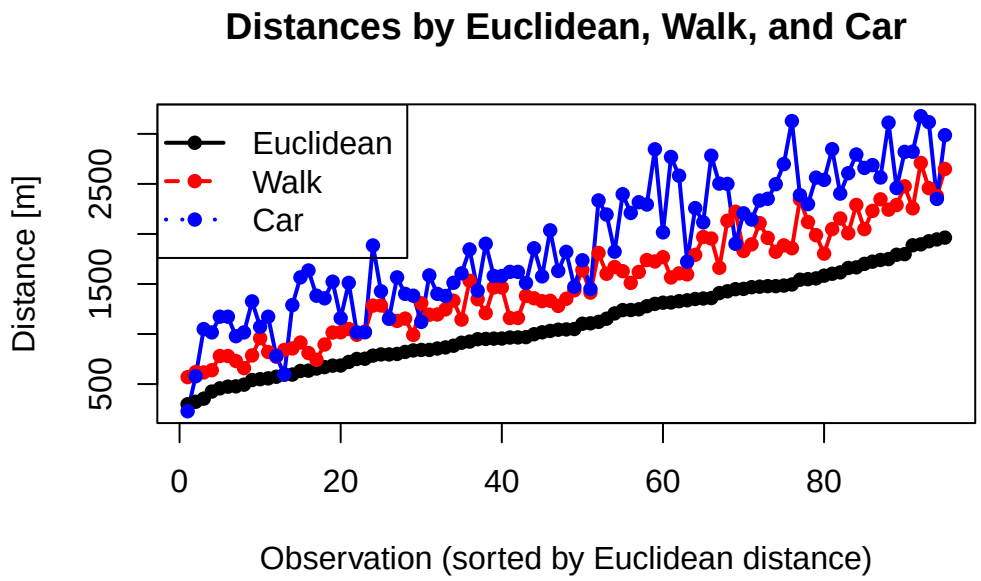
```
summary(SURVEYwalk$distance) # Walk
```

```
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
    569    1090    1465    1505    1925    2710
```

```
summary(SURVEYcar$distance) # Car
```

```
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
    228    1401    1823    1893    2431    3177
```

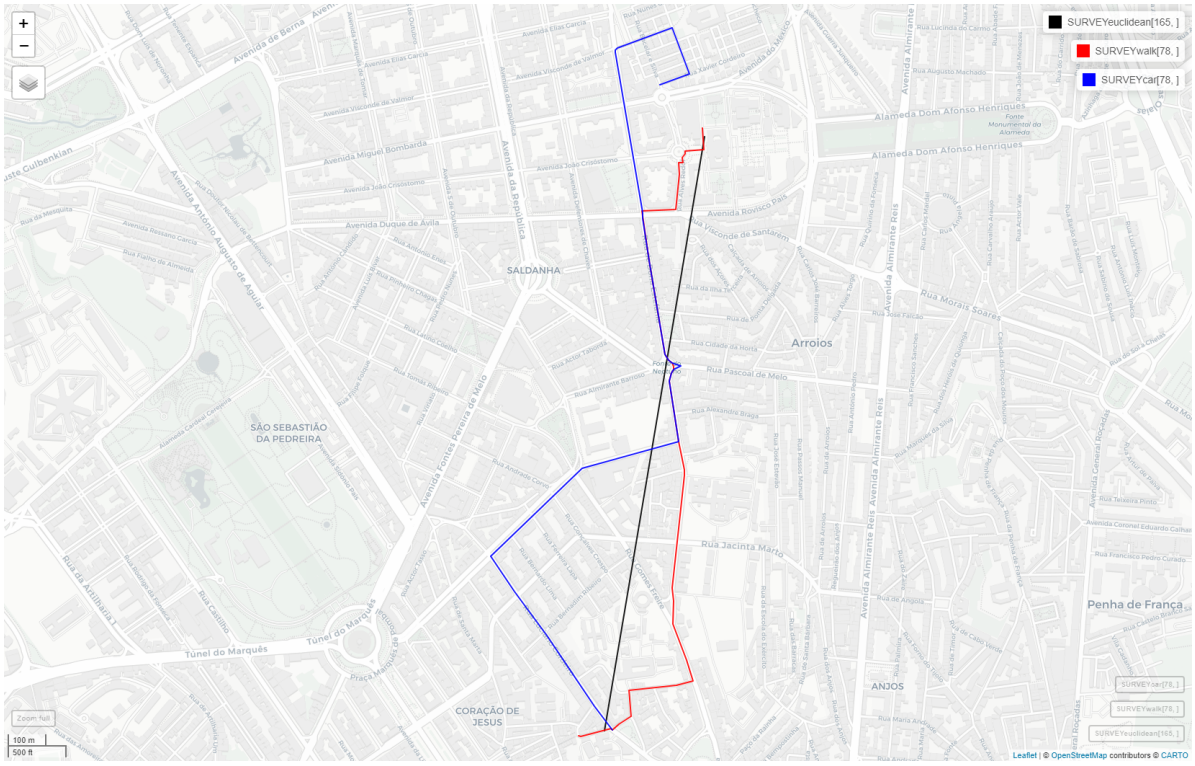
What can you understand from this results?



10.3.1 Circuity

Compare 1 single route.

```
mapview(SURVEYeuclidean[165,], color = "black") + # 1556 meters
  mapview(SURVEYwalk[78,], color = "red") + # 1989 meters
  mapview(SURVEYcar[78,], color = "blue") # 2565 meters
```



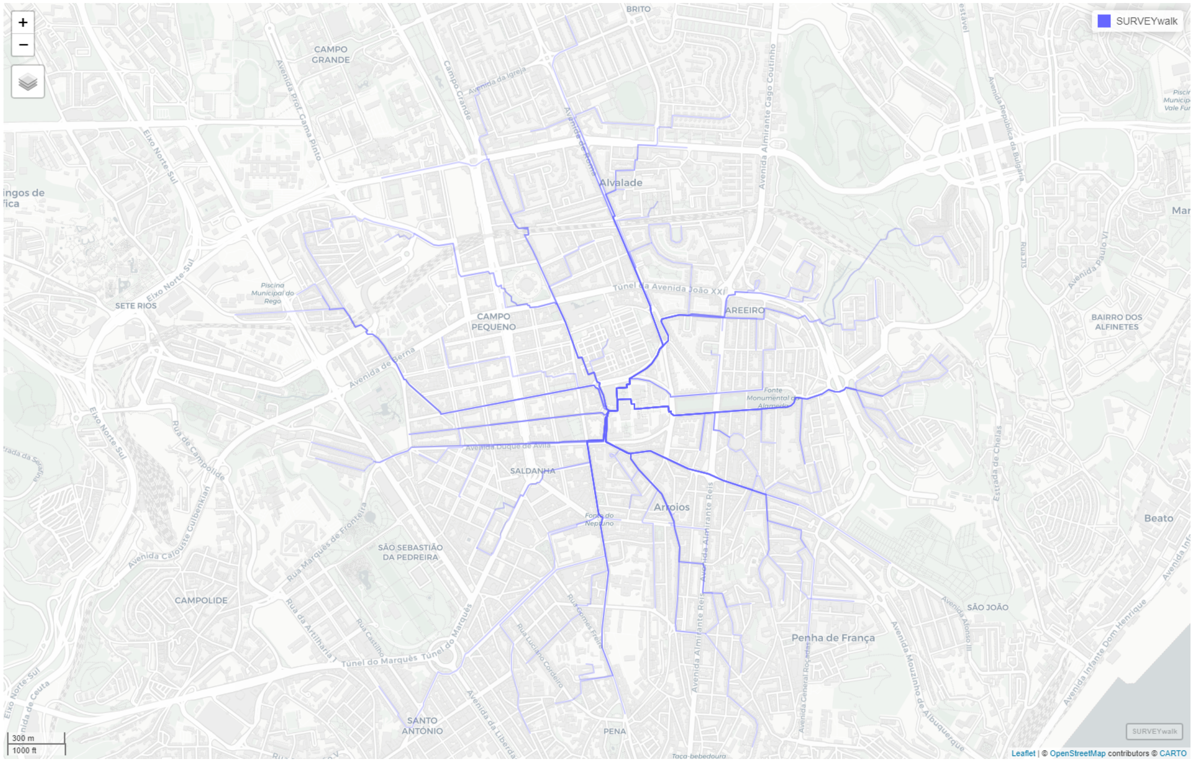
With this we can see the **circuity** of the routes, a measure of route / transportation efficiency, which is the ratio between the routing distance and the euclidean distance.

The cicuity for car (1.65) is usually higher than for walking (1.28) or biking, for shorter distances.

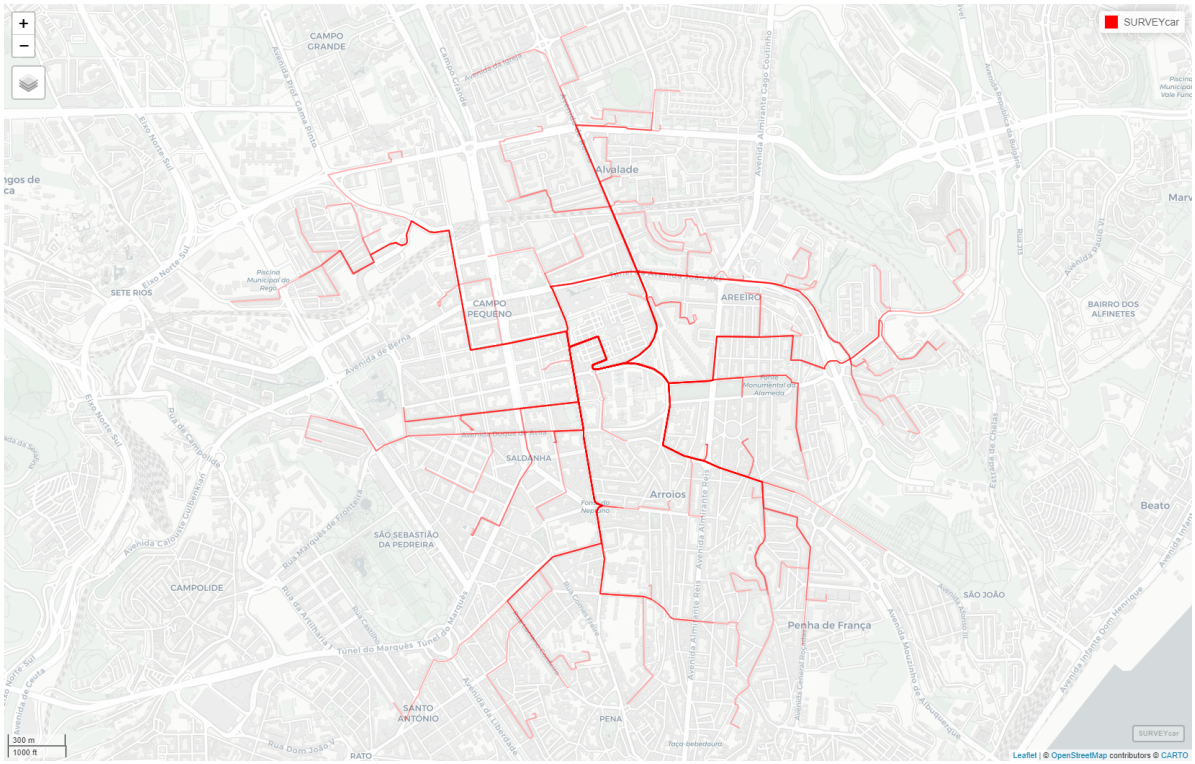
10.4 Visualize routes

Visualize with transparency of 30%, to get a clue when they overlay.

```
mapview(SURVEYwalk, alpha = 0.3)
```

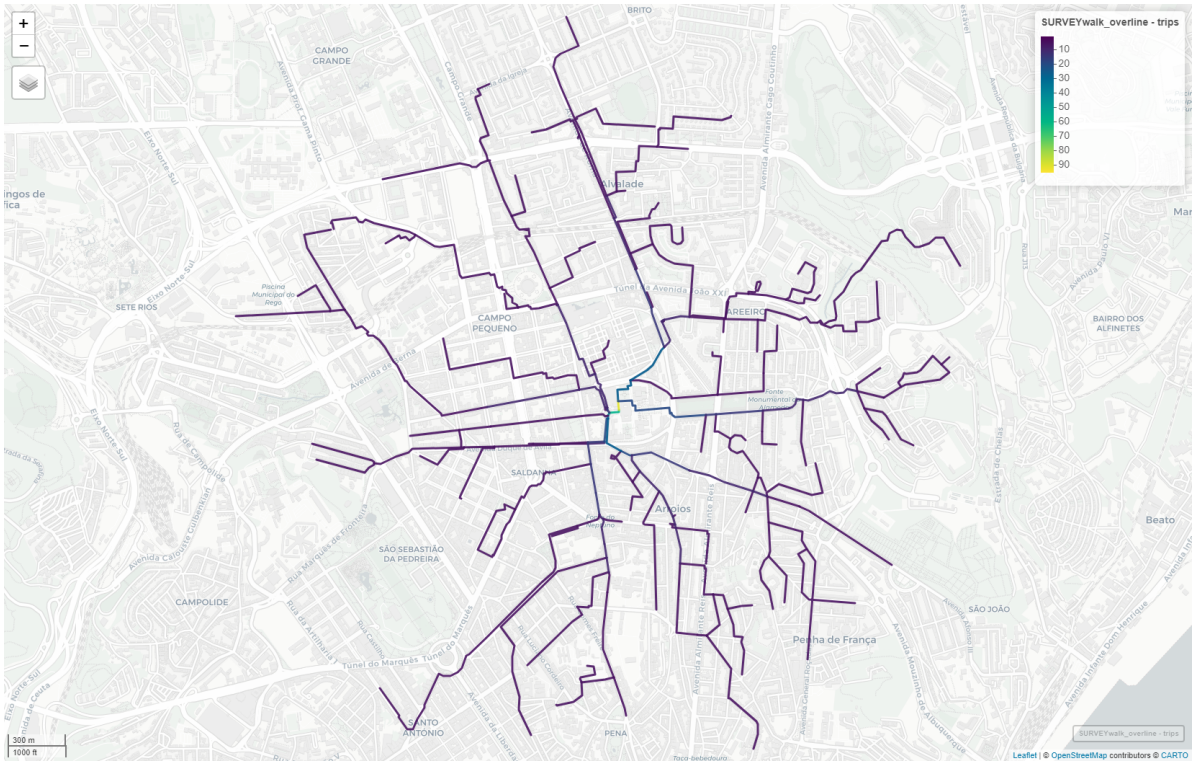



```r
mapview(SURVEYcar, alpha = 0.3, color = "red")
```



We can also use the `overline()` function from `stplanr` package to break up the routes when they *overline*, and add them up.

```
# we create a value that we can later sum
# it can be the number of trips represented by this route
SURVEYwalk$trips = 1 # in this case is only one respondent per route

SURVEYwalk_overline = stplanr::overline(
  SURVEYwalk,
  attrib = "trips",
  fun = sum
)

mapview(SURVEYwalk_overline, zcol = "trips", lwd = 3)
```



With this we can visually inform on how many people travel along a route, from the survey dataset[2].

[2]Assuming all travel by the shortest path.

11 Open transportation data

In this chapter we will guide you through sources of open data for transportation analysis: road networks and public transportation information.

11.1 Road Networks

11.1.1 OpenStreetMap

The OpenStreetMap is a collaborative online mapping project that creates a free editable map of the world.

This is the most used source of road network data for transportation analysis in academia, since it is available almost **everywhere in the world**, is open and free to use.

> Although it can be not 100% accurate, OSM is a good source of data for most of the cases.

You can access it's visualization tool at www.openstreetmap.org. To edit the map, you can use the Editor, once you register.

If you want to **download** the data, you can use the following tools.

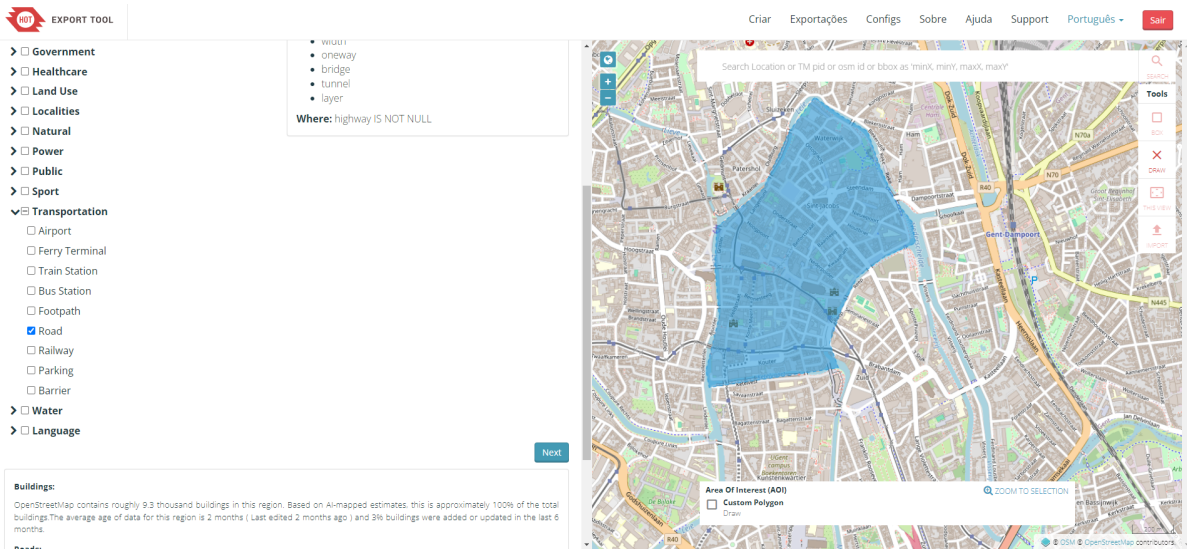
- Overpass API
- Geofabrik

These websites include all the OSM data, with **much more information than you need**.

11.1.2 HOT Export Tool

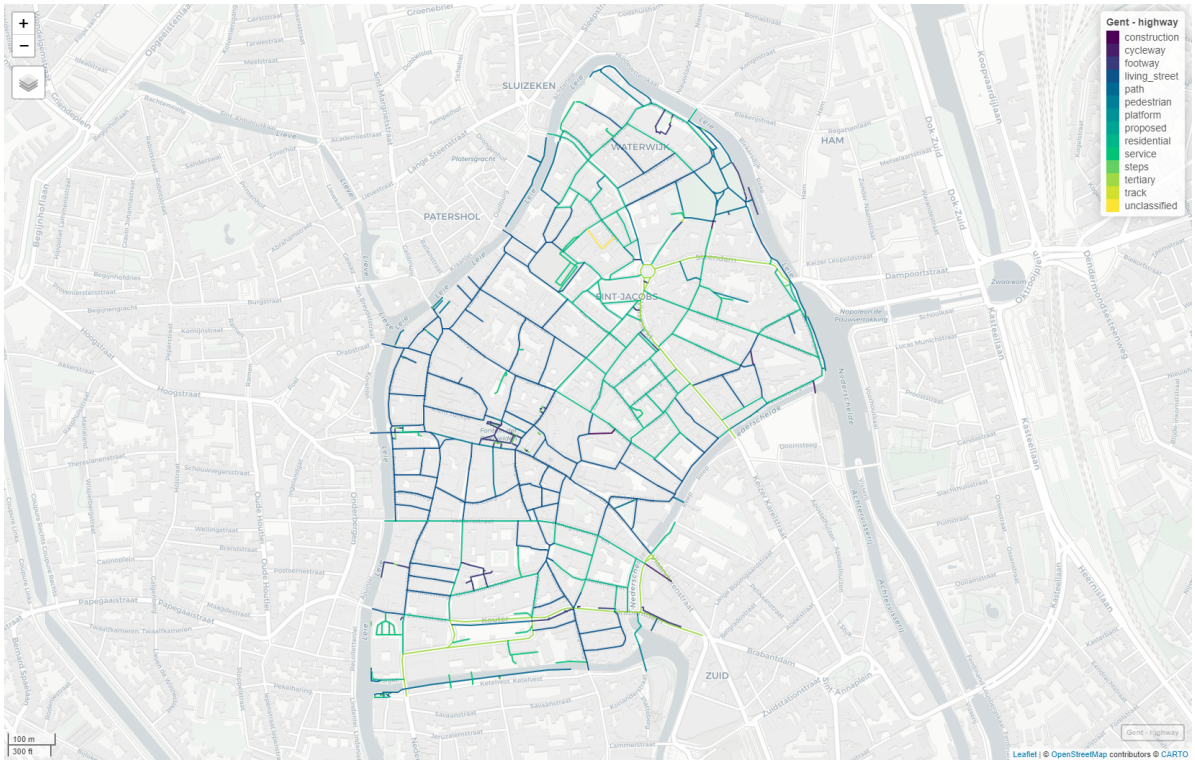
This interactive tool helps you to select the **region** you want to extract, the type of **information** to include, and the output data **format**.

Access via export.hotosm.org. You need an OSM account to use it.



After the export, you can read in R using the **sf** package:

```
Gent = sf::st_read("data/Gent_center.gpkg", quiet = TRUE)

mapview::mapview(Gent, zcol = "highway")
```



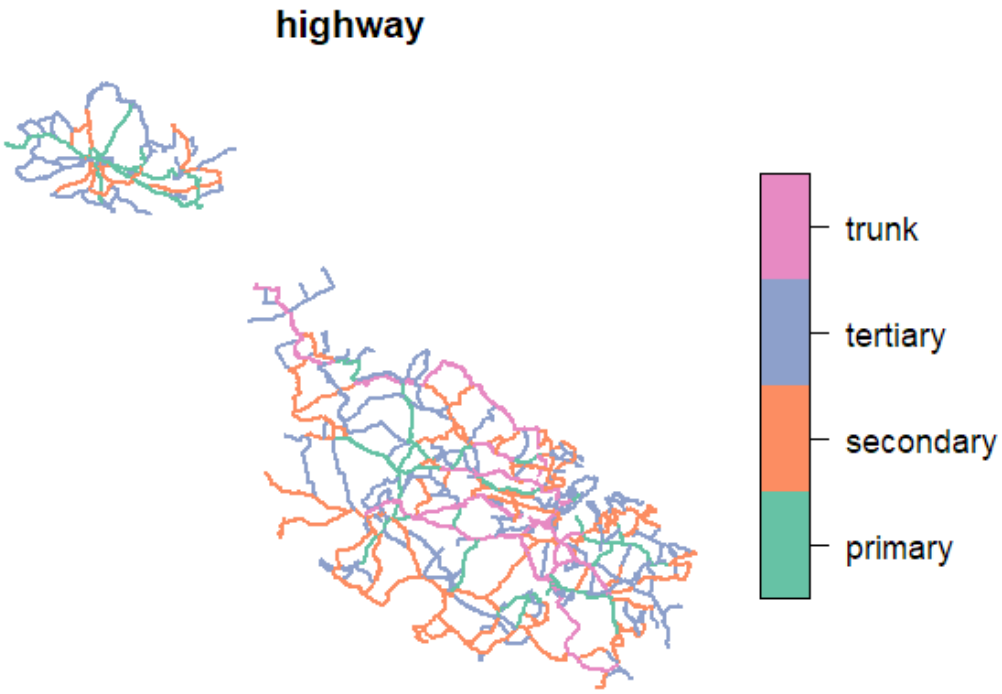
11.1.3 OSM in R

There are also some R packages that can help you to download and work with OpenStreetMap data, such as:

- osmdata
- osmextract

This is an example of how to download OpenStreetMap road network data using the `osmextract` package:

```
library(osmextract)
OSM_Malta = oe_get_network(place = "Malta") # it will geocode the place

Malta_main_roads = OSM_Malta |>
  filter(highway %in% c("primary", "secondary", "tertiary", "trunk"))

plot(Malta_main_roads["highway"])
```



11.2 Transportation Services' Data

11.2.1 GTFS

General Transit Feed Specification (GTFS) is standard format for documenting public transportation information, including: routes, schedules, stop locations, calendar patterns, trips, and possible transfers. Transit agencies are responsible for maintaining the data up-to-date.

This information is used in several applications, such as Google Maps, to provide public transportation directions. It can be offered for a city, a region, or even a whole country, depending on the PT agency.

The recent version 2 of the GTFS standard includes more information, such as **real-time data**.

The data is usually in a `.zip` file that includes several `.txt` files (one for each type of information) with tabular relations.

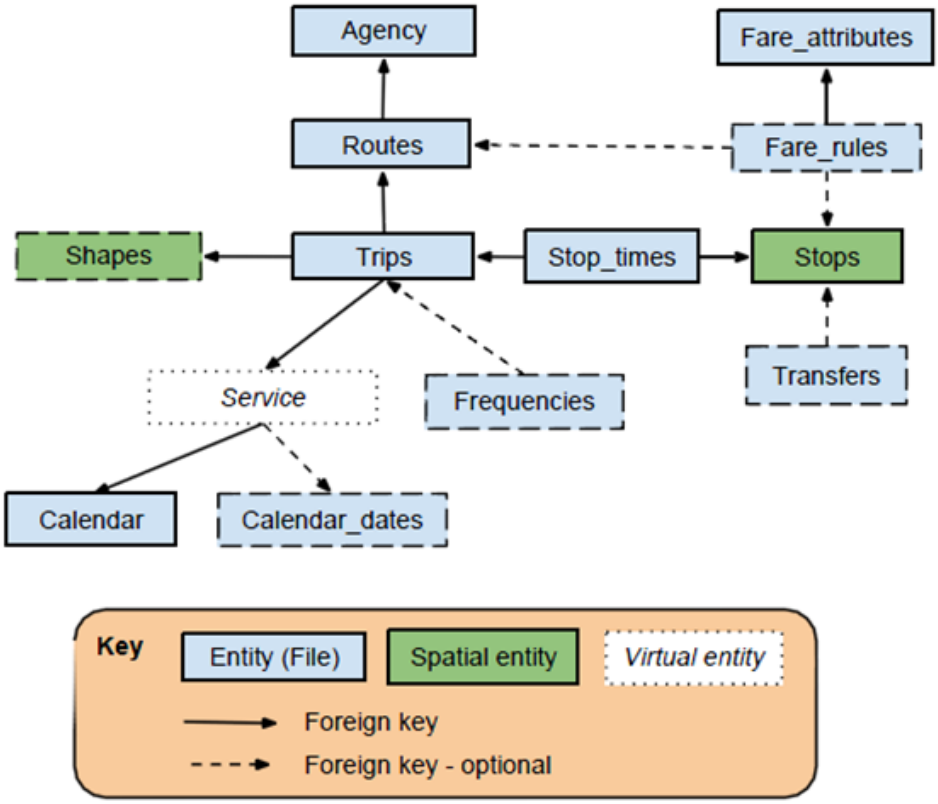


Figure 11.1: Source: trilliumtransit.com

You can find most GTFS data in the following websites:

- TransitLand
- TransitFeeds

Some PT agencies also provide their open-data in their websites.

11.2.2 National Access Points

The European Union has a directive that requires the member states to provide access to transportation data. Data includes not only **Public Transportation** data, but also **road network**s, car **parking**, and other transportation-related information.

List of the European Union members states with National Access Points for Transportation data

Example of Bus services data in Belgium:

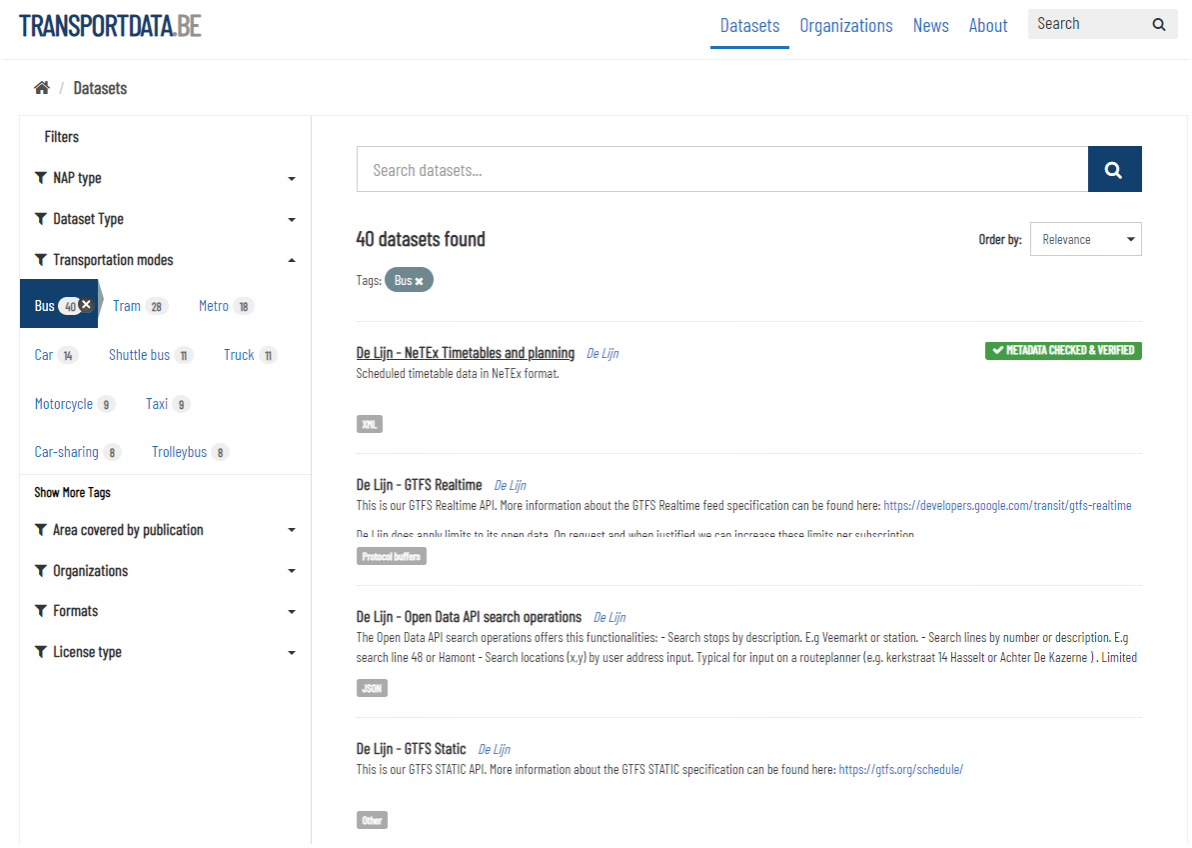


Figure 11.2: Source: Transport Data Belgium

12 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/

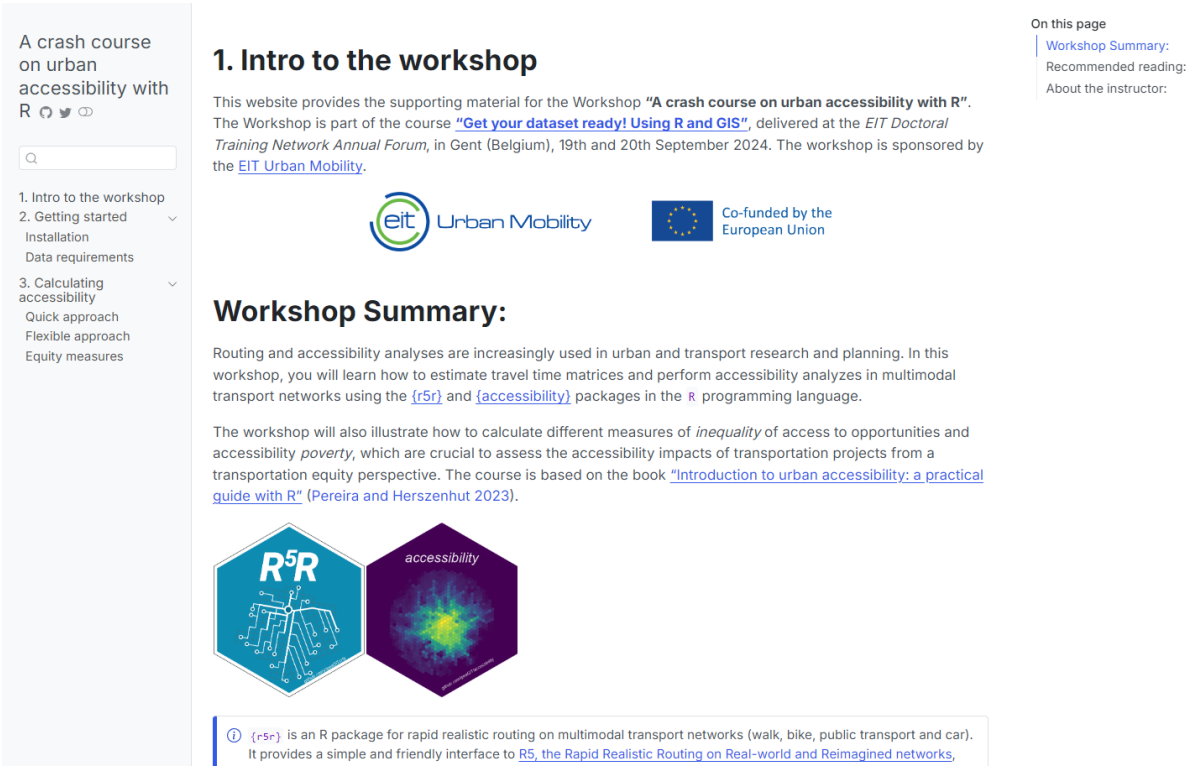


Figure 12.1: Screenshot of the website for this learning module.

About the instructors

Rosa Félix

Senior Posdoctoral Researcher
CERIS, Instituto Superior Técnico - University of Lisbon
Website | Google Scholar | Twitter | GitHub | Linkedin

Short bio

Rosa Félix is a senior post-doctoral researcher at the Instituto Superior Técnico – University of Lisbon and member of the U-Shift lab, in the Transportation Research Group of CERIS. Having a background of Urban Planning Engineering, she completed her Ph.D. in Transport Systems in 2019 at Instituto Superior Técnico (MIT Portugal program), and was a Visiting Scholar at Portland State University in 2017/18.

Rosa is an active mobility researcher, and excels in R and GIS. She is an open source and reproducible research enthusiast. Her publications include articles on cycling and behavior change, and open source code solutions to specific GIS and mobility problems. Every year, Rosa lectures a course for cycling infrastructure planning and design for practitioners, and also teaches GIS for transportation and introduction to programming for MSc course of Transportation Systems.

Rosa has worked in multiple R&D and consultancy projects with both municipalities and industry, such as the Municipality of Lisbon (2019-2022) and the Department for Transportation of Lisbon Metro (2023), in which she developed a digital tool[1] to support the planning of the metropolitan cycling network, in collaboration with Institute for Transport Studies of the University of Leeds.

[1]Biclar. Available at: https://biclar.tmlmobilidade.pt

Gabriel Valença

Posdoctoral Researcher
CERIS, Instituto Superior Técnico - University of Lisbon
Google Scholar | Linkedin

Short bio

Gabriel Valença is a post-doctoral researcher at the Instituto Superior Técnico - University of Lisbon and member of the U-Shift lab in Transportation Research Group of CERIS, where he recently completed his Ph.D. in Transportation Systems.

He has experience in R programming with applications related to transport demand modelling, data science, GIS and machine learning. He is a member of the European Doctoral Training

Network from the 1st intake. He did his international placement at the Technical University of Denmark for 6 months.

His background is in Civil Engineering, where he graduated at the Federal University of Rio Grande do Norte (UFRN) in Brazil, while studying part of his degree at the University of Toronto, in Canada. His main research areas are in integrating concepts related to street design, smart cities, urban mobility and intelligent transportation systems focusing on traveler behavior, transport demand modelling and artificial intelligence.

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.

References

Engel, Claudia A. 2023. *Introduction to r*. cengel.github.io/R-intro/.

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

———. 2022. "Censos 2021- XVI Recenseamento Geral da População. VI Recenseamento Geral da Habitação." Lisboa: Instituto National de Estatística. https://censos.ine.pt/xurl/pub/65586079.

Lovelace, Robin, and Richard Ellison. 2018. "Stplanr: A Package for Transport Planning." *The R Journal* 10 (2): 10. https://doi.org/10.32614/RJ-2018-053.

Lovelace, Robin, Rosa Félix, and Dustin Carlino. 2022. "Jittering: A Computationally Efficient Method for Generating Realistic Route Networks from Origin-Destination Data." *Findings*, April. https://doi.org/10.32866/001c.33873.

Lovelace, Robin, and Malcolm Morgan. 2024. *Od: Manipulate and Map Origin-Destination Data*. https://github.com/itsleeds/od.

Lovelace, Robin, Jakub Nowosad, and Jannes Muenchow. 2024. *Geocomputation with r*. Second. Chapman; Hall/CRC. https://r.geocompx.org/.

Pebesma, Edzer, and Roger Bivand. 2023. *Spatial Data Science: With Applications in R*. Boca Raton: Chapman; Hall/CRC. https://doi.org/10.1201/9780429459016.

Pereira, Rafael H. M., Marcus Saraiva, Daniel Herszenhut, Carlos Kaue Vieira Braga, and Matthew Wigginton Conway. 2021. "R5r: Rapid Realistic Routing on Multimodal Transport Networks with r^5 in r." *Findings*, March. https://doi.org/10.32866/001c.21262.

Pereira, Rafael HM, and Daniel Herszenhut. 2023. *Introduction to Urban Accessibility: A Practical Guide with r*. Instituto de Pesquisa Econômica Aplicada (Ipea). https://ipeagit.github.io/intro_access_book/.

Zomorrodi, Ryan. 2024. *Centr: Weighted and Unweighted Spatial Centers*. https://ryanzomorrodi.github.io/centr/.