MQAT

Exercises in R

Rosa Félix Gabriel Valença Filipe Moura

2025-09-06

Table of contents

1 Introduction **6**

Why R and GIS . . . 6

Recommended readings . . . 7

2 Software **8**

2.1 R . . . 8

2.1.1 Windows . . . 8

2.1.2 Mac . . . 8

2.1.3 Ubuntu . . . 9

2.2 RStudio . . . 9

2.2.1 Windows 10/11 . . . 9

2.2.2 MacOS . . . 10

2.2.3 Ubuntu . . . 10

2.3 R packages . . . 10

2.4 Open Route Service . . . 11

I Data and Models **12**

3 R basics **13**

3.1 Math operations . . . 13

3.1.1 Sum . . . 13

3.1.2 Subtraction . . . 13

3.1.3 Multiplication . . . 13

3.1.4 Division . . . 13

3.1.5 Round the number . . . 14

3.2 Basics . . . 14

3.2.1 Combinations . . . 14

3.2.2 Create a comment with `ctrl + shift + c` . . . 15

3.2.3 Create a table . . . 15

3.3 Practical exercise . . . 16

3.3.1 Import dataset . . . 16

3.3.2 Take a first look at the data . . . 16

3.3.3 Explore the data . . . 18

3.3.4 Modify original data . . . 18

3.3.5 Export data . . . 19
3.3.6 Import data . . . 19

4 Data manipulation **20**
4.1 Select variables . . . 20
4.1.1 Using pipes! . . . 21
4.2 Filter observations . . . 22
4.3 Create new variables . . . 22
4.4 Change data type . . . 22
4.4.1 Factors . . . 23
4.5 Join data tables . . . 25
4.6 group_by and summarize . . . 27
4.7 Arrange data . . . 29
4.8 All together now! . . . 29
4.9 Other dplyr functions . . . 30

5 Exploratory Data Analysis **33**
5.1 Intro . . . 33
5.2 Load packages . . . 33
5.3 Dataset . . . 34
5.3.1 Import dataset . . . 34
5.4 Summary statistics . . . 36
5.5 Missing data . . . 38
5.6 Outliers . . . 38
5.6.1 Treating outliers . . . 40
5.7 Histograms . . . 41
5.8 Correlations . . . 45

6 Multiple Linear Regression **50**
6.1 Load packages . . . 50
6.2 Dataset . . . 50
6.2.1 Import dataset . . . 51
6.3 Check the assumptions . . . 52
6.4 Assumption 1: Normal distribution . . . 53
6.5 Assumption 2: Linear relationship . . . 54
6.6 Assumption 3: No multicollinearity . . . 56
6.6.1 Declare the model . . . 57
6.6.2 Assessing the model . . . 58
6.6.3 Calculate the Variance Inflation Factor (VIF) . . . 59
6.7 Assumption 4: independence of observations . . . 60
6.8 Assumption 5: Constant Variance (Homoscedasticity) . . . 60
6.9 Assumption 6: Residuals are normally distributed . . . 63

7 Factor Analysis **64**
7.1 Do it yourself with R . . . 64

8 Cluster Analysis **71**
8.1 Do it yourself with R . . . 71

II GIS in R **78**

9 Introduction to spatial data **79**
9.1 Import vector data . . . 79
9.1.1 Projected vs Geographic Coordinate Systems . . . 80
9.2 Join geometries to data frames . . . 81
9.3 Create spatial data from coordinates . . . 82
9.4 Visuzlize spatial data . . . 83
9.5 Export spatial data . . . 86

10 Centroids of transport zones **87**
10.1 Geometric centroids . . . 87
10.2 Weighted centroids . . . 88
10.3 Compare centroids in a map . . . 89
10.3.1 Interactive map . . . 89
10.3.2 Static map . . . 90

11 OD pairs and desire lines **92**
11.1 Desire lines with `od_to_sf` . . . 93
11.1.1 Filtering desire lines . . . 94
11.2 Oneway desire lines . . . 95
11.3 Using population centroids . . . 97

12 Euclidean and routing distances **99**
12.1 Euclidean distances . . . 99
12.1.1 Import survey data frame convert to sf . . . 99
12.1.2 Create new point at the university . . . 100
12.1.3 Straight lines . . . 100
12.1.4 Distance . . . 101
12.2 Routing Engines . . . 102
12.2.1 Routing distances with `r5r` . . . 103
12.3 Compare distances . . . 105
12.3.1 Circuity . . . 106
12.4 Visualize routes . . . 107

13 Open transportation data **110**
13.1 Road Networks . . . 110
13.1.1 OpenStreetMap . . . 110
13.1.2 HOT Export Tool . . . 110
13.1.3 OSM in R . . . 112
13.2 Transportation Services' Data . . . 113
13.2.1 GTFS . . . 113
13.2.2 National Access Points . . . 114

14 Grids **116**
14.1 Make grid . . . 116
14.2 Count points in polygons . . . 116

15 Interactive maps **118**
15.1 Mapview . . . 119
15.1.1 Export . . . 121
15.2 Rmarkdown . . . 122

References **124**

1 Introduction

Materials, 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.

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

> **This is not the official website for the course.**
>
> Please refer to Fénix: Quantitative Methods of Analysis in Transportation

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 reeling with georreferenced transportation data, such as bike sharing route trips' datasets, origin-destination flows datasets, home/work locations, GTFS public transit data, and so on. There is a need to learn how to locate these open datasets, how to explore them and how to integrate them into transportation and urban analysis. Additionally, the use of open source software and datasets allows researchers to perform methods that are reproducible and transparent.

TLDR

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

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.

2 Software

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

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

2.1 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). he most recent version is `4.5.1`, but you can use `>= 4.1.x` if you already have it installed.

2.1.1 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.

2.1.2 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.

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

2.1.3 Ubuntu

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

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

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

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

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

Then run:

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

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

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

2.2 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.

2.2.1 Windows 10/11

Download RStudio 2025.05 and run the executable file.

2.2.2 MacOS

Download RStudio 2025.05 and install it as usual.

2.2.3 Ubuntu

> These are instructions for Ubuntu **24**. If you use other linux distribution, please follow the instructions on Posit RStudio.

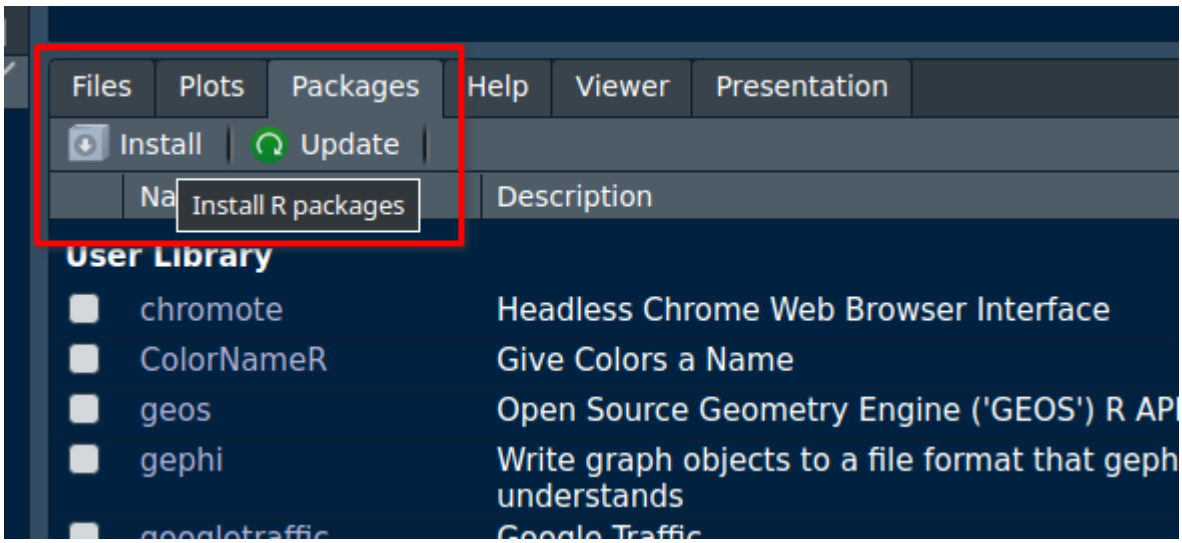
Install it via the terminal:

```
sudo apt install libssl-dev libclang-dev
wget https://download1.rstudio.org/electron/jammy/amd64/rstudio-2025.05.1-513-amd64.deb
sudo dpkg -i rstudio*
rm -v rstudio*
```

2.3 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:

```r
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
```

2.4 Open Route Service

Sign up for an account and create a token.

Copy your API.

In RStudio console, run:

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

openrouteservice::ors_api_key("YOUR-API-KEY")
```

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

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

Part I

Data and Models

3 R basics

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

3.1 Math operations

3.1.1 Sum

```
1+1
```

```
[1] 2
```

3.1.2 Subtraction

```
5-2
```

```
[1] 3
```

3.1.3 Multiplication

```
2*2
```

```
[1] 4
```

3.1.4 Division

```r
8/2
```

```
[1] 4
```

3.1.5 Round the number

```r
round(3.14)
```

```
[1] 3
```

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

```
[1] 3.1
```

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

3.2 Basics

3.2.1 Combinations

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

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

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

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

> **Try it yourself**
>
> Try to write a combination with the numbers 10, 11, 56, 57, 58.

3.2.2 Create a comment with `ctrl + shift + c`

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

3.2.3 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
```

```
  modes trips
1   car   200
```

Look at first row and column

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

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

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

```
NULL
```

3.3 Practical exercise

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

3.3.1 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.

3.3.2 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 0 91 0 ...
 $ Car            : num  12484 353 107 861 114 ...
 $ PTransit       : num  833 0 81 5 0 ...
 $ Other          : num  7 214 0 0 0 0 0 0 0 0 ...
```

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

```
nrow(data)
```

```
[1] 315
```

```
ncol(data)
```

```
[1] 8
```

Open the dataset

```
View(data)
```

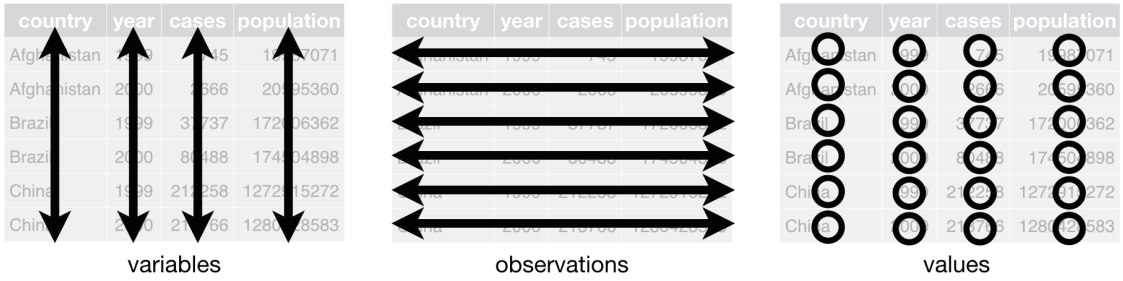


Figure 3.1: The following three rules make a dataset tidy: variables are columns, observations are rows, and values are cells. [@wickham17r]

3.3.3 Explore the data

Check the total number of trips

Use `$` to select a variable of the data

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

```
[1] 5299853
```

Percentage of car trips related to the total

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

```
[1] 59.17638
```

Percentage of active trips related to the total

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

```
[1] 24.44883
```

3.3.4 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)]
```

3.3.5 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.
```

3.3.6 Import data

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

4 Data manipulation

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

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

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

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

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

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

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

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

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

4.1 Select variables

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

```
glimpse(TRIPS)
```

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

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

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

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

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

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

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

4.1.1 Using pipes!

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

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

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

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

Two things to **note**:

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

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

4.2 Filter observations

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

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

You can have other conditions inside the condition.

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

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

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

Other filter conditions:

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

4.3 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
```

4.4 Change data type

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

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

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

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

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

Most used data types are:

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

4.4.1 Factors

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

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

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

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

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

```
[1] 0 1
```

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

```
  0   1
188  48
```

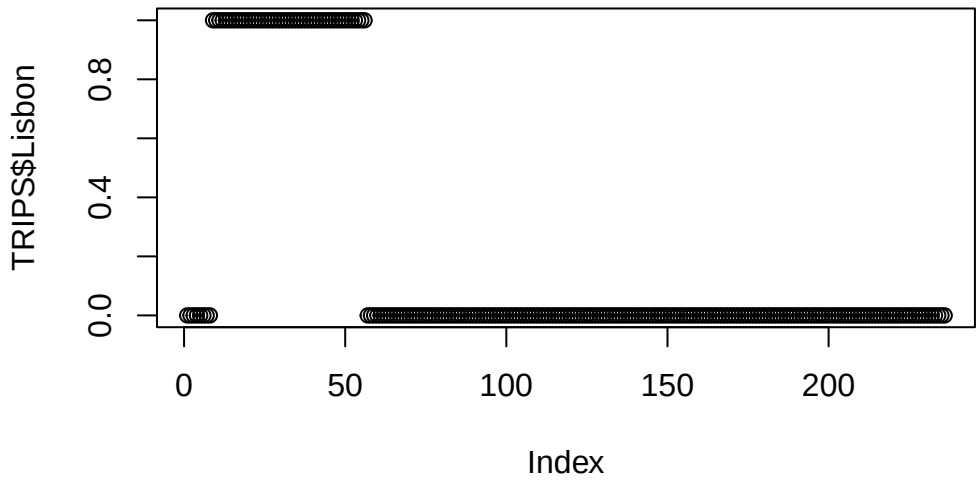
```r
table(TRIPS$Lisbon_factor)
```

```
 No Yes
188  48
```

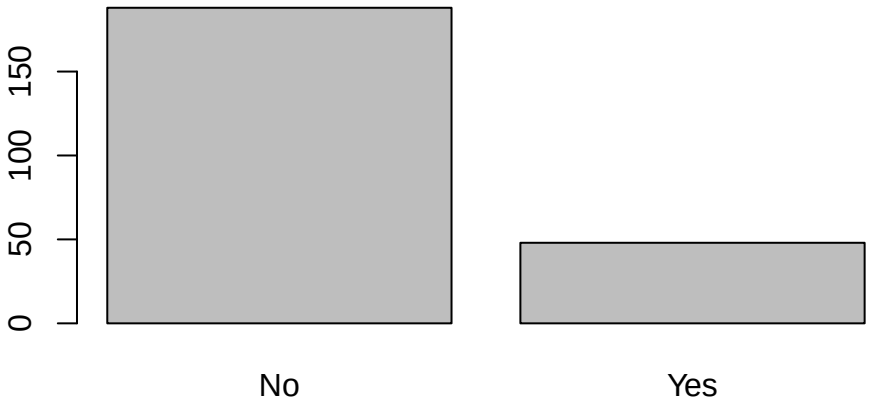
The first number to appear is the first level, and so on.

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

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




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



4.5 Join data tables

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

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

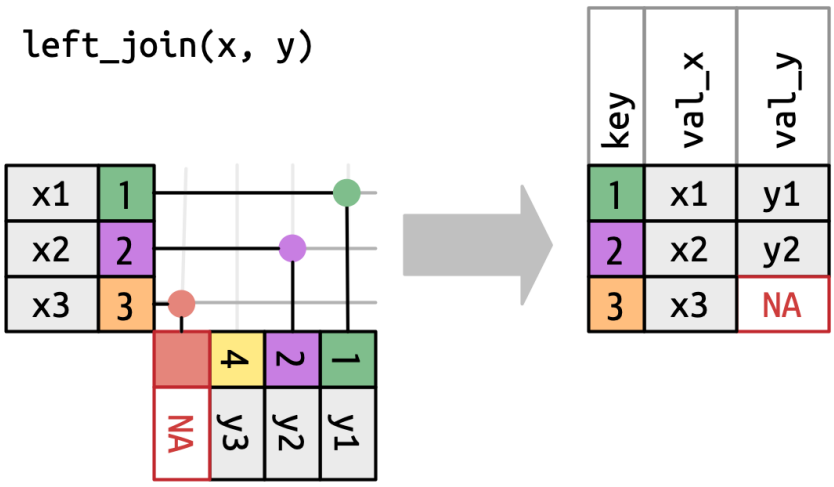


Figure 4.1: A visual representation of the left join where every row in x appears in the output.Source: R for Data Science.

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.

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

```r
head(TRIPS)
```

```
# A tibble: 6 x 13
  Origin Total  Walk  Bike   Car PTransit Other Internal Lisbon Car_perc
  <chr>  <dbl> <dbl> <dbl> <dbl>    <dbl> <dbl>    <dbl>  <dbl>    <dbl>
1 110501 35539 11325  1309 21446     1460     0        0      0     60.3
2 110501 47602  3502   416 37727     5519   437        1      0     79.3
3 110506 37183 12645    40 22379     2057    63        0      0     60.2
4 110506 42313  1418   163 37337     3285   106        1      0     88.2
5 110507 30725  9389  1481 19654      201     0        0      0     64.0
6 110507 54586  2630   168 44611     6963   215        1      0     81.7
# i 3 more variables: Origin_num <int>, Lisbon_factor <fct>,
#   Internal_factor <fct>
```

```r
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`.

4.6 group_by and summarize

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

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

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

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

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

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

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

We summed the total number of trips in each municipality.

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

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

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

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

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

4.7 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.

4.8 All together now!

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

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

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

  filter(Internal_factor == "Inter")|>

  left_join(Municipalities) |>

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

  arrange(desc(Car_perc))
```

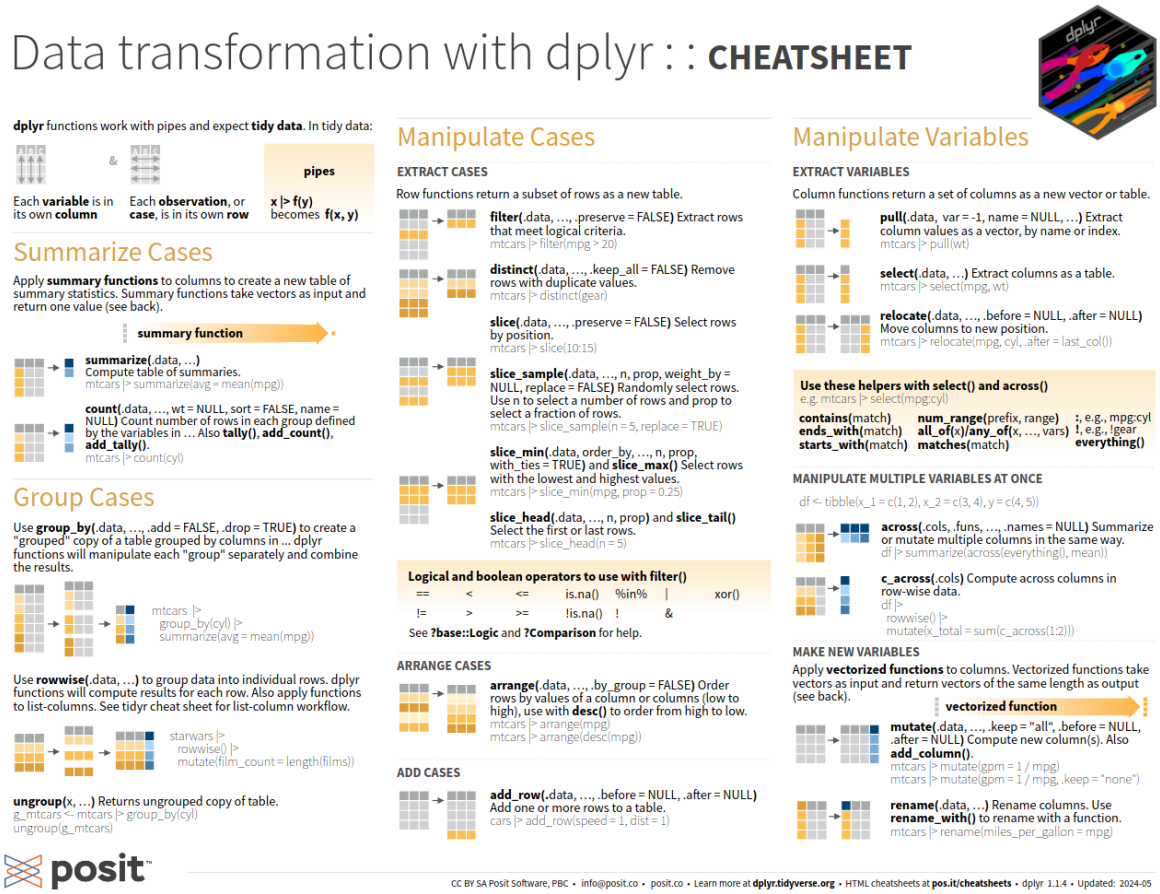
With this code we will have a table with the total number of intercity trips, by municipality, with their names instead of codes, arranged by the percentage of car trips.

```
TRIPS_pipes
```

```
# A tibble: 18 x 4
   Municipality         Total    Car Car_perc
   <chr>                <dbl>  <dbl>    <dbl>
 1 Mafra                65811  46329     70.4
 2 Sesimbra             49370  31975     64.8
 3 Cascais             161194  96523     59.9
 4 Palmela              66428  39688     59.7
 5 Alcochete            16954   9839     58.0
 6 Setúbal             129059  70318     54.5
 7 Montijo              57164  30900     54.1
 8 Seixal              120747  63070     52.2
 9 Sintra              237445 123408     52.0
10 Oeiras              134862  66972     49.7
11 Almada              105841  49012     46.3
12 Loures              132310  60478     45.7
13 Barreiro             52962  24160     45.6
14 Odivelas             93709  39151     41.8
15 Vila Franca de Xira 115152  47201     41.0
16 Moita                51040  17394     34.1
17 Amadora             117727  33818     28.7
18 Lisboa              280079  69038     24.6
```

4.9 Other dplyr functions

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



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

Table 4.1: OD matrix in long format

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

Table 4.2: OD matrix in wide format

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

5 Exploratory Data Analysis

5.1 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`

5.2 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
```

5.3 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.
- `PTransit` - 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^2

5.3.1 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, Yes, No,~
$ Lisboa           <fct> No, No, 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 ...
```

5.4 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 5.1: 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	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	

skim_variable	n_missing	complete_rate	mean	sd	p0	p25	p50	p75	p100	hist
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	

5.5 Missing data

Is there missing data (`NA`)? How many?

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

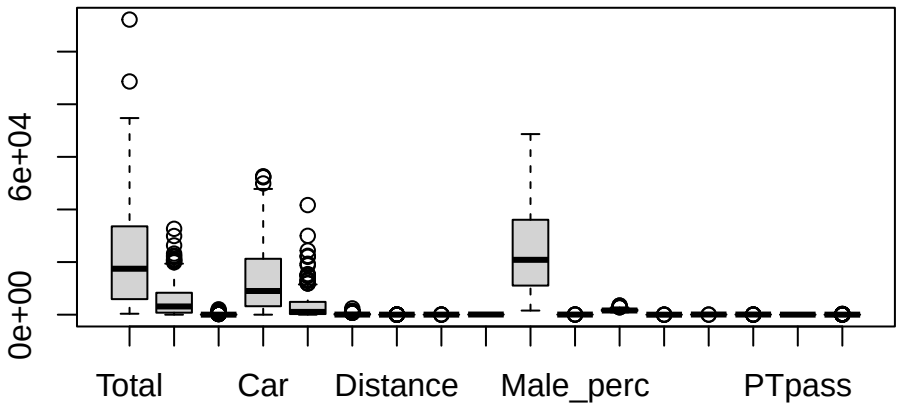
```
FALSE
 4720
```

5.6 Outliers

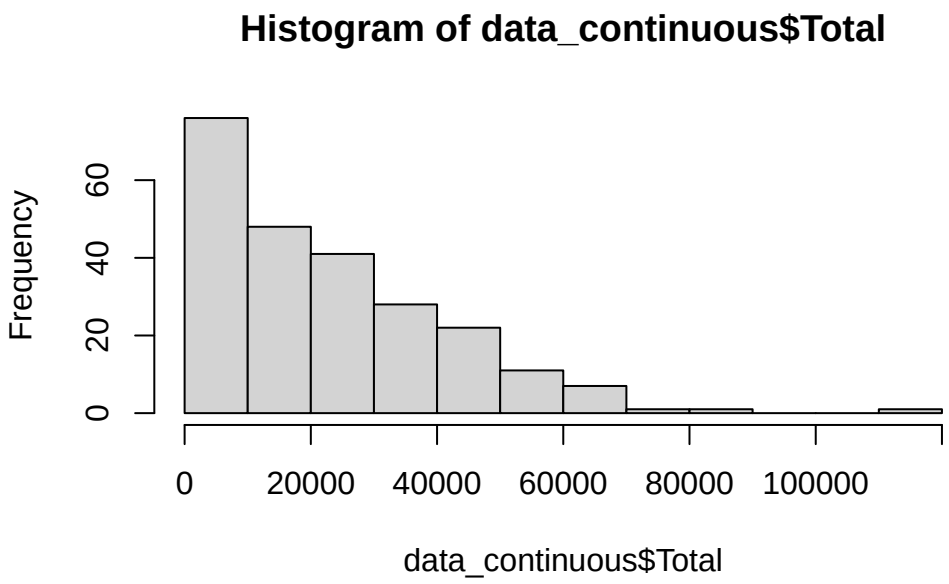
Inspect outliers and distributions

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

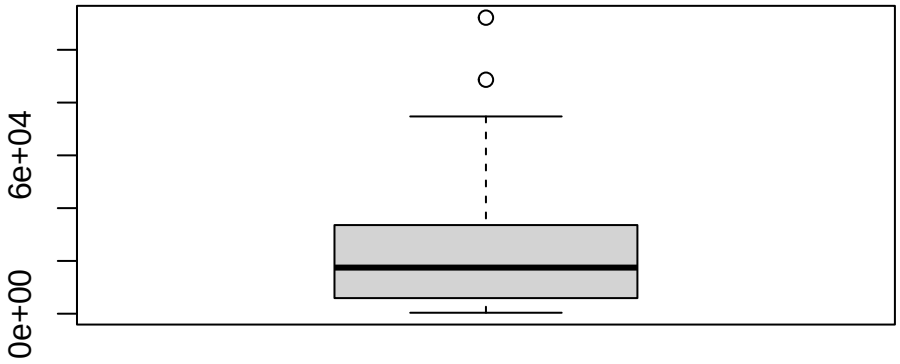
```
data_continuous = data |> select(-Origin_dicofre16, -internal, -Lisboa) # Exclude categorical
boxplot(data_continuous) # Exclude categorical variables
```




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




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



5.6.1 Treating outliers

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

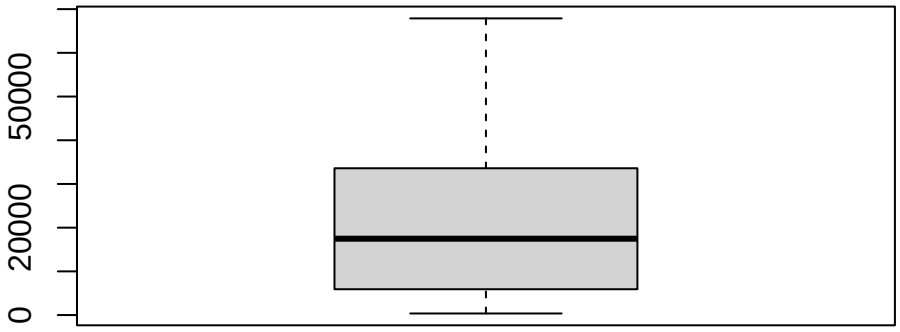
```
outlier = function(x) {
  q = quantile(x, probs = c(0.25, 0.75), na.rm = TRUE)  # Q1 and Q3
  caps = quantile(x, probs = c(0.05, 0.95), na.rm = TRUE) # 5th and 95th percentile
  H = 1.5 * IQR(x, na.rm = TRUE) # interquartile range

  case_when(
    x < (q[1] - H) ~ caps[1], # replace values that are LESS than Q1-1.5*IQR with the P5 valu
    x > (q[2] + H) ~ caps[2], # replace values that are MORE than Q3+1.5*IQR with the P95 val
    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)
```



Compare the variable with and without the outliers

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

```
[1] 22457
```

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

```
[1] 22013.51
```

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

```
[1] 17474
```

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

```
[1] 17474
```

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

```
[1] 19084.45
```

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

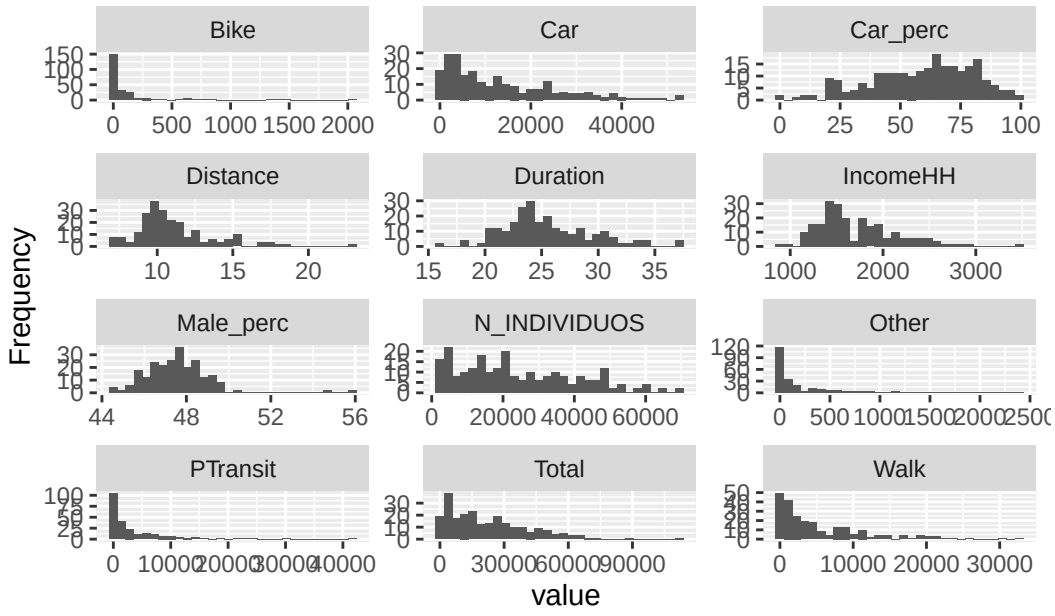
```
[1] 17739.39
```

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

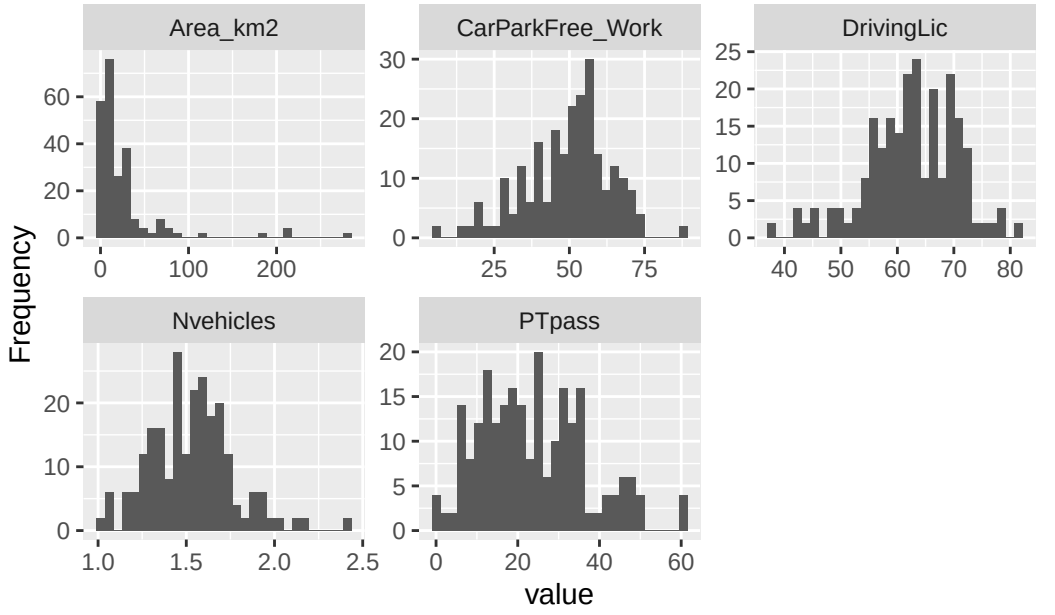
5.7 Histograms

Plot histograms of all the continuous variables

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

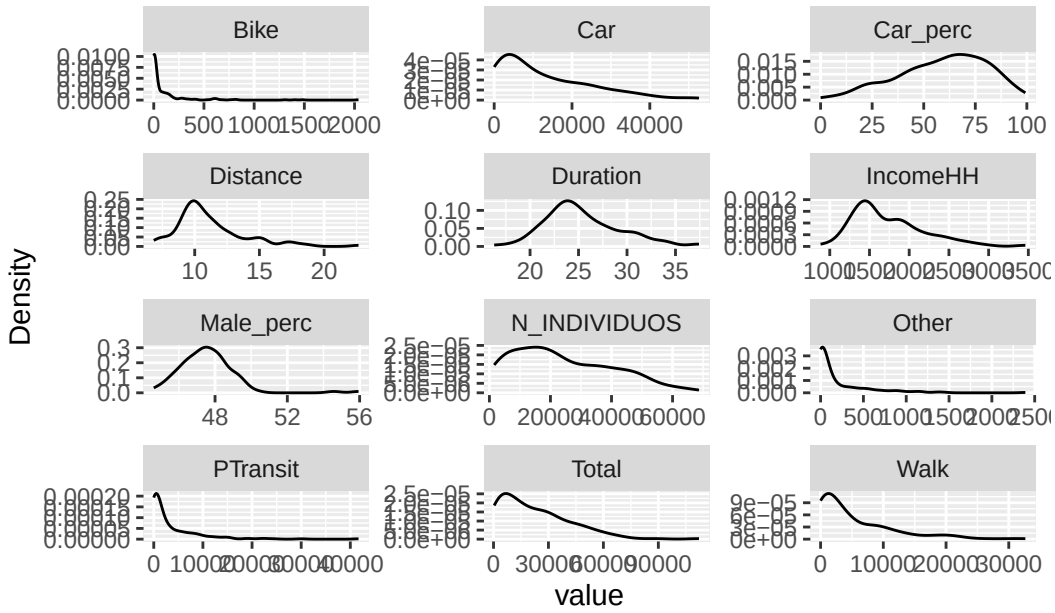


Page 1

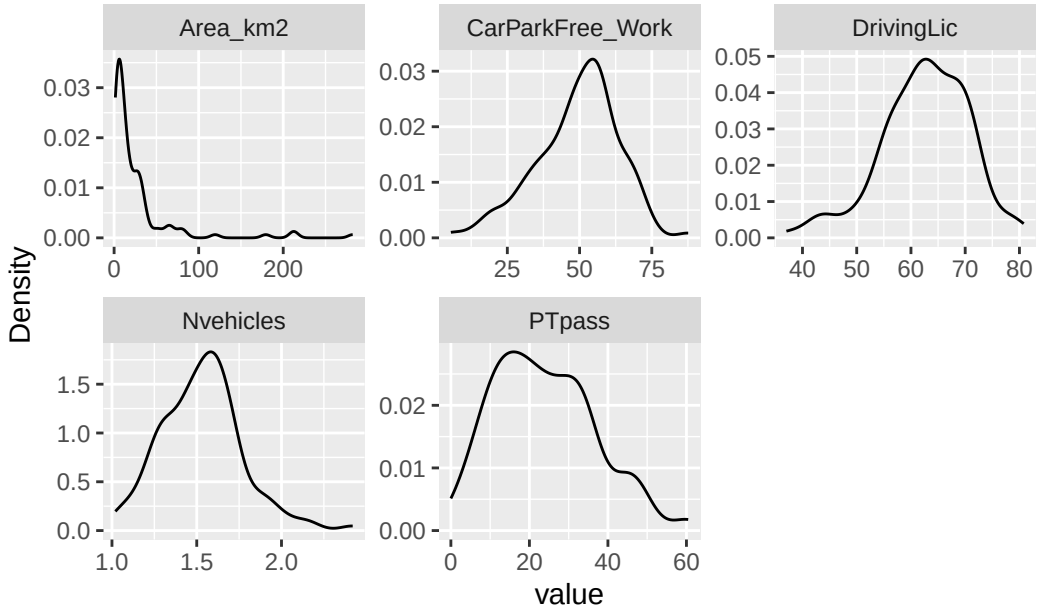


Page 2

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



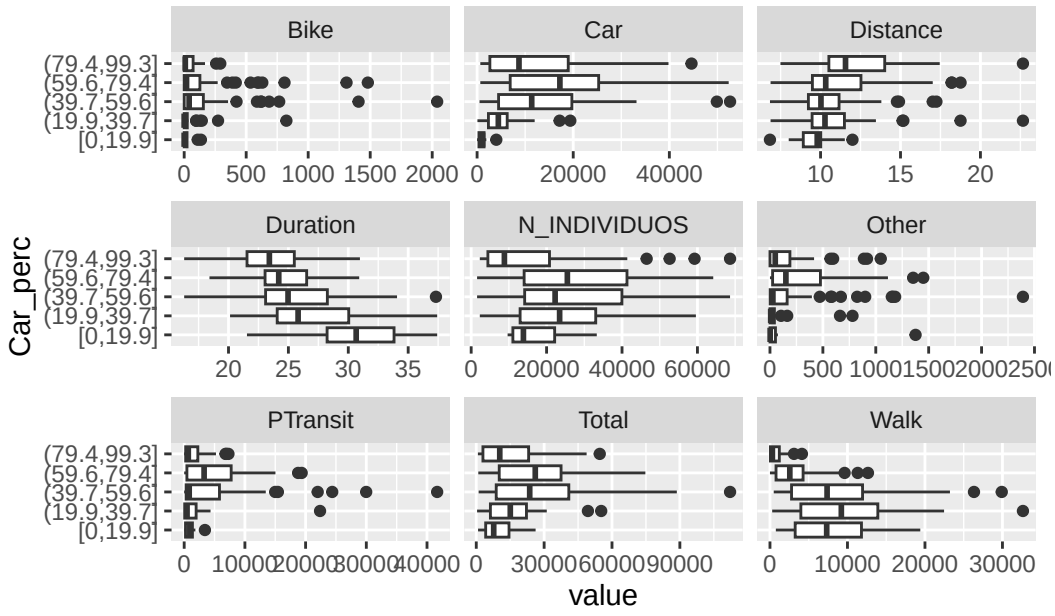
Page 1



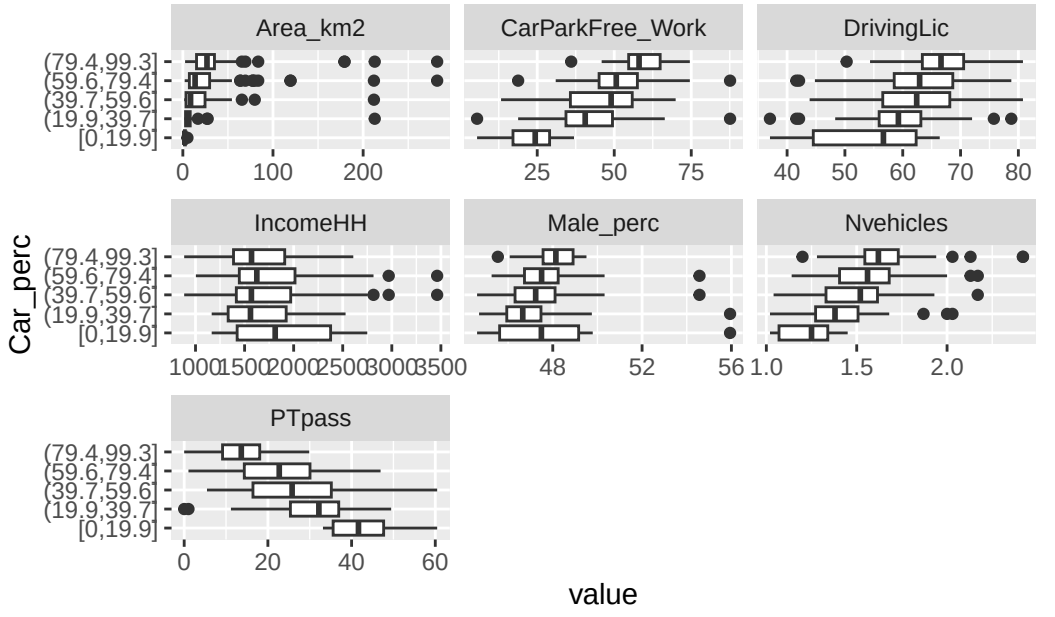
Page 2

Check how other variables are distributed regarding `Car_perc`

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



Page 1



Page 2

Note

When `Car_perc` increases, `PTpass` decreases.

Exercise

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

5.8 Correlations

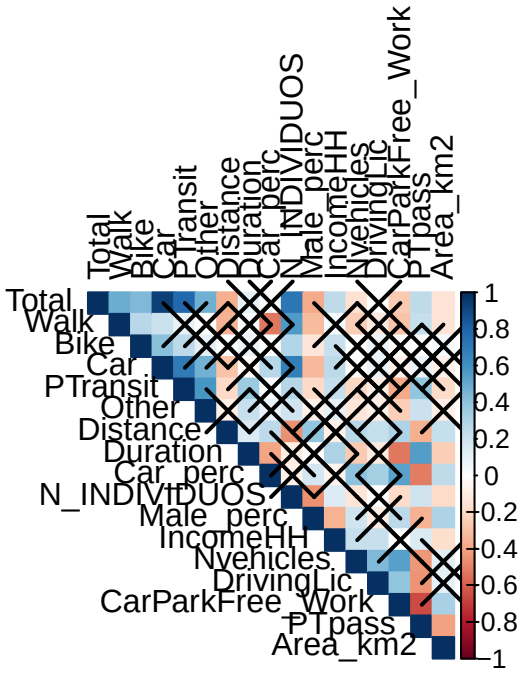
Plot correlation heatmaps, between continuous variables.

```
# 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
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
```

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

corrplot(
  corrmat,
  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
)
```

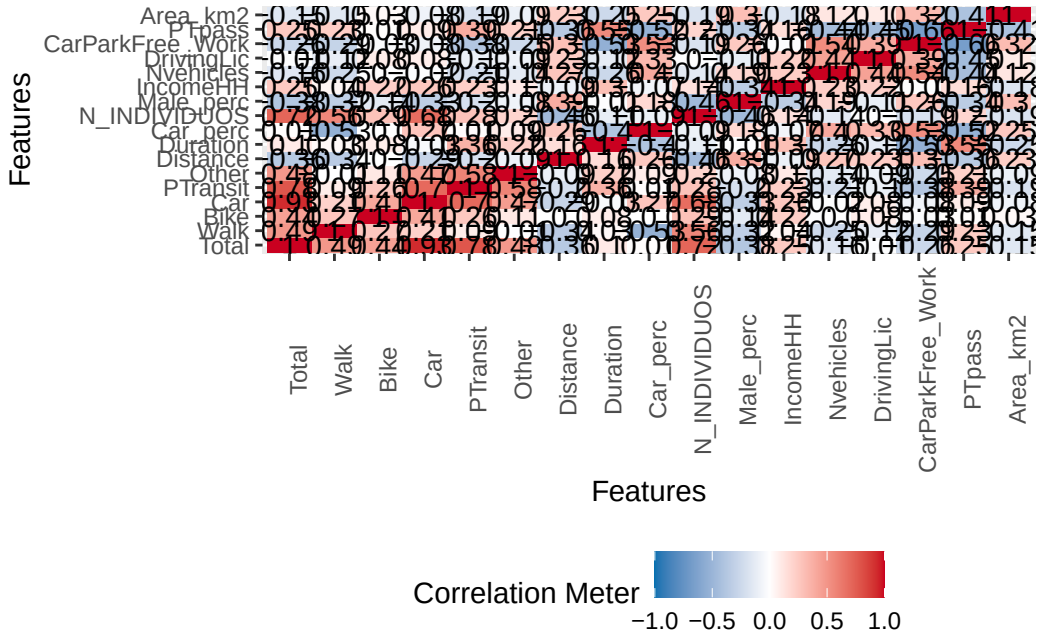


Note

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-value < 0.05).

Other method, using the `DataExplorer` package

```
plot_correlation(data_continuous)
```



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
```

```r
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.

6 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.

6.1 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.)
```

6.2 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)
- `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^2

6.2.1 Import dataset

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

Show summary statistics

```
skim(data)
```

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	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.

6.3 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

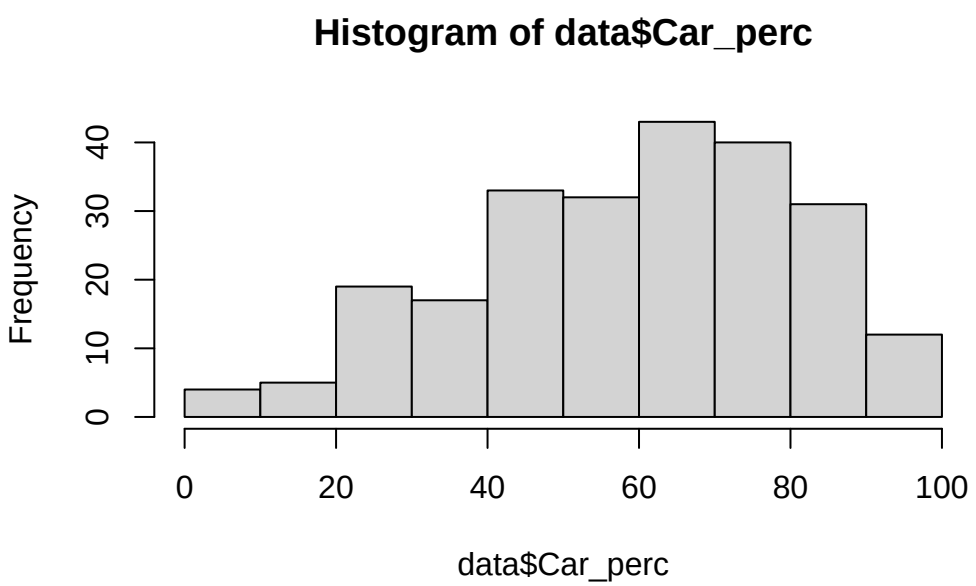
3. No multicollinearity between independent variables (or only very little)
4. The observations are independent
5. Constant Variance (Assumption of Homoscedasticity)
6. Residuals are normally distributed

6.4 Assumption 1: Normal distribution

The Dependent Variable is be normally distributed.

Check the histogram of `Car_perc`:

```
hist(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.

6.5 Assumption 2: Linear relationship

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

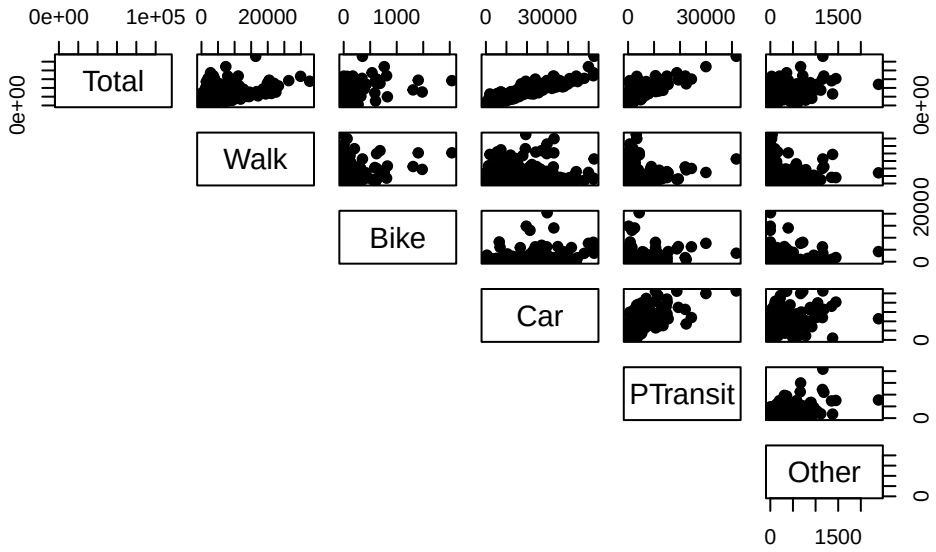
We can check this assumption by plotting scatterplots of the DV against each IV:

```
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 parki
```

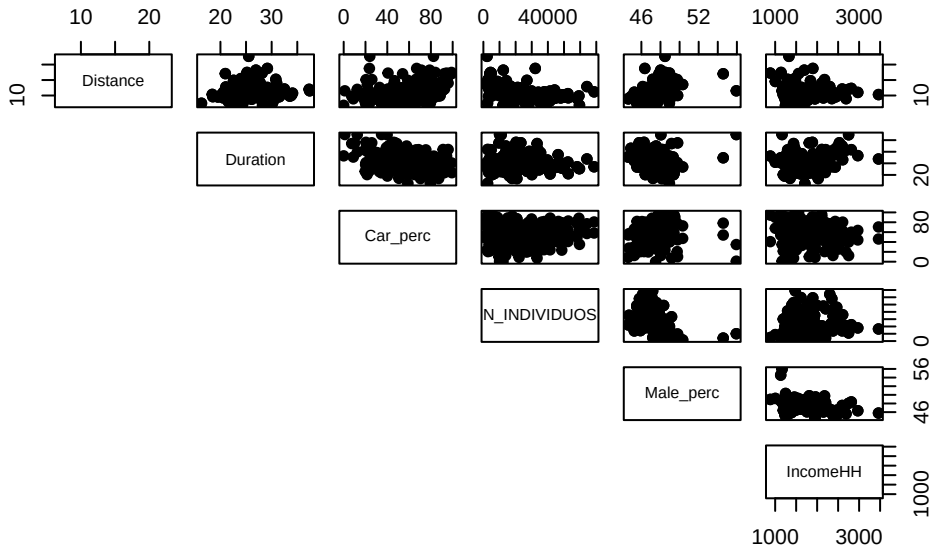
```r
plot(x = data$Car_perc, y = data$PTpass, xlab = "Car_perc", ylab = "PTpass")
plot(x = data$Car_perc, y = data$internal, xlab = "Car_perc", ylab = "internal trips")
plot(x = data$Car_perc, y = data$Lisboa, xlab = "Car_perc", ylab = "Lisboa")
plot(x = data$Car_perc, y = data$Area_km2, xlab = "Car_perc", ylab = "Area_km2")
```

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

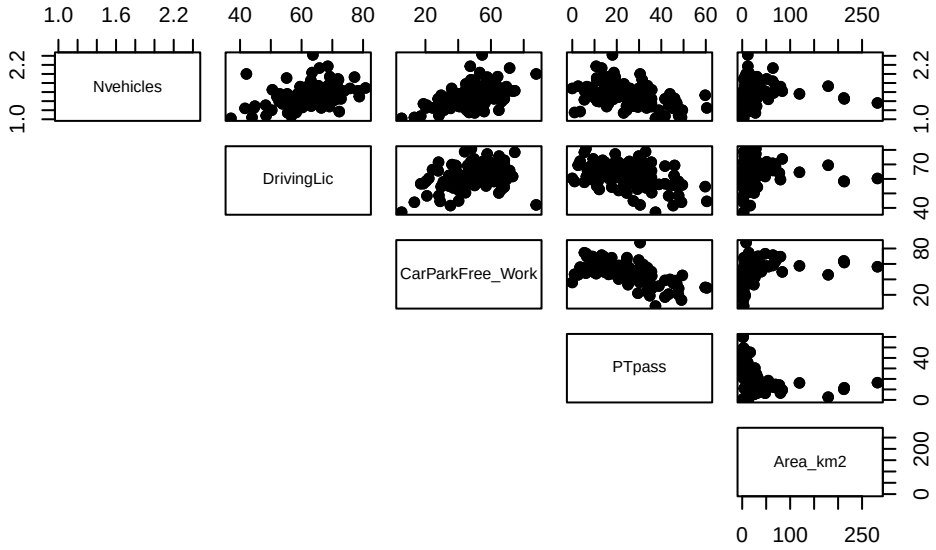
```r
# pairs(data_continuous, pch = 19, lower.panel = NULL) # we have too many variables, let's s
pairs(data_continuous[,1:6], pch = 19, lower.panel = NULL)
```




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




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



6.6 Assumption 3: No multicollinearity

Check the correlation plot before choosing the variables.

6.6.1 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 +
    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
```

6.6.2 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 $>$ Fcritical $= 2.39$, 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 analyze 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.

6.6.3 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)
```

```
       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.

6.7 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   0.102
 Alternative hypothesis: rho != 0
```

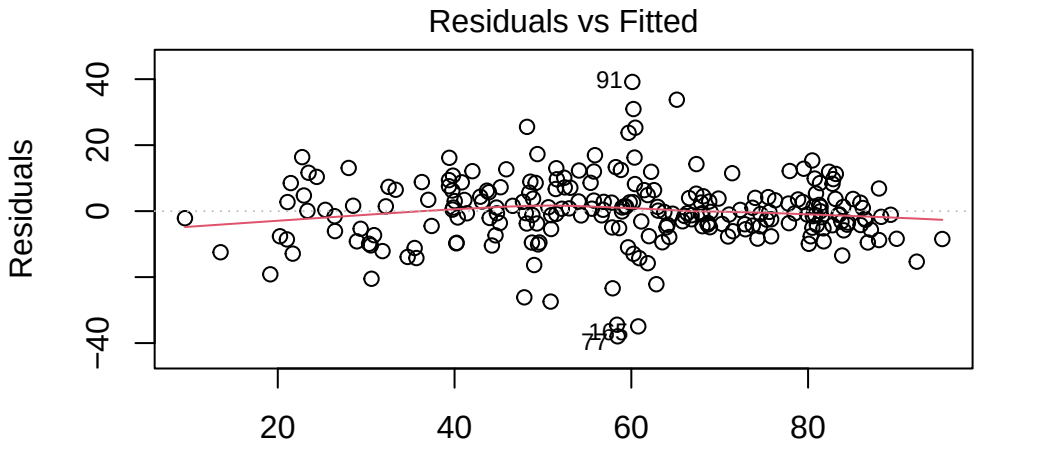
H_0 (null hypothesis): There is no correlation among the residuals. Since p-value > 0.05, we do not reject the null hypothesis and we can not discard that there is autocorrelation in the model.

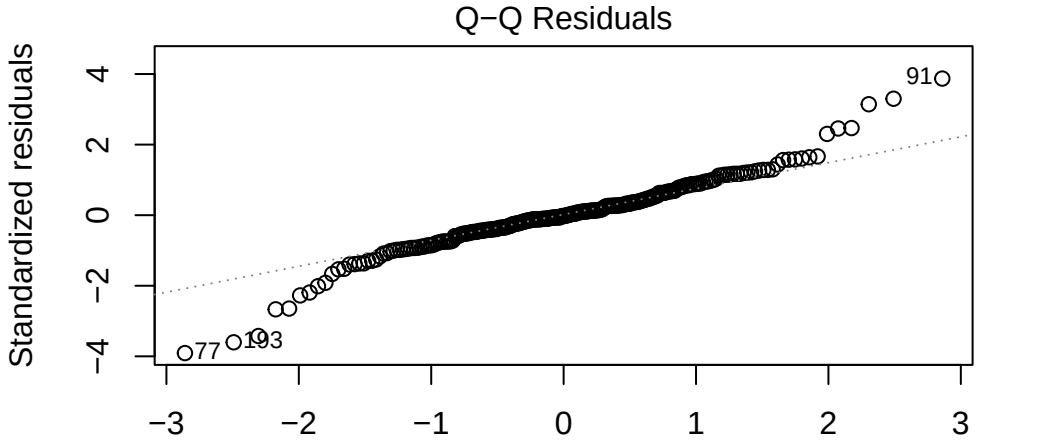
> **Note**
>
> In the Durbin-Watson test, values of the D-W Statistic vary from 0 to 4.
> If the values are from 1.8 to 2.2 this means that there is **no autocorrelation** in the model.

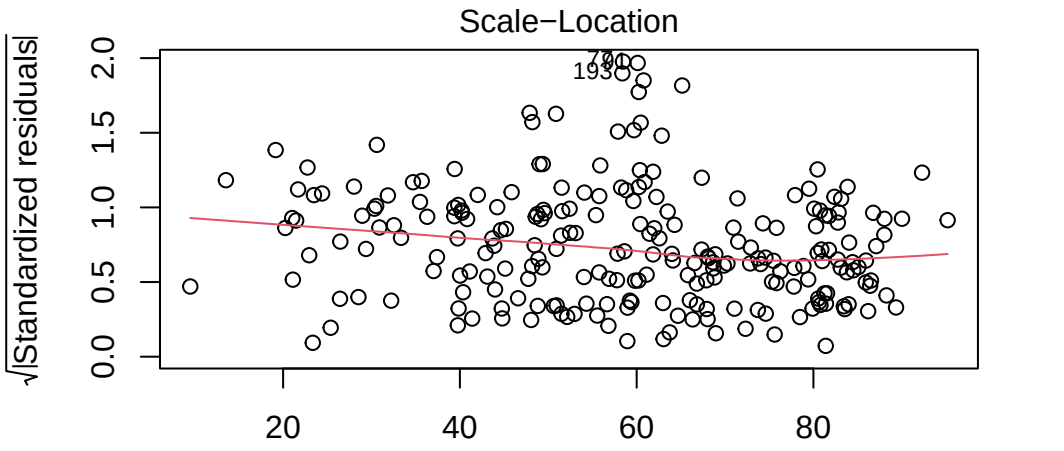
6.8 Assumption 5: Constant Variance (Homoscedasticity)

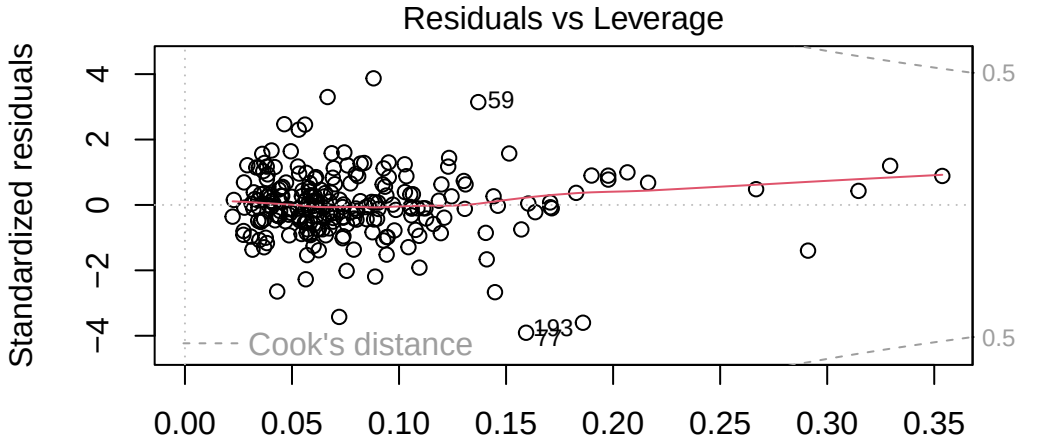
The simplest way to determine if this assumption is met is to create a plot of standardized residuals versus predicted values.

```
plot(model)
```


Residuals vs Fitted
Residuals
Fitted values
lm(Car_perc ~ Total + Walk + Bike + Car + PTransit + Other + Distance + D


Q−Q Residuals
Standardized residuals
Theoretical Quantiles
lm(Car_perc ~ Total + Walk + Bike + Car + PTransit + Other + Distance + D





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.

6.9 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, Kolmogorov-Smironov to validate those results.

7 Factor Analysis

7.1 Do it yourself with R

```
#' #### Example exercise: "Residential location satisfaction in the Lisbon metropolitan area"
#'
#' The aim of this study was to examine the perception of households towards their residentia
#'
#' _Reference:_ Martinez, L. G., de Abreu e Silva, J., & Viegas, J. M. (2010). Assessment of
#'
#' **Your task:** Analyse the data and create meaningful latent factors.
#'
#' ## Data
#' #### Variables:
#'
#' * `DWELCLAS`: Classification of the dwelling;
#' * `INCOME`: Income of the household;
#' * `CHILD13`: Number of children under 13 years old;
#' * `H18`: Number of household members above 18 years old;
#' * `HEMPLOY`: Number of household members employed;
#' * `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;
#' * `EQUINDEX`: Number of undergraduate students/Population over 20 years old (500m)
#'
#'
#' #### 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 variabl
#'
#' #### Assumptions:
#' * Normality;
#' * linearity;
#' * Homogeneity;
#' * Homoscedasticity (some multicollinearity is desirable);
#' * Correlations between variables < 0.3 (not appropriate to use Factor Analysis)
#'
#' ## Let's start!
#' #### Import Libraries
library(foreign) # Library used to read SPSS files
library(nFactors) # Library used for factor analysis
library(tidyverse) # Library used in data science to perform exploratory data analysis
library(summarytools) # Library used for checking the summary of the dataset
library(psych) # Library used for factor analysis
library(GPArotation) # Library used for factor analysis

#'
#' ### Get to know your dataset
#' ##### Import dataset
#'
df <- read.spss("Data/example_fact.sav", to.data.frame = T) #transforms a list into a data.fr

#'
#' ##### Take a look at the main characteristics of the dataset
#'
class(df) #type of data
str(df)

#'
#' ##### Check summary statistics of variables
descriptive_stats <- dfSummary(df)
view(descriptive_stats)
```

```
#' > **Note:** I used a different library of the MLR chapter for perfoming the summary statis
#'
#' ##### Take a look at the first values of the dataset
#'
head(df,5)

#'
#' ##### Make ID as row names or case number
#'
df<-data.frame(df, row.names = 1)

#'
#' ### Evaluating the assumptions for factoral analysis
#'
#' Let's run a random regression model in order to evaluate some assumptions
#'
random = rchisq(nrow(df), 32)
fake = lm(random ~ ., data = df)
standardized = rstudent(fake)
fitted = scale(fake$fitted.values)

#'
#' * **Normality**
#'
hist(standardized)

#'
#' * **Linearity**
#'
qqnorm(standardized)
abline(0,1)

#'
#' * **Homogeneity**
#'
plot(fitted, standardized)
abline(h=0,v=0)

#'
#' * **Correlations between variables**
#' Correlation matrix
corr_matrix <- cor(df, method = "pearson")
```

```
#'
#' The **Bartlett** test examines if there is equal variance (homogeneity) between variables
cortest.bartlett(corr_matrix, n = nrow(df))

#' > **Note:** The null hypothesis is that there is no correlation between variables. Therefo
#'
#' ##### Check for sampling adequacy - KMO test
#'
KMO(corr_matrix)

#'
#' > **Note:** We want at least 0.7 of the overall Mean Sample Adequacy (MSA). If, 0.6 < MSA
#'
#' ##### Determine the number of factors to extract
#'
#' **1. Parallel Analysis**
#'
num_factors = fa.parallel(df, fm = "ml", fa = "fa")

#'
#' > **Note:** `fm` = factor math; `ml` = maximum likelihood; `fa` = factor analysis
#'
#' 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.
#'
#'
#' **2. Kaiser Criterion**
#'
sum(num_factors$fa.values > 1) #Determines the number of factors with eigenvalue > 1

#'
#' You can also consider factors with eigenvalue > 0.7, since some of the literature indicate
#'
#' **3. Principal Component Analysis (PCA)**
#'
#' * Print variance that explains the components
df_pca <- princomp(df, cor=TRUE) #cor = TRUE, standardizes your dataset before running a PCA
summary(df_pca)
```

```
#'
#' * Scree Plot
plot(df_pca,type="lines", npcs = 31)

#'
#' > **Note:** Check the cummulative variance of the first components and the scree plot, and
#'
#' **PCA is not the same thing as Factor Analysis!** PCA only considers the common informatio
#'
#' ## Exploratory Factor Analysis
#'
#' * **Model 1**: No rotation
#' * **Model 2**: Rotation Varimax
#' * **Model 3**: Rotation Oblimin

# No rotation
df_factor <- factanal(df, factors = 4, rotation = "none", scores=c("regression"), fm = "ml")
# Rotation Varimax
df_factor_var <- factanal(df, factors = 4, rotation = "varimax", scores=c("regression"), fm =
# Rotiation Oblimin
df_factor_obl <- factanal(df, factors = 4, rotation = "oblimin", scores=c("regression"), fm =

#'
#' Let's print out the results of `df_factor_obl`, and have a look.
print(df_factor, digits=2, cutoff=0.3, sort=TRUE) #cutoff of 0.3 due to the sample size is hi

#'
#' > **Note:**
#' The variability contained in the factors = Communality + Uniqueness.
#' Varimax assigns orthogonal rotation, and oblimin assigns oblique rotation.
#'
#'
#' Plot factor 1 against factor 2, and compare the results of different rotations
#'
#' * **No Rotation**
plot(
  df_factor$loadings[, 1],
  df_factor$loadings[, 2],
  xlab = "Factor 1",
  ylab = "Factor 2",
  ylim = c(-1, 1),
  xlim = c(-1, 1),
```

```
  main = "No rotation"
)
abline(h = 0, v = 0)
load <- df_factor$loadings[, 1:2]
text(
  load,
  names(df),
  cex = .7,
  col = "blue"
)

#'
#' * **Varimax rotation**
plot(
  df_factor_var$loadings[, 1],
  df_factor_var$loadings[, 2],
  xlab = "Factor 1",
  ylab = "Factor 2",
  ylim = c(-1, 1),
  xlim = c(-1, 1),
  main = "Varimax rotation"
)
abline(h = 0, v = 0)
load <- df_factor_var$loadings[, 1:2]
text(
  load,
  labels = names(df),
  cex = .7,
  col = "red"
)

#'
#'
#' * **Oblimin Rotation**
plot(
  df_factor_obl$loadings[, 1],
  df_factor_obl$loadings[, 2],
  xlab = "Factor 1",
  ylab = "Factor 2",
  ylim = c(-1, 1),
  xlim = c(-1, 1),
  main = "Oblimin rotation"
```

```
)
abline(h = 0, v = 0)
load <- df_factor_obl$loadings[, 1:2]
text(
  load,
  labels = names(df),
  cex = .7,
  col = "darkgreen"
)

#'
#' When you have more than two factors it is difficult to analyse the factors by the plots. W
#'
```

Copy the R code above in your session.

8 Cluster Analysis

8.1 Do it yourself with R

```
#'
#' #### Example exercise: Airports
#'
#' **Your task**: Create and evaluate the many types of clustering methods.
#'
#' #### 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 in each airport;
#' * `Mainairlineflightspercentage`: Percentage of flights of the main airline of each airpor
#' * `Maximumpercentageoftrafficpercountry`: Maximum percentage of flights per country;
#' * `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 airpo
#' * `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 vistors that arrived in the airport;
#' * `GDP corrected`: Corrected value of the Gross Domestic Product;
#' * `Cargoton`: Cargo ton. The total number of cargo transported in a certain period multipl
#'
#' ## Startup
#' ##### Import Libraries
```

```r
library(readxl) # Reading excel files
library(skimr) # Summary statistics
library(tidyverse) # Pack of useful tools
library(mclust) # Model based clustering
library(cluster) # Cluster analysis
library(factoextra) # Visualizing distances

#'
#' ##### Import dataset as a dataframe
dataset <- read_excel("data/Data_Aeroports_Clustersv1.xlsX")
df <- data.frame(dataset)

#'
#' ## Get to know your data
#' ##### Summary statistics
skim(df)

#' ##### Sneak the plot
#' Now let us plot an example and take a look
plot(Numberofairlines ~ Destinations, df) #plot
text(Numberofairlines ~ Destinations, df, label = Airport, pos = 4, cex = 0.6) #labels over t

#'
#' By looking at the plot, you may already have a clue on the number of clusters with this tw
#'
#' ### Prepare data before performing a cluster analysis
#' ##### Deal with missing data
table(is.na(df))

#' In this example we do not have missing values. In case you do have in the future, you can
#'
#' ##### Continuous variables
#' Leave only continuous variables, and make `Ordem` as the row ID variable
df_reduced = df[,!(names(df) %in% c("Code","Airport"))]
df_reduced = data.frame(df_reduced, row.names = 1) #Ordem is the 1st variable in the df

#'
#' Take a look at the scale of the variables. See how they are different!
head(df_reduced)

#'
#' ##### Standardize variables
```

```r
#' Z-score standardization:  $(x_{i} - x_{\text{mean}}) / {\sigma}$
mean <- apply(df_reduced, 2, mean) # The "2" in the function is used to select the columns. I
sd <- apply(df_reduced, 2, sd)
df_scaled <- scale(df_reduced, mean, sd)

#'
#' ## Hierarchical Clustering
#'
#' ##### Measuring Similarity through Euclidean distances
#'
distance <- dist(df_scaled, method = "euclidean")

#'
#' > **Note:** There are other forms of distance measures that can be used such as:
#'   i) Minkowski distance;
#'   ii) Manhattan distance;
#'   iii) Mahanalobis distance.
#'
#' ##### Visualize distances in heatmap
#'
fviz_dist(distance, gradient = list(low = "#00AFBB", mid = "white", high = "#FC4E07"), order

#'
#' #### Types of hierarchical clustering
#' There are many types of hierarchical clustering. Let's explore some.
#'
#' **1. Single linkage (nearest neighbor) clustering algorithm**
#'
#' Based on a bottom-up approach, by linking two clusters that have the closest distance betw
models <- hclust(distance, "single")
plot(models, labels = df$Airport, xlab = "Distance - Single linkage", cex=0.6, hang = -1)

rect.hclust(models, 4, border = "purple") # Visualize the cut on the dendogram, with 4 cluste

#'
#'
#' **2. Complete linkage (Farthest neighbor) clustering algorithm**
#'
#' Complete linkage is based on the maximum distance between observations in each cluster.
modelc <- hclust(distance, "complete")
plot(modelc, labels = df$Airport, xlab = "Distance - Complete linkage", cex=0.6, hang = -1)
rect.hclust(modelc, 4, border = "blue")
```

```
#'
#' **3. Average linkage between groups**
#'
#' The average linkage considers the distance between clusters to be the average of the dista
modela <- hclust(distance, "average")
plot(modela, labels = df$Airport, xlab = "Distance - Average linkage", cex=0.6, hang = -1)
rect.hclust(modelc, 4, border = "red")

#'
#' **4. Ward`s method**
#'
#' The Ward`s method considers the measures of similarity as the sum of squares within the cl
modelw <- hclust(distance, "ward.D2")
plot(modelw, labels = df$Airport, xlab = "Distance - Ward method", cex=0.6, hang = -1)
rect.hclust(modelw, 4, border = "orange")

#'
#' **5. Centroid method**
#'
#' The centroid method considers the similarity between two clusters as the distance between
modelcen <- hclust(distance, "centroid")
plot(modelcen, labels = df$Airport, xlab = "Distance - Centroid method", cex=0.6, hang = -1)
rect.hclust(modelcen, 4, border = "darkgreen")

#'
#' ##### Comparing results from different hierarchical methods
#' Now lets evaluate the **membership** of each observation with the `cutree` function for ea
#'
member_single <- cutree(models, 4)
member_com <- cutree(modelc, 4)
member_av <- cutree(modela, 4)
member_ward <- cutree(modelw, 4)
member_cen <- cutree(modelcen, 4)

#'
#'
#' Compare how common each method is to each other.
table(member_com, member_av) # compare the complete linkage with the average linkage

#'
#' > **Note:** Try comparing other methods, and evaluate how common they are.
#'
```

```r
#' ##### Silhouette Plots: evaluate which method is more appropriate
#' The silhouette plot evaluates how similar an observation is to its own cluster compared to
#'
plot(silhouette(member_single, distance))
plot(silhouette(member_com, distance))
plot(silhouette(member_av, distance))
plot(silhouette(member_ward, distance))
plot(silhouette(member_cen, distance))

#'
#'
#' ## Non-Hirarchical Clustering
#'
#' ##### K-means clustering
#' * k-means with n=3 clusters
km_clust <- kmeans(df_scaled, 3)
km_clust #print the results

#'
#' * Other ways of setting the number of clusters
#'
#' This algorithm will detect how many clusters from 1 to 10 explains more variance
#'
  k <- list()
  for(i in 1:10){
    k[[i]] <- kmeans(df_scaled, i)
  }

#'
#' > **Note**: Try printing the k value and take a look at the ratio `between_SS` / `total_SS
#'
#' Now, lets  plot `between_SS` / `total_SS` into a scree plot
#'
betSS_totSS <- list()
for(i in 1:10){
betSS_totSS[[i]] <- k[[i]]$betweenss/k[[i]]$totss
}
plot(1:10, betSS_totSS, type = "b", ylab = "Between SS / Total SS", xlab = "Number of cluster

#'
#'
#' Let take out the outliers and see the difference in the k-means clustering:
```

```
#'
#' * **Examine the boxplots**
#'
par(cex.axis=0.6, mar=c(11,2,1,1))# Make labels fit in the boxplot
boxplot(df_scaled, las = 2) #labels rotated to vertical

#'
#' * **Detect the outliers**
#'
outliers <- boxplot.stats(df_scaled)$out
outliers

#'
#' * **Remove rows with outliers**
#'
nrow(df_scaled) #32
out_ind <- which(df_scaled %in% c(outliers)) #the row.names that contain outliers
df_no_outliers = df_scaled[-c(out_ind),] #remove those rows from the df_scaled
nrow(df_no_outliers) #31

#'
#' > **Note:** There are many methods to treat outliers. This is just one of them. Note that
#'
#'
#' Execute a k-means clustering with the dataset without the outliers and see the difference
#'
km_no_outliers <- kmeans(df_no_outliers, 3)
km_no_outliers

#'
#' #### Ploting the clusters
#' Finally, plotting the clusters results to check if they make sense.
#' Let us go back to first example and take a look.
#'
#' * **K-means with outliers**
#'
plot(Numberofairlines ~ Destinations, df, col = km_clust$cluster)
with(df, text(Numberofairlines ~ Destinations, label = Airport, pos = 1, cex = 0.6))

#'
#' * **K-means without outliers**
#'
```

```r
plot(Numberofairlines ~ Destinations, df, col = km_no_outliers$cluster)
with(df, text(Numberofairlines ~ Destinations, label = Airport, pos = 1, cex = 0.6))
```

Copy the R code above in your session.

Part II

GIS in R

9 Introduction to spatial data

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

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

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

```
library(sf)
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).

9.1 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)
```

9.1.1 Projected vs Geographic Coordinate Systems

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

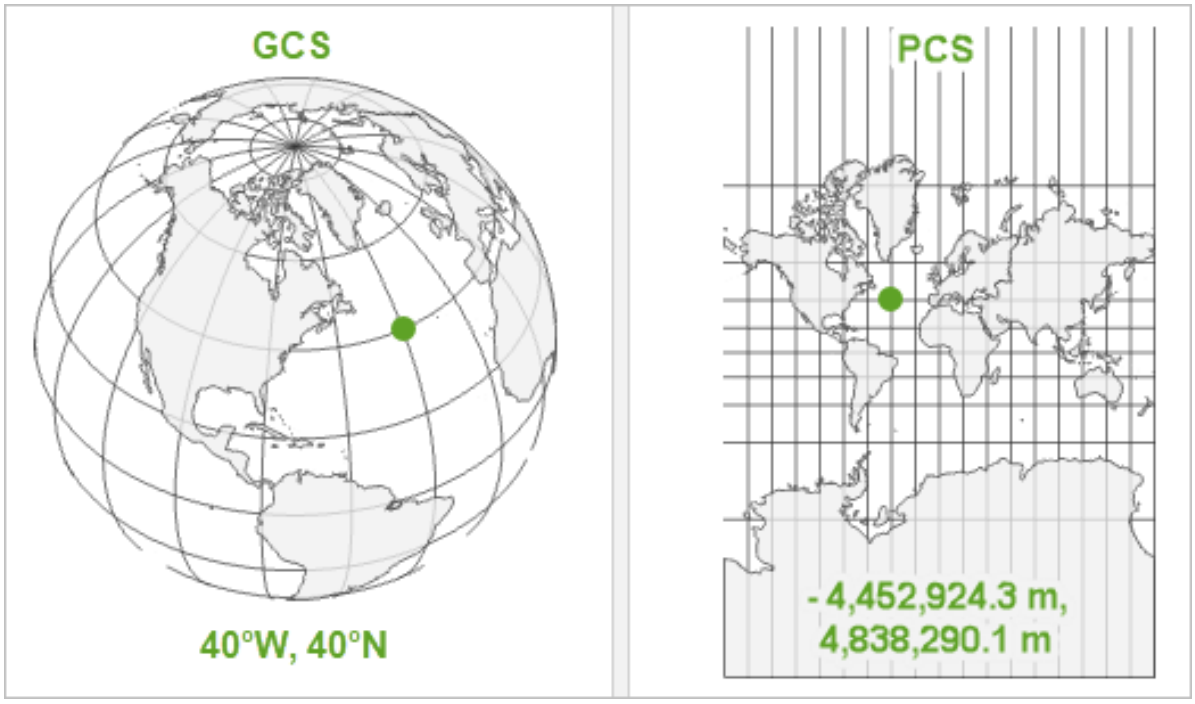


Figure 9.1: Source: ESRI

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 EPGS:**3857** is the code for the Pseudo-Mercator coordinate system.

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

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

9.2 Join geometries to data frames

Import `TRIPSmun.Rds` file and check data class

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

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

```r
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.

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

class(TRIPSgeo)
```

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

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

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

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

Now we have a spatial feature with the data frame.

9.3 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.

```r
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"
```

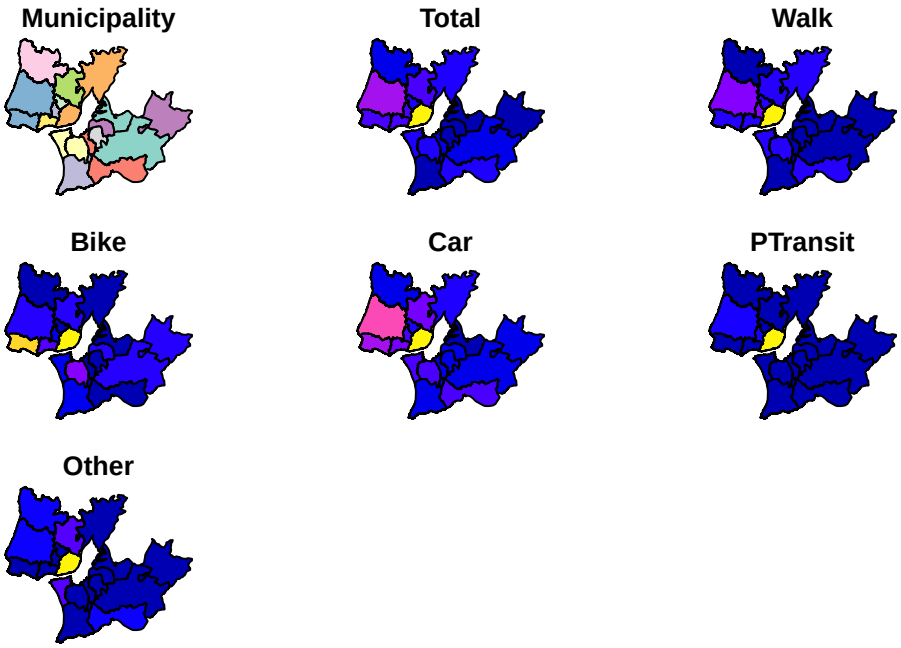
We can also set the **crs** of the spatial feature on the fly.

Check the differences between both data variables.

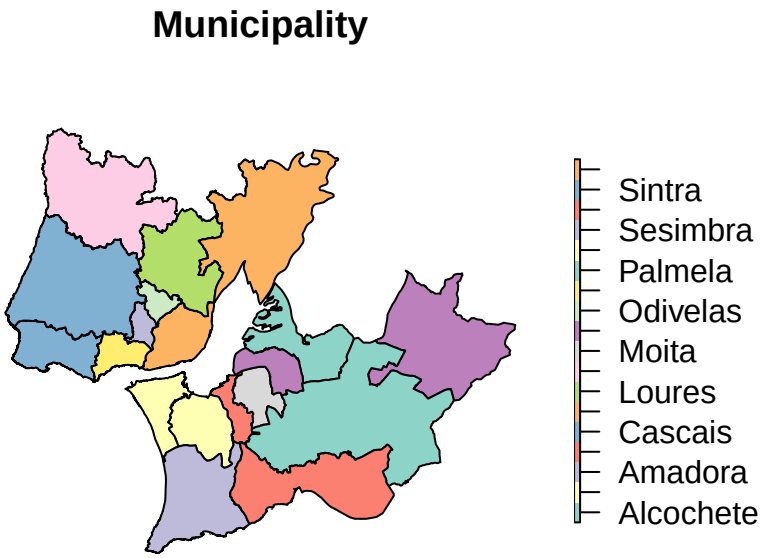
9.4 Visuzlize spatial data

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

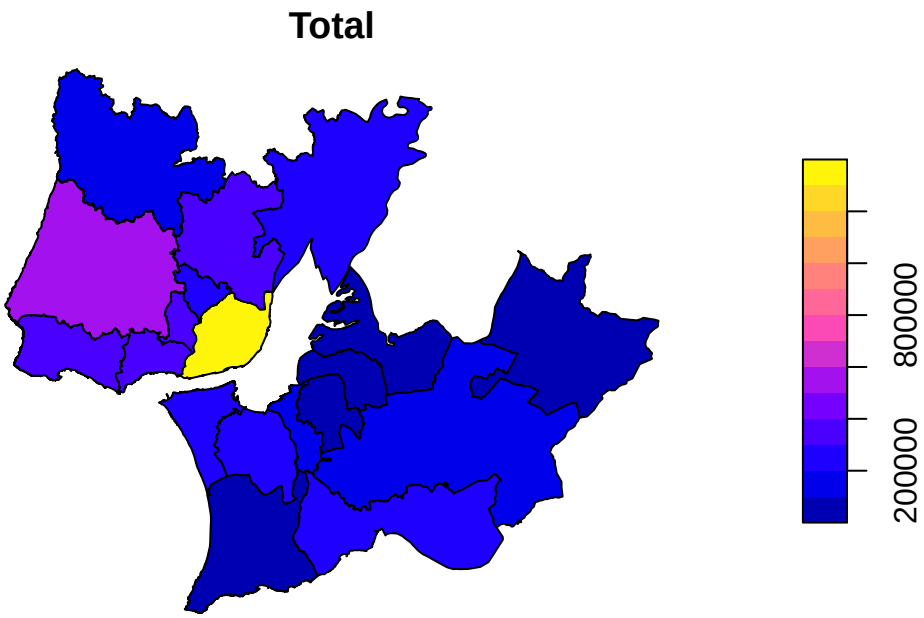
```
plot(TRIPSgeo) # all variables
```



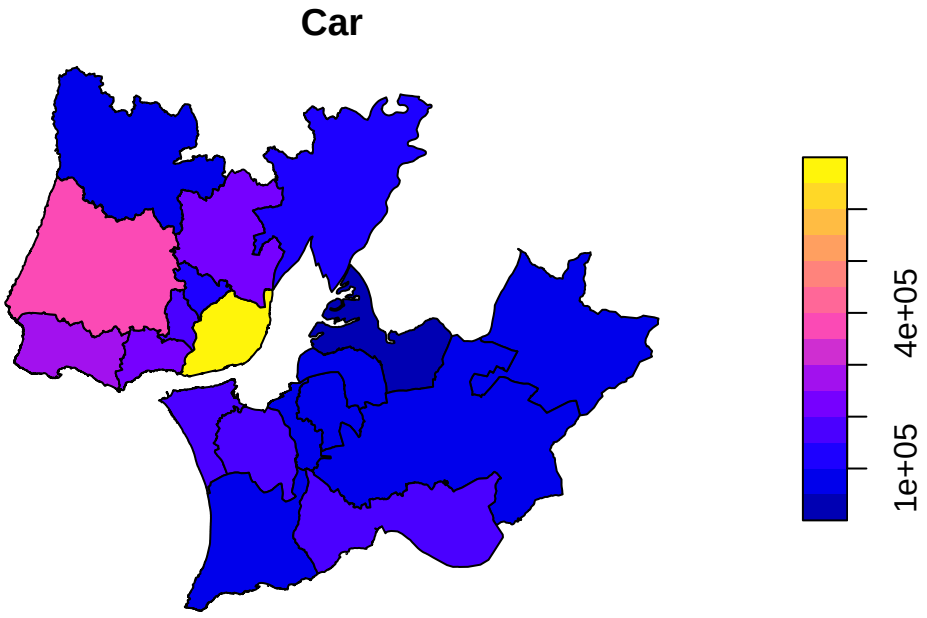

```
plot(TRIPSgeo["Municipality"])
```



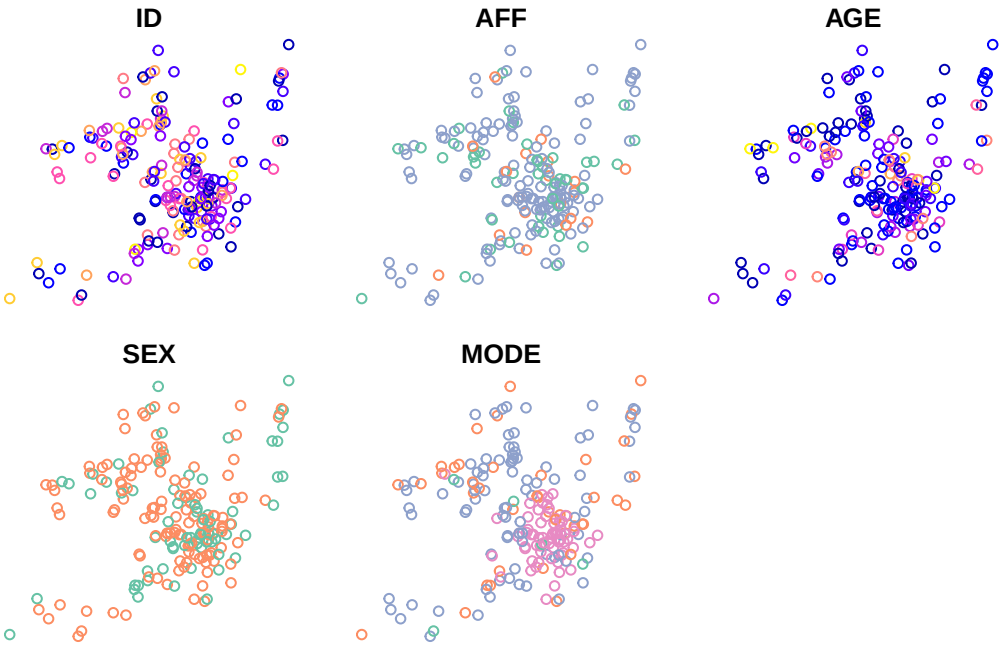

```
plot(TRIPSgeo["Total"])
```

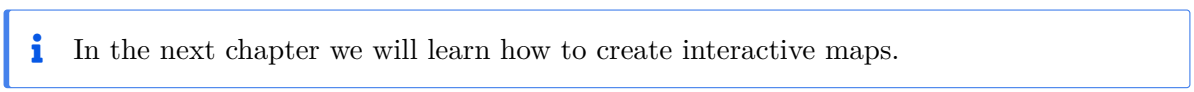



```
plot(TRIPSgeo["Car"])
```




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





9.5 Export spatial data

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

This is also useful to convert spatial data between formats.

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

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

10 Centroids of transport zones

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

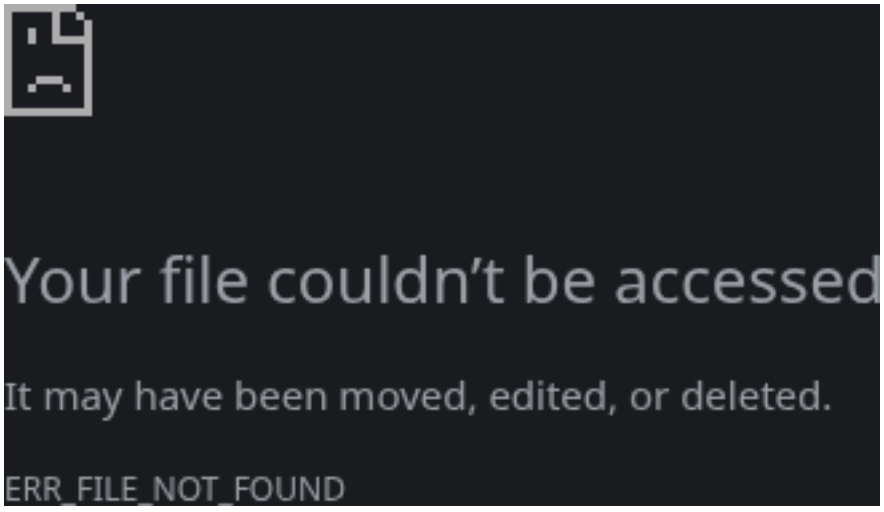
10.1 Geometric centroids

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

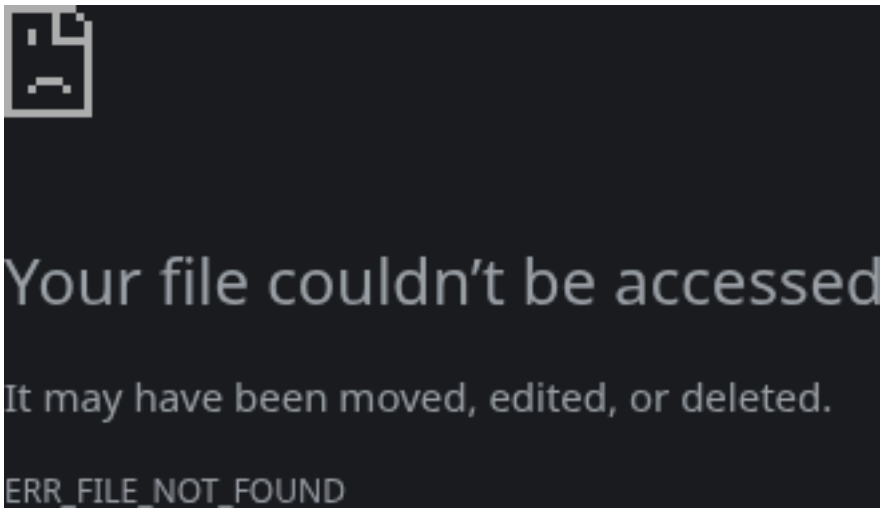
```
library(dplyr)
library(sf)
library(mapview)

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)
```



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



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.

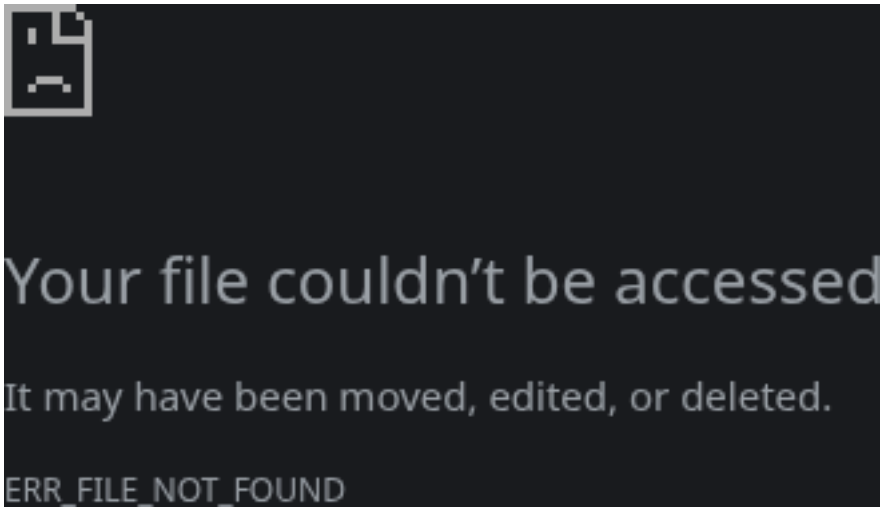
To overcome this, we can use **weighted centroids**.

10.2 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.

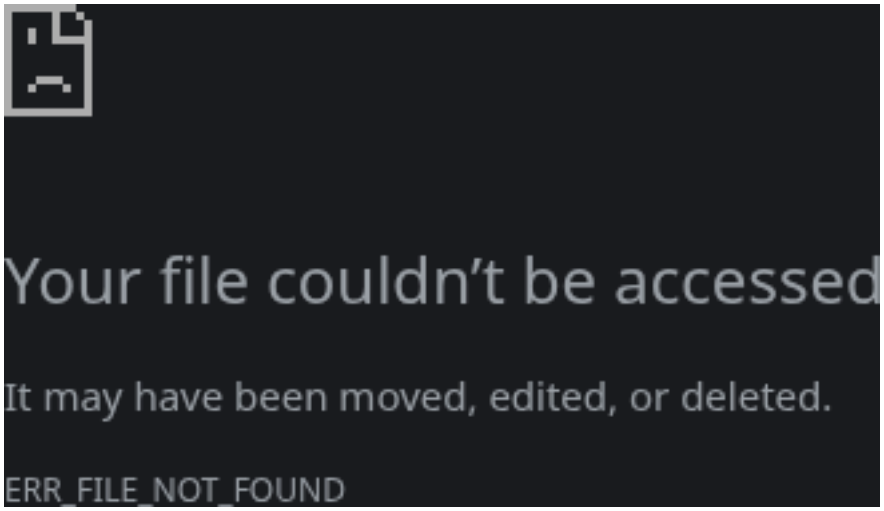
```
Centroid_build = Census |>
  mean_center(group = "Municipality", weight = "Buildings")
```

10.3 Compare centroids in a map

10.3.1 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
```

```
file:////tmp/RtmpfdE0cF/file1d50274db176/widget1d5064b89198.html screenshot completed
```



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

10.3.2 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)
```

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

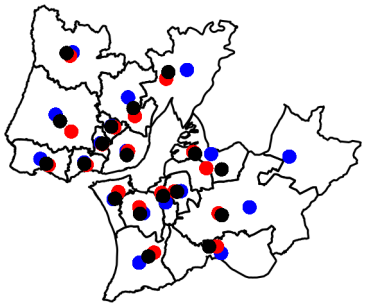


Figure 10.1: Static map of different centroids of Municipalities

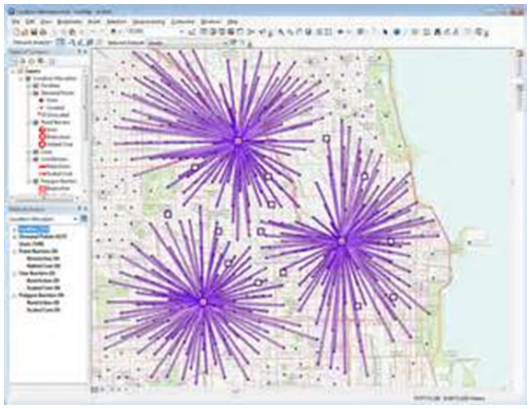
In the next section we will use these centroids to calculate the desire lines between them.

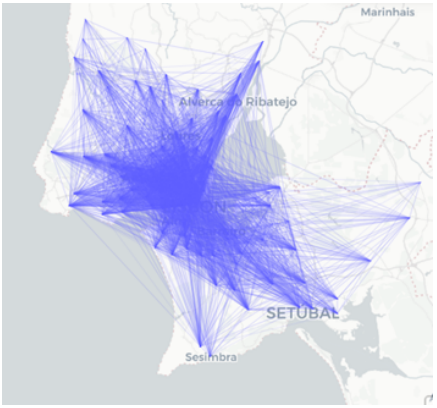
11 OD pairs and desire lines

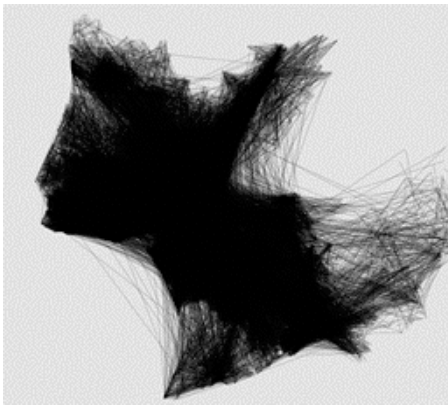
Desire lines are a way to represent the flow of people or goods between two points. They are often used in transport planning to represent the flow of trips between zones.

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

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







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

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

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

11.1 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) # supress mesag
```

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

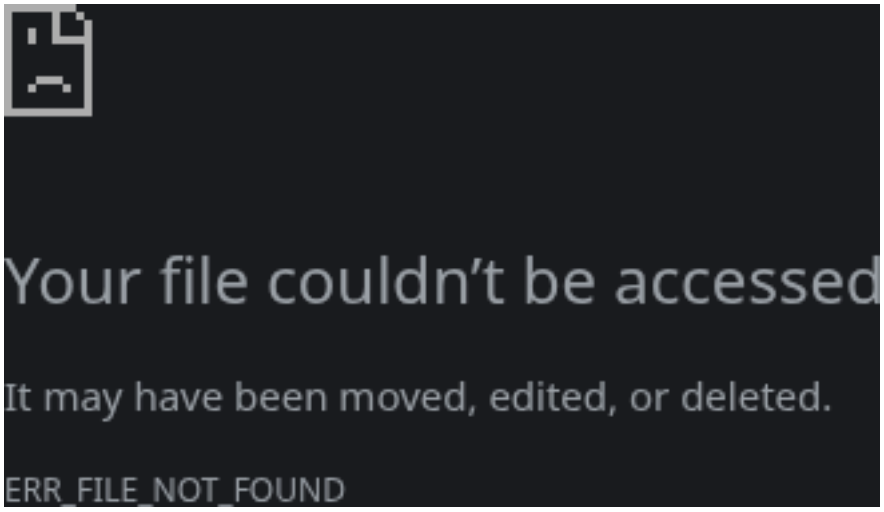
```
# install.packages("od")
library(od)

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

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

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

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

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

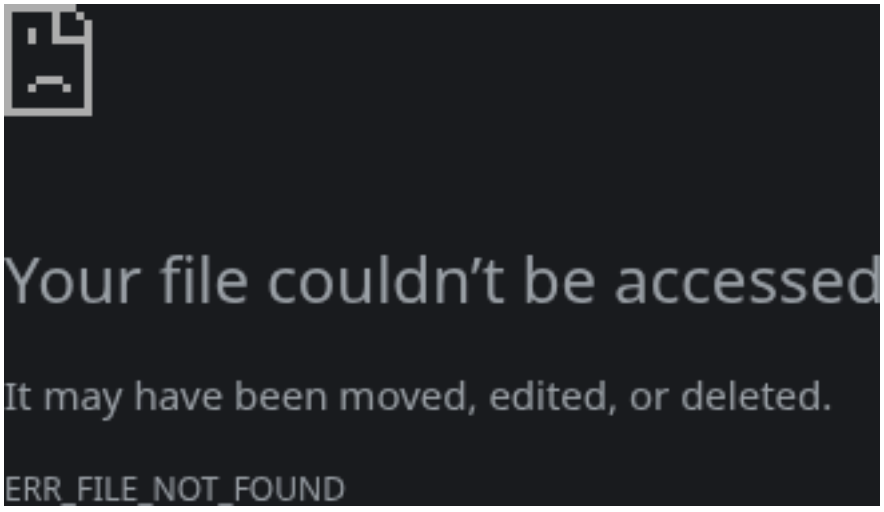


As you can see, this is too much information to be able to understand the flows.

11.1.1 Filtering desire lines

Filter **intrazonal** trips.

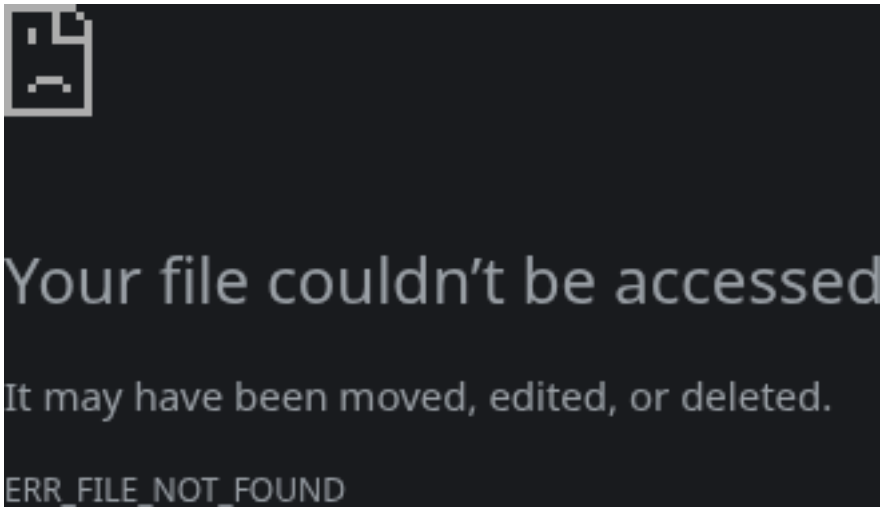
```
library(dplyr)

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

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



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

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

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

```
file:////tmp/RtmpSvWzWQ/file1f6130a8558d/widget1f61221074dd.html screenshot completed
```



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

11.2 Oneway desire lines

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

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

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

```
nrow(TRIPSdlines)
```

```
[1] 315
```

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

```
[1] 150
```

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...
```

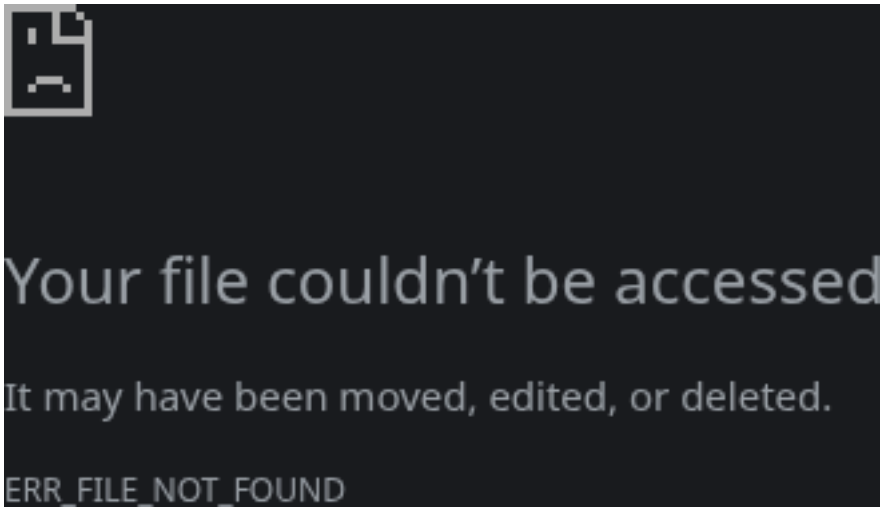
```
tail(TRIPSdlines_oneway[,c(1,2)])
```

```
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)
```

```
file:////tmp/RtmpSvWzWQ/file1f616c7e3f8a/widget1f616fdc9172.html screenshot completed
```



11.3 Using population centroids

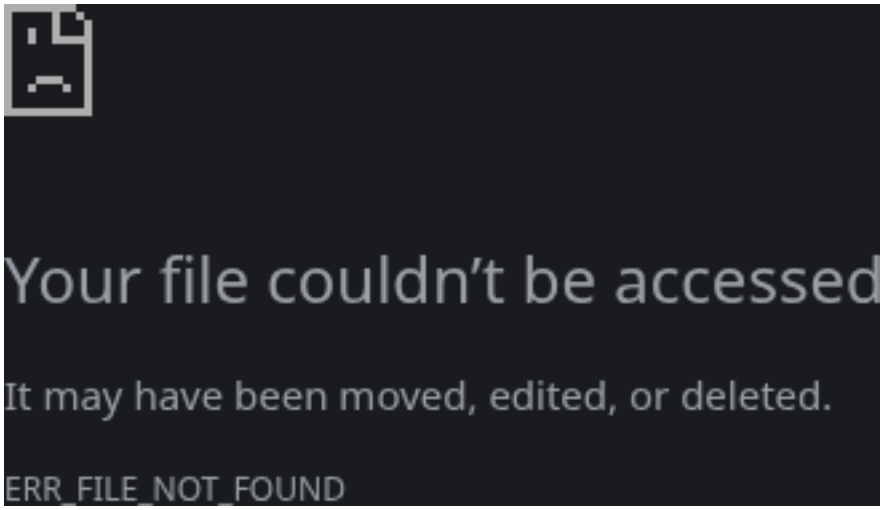
The `od_to_sf()` function uses the geometric center of the zones to create the desire lines. But if we replace those zones by the weighted centroids, we can have a more realistic representation of the flows.

```
# 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:

```r
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")
```

```
file:////tmp/RtmpSvWzWQ/file1f6124ad14e9/widget1f617c48824a.html screenshot completed
```



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

12 Euclidean and routing distances

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

12.1 Euclidean distances

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

12.1.1 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
```

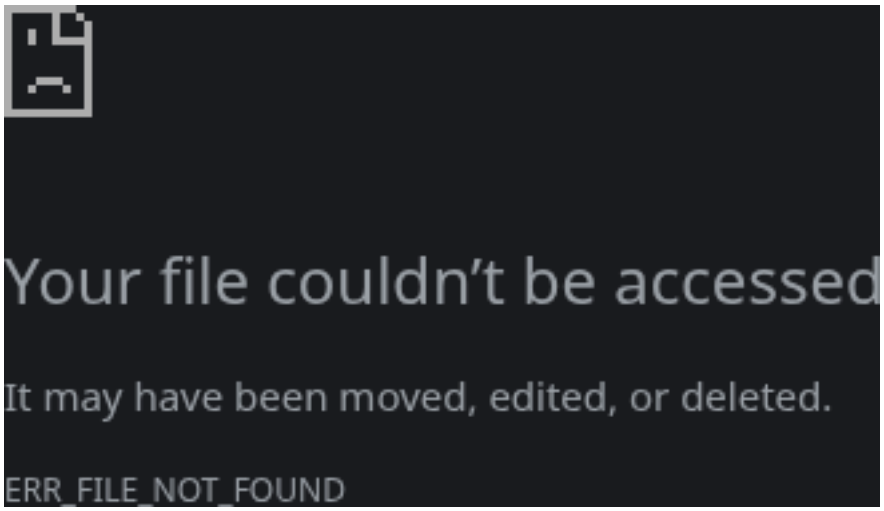
12.1.2 Create new point at the university

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

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

Visualize in a map:

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



12.1.3 Straight lines

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

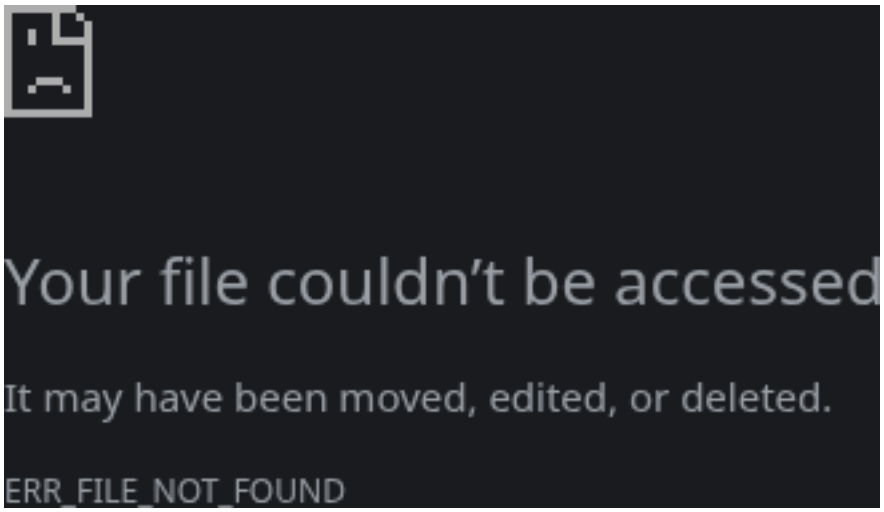
This function finds returns the nearest points between two geometries, and creates a line between them. This can be useful to find the nearest train station to each point, for instance.

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

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

mapview(SURVEYeuclidean)
```

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

```
file:////tmp/Rtmpz58kqB/file21607573cc9c/widget21606b494586.html screenshot completed
```



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.

12.1.4 Distance

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

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

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

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

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

summary(SURVEYgeo$distance)
```

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

`SURVEYgeo` is still a points' sf.

12.2 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 12.1: 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.

12.2.1 Routing distances with `r5r`

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

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

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

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

```
[1] 95
```

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

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

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

12.2.1.1 Distances by car

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

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

names(SURVEYcar)
```

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

The `detailed_itineraries()` function is super detailed!

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

12.2.1.2 Distances by foot

Repeat the same for `WALK`[1].

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

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

12.2.1.3 Distances by PT

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

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

12.3 Compare distances

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

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

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

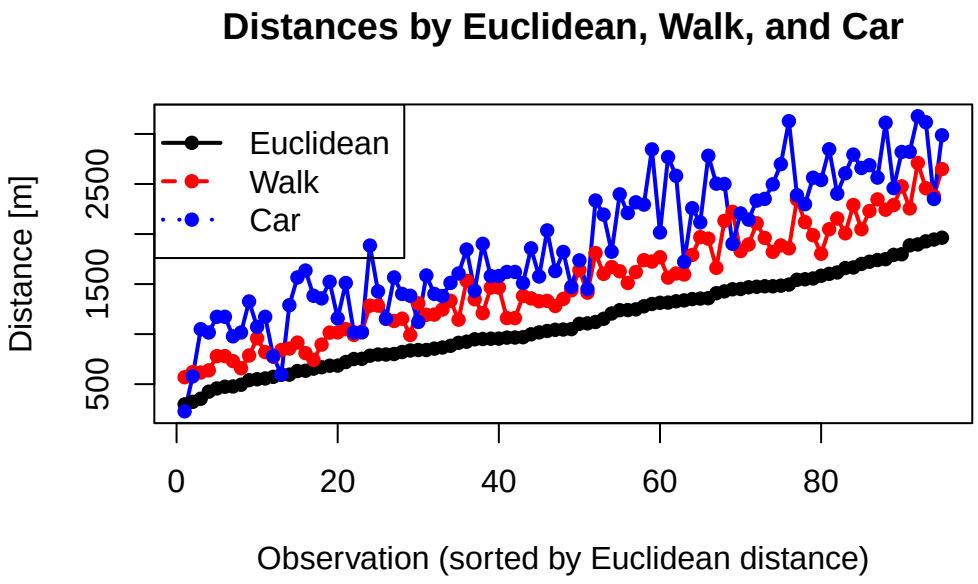
```
summary(SURVEYwalk$distance) # Walk
```

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

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

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

What can you understand from this results?

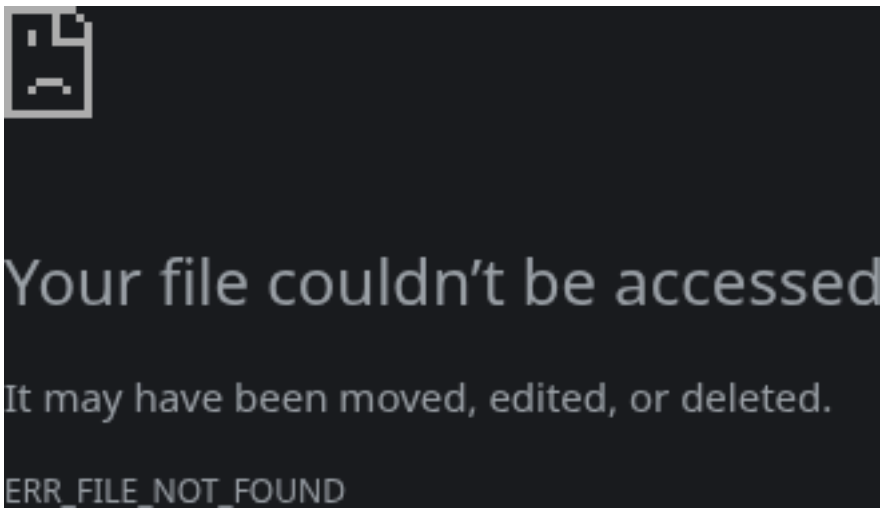


12.3.1 Circuity

Compare 1 single route.

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

```
file:////tmp/Rtmpz58kqB/file2160748fbbf8/widget216012e8a2.html screenshot completed
```



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

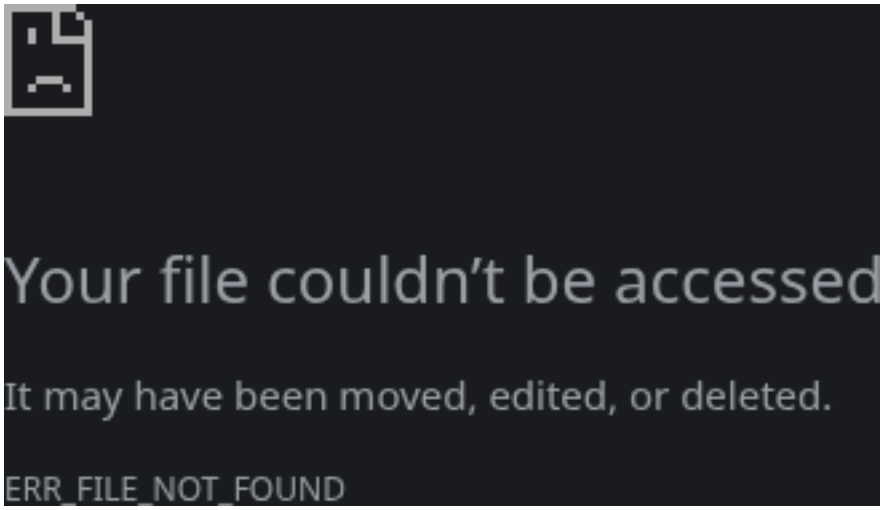
The cicuity for car (1.65) is usually higher than for walking (1.28) or biking, for shorter distances.

12.4 Visualize routes

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

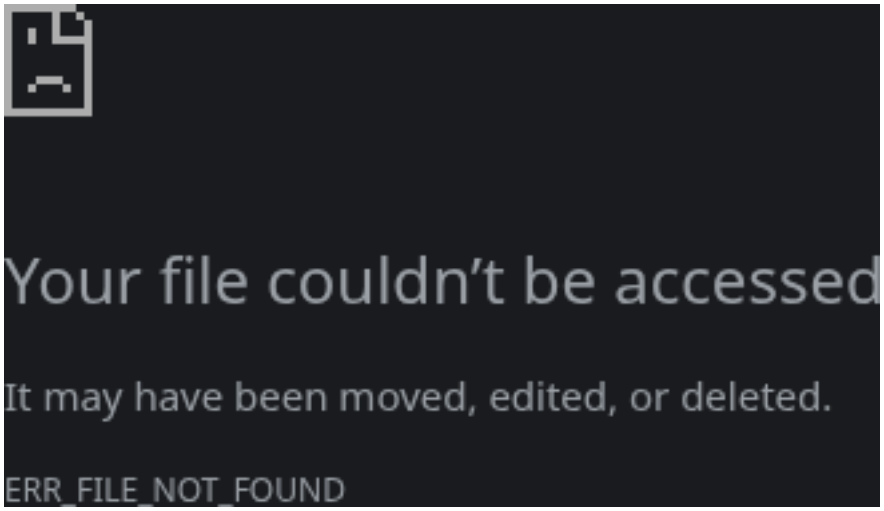
```
mapview(SURVEYwalk, alpha = 0.3)
```

```
file:////tmp/Rtmpz58kqB/file21603292c022/widget21604ea4ec2c.html screenshot completed
```



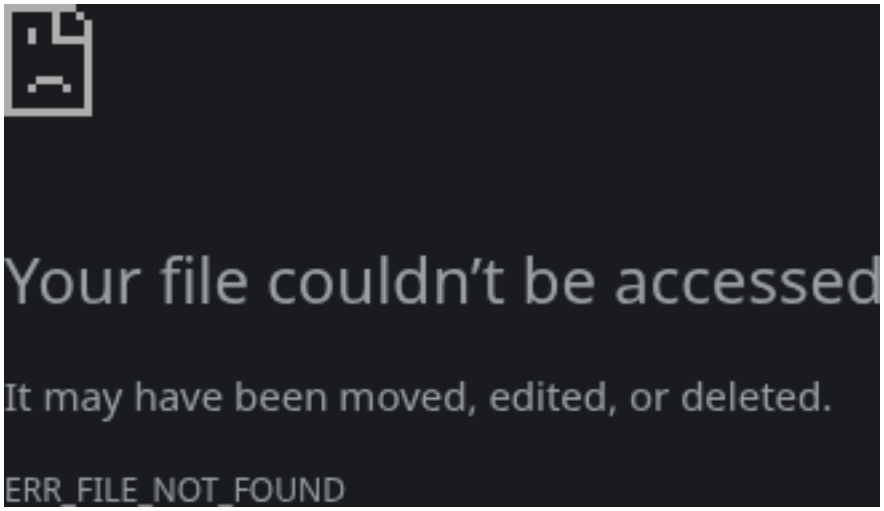

```
mapview(SURVEYcar, alpha = 0.3, color = "red")
```

```
file:////tmp/Rtmpz58kqB/file21604cbba208/widget216014e01dfd.html screenshot completed
```



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

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

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

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



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

[2]Assuming all travel by the shortest path.

13 Open transportation data

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

13.1 Road Networks

13.1.1 OpenStreetMap

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

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

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

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

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

- Overpass API
- Geofabrik

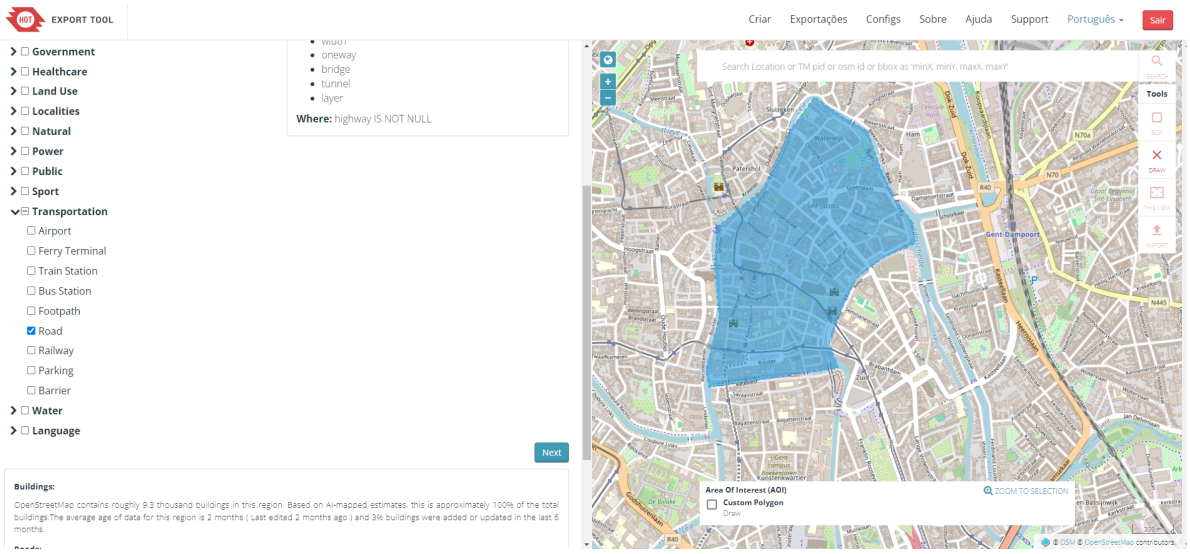
These websites include all the OSM data, with **much more information than you need**.

13.1.2 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[1].

[1] You need an OSM account to use it.

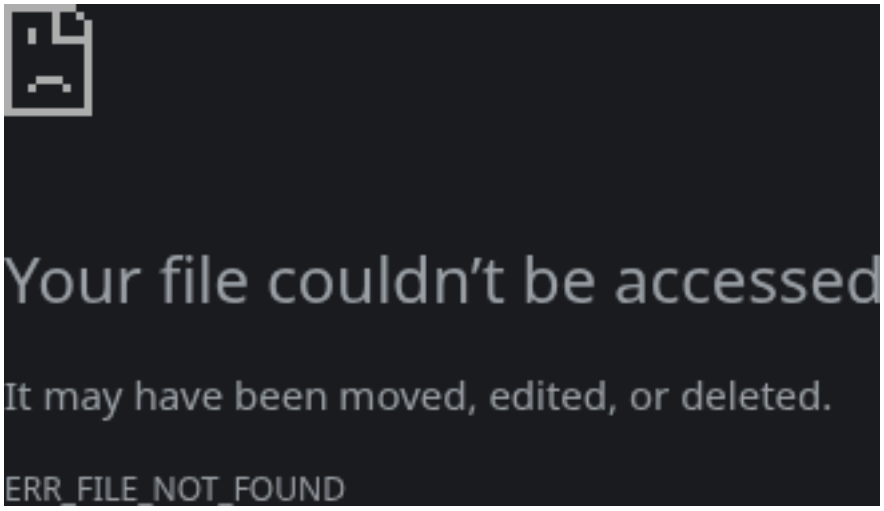


After the export, you can read in R using the `sf` package:

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

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

```
PhantomJS not found. You can install it with webshot::install_phantomjs(). If it is installed
```

```
file:////tmp/RtmpCvwQO3/file23d01651499d/widget23d01e6a62dc.html screenshot completed
```



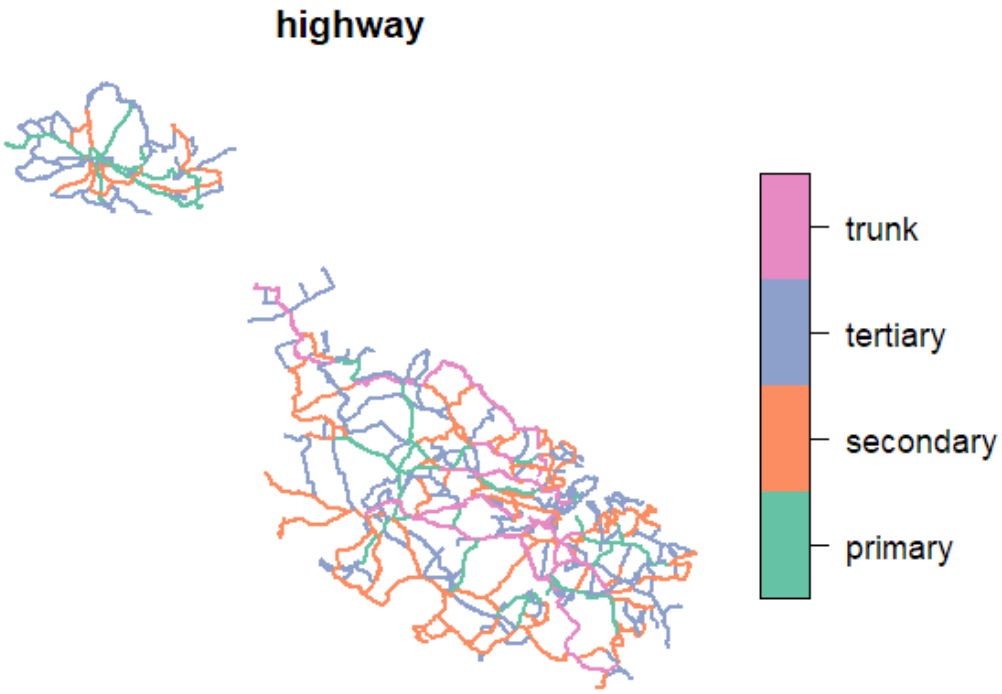
13.1.3 OSM in R

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"])
```



13.2 Transportation Services' Data

13.2.1 GTFS

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

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

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

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

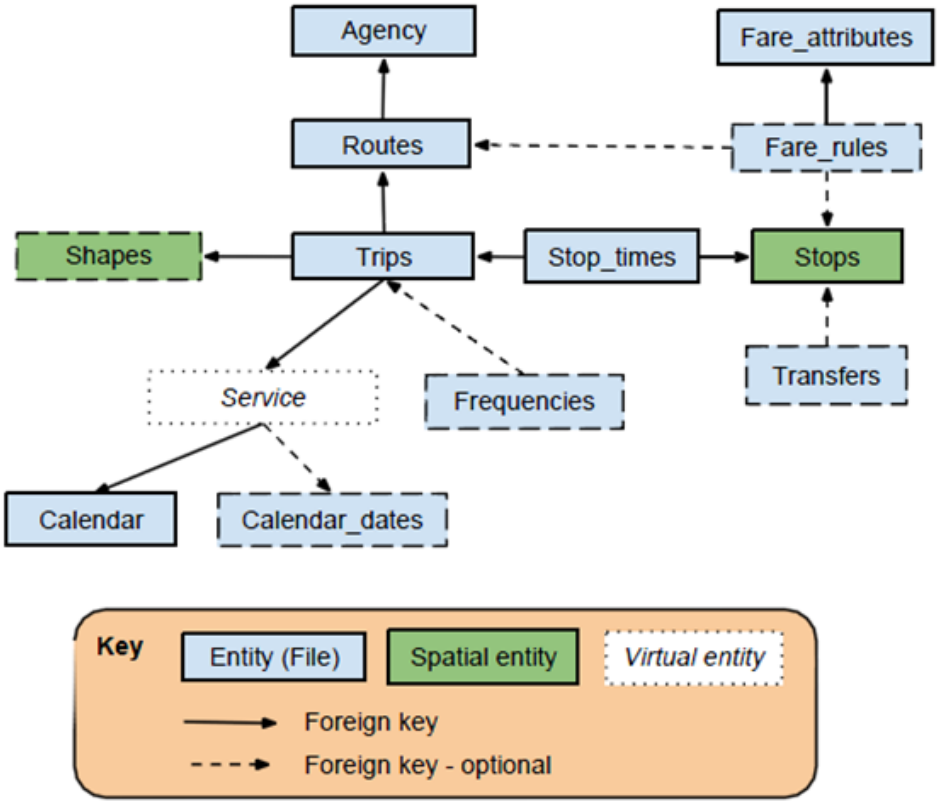


Figure 13.1: Source: trilliumtransit.com

13.2.1.1 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.

13.2.1.2 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 of of them at the time.

13.2.2 National Access Points

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

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

Example of Bus services data in Belgium:

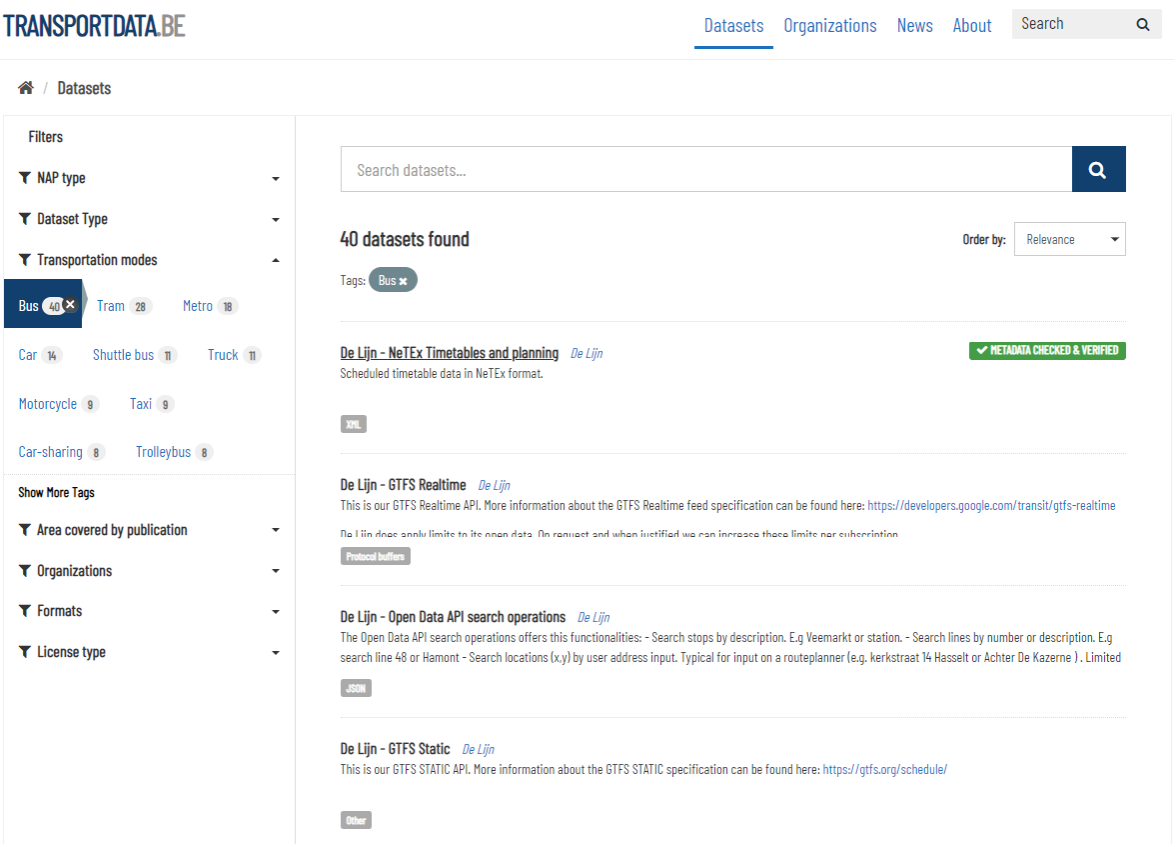


Figure 13.2: Source: Transport Data Belgium

14 Grids

14.1 Make grid

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

MUNICIPIOSgeo = st_read("data/Municipalities_geo.gpkg")
LISBON = MUNICIPIOSgeo |> filter(Municipality == "Lisboa") |> st_transform(3857)
```

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

```
GRID = LISBON |>
  st_make_grid(cellsize = 800, what = "polygons", square = TRUE) |> # 800m, we are using a pr
  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)
```

14.2 Count points in polygons

```
SURVEY = read.csv("data/SURVEY.txt", sep = "\t")
SURVEYgeo = SURVEY |> st_as_sf(coords = c("lon", "lat"), crs = 4326)

SURVEY_with_GRIDid = SURVEYgeo |>
  st_join(GRID,
          join = st_intersects) |>
    st_drop_geometry() |>
```

```
    group_by(id) |>
    summarise(count = n()) |>
    ungroup()

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

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

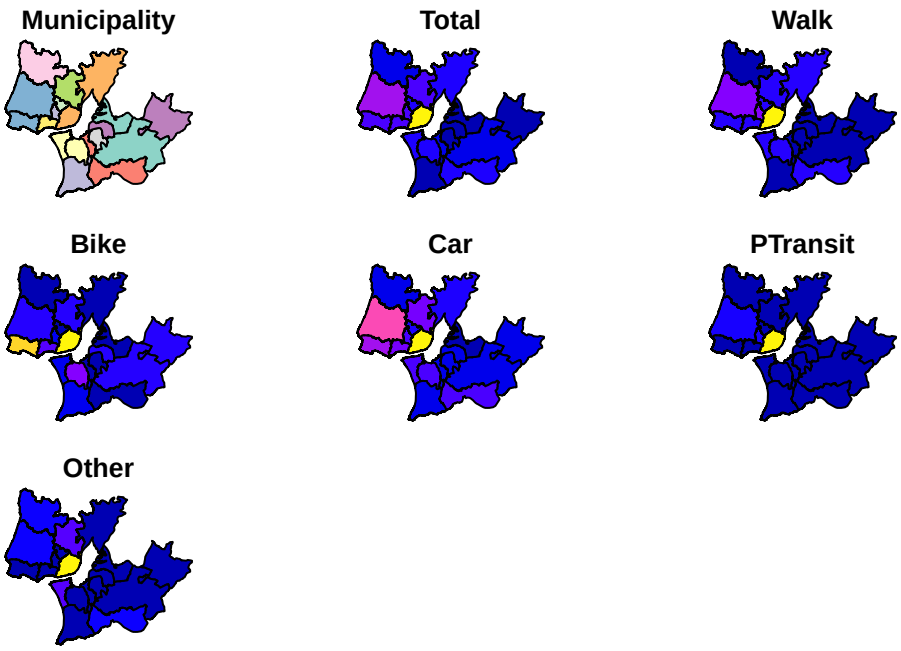
15 Interactive maps

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

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

```
Reading layer `TRIPSgeo' from data source
  `/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)
```



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.

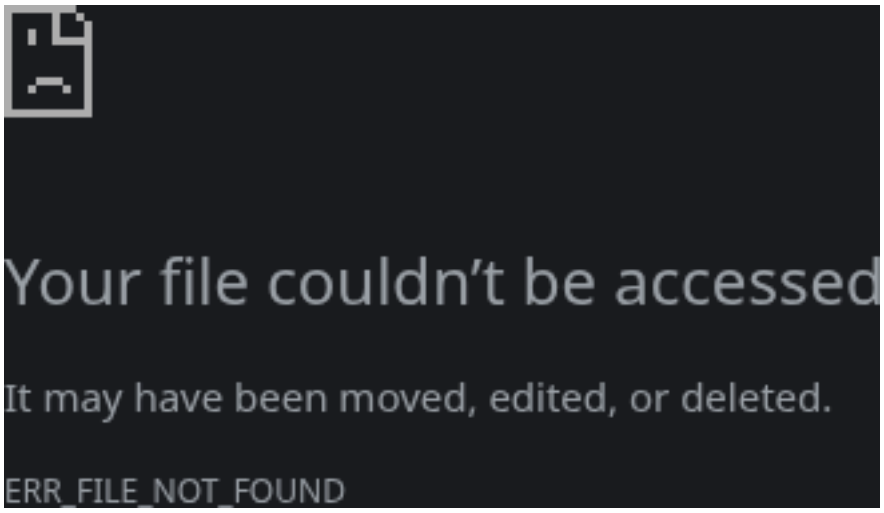
15.1 Mapview

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

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

```
PhantomJS not found. You can install it with webshot::install_phantomjs(). If it is installe
```

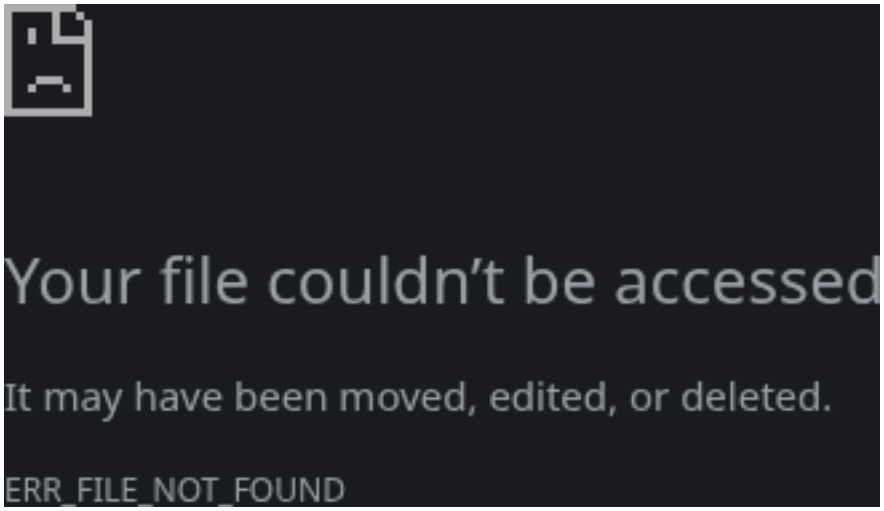
```
file:////tmp/RtmpzsQSl1/file255f2c49803f/widget255f2937675f.html screenshot completed
```



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

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

```
file:////tmp/RtmpzsQSl1/file255f7eb56f86/widget255f7c85c8eb.html screenshot completed
```

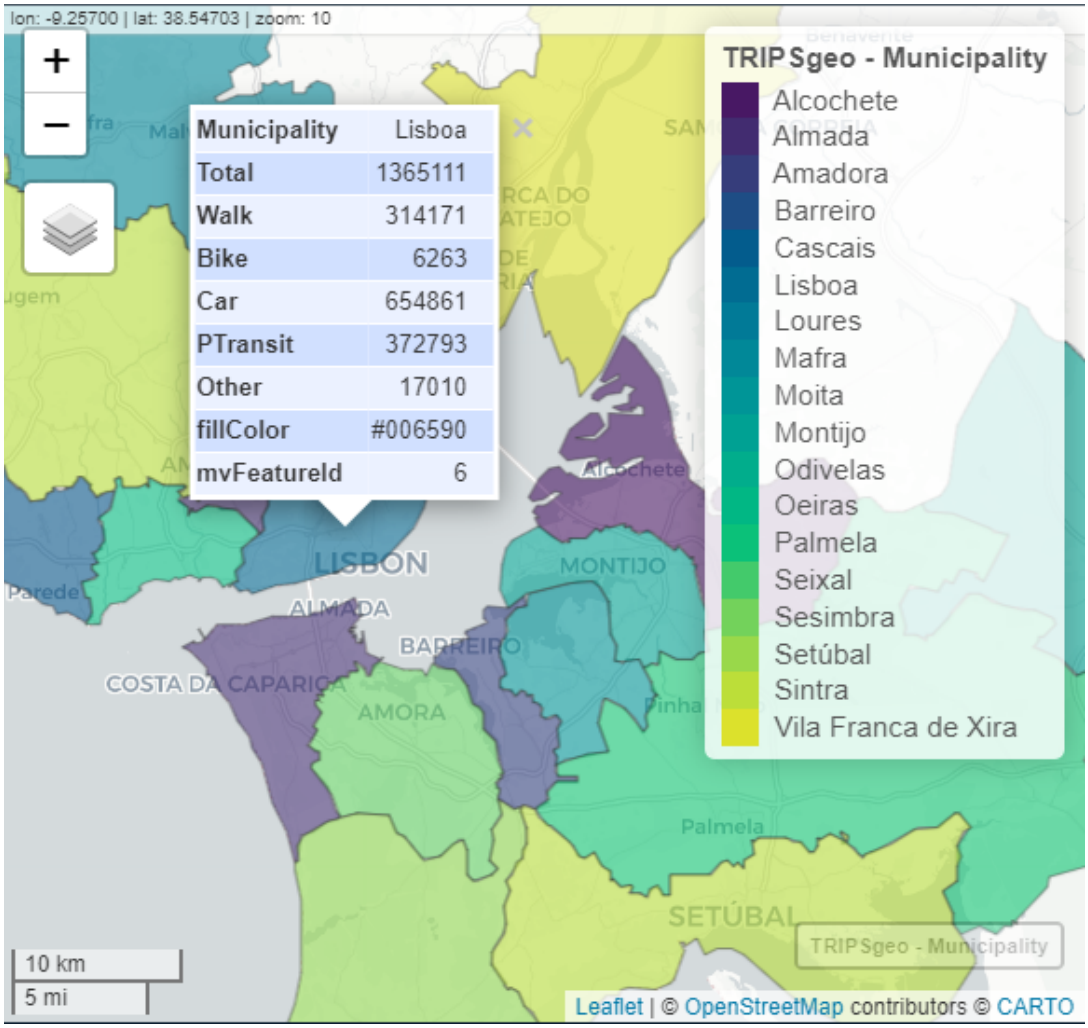


As you can see, a color palette is automatically assigned to the **continuous variable**.

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
        )
```

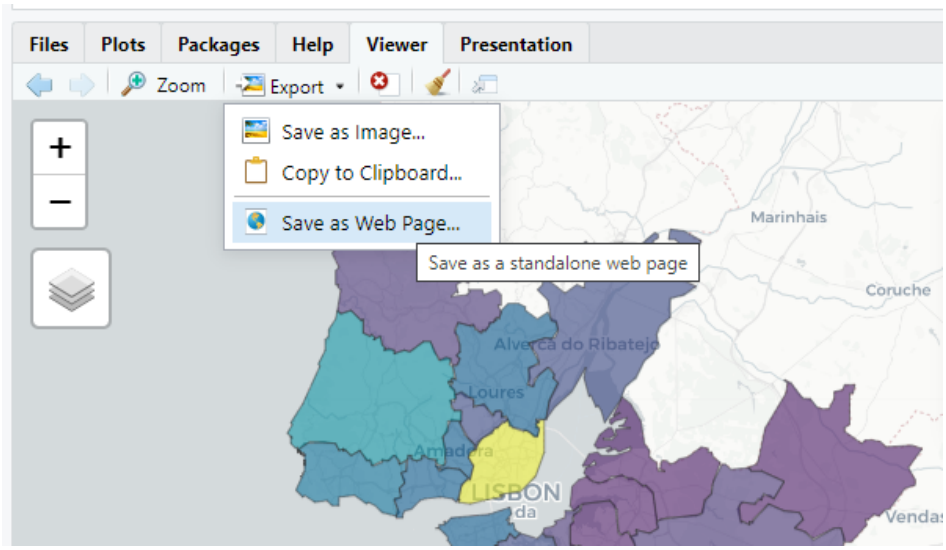
> **i** Note that you can change the **basemap**, and click on the geometries to **see the data** associated with them.



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

15.1.1 Export

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



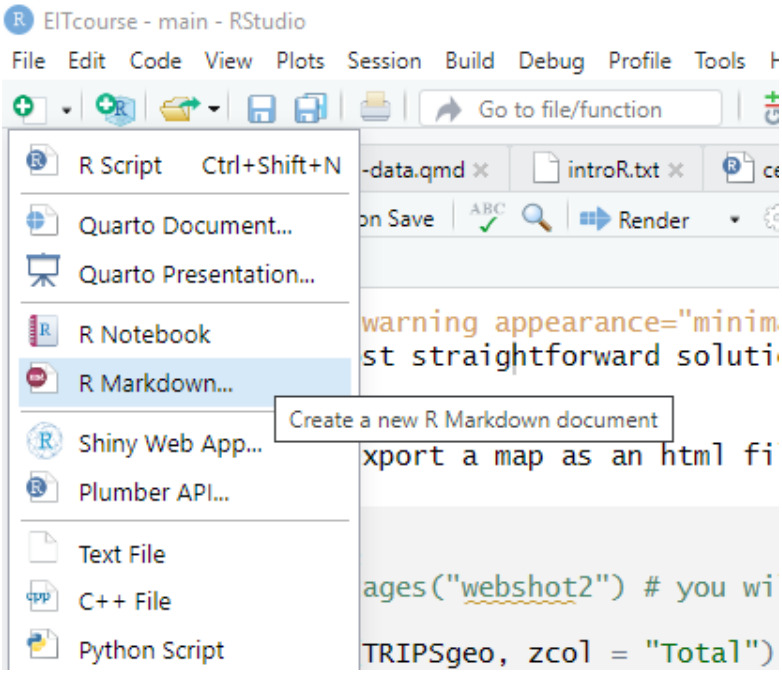
This is the most straightforward solution.

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

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

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

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

15.2 Rmarkdown

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



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

References

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

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

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

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

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

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

Lovelace, Robin, Jakub Nowosad, and Jannes Muenchow. 2024. *Geocomputation with r*. 2nd ed. Chapman; Hall/CRC. https://r.geocompx.org/.

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

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

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/.