

# **Quantitative Methods of Analysis in Transportation**

## **Using R and GIS**

Rosa Félix

Gabriel Valençá

Filipe Moura



2025

# Quantitative Methods of Analysis in Transportation

Using R and GIS

Rosa Félix

*rosamfelix@tecnico.pt*

Gabriel Valença

*gabrielvalenca@tecnico.pt*

Filipe Moura

*fmoura@tecnico.pt*

November 2025

see [Online Version](#)

QUANTITATIVE METHODS OF ANALYSIS IN TRANSPORTATION  
MSc IN TRANSPORTATION SYSTEMS  
INSTITUTO SUPERIOR TÉCNICO  
UNIVERSITY OF LISBON



# Table of contents

<b>1. Quantitative Methods of Analysis in Transportation course</b>	<b>1</b>
<b>2. Introduction</b>	<b>3</b>
Why R and GIS . . . . .	3
Recommended readings . . . . .	3
<b>3. Software</b>	<b>5</b>
3.1. R . . . . .	5
3.1.1. Windows . . . . .	5
3.1.2. Mac . . . . .	5
3.1.3. Ubuntu . . . . .	6
3.2. RStudio . . . . .	6
3.2.1. Windows 10/11 . . . . .	7
3.2.2. MacOS . . . . .	7
3.2.3. Ubuntu . . . . .	7
3.3. R packages . . . . .	7
3.4. TinyTex . . . . .	8
3.5. Open Route Service . . . . .	8
<b>I. Data and Models</b>	<b>9</b>
<b>4. R basics</b>	<b>11</b>
4.1. Math operations . . . . .	11
4.1.1. Sum . . . . .	11
4.1.2. Subtraction . . . . .	11
4.1.3. Multiplication . . . . .	11
4.1.4. Division . . . . .	11
4.1.5. Round the number . . . . .	12
4.2. Basics . . . . .	12
4.2.1. Combinations . . . . .	12
4.2.2. Create a comment with <code>ctrl + shift + c</code> . . . . .	12
4.2.3. Create a table . . . . .	13
4.3. Practical exercise . . . . .	13
4.3.1. Import dataset . . . . .	14
4.3.2. Take a first look at the data . . . . .	14
4.3.3. Explore the data . . . . .	15
4.3.4. Modify original data . . . . .	16
4.3.5. Export data . . . . .	17
4.3.6. Import data . . . . .	17

*Table of contents*

<b>5. Data manipulation</b>	<b>19</b>
5.1. Select variables . . . . .	19
5.1.1. Using pipes! . . . . .	20
5.2. Filter observations . . . . .	20
5.3. Create new variables . . . . .	21
5.4. Change data type . . . . .	21
5.4.1. Factors . . . . .	22
5.5. Join data tables . . . . .	23
5.6. group_by and summarize . . . . .	25
5.7. Arrange data . . . . .	27
5.8. All together now! . . . . .	27
5.9. Other dplyr functions . . . . .	28
<b>6. Exploratory Data Analysis</b>	<b>31</b>
6.1. Intro . . . . .	31
6.2. Load packages . . . . .	31
6.3. Dataset . . . . .	32
6.3.1. Import dataset . . . . .	32
6.4. Summary statistics . . . . .	34
6.5. Missing data . . . . .	35
6.6. Outliers . . . . .	35
6.6.1. Treating outliers . . . . .	37
6.7. Histograms . . . . .	39
6.8. Correlations . . . . .	44
<b>7. Multiple Linear Regression</b>	<b>51</b>
7.1. Load packages . . . . .	51
7.2. Dataset . . . . .	51
7.2.1. Import dataset . . . . .	52
7.3. Check the assumptions . . . . .	53
7.4. Assumption 1: Normal distribution . . . . .	53
7.5. Assumption 2: Linear relationship . . . . .	55
7.6. Assumption 3: No multicollinearity . . . . .	57
7.6.1. Declare the model . . . . .	57
7.6.2. Assessing the model . . . . .	59
7.6.3. Calculate the Variance Inflation Factor (VIF) . . . . .	59
7.6.4. Condition index . . . . .	60
7.7. Assumption 4: Independence of observations . . . . .	60
7.8. Assumption 5: Constant Variance (Homoscedasticity) . . . . .	61
7.9. Assumption 6: Residuals are normally distributed . . . . .	63
<b>8. Factor Analysis</b>	<b>65</b>
8.1. Load packages . . . . .	65
8.2. Dataset . . . . .	65
8.2.1. Import dataset . . . . .	66
8.2.2. Get to know your dataset . . . . .	66
8.3. Assumptions for factorial analysis . . . . .	69
8.3.1. Normality . . . . .	69

8.3.2. Linearity . . . . .	70
8.3.3. Homogeneity . . . . .	71
8.3.4. Correlations between variables . . . . .	71
8.4. Determine the number of factors to extract . . . . .	72
8.4.1. Parallel Analysis . . . . .	73
8.4.2. Kaiser Criterion . . . . .	73
8.4.3. Principal Component Analysis (PCA) . . . . .	74
8.5. Exploratory Factor Analysis (EFA) . . . . .	75
8.5.1. Rotational indeterminacy . . . . .	75
8.5.2. Factor scores and factor loadings . . . . .	80
8.5.3. Visualize Rotation . . . . .	81
<b>9. Cluster Analysis</b>	<b>85</b>
9.1. Load packages . . . . .	85
9.2. Dataset . . . . .	85
9.2.1. Import dataset . . . . .	86
9.2.2. Get to know your dataset . . . . .	86
9.2.3. Prepare data . . . . .	89
9.3. Hierarchical Clustering . . . . .	90
9.3.1. Distance measures . . . . .	90
9.3.2. Types of hierarchical clustering . . . . .	91
9.3.3. Comparing results from different hierarchical methods . . . . .	96
9.3.4. Silhouette Plots . . . . .	99
9.4. Non-Hierarchical Clustering . . . . .	102
9.4.1. Choose the number of clusters . . . . .	102
9.4.2. Predetermined number of clusters . . . . .	103
9.4.3. Plotting the clusters . . . . .	105
<b>II. GIS in R</b>	<b>107</b>
<b>10. Introduction to spatial data</b>	<b>109</b>
10.1. Import vector data . . . . .	109
10.1.1. Projected vs Geographic Coordinate Systems . . . . .	109
10.2. Join geometries to data frames . . . . .	111
10.3. Create spatial data from coordinates . . . . .	112
10.4. Visuzlize spatial data . . . . .	113
10.5. Export spatial data . . . . .	115
<b>11. Interactive maps</b>	<b>117</b>
11.1. Mapview . . . . .	118
11.1.1. Export . . . . .	120
11.2. Rmarkdown . . . . .	121
11.3. Flowmap blue . . . . .	122
<b>12. Centroids of transport zones</b>	<b>125</b>
12.1. Geometric centroids . . . . .	125
12.2. Weighted centroids . . . . .	127

## *Table of contents*

12.3. Compare centroids in a map . . . . .	128
12.3.1. Interactive map . . . . .	128
12.3.2. Static map . . . . .	128
<b>13. OD pairs and desire lines</b>	<b>131</b>
13.1. Desire lines with <code>od_to_sf</code> . . . . .	132
13.1.1. Filtering desire lines . . . . .	133
13.2. Oneway desire lines . . . . .	135
13.3. Using population centroids . . . . .	137
<b>14. Euclidean and routing distances</b>	<b>139</b>
14.1. Euclidean distances . . . . .	139
14.1.1. Import survey data frame convert to sf . . . . .	139
14.1.2. Create new point at the university . . . . .	139
14.1.3. Straight lines . . . . .	140
14.1.4. Distance . . . . .	141
14.2. Routing Engines . . . . .	142
14.3. Routing distances with <code>openrouteservice</code> . . . . .	143
14.3.1. Distances by car . . . . .	143
14.3.2. Distances by foot . . . . .	144
14.3.3. Distances by bike . . . . .	145
14.4. Compare distances . . . . .	146
14.4.1. Circuitry . . . . .	147
14.5. Visualize routes . . . . .	148
<b>15. Buffers vs. Isochrones</b>	<b>151</b>
15.1. Buffer . . . . .	151
15.2. Isochrone . . . . .	152
15.2.1. Isochrone from 1 point - distance . . . . .	152
15.2.2. Isochrone from more than 1 point - time . . . . .	153
<b>16. Open transportation data</b>	<b>157</b>
16.1. Road Networks . . . . .	157
16.1.1. OpenStreetMap . . . . .	157
16.1.2. HOT Export Tool . . . . .	157
16.1.3. OSM in R . . . . .	158
16.2. Transportation Services' Data . . . . .	159
16.2.1. GTFS . . . . .	159
16.2.2. National Access Points . . . . .	161
<b>17. Grids</b>	<b>163</b>
17.1. Make grid . . . . .	163
17.2. Count points in polygons . . . . .	164
<b>References</b>	<b>167</b>

# 1. Quantitative Methods of Analysis in Transportation course

Materials, exercises, data and tutorials for the [Quantitative Methods of Analysis in Transportation](#) course of the MSc in Transportation Systems at Instituto Superior Técnico - University of Lisbon.



TÉCNICO  
LISBOA

MESTRADO  
*EM SISTEMAS DE  
TRANSPORTES*

🔥 This is not the official book for the course.

Please refer to Fénix: [Quantitative Methods of Analysis in Transportation](#)



## 2. Introduction

This book aims to provide tools to deal with transportation datasets using [R programming](#), an open-source and widely used tool for data analytics in urban mobility.

It is based on exercises and examples to explore in **R**, with a focus on modelling and spatial data analysis using GIS techniques.

### 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 of enormous relevance when doing accessibility analysis or dealing with georeferenced 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

### Recommended readings

- Engel (2023) [Introduction to R](#).
- Wickham, Çetinaka-Rundel, and Grolemund (2017) [R for Data Science](#).
- Lovelace, Nowosad, and Muenchow (2024) [Geocomputation with R](#).



# 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<sup>1</sup> 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 4.5.1, but you can use >= 4.1.x if you already have it installed.

### 3.1.1. Windows

[Download R-4.5.1 for Windows](#) and run the executable file.

#### Important

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-4.5.1 for MacOX](#). You will have to choose between the *arm64* or the *x86-64* version.

Download the .pkg file and install it as usual.

---

<sup>1</sup>We will use RStudio, although if you already use other studio such as VScode, that's also fine.

### 3. Software

#### 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 /etc/apt/trusted.gpg.d/cran_ubuntu_key.asc  
# 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)-cran40/"
```

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 2025.09 and run the executable file.

### 3.2.2. MacOS

Download RStudio 2025.09 and install it as usual.

### 3.2.3. Ubuntu

! These are instructions for Ubuntu 24. If you use other linux distribution, please follow the instructions on [Posit RStudio](#).

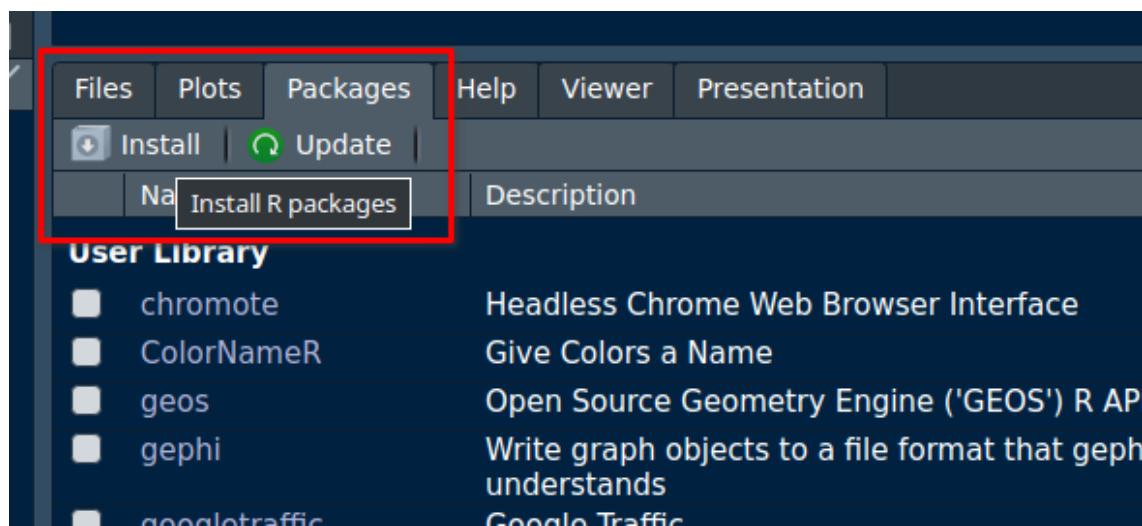
Install it via the terminal:

```
sudo apt install libssl-dev libclang-dev
wget
↳ https://download1.rstudio.org/electron/jammy/amd64/rstudio-2025.09.0-387-amd64.deb
sudo dpkg -i rstudio*
rm -v rstudio*
```

## 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:

### 3. Software

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

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

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

## 3.4. TinyTex

To be able to export your Rmardown document to a **pdf** file, you will need a LaTeX processor. **tinytex** is a lightweight R package with the most used features of **LaTeX** that you can use with R.

You can install it from RStudio console by running:

```
# you need to run both lines
install.packages("tinytex")
tinytex::install_tinytex()
```

To use some LaTeX packages, you also need to install **tlmgr**.

```
tinytex::tlmgr_install()
tinytex::tlmgr_update() # updates all latex pkgs
```

## 3.5. Open Route Service

For some GIS exercises, we will use the [Open Route Service](#) to calculate routes and travel times.

In RStudio console, run:

```
# install.packages("openrouteservice")

# instructions to setup at the U-Shift server
options(openrouteservice.url = "https://...") # the server address
  ↵ provided to you
openrouteservice::ors_api_key("") # empty key
```

**Part I.**

**Data and Models**



## 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. R basics

### 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. Basics

### 4.2.1. 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
```

#### 💡 Try it yourself

Try to write a combination with the numbers 10, 11, 56, 57, 58.

### 4.2.2. Create a comment with `ctrl + shift + c`

```
# Comments help you organize your code.  
# A line with a comment will not be executed.
```

### 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) # key sensitive (uppercase and lowercase  
← letters are different)
```

#### 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 from 0.
```

```
  modes trips  
1   car    200
```

#### Look at first row and column

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

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

```
table_example[1,4]
```

```
NULL
```

### 4.3. Practical exercise

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

## 4. R basics

### 4.3.1. Import dataset

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

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

```
Classes 'tbl_df', 'tbl' and '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 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 ...
```

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

```
nrow(data)
```

```
[1] 315
```

```
ncol(data)
```

```
[1] 8
```

Open the dataset

```
View(data)
```

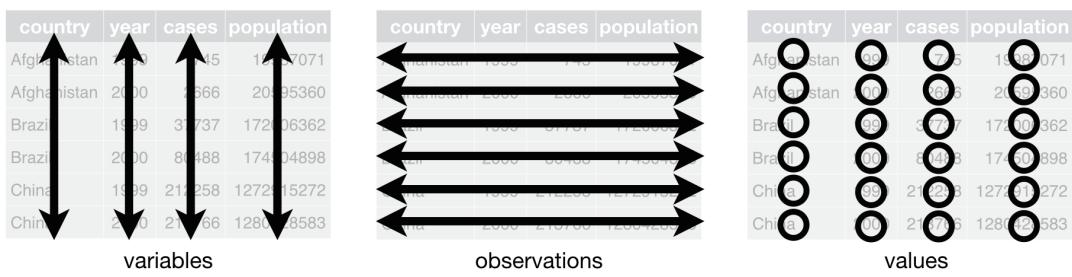


Figure 4.1.: The following three rules make a dataset tidy: variables are columns, observations are rows, and values are cells. [@wickham17r]

### 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
```

#### 4. R basics

```
[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_mun" "Total"          "Walk"           "Bike"  
[5] "Car"             "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<sup>1</sup>.

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)
```

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

---

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

## 5. Data manipulation

```
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 `%>%`. <sup>2</sup> 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.

## 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.

---

<sup>2</sup>You can change this in RStudio > Tools > Global Options > Code.

```
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.

```
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"
```

## 5. Data manipulation

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.

```
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
```

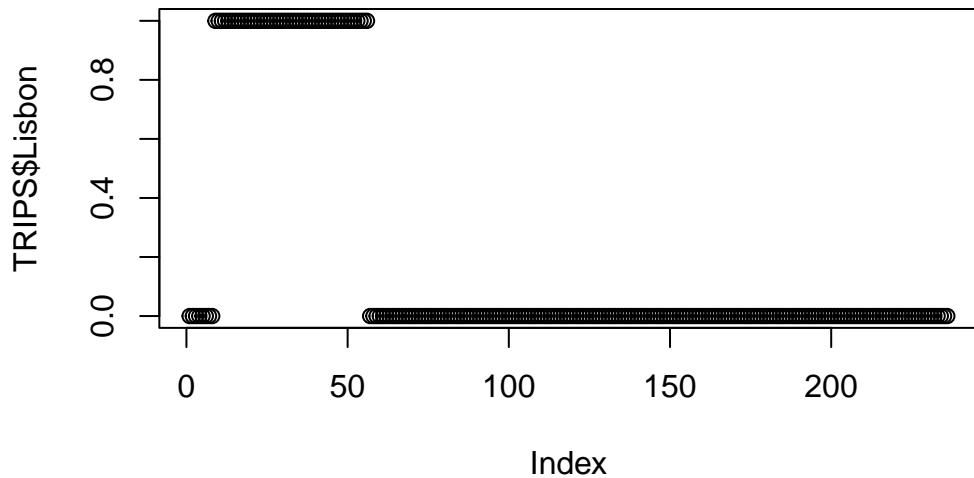
```
table(TRIPS$Lisbon_factor)
```

No	Yes
188	48

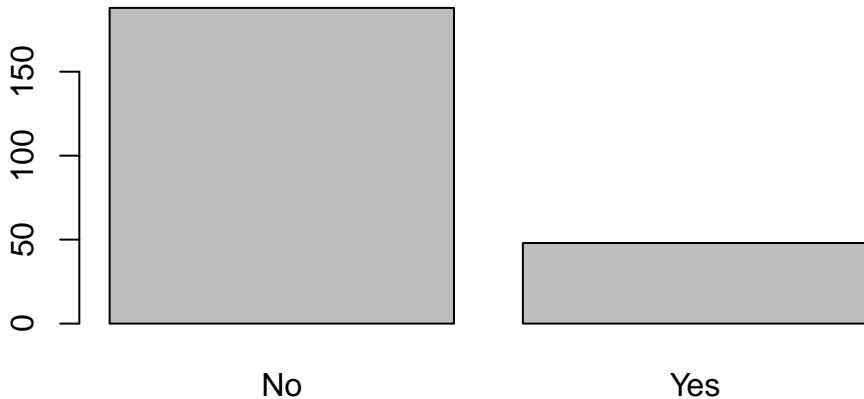
The first number to appear is the first level, and so on.

You can see the difference between using a continuous variable (in this case `Lisbon` has 0 and 1) and a categorical variable (`Lisbon_factor`).

```
plot(TRIPS$Lisbon) # the values range between 0 and 1
```



```
plot(TRIPS$Lisbon_factor) # the values are categorical and labeled with
                           ↪ Yes/No
```



## 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:

## 5. Data manipulation

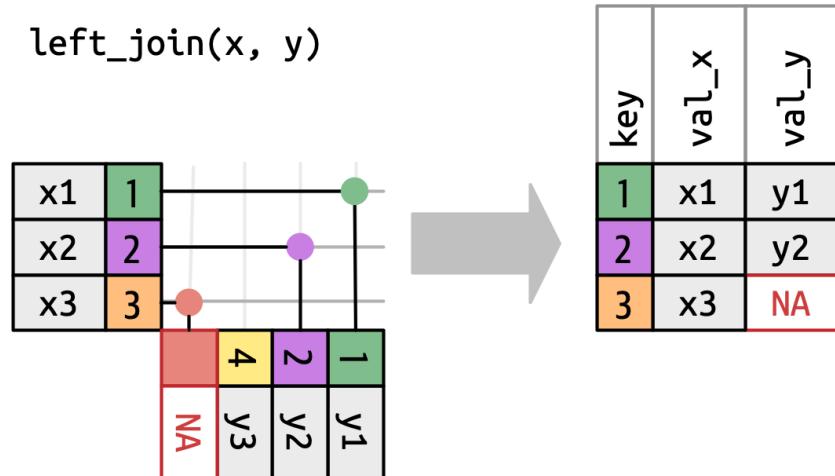


Figure 5.1.: A visual representation of the left join where every row in x appears in the output. Source: R for Data Science.

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>  
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)
```

## 5. Data manipulation

```
# 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 change 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) |>
```

## 5. Data manipulation

```
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**:

# Data transformation with dplyr :: CHEATSHEET

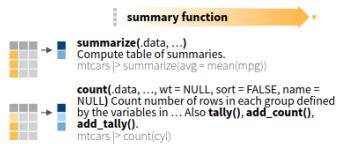


dplyr functions work with pipes and expect **tidy data**. In tidy data:



## Summarize Cases

Apply **summary** functions to columns to create a new table of summary statistics. Summary functions take vectors as input and return one value (see back).



## Group Cases

Use **group\_by**(data, ..., add = FALSE, drop = TRUE) to create a "grouped" copy of a table grouped by columns in ... dplyr functions will manipulate each "group" separately and combine the results.



Use **rowwise**(data, ...) to group data into individual rows. dplyr functions will compute results for each row. Also apply functions to list-columns. See tidyverse cheat sheet for list-column workflow.



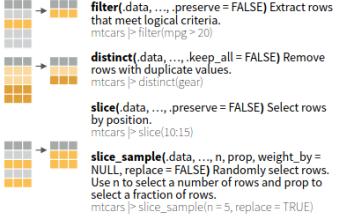
**ungroup(x, ...)** Returns ungrouped copy of table.  
g\_mtcars <- mtcars %> group\_by(cyl)  
ungroup(g\_mtcars)



## Manipulate Cases

### EXTRACT CASES

Row functions return a subset of rows as a new table.

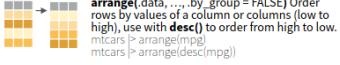


### Logical and boolean operators to use with filter()

== < <= is.na() %in% | xor()  
!= > >= is.na() ! &

See `?base::Logic` and `?Comparison` for help.

### ARRANGE CASES



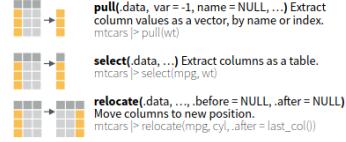
### ADD CASES



## Manipulate Variables

### EXTRACT VARIABLES

Column functions return a set of columns as a new vector or table.

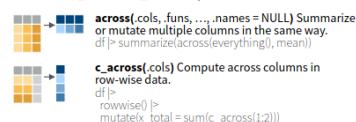


Use these helpers with `select()` and `across()`:

e.g. `mtcars %> select(mpg:cyl)`  
`contains(match)` `num_range(prefix, range)`; e.g. `mpg:cyl`  
`ends_with(match)` `all_of(...)/any_of(..., vars)`; e.g. `!gear`  
`starts_with(match)` `matches(match)` `everything()`

### MANIPULATE MULTIPLE VARIABLES AT ONCE

df <- tibble(x\_1 = c(1, 2), x\_2 = c(3, 4), y = c(4, 5))

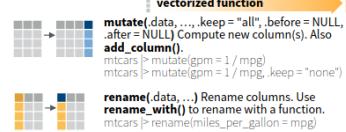


`c_across(.cols)` Compute across columns in row-wise data.

`df[+]` `rowwise()` > `mutate(x_total) = sum(c_across(1:2))`

### MAKE NEW VARIABLES

Apply **vectorized functions** to columns. Vectorized functions take vectors as input and return vectors of the same length as output (see back).



`mutate(data, ..., keep = "all", before = NULL, after = NULL)` Compute new column(s). Also `add_column()`

`mtcars %> mutate(gpm = 1 / mpg)`  
`mtcars %> mutate(gpm = 1 / mpg, keep = "none")`

`rename(data, ...)` Rename columns. Use `rename_with()` to rename with a function.

`mtcars %> rename(miles_per_gallon = mpg)`

CC BY SA Posit Software, PBC • info@posit.co • posit.co • Learn more at [dplyr.tidyverse.org](https://dplyr.tidyverse.org) • HTML cheatsheets at [pos.it/cheatsheets](https://pos.it/cheatsheets) • dplyr 1.1.4 • Updated: 2024-05

Take a particular attention to **pivot\_wider** and **pivot\_longer** (**tidyverse** 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. Exploratory Data Analysis

## 6.1. Intro

This chapter will show you how to use visualization and transformation to explore your data in a systematic way, a task that statisticians call exploratory data analysis, or **EDA** for short. EDA is an iterative cycle, where you should feel free to “fall in love” with your data.

You:

1. Generate questions about your data.
2. Search for answers by visualizing, transforming, and modelling your data.
3. Use what you learn to refine your questions and/or generate new questions.

Note: EDA is not a formal process with a strict set of rules!

See Wickham, Çetinaka-Rundel, and Grolemund (2017) for more, in particular the EDA chapter.

### 💡 Do it yourself with R

Copy the script [ExploratoryDataAnalysis.R](#) and paste it in your session.  
Run each line using CTRL + ENTER

## 6.2. Load packages

```
library(tidyverse) # Pack of most used libraries for data science
library(skimr) # summary of the data
library(DataExplorer) # exploratory data analysis
library(corrplot) # correlation plots
```

## 6. Exploratory Data Analysis

### 6.3. Dataset

The database used in this example is a treated database from the Mobility Survey for the metropolitan areas of Lisbon and Porto in 2018 (INE 2018). We will only focus on trips within the metropolitan area of Lisbon.

Included variables:

- `Origin_dicofre16` - Code of Freguesia (district) as set by INE after 2016 (Distrito + Concelho + Freguesia), for trip origin
- `Total` - number of trips with origin at each district
- `Walk` - number of walking trips
- `Bike` - number of bike trips
- `Car` - number of car trips. Includes taxi and motorcycle.
- `PTtransit` - number of Public Transit trips
- `Other` - number of other trips (truck, van, tractor, aviation)
- `Distance` - average trip distance (km)
- `Duration` - average trip duration (minutes)
- `Car_perc` - percentage of car trips
- `N_INDIVIDUOS` - number of residents (INE 2022)
- `Male_perc` - percentage of male residents (INE 2022)
- `IncomeHH` - average household income
- `Nvehicles` - average number of car/motorcycle vehicles in the household
- `DrivingLic` - percentage of car driving licence holders
- `CarParkFree_Work` - percentage of respondents with free car parking at the work location
- `PTpass` - percentage of public transit monthly pass holders
- `internal` - binary variable (factor). Yes: trip with same TAZ origin and destination, No: trips with different destination
- `Lisboa` - binary variable (factor). Yes: the district is part of Lisbon municipality, No: otherwise
- `Area_km2` - area of in `Origin_dicofre16`, in km<sup>2</sup>

#### 6.3.1. Import dataset

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

Take a look at the dataset

```
View(data) # open in table  
glimpse(data) # glimpse of the dataset
```

Rows: 236  
Columns: 20

```
$ Origin_dicofre16 <chr> "110501", "110501", "110506", "110506", "110507", "11~  
$ Total <dbl> 35539, 47602, 37183, 42313, 30725, 54586, 57747, 6788~  
$ Walk <dbl> 11325, 3502, 12645, 1418, 9389, 2630, 20423, 3573, 62~  
$ Bike <dbl> 1309, 416, 40, 163, 1481, 168, 1406, 809, 13, 37, 3, ~  
$ Car <dbl> 21446, 37727, 22379, 37337, 19654, 44611, 33044, 5225~  
$ PTransit <dbl> 1460, 5519, 2057, 3285, 201, 6963, 2477, 10534, 110, ~  
$ Other <dbl> 0, 437, 63, 106, 0, 215, 396, 717, 2, 169, 3, 824, 0,~  
$ Distance <dbl> 11.779, 11.779, 9.868, 9.868, 9.600, 9.600, 12.875, 1~  
$ Duration <dbl> 23.96, 23.96, 22.78, 22.78, 23.39, 23.39, 26.89, 26.8~  
$ Car_perc <dbl> 60.34497, 79.25507, 60.18611, 88.24002, 63.96745, 81.~  
$ N_INDIVIDUOS <dbl> 44165, 44165, 59238, 59238, 46529, 46529, 64192, 6419~  
$ Male_perc <dbl> 47.66, 47.66, 47.12, 47.12, 45.54, 45.54, 46.14, 46.1~  
$ IncomeHH <dbl> 2204.32, 2204.32, 1896.05, 1896.05, 2430.43, 2430.43,~  
$ Nvehicles <dbl> 1.86, 1.86, 1.73, 1.73, 1.60, 1.60, 1.64, 1.64, 1.20,~  
$ DrivingLic <dbl> 63.03, 63.03, 65.42, 65.42, 67.05, 67.05, 72.75, 72.7~  
$ CarParkFree_Work <dbl> 49.97, 49.97, 57.09, 57.09, 47.43, 47.43, 56.47, 56.4~  
$ PTpass <dbl> 11.14, 11.14, 17.12, 17.12, 18.25, 18.25, 19.73, 19.7~  
$ internal <fct> Yes, No, Yes, No, Yes, No, Yes, No, Yes, No,~  
$ Lisboa <fct> No, No, No, No, No, No, Yes, Yes, Yes, Yes, Y~  
$ Area_km2 <dbl> 39.767414, 39.767414, 20.364310, 20.364310, 8.109861,~
```

```
str(data) # Structure of the dataset
```

```
'data.frame': 236 obs. of 20 variables:  
$ Origin_dicofre16: chr "110501" "110501" "110506" "110506" ...  
$ Total : num 35539 47602 37183 42313 30725 ...  
$ Walk : num 11325 3502 12645 1418 9389 ...  
$ Bike : num 1309 416 40 163 1481 ...  
$ Car : num 21446 37727 22379 37337 19654 ...  
$ PTransit : num 1460 5519 2057 3285 201 ...  
$ Other : num 0 437 63 106 0 215 396 717 2 169 ...  
$ Distance : num 11.78 11.78 9.87 9.87 9.6 ...  
$ Duration : num 24 24 22.8 22.8 23.4 ...  
$ Car_perc : num 60.3 79.3 60.2 88.2 64 ...  
$ N_INDIVIDUOS : num 44165 44165 59238 59238 46529 ...  
$ Male_perc : num 47.7 47.7 47.1 47.1 45.5 ...  
$ IncomeHH : num 2204 2204 1896 1896 2430 ...  
$ Nvehicles : num 1.86 1.86 1.73 1.73 1.6 1.6 1.64 1.64 1.2 1.2 ...  
$ DrivingLic : num 63 63 65.4 65.4 67 ...  
$ CarParkFree_Work: num 50 50 57.1 57.1 47.4 ...  
$ PTpass : num 11.1 11.1 17.1 17.1 18.2 ...  
$ internal : Factor w/ 2 levels "Yes","No": 1 2 1 2 1 2 1 2 1 2 ...  
$ Lisboa : Factor w/ 2 levels "No","Yes": 1 1 1 1 1 1 1 1 2 2 ...  
$ Area_km2 : num 39.77 39.77 20.36 20.36 8.11 ...
```

## 6. Exploratory Data Analysis

### 6.4. Summary statistics

Have an overview of the variables and their statistics' summary.

```
summary(data) # Check the summary statistics
```

```
Origin_dicofre16      Total          Walk          Bike
Length:236    Min. : 361   Min. : 0.0   Min. : 0.00
Class :character  1st Qu.: 5918   1st Qu.: 763.2  1st Qu.: 0.00
Mode :character   Median :17474   Median :3125.0  Median : 13.50
               Mean :22457   Mean :5383.4   Mean : 107.03
               3rd Qu.:33378   3rd Qu.:8298.5  3rd Qu.: 98.75
               Max. :112186   Max. :32646.0  Max. :2040.00
Car            PTransit       Other        Distance
Min. : 0   Min. : 0   Min. : 0.0   Min. : 6.835
1st Qu.: 3243 1st Qu.: 249  1st Qu.: 2.0   1st Qu.: 9.539
Median : 9008 Median :1057   Median : 44.0  Median :10.323
Mean :13289  Mean :3474   Mean :203.4  Mean :11.139
3rd Qu.:21249 3rd Qu.:4853  3rd Qu.:281.5 3rd Qu.:12.097
Max. :52631  Max. :41672  Max. :2391.0  Max. :22.660
Duration      Car_perc      N_INDIVIDUOS  Male_perc
Min. :16.30  Min. : 0.00  Min. : 1566  Min. :44.61
1st Qu.:23.00 1st Qu.:45.40  1st Qu.:11060 1st Qu.:46.58
Median :24.70 Median :62.62  Median :20855  Median :47.51
Mean :25.42  Mean :59.00  Mean :24324  Mean :47.54
3rd Qu.:27.73 3rd Qu.:75.72  3rd Qu.:36079 3rd Qu.:48.29
Max. :37.42  Max. :99.27  Max. :68649  Max. :55.94
IncomeHH      Nvehicles     DrivingLic    CarParkFree_Work
Min. : 884.5 Min. :1.020  Min. :37.04  Min. : 5.47
1st Qu.:1417.8 1st Qu.:1.350  1st Qu.:57.67  1st Qu.:40.39
Median :1594.7 Median :1.545  Median :63.00  Median :50.51
Mean :1732.6  Mean :1.530  Mean :62.50  Mean :49.30
3rd Qu.:1953.5 3rd Qu.:1.670  3rd Qu.:68.84  3rd Qu.:57.92
Max. :3462.3  Max. :2.420  Max. :80.79  Max. :87.60
PTpass        internal     Lisboa       Area_km2
Min. : 0.00  Yes:118  No :188  Min. : 1.494
1st Qu.:13.41 No :118  Yes: 48  1st Qu.: 5.044
Median :22.71                           Median : 11.598
Mean :23.82                           Mean : 25.553
3rd Qu.:32.94                          3rd Qu.: 28.511
Max. :60.45                           Max. :282.125
```

```
skim(data) # In a more organized way
```

Table 6.1.: Data summary

Name	data
Number of rows	236
Number of columns	20
Column type frequency:	
character	1
factor	2
numeric	17
Group variables	
	None

Variable type: character

skim_variable	n_missing	complete_rate	min	max	empty	n_unique	whitespace
Origin_dicofre16	0		1	6	0	118	0

Variable type: factor

skim_variable	n_missing	complete_rate	ordered	n_unique	top_counts
internal	0		1 FALSE	2	Yes: 118, No: 118
Lisboa	0		1 FALSE	2	No: 188, Yes: 48

Variable type: numeric

skim_variable	n_missing	complete_rate	mean	sd	p0	p25	p50	p75	p100	hist
Total	0	1	22457.00	19084.45	361.00	5917.75	17474.00	33377.50	112186.00	
Walk	0	1	5383.44	6224.84	0.00	763.25	3125.00	8298.50	32646.00	
Bike	0	1	107.03	248.65	0.00	0.00	13.50	98.75	2040.00	
Car	0	1	13289.24	12351.61	0.00	3243.00	9008.00	21248.75	52631.00	
PTransit	0	1	3473.79	5467.82	0.00	249.00	1057.00	4853.00	41672.00	
Other	0	1	203.45	336.04	0.00	2.00	44.00	281.50	2391.00	
Distance	0	1	11.14	2.66	6.84	9.54	10.32	12.10	22.66	
Duration	0	1	25.42	3.91	16.30	23.00	24.70	27.73	37.42	
Car_perc	0	1	59.00	21.56	0.00	45.40	62.62	75.72	99.27	
N_INDIVIDUOS	0	1	24323.80	16438.04	1566.00	11060.00	20855.00	36079.00	68649.00	
Male_perc	0	1	47.54	1.57	44.61	46.58	47.50	48.29	55.94	
IncomeHH	0	1	1732.55	453.11	884.46	1417.76	1594.73	1953.50	3462.32	
Nvehicles	0	1	1.53	0.24	1.02	1.35	1.54	1.67	2.42	
DrivingLic	0	1	62.50	8.12	37.04	57.67	63.00	68.84	80.79	
CarParkFree_Work	0	1	49.30	14.30	5.47	40.39	50.51	57.92	87.60	
PTpass	0	1	23.82	12.87	0.00	13.41	22.72	32.94	60.45	
Area_km2	0	1	25.55	42.58	1.49	5.04	11.60	28.51	282.13	

## 6.5. Missing data

Is there missing data (NA)? How many?

```
table(is.na(data))
```

```
FALSE  
4720
```

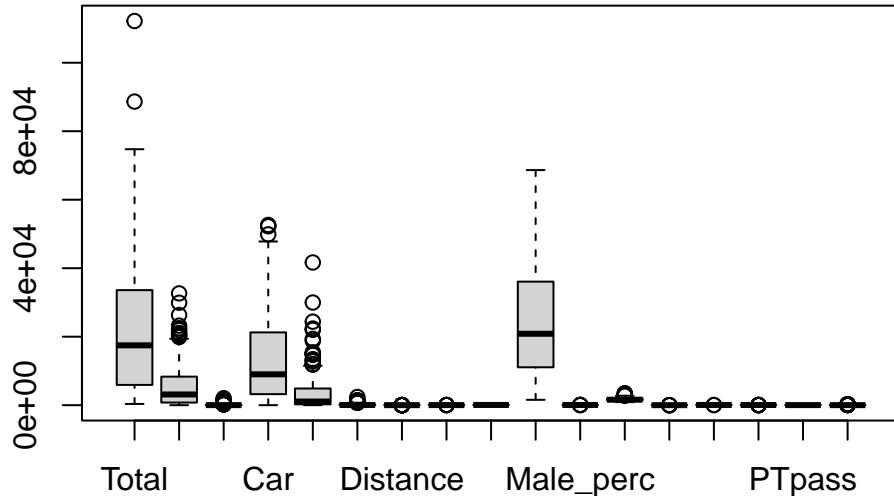
## 6.6. Outliers

Inspect outliers and distributions

```
boxplot(data) # This does now work if variables are not all continuous
```

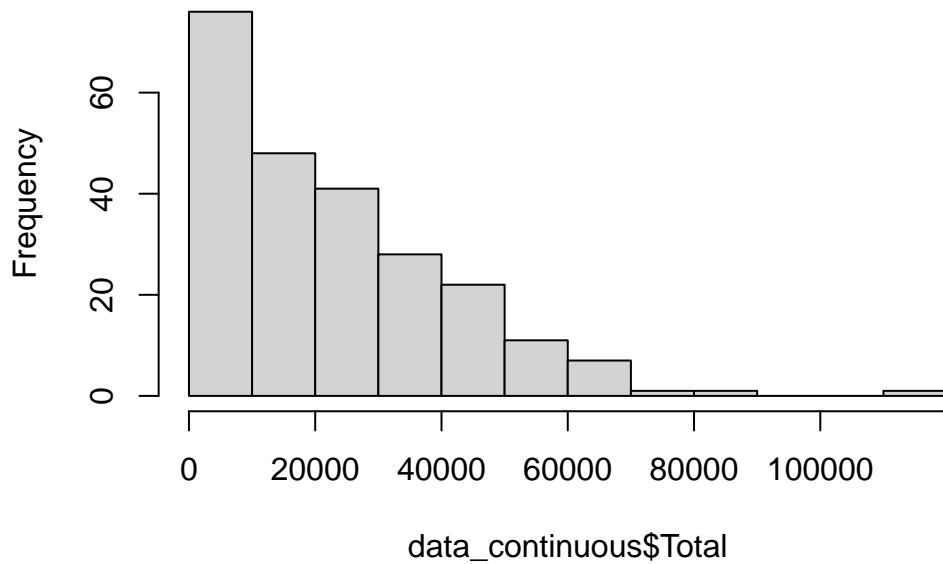
```
data_continuous = data |> select(-Origin_dicofre16, -internal, -Lisboa) #  
  ↵ Exclude categorical variables  
boxplot(data_continuous) # Exclude categorical variables
```

## 6. Exploratory Data Analysis

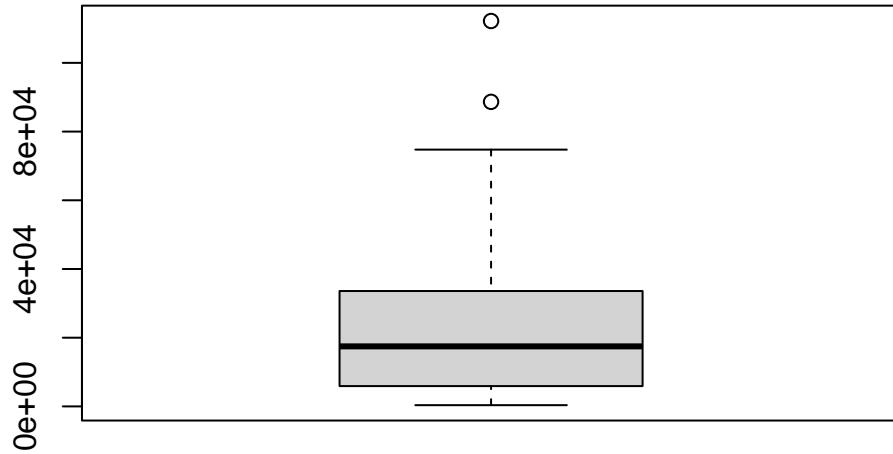


```
hist(data_continuous$Total) # histogram
```

Histogram of data\_continuous\$Total



```
boxplot(data_continuous$Total) # outliers detected
```



### 6.6.1. Treating outliers

Based on the theory, we can create a function to detect the outliers.

```
outlier = function(x) {
  q = quantile(x, probs = c(0.25, 0.75), na.rm = TRUE) # Q1 and Q3
  caps = quantile(x, probs = c(0.05, 0.95), na.rm = TRUE) # 5th and 95th
  ← percentile
  H = 1.5 * IQR(x, na.rm = TRUE) # interquartile range

  case_when(
    x < (q[1] - H) ~ caps[1], # replace values that are LESS than
    ← Q1-1.5*IQR with the P5 value
    x > (q[2] + H) ~ caps[2], # replace values that are MORE than
    ← Q3+1.5*IQR with the P95 value
    TRUE ~ x # otherwise, return the original value
  )
}
```

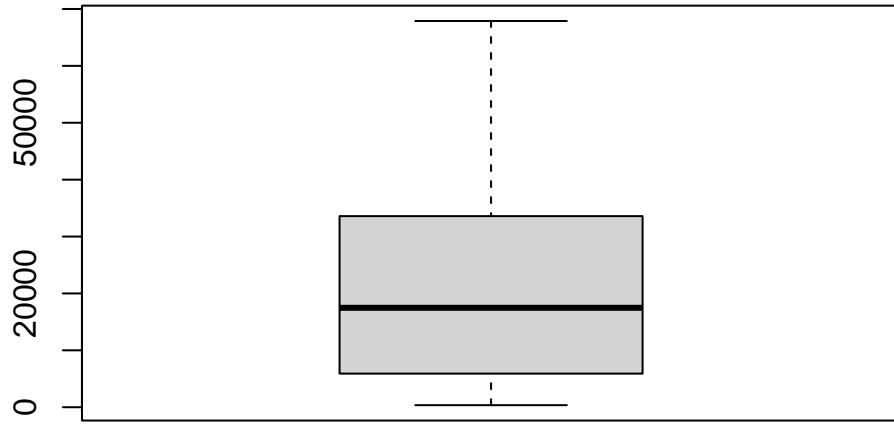
Now, use it in a copy of the table

```
data_outliers = data_continuous # duplicate the table
data_outliers$Total = outlier(data_outliers$Total) # Use the function to
← the same variable
```

Take a look again at the boxplot

```
boxplot(data_outliers$Total)
```

## 6. Exploratory Data Analysis



Compare the variable with and without the outliers

```
# Mean  
mean(data$Total)
```

```
[1] 22457
```

```
mean(data_outliers$Total)
```

```
[1] 22013.51
```

```
# Median  
median(data$Total)
```

```
[1] 17474
```

```
median(data_outliers$Total)
```

```
[1] 17474
```

```
# Standard deviation  
sd(data$Total)
```

```
[1] 19084.45
```

```
sd(data_outliers$Total)
```

```
[1] 17739.39
```

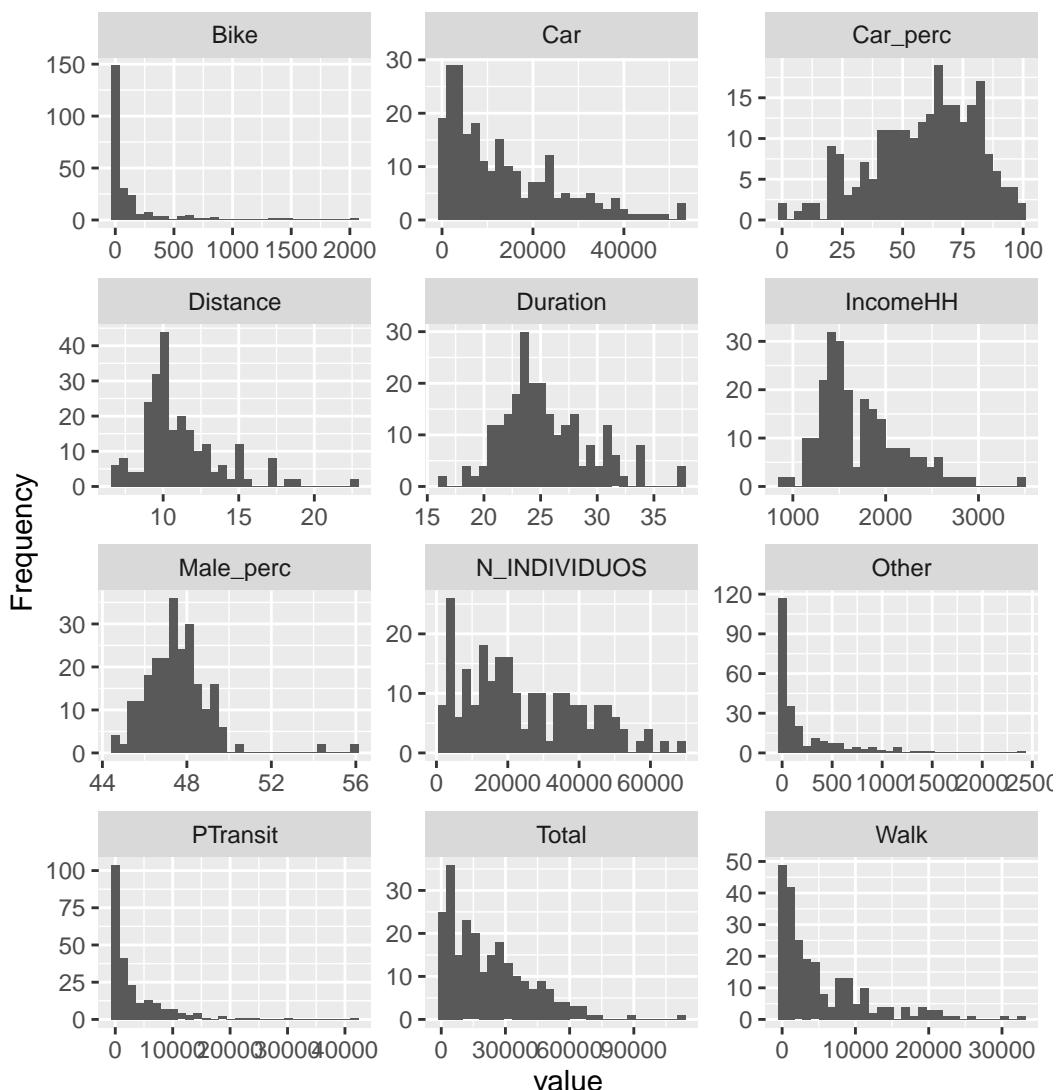
**i Note**

There are many methods to treat outliers. This is just one of them.

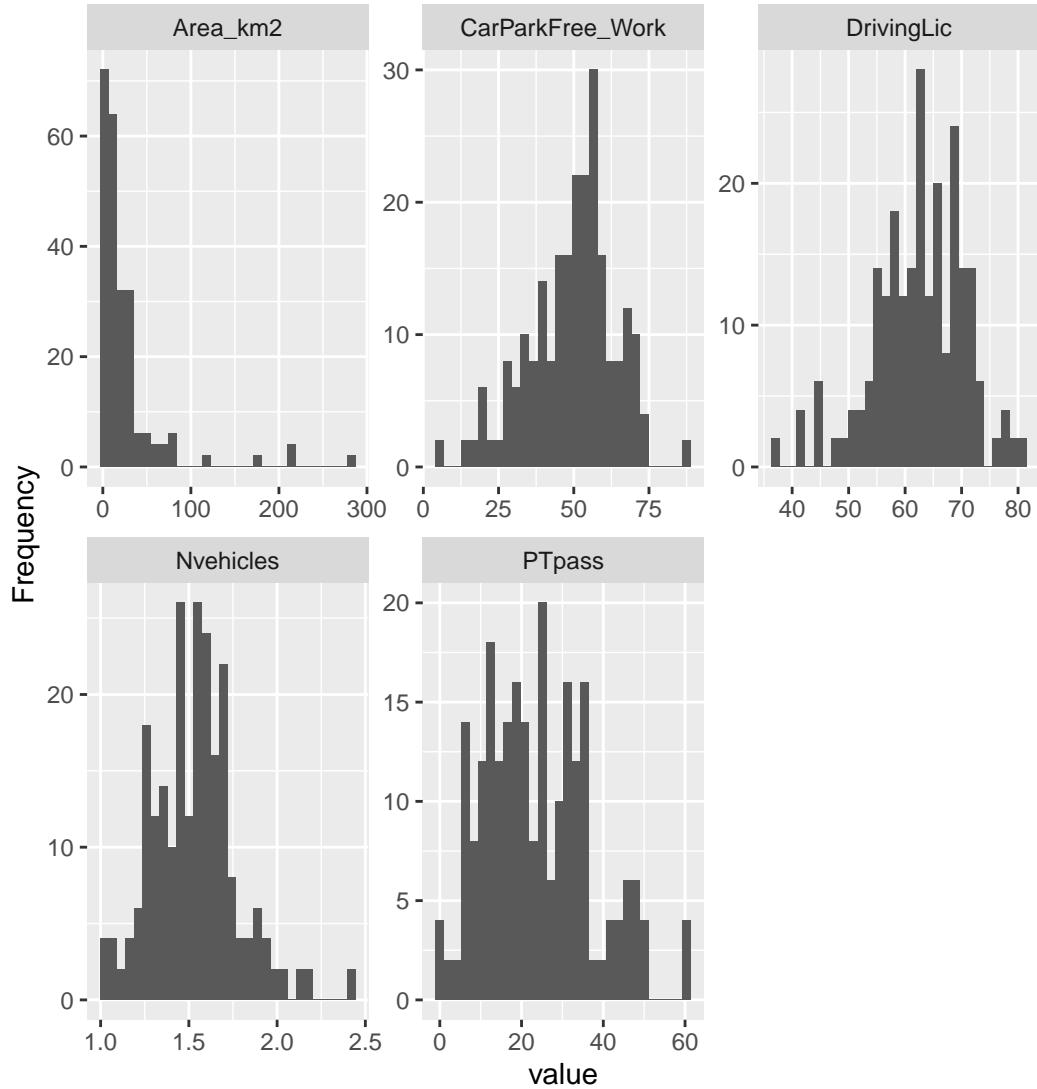
## 6.7. Histograms

Plot histograms of all the continuous variables

```
plot_histogram(data, ncol = 3) # histograms with 3 columns
```

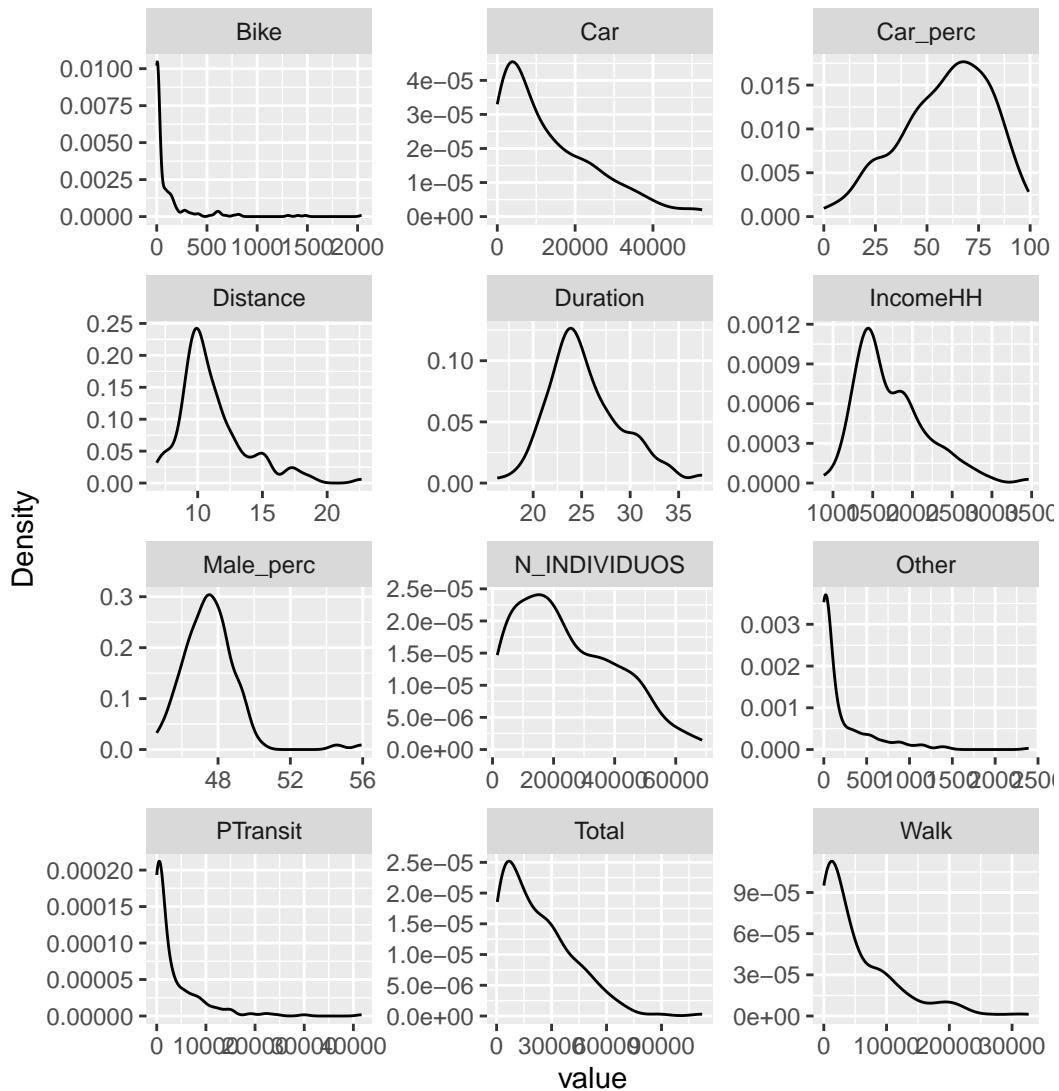


## 6. Exploratory Data Analysis

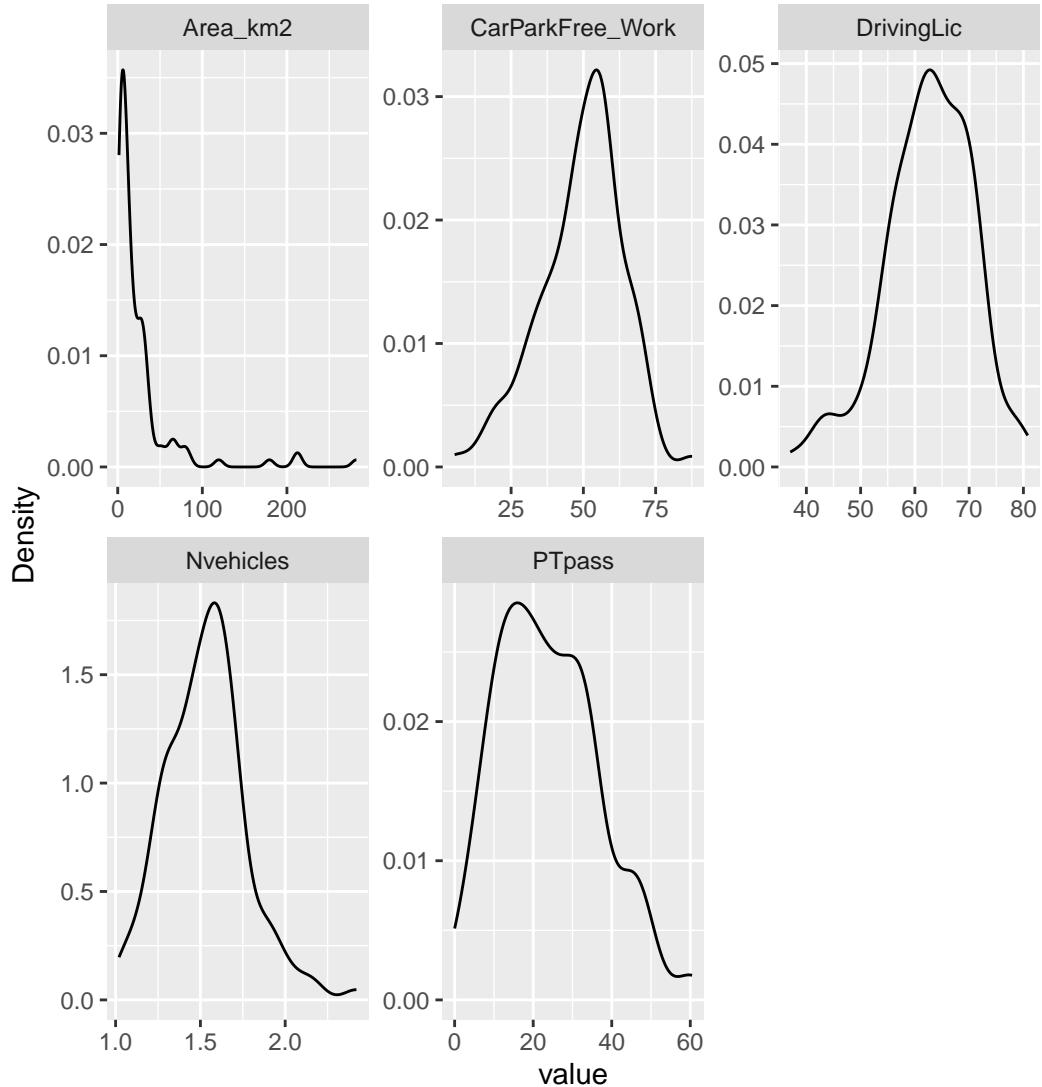


Page 2

```
plot_density(data, ncol = 3) # density plots
```



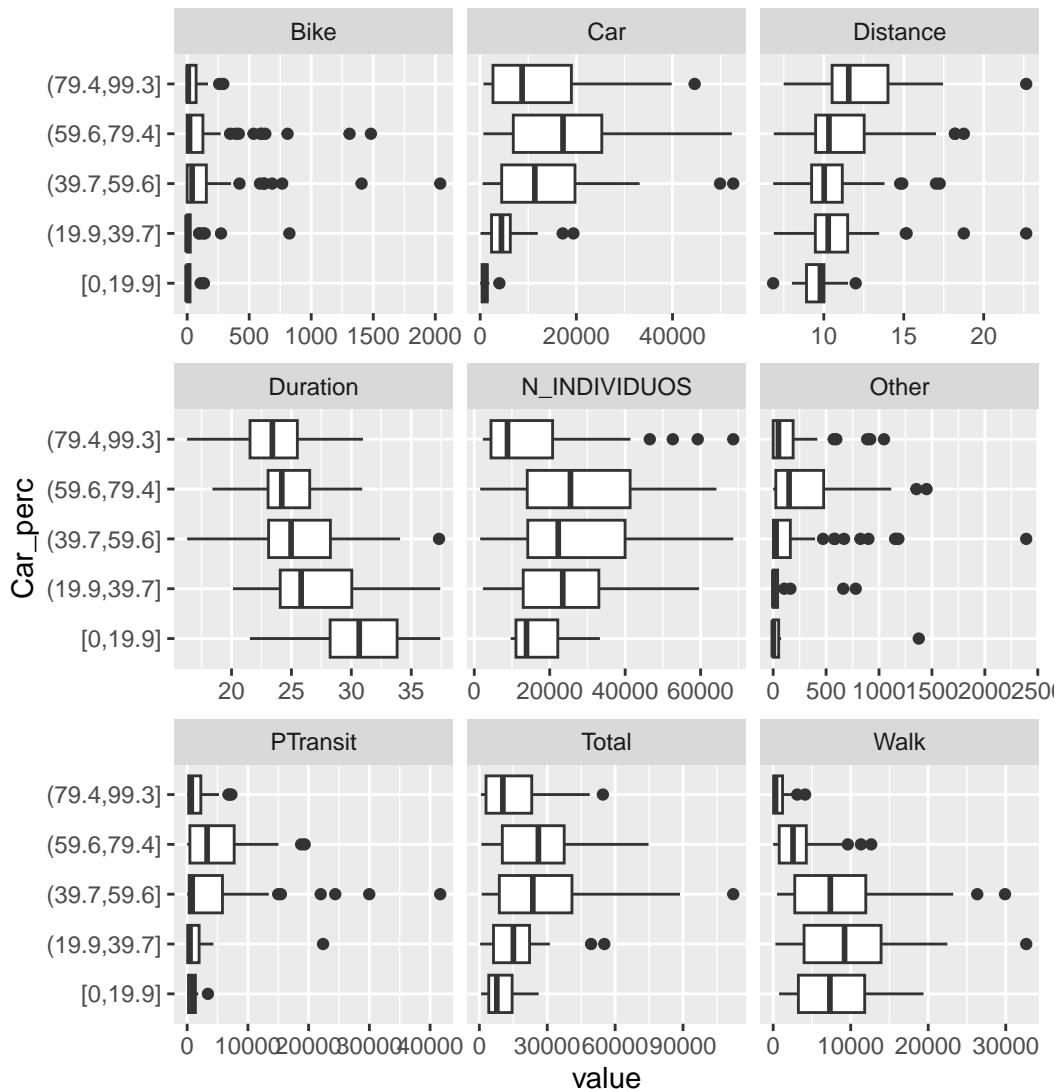
## 6. Exploratory Data Analysis



Page 2

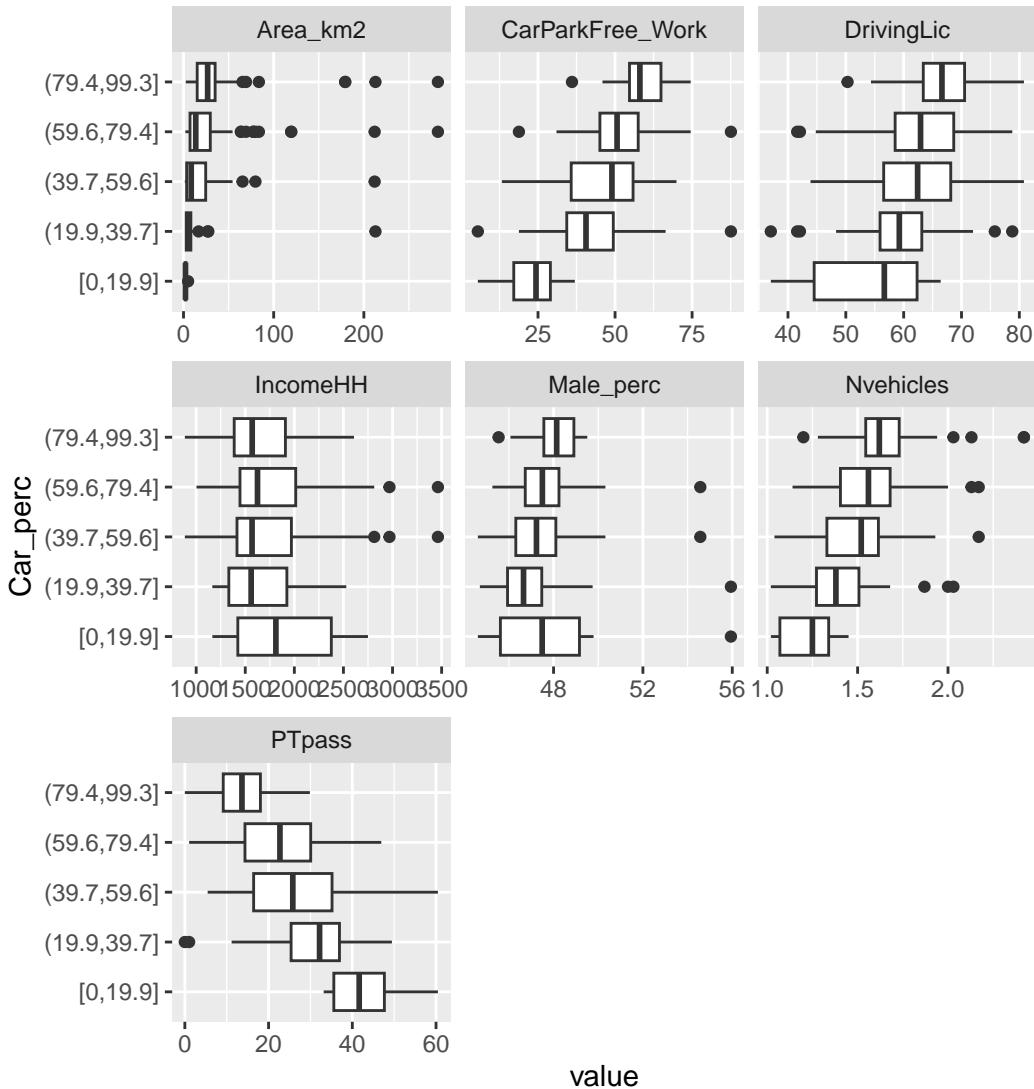
Check how other variables are distributed regarding `Car_perc`

```
plot_boxplot(data, by = "Car_perc", ncol = 3)
```



Page 1

## 6. Exploratory Data Analysis



Page 2

### **i** Note

When `Car_perc` increases, `PTpass` decreases.

### **?** Exercise

Try plotting the boxplots of each independent variable with `Car_perc`.

## 6.8. Correlations

Plot correlation heatmaps, between continuous variables.

```
# estimate correlation matrix
corrmat = cor(data_continuous, method = "pearson") |> round(2)
corrmat
```

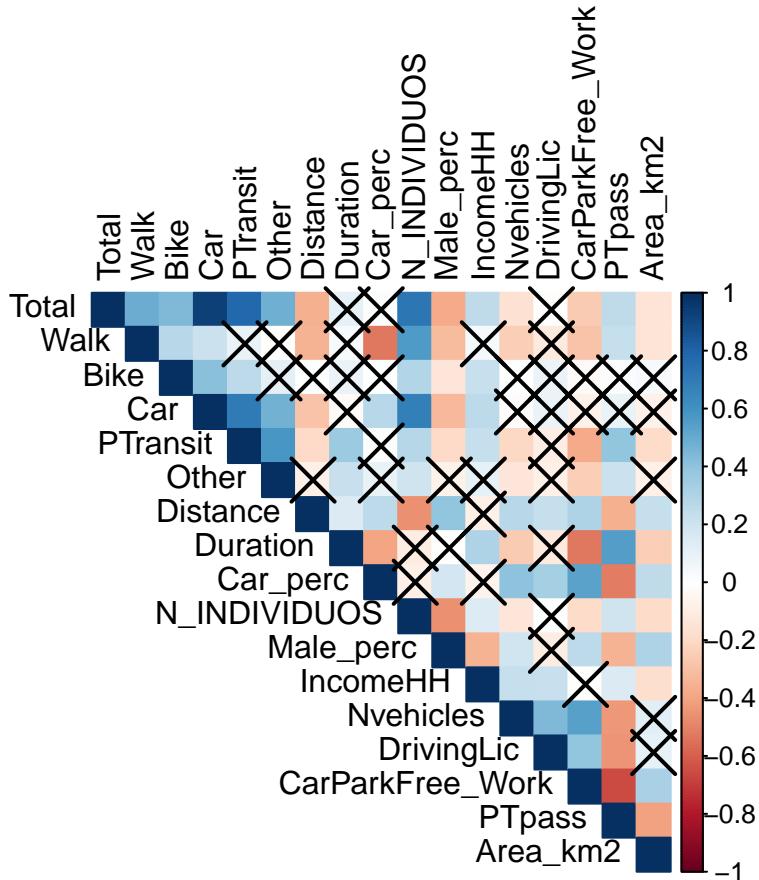
	Total	Walk	Bike	Car	PTransit	Other	Distance	Duration
Total	1.00	0.49	0.44	0.93	0.78	0.48	-0.36	0.10
Walk	0.49	1.00	0.27	0.21	0.09	-0.01	-0.34	0.03
Bike	0.44	0.27	1.00	0.41	0.26	0.11	0.00	0.08
Car	0.93	0.21	0.41	1.00	0.70	0.47	-0.29	-0.03
PTransit	0.78	0.09	0.26	0.70	1.00	0.58	-0.20	0.36
Other	0.48	-0.01	0.11	0.47	0.58	1.00	-0.09	0.22
Distance	-0.36	-0.34	0.00	-0.29	-0.20	-0.09	1.00	0.16
Duration	0.10	0.03	0.08	-0.03	0.36	0.22	0.16	1.00
Car_perc	0.01	-0.53	0.00	0.27	0.01	0.09	0.26	-0.40
N_INDIVIDUOS	0.72	0.56	0.29	0.68	0.28	0.20	-0.46	-0.11
Male_perc	-0.38	-0.32	-0.14	-0.33	-0.20	-0.08	0.39	-0.01
IncomeHH	0.25	0.04	0.22	0.26	0.23	0.10	-0.09	0.30
Nvehicles	-0.16	-0.25	0.00	-0.02	-0.21	-0.14	0.27	-0.26
DrivingLic	-0.01	-0.12	0.08	0.08	-0.10	-0.09	0.23	-0.12
CarParkFree_Work	-0.26	-0.29	-0.03	-0.08	-0.38	-0.25	0.30	-0.53
PTpass	0.25	0.23	0.01	0.09	0.39	0.21	-0.36	0.55
Area_km2	-0.15	-0.15	0.03	-0.08	-0.19	-0.09	0.23	-0.25
	Car_perc	N_INDIVIDUOS	Male_perc	IncomeHH	Nvehicles	DrivingLic		
Total	0.01	0.72	-0.38	0.25	-0.16	-0.01		
Walk	-0.53	0.56	-0.32	0.04	-0.25	-0.12		
Bike	0.00	0.29	-0.14	0.22	0.00	0.08		
Car	0.27	0.68	-0.33	0.26	-0.02	0.08		
PTransit	0.01	0.28	-0.20	0.23	-0.21	-0.10		
Other	0.09	0.20	-0.08	0.10	-0.14	-0.09		
Distance	0.26	-0.46	0.39	-0.09	0.27	0.23		
Duration	-0.40	-0.11	-0.01	0.30	-0.26	-0.12		
Car_perc	1.00	-0.09	0.18	-0.07	0.40	0.33		
N_INDIVIDUOS	-0.09	1.00	-0.46	0.14	-0.14	0.00		
Male_perc	0.18	-0.46	1.00	-0.34	0.19	-0.11		
IncomeHH	-0.07	0.14	-0.34	1.00	0.23	0.22		
Nvehicles	0.40	-0.14	0.19	0.23	1.00	0.44		
DrivingLic	0.33	0.00	-0.11	0.22	0.44	1.00		
CarParkFree_Work	0.53	-0.19	0.26	-0.01	0.54	0.39		
PTpass	-0.52	0.20	-0.34	0.16	-0.44	-0.45		
Area_km2	0.25	-0.19	0.30	-0.18	0.12	0.10		
	CarParkFree_Work	PTpass	Area_km2					
Total		-0.26	0.25	-0.15				
Walk		-0.29	0.23	-0.15				
Bike		-0.03	0.01	0.03				
Car		-0.08	0.09	-0.08				
PTransit		-0.38	0.39	-0.19				

## 6. Exploratory Data Analysis

Other	-0.25	0.21	-0.09
Distance	0.30	-0.36	0.23
Duration	-0.53	0.55	-0.25
Car_perc	0.53	-0.52	0.25
N_INDIVIDUOS	-0.19	0.20	-0.19
Male_perc	0.26	-0.34	0.30
IncomeHH	-0.01	0.16	-0.18
Nvehicles	0.54	-0.44	0.12
DrivingLic	0.39	-0.45	0.10
CarParkFree_Work	1.00	-0.66	0.32
PTpass	-0.66	1.00	-0.41
Area_km2	0.32	-0.41	1.00

```
# store the results so you can call the p-value at the corrplot
res = cor.mtest(data_continuous, conf.level = .95)

corrplot(
  corrrmat,
  method = "color", # or "circle"
  p.mat = res$p,
  sig.level = 0.05,
  type = "upper", # display only the upper triangular
  # order = "hclust", # order by hierarchical clustering
  tl.col = "black" # text label color
)
```



**i Note**

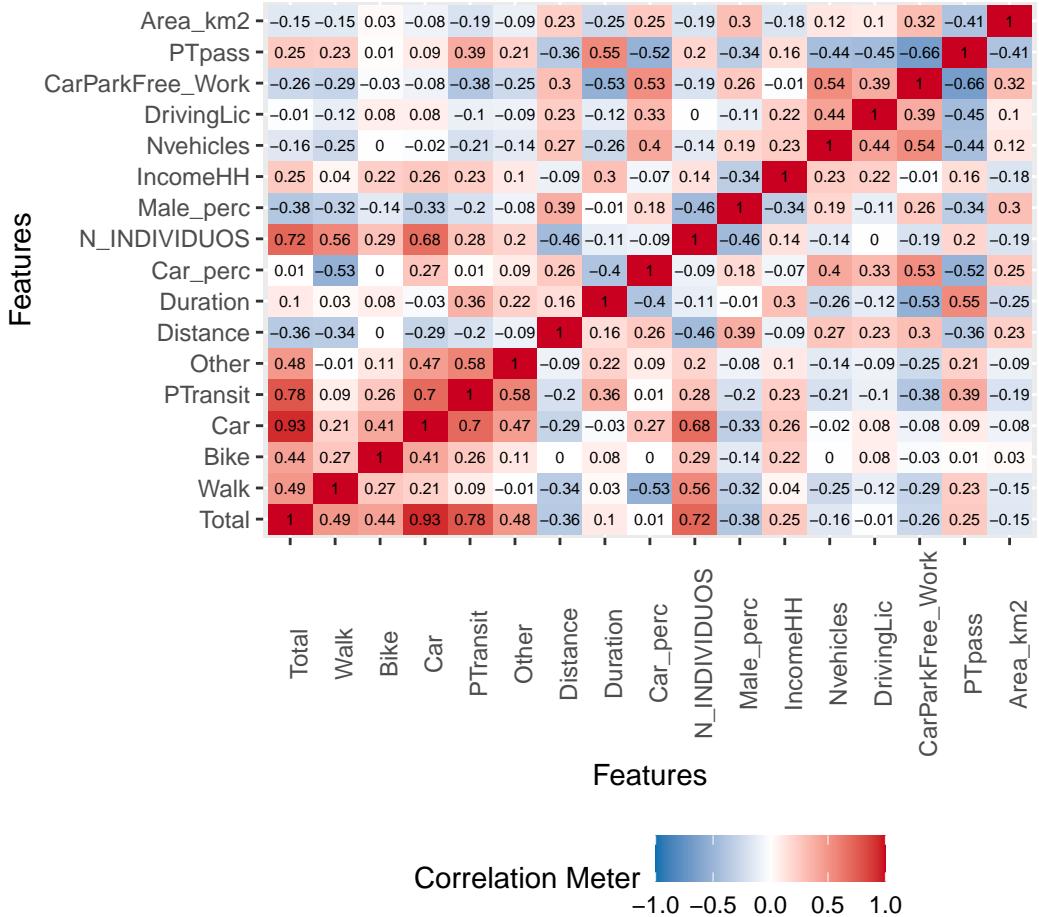
The pairwise correlations that are crossed are statistically not significant. The null hypothesis ( $H_0$ ) is that correlation is zero. The alternative hypothesis ( $H_A$ ) is that correlation is not zero.

This means that the correlations are only significant when you reject the null hypothesis ( $p\text{-value} < 0.05$ ).

Other method, using the `DataExplorer` package

```
plot_correlation(data_continuous)
```

## 6. Exploratory Data Analysis



Check the *p-value* of a crossed pair correlation:

```
cor.test(data$IncomeHH, data$Bike)
```

Pearson's product-moment correlation

```
data: data$IncomeHH and data$Bike
t = 3.5261, df = 234, p-value = 0.0005074
alternative hypothesis: true correlation is not equal to 0
95 percent confidence interval:
 0.09977779 0.34249284
sample estimates:
 cor
0.2246163
```

```
cor.test(data$IncomeHH, data$Duration)
```

```
Pearson's product-moment correlation

data: data$IncomeHH and data$Duration
t = 4.817, df = 234, p-value = 2.62e-06
alternative hypothesis: true correlation is not equal to 0
95 percent confidence interval:
0.1795420 0.4122446
sample estimates:
cor
0.300356
```

```
cor.test(data$Distance, data$Duration)
```

```
Pearson's product-moment correlation

data: data$Distance and data$Duration
t = 2.4605, df = 234, p-value = 0.0146
alternative hypothesis: true correlation is not equal to 0
95 percent confidence interval:
0.0317533 0.2808153
sample estimates:
cor
0.1588098
```

💡 The default for `cor.test` is Pearson, two-sided, with a 95% confident level. Check `?cor.test` for more options.



# 7. Multiple Linear Regression

## 💡 Do it yourself with R

Copy the script [MultipleLinarRegression.R](#) and paste it in your session.

Run each line using CTRL + ENTER

## ❗ Your task

Estimate a linear regression model that predicts the car percentage per district.

## 7.1. Load packages

```
library(tidyverse) # Pack of most used libraries for data science
library(skimr) # summary of the data
library(DataExplorer) # exploratory data analysis
library(corrplot) # correlation plots

library(car) # Testing autocorrelation (Durbin Watson)
library(olsrr) # Testing multicollinearity (VIF, TOL, etc.)
```

## 7.2. Dataset

The database used in this example is a treated database from the Mobility Survey for the metropolitan areas of Lisbon in 2018 (INE 2018).

Included variables:

- **Origin\_dicofre16** - Code of Freguesia (district) as set by INE after 2016 (Distrito + Concelho + Freguesia), for trip origin
- **Total** - number of trips with origin at each district
- **Walk** - number of walking trips
- **Bike** - number of bike trips
- **Car** - number of car trips. Includes taxi and motorcycle.
- **PTransit** - number of Public Transit trips
- **Other** - number of other trips (truck, van, tractor, aviation)
- **Distance** - average trip distance (km)

## 7. Multiple Linear Regression

- Duration - average trip duration (minutes)
- Car\_perc - percentage of car trips
- N\_INDIVIDUOS - number of residents (INE 2022)
- Male\_perc - percentage of male residents (INE 2022)
- IncomeHH - average household income
- Nvehicles - average number of car/motorcycle vehicles in the household
- DrivingLic - percentage of car driving licence holders
- CarParkFree\_Work - percentage of respondents with free car parking at the work location
- PTpass - percentage of public transit monthly pass holders
- internal - binary variable (factor). Yes: trip with same TAZ origin and destination, No: trips with different destination
- Lisboa - binary variable (factor). Yes: the district is part of Lisbon municipality, No: otherwise
- Area\_km2 - area of in Origin\_dicofre16, in km<sup>2</sup>

### 7.2.1. Import dataset

```
data = readRDS("../data/IMOBmodel.Rds")
data_continuous = data |> select(-Origin_dicofre16, -internal, -Lisboa) #
    ↵ Exclude categorical variables
```

Show summary statistics

```
skim(data)
```

Table 7.1.: Data summary

Name	data
Number of rows	236
Number of columns	20
<hr/>	
Column type frequency:	
character	1
factor	2
numeric	17
<hr/>	
Group variables	None

Variable type: character

skim_variable	n_missing	complete_rate	min	max	empty	n_unique	whitespace
Origin_dicofre16	0	1	6	6	0	118	0

Variable type: factor

skim_variable	n_missing	complete_rate	ordered	n_unique	top_counts
internal	0	1	FALSE	2	Yes: 118, No: 118
Lisboa	0	1	FALSE	2	No: 188, Yes: 48

Variable type: numeric

skim_variable	n_missing	complete_rate	mean	sd	p0	p25	p50	p75	p100	hist
Total	0	1	22457.00	19084.45	361.00	5917.75	17474.00	33377.50	112186.00	
Walk	0	1	5383.44	6224.84	0.00	763.25	3125.00	8298.50	32646.00	
Bike	0	1	107.03	248.65	0.00	0.00	13.50	98.75	2040.00	
Car	0	1	13289.24	12351.61	0.00	3243.00	9008.00	21248.75	52631.00	
PTransit	0	1	3473.79	5467.82	0.00	249.00	1057.00	4853.00	41672.00	
Other	0	1	203.45	336.04	0.00	2.00	44.00	281.50	2391.00	
Distance	0	1	11.14	2.66	6.84	9.54	10.32	12.10	22.66	
Duration	0	1	25.42	3.91	16.30	23.00	24.70	27.73	37.42	
Car_perc	0	1	59.00	21.56	0.00	45.40	62.62	75.72	99.27	
N_INDIVIDUOS	0	1	24323.80	16438.04	1566.00	11060.00	20855.00	36079.00	68649.00	
Male_perc	0	1	47.54	1.57	44.61	46.58	47.50	48.29	55.94	
IncomeHH	0	1	1732.55	453.11	884.46	1417.76	1594.73	1953.50	3462.32	
Nvehicles	0	1	1.53	0.24	1.02	1.35	1.54	1.67	2.42	
DrivingLic	0	1	62.50	8.12	37.04	57.67	63.00	68.84	80.79	
CarParkFree_Work	0	1	49.30	14.30	5.47	40.39	50.51	57.92	87.60	
PTpass	0	1	23.82	12.87	0.00	13.41	22.72	32.94	60.45	
Area_km2	0	1	25.55	42.58	1.49	5.04	11.60	28.51	282.13	

The dependent variable is continuous.

### 7.3. Check the assumptions

Before running the model, you need to check if the assumptions are met.

1. The dependent variable is normally distributed
2. Linear relationship between the dependent variable and the independent variables
3. No multicollinearity between independent variables (or only very little)
4. The observations are independent
5. Constant Variance (Assumption of Homoscedasticity)
6. Residuals are normally distributed

### 7.4. Assumption 1: Normal distribution

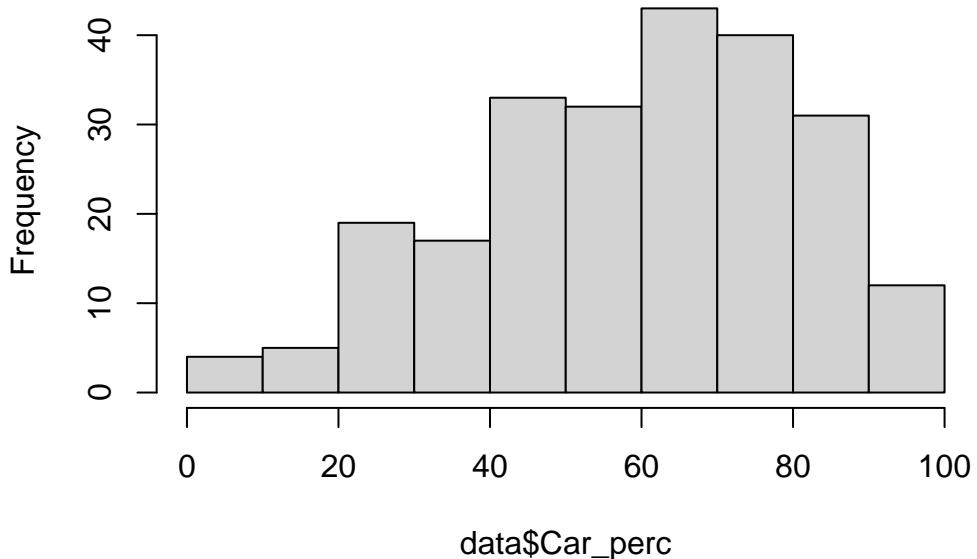
The Dependent Variable is be normally distributed.

Check the histogram of Car\_perc:

```
hist(data$Car_perc)
```

## 7. Multiple Linear Regression

Histogram of data\$Car\_perc



If the sample is small (< 50 observations), we use **Shapiro-Wilk** test:

```
shapiro.test(data$Car_perc)
```

```
Shapiro-Wilk normality test

data: data$Car_perc
W = 0.97284, p-value = 0.0001709
```

If not, use the **Kolmogorov-Smirnov** test:

```
ks.test(
  data$Car_perc,
  "pnorm",
  mean = mean(data$Car_perc),
  sd = sd(data$Car_perc)
)
```

```
Asymptotic one-sample Kolmogorov-Smirnov test

data: data$Car_perc
D = 0.072477, p-value = 0.1675
alternative hypothesis: two-sided
```

The null hypothesis for both tests is that the distribution is normal. Therefore, for the distribution to be normal, the pvalue must be  $> 0.05$  and the null hypothesis is not rejected. From the output obtained we can assume normality.

## 7.5. Assumption 2: Linear relationship

There is a linear relationship between dependent variable (DV) and independent variables (IV).

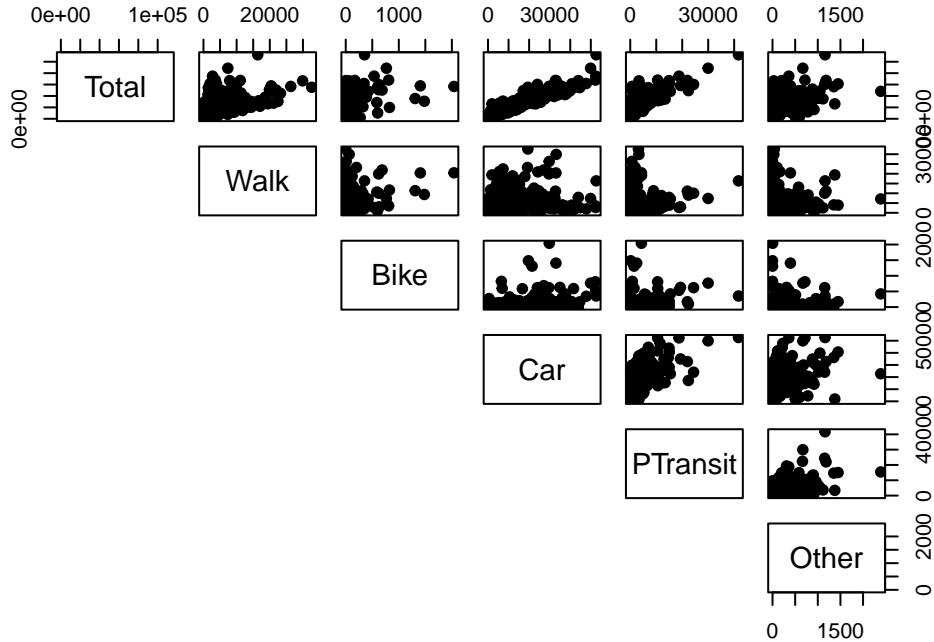
We can check this assumption by plotting scatterplots of the DV against each IV:

```
plot(x = data$Car_perc, y = data$Total, xlab = "Car_perc (%)", ylab =
  "Total (number of trips)")
plot(x = data$Car_perc, y = data$Walk, xlab = "Car_perc", ylab = "Walk")
plot(x = data$Car_perc, y = data$Bike, xlab = "Car_perc", ylab = "Bike")
plot(x = data$Car_perc, y = data$Car, xlab = "Car_perc", ylab = "Car")
plot(x = data$Car_perc, y = data$PTransit, xlab = "Car_perc", ylab =
  "PTransit")
plot(x = data$Car_perc, y = data$Other, xlab = "Car_perc", ylab = "Other")
plot(x = data$Car_perc, y = data$Distance, xlab = "Car_perc", ylab =
  "Distance")
plot(x = data$Car_perc, y = data$Duration, xlab = "Car_perc", ylab =
  "Duration")
plot(x = data$Car_perc, y = data$N_INDIVIDUOS, xlab = "Car_perc", ylab =
  "N_INDIVIDUOS")
plot(x = data$Car_perc, y = data$Male_perc, xlab = "Car_perc", ylab =
  "Male_perc")
plot(x = data$Car_perc, y = data$IncomeHH, xlab = "Car_perc", ylab =
  "IncomeHH")
plot(x = data$Car_perc, y = data$Nvehicles, xlab = "Car_perc", ylab =
  "Nvehicles")
plot(x = data$Car_perc, y = data$DrivingLic, xlab = "Car_perc", ylab =
  "Driving License")
plot(x = data$Car_perc, y = data$CarParkFree_Work, xlab = "Car_perc", ylab =
  "Free car parking at work")
plot(x = data$Car_perc, y = data$PTpass, xlab = "Car_perc", ylab =
  "PTpass")
plot(x = data$Car_perc, y = data$internal, xlab = "Car_perc", ylab =
  "internal trips")
plot(x = data$Car_perc, y = data$Lisboa, xlab = "Car_perc", ylab =
  "Lisboa")
plot(x = data$Car_perc, y = data$Area_km2, xlab = "Car_perc", ylab =
  "Area_km2")
```

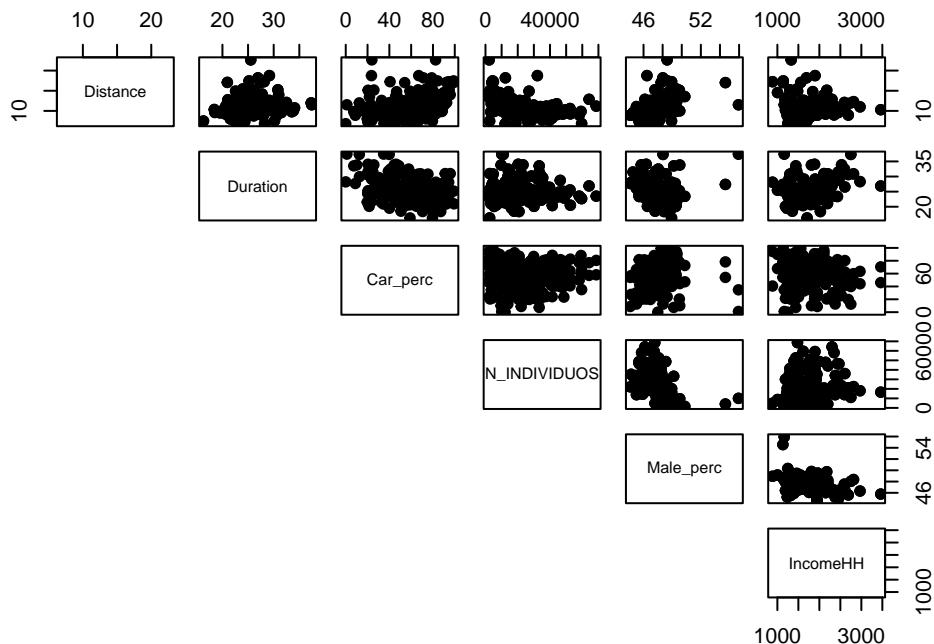
Or you can make a pairwise scatterplot matrix, that compares every variable with each other:

## 7. Multiple Linear Regression

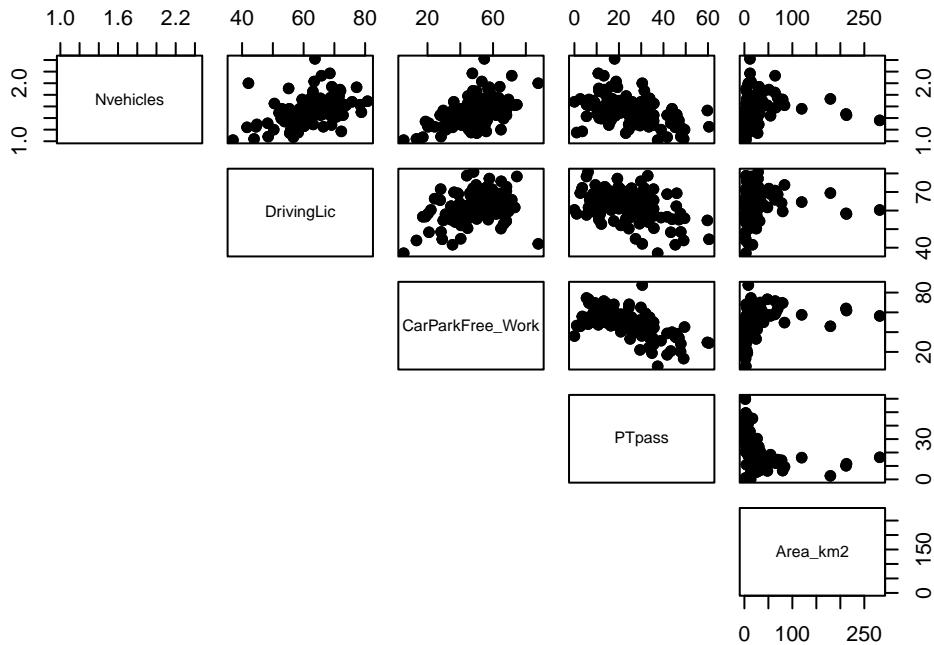
```
# pairs(data_continuous, pch = 19, lower.panel = NULL) # we have too many
# variables, let's split the plots
pairs(data_continuous[,1:6], pch = 19, lower.panel = NULL)
```



```
pairs(data_continuous[,7:12], pch = 19, lower.panel = NULL)
```



```
pairs(data_continuous[,13:17], pch = 19, lower.panel = NULL)
```



## 7.6. Assumption 3: No multicollinearity

Check the [correlation plot](#) before choosing the variables.

### 7.6.1. Declare the model

💡 Use **CTRL + SHIFT + C** to comment/uncomment lines (variables)

```
# names(data) # to see the names of the variables

model = lm(
  Car_perc ~
  Total +
  Walk +
  Bike +
  Car +
  PTransit +
  Other +
  Distance +
  Duration +
  N_INDIVIDUOS +
  Male_perc +
```

## 7. Multiple Linear Regression

```

IncomeHH +
Nvehicles +
DrivingLic +
CarParkFree_Work +
PTpass +
internal +
Lisboa +
Area_km2,
data = data
)

summary(model)

```

Call:

```
lm(formula = Car_perc ~ Total + Walk + Bike + Car + PTransit +
Other + Distance + Duration + N_INDIVIDUOS + Male_perc +
IncomeHH + Nvehicles + DrivingLic + CarParkFree_Work + PTpass +
internal + Lisboa + Area_km2, data = data)
```

Residuals:

Min	1Q	Median	3Q	Max
-37.952	-4.907	-0.232	5.305	39.157

Coefficients:

	Estimate	Std. Error	t value	Pr(> t )
(Intercept)	4.047e+00	3.192e+01	0.127	0.89923
Total	1.633e-01	3.671e-01	0.445	0.65700
Walk	-1.640e-01	3.671e-01	-0.447	0.65552
Bike	-1.614e-01	3.669e-01	-0.440	0.66044
Car	-1.627e-01	3.671e-01	-0.443	0.65806
PTransit	-1.640e-01	3.672e-01	-0.447	0.65552
Other	-1.622e-01	3.671e-01	-0.442	0.65897
Distance	6.004e-01	3.629e-01	1.654	0.09948 .
Duration	-3.244e-01	3.325e-01	-0.976	0.33032
N_INDIVIDUOS	-6.627e-05	8.876e-05	-0.747	0.45612
Male_perc	2.723e-01	6.151e-01	0.443	0.65849
IncomeHH	-5.218e-04	2.199e-03	-0.237	0.81264
Nvehicles	7.301e+00	3.959e+00	1.844	0.06650 .
DrivingLic	2.910e-01	1.161e-01	2.506	0.01294 *
CarParkFree_Work	1.943e-01	7.882e-02	2.465	0.01449 *
PTpass	-8.254e-02	9.989e-02	-0.826	0.40956
internalNo	1.962e+01	2.136e+00	9.182	< 2e-16 ***
LisboaYes	-9.647e+00	3.182e+00	-3.031	0.00273 **
Area_km2	1.689e-02	1.860e-02	0.908	0.36486
---				

```

Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 10.59 on 217 degrees of freedom
Multiple R-squared:  0.7772,    Adjusted R-squared:  0.7587
F-statistic: 42.06 on 18 and 217 DF,  p-value: < 2.2e-16

```

### 7.6.2. Assessing the model

1. First check the **pvalue** and the **F statistics** of the model to see if there is any statistical relation between the dependent variable and the independent variables. If  $pvalue < 0.05$  and the F statistics  $> F_{critical} = 2.39$ , then the model is statistically acceptable.
2. The **R-square** and **Adjusted R-square** evaluate the amount of variance that is explained by the model. The difference between one and another is that the R-square does not consider the number of variables. If you increase the number of variables in the model, the R-square will tend to increase which can lead to overfitting. On the other hand, the Adjusted R-square adjust to the number of independent variables.
3. Take a look at the **t-value** and the  $Pr(>|t|)$ . If the t-value  $> 1.96$  or  $Pr(>|t|) < 0.05$ , then the IV is statistically significant to the model.
4. To analyse the **estimates** of the variables, you should first check the **signal** and assess if the independent variable has a direct or inverse relationship with the dependent variable. It is only possible to evaluate the **magnitude** of the estimate if all variables are continuous and standardized or by calculating the elasticities. Do not forget to access the **Intercept...**

We can see from the output that the R-squared value for the model (with ALL variables) is **0.7772**. We can also see that the overall F-statistic is **42.06** and the corresponding p-value is **<2.2e-16**, which indicates that the overall regression model is significant. Also, the predictor variables **DrivingLic** and **CarParkFree\_Work**, **internal(No)** and **Lisboa(Yes)** are statistically significant at the 0.05 significance level.

 Your turn

Now try to remove some variables from the model and assess it again. Elaborate a justification to exclude those variables.

### 7.6.3. Calculate the Variance Inflation Factor (VIF)

We use the **vif()** function from the car package to calculate the VIF for each predictor variable in the model:

```
car::vif(model)
```

## 7. Multiple Linear Regression

Total	Walk	Bike	Car
1.028324e+08	1.094004e+07	1.743544e+04	4.307486e+07
PTransit	Other	Distance	Duration
8.443136e+06	3.186972e+04	1.957062e+00	3.544018e+00
N_INDIVIDUOS	Male_perc	IncomeHH	Nvehicles
4.459671e+00	1.961472e+00	2.079348e+00	1.882275e+00
DrivingLic	CarParkFree_Work	PTpass	internal
1.863227e+00	2.660375e+00	3.464699e+00	2.400178e+00
Lisboa	Area_km2		
3.451852e+00	1.313708e+00		

A common rule of thumb is that a VIF value greater than 5 indicates a high level of multicollinearity among the predictor variables, which is potentially concerning.

### 7.6.4. Condition index

Condition index is another diagnostic tool for detecting multicollinearity, with values above 10 indicating moderate multicollinearity and values above 30 indicating severe multicollinearity. You will learn how to do that in the next class: [Factor analysis](#).

## 7.7. Assumption 4: Independence of observations

Multiple linear regression assumes that each observation in the dataset is independent.

The error (E) is independent across observations and the error variance is constant across IV

The simplest way to determine if this assumption is met is to perform a Durbin-Watson test, which is a formal statistical test that tells us whether or not the **residuals** (and thus the **observations**) exhibit autocorrelation.

```
durbinWatsonTest(model)
```

```
lag Autocorrelation D-W Statistic p-value
 1      0.07881007    1.840798   0.116
Alternative hypothesis: rho != 0
```

#### i Note

In the Durbin-Watson test, values of the D-W Statistic vary from 0 to 4. If the values are from 1.8 to 2.2 this means that there is **no autocorrelation** in the model.

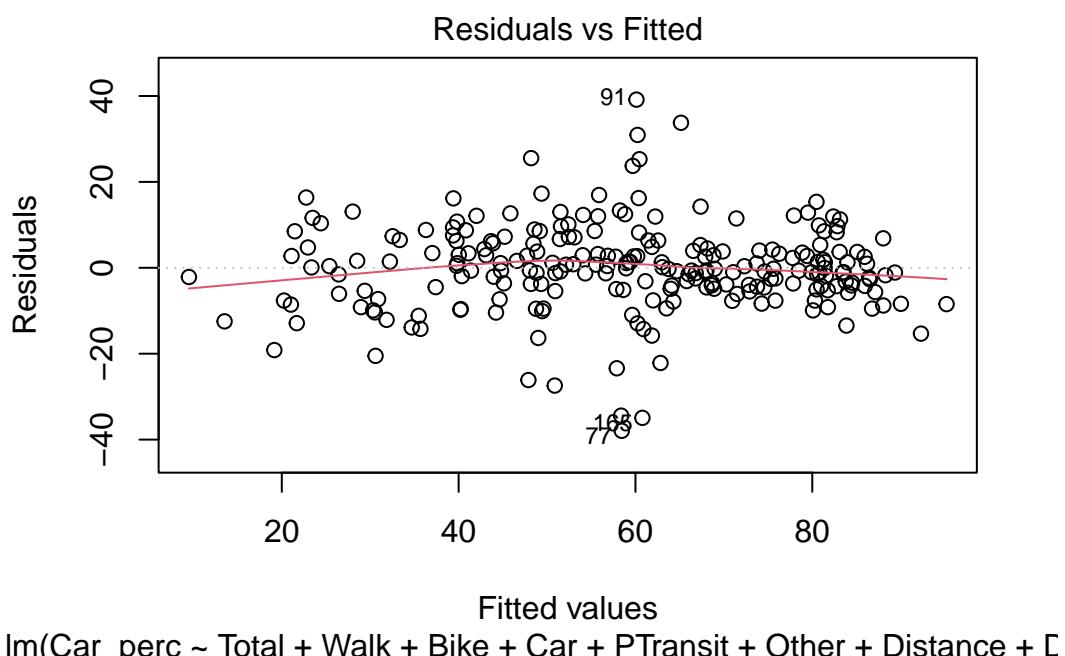
**H<sub>0</sub> (null hypothesis):** There is no correlation among the residuals. Since p-value > 0.05, we do not reject the null hypothesis and we can not discard that there is autocorrelation in the model.

If the observations are not independent, you may consider to treat the observations as a panel data.

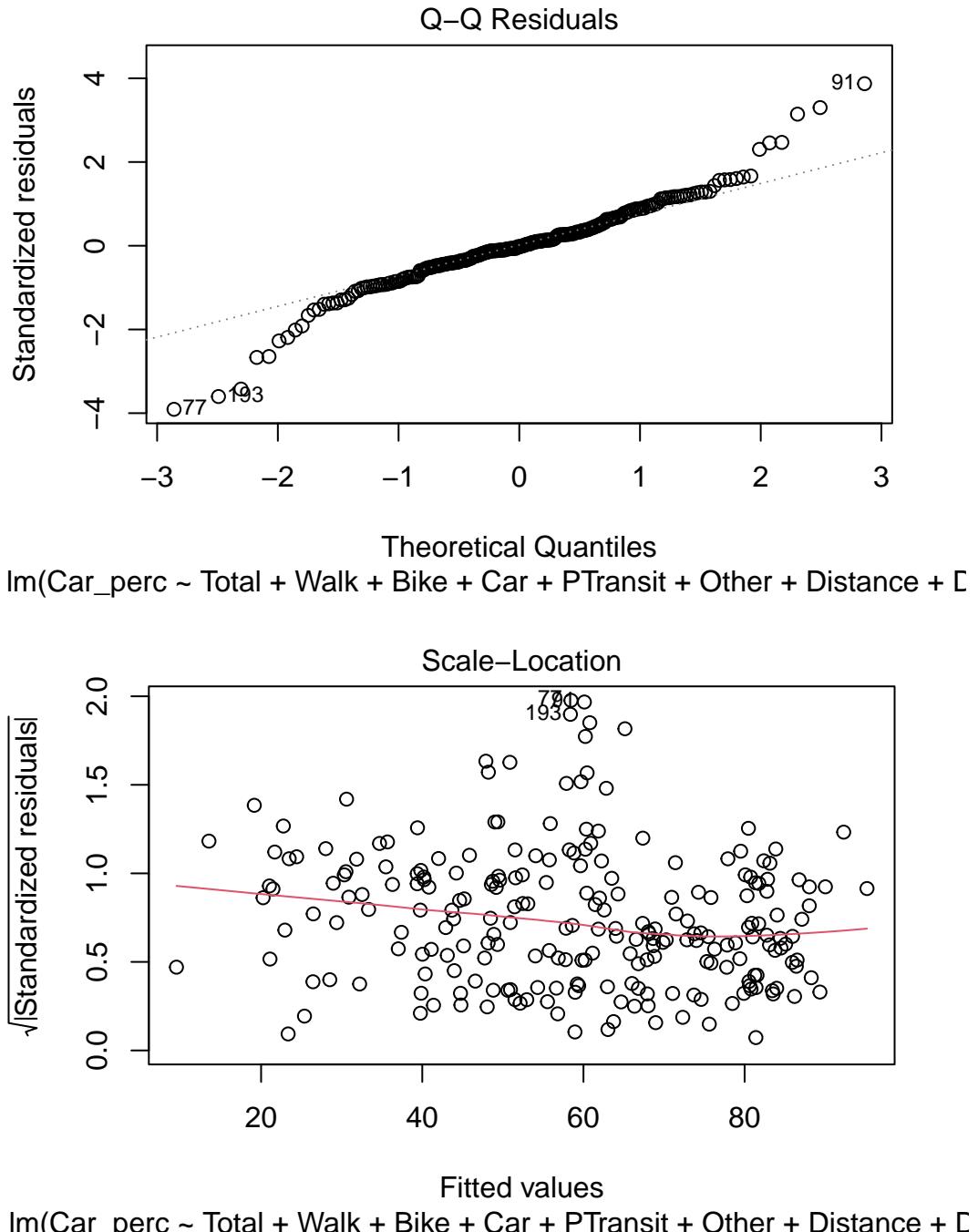
## 7.8. Assumption 5: Constant Variance (Homoscedasticity)

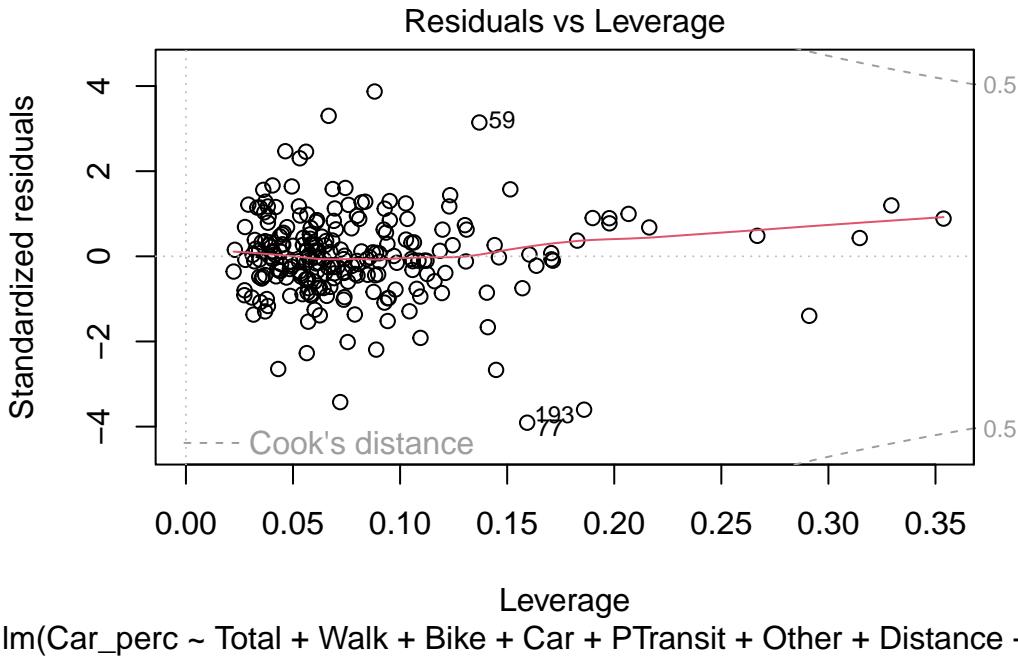
The simplest way to determine if this assumption is met is to create a plot of standardized residuals versus predicted values.

```
plot(model)
```



## 7. Multiple Linear Regression





For the Residuals, check the following **assumptions**:

- **Residuals vs Fitted:** This plot is used to detect non-linearity, heteroscedasticity, and outliers.
- **Normal Q-Q:** The quantile-quantile (Q-Q) plot is used to check if the disturbances follow a normal distribution
- **Scale-Location:** This plot is used to verify if the residuals are spread equally (homoscedasticity) or not (heteroscedasticity) through the sample.
- **Residuals vs Leverage:** This plot is used to detect the impact of the outliers in the model. If the outliers are outside the Cook-distance, this may lead to serious problems in the model.

Try analyzing the plots and check if the model meets the assumptions.

## 7.9. Assumption 6: Residuals are normally distributed

Assess the Q-Q Residuals plot above. When the residuals clearly depart from a straight diagonal line, it indicates that they do not follow a normal distribution.

Use a formal statistical test like Shapiro-Wilk or Kolmogorov-Smirnov to validate those results.



# 8. Factor Analysis

## 💡 Do it yourself with R

Copy the script [FactorAnalysis.R](#) and paste it in your session.

Run each line using CTRL + ENTER

This exercise is based on the paper “**Residential location satisfaction in the Lisbon metropolitan area**”, by Martinez, Abreu e Silva, and Viegas (2010) .

The aim of this study was to examine the perception of households towards their residential location considering land use and accessibility factors, as well as household socio-economic and attitudinal characteristics.

## ❗ Your task

Create meaningful latent factors.

## 8.1. Load packages

```
library(tidyverse) # Pack of most used libraries for data science
library(summarytools) # Summary of the dataset
library(foreign) # Read SPSS files
library(nFactors) # Factor analysis
library(GPArotation) # GPA Rotation for Factor Analysis
library(psych) # Personality, psychometric, and psychological research
```

## 8.2. Dataset

Included variables:

- RespondentID - ID of the respondent
- DWELCLAS - Classification of the dwelling
- INCOME - Income of the household
- CHILD13 - Number of children under 13 years old
- H18 - Number of household members over 18 years old
- HEMPLOY - Number of household members employed

## 8. Factor Analysis

- HSIZE - Household size
- IAGE - Age of the respondent
- ISEX - Sex of the respondent
- NCARS - Number of cars in the household
- AREA - Area of the dwelling
- BEDROOM - Number of bedrooms in the dwelling
- PARK - Number of parking spaces in the dwelling
- BEDSIZE - BEDROOM/HSIZE
- PARKSIZE - PARK/NCARS
- RAGE10 - 1 if Dwelling age <= 10
- TCBD - Private car distance in time to CBD
- DISTTC - Euclidean distance to heavy public transport system stops
- TWCBD - Private car distance in time of workplace to CBD
- TDWWK - Private car distance in time of dwelling to work place
- HEADH - 1 if Head of the Household
- POPDENS - Population density per hectare
- EDUINDEX - Number of undergraduate students/Population over 20 years old (500m)

### 8.2.1. Import dataset

```
data = read.spss("../data/example_fact.sav", to.data.frame = T)
```

### 8.2.2. Get to know your dataset

Take a look at the first values of the dataset

```
head(data, 5)
```

	RespondentID	DWELCLAS	INCOME	CHILD13	H18	HEMPLOY	HSIZE	AVADUAGAE	IAGE	ISEX
1	799161661	5	7500	1	2	2	3	32.00000	32	1
2	798399409	6	4750	0	1	1	1	31.00000	31	1
3	798374392	6	4750	2	2	2	4	41.50000	42	0
4	798275277	5	7500	0	3	2	4	44.66667	52	1
5	798264250	6	2750	1	1	1	2	33.00000	33	0
	NCARS	AREA	BEDROOM	PARK	BEDSIZE	PARKSIZE	RAGE10	TCBD	DISTHTC	
1	2	100	2	1	0.6666667	0.5	1	36.791237	629.1120	
2	1	90	2	1	2.0000000	1.0	0	15.472989	550.5769	
3	2	220	4	2	1.0000000	1.0	1	24.098125	547.8633	
4	3	120	3	0	0.7500000	0.0	0	28.724796	2350.5782	
5	1	90	2	0	1.0000000	0.0	0	7.283384	698.3000	
	TWCBD	TDWWK	HEADH	POPDENS	EDUINDEX	GRAVCPC	GRAVCPT	GRAVPCPT		
1	10.003945	31.14282	1	85.70155	0.06406279	0.2492962	0.2492607	1.0001423		
2	15.502989	0.00000	1	146.43494	0.26723192	0.3293831	0.3102800	1.0615674		

```

3 12.709374 20.38427      1 106.60810 0.09996816 0.2396229 0.2899865 0.8263245
4 3.168599 32.94246      1 36.78380 0.08671065 0.2734539 0.2487830 1.0991661
5 5.364160 13.04013      1 181.62720 0.13091674 0.2854017 0.2913676 0.9795244
  NSTRTC      DISTHW      DIVIDX      ACTDENS      DISTCBD
1     38 2036.4661 0.3225354 0.6722406 9776.142
2     34 747.7683 0.3484588 2.4860345 3523.994
3     33 2279.0577 0.3237884 1.6249059 11036.407
4      6 1196.4665 0.3272149 1.7664923 6257.262
5     31 3507.2402 0.3545181 11.3249309 1265.239

```

```
View(data) # open in table
```

Make the `RespondentID` variable as **row names** or case number

```
data = data |> column_to_rownames(var = "RespondentID")
```

Take a look at the main characteristics of the dataset

```
str(data)
```

```
'data.frame': 470 obs. of 31 variables:
 $ DWELCLAS: num  5 6 6 5 6 6 4 2 6 5 ...
 $ INCOME   : num  7500 4750 4750 7500 2750 1500 12500 1500 1500 1500 ...
 $ CHILD13  : num  1 0 2 0 1 0 0 0 0 0 ...
 $ H18       : num  2 1 2 3 1 3 3 4 1 1 ...
 $ HEMPLOY   : num  2 1 2 2 1 2 0 2 1 1 ...
 $ HSIZZE   : num  3 1 4 4 2 3 3 4 1 1 ...
 $ AVADUAGE : num  32 31 41.5 44.7 33 ...
 $ IAGE      : num  32 31 42 52 33 47 62 21 34 25 ...
 $ ISEX      : num  1 1 0 1 0 1 1 0 0 0 ...
 $ NCARS     : num  2 1 2 3 1 1 2 3 1 1 ...
 $ AREA      : num  100 90 220 120 90 100 178 180 80 50 ...
 $ BEDROOM   : num  2 2 4 3 2 2 5 3 2 1 ...
 $ PARK      : num  1 1 2 0 0 0 2 0 0 1 ...
 $ BEDSIZE   : num  0.667 2 1 0.75 1 ...
 $ PARKSIZE  : num  0.5 1 1 0 0 0 1 0 0 1 ...
 $ RAGE10    : num  1 0 1 0 0 0 0 0 1 1 ...
 $ TCBD      : num  36.79 15.47 24.1 28.72 7.28 ...
 $ DISTHTC   : num  629 551 548 2351 698 ...
 $ TWCBD     : num  10 15.5 12.71 3.17 5.36 ...
 $ TDWWK     : num  31.1 0 20.4 32.9 13 ...
 $ HEADH     : num  1 1 1 1 1 1 0 1 1 ...
 $ POPDENS   : num  85.7 146.4 106.6 36.8 181.6 ...
 $ EDUINDEX  : num  0.0641 0.2672 0.1 0.0867 0.1309 ...
 $ GRAVCPC   : num  0.249 0.329 0.24 0.273 0.285 ...
```

## 8. Factor Analysis

```
$ GRAVCPT : num  0.249 0.31 0.29 0.249 0.291 ...
$ GRAVPCPT: num  1 1.062 0.826 1.099 0.98 ...
$ NSTRTC   : num  38 34 33 6 31 45 12 6 4 22 ...
$ DISTHW   : num  2036 748 2279 1196 3507 ...
$ DIVIDX   : num  0.323 0.348 0.324 0.327 0.355 ...
$ ACTDENS  : num  0.672 2.486 1.625 1.766 11.325 ...
$ DISTCBD  : num  9776 3524 11036 6257 1265 ...
- attr(*, "variable.labels")= Named chr(0)
..- attr(*, "names")= chr(0)
- attr(*, "codepage")= int 1252
```

Check summary statistics of variables

```
skimr::skim(data)
```

Table 8.1.: Data summary

Name	data
Number of rows	470
Number of columns	31
<hr/>	
Column type frequency:	
numeric	31
<hr/>	
Group variables	None

**Variable type: numeric**

skim_variable	n_missing	complete_rate	mean	sd	p0	p25	p50	p75	p100	hist
DWELCLAS	0	1	5.15	1.28	1.00	4.00	5.00	6.00	7.00	
INCOME	0	1	4259.57	3001.82	700.00	2750.00	2750.00	4750.00	12500.00	
CHILD13	0	1	0.40	0.78	0.00	0.00	0.00	0.00	4.00	
H18	0	1	2.11	0.93	0.00	2.00	2.00	2.75	6.00	
HEMPLOY	0	1	1.53	0.74	0.00	1.00	2.00	2.00	5.00	
HSIZE	0	1	2.58	1.25	1.00	2.00	2.00	4.00	7.00	
AVADUAGE	0	1	37.79	9.94	0.00	30.33	36.00	43.00	78.00	
IAGE	0	1	36.89	11.59	0.00	28.00	34.00	43.00	78.00	
ISEX	0	1	0.54	0.50	0.00	0.00	1.00	1.00	1.00	
NCARS	0	1	1.71	0.89	0.00	1.00	2.00	2.00	5.00	
AREA	0	1	132.96	121.54	30.00	90.00	110.00	150.00	2250.00	
BEDROOM	0	1	2.88	1.07	0.00	2.00	3.00	3.00	7.00	
PARK	0	1	0.83	0.97	0.00	0.00	1.00	1.00	4.00	
BEDSIZE	0	1	1.35	0.80	0.00	0.80	1.00	1.50	5.00	
PARKSIZE	0	1	0.47	0.57	0.00	0.00	0.25	1.00	3.00	
RAGE10	0	1	0.24	0.43	0.00	0.00	0.00	0.00	1.00	
TCBD	0	1	24.70	16.24	0.78	9.95	23.81	35.70	73.28	
DISTHTC	0	1	1346.98	1815.78	49.05	400.47	719.04	1525.45	17732.68	
TWCBD	0	1	17.00	16.24	0.31	5.25	9.86	25.27	67.82	
TDWWK	0	1	23.53	17.09	0.00	10.14	22.18	33.72	80.72	
HEADH	0	1	0.86	0.34	0.00	1.00	1.00	1.00	1.00	
POPDENS	0	1	92.00	58.20	0.00	45.86	83.17	135.01	255.55	
EDUINDEX	0	1	0.20	0.12	0.00	0.10	0.17	0.28	0.66	
GRAVCP	0	1	0.29	0.07	0.07	0.25	0.31	0.34	0.39	
GRAVCPT	0	1	0.26	0.06	0.03	0.22	0.28	0.31	0.37	
GRAVPCPT	0	1	1.15	0.32	0.45	1.00	1.09	1.22	2.88	
NSTRTC	0	1	22.22	12.62	0.00	14.00	22.00	30.00	84.00	
DISTHW	0	1	1883.41	1748.30	74.67	712.79	1338.74	2533.30	16590.06	
DIVIDX	0	1	0.38	0.08	0.26	0.32	0.35	0.40	0.65	
ACTDENS	0	1	5.78	8.51	0.02	1.32	2.49	4.85	63.18	
DISTCBD	0	1	7967.41	7442.86	148.93	2025.26	5542.34	11803.20	44004.60	

```
# print(dfSummary(data), method = "render")
```

**i** We used a different library for the summary statistics.  
**R** allows you to do the same or similar tasks with different packages.

### 8.3. Assumptions for factorial analysis

There are some rules of thumb and assumptions that should be checked before performing a factor analysis.

#### Rules of thumb:

- At least 10 variables
- $n < 50$  (Unacceptable);  $n > 200$  (recommended)
- It is recommended to use continuous variables. If your data contains categorical variables, you should transform them to dummy variables.

As we have 31 variables (none categorical) and more than 200 observations, we can proceed to check the assumptions.

#### Assumptions:

- Normality;
- Linearity;
- Homogeneity;
- Homoscedasticity (some multicollinearity is desirable);
- Correlations between variables  $< 0.3$  are not appropriate to use Factor Analysis

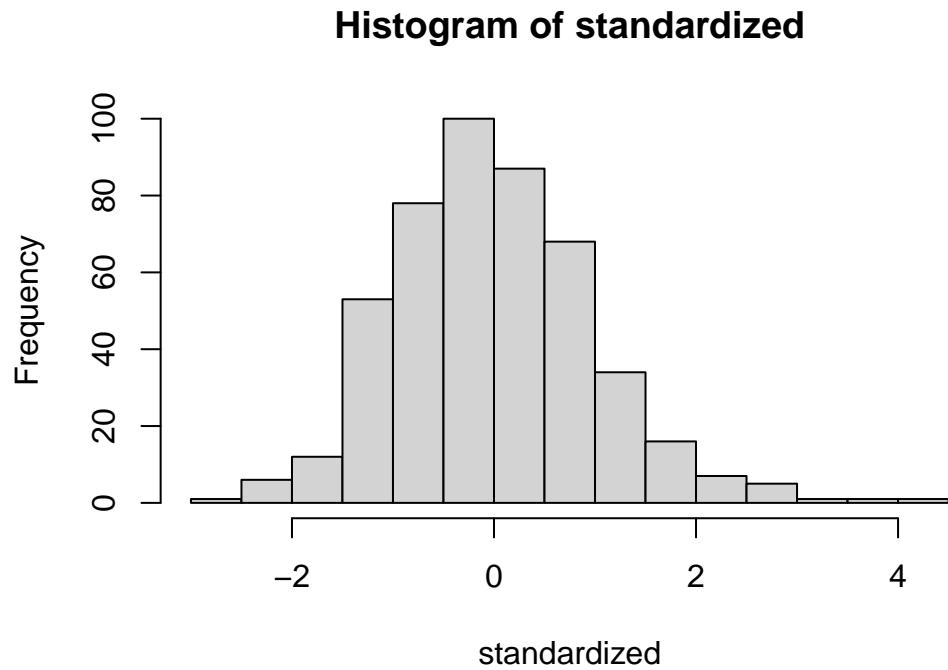
Let's run a random regression model in order to evaluate some of the assumptions.

```
# random regression model
random = rchisq(nrow(data), ncol(data))
fake = lm(random ~ ., data = data)
standardized = rstudent(fake)
fitted = scale(fake$fitted.values)
```

#### 8.3.1. Normality

```
hist(standardized)
```

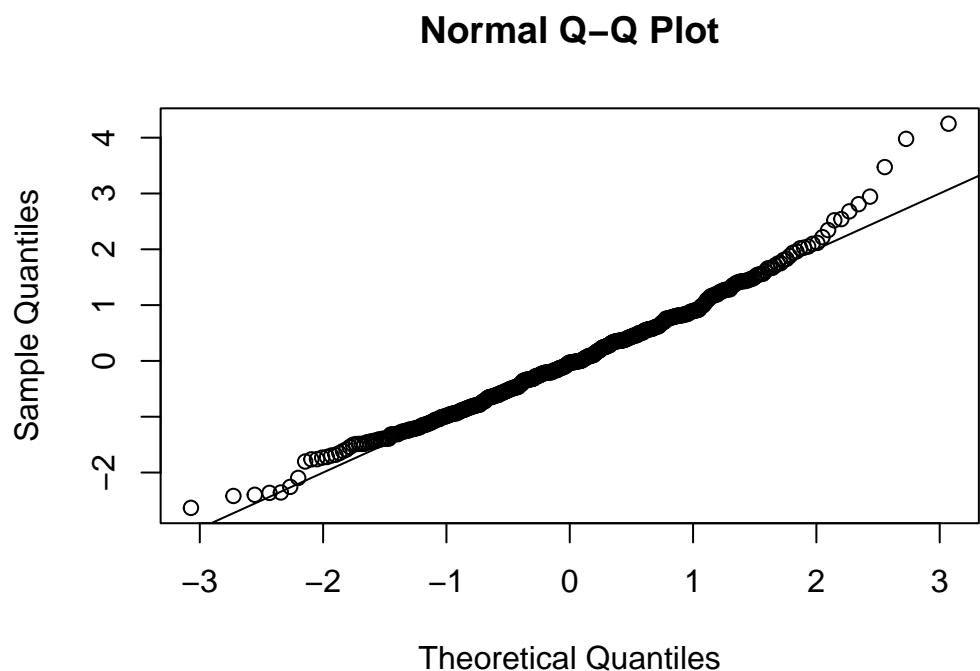
## 8. Factor Analysis



The histogram looks like a normal distribution.

### 8.3.2. Linearity

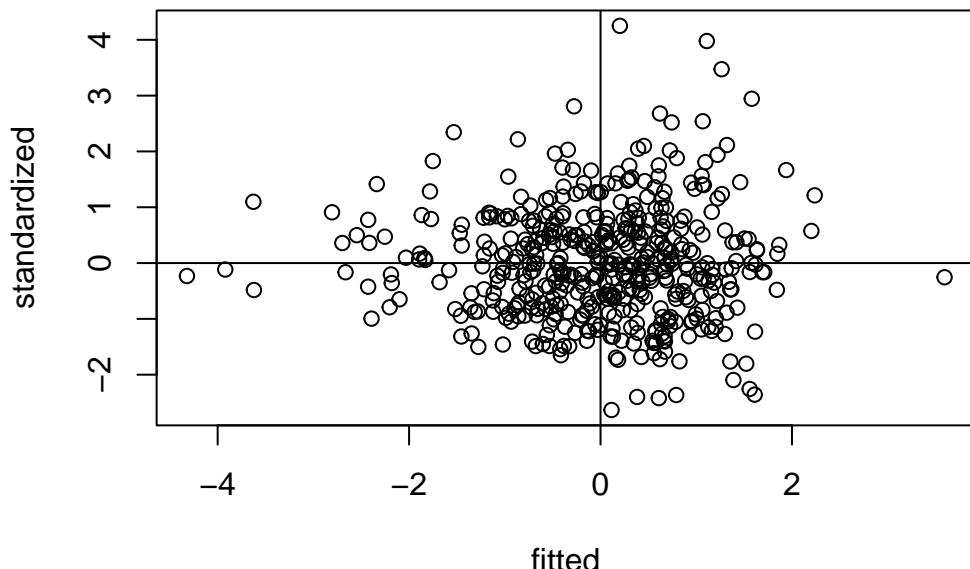
```
qqnorm(standardized)  
abline(0, 1)
```



The QQ plot shows that the points are close to the diagonal line, indicating linearity.

### 8.3.3. Homogeneity

```
plot(fitted, standardized)
abline(h=0, v=0)
```



The residuals are randomly distributed around zero, indicating homogeneity.

### 8.3.4. Correlations between variables

Correlation matrix

```
corr_matrix = cor(data, method = "pearson")
```

The **Bartlett's test** examines if there is equal variance (homogeneity) between variables. Thus, it evaluates if there is any pattern between variables.

#### 8.3.4.1. Correlation adequacy

Check for correlation adequacy - Bartlett's Test

```
cortest.bartlett(corr_matrix, n = nrow(data))
```

## 8. Factor Analysis

```
$chisq  
[1] 9880.074  
  
$p.value  
[1] 0  
  
$df  
[1] 465
```

The null hypothesis is that there is no correlation between variables. Therefore, you want to **reject** the null hypothesis.

**Note:** A p-value  $< 0.05$  indicates that there are correlations between variables, and that **factor analysis** may be useful with your data.

### 8.3.4.2. Sampling adequacy

Check for sampling adequacy - KMO test

```
KMO(corr_matrix)
```

```
Kaiser-Meyer-Olkin factor adequacy  
Call: KMO(r = corr_matrix)  
Overall MSA =  0.68  
MSA for each item =  
DWELCLAS INCOME CHILD13 H18 HEMPLOY HSIZE AVADUAGE IAGE  
0.70 0.85 0.33 0.58 0.88 0.59 0.38 0.40  
ISEX NCARS AREA BEDROOM PARK BEDSIZE PARKSIZE RAGE10  
0.71 0.74 0.60 0.53 0.62 0.58 0.57 0.84  
TCBD DISTHTC TWCBD TDWWK HEADH POPDENS EDUINDEX GRAVCPC  
0.88 0.88 0.82 0.89 0.47 0.82 0.85 0.76  
GRAVCPT GRAVPCPT NSTRTC DISTHW DIVIDX ACTDENS DISTCBD  
0.71 0.31 0.83 0.76 0.63 0.70 0.86
```

We want at least **0.7** of the overall Mean Sample Adequacy (MSA). If,  $0.6 < \text{MSA} < 0.7$ , it is not a good value, but acceptable in some cases.

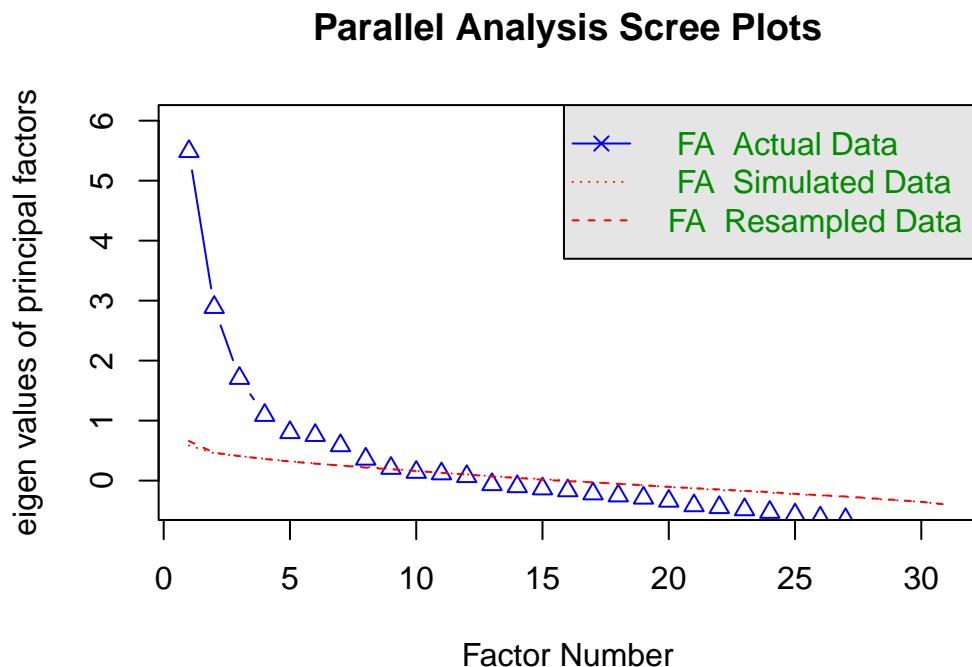
## 8.4. Determine the number of factors to extract

There are several ways to determine the number of factors to extract. Here we will use three different methods:

- Parallel Analysis
- Kaiser Criterion
- Principal Component Analysis (PCA)

### 8.4.1. Parallel Analysis

```
num_factors = fa.parallel(
  x = data,
  fm = "ml", # factor method = maximum likelihood
  fa = "fa") # factor analysis
```



Parallel analysis suggests that the number of factors = 8 and the number of components = 4.

The selection of the number of factors in the Parallel analysis can be threefold:

- Detect where there is an “elbow” in the graph;
- Detect the intersection between the “FA Actual Data” and the “FA Simulated Data”;
- Consider the number of factors with eigenvalue > 1.

### 8.4.2. Kaiser Criterion

```
sum(num_factors$fa.values > 1) # Number of factors with eigenvalue > 1
```

[1] 4

## 8. Factor Analysis

```
sum(num_factors$fa.values > 0.7) # Number of factors with eigenvalue > 0.7
```

```
[1] 6
```

You can also consider factors with eigenvalue  $> 0.7$ , since some of the literature indicate that this value does not overestimate the number of factors as much as considering an eigenvalue = 1.

### 8.4.3. Principal Component Analysis (PCA)

#### 🔥 PCA is not the same thing as Factor Analysis

PCA only considers the common information (variance) of the variables, while factor analysis takes into account also the unique variance of the variable. Both approaches are often mixed up. In this example we use PCA as only a first criteria for choosing the number of factors. PCA is usually used in image recognition and data reduction of big data.

#### 8.4.3.1. Variance explained by components

Print variance that explains the components

```
data_pca = princomp(data,
                     cor = TRUE) # standardizes your dataset before running
                     # a PCA
summary(data_pca)
```

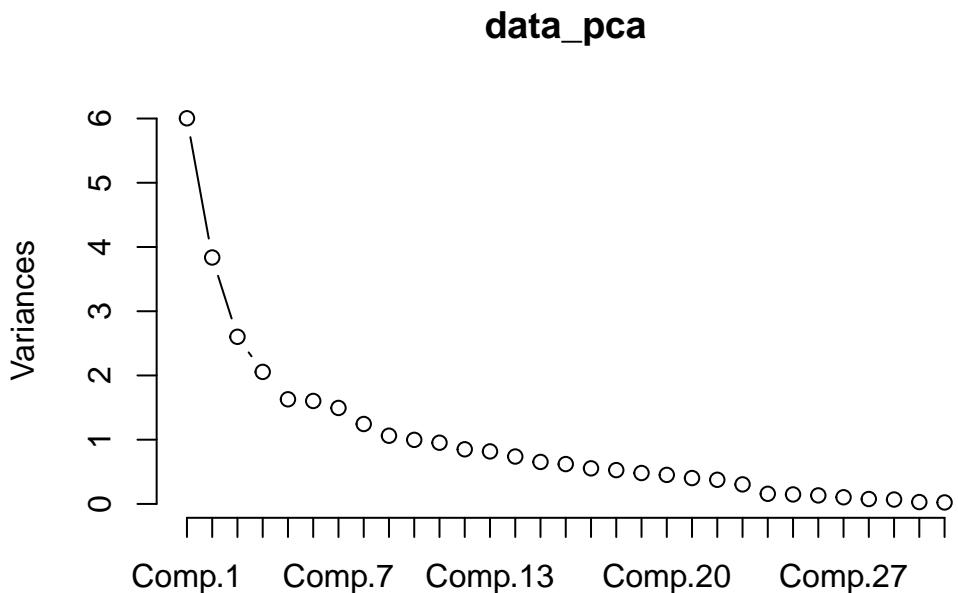
Importance of components:

	Comp.1	Comp.2	Comp.3	Comp.4	Comp.5
Standard deviation	2.450253	1.9587909	1.61305418	1.43367870	1.27628545
Proportion of Variance	0.193669	0.1237697	0.08393367	0.06630434	0.05254531
Cumulative Proportion	0.193669	0.3174388	0.40137245	0.46767679	0.52022210
	Comp.6	Comp.7	Comp.8	Comp.9	Comp.10
Standard deviation	1.26612033	1.22242045	1.11550534	1.0304937	0.99888665
Proportion of Variance	0.05171164	0.04820361	0.04014039	0.0342554	0.03218628
Cumulative Proportion	0.57193374	0.62013734	0.66027774	0.6945331	0.72671941
	Comp.11	Comp.12	Comp.13	Comp.14	Comp.15
Standard deviation	0.97639701	0.92221635	0.9042314	0.85909928	0.80853555
Proportion of Variance	0.03075326	0.02743494	0.0263753	0.02380812	0.02108806
Cumulative Proportion	0.75747267	0.78490761	0.8112829	0.83509102	0.85617908
	Comp.16	Comp.17	Comp.18	Comp.19	Comp.20
Standard deviation	0.7877571	0.74436225	0.72574751	0.69380677	0.67269732
Proportion of Variance	0.0200181	0.01787339	0.01699063	0.01552799	0.01459747

Cumulative Proportion	0.8761972	0.89407058	0.91106120	0.92658920	0.94118667
	Comp.21	Comp.22	Comp.23	Comp.24	Comp.25
Standard deviation	0.63466979	0.61328635	0.55192724	0.397467153	0.384354087
Proportion of Variance	0.01299373	0.01213291	0.00982657	0.005096133	0.004765421
Cumulative Proportion	0.95418041	0.96631331	0.97613988	0.981236017	0.986001438
	Comp.26	Comp.27	Comp.28	Comp.29	
Standard deviation	0.364232811	0.322026864	0.276201256	0.262018088	
Proportion of Variance	0.004279534	0.003345203	0.002460875	0.002214628	
Cumulative Proportion	0.990280972	0.993626175	0.996087050	0.998301679	
	Comp.30	Comp.31			
Standard deviation	0.1712372644	0.1527277294			
Proportion of Variance	0.0009458774	0.0007524438			
Cumulative Proportion	0.9992475562	1.0000000000			

#### 8.4.3.2. Scree Plot

```
plot(data_pca, type = "lines", n pcs = 31)
```



Check the cumulative variance of the first components and the scree plot, and see if the PCA is a good approach to detect the number of factors in this case.

## 8.5. Exploratory Factor Analysis (EFA)

### 8.5.1. Rotational indeterminacy

We can rotate the factors, so that the loadings will be as close as possible to a desired structure.

## 8. Factor Analysis

There are two types of rotation methods.

### 8.5.1.1. Orthogonal

Orthogonal rotation clarifies factor structure **while preserving independence between factors** - assumes the factors are independent from each other (i.e., their correlation is zero).

The underlying factors after rotation will be **uncorrelated**.

The method rotates the factor axes while **keeping them perpendicular**, so the factors remain uncorrelated.

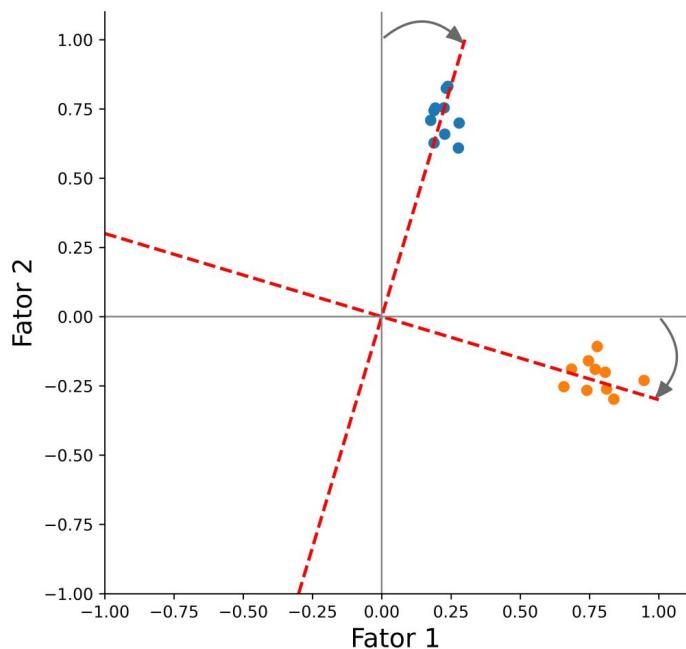


Figure 8.1.: Based in Field (2024)

The goal is to **increase** each item's loading on **one main factor**, and **reduce** its loadings on the other factors, which makes the factors **easier to interpret**.

Methods include:

- Varimax
- Quartimax
- Equimax

### 8.5.1.2. Oblique

Oblique rotation improves factor interpretability while allowing the factors to be **correlated** with each other. Unlike orthogonal rotation, it does **not** force the factors to remain independent.

Because of this, oblique methods are often more suitable for complex and interrelated constructs, where some degree of relationship between factors is expected.

Oblique rotation does **not** require the factor axes to stay perpendicular. As a result, the rotated factors can **correlate**, allowing a more realistic representation of how variables relate to multiple underlying dimensions.

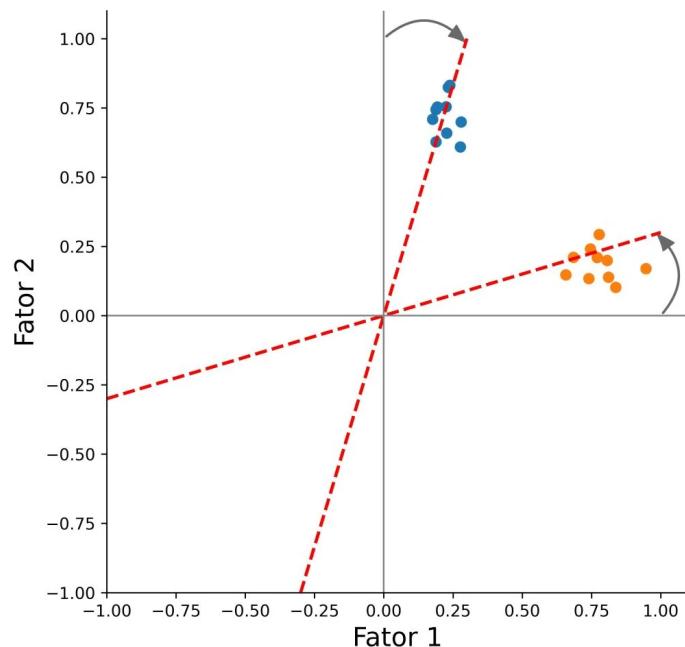


Figure 8.2.: Based in Field (2024)

Methods include:

- Oblimin
- Quartimin
- Promax

Let's run three different factor analysis models with different rotation methods:

- **Model 1:** No rotation
- **Model 2:** Rotation Varimax
- **Model 3:** Rotation Oblimin

## 8. Factor Analysis

```
# No rotation
data_factor = factanal(
  data,
  factors = 4, # change here the number of facotrs based on the EFA
  rotation = "none",
  scores = "regression",
  fm = "ml"
)

# Rotation Varimax
data_factor_var = factanal(
  data,
  factors = 4,
  rotation = "varimax", # orthogonal rotation (default)
  scores = "regression",
  fm = "ml"
)

# Rotation Oblimin
data_factor_obl = factanal(
  data,
  factors = 4,
  rotation = "oblimin", # oblique rotation
  scores = "regression",
  fm = "ml"
)
```

Print out the results of `data_factor_obl`, and have a look.

```
print(data_factor_obl,
      digits = 2,
      cutoff = 0.3, # > 0.3 due to the sample size is higher than 350
      ↵ observations.
      sort = TRUE)
```

Call:

```
factanal(x = data, factors = 4, scores = "regression", rotation = "oblimin", fm = "ml")
```

Uniquenesses:

DWECLAS	INCOME	CHILD13	H18	HEMPLOY	HSIZE	AVADUAGE	IAGE
0.98	0.82	0.09	0.01	0.73	0.01	0.98	0.99
ISEX	NCARS	AREA	BEDROOM	PARK	BEDSIZE	PARKSIZE	RAGE10
0.98	0.64	0.93	0.79	0.89	0.62	0.92	0.91
TCBD	DISTHTC	TWCBD	TDWWK	HEADH	POPDENS	EDUINDEX	GRAVCPC
0.14	0.54	0.80	0.74	0.67	0.78	0.71	0.04

GRAVCPT	GRAVPCPT	NSTRTC	DISTHW	DIVIDX	ACTDENS	DISTCBD
0.05	0.01	0.85	0.66	0.84	0.80	0.31

## Loadings:

	Factor1	Factor2	Factor3	Factor4
TCBD	0.93			
DISTHTC	0.62			
EDUINDEX	-0.53			
GRAVCPC	-0.98			
GRAVCPT	-0.74	-0.55		
DISTHW	0.56			
DISTCBD	0.81			
H18	1.02			
HSIZE	0.74	0.54		
NCARS	0.57			
BEDSIZE	-0.52			
HEADH	-0.56			
GRAVPCPT		1.00		
CHILD13		0.97		
DWELCLAS				
INCOME	0.38			
HEMPLOY	0.47			
AVADUAGE				
IAGE				
ISEX				
AREA				
BEDROOM	0.37			
PARK				
PARKSIZE				
RAGE10				
TWCBD	0.43			
TDWWK	0.49			
POPDENS	-0.41			
NSTRTC	-0.37			
DIVIDX	-0.40			
ACTDENS	-0.42			

	Factor1	Factor2	Factor3	Factor4
SS loadings	5.24	3.13	1.66	1.57

## Factor Correlations:

	Factor1	Factor2	Factor3	Factor4
Factor1	1.000	0.062	0.117	0.021
Factor2	0.062	1.000	0.011	0.199
Factor3	0.117	0.011	1.000	0.054
Factor4	0.021	0.199	0.054	1.000

## 8. Factor Analysis

Test of the hypothesis that 4 factors are sufficient.  
The chi square statistic is 3628.43 on 347 degrees of freedom.  
The p-value is 0

The variability contained in the factors is equal to **Communality + Uniqueness**.

### 8.5.2. Factor scores and factor loadings

In addition to the loading structure, you may also want to know the factor scores of each observation.

We can extract the factor scores with

```
View(data_factor_obl$scores)
# write.csv(data_factor_obl$scores, "data/data_factor_obl_scores.csv", sep
←   = "\t")
head(data_factor_obl$scores)
```

	Factor1	Factor2	Factor3	Factor4
799161661	0.5269517	-0.01526867	-0.4212963	0.6723761
798399409	-0.6999716	-1.28451045	-0.3425305	-0.5906694
798374392	0.3140942	0.17925007	-1.0077065	1.8899099
798275277	0.2576954	1.09054593	-0.1178061	0.3756358
798264250	-0.3366206	-1.08082681	-0.5807561	0.6090050
798235878	-0.5993148	0.84250056	1.2619817	-0.5142385

The individual indicator/subtest scores would be the **weighted sum of the factor scores**, where the weights are the determined by **factor loadings**.

```
View(data_factor_obl$loadings)
# write.csv(data_factor_obl$loadings, "data/data_factor_obl_loadings.csv",
←   sep = "\t")
head(data_factor_obl$loadings)
```

	Factor1	Factor2	Factor3	Factor4
DWELCLAS	-0.0333847085	-0.04628588	0.13740126	0.03391240
INCOME	-0.1309767372	0.37518984	0.04293329	0.10335426
CHILD13	0.0041194104	-0.06692967	0.00769277	0.96689496
H18	0.0008958126	1.01553559	-0.01668507	-0.17758151
HEMPLOY	0.0638121517	0.47320725	0.07699066	0.09956475
HSIZE	-0.0081682615	0.73984312	-0.01057724	0.53609771

### 8.5.3. Visualize Rotation

We will define a plot function to make it easier to visualize several factor pairs.

```
# define a plot function
plot_factor_loading <- function(data_factor,
                                  f1 = 1,
                                  f2 = 2,
                                  method = "No rotation",
                                  color = "blue") {

  # Convert to numeric matrix (works for psych loadings objects)
  L <- as.matrix(data_factor$loadings)

  # Extract selected factors
  df <- data.frame(item = rownames(L), x = L[, f1], y = L[, f2])

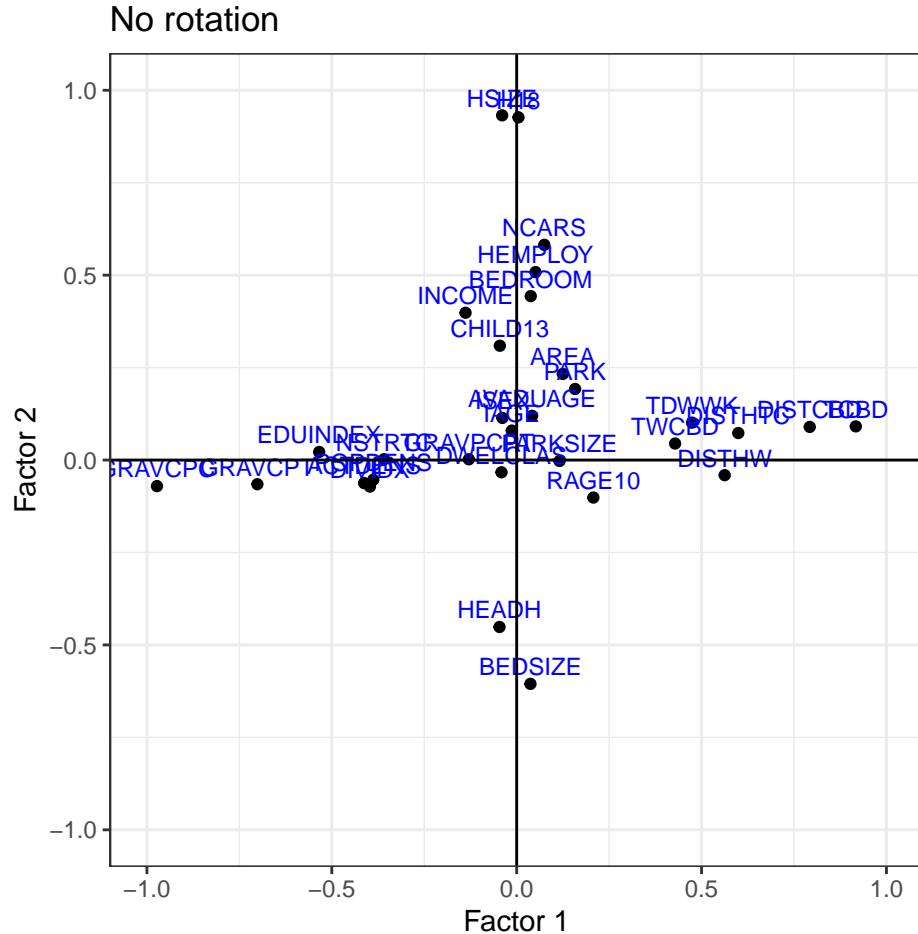
  ggplot(df, aes(x = x, y = y, label = item)) +
    geom_point() +
    geom_text(color = color,
              vjust = -0.5,
              size = 3) +
    geom_hline(yintercept = 0) +
    geom_vline(xintercept = 0) +
    coord_equal(xlim = c(-1, 1), ylim = c(-1, 1)) +
    labs(
      x = paste0("Factor ", f1),
      y = paste0("Factor ", f2),
      title = method
    ) +
    theme_bw()
}
```

#### 8.5.3.1. No Rotation

Plot factor 1 against factor 2, and compare the results

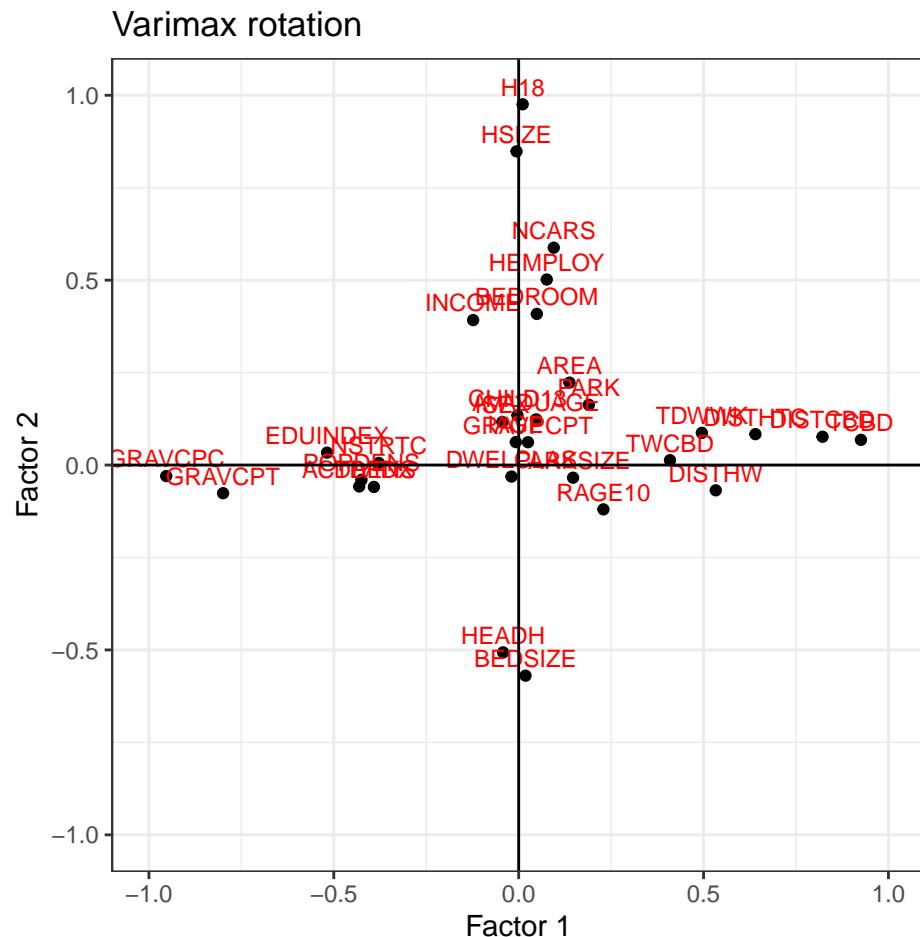
```
plot_factor_loading(
  data_factor = data_factor, # model no rotation
  f1 = 1, # Factor 1
  f2 = 2, # Factor 2
  method = "No rotation", # plot title
  color = "blue"
)
```

## 8. Factor Analysis



### 8.5.3.2. Varimax Rotation

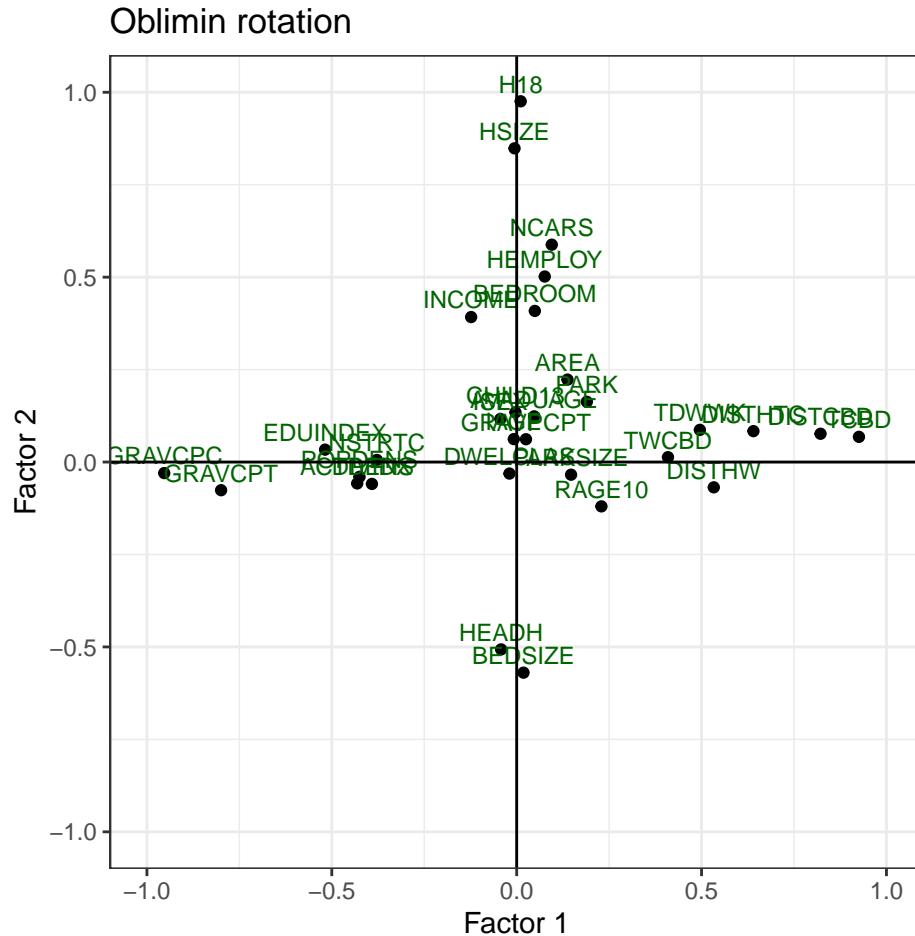
```
plot_factor_loading(  
  data_factor = data_factor_var, # model varimax  
  f1 = 1, # Factor 1  
  f2 = 2, # Factor 2  
  method = "Varimax rotation",  
  color = "red"  
)
```



### 8.5.3.3. Oblimin Rotation

```
plot_factor_loading(
  data_factor = data_factor_var, # model oblimin
  f1 = 1, # Factor 1
  f2 = 2, # Factor 2
  method = "Oblimin rotation",
  color = "darkgreen"
)
```

## 8. Factor Analysis



When we have more than two factors it is difficult to analyse the factors by the plots.

Variables that have low explaining variance in the two factors analysed, could be highly explained by the other factors not present in the graph.

### 💡 Your turn

Try comparing the plots with the factor loadings and plot the other factor pairs (replace f1 and f2) to get more familiar with exploratory factor analysis.

Interpret the factors and try to give them a name.

For instance, we can plot all the factors against each other as follows:

```
# create all combinations
plot_factor_loading(data_factor, 1, 2, method = "No rotation")
plot_factor_loading(data_factor, 1, 3, method = "No rotation")
plot_factor_loading(data_factor, 1, 4, method = "No rotation")
plot_factor_loading(data_factor, 2, 3, method = "No rotation")
plot_factor_loading(data_factor, 2, 4, method = "No rotation")
plot_factor_loading(data_factor, 3, 4, method = "No rotation")
```

# 9. Cluster Analysis

## 💡 Do it yourself with R

Copy the script [ClusterAnalysis.R](#) and paste it in your session.

Run each line using CTRL + ENTER

Based on a dataset of European **Airports**, we will create **clusters** based on the observations.

## ❗ Your task

Create and assess the many types of clustering methods.

## 9.1. Load packages

```
library(tidyverse) # Pack of most used libraries for data science
library(readxl) # Import excel files
library(skimr) # Summary statistics
library(mclust) # Model based clustering
library(cluster) # Cluster analysis
library(factoextra) # Visualizing distances
```

## 9.2. Dataset

Included variables:

- **Code** - Code of the airport
- **Airport** - Name of the airport
- **Ordem** - ID of the observations
- **Passengers** - Number of passengers
- **Movements** - Number of flights
- **Numberofairlines** - Number of airlines at each airport
- **Mainairlineflightspercentage** - Percentage of flights of the main airline of each airport
- **Maximumpercentageoftafficpercountry** - Maximum percentage of flights per country

## 9. Cluster Analysis

- NumberofLCCflightsweekly - Number of weekly low cost flights
- NumberofLowCostAirlines - Number of low cost airlines of each airport
- LowCostAirlinespercentage - Percentage of the number of low cost airlines in each airport
- Destinations - Number of flights arriving at each airport
- Average\_route\_Distance - Average route distance in km
- DistancetoclosestAirport - Distance to closest airport in km
- DistancetoclosestSimilarAirport - Distance to closest similar airport in km
- AirportRegionalRelevance - Relevance of the airport in a regional scale (0 - 1)
- Distancetocitykm - Distance between the airport and the city in km
- Inhabitantscorrected - Population of the city
- numberofvisitorscorrected - Number of visitors arrived in the airport
- GDPcorrected - Corrected value of the Gross Domestic Product
- Cargoton - The total number of cargo [ton] transported in a certain period multiplied by the number of flights.

### 9.2.1. Import dataset

```
data = read_excel("../data/Data_Aeroports_Clustersv1.xlsx")
data = data.frame(data) # as data frame only
```

### 9.2.2. Get to know your dataset

Take a look at the first values of the dataset

```
head(data, 5)
```

	Code	Ordem	Airport	Passengers	Movements	Numberofairlines
1	NCE	1	Nice Côte d'Azur	9830987	119322	64
2	CGN	2	Cologne Bonn	9742300	132200	29
3	LPA	3	Gran Canaria	9155665	101557	47
4	ALC	4	Alicante	9139479	74281	35
5	LTN	5	London Luton	9129053	83013	11
			Mainairlineflightspercentage	Maximumpercentageoftrafficpercountry		
1			18		20	
2			33		13	
3			17		26	
4			29		23	
5			37		22	
			NumberofLCCflightsweekly	NumberofLowCostAirlines	LowCostAirlinespercentage	
1			256	18	28.12500	
2			351	12	41.37931	
3			259	19	40.42553	
4			300	18	51.42857	

## 9. Cluster Analysis

5	227	8	72.72727
Destinations Average_Route_Distance DistancetoclosestAirport			
1	104	1253	23.66681
2	189	1721	63.45766
3	116	3143	122.58936
4	160	1701	63.09924
5	87	1582	45.13247
DistancetoclosestSimilarAirport AirportRegionalrelevance Distancetocitykm			
1	223.83824	0.8698581	6
2	63.45766	0.5127419	15
3	132.45082	0.7840877	19
4	134.50558	0.8098081	9
5	45.13247	0.1947903	55
Inhabitantscorrected numberofvisitorscorrected GDPcorrected Cargoton			
1	3551805.0	2152829.8	26300 11223.39
2	4180133.5	1151381.6	30100 562.00
3	705807.8	1678968.6	20700 25994.00
4	1508358.6	1944196.8	25000 3199.73
5	1562709.8	181063.5	32000 28698.00

Check summary statistics of variables

```
skim(data)
```

Table 9.1.: Data summary

Name	data
Number of rows	32
Number of columns	21
Column type frequency:	
character	2
numeric	19
Group variables	
	None

Variable type: character

skim_variable	n_missing	complete_rate	min	max	empty	n_unique	whitespace
Code	0	1	3	3	0	32	0
Airport	0	1	4	35	0	32	0

Variable type: numeric

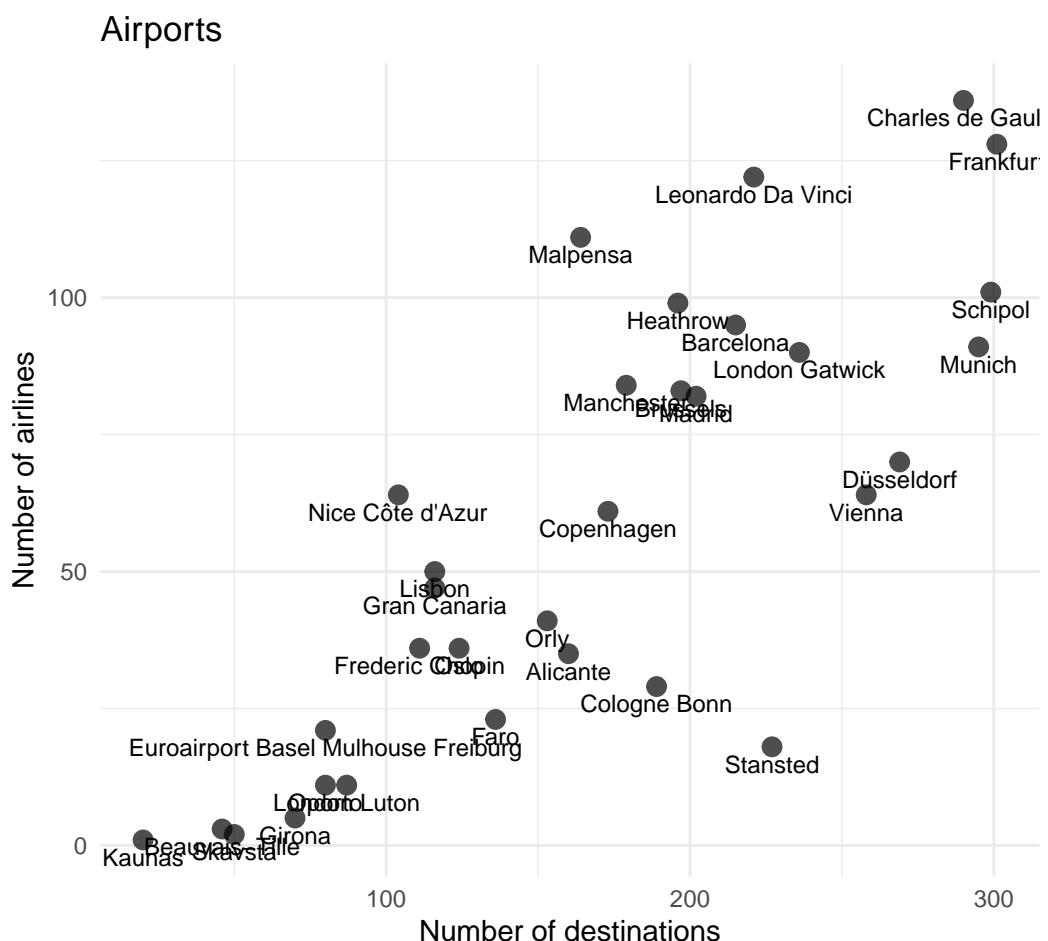
skim_variable	n_missing	complete_rate	mean	sd	p0	p25	p50	p75	p100	hist
Ordem	0	1	16.50	9.38	1.00	8.75	16.50	24.25	32.00	
Passengers	0	1	20750710.88	7601931.34	56698.00	8927021.50	17275317.50	28666511.50	67054745.00	
Movements	0	1	205111.16	143564.45	5698.00	82765.75	191742.50	258654.50	518018.00	
Numberofairlines	0	1	57.81	40.42	1.00	22.50	55.50	90.25	136.00	
Mainairlineflightspercentage	0	1	33.78	22.08	12.00	22.00	28.50	33.00	95.00	
Maximumpercentageoftrafficpercountry	1	17.47	7.31	9.00	12.00	15.00	22.25	35.00		
NumberofLCCflightsweekly	0	1	397.59	221.56	37.00	226.25	366.50	546.75	776.00	
NumberofLowCostAirlines	0	1	11.59	5.60	1.00	7.75	12.00	16.00	23.00	
LowCostAirlinespercentage	0	1	36.44	30.10	6.25	16.29	19.59	50.36	100.00	
Destinations	0	1	167.62	80.13	20.00	109.25	168.50	222.50	301.00	
Average_Route_Distance	0	1	2275.19	930.28	1225.00	1599.50	2152.00	2765.00	5635.00	
DistancetoclosestAirport	0	1	90.19	64.56	13.84	45.83	66.50	111.61	244.50	
DistancetoclosestSimilarAirport	0	1	248.64	183.60	38.16	97.74	206.12	376.15	635.05	

## 9. Cluster Analysis

skim_variable	n_missing	complete_rate	mean	sd	p0	p25	p50	p75	p100	hist
AirportRegionalrelevance	0	1	0.73	0.23	0.19	0.58	0.80	0.91	0.99	
Distancetocitykm	0	1	25.81	25.44	3.00	9.75	14.50	35.00	100.00	
Inhabitanntscorrected	0	1	4528561.95	2590542.88	329240.50	2856960.30	4532760.00	6733158.88	9870818.00	
numberofvisitorscorrected	0	1	2766002.58	2549773.72	80232.50	1018390.89	1806295.60	3450491.78	9732062.00	
GDPcorrected	0	1	30160.75	10510.93	8500.00	25000.00	31150.00	35550.00	56600.00	
Cargoton	0	1	236531.76	478310.12	0.00	10325.00	72749.85	153372.85	1819000.00	

As exploring the data, we can plot the Numberofairlines against the Destinations and observe.

```
ggplot(data, aes(x = Destinations, y = Numberofairlines)) +
  geom_point(size = 3, alpha = 0.7) +
  geom_text(aes(label = Airport), vjust = 1.5, size = 3, show.legend =
    FALSE) +
  labs(
    title = "Airports",
    x = "Number of destinations",
    y = "Number of airlines"
  ) +
  theme_minimal()
```



By looking at the plot, you may already have a clue on the number of clusters with this two variables. However, this is not clear and it does not consider the **other variables** in the analysis.

### 9.2.3. Prepare data

#### 9.2.3.1. Row-names

Make the Code variable as **row names** or case number

```
data = data |> column_to_rownames(var = "Code")
```

#### 9.2.3.2. Remove the non-continuous data

Leave only continuous variables and the ordered ID.

```
data_continuous = data |> select(-Ordem, -Airport) # remove chr and id
                           ↵ variables
```

#### 9.2.3.3. Standardize variables

Take a look at the scale of the variables. See how different they are!

```
head(data_continuous)
```

	Passengers	Movements	Numberofairlines	Mainairlineflightspercentage
NCE	9830987	119322	64	18
CGN	9742300	132200	29	33
LPA	9155665	101557	47	17
ALC	9139479	74281	35	29
LTN	9129053	83013	11	37
WAW	8320927	115934	36	31
	Maximumpercentageoftafficpercountry	NumberofLCCflightsweekly		
NCE		20	256	
CGN		13	351	
LPA		26	259	
ALC		23	300	
LTN		22	227	
WAW		14	341	
	NumberofLowCostAirlines	LowCostAirlinespercentage	Destinations	
NCE	18	28.12500	104	
CGN	12	41.37931	189	
LPA	19	40.42553	116	

## 9. Cluster Analysis

ALC	18	51.42857	160
LTN	8	72.72727	87
WAW	7	19.44445	111
<b>Average_Route_Distance</b>			
NCE	1253	23.66681	
CGN	1721	63.45766	
LPA	3143	122.58936	
ALC	1701	63.09924	
LTN	1582	45.13247	
WAW	1460	244.49577	
<b>DistancetoclosestSimilarAirport</b>			
NCE	223.83824	0.8698581	6
CGN	63.45766	0.5127419	15
LPA	132.45082	0.7840877	19
ALC	134.50558	0.8098081	9
LTN	45.13247	0.1947903	55
WAW	559.31000	0.9810450	10
<b>Inhabitantscorrected</b>			
NCE	3551805.0	2152829.8	26300 11223.39
CGN	4180133.5	1151381.6	30100 562.00
LPA	705807.8	1678968.6	20700 25994.00
ALC	1508358.6	1944196.8	25000 3199.73
LTN	1562709.8	181063.5	32000 28698.00
WAW	6626197.0	770720.5	11200 82756.54

Z-score standardization:

$$Z = \frac{X - \mu}{\sigma}$$

```
data_scaled = data_continuous |>
  mutate(across(everything(), ~ ( . - mean(.) ) / sd(.)))
# Result = z-scores, same as scale()
```

## 9.3. Hierarchical Clustering

### 9.3.1. Distance measures

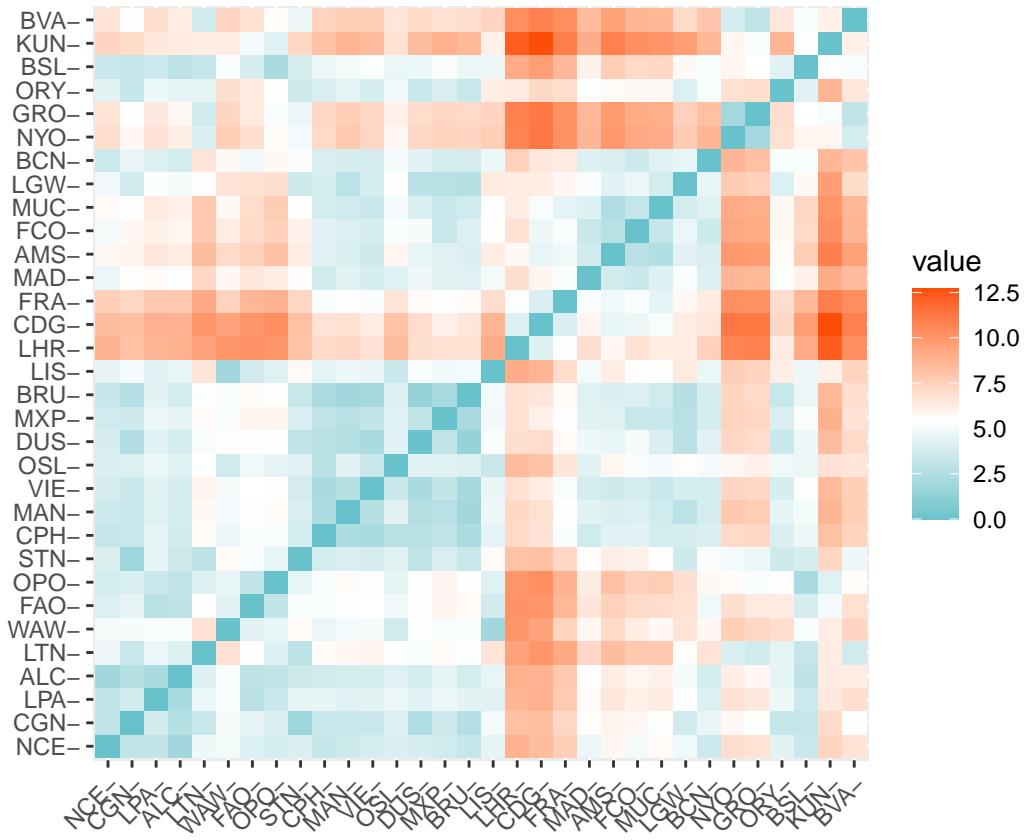
Similarity of observations can be measured through different **distance measures**, including:

- Euclidean distance
- Minkowski distance
- Manhattan distance
- Mahalanobis distance

Let's measure the **euclidean distances** of our standartize data and visualize them on a heatmap

```
# measure
distance = dist(data_scaled, method = "euclidean")

# heatmap
fviz_dist(
  distance,
  gradient = list(
    low = "#00AFBB",
    mid = "white",
    high = "#FC4E07"
  ),
  order = FALSE
)
```



By the color codes, you can have a clue of the airports that are more similar.

### 9.3.2. Types of hierarchical clustering

There are many types of hierarchical clustering. We will explore some of them:

## 9. Cluster Analysis

- Single linkage (nearest neighbour) clustering algorithm
- Complete linkage (Farthest neighbour) clustering algorithm
- Average linkage between groups
- Ward's method
- Centroid method

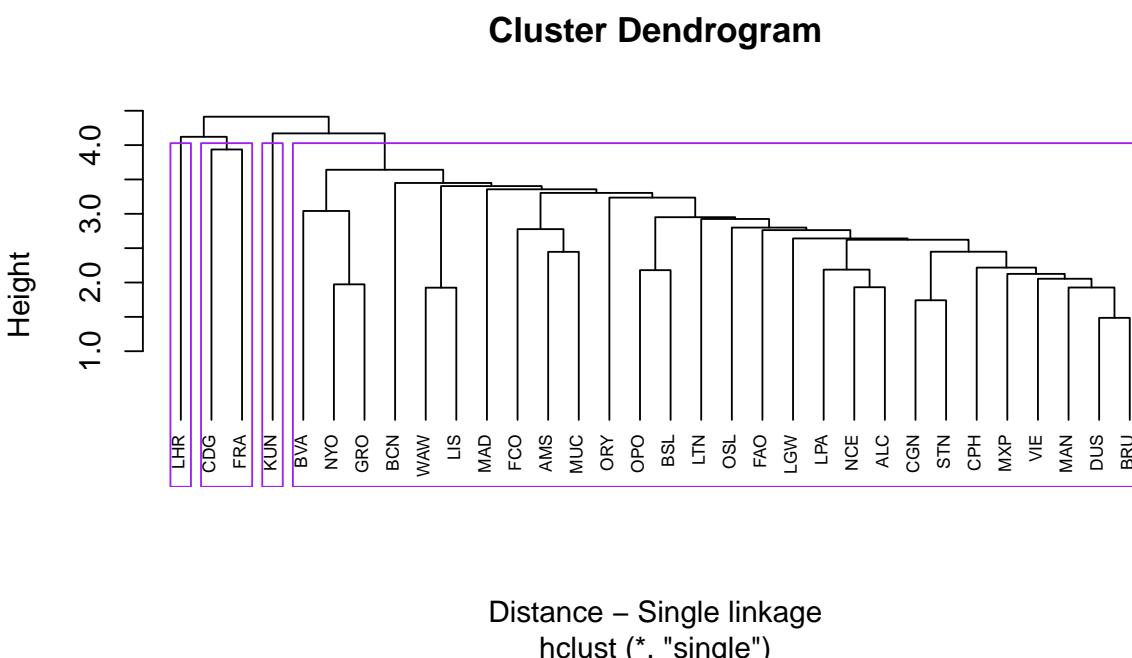
### 9.3.2.1. Single linkage

The *single linkage* method (which is closely related to the minimal spanning tree) adopts a ‘friends of friends’ clustering strategy.

This clustering algorithm is based on a bottom-up approach, by linking two clusters that have the **closest distance** between each other.

```
cluster_single = hclust(distance, "single")

# dendogram
plot(
  cluster_single,
  xlab = "Distance - Single linkage",
  hang = -1, # all to the bottom
  cex = 0.6 # label text size
)
rect.hclust(cluster_single, k = 4, border = "purple") # cut on the
#   dendrogram at 4 clusters
```



This results in 4 clusters, with *Heathrow Airport* and *Kaunas Airport* at their own cluster.

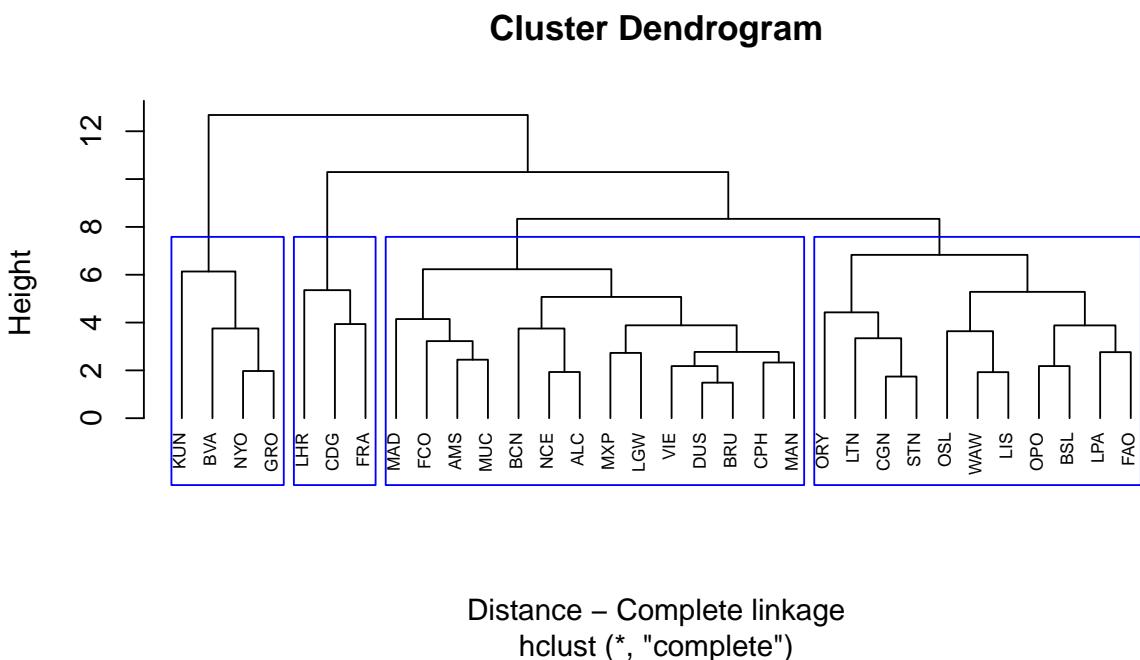
### 9.3.2.2. Complete linkage

The *complete linkage* method finds similar clusters.

Complete linkage is based on the **maximizing distance** between observations in each cluster.

```
cluster_complete = hclust(distance, "complete")

# dendrogram
plot(
  cluster_complete,
  xlab = "Distance - Complete linkage",
  hang = -1, # all to the bottom
  cex = 0.6 # label text size
)
rect.hclust(cluster_complete, k = 4, border = "blue") # cut on the
#   dendrogram at 4 clusters
```



### 9.3.2.3. Average linkage

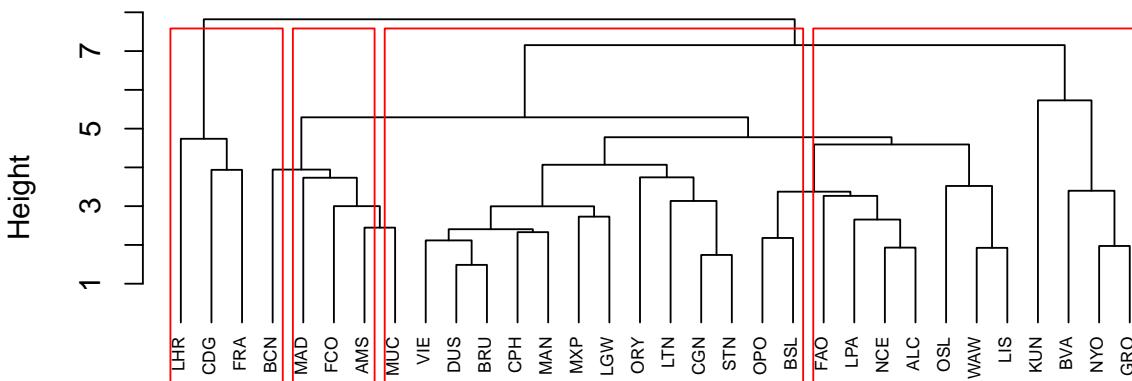
The average linkage considers the distance between clusters to be the average of the **distances between observations in one cluster to all the members in the other cluster**.

## 9. Cluster Analysis

```
cluster_average = hclust(distance, "average")

# dendrogram
plot(
  cluster_average,
  xlab = "Distance - Average linkage",
  hang = -1, # all to the bottom
  cex = 0.6 # label text size
)
rect.hclust(cluster_complete, k = 4, border = "red") # cut on the
#   dendrogram at 4 clusters
```

**Cluster Dendrogram**



Distance – Average linkage  
hclust (\*, "average")

### 9.3.2.4. Ward's method

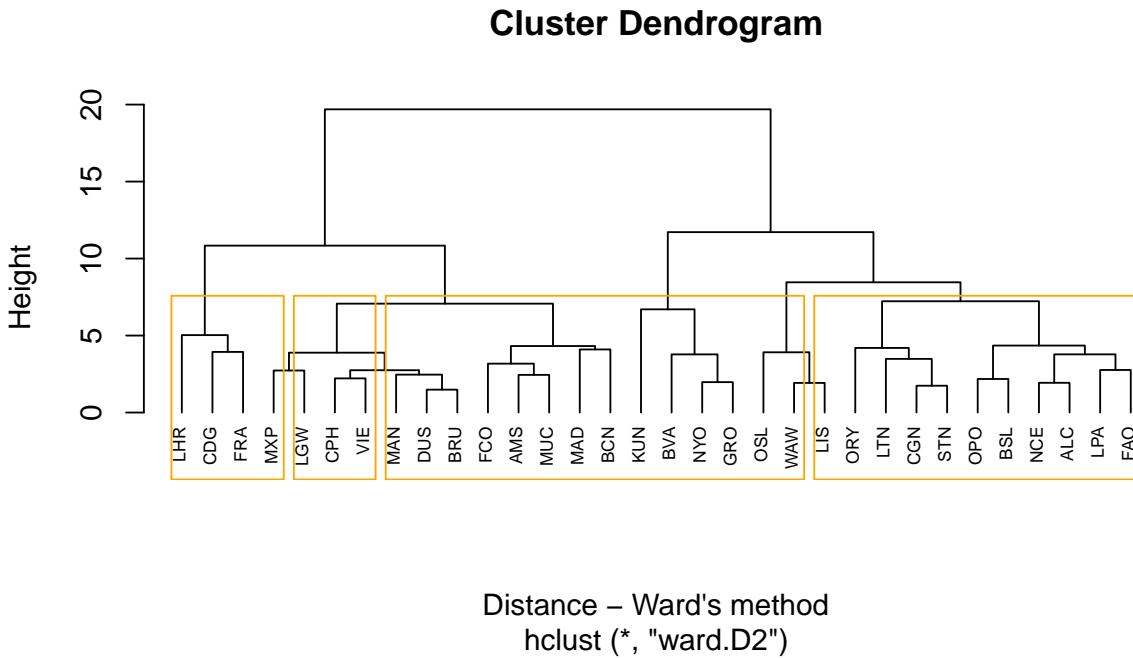
Ward's minimum variance method aims at finding compact, spherical clusters.

The Ward's method considers the measures of similarity as the **sum of squares within the cluster** summed over all variables.

```
cluster_ward = hclust(distance, "ward.D2")

# dendrogram
plot(
  cluster_ward,
  xlab = "Distance - Ward's method",
  hang = -1, # all to the bottom
  cex = 0.6 # label text size
```

```
)
rect.hclust(cluster_complete, k = 4, border = "orange") # cut on the
↪ dendrogram at 4 clusters
```



### 9.3.2.5. Centroid method

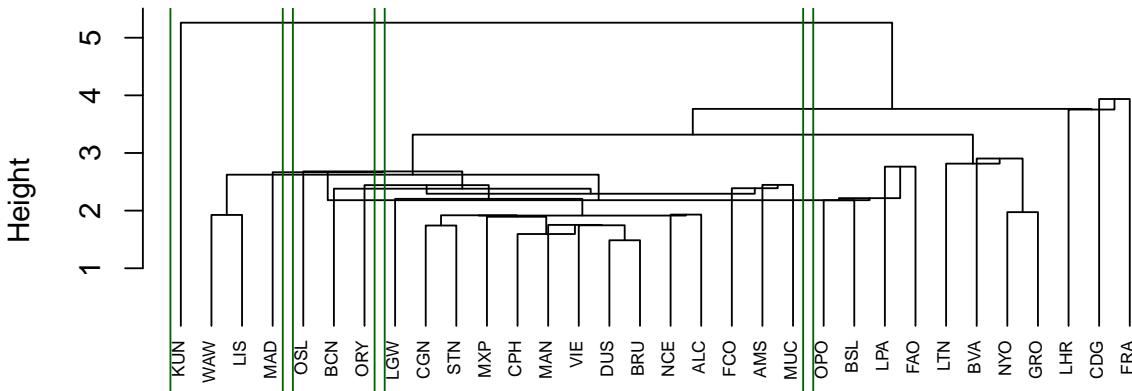
The centroid method considers the similarity between two clusters as the distance between its centroids.

```
cluster_centroid = hclust(distance, "centroid")

# dendrogram
plot(
  cluster_centroid,
  xlab = "Distance - Centroid method",
  hang = -1, # all to the bottom
  cex = 0.6 # label text size
)
rect.hclust(cluster_complete, k = 4, border = "darkgreen") # cut on the
↪ dendrogram at 4 clusters
```

## 9. Cluster Analysis

### Cluster Dendrogram



Distance – Centroid method  
hclust (\*, "centroid")

🔥 As mentioned in R Core Team (2025), note however, that methods `centroid` are *not* leading to a *monotone distance* measure, or equivalently the resulting dendograms can have so called *inversions* or *reversals* which are **hard to interpret**, but note the trichotomies in Legendre and Legendre (2012) .

#### 9.3.3. Comparing results from different hierarchical methods

Now let's assess the **membership** of each observation with the `cutree` function for each method.

```
number_clusters = 4 # change here
member_single = cutree(cluster_single, k = number_clusters)
member_complete = cutree(cluster_complete, k = number_clusters)
member_average = cutree(cluster_average, k = number_clusters)
member_ward = cutree(cluster_ward, k = number_clusters)
member_centroid = cutree(cluster_centroid, k = number_clusters)
```

We can make a table of cluster **memberships** for each observation to each cluster method, and **compare them** with a color code.

```
# make a data frame
cluster_membership = data.frame(member_single,
                                 member_complete,
                                 member_average,
                                 member_ward,
```

```
        member_centroid
    )

# manipulate data for plot
cluster_long = cluster_membership |>
  rownames_to_column(var = "airport") |> # keep airport names
  pivot_longer(
    cols = starts_with("member_"),
    names_to = "method",
    values_to = "cluster") |>
  mutate(method = gsub("member_", "", method), # clean names
         method = factor(method, # preserve the label order
                           levels = c("single", "complete", "average",
                                     "ward", "centroid")))

# plot
ggplot(cluster_long,
       aes(x = method,
           y = airport,
           fill = factor(cluster))) +
  geom_tile(color = "white") +
  scale_fill_brewer(palette = "Set3", name = "Cluster") +
  theme_minimal() +
  labs(title = "Cluster memberships by method",
       x = "Clustering method",
       y = "Airport") +
  theme(axis.text.y = element_text(size = 6))
```

## 9. Cluster Analysis



Compare how common each method is to each other:

```
table(member_complete, member_average) # complete linkage vs. average
#<-- linkage
```

		member_average			
member_complete		1	2	3	4
1	14	0	0	0	
2	11	0	0	0	
3	0	3	0	0	
4	0	0	3	1	

```
table(member_complete, member_ward) # complete linkage vs. ward's method
```

		member_ward			
member_complete		1	2	3	4
1	2	12	0	0	
2	11	0	0	0	

```
3 0 0 3 0
4 0 0 0 4
```

**⚠ Your turn**

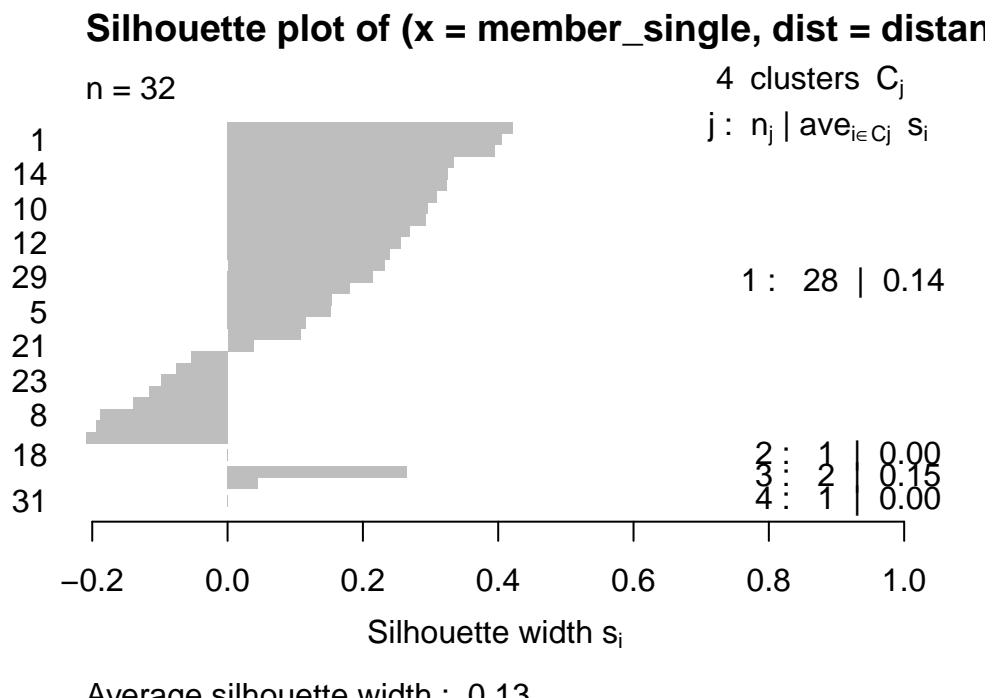
Try comparing other methods, and evaluate how common they are.

### 9.3.4. Silhouette Plots

The silhouette plot evaluates how similar an observation is to its own cluster compared to other clusters.

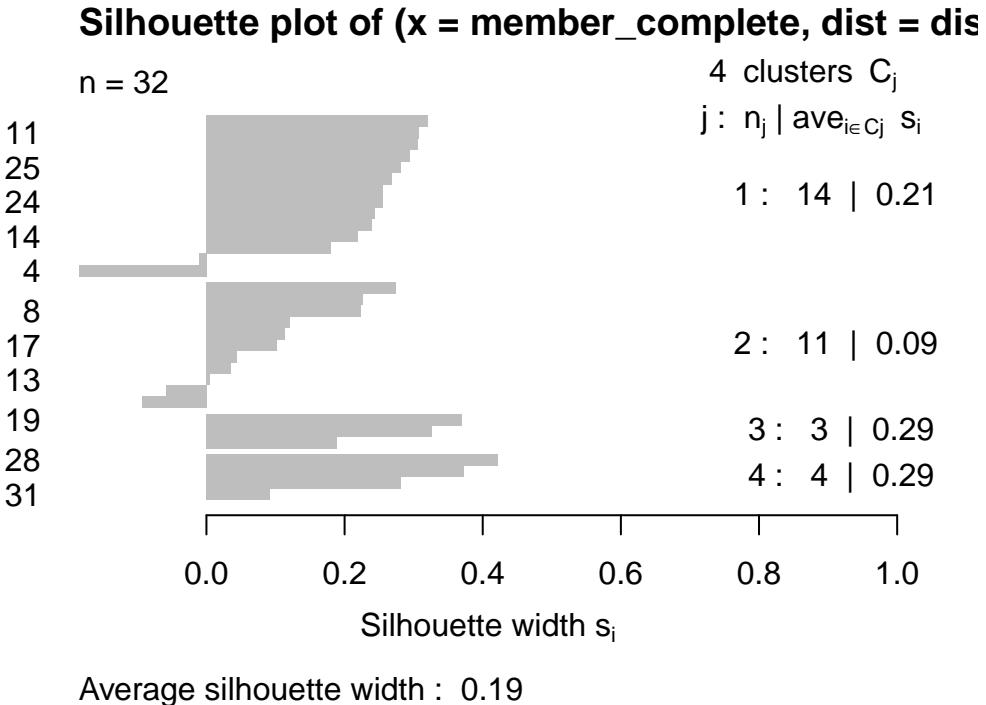
The clustering configuration is appropriate when **most objects have high values**.  
**Low or negative values** indicate that the clustering method **is not appropriate** or the number of clusters **is not ideal**.

```
plot(silhouette(member_single, distance))
```

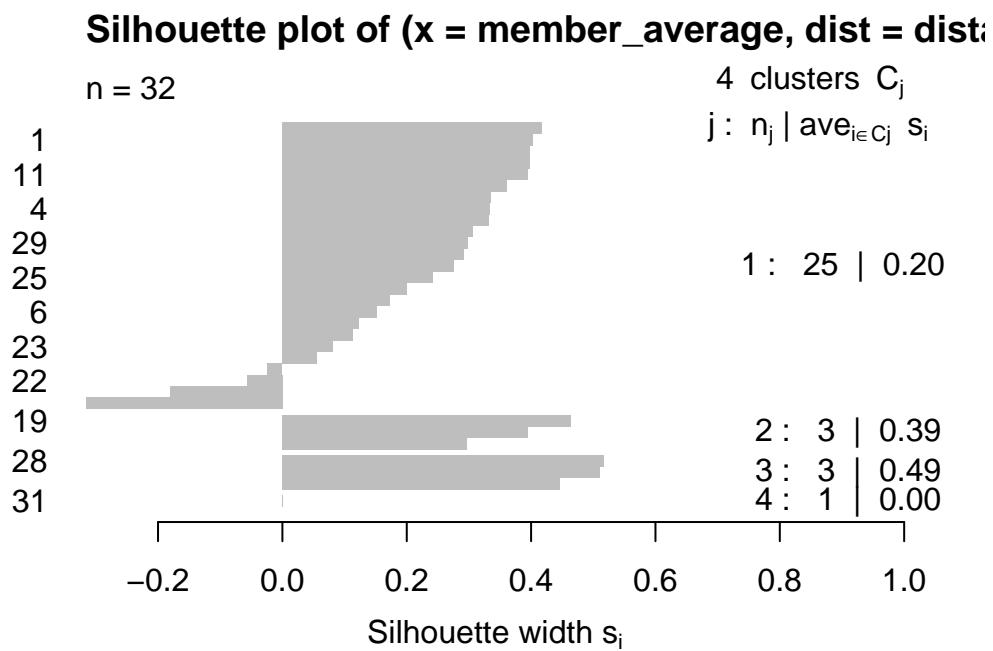


```
plot(silhouette(member_complete, distance))
```

## 9. Cluster Analysis



```
plot(silhouette(member_average, distance))
```



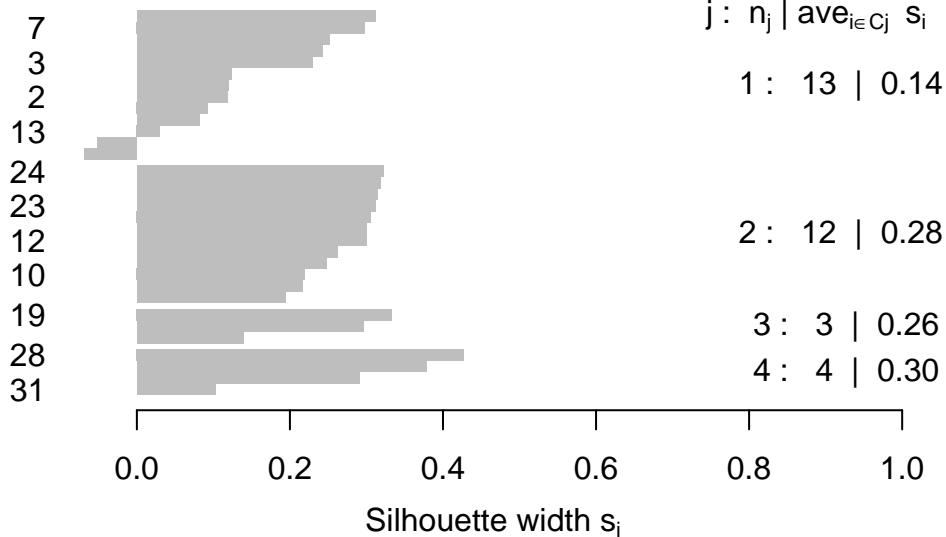
```
plot(silhouette(member_ward, distance))
```

**Silhouette plot of (x = member\_ward, dist = distanc)**

n = 32

4 clusters  $C_j$ j :  $n_j | \text{ave}_{i \in C_j} s_i$ 

1 : 13 | 0.14



Average silhouette width : 0.22

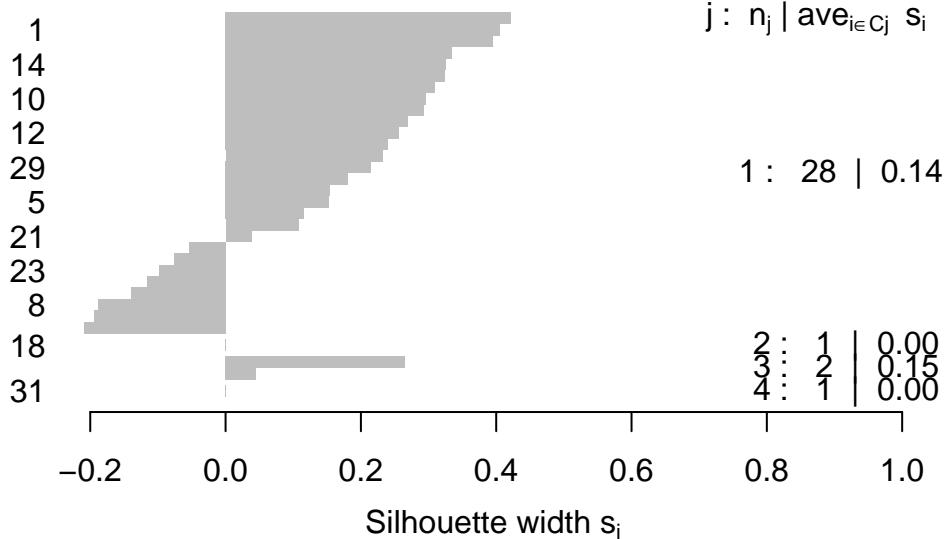
```
plot(silhouette(member_centroid, distance))
```

**Silhouette plot of (x = member\_centroid, dist = dist)**

n = 32

4 clusters  $C_j$ j :  $n_j | \text{ave}_{i \in C_j} s_i$ 

1 : 28 | 0.14



Average silhouette width : 0.13

## 9.4. Non-Hierarchical Clustering

Non-hierarchical clustering is a method that partitions data points into a **predetermined number of clusters**, denoted by  $k$ , without creating a nested tree-like structure.

Unlike hierarchical clustering, it requires the user to **specify in advance** and uses an iterative algorithm to optimize a criterion, such as **minimizing the variance within each cluster**.

Popular examples include *K-Means* and *K-Medoids*, which assign points to the nearest cluster center (centroid or medoid) and repeat the process until the cluster assignments no longer change significantly.

In this exercise, we will use the **K-means clustering**.

### 9.4.1. Choose the number of clusters

We can observe a measure of the goodness of the classification for each k-means, using the following ratio:

$$\frac{\text{Between}_{SS}}{\text{Total}_{SS}}$$

**SS** stands for Sum of Squares, so it's the usual decomposition of deviance in deviance “Between” and deviance “Within”. Ideally you want a clustering that has the properties of internal cohesion and external separation, i.e. the **BSS/TSS ratio** should approach 1.

This algorithm will detect how many clusters, from 1 to 10, explains **more variance, with less clusters**:

```
# loop for the 10 cluster trials
kmeans_diagnostic = data.frame()

for (i in 1:10) {
  km = kmeans(data_scaled, centers = i)
  km_diagn = data.frame(
    k = i,
    between_ss = km$betweenss,
    tot_ss = km$totss,
    ratio = km$betweenss / km$totss
  )
  kmeans_diagnostic = rbind(kmeans_diagnostic, km_diagn)
}

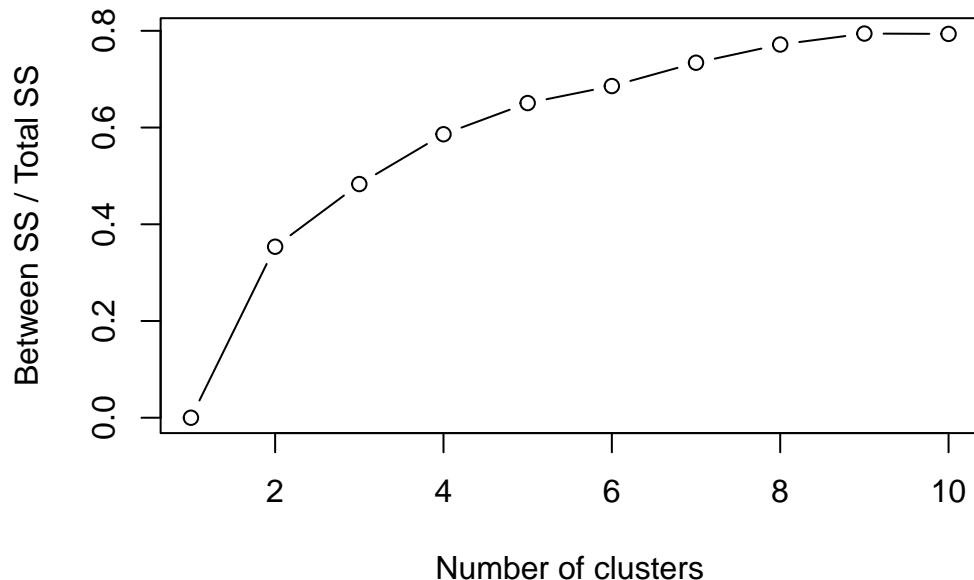
# marginal improvements for each new cluster
kmeans_diagnostic = kmeans_diagnostic |>
  mutate(marginal = ratio - lag(ratio))

kmeans_diagnostic
```

	k	between_ss	tot_ss	ratio	marginal
1	1	1.136868e-13	558	2.037399e-16	NA
2	2	1.972975e+02	558	3.535798e-01	0.3535798112
3	3	2.696136e+02	558	4.831784e-01	0.1295986110
4	4	3.270613e+02	558	5.861313e-01	0.1029528721
5	5	3.630618e+02	558	6.506484e-01	0.0645171259
6	6	3.826730e+02	558	6.857939e-01	0.0351454860
7	7	4.095853e+02	558	7.340238e-01	0.0482298606
8	8	4.305964e+02	558	7.716782e-01	0.0376544470
9	9	4.432340e+02	558	7.943261e-01	0.0226479106
10	10	4.428430e+02	558	7.936255e-01	-0.0007006368

Plot the ratio into a **scree plot**

```
plot(kmeans_diagnostic$k, kmeans_diagnostic$ratio,
      type = "b",
      ylab = "Between SS / Total SS",
      xlab = "Number of clusters")
```



#### 9.4.2. Predetermined number of clusters

In this case we will use the **K-means clustering**, and define **k = 3** (3 clusters).

Here are the cluster results.

## 9. Cluster Analysis

```
km_clust = kmeans(data_scaled, centers = 3) # k = 3
km_clust # print the results
```

K-means clustering with 3 clusters of sizes 16, 3, 13

Cluster means:

	Passengers	Movements	Numberofairlines	Mainairlineflightspercentage
1	-0.7166133	-0.7916255	-0.8241767	0.4910120
2	2.2082860	1.9174235	1.5633070	-0.2919658
3	0.3723812	0.5318260	0.6536081	-0.5369457
	Maximumpercentageoftafficpercountry	NumberofLCCflightsweekly		
1	0.5857676	-0.7739065		
2	-0.7482433	1.5198394		
3	-0.5482732	0.6017682		
	NumberofLowCostAirlines	LowCostAirlinespercentage	Destinations	
1	-0.3851321	0.6868110	-0.7534855	
2	-0.2251255	-0.9329811	1.1819686	
3	0.5259607	-0.6300025	0.6546048	
	Average_Route_Distance	DistancetoclosestAirport		
1	-0.6283075	0.4268811		
2	2.1737031	-0.2311652		
3	0.2716778	-0.4720463		
	DistancetoclosestSimilarAirport	AirportRegionalrelevance	Distancetocitykm	
1	-0.1926471	-0.227674	0.2456769	
2	0.1667423	0.190403	-0.2415823	
3	0.1986251	0.236275	-0.2466218	
	Inhanbitantscorrected	numberofvisitorscorrected	GDPcorrected	Cargoton
1	-0.6365147	-0.5857249	-0.5626763	-0.4289738
2	0.9414886	1.6718238	1.5164455	2.9391006
3	0.5661361	0.3350867	0.3425757	-0.1502862

Clustering vector:

NCE	CGN	LPA	ALC	LTN	WAW	FAO	OPO	STN	CPH	MAN	VIE	OSL	DUS	MXP	BRU	LIS	LHR	CDG	FRA		
1	1	1	1	1	1	1	1	1	1	1	3	3	3	1	3	3	3	1	2	2	2

MAD	AMS	FCO	MUC	LGW	BCN	NYO	GRO	ORY	BSL	KUN	BVA
3	3	3	3	3	3	1	1	3	1	1	1

Within cluster sum of squares by cluster:

```
[1] 193.76889 20.38512 86.77173
(between_SS / total_SS = 46.1 %)
```

Available components:

```
[1] "cluster"      "centers"       "totss"        "withinss"      "tot.withinss"
[6] "betweenss"    "size"          "iter"         "ifault"
```

If we want to export the **cluster means for each variable** and the **cluster membership for each observation**:

```
# cluster means for each variable
var_cluster_means = data.frame(cluster = 1:nrow(km_clust$centers),
                                size = km_clust$size,
                                km_clust$centers)

# cluster membership for each observation
obs_cluster_member = data.frame(km_clust$cluster)
```

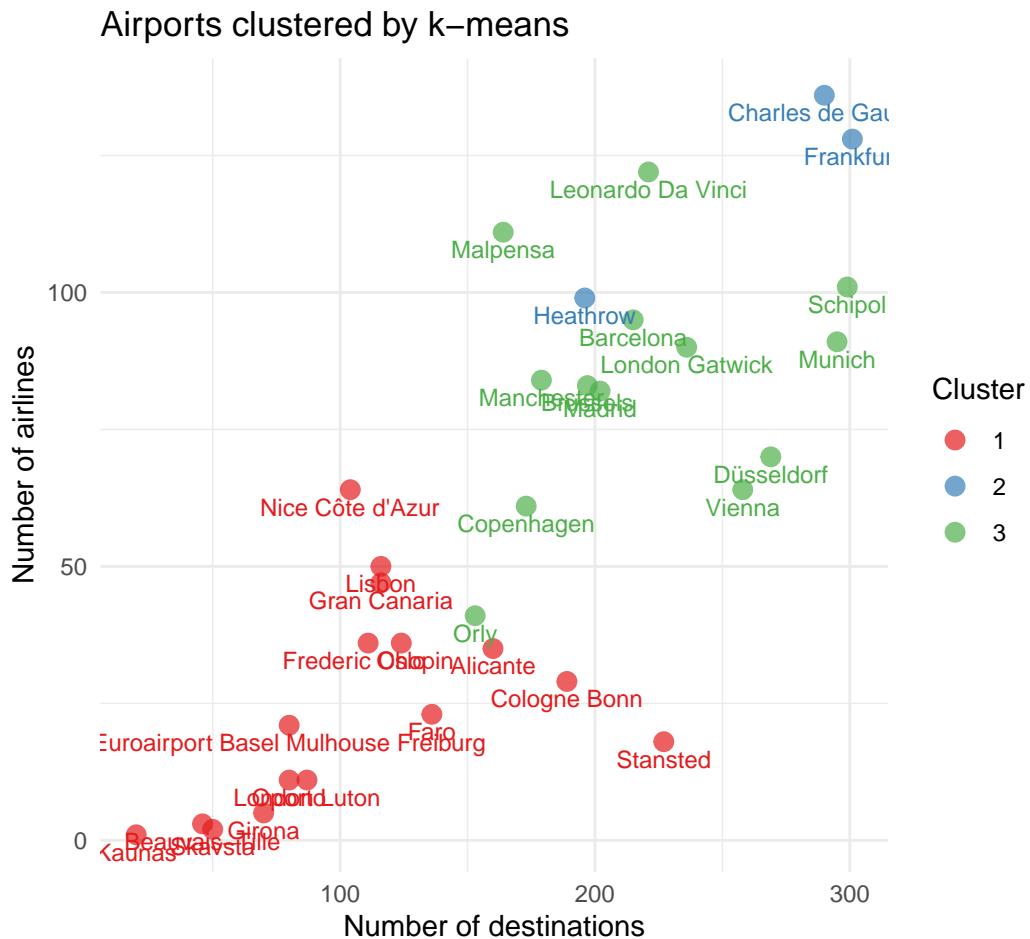
### 9.4.3. Plotting the clusters

Finally, plot again the Numberofairlines against the Destinations and observe the clusters results to check if they make sense.

```
# add cluster membership to original data
data_clust = data |> mutate(cluster = factor(km_clust$cluster))

# plot
ggplot(data_clust, aes(x = Destinations, y = Numberofairlines, color =
  ↪ cluster)) +
  geom_point(size = 3, alpha = 0.7) +
  geom_text(aes(label = Airport), vjust = 1.5, size = 3, show.legend =
  ↪ FALSE) +
  scale_color_brewer(palette = "Set1") +
  labs(
    title = "Airports clustered by k-means",
    x = "Number of destinations",
    y = "Number of airlines",
    color = "Cluster"
  ) +
  theme_minimal()
```

## 9. Cluster Analysis



### 💡 What if... ?

Imagine that one of the airports was not operating any more.  
 Remove one airport from the data, at your choice, and **re-run the cluster analysis.**  
 How different are the results?

## **Part II.**

# **GIS in R**



# 10. 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).

## 10.1. 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")
```



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)
```

### 10.1.1. 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.

## 10. Introduction to spatial data

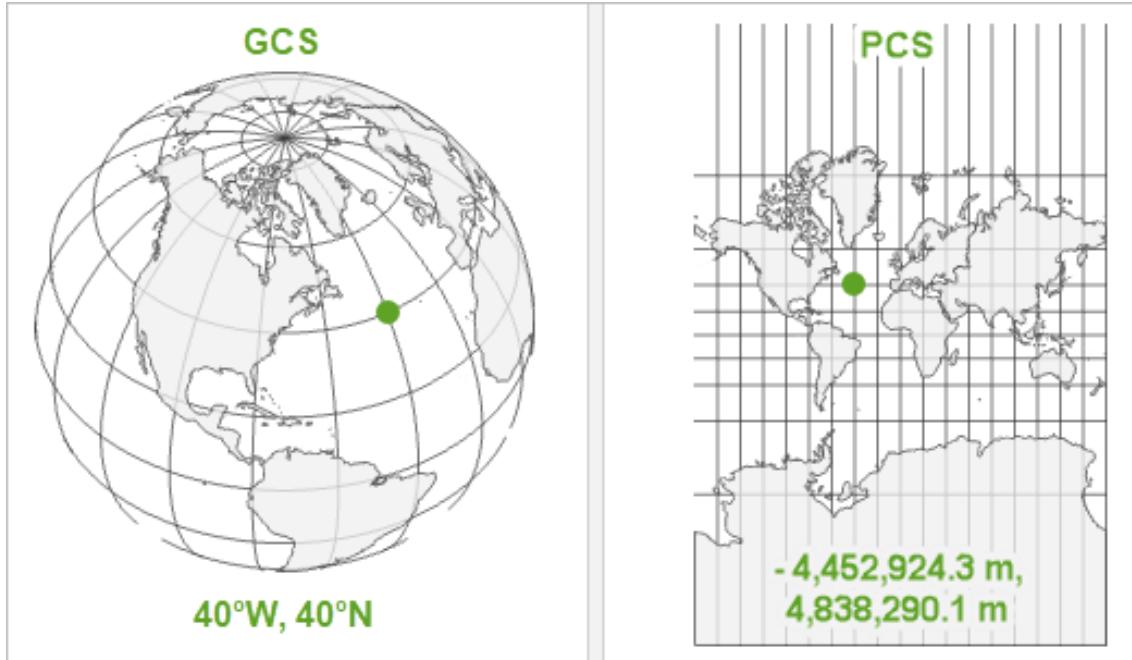


Figure 10.1.: Source: ESRI

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)"],
        MEMBER["World Geodetic System 1984 (G2296)"],
        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 [EPSG: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`.

## 10.2. 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)
```

## 10. Introduction to spatial data

```
[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.

### 10.3. 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"
```

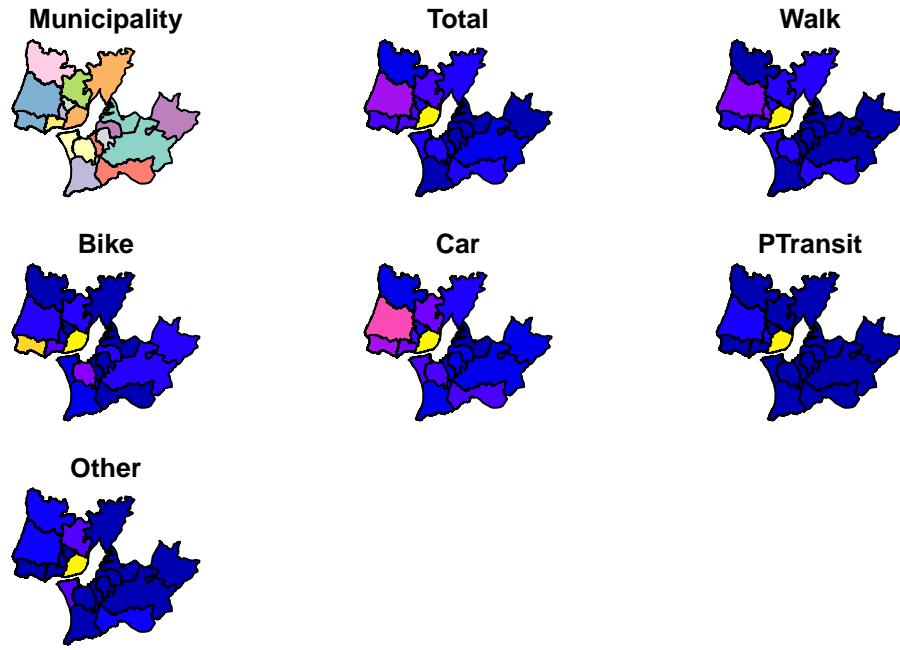
We can also set the `crs` of the spatial feature on the fly.

Check the differences between both data variables.

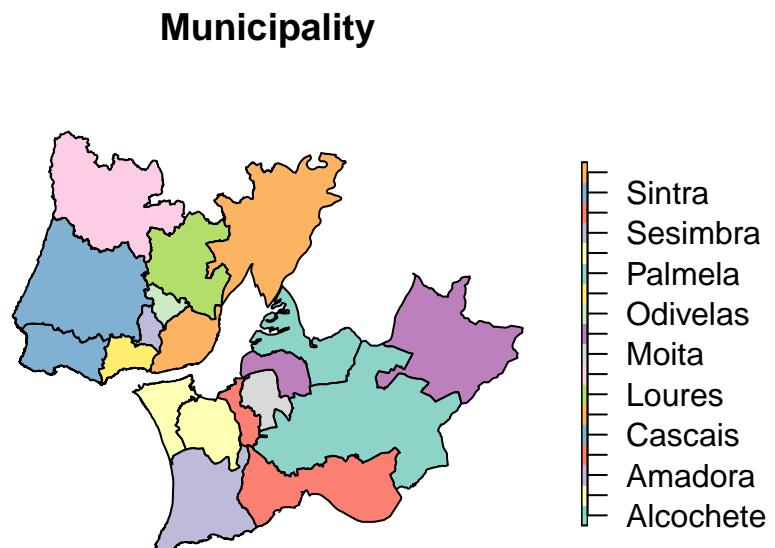
## 10.4. Visualize spatial data

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

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

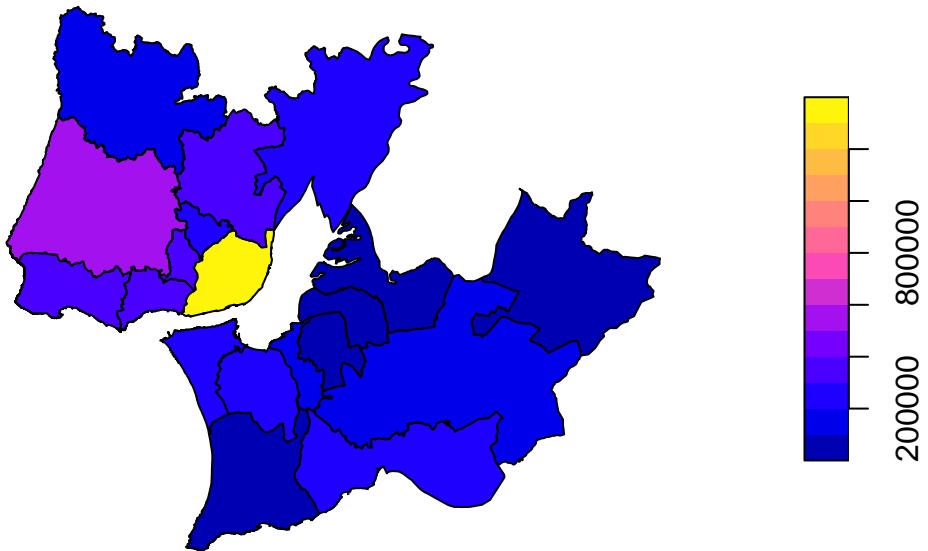


```
plot(TRIPSgeo["Municipality"])
```



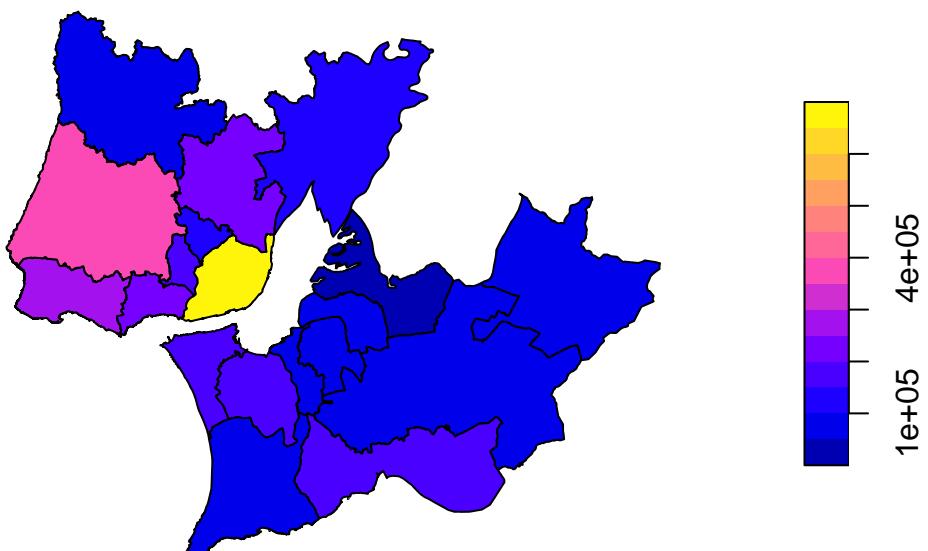
```
plot(TRIPSgeo["Total"])
```

**Total**

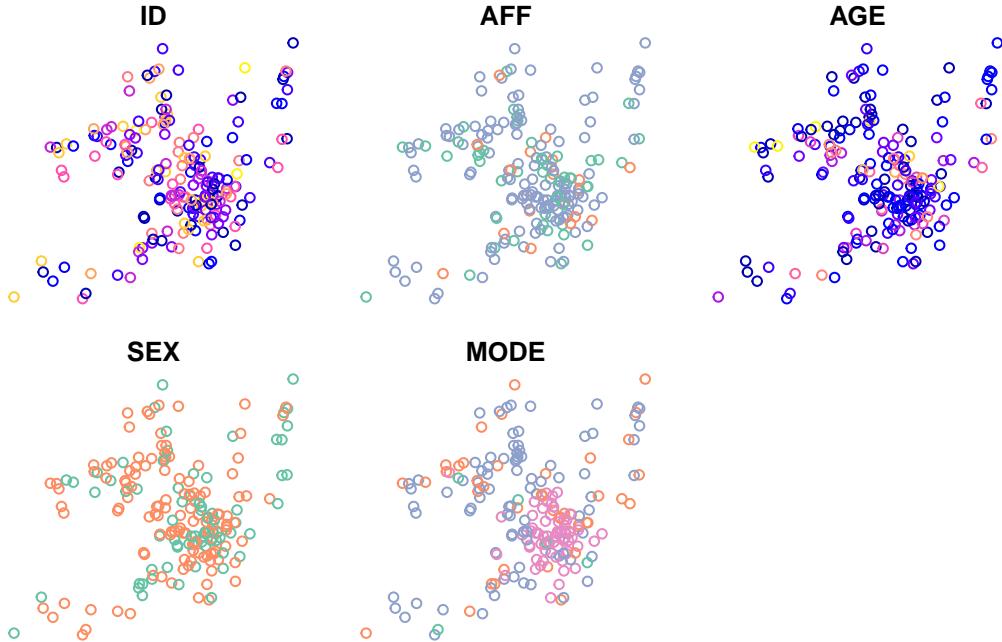


```
plot(TRIPSgeo["Car"])
```

**Car**



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



**i** In the next chapter we will learn how to create interactive maps.

## 10.5. 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 WKT geometry
```

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



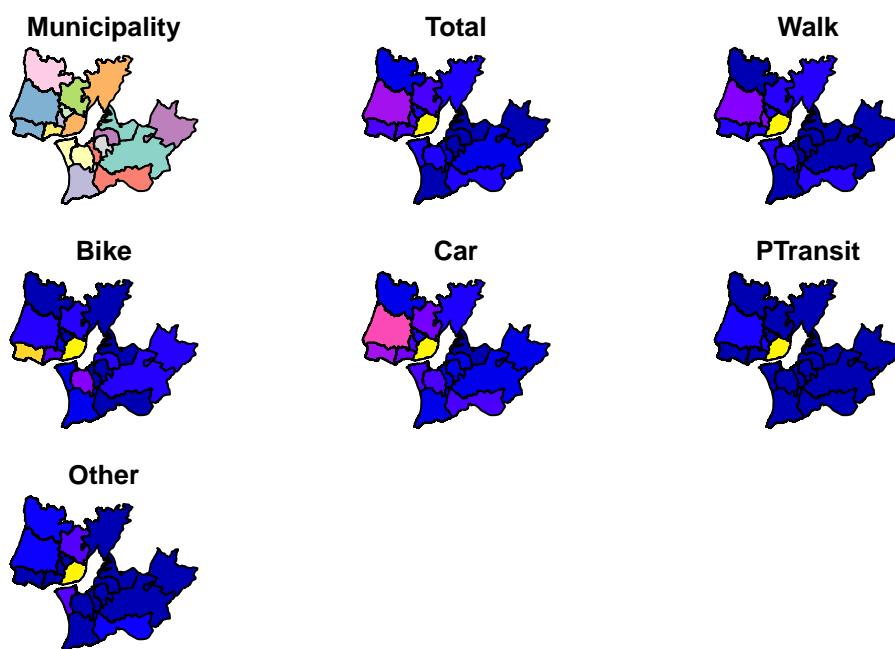
# 11. Interactive maps

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

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

```
Reading layer `TRIPSgeo' from data source
`/media/rosa/Dados/GIS/MQAT/data/TRIPSgeo.gpkg' using driver `GPKG'
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) # all variables
```



```
# one variable at a time
plot(TRIPSgeo["Municipality"])

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

## 11. Interactive maps

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

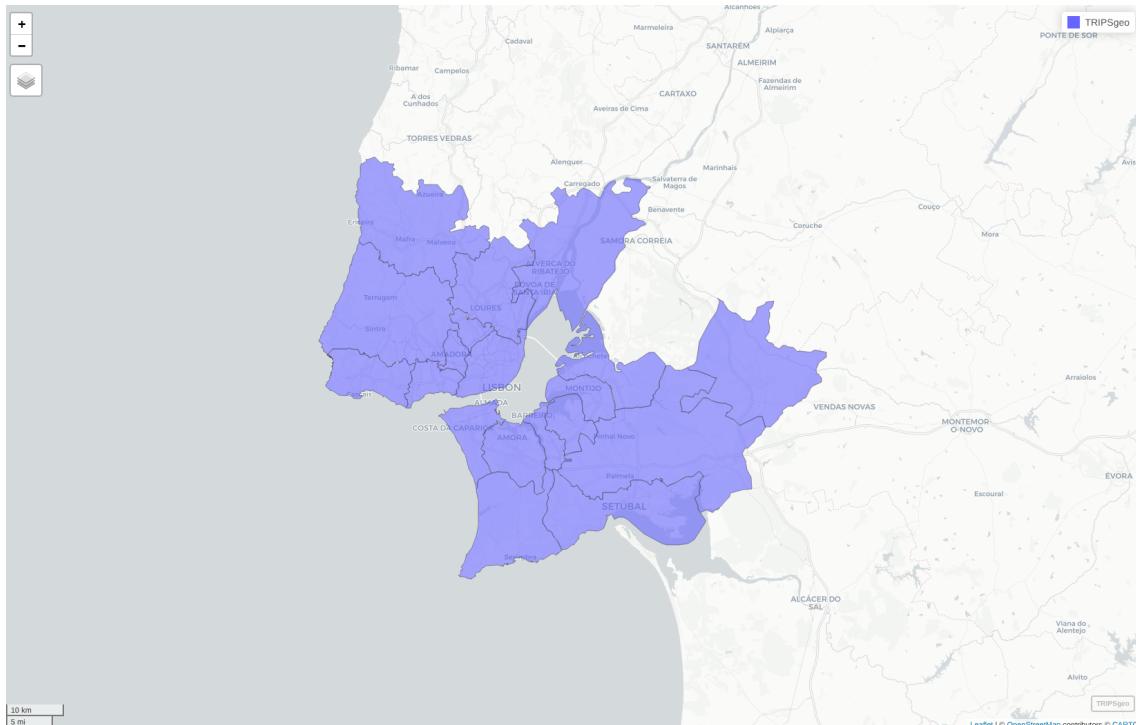
Interactive maps are useful to explore the data, as you can zoom in and out, and click on the points to see the data associated with them.

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

### 11.1. Mapview

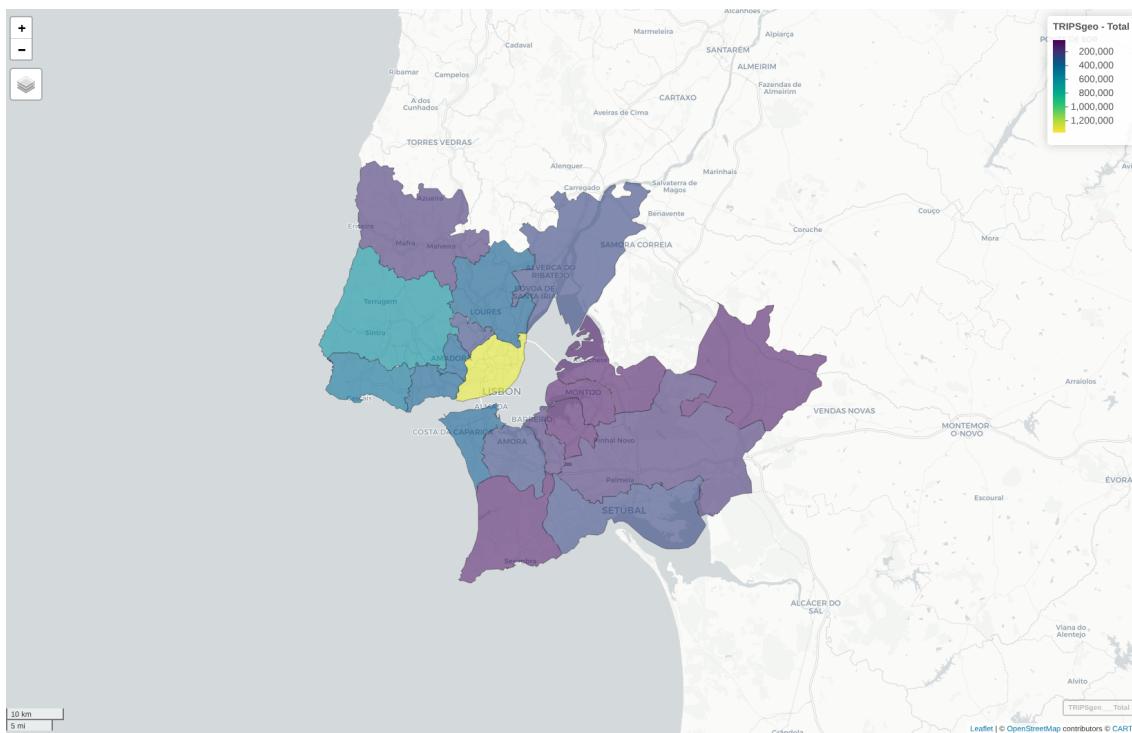
Mapview allows to create quick interactive maps, only by declaring the function `mapview()`.

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



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

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



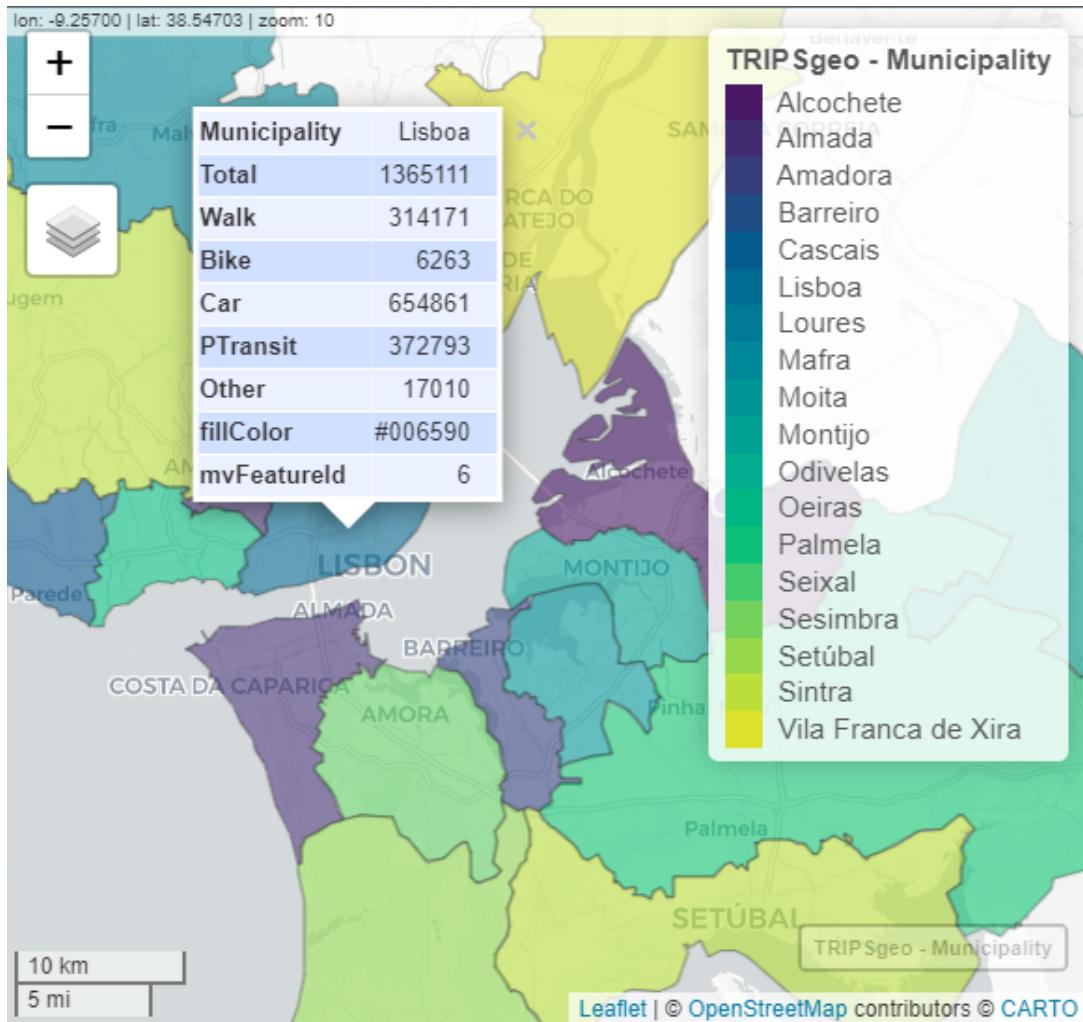
As you can see, a color palette is automatically assigned to the **continuous variable**.

Try to use a **categorical variable**.

```
mapview(TRIPSGeo,
        zcol = "Municipality", # depending on the variable
        alpha.regions = 0.4, # also add transparency
        color = "white" # border color
      )
```

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

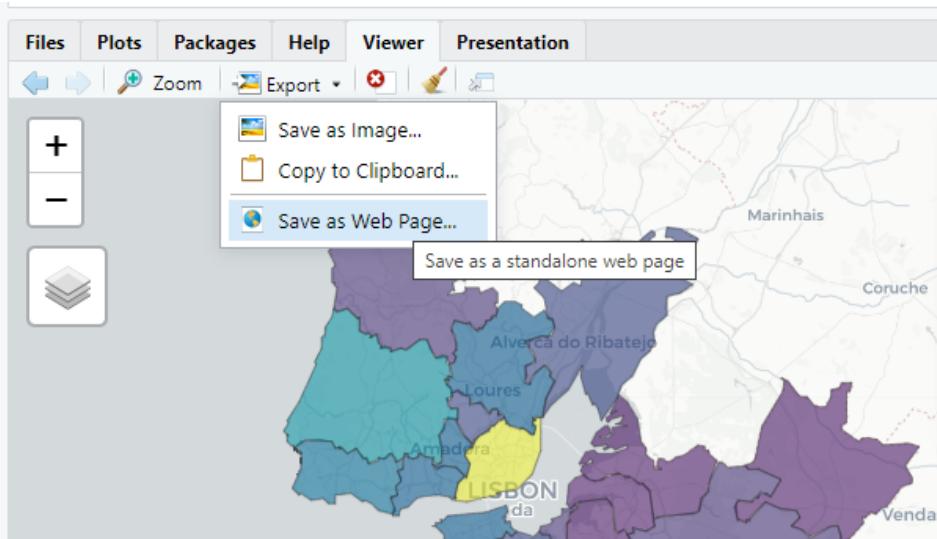
## 11. Interactive maps



You can go crazy with all the options that `mapview` offers. Please refer to the [documentation](#) to see all the options.

### 11.1.1. 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

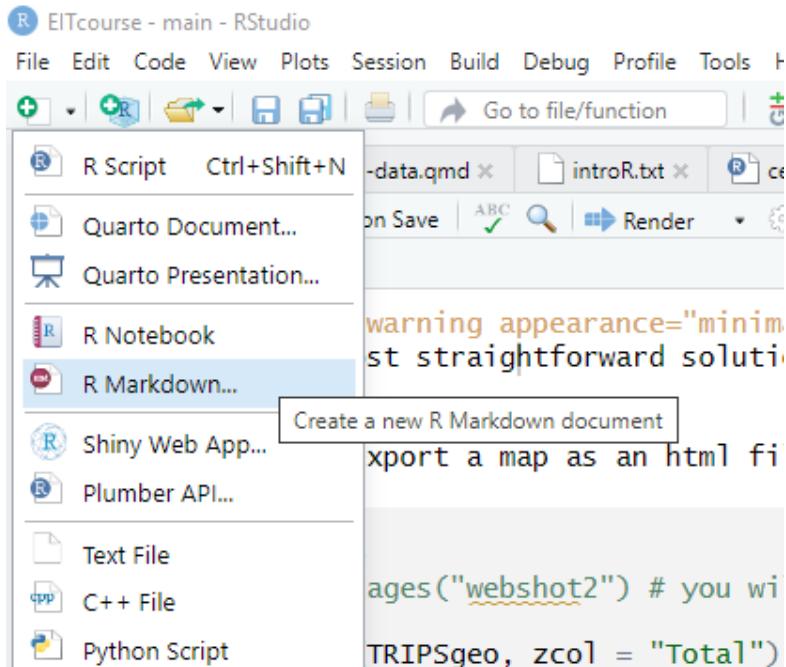
map = mapview(TRIPSgeo, zcol = "Total") # first create an objet with the
# desired map

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

## 11.2. Rmarkdown

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

## 11. Interactive maps



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!

### 11.3. Flowmap blue

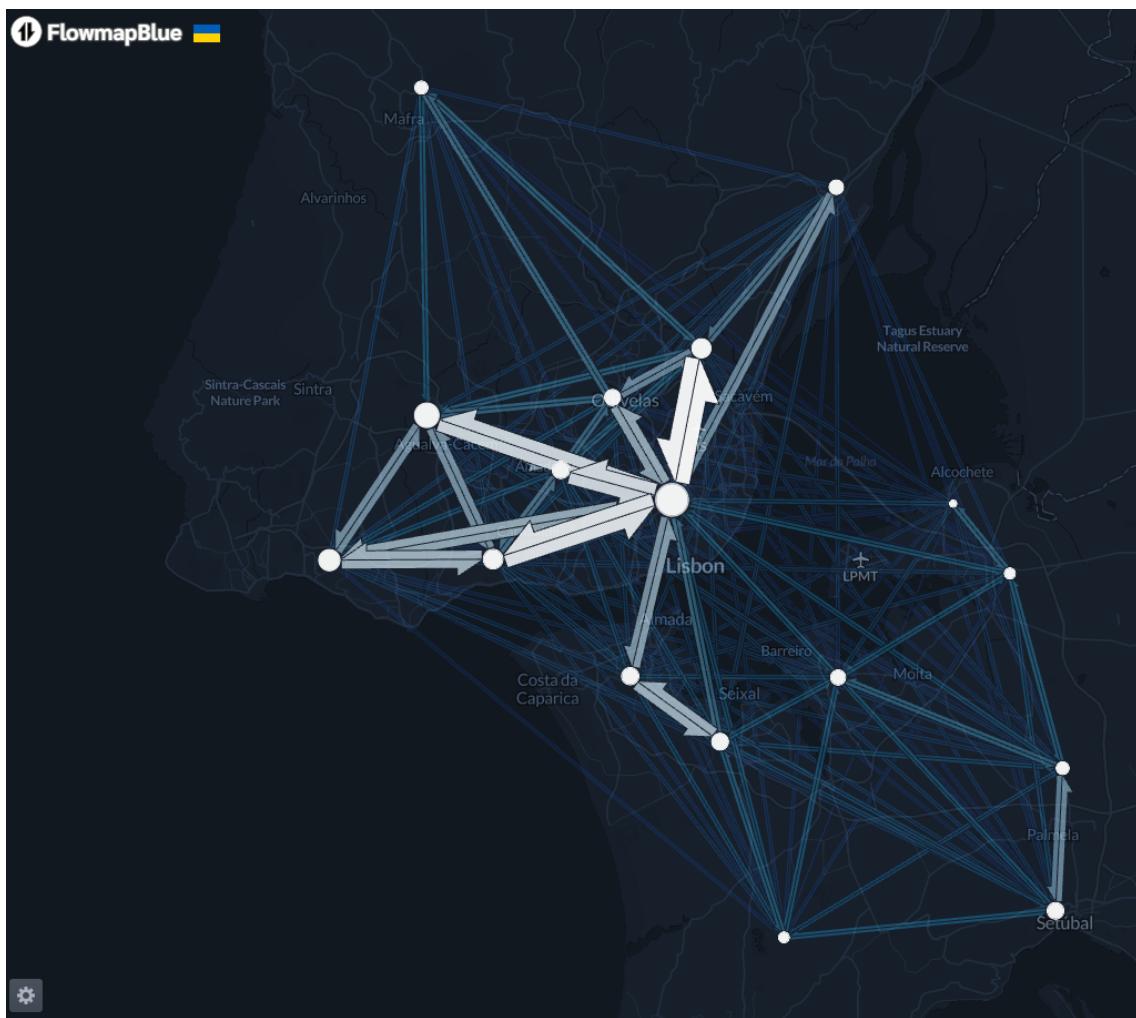
A way to visualize traffic or travel volumes between areas (OD pairs) is through the online tool [flowmap.blue](#).

But you need to prepare the data so that they are exactly in the requested format. (See the used [source code](#)).

You need two files:

1. [locations](#)
2. [flows](#)

Try to copy and paste the content of these files in the corresponding fields the information about **car** trips in the Lisbon metropolitan area according to INE (2018).



💡 Try it yourself

Visualize the **public transit** trips in the Lisbon metropolitan area. ([flows\\_tp](#))

- Which differences do you see between car and public transit trips?
- How many trips are made daily in public transit with origin or destination in Sintra?



# 12. Centroids of transport zones

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

## 12.1. 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)

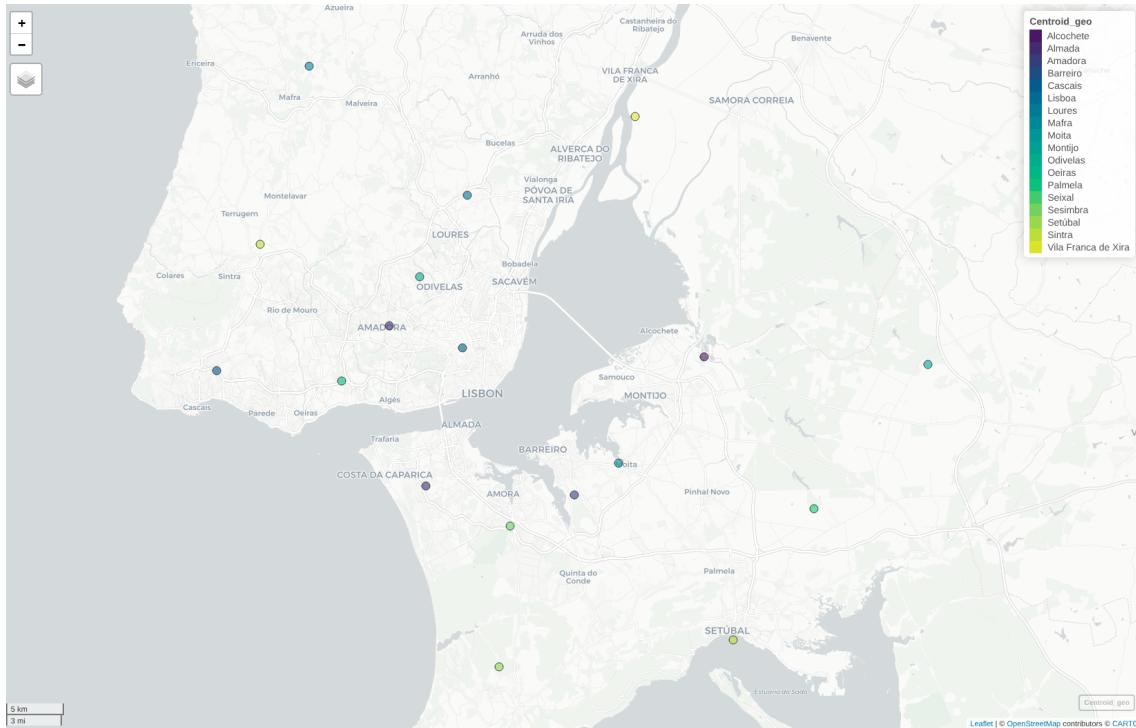
Municipalities_geo = st_read("../data/Municipalities_geo.gpkg", quiet =
  TRUE)

Centroid_geo = st_centroid(Municipalities_geo)
```

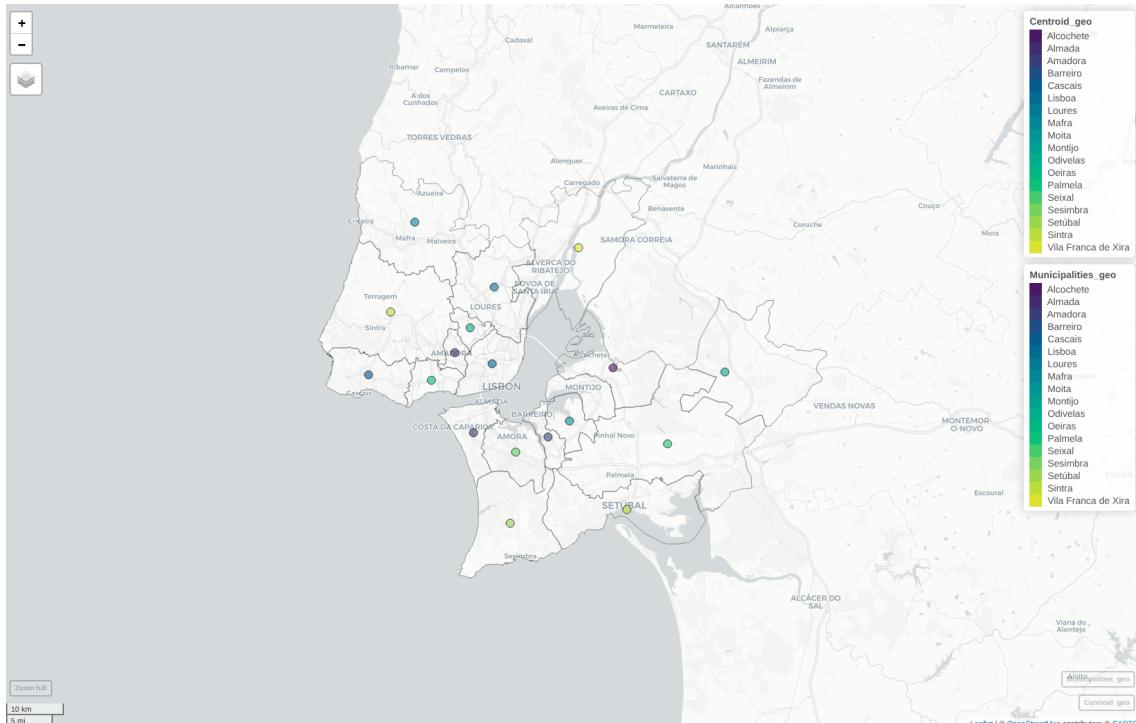
This creates points at the geometric center of each polygon.

```
mapview(Centroid_geo)
```

## 12. Centroids of transport zones



```
mapview(Centroid_geo) + mapview(Municipalities_geo, alpha.regions = 0) #  
↪ both maps, with full transparency in polygons
```



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.

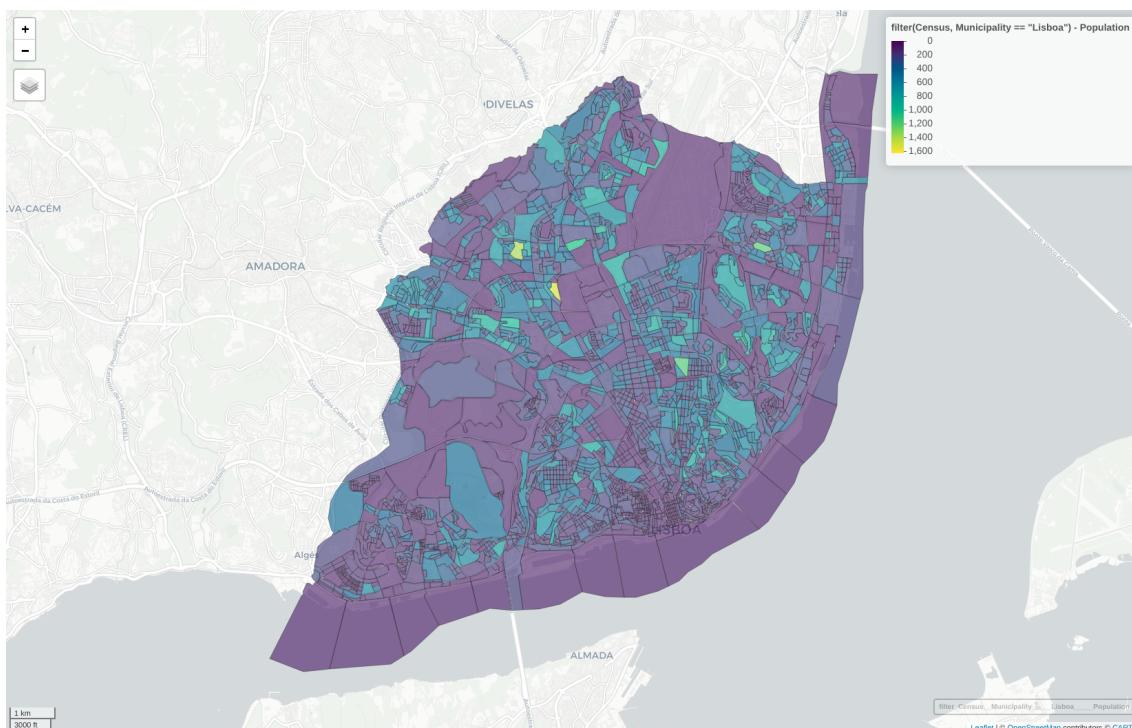
To overcome this, we can use **weighted centroids**.

## 12.2. 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.

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

We can do the same for buildings.

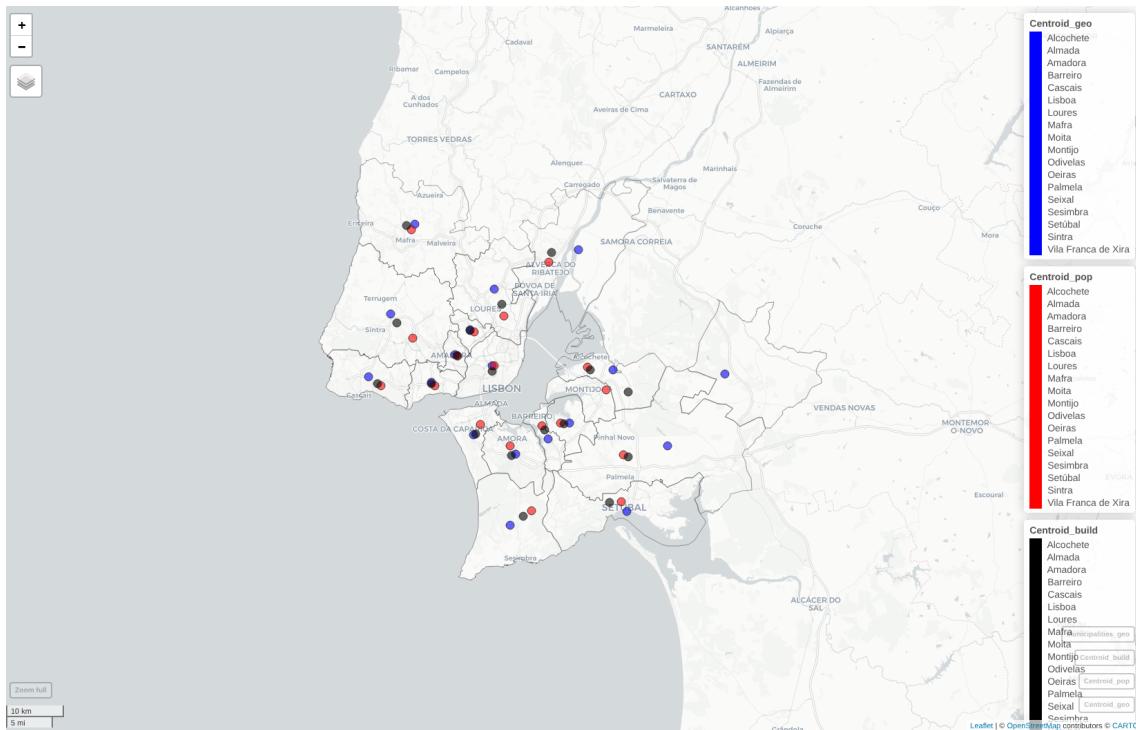
## 12. Centroids of transport zones

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

### 12.3. Compare centroids in a map

#### 12.3.1. Interactive map

```
mapview(Centroid_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.

#### 12.3.2. 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(Centroid_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(Centroid_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)
```

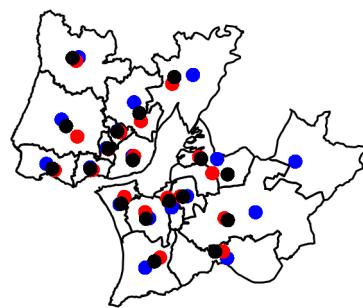


Figure 12.1.: Static map of different centroids of Municipalities

In the [next section](#) we will use these centroids to calculate the desire lines between them.

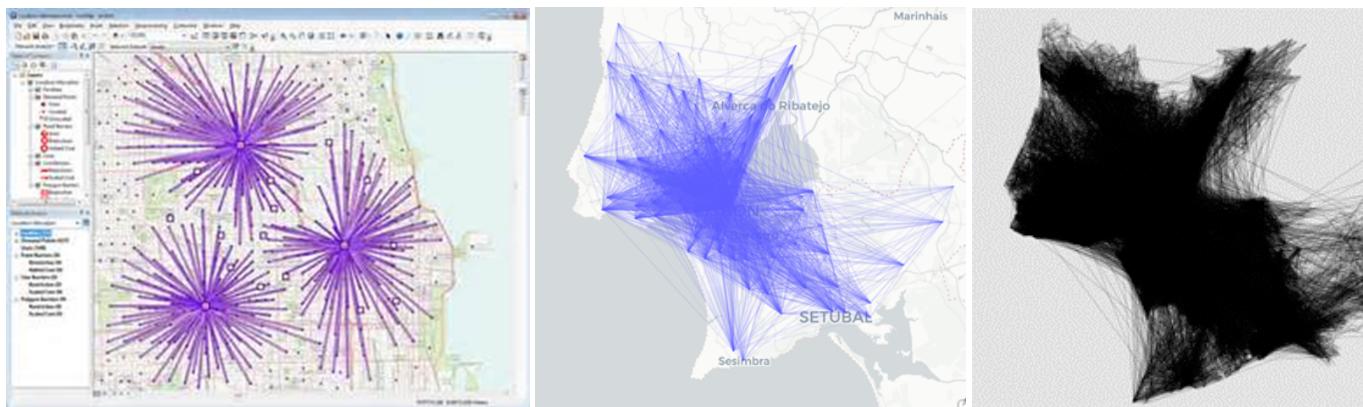


## 13. OD pairs and desire lines

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 destination
  - students' homes 1 school
- origin zone to destination zone
  - for aggregated trips (macro representation)
- OD jittering<sup>1</sup>



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).

---

<sup>1</sup>See (Lovelace, Félix, and Carlino 2022).

### 13.1. Desire lines with od\_to\_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) # suppress message
```

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")
```

## 13. OD pairs and desire lines



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

### 13.1.1. Filtering desire lines

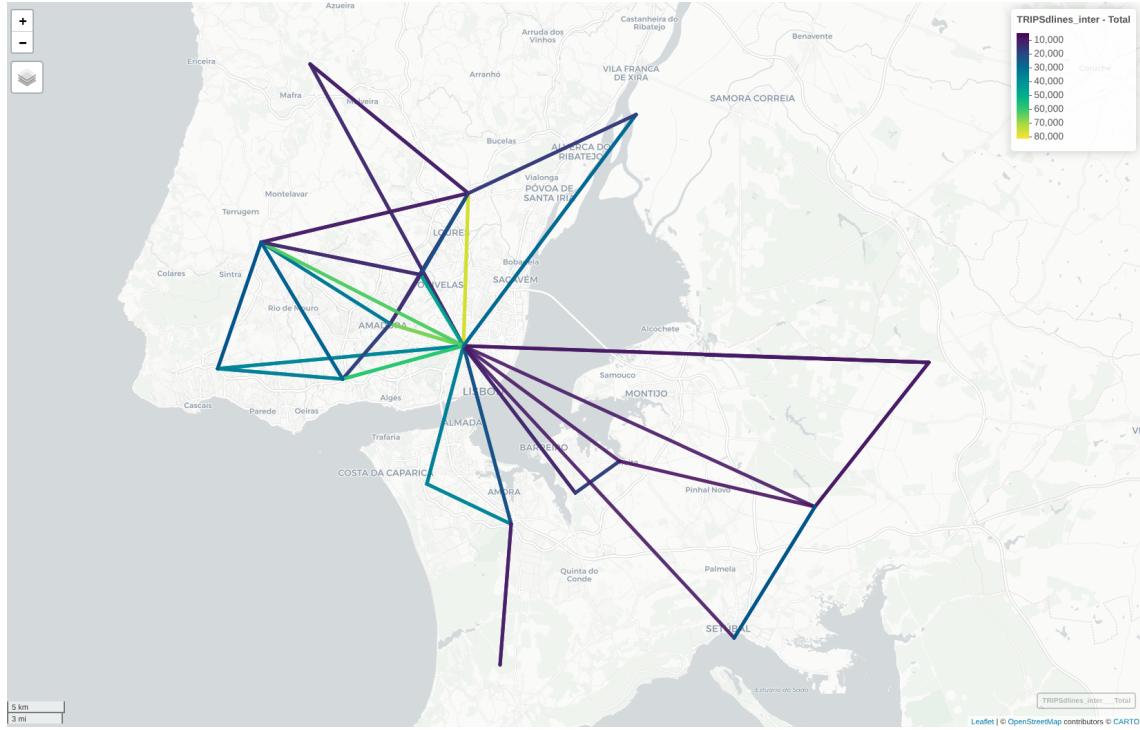
Filter **intrazonal** trips.

```
library(dplyr)

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

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

### 13. OD pairs and desire lines



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.

## 13.2. 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 overlapping.

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) |>
  filter(o != d) # remove empty geometries
nrow(TRIPSdlines_oneway)
```

```
[1] 150
```

### 13. OD pairs and desire lines

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)
Geometry type: LINESTRING
Dimension: XY
Bounding box: xmin: -9.229502 ymin: 38.62842 xmax: -8.915878 ymax: 38.75981
Geodetic CRS: WGS 84
      o          d           geometry
1 Alcochete   Almada LINESTRING (-8.915878 38.73...
2 Alcochete   Amadora LINESTRING (-8.915878 38.73...
3   Almada   Amadora LINESTRING (-9.193061 38.63...
4 Alcochete Barreiro LINESTRING (-8.915878 38.73...
5   Almada Barreiro LINESTRING (-9.193061 38.63...
6   Amadora Barreiro LINESTRING (-9.229502 38.75...
```

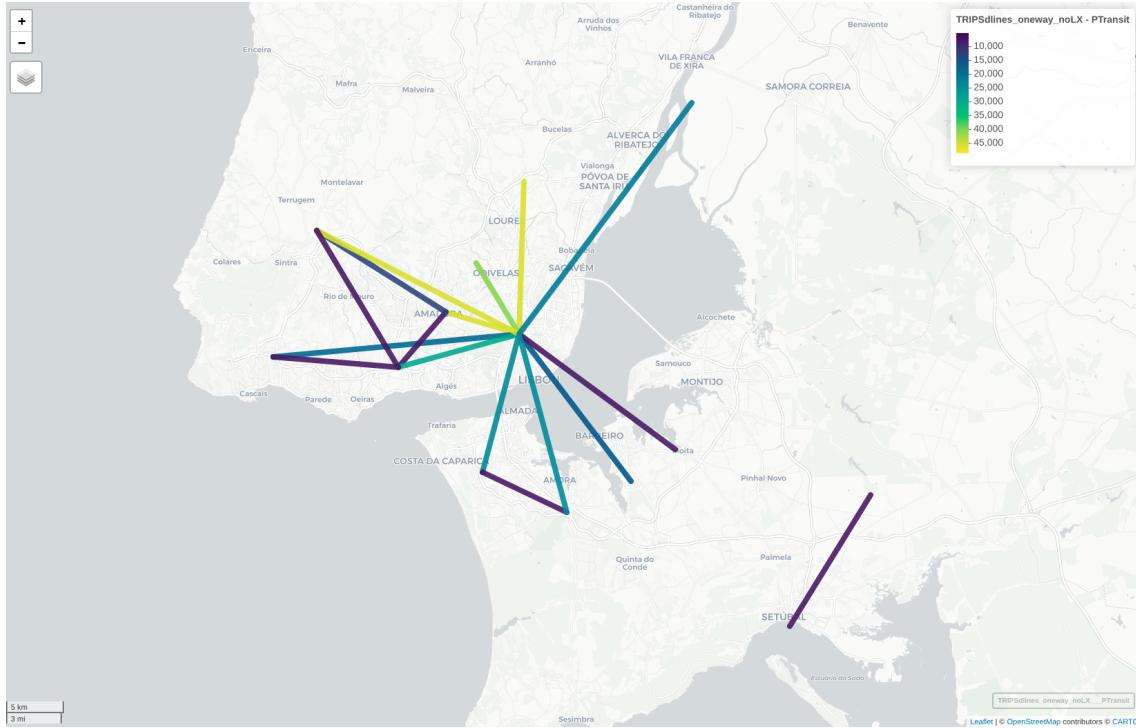
```
tail(TRIPSdlines_oneway[,c(1,2)])
```

```
Simple feature collection with 6 features and 2 fields
Attribute-geometry relationships: identity (2)
Geometry type: LINESTRING
Dimension: XY
Bounding box: xmin: -9.357651 ymin: 38.49491 xmax: -8.80664 ymax: 38.92208
Geodetic CRS: WGS 84
      o          d           geometry
145  Oeiras Vila Franca de Xira LINESTRING (-9.276317 38.71...
146  Palmela Vila Franca de Xira LINESTRING (-8.80664 38.617...
147  Seixal Vila Franca de Xira LINESTRING (-9.108785 38.60...
148 Sesimbra Vila Franca de Xira LINESTRING (-9.120129 38.49...
149 Setúbal Vila Franca de Xira LINESTRING (-8.887489 38.51...
150 Sintra Vila Franca de Xira LINESTRING (-9.357651 38.82...
```

Example of visualization with Public Transit trips in both ways.

```
TRIPSdlines_oneway_noLX = TRIPSdlines_oneway |>
  filter(PTransit > 5000) # reduce noise

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



### 13.3. 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")

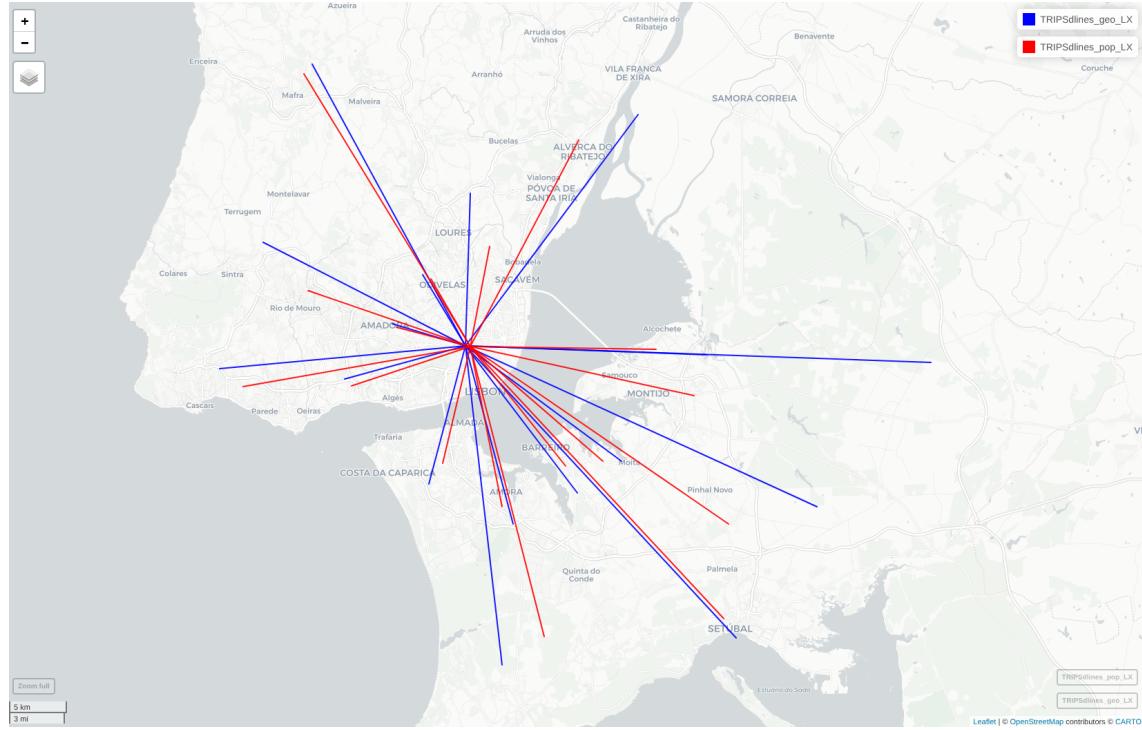
TRIPSdlines_pop = od_to_sf(TRIPSmode, z = Centroid_pop) |> # works the
  ↵ same way
  od_oneway() |> # oneway
  filter(o != d) # remove empty geometries
```

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")
```

### 13. OD pairs and desire lines



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

# 14. Euclidean and routing distances

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

## 14.1. Euclidean distances

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

### 14.1.1. 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
```

### 14.1.2. Create new point at the university

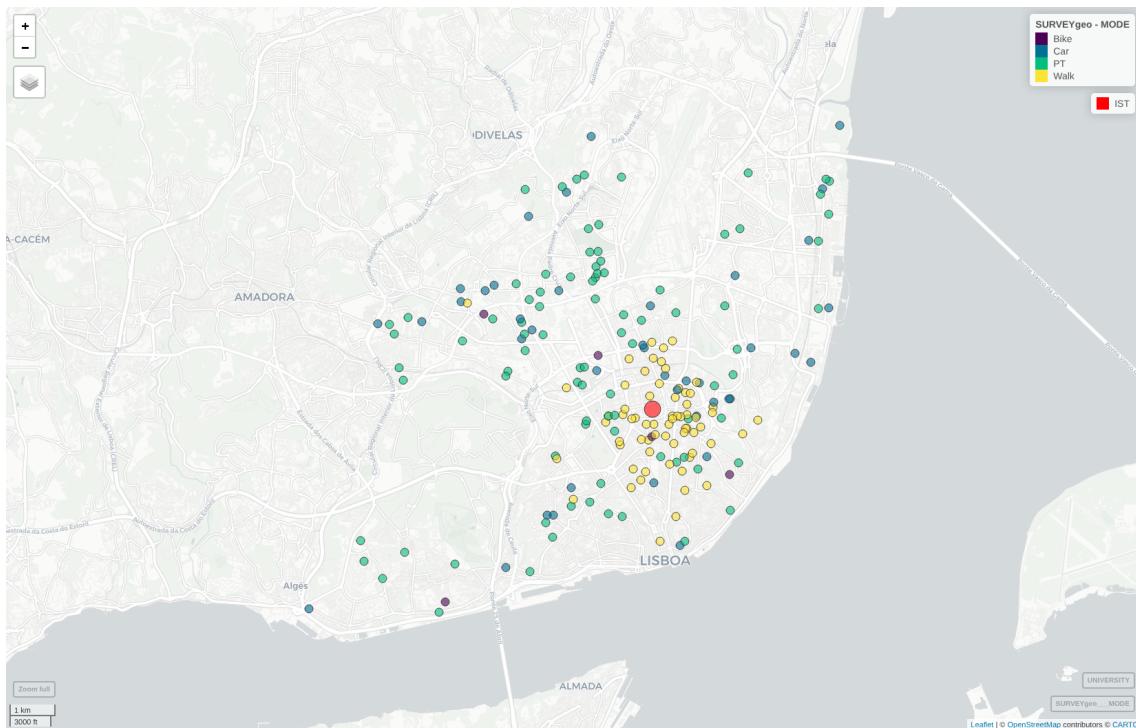
Using coordinates from Instituto Superior Técnico, we can directly create a simple feature and assign its crs.

## 14. Euclidean and routing distances

```
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)
```



### 14.1.3. Straight lines

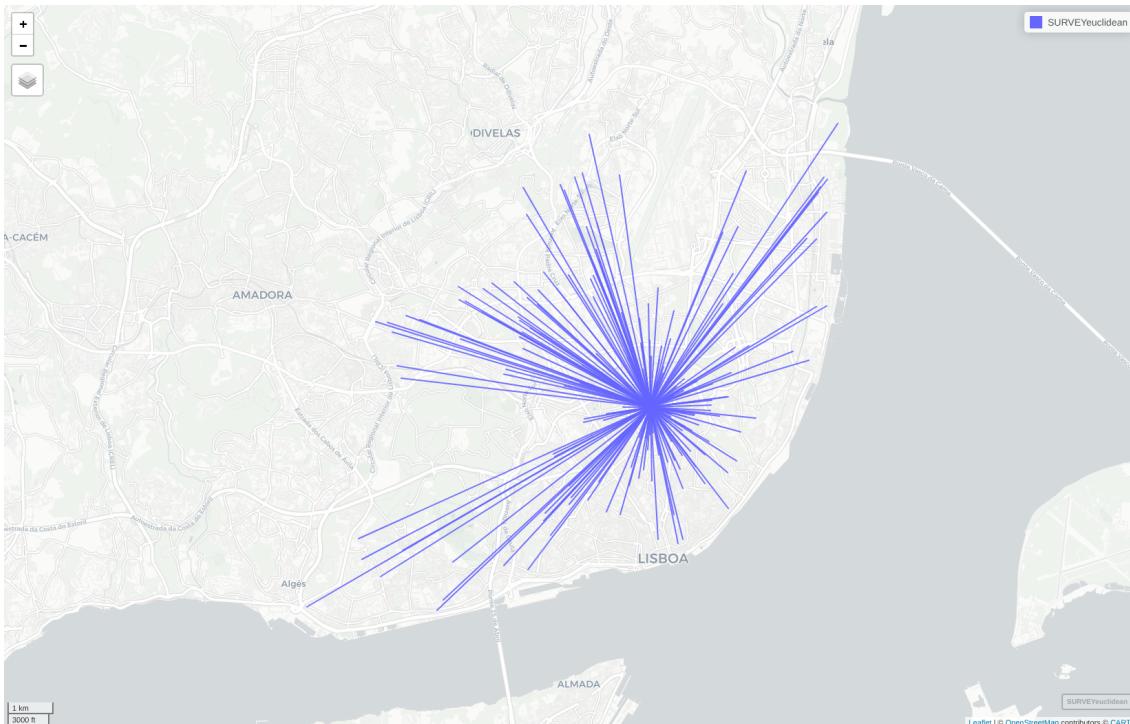
First we will create lines connecting the survey locations to the university, using the `st_nearest_points()` function.

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)
```



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.

#### 14.1.4. 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.

## 14. Euclidean and routing distances

```
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.

## 14.2. 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 14.1.: 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](#).

### 14.3. Routing distances with openrouteservice

We will use the `openrouteservice` r package to estimate the distances using a road network (Oleś 2025).

**i** To properly use the `openrouteservice` package, you need to setup the ORS provider. See the [setup instructions](#) for more details.

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, so we can [compare](#) them later.

Also, we need to extract the coordinates, from both datasets, to be used in the routing functions of `openrouteservice::directions()`.

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

UNIVERSITY = UNIVERSITY |>
  mutate(id = 1) |> # only one row
  mutate(coordinates = st_coordinates(geometry)) # extract coordinates
```

#### 14.3.1. Distances by car

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

**⚠** This one is not that easy to set-up because the function is prepared to retrieve only one result per request :( (So we do a loop function.

## 14. Euclidean and routing distances

```
library(openrouteservice)

SURVEYcar = data.frame() # initial empty data frame

# loop - the origin (i) is the survey location, and the UNIVERSITY is
# always the same destination
for (i in 1:nrow(SURVEYsample)) {
  ROUTES1 = ors_directions(
    data.frame(
      lon = c(SURVEYsample$coordinates[i, 1], UNIVERSITY$coordinates[1,
        1]),
      lat = c(SURVEYsample$coordinates[i, 2], UNIVERSITY$coordinates[1,
        2]))
  ),
  profile = "driving-car", # or cycling-regular foot-walking
  preference = "fastest", # or shortest
  output = "sf"
)
  ROUTES1$distance = ROUTES1$summary[[1]]$distance # extract these values
# from summary
  ROUTES1$duration = ROUTES1$summary[[1]]$duration

  SURVEYcar = rbind(SURVEYcar, ROUTES1) # to keep adding in the same df
}

SURVEYcar = SURVEYcar |>
  select(distance, duration, geometry) |> # discard unnecessary variables
  mutate(ID = SURVEYsample$ID) # cbind with survey ID
```

### 14.3.2. Distances by foot

Repeat the same for foot-walking.

```
SURVEYwalk = data.frame() # initial empty data frame

# loop - the origin (i) is the survey location, and the UNIVERSITY is
# always the same destination
for (i in 1:nrow(SURVEYsample)) {
  ROUTES1 = ors_directions(
    data.frame(
      lon = c(SURVEYsample$coordinates[i, 1], UNIVERSITY$coordinates[1,
        1]),
      lat = c(SURVEYsample$coordinates[i, 2], UNIVERSITY$coordinates[1,
        2]))
  ),
  profile = "foot-walking", # or driving-car cycling-regular
# cycling-electric
```

```

preference = "fastest", # or shortest
output = "sf"
)
ROUTES1$distance = ROUTES1$summary[[1]]$distance # extract these values
← from summary
ROUTES1$duration = ROUTES1$summary[[1]]$duration

SURVEYwalk = rbind(SURVEYwalk, ROUTES1) # to keep adding in the same df
}

SURVEYwalk = SURVEYwalk |>
  select(distance, duration, geometry) |> # discard unnecessary variables
  mutate(ID = SURVEYsample$ID) # cbind with survey ID

```

### 14.3.3. Distances by bike

And for cycling-regular.

```

SURVEYbike = data.frame() # initial empty data frame

# loop - the origin (i) is the survey location, and the UNIVERSITY is
← always the same destination
for (i in 1:nrow(SURVEYsample)) {
  ROUTES1 = ors_directions(
    data.frame(
      lon = c(SURVEYsample$coordinates[i, 1], UNIVERSITY$coordinates[1,
        ← 1]),
      lat = c(SURVEYsample$coordinates[i, 2], UNIVERSITY$coordinates[1,
        ← 2])
    ),
    profile = "cycling-regular",
    preference = "fastest", # or shortest
    output = "sf"
  )
  ROUTES1$distance = ROUTES1$summary[[1]]$distance # extract these values
  ← from summary
  ROUTES1$duration = ROUTES1$summary[[1]]$duration

  SURVEYbike = rbind(SURVEYbike, ROUTES1) # to keep adding in the same df
}

SURVEYbike = SURVEYbike |>
  select(distance, duration, geometry) |> # discard unnecessary variables
  mutate(ID = SURVEYsample$ID) # cbind with survey ID

```

## 14. Euclidean and routing distances

```
names(SURVEYcar)
```

```
[1] "distance" "duration" "ID"      "geometry"
```

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

### 14.4. 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.
467.3	1009.0	1447.3	1471.8	1953.5	2769.4

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

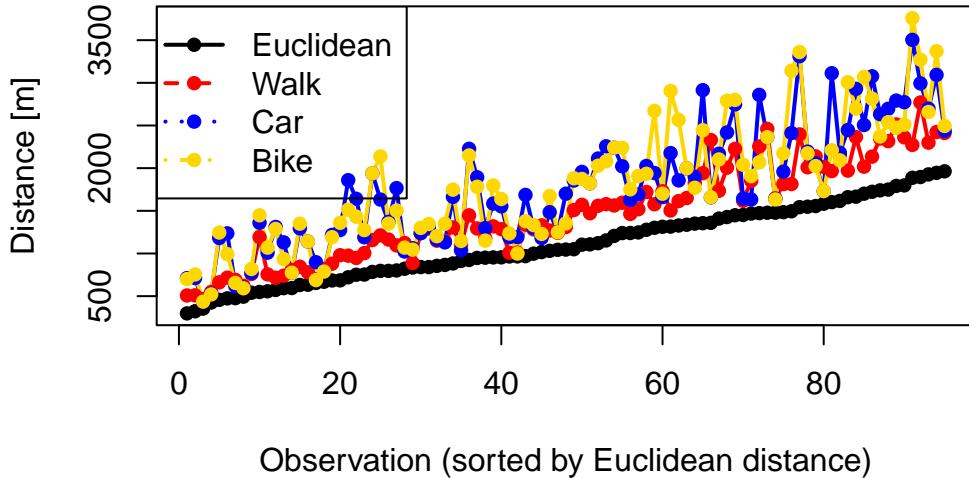
Min.	1st Qu.	Median	Mean	3rd Qu.	Max.
435.4	1227.0	1686.7	1771.4	2210.3	3503.2

```
summary(SURVEYbike$distance) # Bike
```

Min.	1st Qu.	Median	Mean	3rd Qu.	Max.
436	1247	1752	1791	2225	3758

What can you understand from this results?

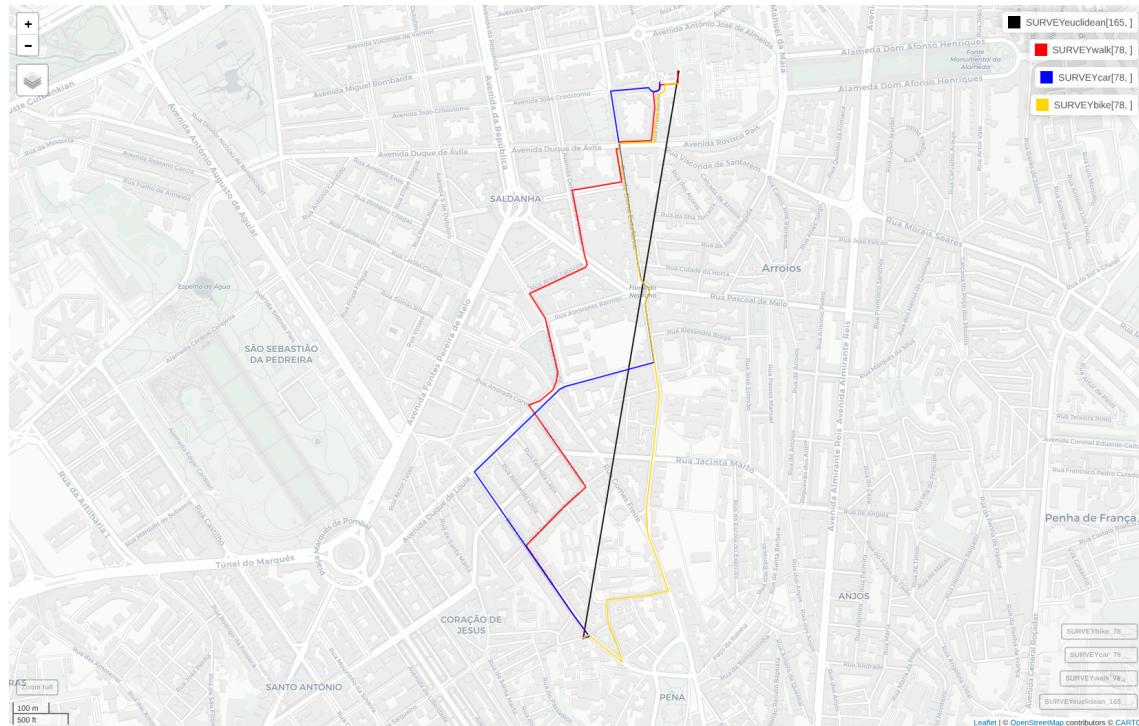
## Distances by Euclidean, Walk, Car, and Bike



### 14.4.1. Circuitry

Compare 1 single route.

```
mapview(SURVEYeuclidean[165,], color = "black") + # 1556 meters
mapview(SURVEYwalk[78,], color = "red") + # 2126 meters
mapview(SURVEYcar[78,], color = "blue") + # 2058 meters
mapview(SURVEYbike[78,], color = "gold") # 2025 meters
```



## 14. Euclidean and routing distances

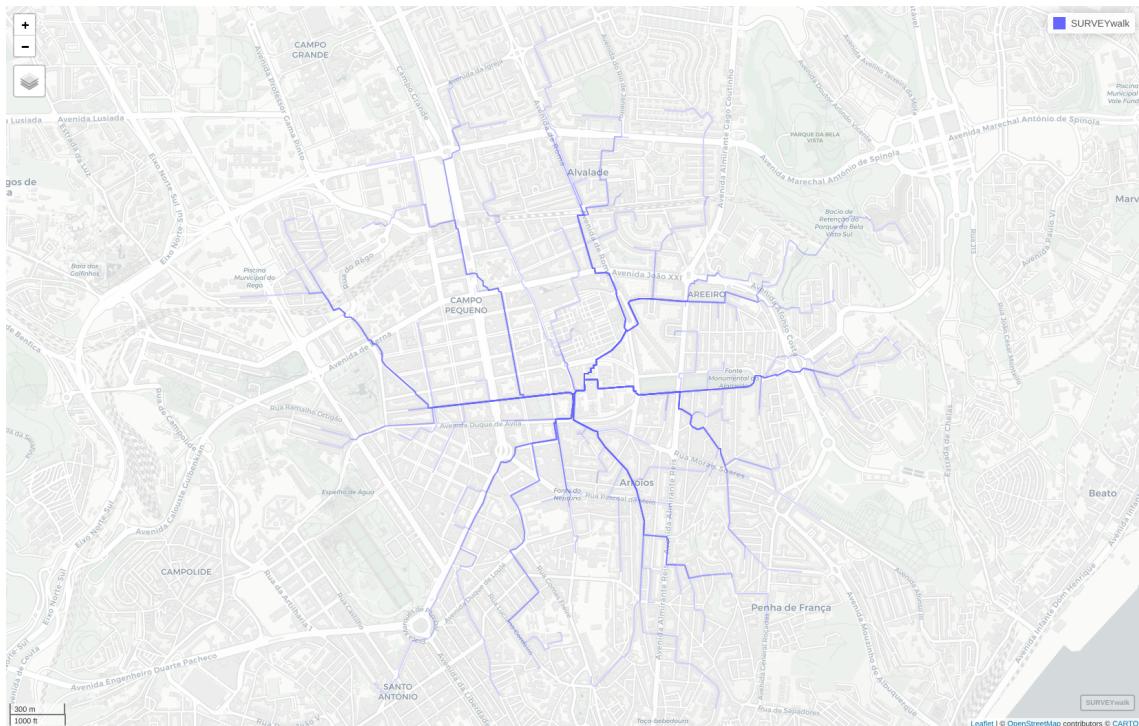
With this we can see the **circuitry** of the routes, a measure of route / transportation efficiency, which is the ratio between the routing distance and the euclidean distance.

The circuity for car (1.32) is usually lower than for walking (1.37) or biking, for longer distances, and higher opposite for shorter distances.

### 14.5. Visualize routes

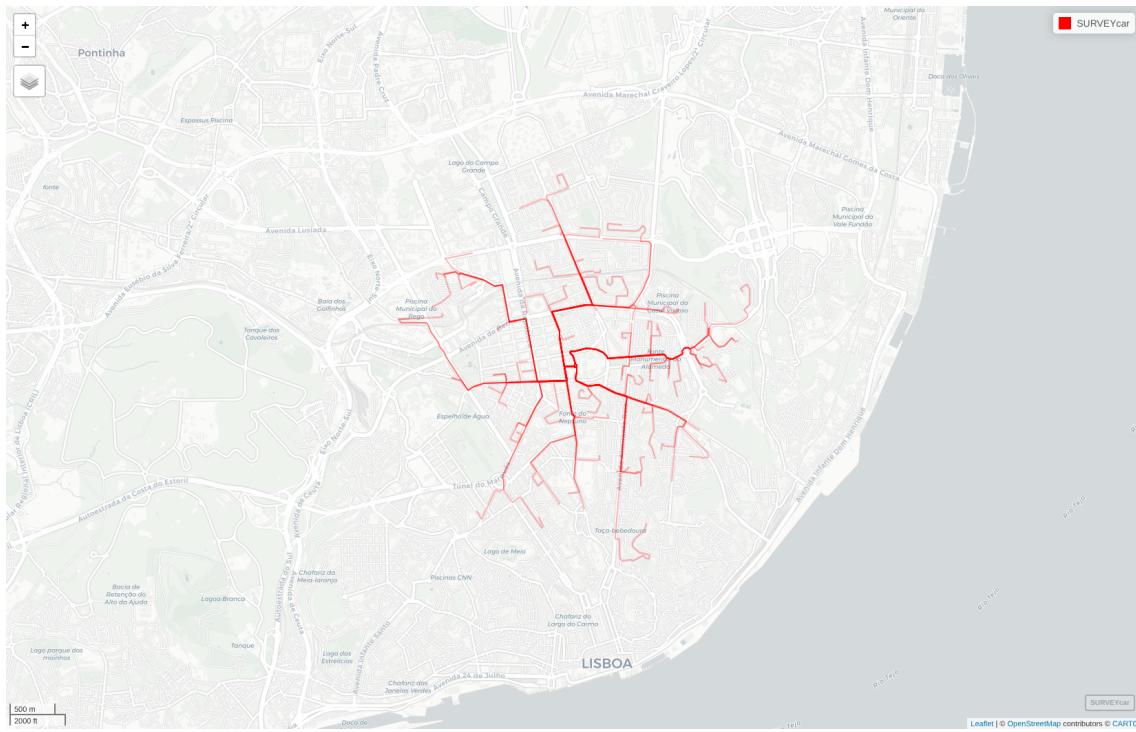
Visualize with transparency of 30%, to get a clue when they overlay.

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

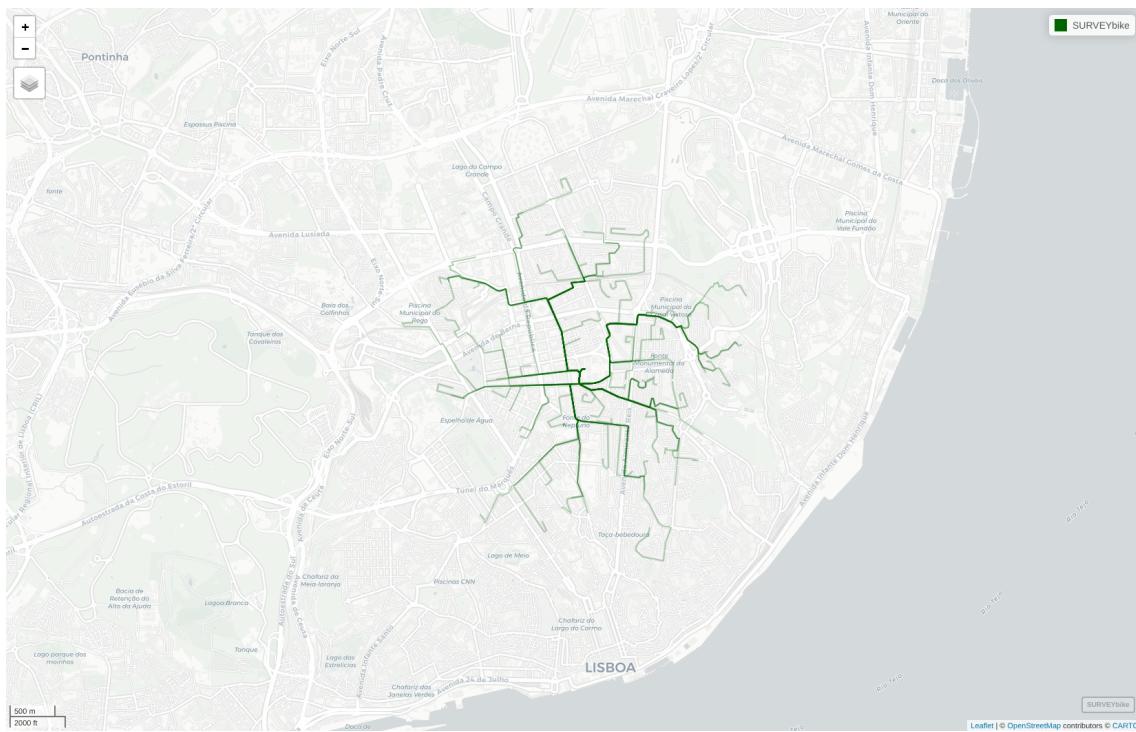


```
mapview(SURVEYcar, alpha = 0.3, color = "red")
```

## 14. Euclidean and routing distances



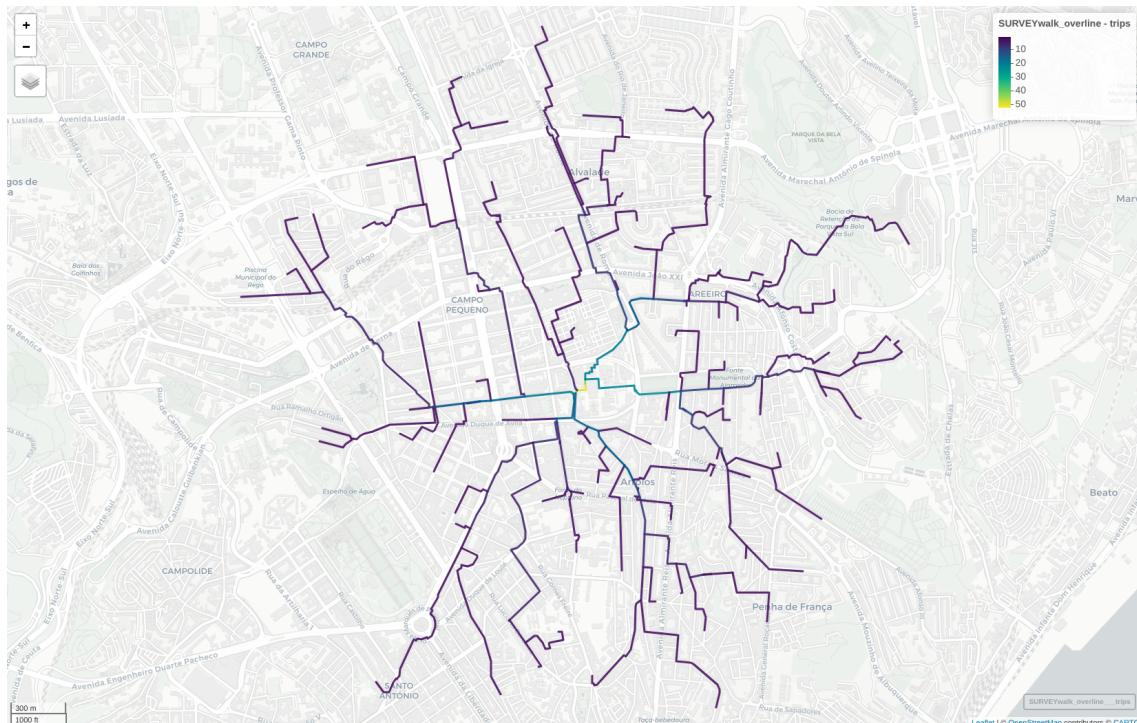
```
mapview(SURVEYbike, alpha = 0.3, color = "darkgreen")
```



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

## 14. Euclidean and routing distances

```
# 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<sup>1</sup>.

### 💡 Questions

- How many people are entering IST by the stairs near *Bar de Civil*?
- And by the North gate?
- And from Alameda stairs?

<sup>1</sup>Assuming all travel by the shortest path.

# 15. Buffers vs. Isochrones

We will use the `stplnar` package to create buffers in meters, and the `openrouteservice` package to create isochrones.

## 15.1. Buffer

Represent a buffer of 500 m and 2000 m from IST<sup>1</sup>.

```
IST = st_sfc(st_point(c(-9.1397404, 38.7370168)), crs = 4326) # create a
← point
IST$coordinates = st_coordinates(IST)

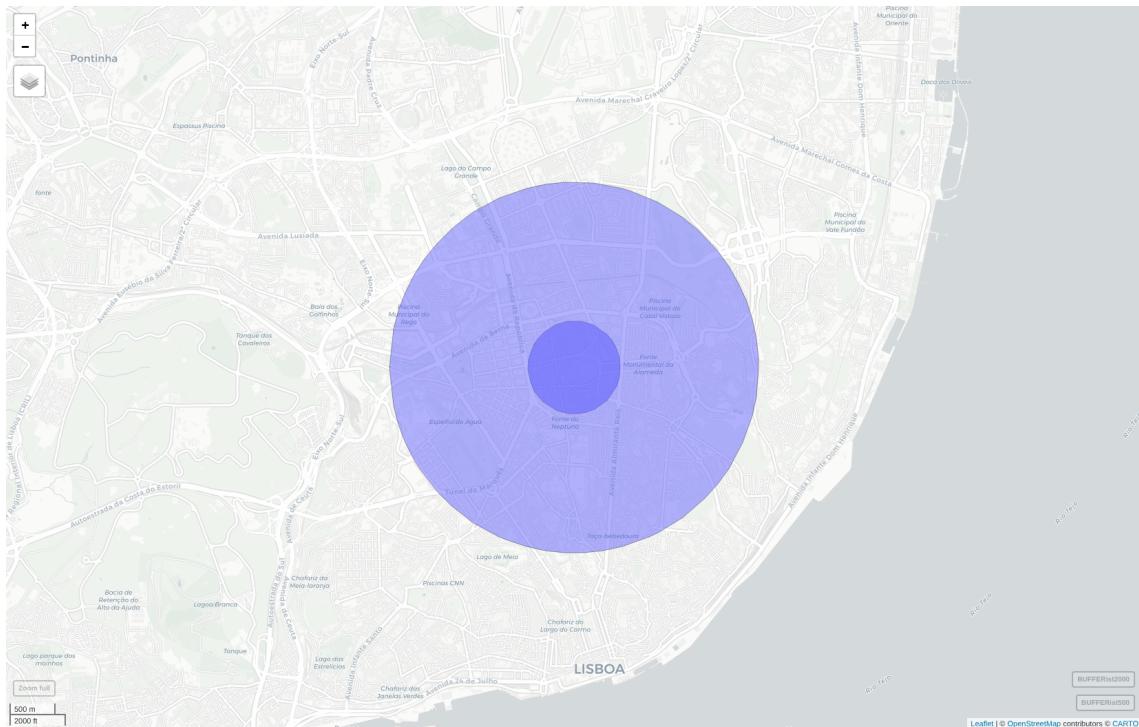
# BUFFERist500 = st_buffer(IST, dist = 500) # non projected - results may
← be weird
BUFFERist500 = geo_buffer(IST[1], dist = 500) # from stplnar, to make sure
← it is in meters.
BUFFERist2000 = geo_buffer(IST[1], dist = 2000)

mapview(BUFFERist500) + mapview(BUFFERist2000, alpha.regions = 0.5)
```

---

<sup>1</sup>Here I selected only the first variable because now we also have the coordinates information (unnecessary for this procedure)

## 15. Buffers vs. Isochones



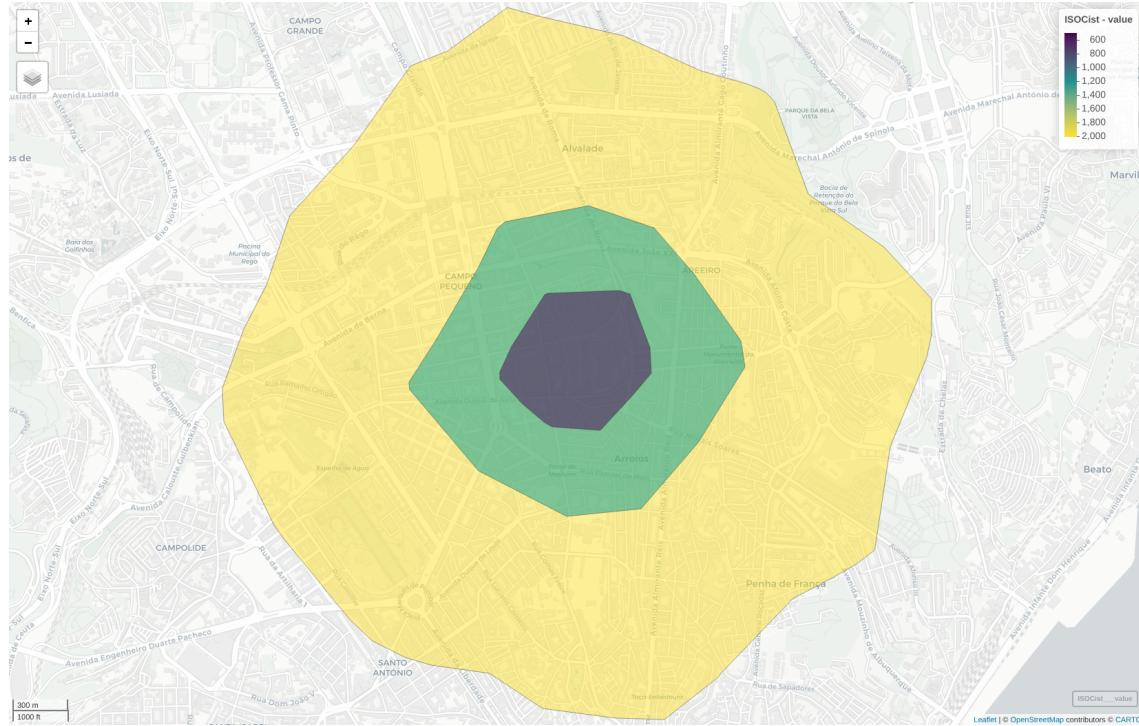
## 15.2. Isochrone

### 15.2.1. Isochrone from 1 point - distance

We use again the `openrouteservice` r package.

```
ISOcist = ors_isochrones(  
  IST$coordinates,  
  profile = "foot-walking",  
  range_type = "distance", # or time  
  range = c(500, 1000, 2000),  
  output = "sf"  
)  
  
ISOcist = arrange(ISOcist, -value) |> # to make the larger polygons on  
# top of the table so they are displayed behind.  
  select(-center) # unnecessary variable
```

```
mapview(ISOcist, zcol = "value", alpha.regions = 0.5)
```



As you can see, the distance buffer of 500m is larger than the isochrone of 500m. Actually we can measure their area of reach.

```
ISOcist$area = st_area(ISOcist)
BUFFERist500$area = st_area(BUFFERist500)
BUFFERist2000$area = st_area(BUFFERist2000)

ratio1 = BUFFERist500$area / ISOcist$area[ISOcist$value == 500] # 1.71
ratio2 = BUFFERist2000$area / ISOcist$area[ISOcist$value == 2000] # 1.22
```

The euclidean buffer of 500m is 1.7 times larger than its isochrone, and the buffer of 2000m is 1.23 times larger than its isochrone.

### 15.2.2. Isochrone from more than 1 point - time

For this purpose we will use the high schools dataset.

```
# import schools
SCHOOLS = st_read("../geo/SCHOOLS_basicsec.gpkg", quiet = TRUE)

SCHOOLS$coordinates = st_coordinates(SCHOOLS) # create coordinate variable

SCHOOLShigh = SCHOOLS |>
  filter(Nivel == "Secundario") # filter the high schools
SCHOOLShigh = SCHOOLShigh |> mutate(id = 1:nrow(SCHOOLShigh)) # provide
  ↵ and id
```

## 15. Buffers vs. Isochones

```
# list of XY coordinates for ORS
coord_schools = data.frame(lon = SCHOOLShigh$coordinates[, 1],
                            lat = SCHOOLShigh$coordinates[, 2])
# id = SCHOOLShigh$id)
```

And proceed with the time isochrones, for a range of 20 min, with 5 min intervals.

```
ISOCschools = ors_isochrones(
  coord_schools,
  profile = "foot-walking",
  range_type = "time", # or distance
  range = 20*60, # 20 minutes in seconds
  interval = 5*60, # to have intervals of 5 minutes
  attributes = "area", #you can directly get area, population, and so on.
  output = "sf"
)

ISOCschools = arrange(ISOCschools, -value) |> # larger polygons on top of
  # the table so they are displayed behind.
  select(-center)
```

And now merge this information with the schools' names.

```
ISOCschools = ISOCschools |>
  mutate(id = group_index + 1, # because group_index starts at 0
         value = value/60) |>
  left_join(SCHOOLShigh |>
    st_drop_geometry() |>
    select(id, INF_NOME, Alunos))

mapview(ISOCschools, zcol = "value", alpha.regions = 0.5)
```

## 15. Buffers vs. Isochrones



And estimate areas for the 20min isochrones

```
summary(ISOschools$area[ISOschools$value==20])/1000000 # in km2
```

Min.	1st Qu.	Median	Mean	3rd Qu.	Max.
3.295	4.303	5.363	5.098	5.804	6.511



# 16. Open transportation data

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

## 16.1. Road Networks

### 16.1.1. 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 its visualization tool at [www.openstreetmap.org](http://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**.

### 16.1.2. HOT Export Tool

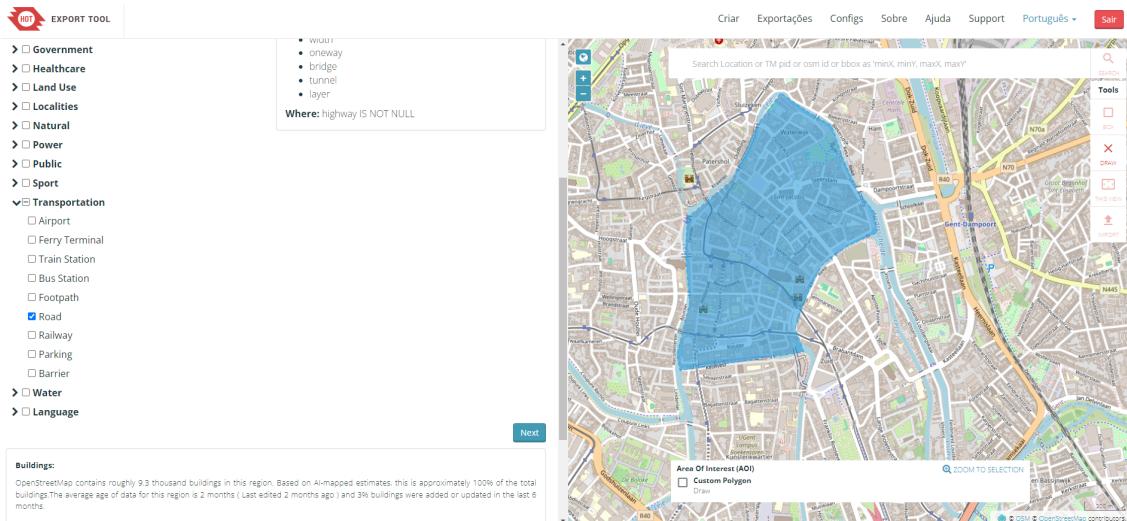
This interactive tool helps you to select the **region** you want to extract, the type of **information** to include, and the output data **format**.

Access via [export.hotosm.org](http://export.hotosm.org)<sup>1</sup>.

---

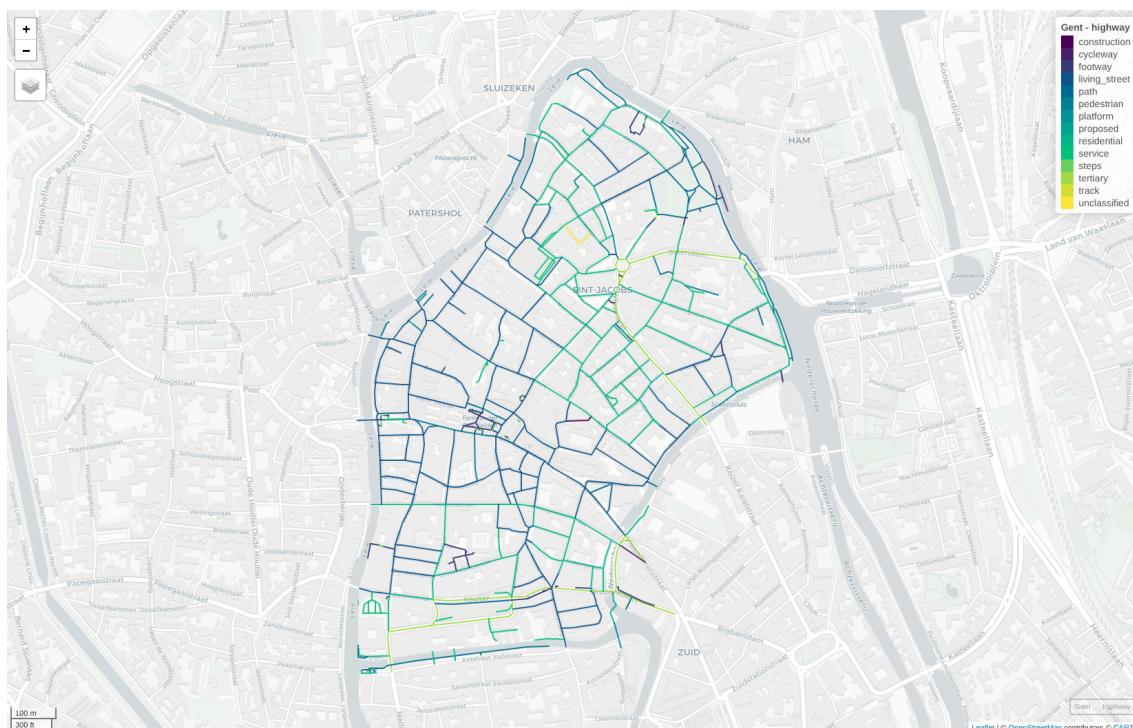
<sup>1</sup>You need an OSM account to use it.

## 16. Open transportation data



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")
```



### 16.1.3. OSM in R

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

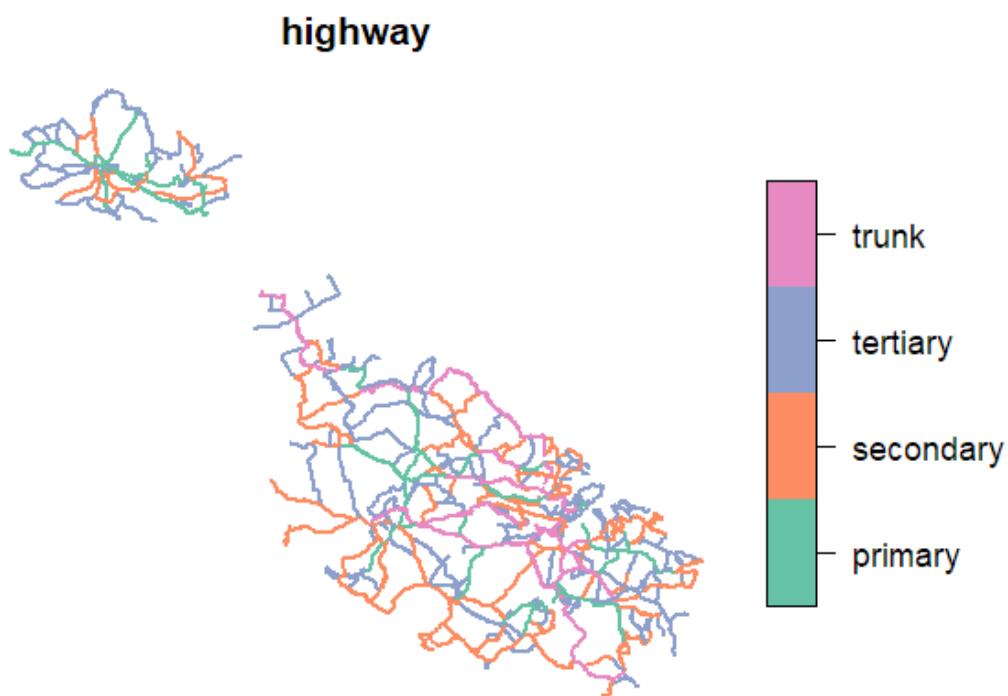
- [osmdata](#)
- [osmextract](#)

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"]])
```



## 16.2. Transportation Services' Data

### 16.2.1. 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.

## 16. Open transportation data

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.

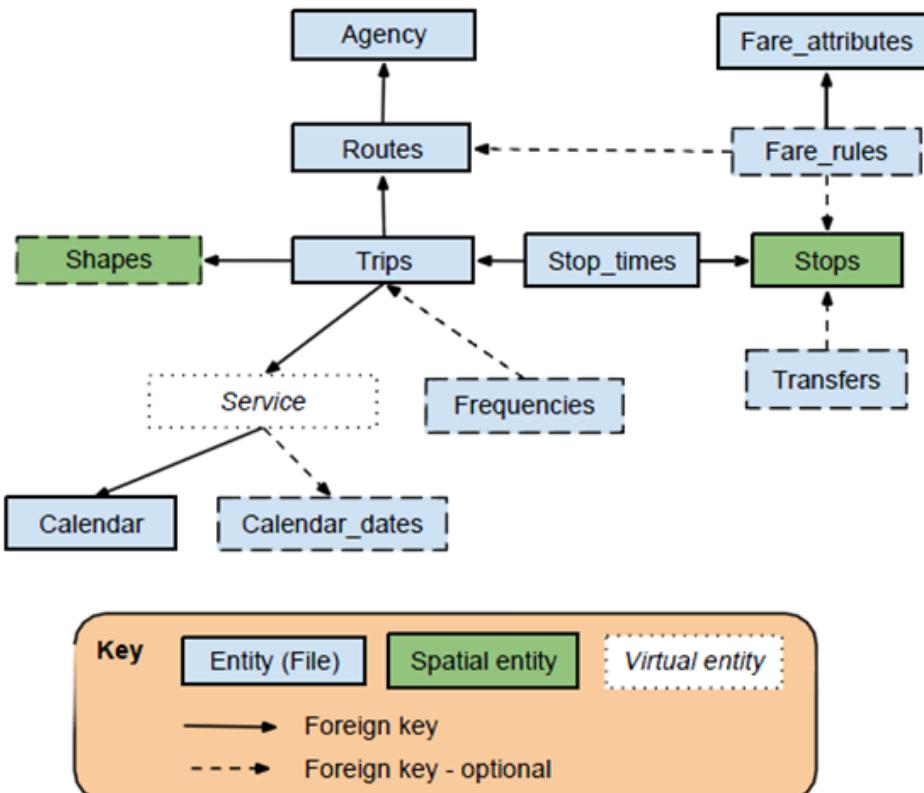


Figure 16.1.: Source: [trilliumtransit.com](http://trilliumtransit.com)

### 16.2.1.1. Online sources

You can find most GTFS data in the following websites:

- [TransitLand](#)
- [TransitFeeds](#)

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

### 16.2.1.2. R packages

There are some nice R packages to read and manipulate GTFS data, such as:

- [tidytransit](#)
- [gtfstools](#)

**i** Be aware that they may share the same function names, so it is important to use one of them at the time.

### 16.2.2. 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 networks**, 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:

The screenshot shows the TransportData.BE website interface. At the top, there is a navigation bar with links for Datasets, Organizations, News, About, and a search bar. The main content area is titled 'Datasets' and shows a search bar with the placeholder 'Search datasets...'. Below the search bar, it says '40 datasets found' and 'Order by: Relevance'. A 'Tags' filter is set to 'Bus'. The results list includes:

- De Lijn - NeTEx Timetables and planning** (De Lijn) - Scheduled timetable data in NeTEx format. Status: METADATA CHECKED & VERIFIED. Format: XML.
- De Lijn - GTFS Realtime** (De Lijn) - This is our GTFS Realtime API. More information about the GTFS Realtime feed specification can be found here: <https://developers.google.com/transit/gtfs-realtime>. Status: Protocol buffers.
- De Lijn - Open Data API search operations** (De Lijn) - The Open Data API search operations offers this functionalities: - Search stops by description. E.g Veemarkt or station. - Search lines by number or description. E.g search line 48 or Hamont - Search locations (x,y) by user address input. Typical for input on a routeplanner (e.g. kerkstraat 14 Hasselt or Achter De Kazerne ). Limited. Status: JSON.
- De Lijn - GTFS Static** (De Lijn) - This is our GTFS STATIC API. More information about the GTFS STATIC specification can be found here: <https://gtfs.org/schedule/>. Status: Other.

On the left side, there is a sidebar with filters for 'NAP type', 'Dataset Type', and 'Transportation modes'. Under 'Transportation modes', 'Bus' is selected, along with 'Tram' (28), 'Metro' (18), 'Car' (14), 'Shuttle bus' (11), 'Truck' (11), 'Motorcycle' (9), 'Taxi' (9), 'Car-sharing' (8), and 'Trolleybus' (8). There are also buttons for 'Show More Tags', 'Area covered by publication', 'Organizations', 'Formats', and 'License type'.

Figure 16.2.: Source: Transport Data Belgium



# 17. Grids

## 17.1. Make grid

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

MUNICIPIOSgeo = st_read("../data/Municipalities_geo.gpkg", quiet = TRUE)
LISBON = MUNICIPIOSgeo |> filter(Municipality == "Lisboa") # filter only
  ↵ Lisbon
```

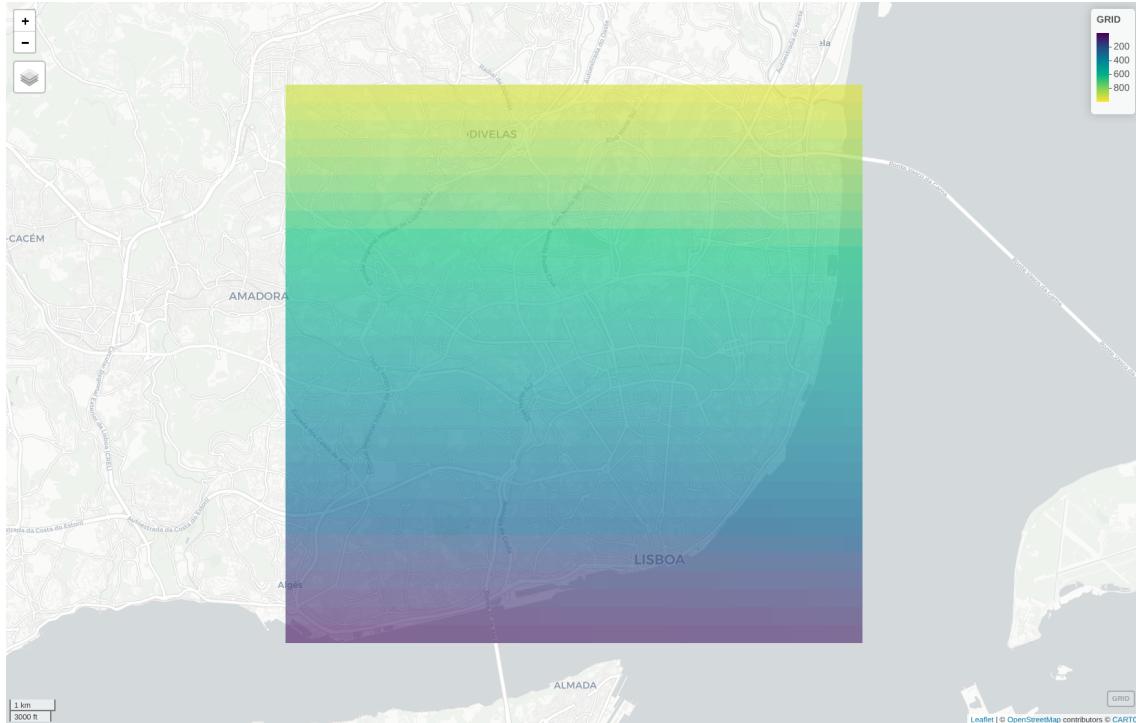
To make a grid we use the `st_make_grid()` function.

```
GRID = LISBON |>
  st_transform(crs = 3857) |> # to a projected crs
  st_make_grid(cellsize = 500, # meters, we are using a projected crs
               what = "polygons",
               square = TRUE) |> # if FALSE, hexagons
  st_sf() |> # from list to sf
  st_transform(crs = 4326) # to WGS84

GRID = GRID |>
  mutate(id = c(1:nrow(GRID))) # just to give an ID to each cell

mapview(GRID, alpha = 0)
```

## 17. Grids



### 17.2. Count points in polygons

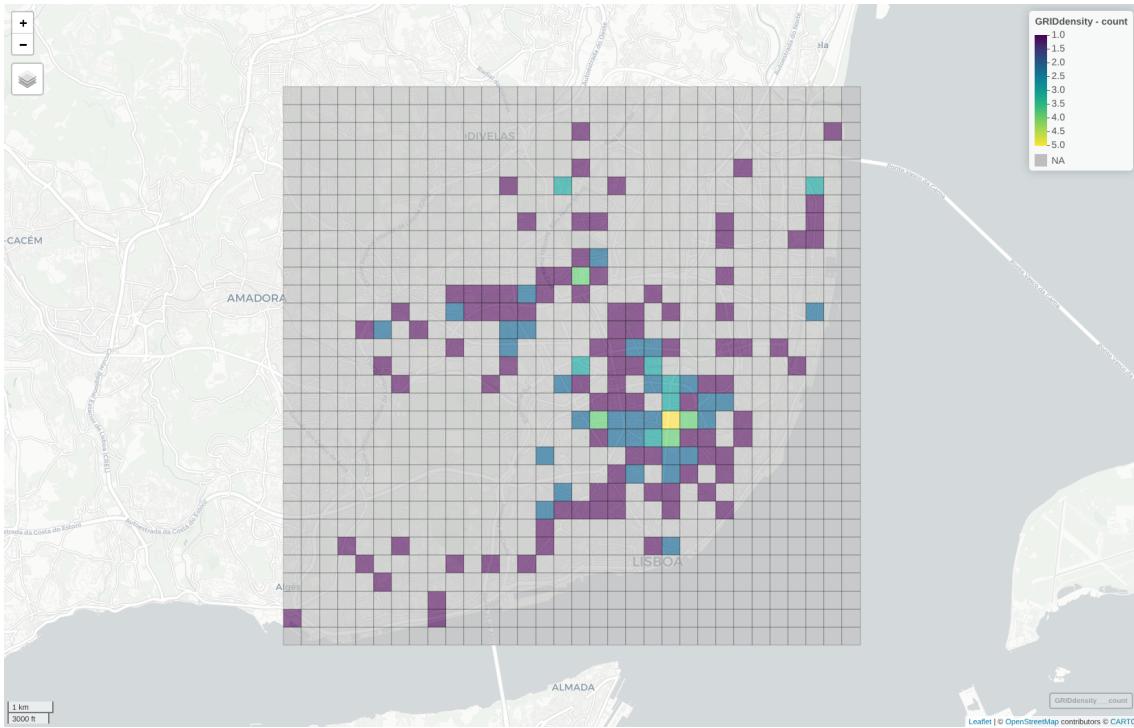
We can use the grid to count survey respondents in each cell.

```
SURVEY = read.csv("../data/SURVEY.txt", sep = "\t")
SURVEYgeo = SURVEY |> st_as_sf(coords = c("lon", "lat"), crs = 4326) #
  ↵ convert to sf from coordinates

SURVEY_with_GRIDid = SURVEYgeo |>
  st_join(GRID,
    join = st_intersects) |>
  st_drop_geometry() |>
  group_by(id) |>
  summarise(count = n()) |> # variable with the number of points
  ungroup()

# back to grid
GRIDdensity = GRID |> left_join(SURVEY_with_GRIDid)

mapview(GRIDdensity, zcol = "count")
```



Or to count bus stops in each cell.

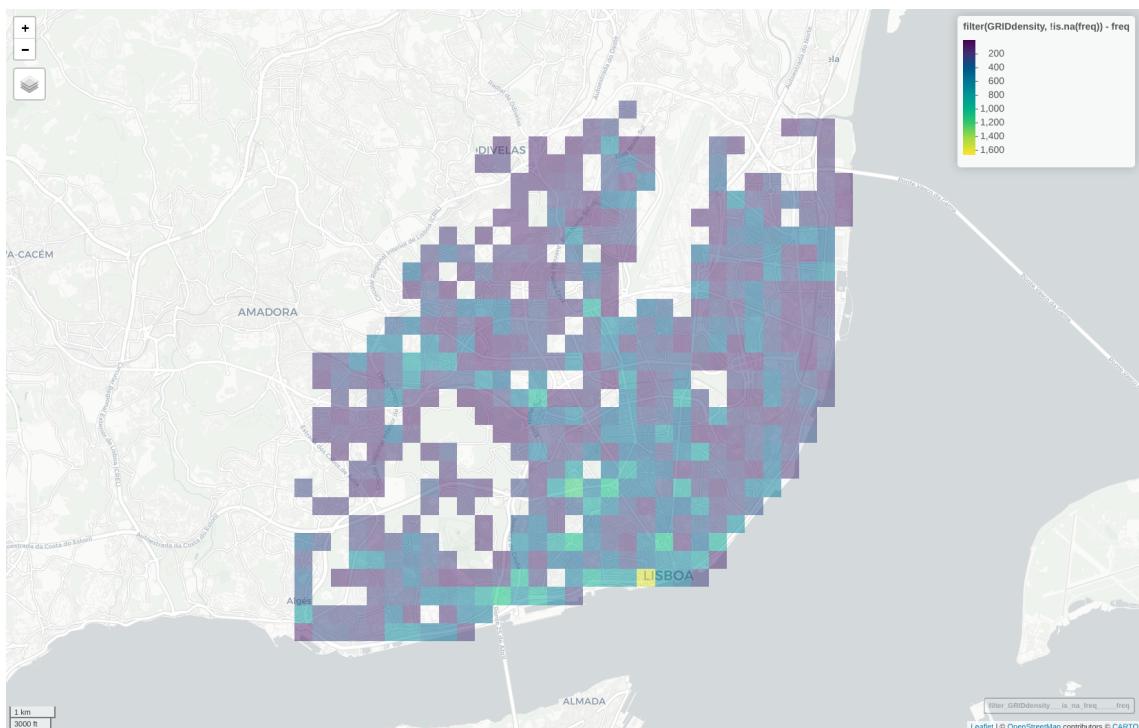
```
BUSstops = st_read("../geo/Carris_stops.gpkg", quiet = TRUE)

BUS_with_GRIDid = BUSstops |>
  st_join(GRID,
    join = st_intersects) |>
  st_drop_geometry() |>
  group_by(id) |>
  summarise(freq = sum(frequency)) |>
  ungroup()

# back to grid
GRIDdensity = GRID |> left_join(BUS_with_GRIDid)

mapview(GRIDdensity |> filter(!is.na(freq)), # exclude NAs
        zcol = "freq",
        alpha.regions = 0.5, # area transparency
        alpha = 0) # hide cell limits
```

## 17. Grids



# References

- Engel, Claudia A. 2023. *Introduction to r*. [cengel.github.io/R-intro/](https://cengel.github.io/R-intro/).
- Field, A. 2024. *Discovering Statistics Using IBM SPSS Statistics*. 6th ed. SAGE Publications. <https://books.google.pt/books?id=83L2EAAAQBAJ>.
- 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](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>.
- Legendre, Pierre, and Louis Legendre. 2012. “Chapter 8 - Cluster Analysis.” In *Numerical Ecology*, edited by Pierre Legendre and Louis Legendre, 3rd ed., 24:337–424. Developments in Environmental Modelling. Elsevier. <https://doi.org/https://doi.org/10.1016/B978-0-444-53868-0.50008-3>.
- 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*. 2nd ed. Chapman; Hall/CRC. <https://r.geocompx.org/>.
- Martinez, Luís Garrido, João de Abreu e Silva, and José Manuel Viegas. 2010. “Assessment of Residential Location Satisfaction in the Lisbon Metropolitan Area.” In. <https://trid.trb.org/View/909819>.
- Oleś, Andrzej K. 2025. *Openrouteservice: An ‘Openrouteservice’ API Client*. <https://github.com/GIScience/openrouteservice-r>.
- Pebesma, Edzer, and Roger Bivand. 2023. *Spatial Data Science: With Applications in R*. Boca Raton: Chapman; Hall/CRC. <https://doi.org/10.1201/9780429459016>.
- R Core Team. 2025. *R: A Language and Environment for Statistical Computing*. Vienna, Austria: R Foundation for Statistical Computing. <https://www.R-project.org/>.
- Wickham, Hadley, Mine Çetinaka-Rundel, and Garrett Grolemund. 2017. *R for Data Science*. 2nd ed. O'Reilly Sebastopol. <https://r4ds.hadley.nz/>.
- Zomorrodi, Ryan. 2024. *Centr: Weighted and Unweighted Spatial Centers*. <https://ryanzomorrodi.github.io/centr/>.

