Creating a plot with TOP10NL map files

Han Oostdijk (www.hanoostdijk.nl)

March 22, 2016

Introduction

In this document I show you how to create a detailed map of an area in the Netherlands. I will use vector files that are are created by **Het Kadaster** (the Dutch Cadastre, Land Registry and Mapping Agency). For more information about these files (in Dutch only) see TOP10NL. They were downloaded in **gml** format from www.pdok.nl. As examples I choose Amstelveen and Schiphol because that is the place where I live (in the neighbourhood), but the code will also work for any other area in the Netherland. Because the information of Amstelveen is splitted over two map sheets we needed the files **TOP10NL_25W.gml** and **TOP10NL_25O.gml** that were downloaded on 11 and 15 March 2016. For Schiphol we need only the **TOP10NL_25W.gml** file.

*I assume this method also works for places outside the Netherlands but in that case the inputs maps have to be sourced from elsewhere. The functions that actually read and write the map files **readOGR** and **writeOGR** can also handle other map formats than **gml**.*

Information in the files

The TOP10NL page gives some information about the contents of the TOP10NL files. Because the webpage is in Dutch and the code needs to name the topographical elements (or layers) in Dutch I will translate the names of the elements used here:

- wegdeel: road
- spoorbaandeel: railway
- waterdeel: water
- gebouw: building
- plaats: locality
- functioneel gebied: area with specific function

Each element can have one or more of the geometries **wkbPolygon**, **wkbLineString** or**wkbPoint**. These would be plotted by ggplot2 with the corresponding functions **geom_polygon**, **geom_path** and **geom_point**.

Method used

I used vector files. Apart from the fact that they are nicely scalable, they give me full information about the various topographical elements (layers). So I can determine exactly which elements to use and how. E.g. I decided to include in the map the information of the layer **FunctioneelGebied** but only when that is related to sport and to color it green. If I would have information about e.g. the number of yearly visitors I could color the sport areas based on that number. And by colouring the runways of the airport in yellow the extension of the airport becomes more apparent.

The vector files are rather large files: the unzipped gml files for the two map sheets (for Amstelveen) are respectively about 700 and 500 MB. Therefore I decided to select from these files only the information related to Amstelveen and Schiphol. This is done in the function **copy_layer01**. In this function we select the elements in the layer that are related to the location. (We say that an element in a layer is related to Amstelveen resp. Schiphol if the intersection of the element with the *bbox* of Amstelveen resp. Haarlemmermeer (the municipality of Schiphol) is not empty). In the case of Amstelveen we will have two calls to **copy_layer01** (one for each of the two sheets) and the resulting files are then combined with the function **combine_layers1**. This is done for all combination of layers and geometries in the function **handle_layers**.

The template variables *dsnt0* and *dsnt1* in combination with the variables *sheet1*, *sheet2* and *sheet3* determine the names of the files that we will use. Logically seen we have four types of file:

- the input files from the kadaster. The name is generated when *level==0* and parameter *sheet* (that can be *sheet1*or *sheet2*) determines which of two names is generated from template *dsnt0*.
- the files that are directly created from the input files. The name is generated when *level==1* and parameter *sheet* (that can be *sheet1* or *sheet2*) determines in combination with the parameters *layer* and *geom* which name is generated from template *dsnt1*.
- the combined files. The name is generated when *level==1* and parameter *sheet* (that can only be *sheet3* determines in combination with the parameters *layer* and *geom* which name is generated from template *dsnt1*.
- the *bbox* file. The name is generated with *level==1*, parameter *sheet=sheet1*, *layer='pol'*, *geom='bbox'* and *suf='rds'* from template *dsnt1*.

With the values of the constants in function **handle_object** we will use four folders for *map_object* Amstelveen:

- 'D:/data/maps: that contains the input files TOP10NL_25W.gml and TOP10NL_25O.gml
- 'D:/data/maps/Aveen25W': for the files created for sheet '25W'
- 'D:/data/maps/Aveen25O': for the files created for sheet '25O'
- 'D:/data/maps/Aveen25WO': for the combined files

and two files for *map_object* Schiphol (only one sheet is needed):

- 'D:/data/maps: that contains the input files TOP10NL_25W.gml
- 'D:/data/maps/Schiphol25W': for the files created for sheet '25W'

NB. These folders have to exist before running the code.

Description of the code

In the following description of the R-functions we will start with the auxiliary functions that are the same for the two cases Amstelveen en Schiphol. At the end we will discuss the functions **handle_object**, **plot_Amstelveen** and **plot_Schiphol** that are location specific. For another location only the last function need to be adapted.

Used libraries

```r
library(dplyr)
library(ggplot2)
library(magrittr)
library(rgdal)
library(rgeos)
library(maptools)
```

Prepare data

The first functions mentioned here are used to read and write and select the information about the layers.

Define name_dsn function

We start by defining the function **name_dsn** that generates file names from the variables in **handle_object**. See the last section in Method used how they are used there.

```
name_dsn     <- function (sheet, layer, geom, level=1, suf='gml') {
  if (level == 0) {
    nd = sprintf(dsnt0,sheet,suf)
  } else{
    nd = sprintf(dsnt1,sheet,layer, geom, suf)
  }
  return(nd)
}
```

Function that calculates the *bbox* of a location

The first thing to do is determining the *bbox* of a location (here Amstelveen or Schiphol). The *bbox* is the smallest rectangle with sides 'parallel' to the parallels and meridians. It is needed in the form of a *wkbPolygon*. An already saved version will be reused unless explicitly requested otherwise. The function **create_load_bbox** with its auxiliary functions **bbox2pol** and **get_pol_bbox** ensures that the variables *pol_bbox_1* becomes available for the **copy_layer01** function. See copy_layer01 how it is used there. The variable *sel1* (copied from the *bbox_sel* variable in **handle_object**) determines for which location the *bbox* is calculated.

```
bbox2pol    <- function (bb) {
  paste('POLYGON((',
    bb[1,1],bb[2,1], ',',
    bb[1,2],bb[2,1], ',',
    bb[1,2],bb[2,2], ',',
    bb[1,1],bb[2,2], ',',
    bb[1,1],bb[2,1],'))'
  )
}

get_pol_bbox <- function(sheet1, layer1,geom1, sel1) {
  spdf          = read_layer(sheet1, layer1, geom1, level=0)
  if (!is.null(spdf)) {
    spdf        = spdf[grepl(sel1,spdf$naamNL,fixed=F),]
    bbox_lg     = spdf@bbox
    pol_bbox    = readWKT(bbox2pol(bbox_lg))
    proj4string(pol_bbox) <- proj4string(spdf)
    return(pol_bbox)
  } else {
    return(NULL)
  }
}

create_load_bbox <- function (sheet1, layer1, geom1, sel1,
                              create = bbox_create) {
  outsds = name_dsn(sheet1, 'pol', 'bbox', suf = 'rds')
  if (file.exists(outsds) == F) {
    create = T
  }
  if (create == T) {
    pol_bbox_1 = get_pol_bbox(sheet1, layer1, geom1, sel1)
    saveRDS(pol_bbox_1, outsds)
  } else {
    pol_bbox_1 = readRDS(outsds)
  }
  assign('pol_bbox_1', pol_bbox_1, envir = parent.env(environment()))
}
```

Functions that read information from a layer

The function that actually reads the information from a graphical file is **readOGR** from the **rgdal** package. Because we try to hide as much as possible the messages given this function we call this function via the **readOGR_logm** function that can handle errors (e.g. trying to read a layer-geom combination that does not exist) and suppress messages. **readOGR_logm** is called by **read_layer** with the parameters *sheet*, *layer*, *geom* and *level*.

NB. in the sections were we actually use these functions (Call main function for Schiphol) and(Call main function for Amstelveen) you will see that not all messages are suppressed. This document is created in **knitr** and its author Yihui Xie indicates that the *sink-solution* will not fully work in **knitr**.

```
fmt_msg      <- function (dsn1,layer1,geom1) {
  paste0("dsn '",dsn1,"' layer '",layer1,"' geom '",geom1,"'")
}

readOGR_logm <- function (dsn1, layer1, geom1, logm = F, tc = T) {
  if (logm == F) {
    con <- file("testOGR.logm")
    sink(con, append = TRUE)
    sink(con, append = TRUE, type = "message")
  }
  if (tc == T) {
    spdf <- tryCatch(
      {
        readOGR(dsn1, layer1, require_geomType = geom1, verbose=F)
      },
      error   = function(cond) { return(NULL) },
      warning = function(cond) { return(NULL) },
      finally={ }
    )
  } else {
    spdf = readOGR(dsn1, layer1, require_geomType = geom1, verbose=F)
  }
  if (logm == F) {
    sink(NULL)
    unlink("testOGR.logm")
  }
  return(spdf)
}

read_layer   <-   function (sheet1, layer1, geom1,
                level = 1, verbose = F) {
    dsn1        = name_dsn(sheet1, layer1, geom1, level)
    if (verbose == T) {
      fmt         = fmt_msg(dsn1, layer1, geom1)
      cat(paste0("--- \n start reading from ", fmt, "\n"))
    }
    spdf    = readOGR_logm (dsn1, layer1, geom1, logm = verbose)
    if (verbose == T) {
      if (! is.null(spdf)) {
        cat(paste('data frame has dimensions:',
          x=paste(dim(spdf@data),collapse=' ')),'\n')
        cat('proj4string:\n')
        cat(strwrap(proj4string(spdf)), sep = "\n")
      }
      cat(paste0("--- \n end reading from ", fmt, "\n"))
    }
    return(spdf)
}
```

Function that writes layer information to a file

The function that actually writes the information to a graphical file is **writeOGR** from the **rgdal** package. The extra functionality in **write_layer** is that an existing output file is removed before trying to write to it. The **gml** driver has no update possibility and does not replace the file when it already exists. NB: I have not tried other graphical output formats

```
write_layer   <- function (spdf, sheet1, layer1, geom1, level=1) {
  dsn1        = name_dsn(sheet1, layer1, geom1, level)
  unlink(dsn1)
  writeOGR(spdf, dsn1, layer1, driver = 'GML')
  cat(paste0("--- layer information written to ",dsn1, "\n"))
}
```

Function that copies only the relevant part of a layer

In the **copy_layer01** function we read the information for a layer-geom combination from a Kadaster input file. If there is no information the **read_layer** function will return NULL and the function ends without a message. If **read_layer** returns a SpatialDataFrame its elements are checked for intersection with the *bbox* given as parameter. Only the elements for which the intersection is non empty are retained in the SpatialDataFrame that is then written to disk. If there are no intersecting elements a warning is given.

```
copy_layer01   <- function (sheet1, layer1, geom1, pol_bbox_1) {
  spdf        = read_layer(sheet1, layer1, geom1, level=0)
  if (!is.null(spdf)) {
    g_inters     = gIntersects(spdf, pol_bbox_1, byid = T)
    spdf         = spdf[as.vector(g_inters),]
    if (nrow(spdf) > 0) {
      spdf       = spTransform(spdf, CRS("+init=epsg:4238"))
      proj4string(spdf) <-CRS("+init=epsg:4238")
      write_layer(spdf, sheet1,layer1, geom1)
    } else {
      fmt = fmt_msg(name_dsn(sheet1, layer1, geom1, level=0),layer1, geom1)
      cat(paste0("nothing in selection for ", fmt, "\n"))
    }
  }
}
```

Function that combines the information of two sheets.

Because the information for Amstelveen is in two sheets it is convenient to combine the information from the (at most) two files in one file. The **combine_layers1** function reads the information for a layer-geom combination for both sheets in the variables *lg1* and *lg2*. They both can be a SpatialDataFrame or NULL. In the interesting case that both contain data we ensure that only the data columns that are in both DataFrames are retained. And we only keep data from the second DataFrame that is not already in the first DataFrame before using *rbind* to combine them. The resulting SpatialDataFrame is written to disk. If only one of *lg1* and *lg2* is not NULL then that one is written to disk.

```r
combine_layers1 <- function(sheet1, sheet2, sheet3, layer1, geom1) {
  lg1 = read_layer(sheet1, layer1, geom1)
  lg2 = read_layer(sheet2, layer1, geom1)
  if ((!is.null(lg1)) & (!is.null(lg2))) {
    n1       = names(lg1@data)
    n2       = names(lg2@data)
    n12      = n1 [n1 %in% n2]
    lg1@data = lg1@data[, n12]
    lg2@data = lg2@data[, n12]
    s2       = rownames(lg2@data) %in% rownames(lg1@data)
    lg2      = lg2[!s2,]
    lg3      = rbind(lg1, lg2)
  } else if (!is.null(lg1)) {
    lg3 = lg1
  } else if (!is.null(lg2)) {
    lg3 = lg2
  } else {
    lg3 = NULL
  }
  if (!is.null(lg3)) {
    proj4string(lg3) <- CRS("+init=epsg:4238")
    write_layer(lg3, sheet3,layer1, geom1)
  }
}
```

Function that reads and combines all layer-geom combinations.

For all layer-geom combinations (that we are interested in) we read the data from the Kadaster input files for both sheets and combine them.

```r
handle_layers <- function(sheet1,sheet2,sheet3, only1sheet = F) {
  xlayers = c("Plaats", "Wegdeel", "Waterdeel",
    "Gebouw", "Spoorbaandeel", "FunctioneelGebied")
  xgeoms = c('wkbPolygon', 'wkbPoint', 'wkbLineString')
  for (xlayer in xlayers) {
    for (xgeom in xgeoms) {
      cat(paste('handle_layers: layer', xlayer, 'with geom', xgeom),'\n')
      copy_layer01(sheet1, xlayer, xgeom, pol_bbox_1)
      if (only1sheet == F) {
        copy_layer01(sheet2, xlayer, xgeom, pol_bbox_1)
        combine_layers1(sheet1, sheet2, sheet3, xlayer, xgeom)
      }
    }
  }
}
```

Select the information we need for the plots

After having prepared the (combined) files we now will select the information that we need for plotting our map.

Functions to use for all locations

Because we will use the **ggplot2** package we need the coordinates for each of the elements (groups) in each layer-geom combination in *data.frame* format. Therefore we define the function **get_plot_data** that converts the SpatialDataFrame for each combination to a data.frame. We also define functions to format the coordinates.

In the functions that are specific for a location (**handle_object**, **plot_Amstelveen** and **plot_Schiphol** that are listed later on) we will further filter, select and manipulate this data using functionality from the **dplyr** and **magrittr** packages.

*NB. in the function **format_WENS** the formatting is less than optimal: trailing zeros disappear. However I could not get working the combination of **ggplot2** and **bquote** in this case.*

```
get_plot_data = function (sheet1, layer1, geom1) {
  cat(paste('get_plot_data:'),sheet1,layer1,geom1,'\n')
  gl   = read_layer(sheet1, layer1, geom1)
  gl.f = fortify(gl)
  gl@data$id = rownames(gl@data)
  gl.f = merge(gl.f,  gl@data,
    by.x = "id",  by.y = "id")
}

format_WE <- function(x,dig=3) {
  format_WENS(x,'WE',dig)
}

format_NS <- function(x,dig=3) {
  format_WENS(x,'NS',dig)
}

format_WENS <- function(x,WENS,dig=3) {
  if (WENS=='WE') {
    Z1 = 'W' ; Z2 = 'E'
  } else {
    Z1 = 'S' ; Z2 = 'N'
  }
  f  = sprintf('%%.%.0ff',dig)
  xf = sprintf(f,abs(x))
  Z  = ifelse(x < 0, Z1, ifelse(x > 0, Z2,''))
  # e= bquote(.(xf)*degree*~.(quote(Z)))
  # e=as.expression(e) # fails in plot why ?
  f = parse(text = paste(xf, "*degree~", Z),keep.source = F)
}
```

Plots functions specific for a location

The two functions **plot_Schiphol** and **plot_Amstelveen** determine what will be plotted. They are specific for a location but very similar. The main difference is the specification of the location to plot with the **coord_cartesian** function. Another difference is that for Amstelveen the location proper has been given a yellow color.

```
plot_Schiphol <- function (e) {
  # specify general aesthetics:
  p = ggplot( e$pol_sport,aes(x=long,y=lat,group = group)) +
    # select (in-)precise location to plot
    coord_cartesian(xlim = c(4.7, 4.82),
      ylim = c(52.27, 52.37)) +
    # include railways (lines)
    geom_path(data=e$lne_rail,color='red',size=2,alpha=0.9) +
    # include sporting fields (polygons)
    geom_polygon(data=e$pol_sport,
      color='green',fill='green',alpha=0.3) +
    # include buildings (polygons)
    geom_polygon(data=e$pol_gebouw,
      color='brown',fill='brown',alpha=0.5) +
    # include water areas (polygons)
    geom_polygon(data=e$pol_water,
      color='blue',fill='blue',alpha=0.3) +
    # include water areas (lines)
    geom_path(data=e$lne_water,color='blue') +
    # include roads and runways (polygons)
    geom_polygon(data=e$pol_weg,
      color='black',fill=e$pol_weg$kleur)   +
    # specify axis labels
    labs(x = "longitude", y = "latitude") +
    # specify format for axis tick labels
    scale_x_continuous(labels=format_WE) +
    scale_y_continuous(labels=format_NS)
  print(p)
}

plot_Amstelveen <- function (e) {
  # specify general aesthetics:
  p =ggplot( e$pol_plaats,aes(x=long,y=lat,group = group)) +
    # select (in-)precise location to plot
    coord_cartesian(xlim = c(4.815, 4.89),
      ylim = c(52.265, 52.325)) +
    # include location background (polygon)
    geom_polygon(data=e$pol_plaats,
      color='yellow',fill='yellow',alpha=0.3) +
    # include railways (lines)
    geom_path(data=e$lne_rail,color='red',size=2,alpha=0.9) +
    # include sporting fields (polygons)
    geom_polygon(data=e$pol_sport,
      color='green',fill='green',alpha=0.3) +
    # include buildings (polygons)
    geom_polygon(data=e$pol_gebouw,
      color='brown',fill='brown',alpha=0.5) +
    # include water areas (polygons)
    geom_polygon(data=e$pol_water,
      color='blue',fill='blue',alpha=0.3) +
    # include water areas (lines)
    geom_path(data=e$lne_water,color='blue') +
     # include roads (polygons) (no runways in Amstelveen)
    geom_polygon(data=e$pol_weg,
      color='black',fill=e$pol_weg$kleur) +
    # specify axis labels
    labs(x = "longitude", y = "latitude")  +
    # specify format for axis tick labels
    scale_x_continuous(labels=format_WE) +
    scale_y_continuous(labels=format_NS)

  print(p)
}
```

Main function

The function **handle_object** is the main function. It sets parameters for the two cases (Amstelveen or Schiphol) and calls the functions that extract the information from the map files. And after that it calls the appropriate plot function.

```
handle_object <- function (map_object,plot_only=F) {
  if (map_object == 'Schiphol') {
    bbox_sel       =  'Haarlemmermeer'
    bbox_layer     =  'RegistratiefGebied'
    bbox_geom      =  'wkbPolygon'
    bbox_create    =  F
    only1sheet     =  T
    sheet1         =  '25W'
    sheet2         =  NULL
    sheet3         =  '25W'
    dsnt0          =  'D:/data/maps/TOP10NL_%s.%s'
    dsnt1          =  'D:/data/maps/Schiphol%s/%s_%s.%s'
  } else {
    bbox_sel       =  'Amstelveen'
    bbox_layer     =  'RegistratiefGebied'
    bbox_geom      =  'wkbPolygon'
    bbox_create    =  F
    only1sheet     =  F
    sheet1         =  '25W'
    sheet2         =  '25O'
    sheet3         =  '25WO'
    dsnt0          =  'D:/data/maps/TOP10NL_%s.%s'
    dsnt1          =  'D:/data/maps/Aveen%s/%s_%s.%s'
  }
  ep = parent.env(environment())
  assign('dsnt0',dsnt0,ep)
  assign('dsnt1',dsnt1,ep)

  if (plot_only==F) {
    create_load_bbox(sheet1, bbox_layer, bbox_geom, bbox_sel,
      create = bbox_create)
    handle_layers(sheet1, sheet2, sheet3, only1sheet = only1sheet)
  }

  if (map_object == 'Amstelveen') {
    pol_plaats    = get_plot_data(sheet3, 'Plaats', 'wkbPolygon')
    pol_plaats    = pol_plaats %>%
      filter(naamNL == 'Amstelveen') %>%
      select(long, lat, group, naamNL)
  }
  pol_weg       = get_plot_data(sheet3, 'Wegdeel', 'wkbPolygon')
  pol_weg       = pol_weg %>%
    mutate(kleur = ifelse(
      typeWeg1 %in% c('startbaan, landingsbaan', 'rolbaan, platform'),
      'yellow', 'grey' )) %>%
    select(long, lat, group, kleur)
  pol_water     = get_plot_data(sheet3, 'Waterdeel', 'wkbPolygon')
  pol_water     = pol_water %>%
    select(long, lat, group)
  lne_water     = get_plot_data(sheet3, 'Waterdeel', 'wkbLineString')
  lne_water     = lne_water %>%
    select(long, lat, group)
  pol_gebouw   = get_plot_data(sheet3, 'Gebouw', 'wkbPolygon')
  pol_gebouw   = pol_gebouw %>%
    select(long, lat, group)
  lne_rail     = get_plot_data(sheet3, 'Spoorbaandeel', 'wkbLineString')
  lne_rail     = lne_rail %>%
    filter(typeSpoorbaan  %in% c('trein')) %>%
    select(long, lat, group)
  pol_sport    = get_plot_data(sheet3, 'FunctioneelGebied', 'wkbPolygon')
  pol_sport    = pol_sport %>%
    filter(
      typeFunctioneelGebied %in%
        c('sportterrein, sportcomplex', 'tennispark', 'golfterrein') ) %>%
    select(long, lat, group)
  e = environment()
  if (map_object == 'Amstelveen') {
    plot_Amstelveen(e)
  } else {
    plot_Schiphol(e)
  }
}
```

Actual calls of the main function

Here we show only the reporting part of the main function by using *plot_only=T*.

Call main function for Schiphol

See resulting plot in Figure 1 on page 11.

```
handle_object('Schiphol',plot_only=T)
```

```
## get_plot_data: 25W Wegdeel wkbPolygon

## NOTE: keeping only 47554 wkbPolygon of 47554 features

## Regions defined for each Polygons

## get_plot_data: 25W Waterdeel wkbPolygon

## NOTE: keeping only 4100 wkbPolygon of 4100 features
##
## Regions defined for each Polygons

## get_plot_data: 25W Waterdeel wkbLineString

## NOTE: keeping only 13527 wkbLineString of 13527 features

## get_plot_data: 25W Gebouw wkbPolygon

## NOTE: keeping only 35524 wkbPolygon of 35524 features
##
## Regions defined for each Polygons
## get_plot_data: 25W Spoorbaandeel wkbLineString

## NOTE: keeping only 272 wkbLineString of 272 features

## get_plot_data: 25W FunctioneelGebied wkbPolygon

## NOTE: keeping only 270 wkbPolygon of 270 features
##
## Regions defined for each Polygons
```

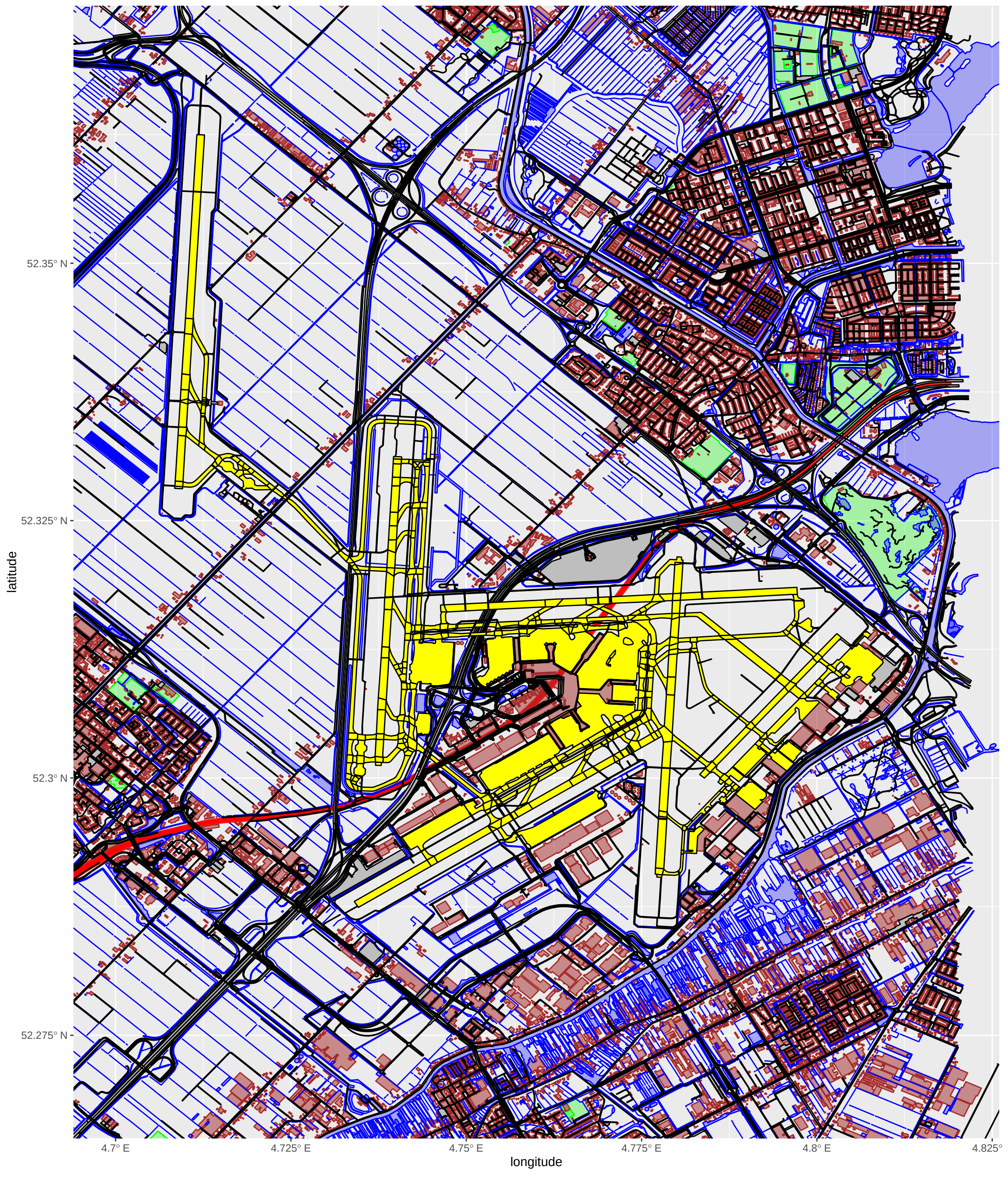

Figure 1: Map for Schiphol

Call main function for Amstelveen

See resulting plot in Figure 2 on page 13.

```r
handle_object('Amstelveen',plot_only=T)
```

```
## get_plot_data: 25WO Plaats wkbPolygon

## NOTE: keeping only 7 wkbPolygon of 7 features

## Regions defined for each Polygons

## get_plot_data: 25WO Wegdeel wkbPolygon

## NOTE: keeping only 10970 wkbPolygon of 10970 features
##
## Regions defined for each Polygons

## get_plot_data: 25WO Waterdeel wkbPolygon

## NOTE: keeping only 1125 wkbPolygon of 1125 features
##
## Regions defined for each Polygons

## get_plot_data: 25WO Waterdeel wkbLineString

## NOTE: keeping only 3346 wkbLineString of 3346 features

## get_plot_data: 25WO Gebouw wkbPolygon

## NOTE: keeping only 6034 wkbPolygon of 6034 features
##
## Regions defined for each Polygons

## get_plot_data: 25WO Spoorbaandeel wkbLineString

## NOTE: keeping only 37 wkbLineString of 37 features

## get_plot_data: 25WO FunctioneelGebied wkbPolygon

## NOTE: keeping only 56 wkbPolygon of 56 features
##
## Regions defined for each Polygons
```

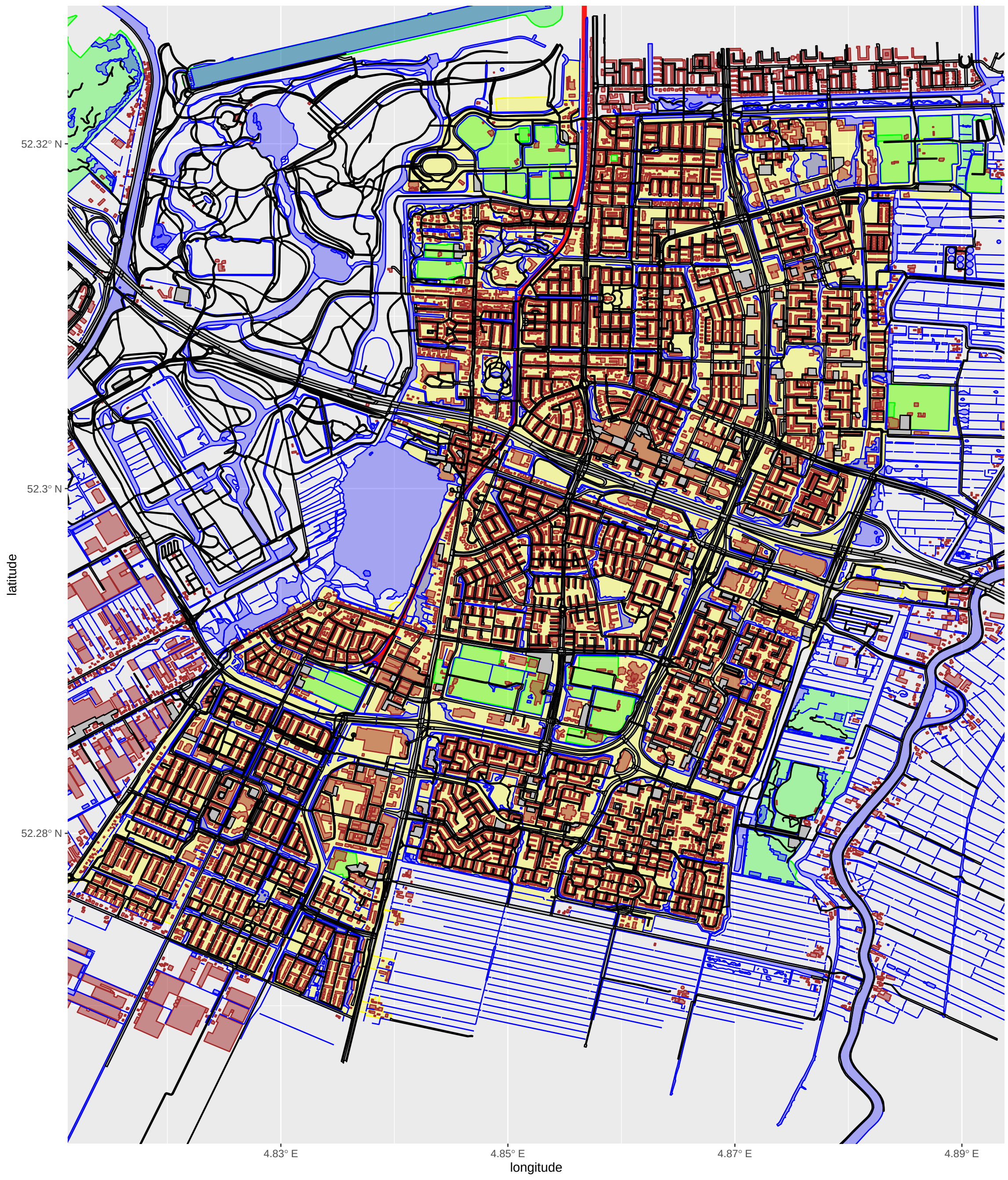

Figure 2: Map for Amstelveen

Session Info

```r
sessionInfo()
```

```
## R version 3.2.4 (2016-03-10)
## Platform: x86_64-w64-mingw32/x64 (64-bit)
## Running under: Windows 10 x64 (build 10586)
##
## locale:
## [1] LC_COLLATE=English_United States.1252  LC_CTYPE=English_United States.1252
## [3] LC_MONETARY=English_United States.1252 LC_NUMERIC=C
## [5] LC_TIME=English_United States.1252
##
## attached base packages:
## [1] stats     graphics  grDevices utils     datasets  methods   base
##
## other attached packages:
## [1] knitr_1.12.3    maptools_0.8-39 rgeos_0.3-17    rgdal_1.1-3     sp_1.2-2
## [6] magrittr_1.5    ggplot2_2.1.0   dplyr_0.4.3
##
## loaded via a namespace (and not attached):
##  [1] Rcpp_0.12.3      munsell_0.4.3    colorspace_1.2-6 lattice_0.20-33  R6_2.1.2
##  [6] stringr_1.0.0    plyr_1.8.3       tools_3.2.4      parallel_3.2.4   grid_3.2.4
## [11] gtable_0.2.0     DBI_0.3.1        htmltools_0.3    lazyeval_0.1.10  yaml_2.1.13
## [16] assertthat_0.1   digest_0.6.9     formatR_1.3      evaluate_0.8.3   rmarkdown_0.9.5
## [21] labeling_0.3     stringi_1.0-1    scales_0.4.0     foreign_0.8-66
```