Exercise 1 : Setting up GIS in R and GIS basics

2024-03-04

### Key packages

* Geo-computations : sf, terra, raster
* Plotting : ggplot2
* R Project management: here
* Generic data wrangling : dplyr, here, tidyr
* Map of the world : rnaturalearth and rnaturalearth

### Vectors

Geographic vectors data models are based on points located within a Coordinates Reference System (CRS):

* Points can be self standing features
* Two points can be connected to make a linestrings (you can also have multilinestrings)
* Many points can be connected to get a polygon (you can also have multipolygons)
* Many different types of geometries all together (eg. a point, a linestring and a few polygons) to get a geometry collection

You can recognize geo-data because they came with a variable called geometry that contains 2 numbers: longitude and latitude

## SF package

One of the most popular and effective way to deal with vector data is to use simple feature (sf) that supports 17 types of geometries and their combination.

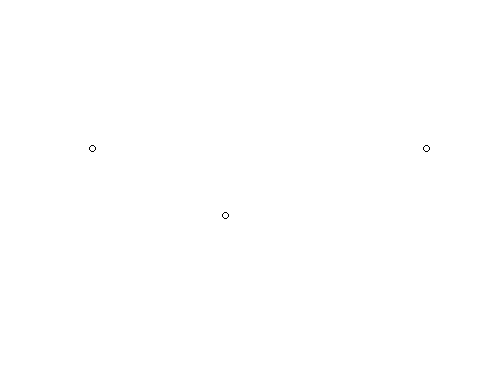
### Let´s create some geometries and look at them

The functions to create simple features geometries (sfg) have very intuitive names:

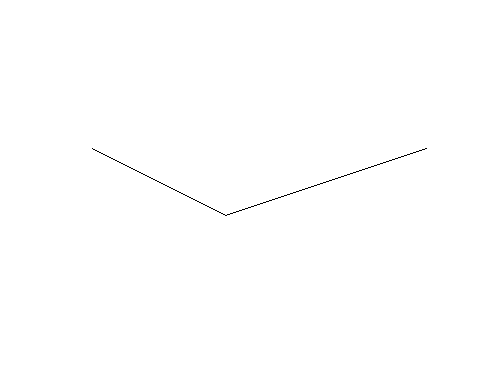
* st\_point()
* st\_linestring()
* st\_polygon()
* st\_multipoint()
* st\_multilinestring()

We can create sfg starting from 3 types of data objects: numeric vector, a matrix , a list depending on which shape we want to generate

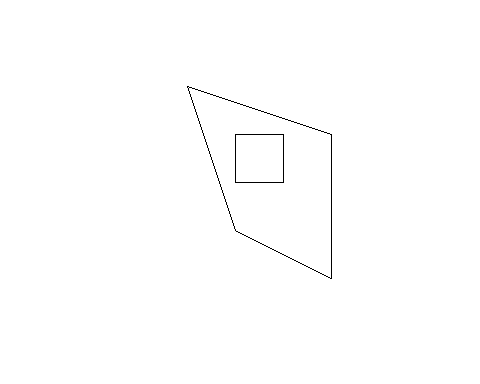
## Points , starting with numeric vectors  
  
p\_1 <- st\_point(c(5,2))  
  
p\_2 <- st\_point(c(7,3,5))  
  
p\_3 <- st\_point(c(8,1,4), dim = "XYM")  
  
p\_4 <- st\_point(c(6,3,5,9), dim = "XYZM")  
  
  
## Multipoints , starting from matrices  
  
multi\_point\_matrix <- rbind(c(4,5), c(6,4), c(9,5))  
multipoint <- st\_multipoint(multi\_point\_matrix)  
  
plot(multipoint)



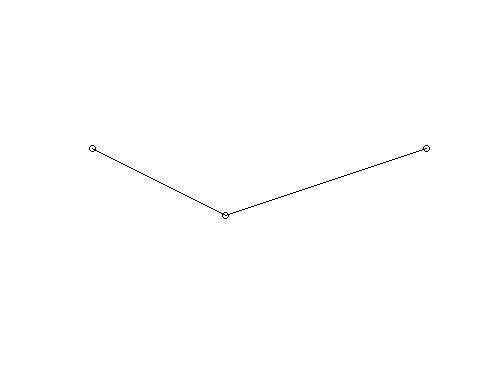
## Linestrings starting from matrices  
  
line\_matrix <- rbind(c(4,5), c(6,4), c(9,5))  
linestring <- st\_linestring(line\_matrix)  
  
plot(linestring)



## Polygons using lists  
  
  
border <- rbind(c(1,5), c(2,2),c(4,1),c(4,4),c(1,5))  
   
hole <-rbind(c(2,4),c(3,4),c(3,3),c(2,3), c(2,4))  
  
poly\_with\_hole <- list(border, hole)  
  
polygon <- st\_polygon(poly\_with\_hole)  
  
plot(polygon)



## Geometry Collection   
  
geom\_list <- list(multipoint, linestring)  
  
geom\_collection <- st\_geometrycollection(geom\_list)  
  
plot(geom\_collection)



## Simple featury columns (sfc)

This is the column of the data were the information on the geometries are saved and where also the crs is saved.

## If we have two geometries with the same features, for eg. p\_1 and p\_2 from above we can put them together with st\_sfc()  
  
points <- st\_sfc(p\_1, p\_2)  
  
## In general to check for which type of geometries is the data you are working on you can always use st\_geometry\_type()  
  
st\_geometry\_type(linestring)

## [1] LINESTRING  
## 18 Levels: GEOMETRY POINT LINESTRING POLYGON MULTIPOINT ... TRIANGLE

## What about Coordinates Reference Systems (crs) ?

As you probably know how we look at spatial information and any type of analysis based on the spatial relative location of different objects is determined by the way that we use to reference the objects into the space. So in fact by how we standardize the map of the world, this type of information is stored into the crs.

If you are working with a spatial object there is a function from the sf package called st\_crs() and if you run it on your data you can see which projection is stored together with the data. The result could as well be NULL. So on the choice of the CRS there is an entire literature but the most widely used in economics is WGS84 that corresponds to the number 4326.

The argument that contains the crs information is either epsg (SRID) or proj4string.

### First let´s look at the deafult (that is NA)   
  
st\_crs(multipoint)

## Coordinate Reference System: NA

multipoint <- st\_sfc(multipoint)  
  
## we can use the st\_set\_crs() function to assign a projection or also assign it using the same st\_crs function:  
  
multi <- st\_set\_crs(multipoint, 4326)  
  
  
st\_crs(multi)

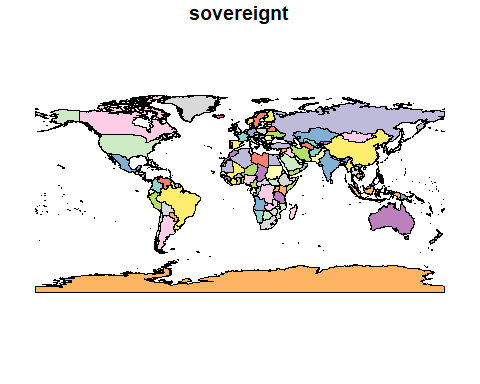
## Coordinate Reference System:  
## User input: EPSG:4326   
## wkt:  
## GEOGCRS["WGS 84",  
## ENSEMBLE["World Geodetic System 1984 ensemble",  
## MEMBER["World Geodetic System 1984 (Transit)"],  
## MEMBER["World Geodetic System 1984 (G730)"],  
## MEMBER["World Geodetic System 1984 (G873)"],  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

Warning : the st\_set\_crs() is a good idea when there is no crs at all, but if you have the “wrong” one using it would just change the value without re-projecting the data

## From one projection to the other

Different projections can produce different worldviews, here we look at it using a default world map from the r natural earth data package and projecting it in two different ways.

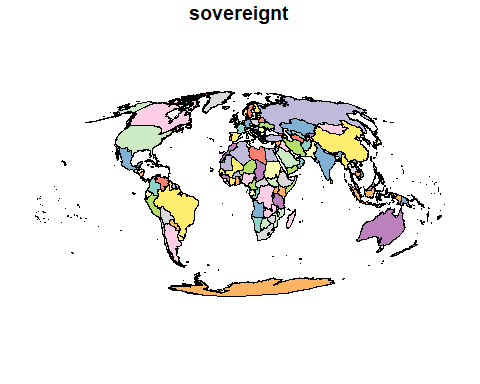
world\_map <- ne\_countries(scale = "medium", returnclass = "sf")  
  
plot(world\_map["sovereignt"])



st\_crs(world\_map)

## Coordinate Reference System:  
## User input: WGS 84   
## wkt:  
## GEOGCRS["WGS 84",  
## DATUM["World Geodetic System 1984",  
## ELLIPSOID["WGS 84",6378137,298.257223563,  
## LENGTHUNIT["metre",1]]],  
## PRIMEM["Greenwich",0,  
## ANGLEUNIT["degree",0.0174532925199433]],  
## CS[ellipsoidal,2],  
## AXIS["latitude",north,  
## ORDER[1],  
## ANGLEUNIT["degree",0.0174532925199433]],  
## AXIS["longitude",east,  
## ORDER[2],  
## ANGLEUNIT["degree",0.0174532925199433]],  
## ID["EPSG",4326]]

## We see that is WGS84 and we want to change to crs to an alternative one for viz purposes, we use st\_transform  
  
another\_wolrd <- st\_transform(world\_map, crs= "ESRI:54009")  
  
plot(another\_wolrd["sovereignt"])



### Data on Dresden geography, water, forest, markets etc..

We will use data from the Open data portal of the municipality of Dresden : [https://opendata.dresden.de/informationsportal/#app/mainpage/0%20{%3E}%20{00003}%20Geographie,%20Geologie,%20Geobasisdaten](https://opendata.dresden.de/informationsportal/#app/mainpage/0%20%7B%3E%7D%20%7B00003%7D%20Geographie,%20Geologie,%20Geobasisdaten)

We download the data in csv and then look at them to see where are the geo-location information stored. If we see that there is a column called geometry or two columns called longitude and latitude then our job is pretty easy and we just need to use the st\_read function from the sf package and the data will be translated in simple features geometries.

## use the read\_csv function to open the data and check the csv before to see which delimiter is used, here is ;  
  
tram\_lines <- read.csv(here("data", "raw", "tram\_lines.csv"),sep=';')  
  
## now we look for the columns name and data structure  
  
head(tram\_lines)

## id schluessel von bis von\_str bis\_str wdm  
## 1 14 1080 10010 10014 Leipziger Str. Marienbrücke B  
## 2 2 1020 10001 10002 Turnerweg Erna-Berger-Str. B  
## 3 3 1030 10002 10003 Erna-Berger-Str. Schlesischer Platz B  
## 4 4 1040 10003 10004 Schlesischer Platz Hansastraße/Hainstr. B  
## 5 7 1050 10004 10007 Hainstr. Eisenbahnstr. B  
## 6 10 1070 10007 10010 Eisenbahnstr. Leipziger Str. B  
## wdm\_klar strasse status historie straba laenge aend  
## 1 Bundesstraße Antonstraße 20 AKT G 251.34 2024-02-06  
## 2 Bundesstraße Antonstraße 0 AKT S 88.07 2024-02-06  
## 3 Bundesstraße Antonstraße 0 AKT S 94.19 2024-02-06  
## 4 Bundesstraße Antonstraße 20 AKT G 125.93 2024-02-06  
## 5 Bundesstraße Antonstraße 20 AKT G 170.24 2024-02-06  
## 6 Bundesstraße Antonstraße 20 AKT G 117.29 2024-02-06  
## geom  
## 1 SRID=4326;LINESTRING(13.73706089957408416069 51.0632117303521297913,13.73699979936916548695 51.06317220746124974085,13.73581990711522848869 51.06240897275258561194,13.73545034992435276422 51.06216178990851517483,13.73506010499288976234 51.06190728570199155456,13.73453524536658321153 51.06160847800499880123)  
## 2 SRID=4326;LINESTRING(13.74466098226822907691 51.06427023328757996978,13.74406537526516025594 51.06439644679269207472,13.74346976369182371513 51.06452265542366575346)  
## 3 SRID=4326;LINESTRING(13.74346976369182371513 51.06452265542366575346,13.74299786220141861293 51.064578993355894454,13.74243552099330933913 51.06464944363409586003,13.74215458670344425229 51.06469590822305093525)  
## 4 SRID=4326;LINESTRING(13.74215458670344425229 51.06469590822305093525,13.74146875208955442815 51.06475151029336956299,13.74105424293899346821 51.06477011823741207763,13.74084087048914071261 51.06477708685204675021,13.74064282203832121354 51.0647610572983907673,13.74051851608217589273 51.06473655854279058985,13.74037689518209681694 51.06471334248091409336)  
## 5 SRID=4326;LINESTRING(13.74037689518209681694 51.06471334248091409336,13.74036372000699479656 51.06470785513534593747,13.73998181652420136345 51.06454878541499908806,13.73874566914908790238 51.06410862268154460253,13.73830218364829214295 51.06391913448099018069)  
## 6 SRID=4326;LINESTRING(13.73830218364829214295 51.06391913448099018069,13.73829239101153554259 51.0639134999597672504,13.73766036992455497057 51.06354983618346921048,13.73708071652358775339 51.06322290768205363065,13.73706089957408416069 51.0632117303521297913)

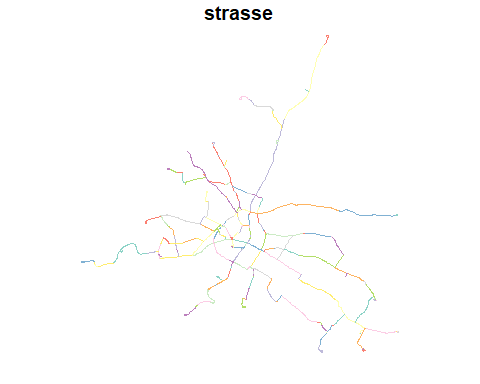
## the column containing the geo-informations is called geom and is as usual the last column, now we can convert the tram\_lines from being a database to an sf object  
  
tram\_lines\_sf <- st\_as\_sf(tram\_lines, wkt = "geom")  
  
names(tram\_lines\_sf)

## [1] "id" "schluessel" "von" "bis" "von\_str"   
## [6] "bis\_str" "wdm" "wdm\_klar" "strasse" "status"   
## [11] "historie" "straba" "laenge" "aend" "geom"

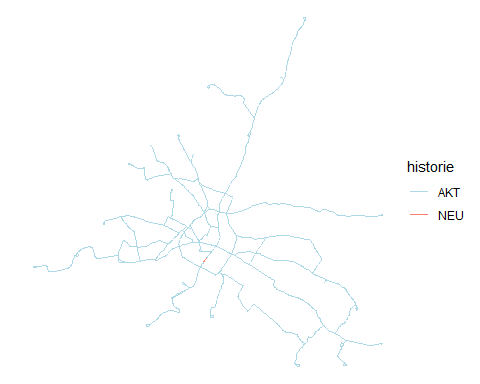
## check the crs   
  
st\_crs(tram\_lines\_sf)

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

## To quickly look at the data and see if they make sense   
  
plot(tram\_lines\_sf["strasse"])



## to get a more nice vizualization we can use the geom\_sf option in ggplot   
  
ggplot() +  
 geom\_sf(data = tram\_lines\_sf, aes(fill = historie, color = historie)) +  
 scale\_color\_manual(values = c(AKT = "lightblue", NEU = "salmon")) +  
 theme\_void()



## Now which type of geometry do you think we are visualizing here? check it out with the structure function   
  
str(tram\_lines\_sf)

## Classes 'sf' and 'data.frame': 881 obs. of 15 variables:  
## $ id : int 14 2 3 4 7 10 1 17 30 21 ...  
## $ schluessel: int 1080 1020 1030 1040 1050 1070 1010 2010 3050 3020 ...  
## $ von : int 10010 10001 10002 10003 10004 10007 10000 10015 10025 10016 ...  
## $ bis : int 10014 10002 10003 10004 10007 10010 10001 10014 10028 10019 ...  
## $ von\_str : chr "Leipziger Str." "Turnerweg" "Erna-Berger-Str." "Schlesischer Platz" ...  
## $ bis\_str : chr "Marienbrücke" "Erna-Berger-Str." "Schlesischer Platz" "Hansastraße/Hainstr." ...  
## $ wdm : chr "B" "B" "B" "B" ...  
## $ wdm\_klar : chr "Bundesstraße" "Bundesstraße" "Bundesstraße" "Bundesstraße" ...  
## $ strasse : chr "Antonstraße" "Antonstraße" "Antonstraße" "Antonstraße" ...  
## $ status : int 20 0 0 20 20 20 0 0 20 20 ...  
## $ historie : chr "AKT" "AKT" "AKT" "AKT" ...  
## $ straba : chr "G" "S" "S" "G" ...  
## $ laenge : num 251.3 88.1 94.2 125.9 170.2 ...  
## $ aend : chr "2024-02-06" "2024-02-06" "2024-02-06" "2024-02-06" ...  
## $ geom :sfc\_LINESTRING of length 881; first list element: 'XY' num [1:6, 1:2] 13.7 13.7 13.7 13.7 13.7 ...  
## - attr(\*, "sf\_column")= chr "geom"  
## - attr(\*, "agr")= Factor w/ 3 levels "constant","aggregate",..: NA NA NA NA NA NA NA NA NA NA ...  
## ..- attr(\*, "names")= chr [1:14] "id" "schluessel" "von" "bis" ...

## It is a single feature collection LINESTRING

Now let´s give a look at how the open markets are spatially distributed in Dresden

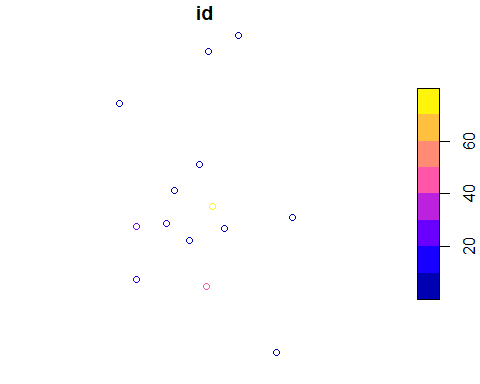
market <- read.csv(here("data", "raw", "markets.csv"),sep=';')  
head(market)

## id einrichtung zeitraum oeffnung einrichtungsart  
## 1 3 Wochenmarkt ganzjährig NA Markt  
## 2 8 Wochenmarkt Hellerau ganzjährig NA Markt  
## 3 10 Wochenmarkt ganzjährig NA Markt  
## 4 7 Wochenmarkt ganzjährig NA Markt  
## 5 5 Dresdner Bauernmarkt ganzjährig NA Markt  
## 6 13 Wochenmarkt ganzjährig NA Markt  
## strasse hnr hnrz plz fon fax  
## 1 Reißigerstraße NA NA NA 0351-47000220 0351-47000230  
## 2 Markt NA NA NA 0351-47000220 0351-47000230  
## 3 Jacob-Winter-Platz NA NA NA 0351-47000220 0351-47000230  
## 4 Stralsunder Straße NA NA NA 0351-47000220 0351-47000230  
## 5 Königstraße NA NA NA 0351-47000220 0351-47000230  
## 6 Münchener Platz NA NA NA 0351-47000220 0351-47000230  
## url  
## 1 www.dresden.de/de/leben/sport-und-freizeit/maerkte/reissigerstrasse.php  
## 2 www.dresden.de/de/leben/sport-und-freizeit/maerkte/hellerau.php  
## 3 www.dresden.de/de/leben/sport-und-freizeit/maerkte/jacob-winter-platz.php  
## 4 www.dresden.de/de/leben/sport-und-freizeit/maerkte/stralsunder-strasse.php  
## 5 www.dresden.de/de/leben/sport-und-freizeit/maerkte/koenigstrasse.php  
## 6 www.dresden.de/de/leben/sport-und-freizeit/maerkte/muenchner-platz.php  
## e\_mail adr\_nr stadtteil  
## 1 wirtschaftsfoerderung@dresden.de -1 7  
## 2 wirtschaftsfoerderung@dresden.de -1 32  
## 3 wirtschaftsfoerderung@dresden.de -1 71  
## 4 wirtschaftsfoerderung@dresden.de -1 31  
## 5 wirtschaftsfoerderung@dresden.de -1 13  
## 6 wirtschaftsfoerderung@dresden.de -1 81  
## geom  
## 1 SRID=4326;POINT(13.76983475267094192418 51.04804406503568969811)  
## 2 SRID=4326;POINT(13.76128183905927926389 51.10939474715688390916)  
## 3 SRID=4326;POINT(13.79843627215047696666 51.00523809125896690375)  
## 4 SRID=4326;POINT(13.77787262909161292157 51.11489969328145832606)  
## 5 SRID=4326;POINT(13.74262573632443285021 51.0611844312687708225)  
## 6 SRID=4326;POINT(13.72144530438137266515 51.03049029225620358829)  
## traeger ds\_modified  
## 1 2024-02-05 11:27:21  
## 2 2024-02-05 11:27:21  
## 3 2024-02-05 11:27:21  
## 4 2024-02-05 11:27:21  
## 5 Deutsche Marktgilde 2024-02-05 11:27:21  
## 6 2024-02-05 11:27:21

market\_sf <- st\_as\_sf(market, wkt = "geom")  
names(market\_sf)

## [1] "id" "einrichtung" "zeitraum" "oeffnung"   
## [5] "einrichtungsart" "strasse" "hnr" "hnrz"   
## [9] "plz" "fon" "fax" "url"   
## [13] "e\_mail" "adr\_nr" "stadtteil" "traeger"   
## [17] "ds\_modified" "geom"

plot(market\_sf["id"])



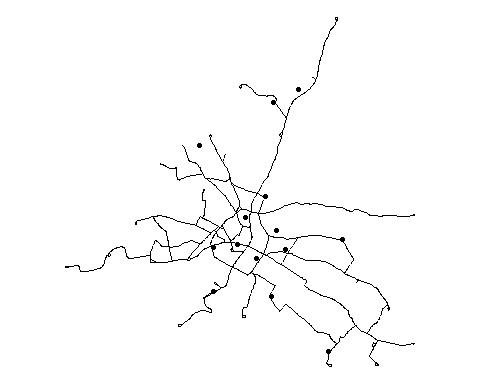
st\_crs(market\_sf)

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

Now we see only points to give it a sense with respect to the city we can plot it together with the tram lines data

## Plot the two dataset together

ggplot() +  
 geom\_sf(data = tram\_lines\_sf) +  
 geom\_sf(data= market\_sf) +  
 theme\_void()



Now you can see then city tram lines and the markets locations all in the same map , and notice that this is happening without that the two information are in the same dataset.

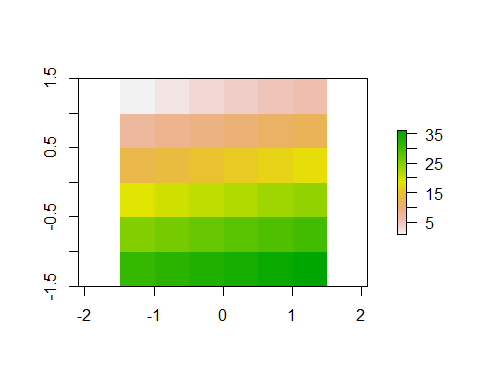
# Rasters

Now let´s step into the second main type of geo-data that we will be using in this course: raster data. To deal with this type of data we are using a package called raster and for today we are going to work with the most basic type of raster: a Raster Layer.

This data type usually has a raster header and then a matrix with rows and columns that represent equally spaced cells that we usually call pixels or grid –> in short raster data are images ! One cell can contain only one value and it can be either numeric or categorical. Rasters are usually employed for continuous measures and are generated thanks to satellite images.

Each raster has values and a resolution representing how big is each cell into the grid.

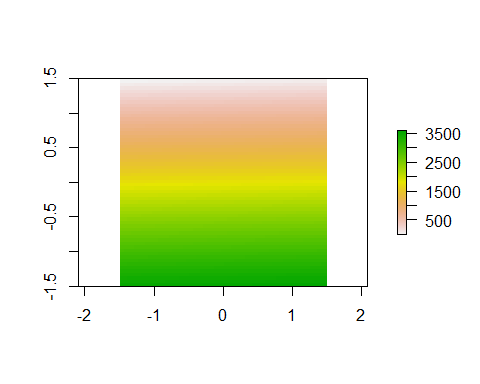
## Let´s produce a small raster of our own to look into its structure  
  
## first we define   
raster\_extent <- extent(-1.5, 1.5, -1.5, 1.5)  
raster\_resolution <- 0.5  
raster\_ncols <- 6  
raster\_nrows <- 6  
values <- 1:36  
  
our\_raster <- raster(nrows = raster\_nrows, ncols = raster\_ncols,   
 res = raster\_resolution, ext = raster\_extent, vals = values)  
  
plot(our\_raster)



Ok now what happens if we change the resolution of the raster ?

Let´s make it smaller !

## Let´s produce a small raster of our own to look into its structure  
  
# using the same extent as before  
raster\_resolution <- 0.05 # smaller resolution  
raster\_ncols <- 60 # 6 / 0.05  
raster\_nrows <- 60 # 6 / 0.05  
  
# Create the smaller raster  
smaller\_raster <- raster(nrows = raster\_nrows, ncols = raster\_ncols,   
 res = raster\_resolution, ext = raster\_extent)  
  
values(smaller\_raster) <- 1:ncell(smaller\_raster)  
  
plot(smaller\_raster)



The resolutions is a synonim for the granularity of the image and usually when looking for data online the data sources provide resolution in terms of degrees so our 0.5 will be a 0.5 x 0.5 dg. raster that usually corresponds to a satellite image where each pixel contains roughly 55 km x 55km.

### Projections in Raster data

With raster data the st\_crs() function equivalent is called projection(). To look at it we can use the volcano built-in data from the raster package and read it as a raster .

The projection function requires you to specify the full proj4 definition (while for sf objects st\_set\_crs and st\_transform are doing it automatically). The main projection used in economics is “+proj=longlat +datum=WGS84 +ellps=WGS84 +towgs84=0,0,0”

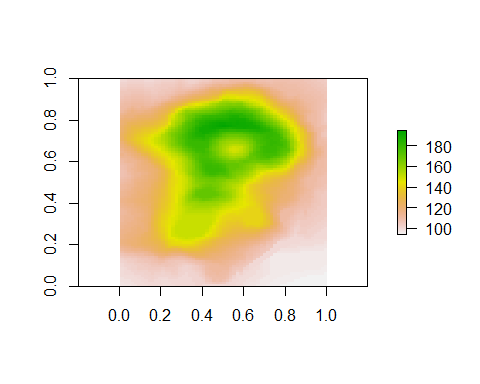
data(volcano)  
  
# Convert the volcano matrix to a raster object  
volcano\_raster <- raster(volcano)  
  
## now let´s check the projection   
  
projection(volcano\_raster)

## [1] NA

##NA so let´s set the projection using the proj4 definition   
  
proj4\_string <- "+proj=longlat +datum=WGS84 +ellps=WGS84 +towgs84=0,0,0"  
  
projection(volcano\_raster) <- proj4\_string  
  
projection(volcano\_raster)

## [1] "+proj=longlat +datum=WGS84 +no\_defs"

## Now to plot the raster data we can use the usual plot but calling the raster package before as below  
  
raster::plot(volcano\_raster)

 But if we want a higher quality plot we can also use geom\_raster from ggplot that although requires us to convert the raster back to a dataframe using as.data.frame and specifying the option xy = TRUE.

volcano\_df <- as.data.frame(volcano\_raster, xy = TRUE)  
  
# Create the plot  
ggplot() +  
 geom\_raster(data = volcano\_df, aes(x = x, y = y, fill = layer)) +  
 scale\_fill\_gradientn(colors = terrain.colors(10)) +   
 labs(x = "Longitude", y = "Latitude", title = "Volcano Raster Plot") +   
 theme\_minimal() +   
 theme(plot.title = element\_text(size = 16, face = "bold"),   
 axis.text = element\_text(size = 12),   
 axis.title = element\_text(size = 14, face = "bold"))

