

# Impresión 3D por medio de R

Presentación de LatinR 2018

*Lucas Pogorelsky*

## Objetivo

El objetivo de este trabajo es representar, por medio de la impresión 3D, la cantidad de personas que utilizan el Subte de Buenos Aires.

Para ello se usarán datos provenientes del portal de datos abiertos Buenos Aires Data referentes a la red de Subterráneos de Buenos Aires y las librerías `sf` para explotar los datos geográficos, `tidyverse` para la manipulación y `r2stl` para la generación del archivo `.stl` necesario para la impresión en 3D.

La impresión en 3D constará de un mapa de la Ciudad de Buenos Aires con la delimitación de los barrios, las líneas de subte y, representado por barras verticales, la cantidad de personas que se subieron en cada estación de Subte.

## Archivos stl y paquete r2stl

Los archivos `stl` describen una superficie utilizando triángulos y es un standard en la impresión 3D.

La función `r2stl` del paquete homónimo toma como argumento coordenadas X, Y y Z y genera el archivo `stl`.

En nuestro caso, las coordenadas X e Y, las vamos a extraer de los datos geográficos, mientras que el valor de Z se desprenderá de la cantidad de gente que use los molinetes de cada estación.

## Datos geográficos y Simple Features

La librería `sf` utiliza Simple Features que es un standard que describe cómo representar objetos del mundo real con la computadora.

En nuestro caso, se usarán los tipos geométricos:

- `POINT`: Un punto.
- `LINESTRING`: Una secuencia de puntos conectados por líneas rectas que no se intersectan.
- `POLYGON`: Una secuencia depuntos conectados por líneas que forman un anillo cerrado.

## Sistema

### Librerías

Comenzamos con la importación de las librerías a utilizar.

```
suppressMessages(library(tidyverse))
#Para manipular data GEO
library(sf)

## Linking to GEOS 3.6.2, GDAL 2.2.3, proj.4 4.9.3
```

```

#Para imprimir 3D
library(r2stl)
#Gráficos
library(plotly)

##
## Attaching package: 'plotly'
## The following object is masked from 'package:ggplot2':
##
##     last_plot
## The following object is masked from 'package:stats':
##
##     filter
## The following object is masked from 'package:graphics':
##
##     layout
#Necesario para poder observar gráficos en 3D
library(rgl)

```

## Especificaciones Técnicas

```

sessionInfo()

## R version 3.5.1 (2018-07-02)
## Platform: x86_64-pc-linux-gnu (64-bit)
## Running under: Linux Mint 19
##
## Matrix products: default
## BLAS: /usr/lib/x86_64-linux-gnu/openblas/libblas.so.3
## LAPACK: /usr/lib/x86_64-linux-gnu/libopenblas-r0.2.20.so
##
## locale:
## [1] LC_CTYPE=en_US.UTF-8          LC_NUMERIC=C
## [3] LC_TIME=en_US.UTF-8          LC_COLLATE=en_US.UTF-8
## [5] LC_MONETARY=es_AR.UTF-8      LC_MESSAGES=en_US.UTF-8
## [7] LC_PAPER=es_AR.UTF-8         LC_NAME=C
## [9] LC_ADDRESS=C                 LC_TELEPHONE=C
## [11] LC_MEASUREMENT=es_AR.UTF-8   LC_IDENTIFICATION=C
##
## attached base packages:
## [1] stats      graphics    grDevices  utils      datasets   methods    base
##
## other attached packages:
## [1] rgl_0.99.16    plotly_4.8.0    r2stl_1.0.0    sf_0.6-3
## [5]forcats_0.3.0  stringr_1.3.1   dplyr_0.7.6    purrr_0.2.5
## [9]readr_1.1.1    tidyr_0.8.1    tibble_1.4.2    ggplot2_3.0.0
## [13]tidyverse_1.2.1
##
## loaded via a namespace (and not attached):
## [1] Rcpp_0.12.18       lubridate_1.7.4
## [3] lattice_0.20-35    class_7.3-14

```

```

## [5] assertthat_0.2.0          rprojroot_1.3-2
## [7] digest_0.6.15            mime_0.5
## [9] R6_2.2.2                 cellranger_1.1.0
## [11] plyr_1.8.4               backports_1.1.2
## [13] evaluate_0.11            e1071_1.7-0
## [15] httr_1.3.1               pillar_1.3.0
## [17] rlang_0.2.2              lazyeval_0.2.1
## [19] readxl_1.1.0             miniUI_0.1.1.1
## [21] rstudioapi_0.7            data.table_1.11.4
## [23] rmarkdown_1.10            webshot_0.5.0
## [25] htmlwidgets_1.2           munsell_0.5.0
## [27] shiny_1.1.0               broom_0.5.0
## [29] compiler_3.5.1            httpuv_1.4.5
## [31] modelr_0.1.2              pkgconfig_2.0.2
## [33] htmltools_0.3.6            tidyselect_0.2.4
## [35] viridisLite_0.3.0          crayon_1.3.4
## [37] withr_2.1.2               later_0.7.3
## [39] grid_3.5.1                nlme_3.1-137
## [41] spData_0.2.9.3            jsonlite_1.5
## [43] xtable_1.8-2              gtable_0.2.0
## [45] DBI_1.0.0                 magrittr_1.5
## [47] units_0.6-0               scales_1.0.0
## [49] cli_1.0.0                 stringi_1.2.4
## [51] promises_1.0.1             bindrcpp_0.2.2
## [53] xml2_1.2.0                tools_3.5.1
## [55] manipulateWidget_0.10.0     glue_1.3.0
## [57] hms_0.4.2                 crosstalk_1.0.0
## [59] yaml_2.2.0                 colorspace_1.3-2
## [61] classInt_0.2-3             rvest_0.3.2
## [63] knitr_1.20                 bindr_0.1.1
## [65] haven_1.1.2

```

- Procesador: Intel® Core™ i7-8550U
- Memoria Ram: 16GB

## Procesamiento

### Descarga de Archivos

Generamos la estructura de archivos, una carpeta para descarga y una para los datos descomprimidos.

```

dir.create('downloads', showWarnings=FALSE)
dir.create('data', showWarnings=FALSE)

```

Creamos la función que descarga y descomprime los archivos.

```

descargaDatos <- function(archivoOrigen, archivoDestino, descargar) {
  if (descargar){
    #Parámetros de descarga y descompresión
    urlArchivo <- paste0('https://data.buenosaires.gob.ar/api/files/', archivoOrigen, '/download')
    pathDescarga <- paste0('downloads/', archivoDestino)
    pathDatos <- paste0('data/', strsplit(archivoDestino, '\\.')[[1]][1])

    #Descarga del archivo
  }
}

```

```

download.file(urlArchivo,pathDescarga)

if(substr(archivoDestino,nchar(archivoDestino)-3,nchar(archivoDestino))=='.zip'){
  #Descompresión de archivos .zip
  unzip(pathDescarga,
        exdir = pathDatos)
} else {
  #Descompresión de archivos .rar
  dir.create(pathDatos,showWarnings = F)
  #Esta línea depende del sistema operativo y del programa para descomprimir rar instalado
  cmd = paste0('unrar x -y "', getwd(), '/', pathDescarga , '" "' , getwd(), '/', pathDatos , '"')
  system(cmd)
}
}
}
}

```

Descargamos y extraemos los archivos referentes a:

- Cantidad de usuarios de molinetes en 2018 <https://data.buenosaires.gob.ar/dataset/subte-viajes-molinetes>
- Contorno de los barrios de Buenos Aires <https://data.buenosaires.gob.ar/dataset/barrios>
- Referencia geográfica de la líneas y estaciones de Subte <https://data.buenosaires.gob.ar/dataset/subte-estaciones>

```

#Cambiar a TRUE en caso de querer descargar los datos
descargar <- FALSE
descargaDatos('molinetes-2018.zip','molinetes.zip',descargar)
descargaDatos('barrios-rar.rar','barrios.rar',descargar)
descargaDatos('estaciones-de-subte-rar.rar','estaciones.rar',descargar)
descargaDatos('lineas-de-subte-rar.rar','lineas.rar',descargar)

```

## Molinetes

El dataset con el uso de molinetes contiene “Cantidad de pasajeros por molinete en cada estación en rangos de a 15 minutos y discriminando según tipo de pasaje correspondiente al año 2018.”

Comencemos importando el archivo csv.

```

molinetes <- read_csv(paste('data/molinetes',dir('data/molinetes')[1],sep='/'),progress = F)

## Parsed with column specification:
## cols(
##   PERIODO = col_integer(),
##   FECHA = col_character(),
##   DESDE = col_time(format = ""),
##   HASTA = col_time(format = ""),
##   LINEA = col_character(),
##   MOLINETE = col_character(),
##   ESTACION = col_character(),
##   PAX_PAGOS = col_integer(),
##   PAX_PASES_PAGOS = col_integer(),
##   PAX_FRANQ = col_integer(),
##   TOTAL = col_integer()
## )

```

```

head(molinetes)

## # A tibble: 6 x 11
##   PERIODO FECHA DESDE HASTA
##   <int> <chr> <tim> <chr> <chr> <int>
## 1 201801 01/0~ 08:00 08:15 LINE~ LINEA_A~ CASTRO ~     1
## 2 201801 01/0~ 08:00 08:15 LINE~ LINEA_A~ LIMA        4
## 3 201801 01/0~ 08:00 08:15 LINE~ LINEA_A~ PASCO       1
## 4 201801 01/0~ 08:00 08:15 LINE~ LINEA_A~ PERU        4
## 5 201801 01/0~ 08:00 08:15 LINE~ LINEA_A~ PRIMERA~    2
## 6 201801 01/0~ 08:00 08:15 LINE~ LINEA_A~ SAN PED~     1
## # ... with 3 more variables: PAX_PASES_PAGOS <int>, PAX_FRANQ <int>,
## #   TOTAL <int>

summary(molinetes)

##      PERIODO          FECHA          DESDE          HASTA
## Min.   :201801  Length:4953998  Length:4953998  Length:4953998
## 1st Qu.:201802  Class :character  Class1:hms    Class1:hms
## Median :201803  Mode   :character  Class2:difftime Class2:difftime
## Mean   :201803                    Mode   :numeric   Mode   :numeric
## 3rd Qu.:201805
## Max.  :201806

##      LINEA          MOLINETE          ESTACION          PAX_PAGOS
## Length:4953998  Length:4953998  Length:4953998  Min.   :  0.00
## Class :character Class :character  Class :character  1st Qu.:  6.00
## Mode  :character Mode  :character  Mode  :character  Median : 17.00
##                               Mean   : 26.62
##                               3rd Qu.: 37.00
##                               Max.  :462.00

##      PAX_PASES_PAGOS      PAX_FRANQ          TOTAL
## Min.   : 0.00000  Min.   : 0.0000  Min.   :  0.00
## 1st Qu.: 0.00000  1st Qu.: 0.0000  1st Qu.:  6.00
## Median : 0.00000  Median : 0.0000  Median : 18.00
## Mean   : 0.07166  Mean   : 0.8933  Mean   : 27.58
## 3rd Qu.: 0.00000  3rd Qu.: 1.0000  3rd Qu.: 39.00
## Max.   :91.00000  Max.   :89.0000  Max.   :464.00

```

## Fechas

Como podemos observar, la variable FECHA fue importada como texto. Veamos algunos casos para ver el formato de fecha y convertirlo adecuadamente.

```

set.seed(1234)
sample(unique(molinetes$FECHA), 15)

## [1] "18/01/2018" "04/05/2018" "01/05/2018" "03/05/2018" "06/06/2018"
## [6] "29/06/2018" "02/01/2018" "03/02/2018" "06/05/2018" "15/03/2018"
## [11] "08/05/2018" "18/03/2018" "09/02/2018" "07/06/2018" "10/02/2018"

```

Vemos que el formato es '%d/%m/%Y'.

```

molinetes$FECHA <- as.Date(molinetes$FECHA, '%d/%m/%Y')
summary(molinetes)

```

	PERIODO	FECHA	DESDE	HASTA
--	---------	-------	-------	-------

```

## Min.    :201801   Min.    :2018-01-01   Length:4953998   Length:4953998
## 1st Qu.:201802   1st Qu.:2018-02-07   Class1:hms      Class1:hms
## Median :201803   Median :2018-03-16   Class2:difftime  Class2:difftime
## Mean   :201803   Mean   :2018-03-28   Mode  :numeric   Mode  :numeric
## 3rd Qu.:201805   3rd Qu.:2018-05-24
## Max.    :201806   Max.    :2018-06-30

##          LINEA           MOLINETE          ESTACION          PAX_PAGOS
## Length:4953998   Length:4953998   Length:4953998   Min.    : 0.00
## Class :character  Class :character  Class :character  1st Qu.: 6.00
## Mode  :character  Mode  :character  Mode  :character  Median  :17.00
##                                     Mean   :26.62
##                                     3rd Qu.:37.00
##                                     Max.   :462.00

##          PAX_PASES_PAGOS      PAX_FRANQ        TOTAL
## Min.    : 0.00000   Min.    : 0.0000   Min.    : 0.00
## 1st Qu.: 0.00000   1st Qu.: 0.0000   1st Qu.: 6.00
## Median : 0.00000   Median : 0.0000   Median :18.00
## Mean   : 0.07166   Mean   : 0.8933   Mean   :27.58
## 3rd Qu.: 0.00000   3rd Qu.: 1.0000   3rd Qu.:39.00
## Max.   :91.00000   Max.   :89.0000   Max.   :464.00

```

Comprobemos si la variable PERIODO es correcta.

```

molinetes %>%
  group_by(PERIODO) %>%
  summarise(minFecha = min(FECHA),
            maxFecha = max(FECHA))

## # A tibble: 5 x 3
##   PERIODO minFecha  maxFecha
##   <int>     <date>    <date>
## 1 201801 2018-01-01 2018-01-31
## 2 201802 2018-02-01 2018-02-28
## 3 201803 2018-03-01 2018-03-31
## 4 201805 2018-05-01 2018-05-31
## 5 201806 2018-06-01 2018-06-30

```

Como vemos, la variable es correcta, pero no tenemos los datos correspondientes a abril de 2018.

## Limpieza de nombres

Comprobemos si están todas las líneas presentes.

```
length(unique(molinetes$LINEA))
```

```
## [1] 12
```

Deberíamos tener 6 valores únicos, veamos qué sucede.

```
unique(molinetes$LINEA)
```

```
## [1] "LINEA_A" "LINEA_B" "LINEA_C" "LINEA_D" "LINEA_E" "LINEA_H" "LineaA"
## [8] "LineaB"  "LineaC"  "LineaD"  "LineaE"  "LineaH"
```

Nos quedamos únicamente con la letra de la línea.

```
molinetes$LINEA <- substr(molinetes$LINEA,nchar(molinetes$LINEA),nchar(molinetes$LINEA))
```

Veamos si las estaciones están bien. En teoría, deberíamos tener:

LINEA	CANTIDAD DE ESTACIONES
A	18
B	17
C	9
D	16
E	15
H	12

```
molinetes %>%
  group_by(LINEA) %>%
  summarise(~CANTIDAD DE ESTACIONES` = n_distinct(ESTACION))
```

```
## # A tibble: 6 x 2
##   LINEA ~CANTIDAD DE ESTACIONES~
##   <chr>          <int>
## 1 A                  37
## 2 B                  34
## 3 C                  18
## 4 D                  33
## 5 E                  30
## 6 H                  23
```

Vemos que tenemos al menos el doble de estaciones por línea, analicemos qué puede estar sucediendo.

```
molinetes %>%
  select(LINEA,ESTACION) %>%
  distinct() %>%
  arrange(LINEA,ESTACION) %>%
  head(10)
```

```
## # A tibble: 10 x 2
##   LINEA ESTACION
##   <chr> <chr>
## 1 A      Acoyte
## 2 A      ACOYTE
## 3 A      Alberti
## 4 A      ALBERTI
## 5 A      Carabobo
## 6 A      CARABOBO
## 7 A      Castro Barros
## 8 A      CASTRO BARROS
## 9 A      Congreso
## 10 A     CONGRESO
```

Pasamos a mayúscula el nombre de todas las estaciones y comprobamos si se soluciona.

```
molinetes$ESTACION <- toupper(molinetes$ESTACION)
molinetes %>%
  group_by(LINEA) %>%
  summarise(ESTACIONES=n_distinct(ESTACION))
```

```
## # A tibble: 6 x 2
##   LINEA ESTACIONES
```

```

## <chr>     <int>
## 1 A          20
## 2 B          17
## 3 C           9
## 4 D          18
## 5 E          15
## 6 H          12

```

Las líneas A y D siguen teniendo más estaciones que las que deberíamos, veamos lo que sucede.

```

molinetes %>%
  filter(LINEA %in% c('A', 'D')) %>%
  select(LINEA, ESTACION) %>%
  distinct() %>%
  arrange(LINEA, ESTACION)

```

```

## # A tibble: 38 x 2
##   LINEA ESTACION
##   <chr> <chr>
## 1 A      ACOYTE
## 2 A      ALBERTI
## 3 A      CARABOBO
## 4 A      CASTRO BARROS
## 5 A      CONGRESO
## 6 A      FLORES
## 7 A      LIMA
## 8 A      LORIA
## 9 A      PASCO
## 10 A     PERU
## # ... with 28 more rows

```

El problema está en las estaciones SAENZ PEÑA y AGÜERO, lo arreglamos manualmente.

```

molinetes <- molinetes %>% mutate(ESTACION=case_when(LINEA=='A' & substr(ESTACION,1,5)=='SAENZ' ~ 'SAENZ'
                                                       LINEA=='D' & substr(ESTACION,1,2)=='AG' ~ 'AGÜERO'
                                                       TRUE ~ ESTACION))

molinetes %>%
  group_by(LINEA) %>%
  summarise(ESTACIONES=n_distinct(ESTACION))

```

```

## # A tibble: 6 x 2
##   LINEA ESTACIONES
##   <chr>     <int>
## 1 A          18
## 2 B          17
## 3 C           9
## 4 D          16
## 5 E          15
## 6 H          12

```

Finalmente tenemos los datos de estaciones limpios.

## Combinaciones

Debido a que los usuarios no necesariamente utilizan el molinete de la línea en la que viajan, vamos a considerar las combinaciones como una estación en su totalidad.

Generamos el DataFrame que indica qué estaciones corresponden a las combinaciones:

```
combinaciones <- c('A', 'LIMA', 'COMB A-C',
                  'C', 'AVENIDA DE MAYO', 'COMB A-C',
                  'A', 'PERU', 'COMB A-D-E',
                  'D', 'CATEDRAL', 'COMB A-D-E',
                  'E', 'BOLIVAR', 'COMB A-D-E',
                  'A', 'PLAZA MISERERE', 'COMB A-H',
                  'H', 'ONCE', 'COMB A-H',
                  'B', 'CARLOS PELLEGRINI', 'COMB B-C-D',
                  'C', 'DIAGONAL NORTE', 'COMB B-C-D',
                  'D', '9 DE JULIO', 'COMB B-C-D',
                  'B', 'PUEYRREDON', 'COMB B-H',
                  'H', 'CORRIENTES', 'COMB B-H',
                  'C', 'INDEPENDENCIA', 'COMB C-E',
                  'E', 'INDEPENDENCIA.H', 'COMB C-E',
                  'D', 'PUEYRREDON.D', 'COMB D-H',
                  'H', 'SANTA FE', 'COMB D-H',
                  'E', 'JUJUY', 'COMB E-H',
                  'H', 'HUMBERTO I', 'COMB E-H'
)

combinaciones <- data.frame(LINEA = combinaciones[seq(1,length(combinaciones),3)],
                           ESTACION = combinaciones[seq(2,length(combinaciones),3)],
                           COMBINACION = combinaciones[seq(3,length(combinaciones),3)],
                           stringsAsFactors = F)
```

## Cantidad de Usuarios por Estación

Finalmente, ya con los datos limpios y las combinaciones establecidas, procedemos a calcular la cantidad mensual promedio de pasajeros que comienzan el recorrido en cada estación.

```
molinetesCantidad <- molinetes %>%
  left_join(combinaciones, by=c('LINEA', 'ESTACION')) %>%
  mutate(COMBINACION = ifelse(is.na(COMBINACION), paste(LINEA, ESTACION, sep=' | '), COMBINACION))
  group_by(COMBINACION, PERIODO) %>%
  summarise(cantidad = sum(TOTAL)) %>%
  group_by(COMBINACION) %>%
  summarise(cantidad = mean(cantidad)) %>%
  ungroup() %>%
  left_join(combinaciones, by="COMBINACION") %>%
  mutate(LINEA = ifelse(is.na(LINEA), substr(COMBINACION, 1, 1), LINEA),
        ESTACION = ifelse(is.na(ESTACION), substr(COMBINACION, 3, nchar(COMBINACION)), ESTACION))

head(molinetesCantidad)

## # A tibble: 6 x 4
##   COMBINACION      cantidad LINEA ESTACION
##   <chr>          <dbl> <chr> <chr>
## 1 A|ACOYTE       350683  A     ACOYTE
```

```

## 2 A|ALBERTI      122973 A    ALBERTI
## 3 A|CARABOBO     286901 A    CARABOBO
## 4 A|CASTRO BARROS 234639. A   CASTRO BARROS
## 5 A|CONGRESO      333369. A   CONGRESO
## 6 A|FLORES        294259. A   FLORES

```

Veamos cómo se distribuye la cantidad:

```

molinetesCantidad %>%
  select(COMBINACION, cantidad) %>%
  distinct() %>%
  arrange(desc(cantidad)) %>%
  mutate(COMBINACION=factor(COMBINACION, levels=COMBINACION)) %>%
  plot_ly(y=~COMBINACION,
          x=-cantidad,
          type='bar') %>%
  layout(title="CANTIDAD DE PASAJEROS POR ESTACIÓN")

```

La diferencia entre la mayor barra y las menores, permite que se puedan representar todas sin necesidad de recurrir a una escala logarítmica.

## Barrios

Comencemos viendo los archivos que contenía el .rar que descargamos.

```

dir('data/barrios')

## [1] "barrios_badata.dbf"      "barrios_badata.prj"
## [3] "barrios_badata.shp"       "barrios_badata.shp.xml"
## [5] "barrios_badata.shx"

Como vemos, hay un archivo .shp con los datos que necesitamos, lo abrimos con la función st_read de la librería sf.

barrios <- st_read(paste('data/barrios', dir('data/barrios', '\\.shp$'), sep='/'))

```

## Reading layer `barrios\_badata` from data source `/home/pogo/Google Drive/EANT/LatinR/2018/data/barrios`

## Simple feature collection with 48 features and 4 fields

## geometry type: POLYGON

## dimension: XY

## bbox: xmin: 93743.42 ymin: 91566.42 xmax: 111752 ymax: 111285.1

## epsg (SRID): NA

## proj4string: +proj=tmerc +lat\_0=-34.6297166 +lon\_0=-58.4627 +k=0.9999980000000001 +x\_0=100000 +y\_0=0

Veamos la estructura de este tipo de datos.

```

str(barrios)

## Classes 'sf' and 'data.frame': 48 obs. of 5 variables:
## $ BARRIO : Factor w/ 48 levels "AGRONOMIA","ALMAGRO",...: 9 26 36 37 2 8 46 18 44 13 ...
## $ COMUNA : num 15 15 15 11 5 6 11 10 10 7 ...
## $ PERIMETRO: num 7726 7088 8133 7705 8538 ...
## $ AREA    : num 3118101 2229829 3613584 3399596 4050752 ...
## $ geometry: sfc_POLYGON of length 48; first list element: List of 1
##   ..$ : num [1:572, 1:2] 100961 100885 100874 100870 100780 ...
##   ..- attr(*, "class")= chr "XY" "POLYGON" "sfg"
##   - attr(*, "sf_column")= chr "geometry"
##   - attr(*, "agr")= Factor w/ 3 levels "constant","aggregate",...: NA NA NA NA

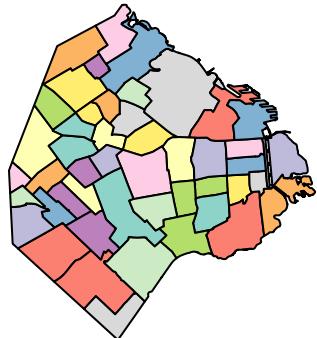
```

```
## ..- attr(*, "names")= chr "BARRIO" "COMUNA" "PERIMETRO" "AREA"
```

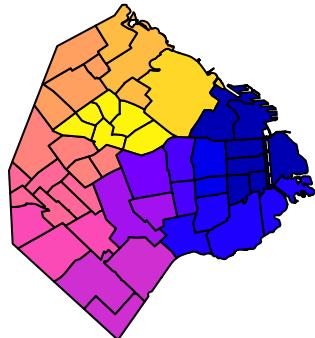
Graficamos:

```
plot(barrios)
```

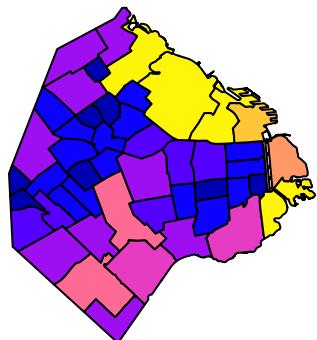
**BARRIO**



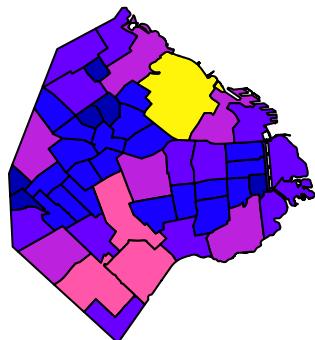
**COMUNA**



**PERIMETRO**



**AREA**



Para trabajar con los datos, vamos a extraer los puntos que forman cada polígono con la función `st_coordinates`.

```
head(st_coordinates(barrios))
```

```
##           X      Y L1 L2
## [1,] 100961.3 103692.0  1  1
## [2,] 100884.7 103629.1  1  1
## [3,] 100874.4 103620.4  1  1
## [4,] 100869.9 103616.6  1  1
## [5,] 100780.2 103540.7  1  1
## [6,] 100716.9 103487.2  1  1
```

Como podemos ver, tenemos puntos y las variables L1 y L2 que nos indican qué polígono están formando esos puntos.

Pasemos estos datos a un DataFrame redondeando los puntos ya que para la tarea propuesta, no necesitamos tanta precisión.

```
barriosDF <- data.frame(round(st_coordinates(barrios))) %>% distinct()
```

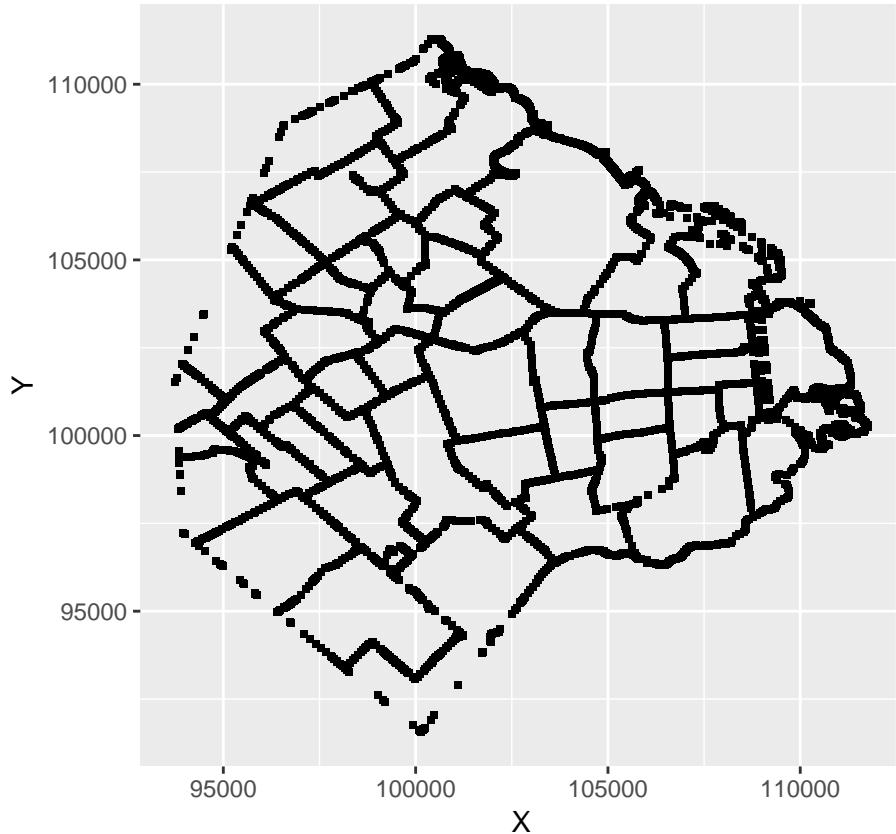
Grafiquemos los puntos:

```

graficaCoord <- function(df) {
  ggplot(df, aes(x=X,y=Y)) + geom_point(shape=15, size=1) + theme(aspect.ratio=1)
}

graficaCoord(barriosDF)

```



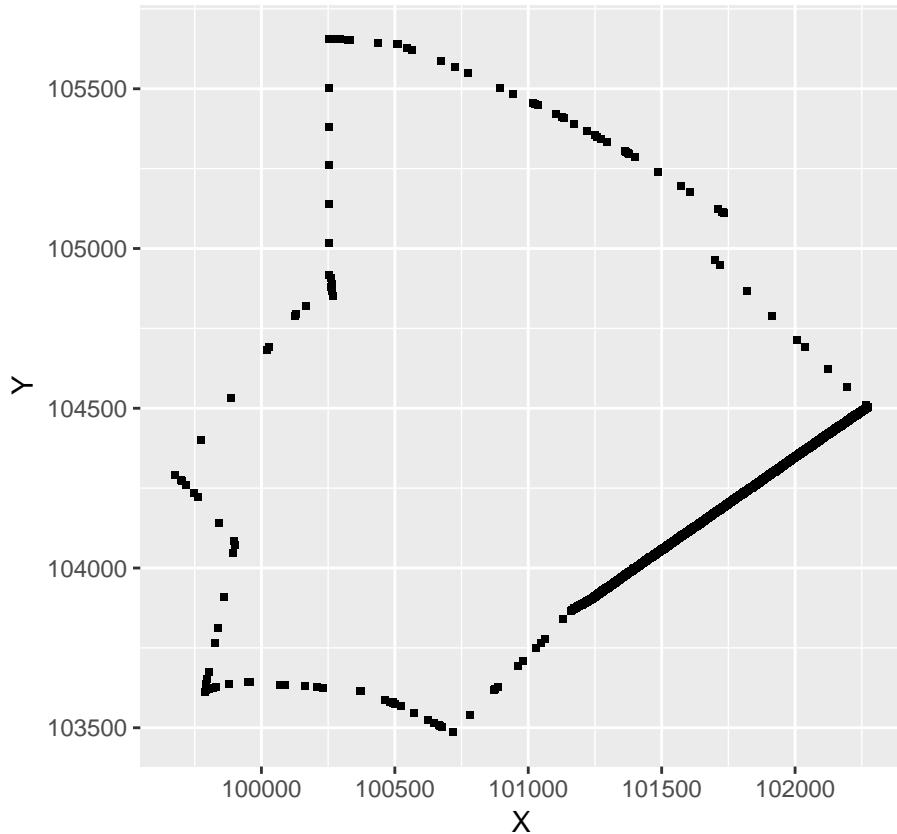
Como podemos ver las líneas de los polígonos no están completas, para ello, vamos a completar los puntos de los polígonos faltantes

Comencemos con un solo polígono:

```

muestra <- filter(barriosDF,L1==1 & L2==1) %>% select (X,Y)
graficaCoord(muestra)

```



Generamos las funciones para completar las líneas.

```
completarLinea <- function(x0,y0,x1,y1){
#Recibe 2 coordenadas y devuelve un DataFrame con todos los puntos de la linea entre estos 2 puntos
  if (x0!=x1){
    b <- ((y1-y0)/(x1-x0))
    x <- x0:x1
    y <- round(y0 + b*(x-x0))
  } else {
    y <- y0:y1
    x=rep(x0,length(y))
  }

  return(data.frame(X=x,Y=y))
}

dataOrigenDestino <- function(df,polygono=TRUE){
#Recibe un DataFrame con coordenadas y devuelve un DataFrame con las coordenadas previas en la linea
  if(polygono){
    #Si es un polígono, unir el último punto con el primero
    df$X1 <- c(df$X[nrow(df)],df$X[-nrow(df)])
    df$Y1 <- c(df$Y[nrow(df)],df$Y[-nrow(df)])
  } else {
    df$X1<-c(df$X[-1],NA)
    df$Y1<-c(df$Y[-1],NA)
    df <- df[-nrow(df),]
```

```

    }
    return(df)
}

dataCompleta <- function(df,polygono=TRUE){
#Recibe un DataFrame con varias líneas y polígonos y los completa
  apply(dataOrigenDestino(df,polygono),
  1,
  function(linea) completarLinea(linea[1],linea[2],linea[3],linea[4])) %>%
  bind_rows() %>%
  distinct()
}

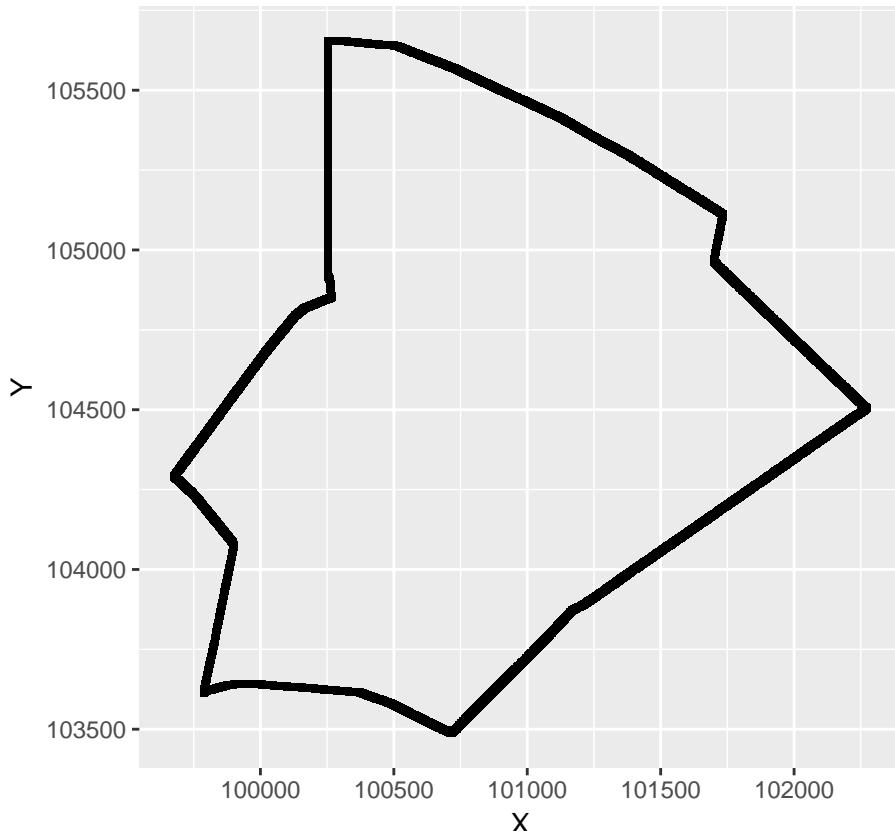
```

Probemos cómo resulta con un solo polígono.

```

muestraCompleta <- dataCompleta(muestra)
graficaCoord(muestraCompleta)

```



Aplicaremos a todos los polígonos de Barrio.

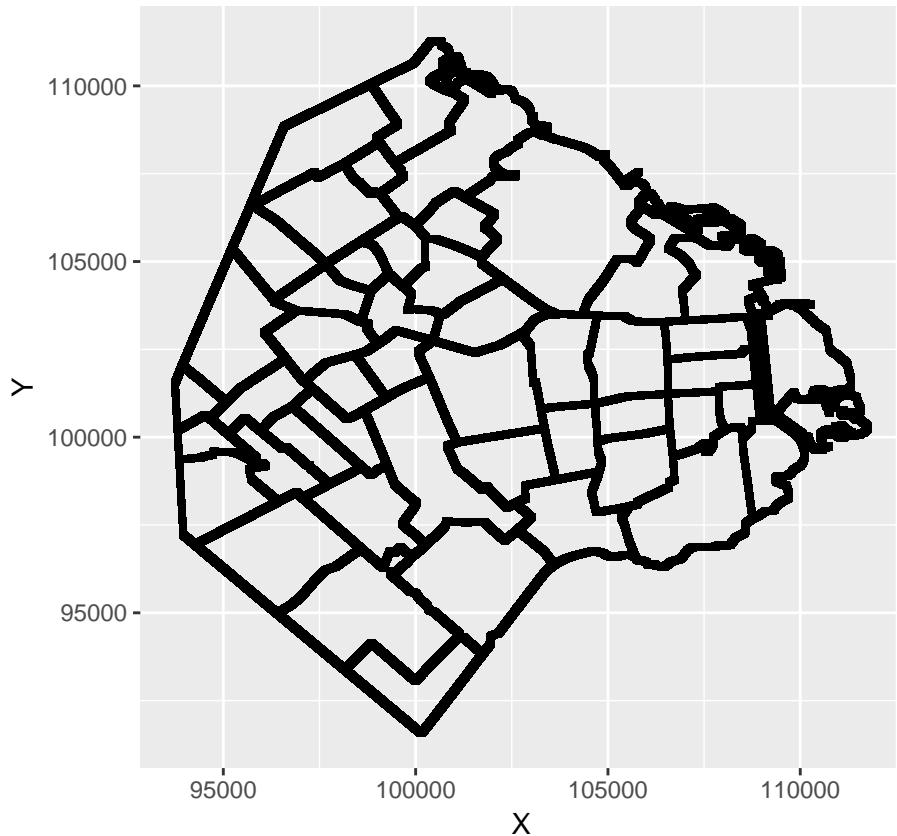
```

barriosCompleta <- lapply((barriosDF %>% group_by(L1,L2) %>% nest())$data,
                           dataCompleta) %>%
  bind_rows() %>%
  distinct()

```

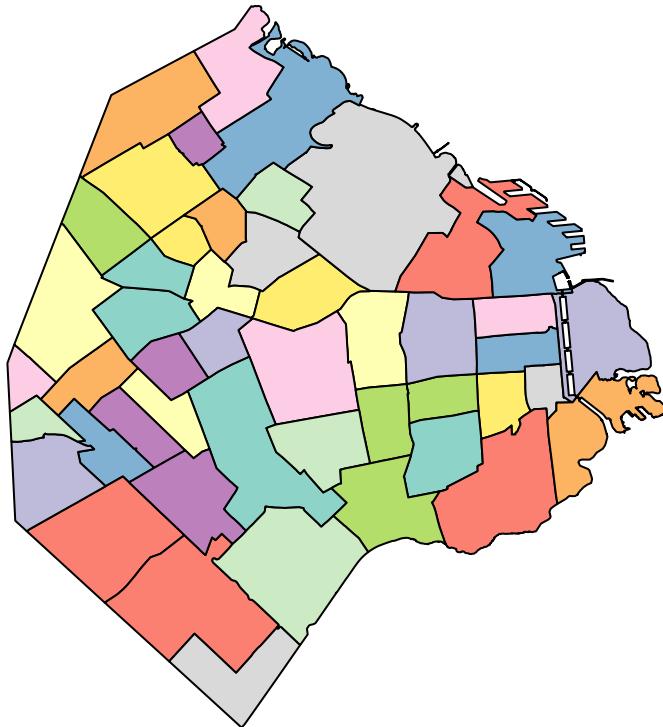
Grafiquemos el resultado y comparemos

```
graficaCoord(barriosCompleta)
```



```
plot(barrios[1])
```

## BARRIO



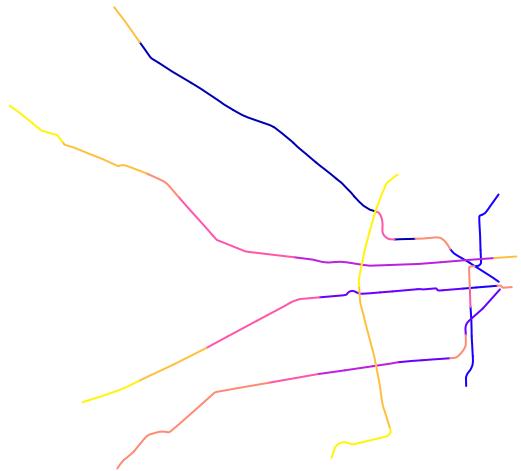
Como podemos ver, se respeta la forma del original

### Lineas

Importemos y procesemos las líneas de Subte:

```
lineas <- st_read(paste('data/lineas', dir('data/lineas', '\\.shp$'), sep='/'))  
  
## Reading layer `red_de_subte` from data source `/home/pogo/Google Drive/EANT/LatinR/2018/data/lineas/  
## Simple feature collection with 80 features and 2 fields  
## geometry type:  LINESTRING  
## dimension:      XY  
## bbox:            xmin: 97881.68 ymin: 98442.25 xmax: 108564.5 ymax: 108167.8  
## epsg (SRID):    NA  
## proj4string:    +proj=tmerc +lat_0=-34.6297166 +lon_0=-58.4627 +k=0.9999980000000001 +x_0=100000 +y_0=0  
plot(lineas)
```

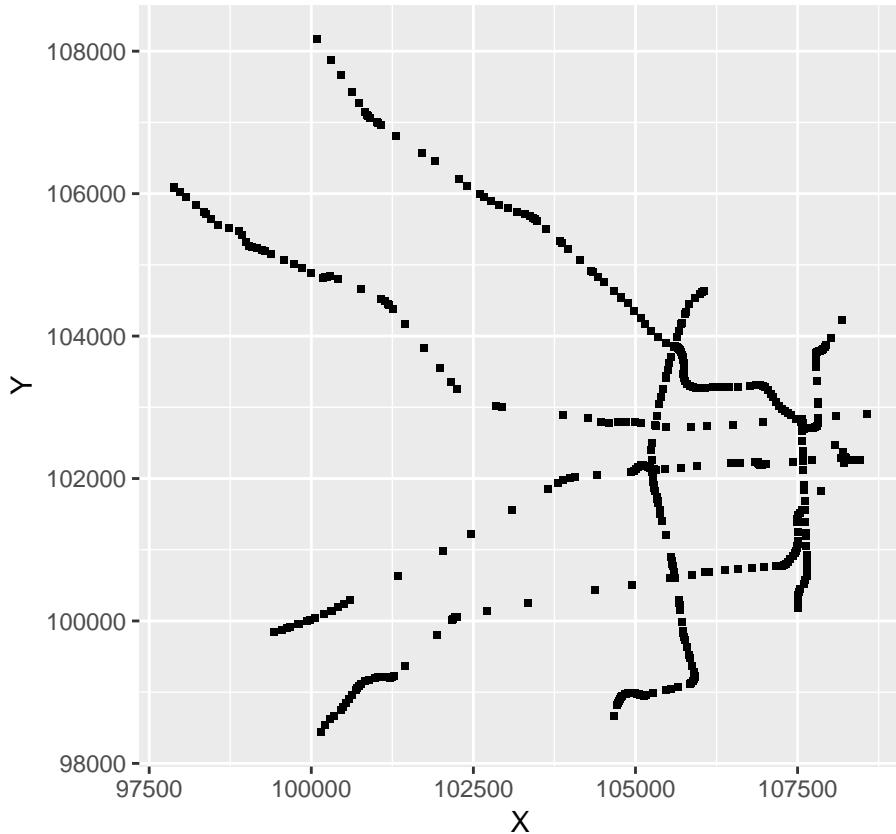
**ID**



**LINEASUB**



```
lineasDF <- data.frame(round(st_coordinates(lineas))) %>% distinct()
graficaCoord(lineasDF)
```

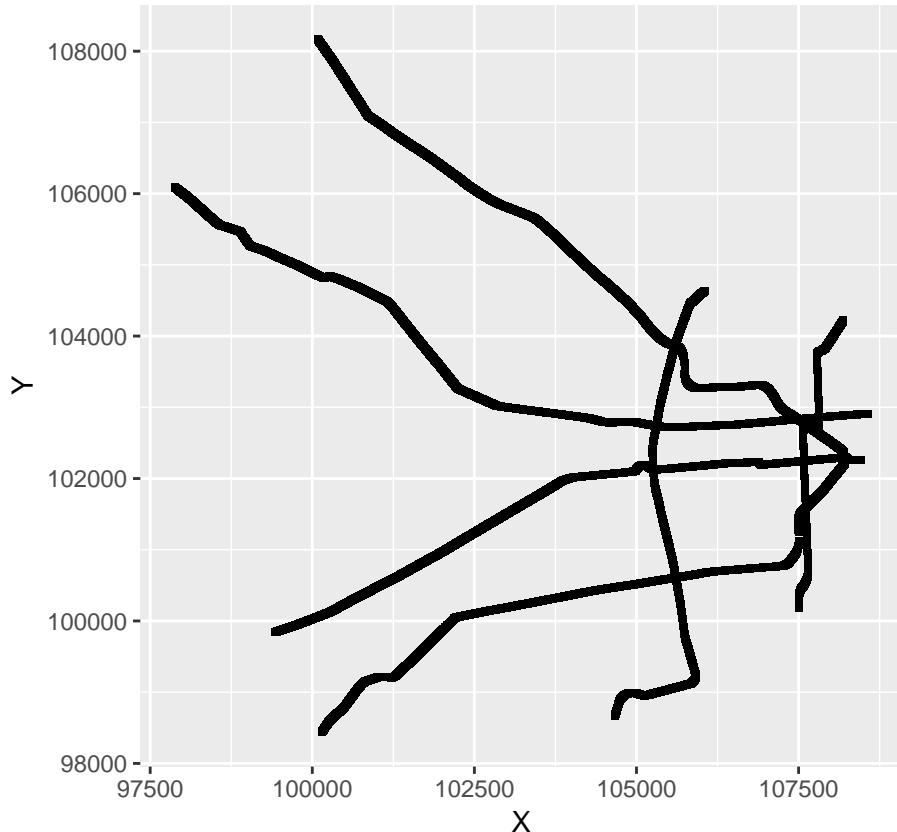


Procesamos todas las líneas. En este caso la variable que diferencia cada línea es L1.

```
lineasCompleta <- lapply((group_by(lineasDF,L1) %>% nest())$data,
                           function(df){df <- dataCompleta(df,polygono = F)}) %>%
                           bind_rows() %>%
                           distinct()
```

Comparemos con el original:

```
graficaCoord(lineasCompleta)
```



```
plot(lineas[2])
```

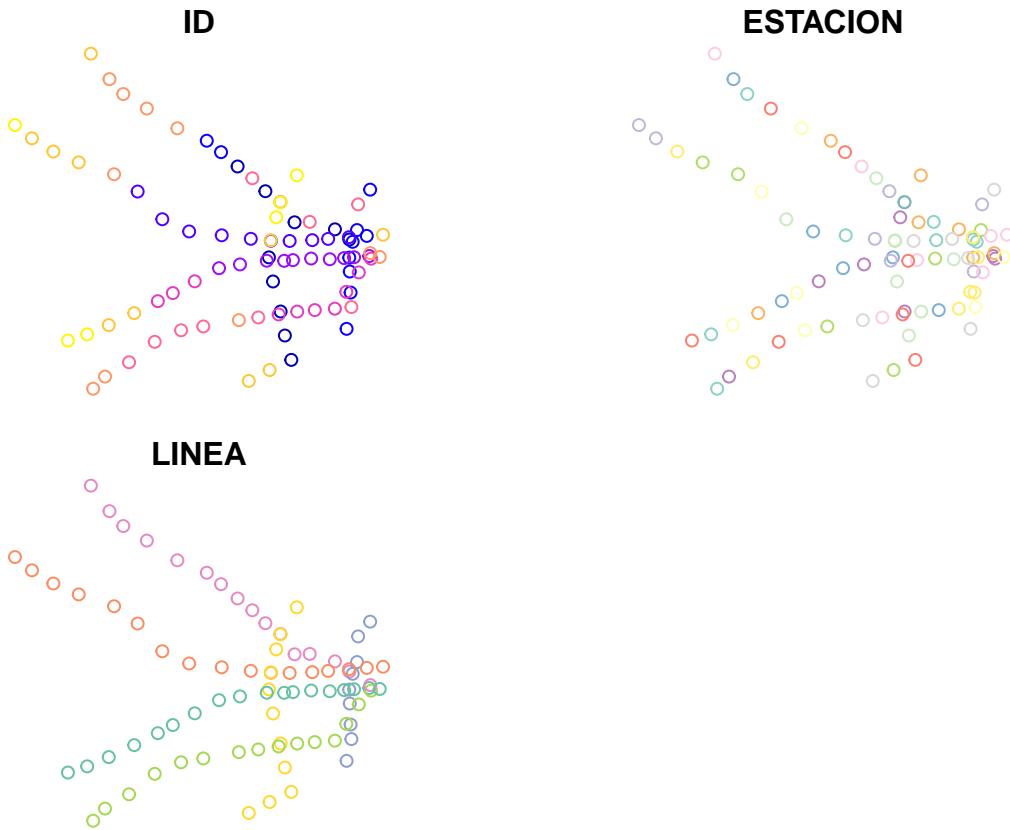
## LINEASUB



Se respeta la estructura del original.

### Estaciones

```
estaciones <- st_read(paste('data/estaciones', dir('data/estaciones', '\\\\.shp$'), sep='/'))  
  
## Reading layer `estaciones_de_subte' from data source `/home/pogo/Google Drive/EANT/LatinR/2018/data/  
## Simple feature collection with 86 features and 3 fields  
## geometry type: POINT  
## dimension: XY  
## bbox: xmin: 97881.68 ymin: 98442.25 xmax: 108564.5 ymax: 108167.8  
## epsg (SRID): NA  
## proj4string: +proj=tmerc +lat_0=-34.6297166 +lon_0=-58.4627 +k=0.9999980000000001 +x_0=100000 +y_0=0  
plot(estaciones)
```



En este caso vamos a querer conservar atributos de los datos.

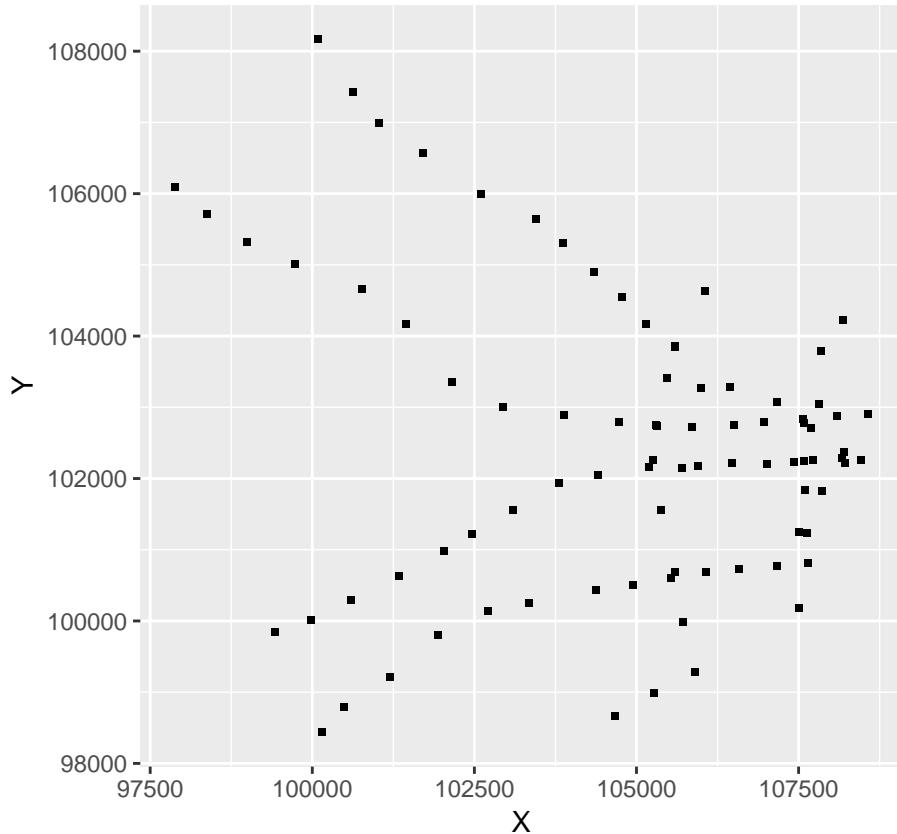
```
str(estaciones)

## Classes 'sf' and 'data.frame':  86 obs. of  4 variables:
## $ ID      : num  1 2 3 4 5 6 7 8 9 10 ...
## $ ESTACION: Factor w/ 83 levels "9 DE JULIO","ACOYTE",...: 16 35 34 83 53 1 30 78 3 68 ...
## $ LINEA   : Factor w/ 6 levels "A","B","C","D",...: 6 6 6 6 6 4 4 4 4 4 ...
## $ geometry:sfc_POINT of length 86; first list element: 'XY' num  105902 99279
## - attr(*, "sf_column")= chr "geometry"
## - attr(*, "agr")= Factor w/ 3 levels "constant","aggregate",...: NA NA NA
## ..- attr(*, "names")= chr "ID" "ESTACION" "LINEA"
```

Además de las coordenadas, vamos a querer mantener las líneas y los nombres de las estaciones.

```
estacionesDF <- data.frame(estaciones,stringsAsFactors = F) %>%
  select(LINEA,ESTACION) %>%
  mutate(LINEA=as.character(LINEA),
        ESTACION=as.character(ESTACION)) %>%
  bind_cols(data.frame(round(st_coordinates(estaciones))))
```

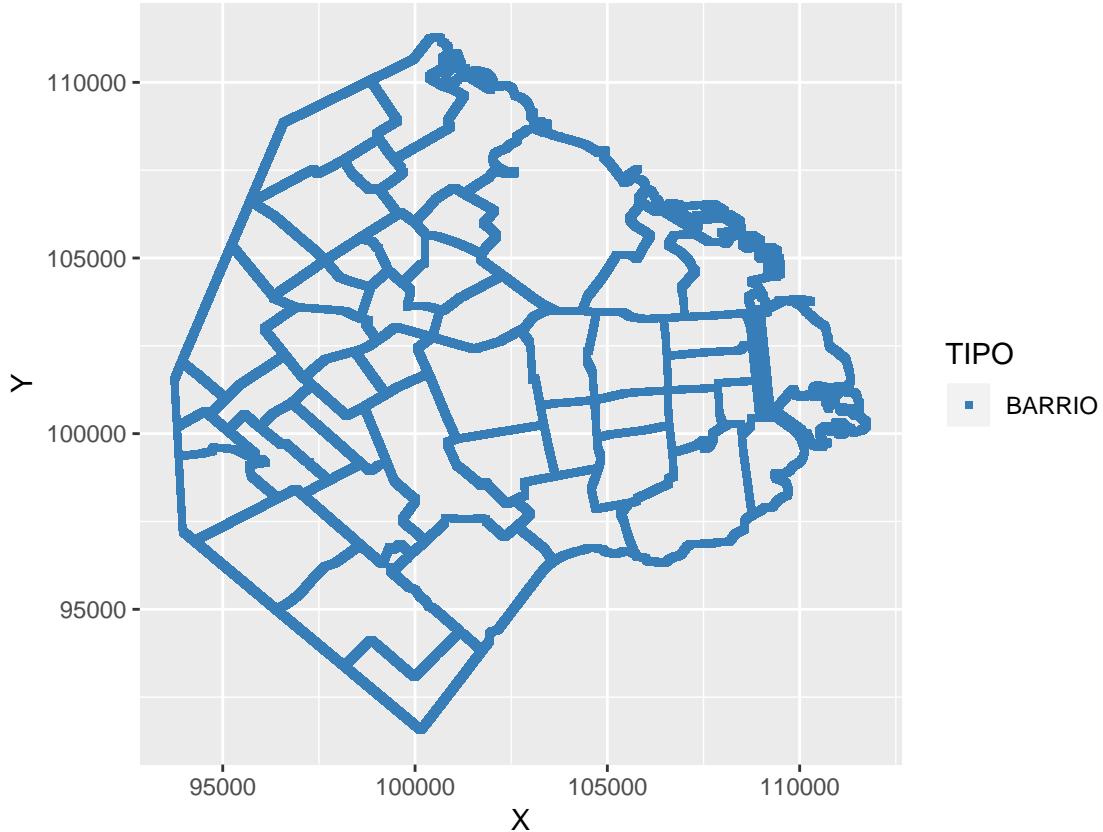
graficaCoord(estacionesDF)



## Representación de la ciudad

Para representar la ciudad, primero revisemos si las coordenadas quedaron alineadas

```
barriosCompleta %>%
  mutate(TIPO="BARRIO") %>%
  bind_rows(lineasCompleta %>%
             mutate(TIPO="LINEA")) %>%
  bind_rows(estacionesDF %>%
              mutate(TIPO="ESTACION")) %>%
  mutate(TIPO = factor(TIPO,levels=c('BARRIO','LINEA','ESTACION'))) %>%
  filter(TIPO == 'BARRIO') %>%
  ggplot(aes(x=X,y=Y, color=TIPO)) + geom_point(shape=15,size = 1) + scale_colour_manual(values=c('#3777AA', '#E69138', '#4DB6AC'))
```



## Generación del archivo .stl

Debido a que para generar el archivo, tenemos que generar una matriz de la superficie a imprimir, se utilizar el DataFrame como está, tendremos una matriz de  $3.551572e+08$  datos, por eso probemos escalar nuestros datos para reducir la dimensión del la matriz resultante.

```
escalar <- function(escala){
  barriosCompleta %>%
    mutate(TIPO="BARRIO",X=round(X/escala),Y=round(Y/escala)) %>%
    bind_rows(lineasCompleta %>%
      mutate(TIPO="LINEA",X=round(X/escala),Y=round(Y/escala))) %>%
    bind_rows(estacionesDF %>%
      mutate(TIPO="ESTACION",X=round(X/escala),Y=round(Y/escala))) %>%
    mutate(TIPO = factor(TIPO,levels=c('BARRIO','LINEA','ESTACION'))) %>%
    ggplot(aes(x=X,y=Y, color=TIPO)) + geom_point(shape=15,size=1) + scale_colour_manual(values=c('#3777AA'))
}

escalar(50)
```



Utilizando 50, dividimos por 2500 la cantidad de observaciones necesarias sin comprometer la calidad de la impresión.

Generamos el data.frame con las coordenadas en escala:

```
escala <- 50
df3D <- barriosCompleta %>%
  mutate(TIPO="BARRIO",altura=0.5,X=round(X/escala),Y=round(Y/escala)) %>%
  bind_rows(lineasCompleta %>%
    mutate(TIPO="LINEA",altura=0.75,X=round(X/escala),Y=round(Y/escala))) %>%
  bind_rows(estacionesDF %>%
    mutate(TIPO="ESTACION",altura=1,X=round(X/escala),Y=round(Y/escala)))
```

### Impresión de la Ciudad de Buenos Aires

La librería `r2stl` requiere que se le pasen como argumentos las coordenadas  $x$ ,  $y$ ,  $z$  para generar un archivo de extensión `.stl`, el cual se caracteriza por representar cuerpos geométricos utilizando triángulos.

Para facilitar la construcción, los puntos en el eje  $z$  los vamos a volcar en una matriz.

La representación estará contenida dentro de un cubo y, debido a que la librería va a forzar a que los bordes toquen el cubo (a pesar de ser un parámetro, empíricamente el mismo no funcionaría), para mantener la relación de aspecto, la matriz será cuadrada.

Veamos cuál es el eje que posee mayor rango:

```
print(paste('Rango del Eje X:',max(df3D$X)-min(df3D$X)))
```

```
## [1] "Rango del Eje X: 360"
```

```

print(paste('Rango del Eje Y:',max(df3D$Y)-min(df3D$Y)))

## [1] "Rango del Eje Y: 395"

Por lo tanto, vamos a hacer una matriz cuadrada de 395 X 395.

#Creamos una matriz de 1's para imprimir el mapa con cierta altura
matriz <- matrix(1,ncol=max(df3D$Y)-min(df3D$Y)+2,nrow=max(df3D$Y)-min(df3D$Y)+2)

# Para dar grosor a la linea, que cada elemento de la matriz tenga el mismo valor que sus contiguos
for (i in 0:2){
  for (j in 0:2){
    #Los lmites de los barrios los representaremos hundidos
    matriz[as.matrix(df3D %>%
      mutate(X=X-min(X)+i + round(((max(df3D$Y)-min(df3D$Y))-(max(df3D$X)-min(df3D$X)))
        Y=Y-min(Y)+j) %>%
        filter(TIPO=='BARRIO') %>%
        select(X,Y))] <- 0.98
  }
}

for (i in 0:2){
  for (j in 0:2){
    #Las líneas las representaremos sobresaliendo de la superficie
    matriz[as.matrix(df3D %>%
      mutate(X=X-min(X)+i + round(((max(df3D$Y)-min(df3D$Y))-(max(df3D$X)-min(df3D$X)))
        Y=Y-min(Y)+j) %>%
        filter(TIPO=='LINEA') %>%
        select(X,Y))] <- 1.02
  }
}

for (i in 0:2){
  for (j in 0:2){
    #Las estaciones sobresalen por encima de las líneas
    matriz[as.matrix(df3D %>%
      mutate(X=X-min(X)+i + round(((max(df3D$Y)-min(df3D$Y))-(max(df3D$X)-min(df3D$X)))
        Y=Y-min(Y)+j) %>%
        filter(TIPO=='ESTACION') %>%
        select(X,Y))] <- 1.04
  }
}

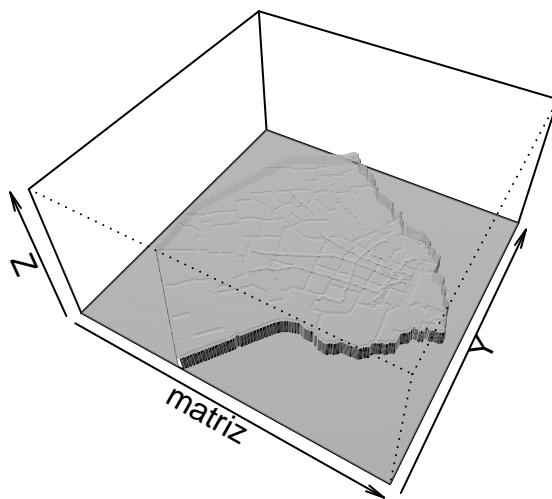
#Todo punto por fuera del contorno de la ciudad no tendrá altura
for (i in 1:nrow(matriz)){
  contorno <- which(matriz[i,] != 1)
  if (length(contorno)==0) {
    matriz[i,]<-0
  } else {
    if (contorno[1]>1){
      matriz[i,1:(contorno[1]-1)] <-0
    }
    if (contorno[length(contorno)]<ncol(matriz)){
      matriz[i,(contorno[length(contorno)]+1):ncol(matriz)] <-0
    }
  }
}

```

```
    }  
}  
  
#Debido a que la librería nos exige una forma cúbica, para mantener las proporciones ponemos un punto d  
matriz[1,1]<-10
```

Generemos una impresión en 3D

```
persp(matriz,theta = 30, phi = 45,border = NA, expand = 0.5 ,col = "white",shade= 0.75)
```



Creamos el directorio donde volcar nuestros archivos

```
dir.create('stl',showWarnings=FALSE)
```

Exportamos nuestros archivo.

```
r2stl(x=seq_len(nrow(matriz)),  
      y=seq_len(ncol(matriz)),  
      z=matriz,  
      file="stl/ciudad.stl")
```

## Impresión Cantidad de Usuarios por estación

### Unión de datos

Para representar la cantidad de usuarios por estación, debemos unir los datasets correspondientes. Recordemos cómo eran:

```

head(molinetesCantidad)

## # A tibble: 6 x 4
##   COMBINACION      cantidad LINEA ESTACION
##   <chr>            <dbl> <chr> <chr>
## 1 A|ACOYTE        350683  A     ACOYTE
## 2 A|ALBERTI       122973  A     ALBERTI
## 3 A|CARABOBO      286901  A     CARABOBO
## 4 A|CASTRO BARROS 234639. A    CASTRO BARROS
## 5 A|CONGRESO      333369. A    CONGRESO
## 6 A|FLORES        294259. A    FLORES

head(filter(df3D, TIPO=='ESTACION'))

```

```

## # A tibble: 6 x 5
##   X     Y     TIPO altura LINEA      ESTACION
##   <dbl> <dbl> <chr> <dbl> <chr>
## 1 2118 1986 ESTACION     1     H     CASEROS
## 2 2114 2000 ESTACION     1     H     INCLAN
## 3 2112 2014 ESTACION     1     H     HUMBERTO 1°
## 4 2107 2031 ESTACION     1     H     VENEZUELA
## 5 2105 2045 ESTACION     1     H     ONCE - 30 DE DICIEMBRE
## 6 2152 2055 ESTACION     1     D     9 DE JULIO

```

Veamos si los nombres coinciden

```

comprobacionNombres <- molinetesCantidad %>%
  full_join(filter(df3D, TIPO=='ESTACION') , by=c('LINEA','ESTACION')) %>%
  mutate(MOLINETES=is.na(cantidad),
        ESTACIONES=is.na(altura)) %>%
  select(LINEA,ESTACION,MOLINETES,ESTACIONES)

head(comprobacionNombres %>% filter(!(MOLINETES & ESTACIONES)) %>% arrange(LINEA,ESTACION))

```

```

## # A tibble: 6 x 4
##   LINEA ESTACION      MOLINETES ESTACIONES
##   <chr> <chr>        <lgl>     <lgl>
## 1 A     FLORES       TRUE      FALSE
## 2 A     PLAZA DE MISERERE FALSE     TRUE
## 3 A     PLAZA MISERERE TRUE      FALSE
## 4 A     SAN JOSÉ DE FLORES FALSE     TRUE
## 5 B     C. PELLEGRINI FALSE     TRUE
## 6 B     CALLAO        FALSE     TRUE

```

Como podemos observar algunos nombres no coinciden, por lo cual tendremos que limpiar los datos.

```

nombresMolinete <- (comprobacionNombres %>%
  filter(MOLINETES & !ESTACIONES) %>%
  select(LINEA,ESTACION) %>%
  arrange(LINEA) %>%
  group_by(LINEA) %>%
  nest()$data

nombresEstaciones <- (comprobacionNombres %>%
  filter(ESTACIONES & !MOLINETES) %>%
  arrange(LINEA) %>%
  select(LINEA,ESTACION) %>%
  group_by(LINEA) %>% nest()$data

```

Usando la función `agrep` hacemos Fuzzy Matching.

```
nombreEstacionesInter <- lapply(1:6,function(i){  
  ne <- nombresEstaciones[[i]]  
  nm <- nombresMolinetes[[i]]  
  ESTACION_FINAL <- sapply(nm$ESTACION,function(x) agrep(x,ne$ESTACION,value=TRUE,max.distance=0.2))  
  if (class(ESTACION_FINAL)=='list') {  
    ESTACION_FINAL <- do.call('c',ESTACION_FINAL)  
  }  
  nm$ESTACION_FINAL <- ESTACION_FINAL[nm$ESTACION]  
  nm$LINEA <- c(LETTERS[1:5], 'H')[i]  
  return(nm)  
}) %>% bind_rows()
```

Comprobemos si se realizó correctamente.

```
nombreEstacionesInter %>%  
  filter(ESTACION != ESTACION_FINAL | is.na(ESTACION_FINAL))
```

```
## # A tibble: 27 x 3  
##   ESTACION      ESTACION_FINAL     LINEA  
##   <chr>        <chr>          <chr>  
## 1 FLORES       SAN JOSÉ DE FLORES      A  
## 2 PLAZA MISERERE PLAZA DE MISERERE      A  
## 3 CALLAO.B     CALLAO           B  
## 4 ECHEVERRIA   ECHEVERRÍA         B  
## 5 LOS INCAS    DE LOS INCAS -PQUE. CHAS  B  
## 6 MALABIA      MALABIA - OSVALDO PUGLIESE  B  
## 7 PASTEUR       PASTEUR - AMIA         B  
## 8 ROSAS         JUAN MANUEL DE ROSAS - VILLA URQUIZA B  
## 9 TRONADOR     TRONADOR - VILLA ORTÚZAR  B  
## 10 CARLOS PELLEGRINI <NA>            B  
## # ... with 17 more rows
```

Como podemos observar, los casos que hizo el matching lo hizo correctamente. Los siguientes casos no pudo encontrar una correspondencia:

```
filter(nombreEstacionesInter, is.na(ESTACION_FINAL))
```

```
## # A tibble: 7 x 3  
##   ESTACION      ESTACION_FINAL LINEA  
##   <chr>        <chr>          <chr>  
## 1 CARLOS PELLEGRINI <NA>          B  
## 2 GENERAL SAN MARTIN  <NA>          C  
## 3 MARIANO MORENO    <NA>          C  
## 4 AVENIDA DE MAYO   <NA>          C  
## 5 AVENIDA LA PLATA  <NA>          E  
## 6 GENERAL BELGRANO <NA>          E  
## 7 FACULTAD DE DERECHO <NA>          H
```

Veamos los valores para solucionarlo a mano.

```
lapply(nombresEstaciones,function(x) x$ESTACION)  
  
## [[1]]  
## [1] "PLAZA DE MISERERE"  "SAN JOSÉ DE FLORES"  
##  
## [[2]]
```

```

## [1] "C. PELLEGRINI"
## [2] "CALLAO"
## [3] "PASTEUR - AMIA"
## [4] "MALABIA - OSVALDO PUGLIESE"
## [5] "TRONADOR - VILLA ORTÚZAR"
## [6] "DE LOS INCAS -PQUE. CHAS"
## [7] "ECHEVERRÍA"
## [8] "JUAN MANUEL DE ROSAS - VILLA URQUIZA"
##
## [[3]]
## [1] "AV. DE MAYO" "MORENO"      "SAN MARTIN"
##
## [[4]]
## [1] "TRIBUNALES - TEATRO COLÓN" "R.SCALABRINI ORTIZ"
## [3] "PUEYRREDON"
##
## [[5]]
## [1] "BELGRANO"                  "INDEPENDENCIA"
## [3] "ENTRE RIOS - RODOLFO WALSH" "AV. LA PLATA"
## [5] "PLAZA DE LOS VIRREYES - EVA PERON"
##
## [[6]]
## [1] "HUMBERTO 1°"               "ONCE - 30 DE DICIEMBRE"
## [3] "PARQUE PATRICIOS"          "CÓRDOBA"
## [5] "SANTA FE - CARLOS JAUREGUI"

```

Como podemos observar, en los datos de referencia geográfica, aún no están los correspondientes a la estación FACULTAD DE DERECHO de la línea H.

Procedemos a crear una tabla con los nombres finales.

```

nombreEstacionesInter <- nombreEstacionesInter %>%
  filter(ESTACION != 'FACULTAD DE DERECHO') %>%
  mutate(ESTACION_FINAL = case_when(LINEA == 'B' & ESTACION == 'CARLOS PELLEGRINI' ~ 'C. PELLEGRINI',
                                     LINEA == 'C' & ESTACION == 'AVENIDA DE MAYO' ~ 'AV. DE MAYO',
                                     LINEA == 'C' & ESTACION == 'GENERAL SAN MARTIN' ~ 'SAN MARTIN',
                                     LINEA == 'C' & ESTACION == 'MARIANO MORENO' ~ 'MORENO',
                                     LINEA == 'E' & ESTACION == 'AVENIDA LA PLATA' ~ 'AV. LA PLATA',
                                     LINEA == 'E' & ESTACION == 'GENERAL BELGRANO' ~ 'BELGRANO',
                                     TRUE ~ ESTACION_FINAL)) %>%
  bind_rows(comprobacionNombres %>%
    filter(MOLINETES & ESTACIONES) %>%
    mutate(ESTACION_FINAL = ESTACION) %>%
    select(ESTACION, ESTACION_FINAL, LINEA))

```

Generamos la tabla final.

```

estacionesTOTAL <- molinetesCantidad %>%
  inner_join(nombreEstacionesInter, by = c('LINEA', 'ESTACION')) %>%
  mutate(ESTACION = ESTACION_FINAL) %>%
  inner_join(filter(df3D, TIPO == 'ESTACION'), by = c('LINEA', 'ESTACION')) %>%
  group_by(COMBINACION) %>%
  summarise(cantidad = round(sqrt(mean(cantidad))), X = round(mean(X)),
            Y = round(mean(Y))) %>%
  ungroup() %>%

```

```

arrange(cantidad)

head(estacionesTOTAL)

## # A tibble: 6 x 4
##   COMBINACION      cantidad      X      Y
##   <chr>          <dbl> <dbl> <dbl>
## 1 E|MEDALLA MILAGROSA     217  2024  1984
## 2 E|VARELA            241  2010  1976
## 3 E|PICHINCHA         250  2121  2014
## 4 E|ENTRE RIOS        256  2132  2014
## 5 E|SAN JOSE           261  2143  2015
## 6 E|GENERAL BELGRANO    273  2157  2036

```

## Generación de la Matriz

Ahora cada punto donde se ubique una estación no va a tener un valor fijo, sino que el valor de Z los dará la cantidad de personas que haya utilizado la estación.

```

matriz <- matrix(1, ncol=max(df3D$Y)-min(df3D$Y)+2, nrow=max(df3D$Y)-min(df3D$Y)+2)

for (i in 0:2){
  for (j in 0:2){
    matriz[as.matrix(df3D %>%
                      mutate(X=X-min(X)+i + round(((max(df3D$Y)-min(df3D$Y))-(max(df3D$X)-min(df3D$X)))
                      Y=Y-min(Y)+j) %>%
                      filter(TIPO=='BARRIO') %>%
                      select(X,Y))] <- 0.98
  }
}

for (i in -1:3){
  for (j in -1:3){
    matriz[as.matrix(df3D %>%
                      mutate(X=X-min(X)+i + round(((max(df3D$Y)-min(df3D$Y))-(max(df3D$X)-min(df3D$X)))
                      Y=Y-min(Y)+j) %>%
                      filter(TIPO=='LINEA') %>%
                      select(X,Y))] <- 1.02
  }
}

#En las coordenadas correspondientes a las estaciones, la altura será determinada por la cantidad de personas
#El ancho debe ser mayor ya que de ser muy fina, la columna vertical puede caerse
for (i in -2:4){
  for (j in -2:4){
    #Se le da el valor correspondiente, escalado a 9
    matriz[cbind(estacionesTOTAL$X-min(df3D$X)+ i + round(((max(df3D$Y)-min(df3D$Y))-(max(df3D$X)-min(df3D$X))
    estacionesTOTAL$Y-min(df3D$Y)+ j)] <- (9/max(estacionesTOTAL$cantidad))*estacionesTOTAL$cantidad
  }
}

for (i in 1:nrow(matriz)){
  contorno <- which(matriz[i,] != 1)

```

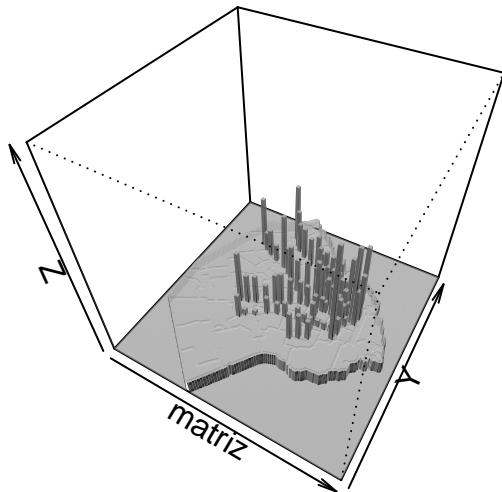
```

if (length(contorno)==0) {
  matriz[i,]<-0
} else {
  if (contorno[1]>1){
    matriz[i,1:(contorno[1]-1)] <-0
  }
  if (contorno[length(contorno)]<ncol(matriz)){
    matriz[i,(contorno[length(contorno)]+1):ncol(matriz)] <-0
  }
}
}

matriz[1,1] <- 15

#Cambiar por persp3d en caso de usar modo interactivo
persp(matriz,theta = 30, phi = 45,border = NA, expand = 1 ,col = "white",shade= 0.75)

```



Exportamos el archivo

```

r2stl(x=seq_len(nrow(matriz)),
      y=seq_len(ncol(matriz)),
      z=matriz,
      file="stl/ciudadBarras.stl")

```

## Contacto

Twitter: <http://www.twitter.com/lucpogo>

Github: <http://www.github.com/lucpogo/impresion3D>