Alessandro Fois - 4115937

THESIS: Empirical approach to the problem of point pattern analysis in case of geocoded data

Logbook of the thesis activities

The thesis work started in **March**, even though I started reflecting on the topic months before. After the first meeting with my supervisors, it was decided to proceed with the data processing.
The data elaboration process ended in **May**. During the elaboration of the data I began to write down some arguments that constituted the initial structure of the thesis. The details of the work carried out are shown in the *List of commands and activities.*
During the month of **June**, until the 22nd I worked on the presentation of the results, on the 24th.
The writing process started in **July** and ended on **August** 15th.

List of commands and activities

Creating a database on PostgreSQL/PostGIS on my laptop

Making a PostgreSQL DB from the terminal

```
createdb thesis
```

Accessing the DB

```
psql thesis
```

Loading the PostGIS libraries and the *sfcgal* tools.

```
CREATE EXTENSION postgis;
CREATE EXTENSION postgis_topology;
CREATE EXTENSION postgis_sfcgal;
```

Set R working directory

Here is where the R file will be stored

```
setwd("/run/media/alessandro/095fcd10-81ae-4f36-b3f8-35d13831cf09/DATI_ReMA_SS/Thesis/thesis_texts")
```

R can log into the db (called *thesis*) and issue the commands.

```
library(rpostgis)

## Loading required package: RPostgreSQL

## Loading required package: DBI

drv <- dbDriver("PostgreSQL")
db <- dbConnect(drv, dbname = "thesis", host = "localhost", port = 5432, user
= "alessandro", password ="alessandro")
```

Read the PostGIS version

The option *connection=db* makes possible the interaction between the code written here and the PostGIS db. "db" is the name of the variable defined in R, so it can be anything.

```
SELECT PostGIS_full_version();
```

Load the raw data into the DB

The data is mainly composed by the population density map (raster), the list of addresses (points), the map of municipalities (polygons) in order to have boundaries. Additionally, there is a subdivision of the Netherlands in a self-made grid and later will be loaded with the data from the real cases.

1. Loading the population density raster (after a re-projection from WGS84 to 28992)

```
# load the population raster
raster2pgsql -s 28992 -I -t auto -C -M
/run/media/alessandro/095fcd10-81ae-4f36-b3f8-35d13831cf09/DATI_ReMA_SS/Thesi
s/raw_data/population_nld_2019-07-01_28992.tif -F  public.population_density
| psql -d thesis -U alessandro
#
# INSERT 0 1
# CREATE INDEX
# ANALYZE
# NOTICE:  Adding SRID constraint
# NOTICE:  Adding scale-X constraint
# NOTICE:  Adding scale-Y constraint
# NOTICE:  Adding blocksize-X constraint
# NOTICE:  Adding blocksize-Y constraint
# NOTICE:  Adding alignment constraint
# NOTICE:  Adding number of bands constraint
# NOTICE:  Adding pixel type constraint
# NOTICE:  Adding nodata value constraint
# NOTICE:  Adding out-of-database constraint
# NOTICE:  Adding maximum extent constraint
#  addrasterconstraints
# ----------------------
#  t
# (1 row)
#
# COMMIT
```

```
# VACUUM
#
```

2. Loading data on the database from PDOK WFS server

The idea was to retrieve the data using the WFS server from PDOK.

```
ogrinfo -so -ro
WFS:https://geodata.nationaalgeoregister.nl/bag/wfs/v1_1?service=WFS&version=
1.1.0&typeName=bag:verblijfsobject
```

INFO: Open of `WFS:https://geodata.nationaalgeoregister.nl/bag/wfs/v1_1?service=WFS' using driver`WFS' successful.

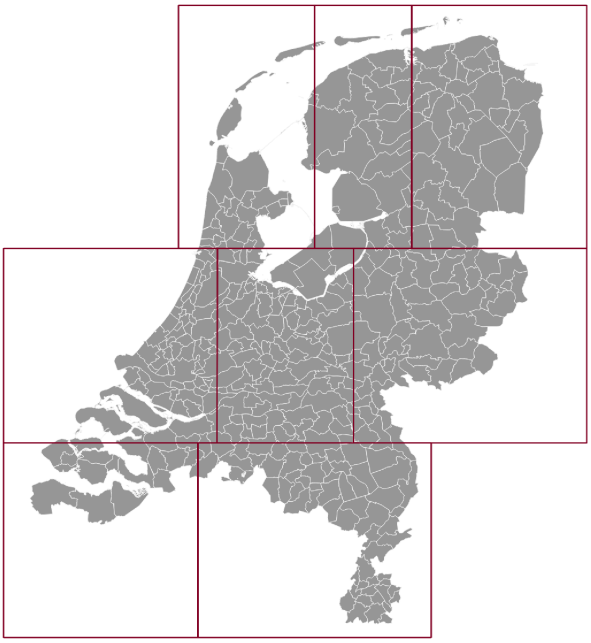
Metadata: ABSTRACT=Deze service wordt dagelijks geactualiseerd. De gegevens bestaan uit BAG-panden, een deelselectie van BAG-gegevens van deze panden en de verblijfsobjecten die zich hierin bevinden. De ligplaatsen en standplaatsen zijn hierin ook opgenomen met een deelselectie van BAG-gegevens. Nummeraanduidingen (en de bijhorende Nummeraanduidingidentificaties) worden in de WFS niet opgenomen als een losstaand object. De gegevens van de nummeraanduidingen zijn in deze services onderdeel van de adresseerbare objecten. Hierbij wordt alleen het hoofdadres opgenomen. Objecten met meerdere adressen (hoofd- en nevenadressen) zijn dus niet compleet. Dit betekent dat niet alle BAG adressen zijn opgenomen. Als u behoefte heeft aan de volledige BAG gegevens adviseren wij u gebruik te maken van andere BAG producten. De BAG API Individuele bevragingen https://www.kadaster.nl/zakelijk/producten/adressen-en-gebouwen/bag-api-individuele-bevragingen bevat alle BAG gegevens. Dit geldt ook voor de BAG viewer https://bagviewer.kadaster.nl/lvbag/bag-viewer/ en het BAG 2.0 Extract https://www.kadaster.nl/zakelijk/producten/adressen-en-gebouwen/bag-2.0-extract. Als u wilt nagaan welk BAG product het beste aansluit bij uw behoefte, raadpleeg dan de beslisboom voor BAG 2.0 producten: https://www.kadaster.nl/-/beslisboom-bag-2.0-producten.

PROVIDER_NAME=PDOK TITLE=BAG WFS 1: bag:ligplaats (title: ligplaats) 2: bag:pand (title: pand) 3: bag:standplaats (title: standplaats) 4: bag:verblijfsobject (title: verblijfsobject) 5: bag:woonplaats (title: woonplaats)

[1]- Done ogrinfo -so -ro WFS:https://geodata.nationaalgeoregister.nl/bag/wfs/v1_1?service=WFS

[2]+ Done version=1.1.0

The following lines should call the addresses included in the bound boxes visible in the figure below.



Grid

```
ogr2ogr -update -append -f PostgreSQL PG:"user=alessandro password=password
dbname=thesis"
"WFS:https://geodata.nationaalgeoregister.nl/bag/wfs/v1_1?service=WFS&version
=1.1.0&request=GetFeature&typeName=bag:verblijfsobject&srsName=EPSG:28992&bbo
x=100000,300000,220000,400000" -nln addressess
ogr2ogr -update -append -f PostgreSQL PG:"user=alessandro password=password
dbname=thesis"
"WFS:https://geodata.nationaalgeoregister.nl/bag/wfs/v1_1?service=WFS&version
=1.1.0&request=GetFeature&typeName=bag:verblijfsobject&srsName=EPSG:28992&bbo
x=110000,400000,180000,500000" -nln addressess
ogr2ogr -update -append -f PostgreSQL PG:"user=alessandro password=password
dbname=thesis"
"WFS:https://geodata.nationaalgeoregister.nl/bag/wfs/v1_1?service=WFS&version
=1.1.0&request=GetFeature&typeName=bag:verblijfsobject&srsName=EPSG:28992&bbo
x=     0,300000,100000,400000" -nln addressess
ogr2ogr -update -append -f PostgreSQL PG:"user=alessandro password=password
dbname=thesis"
"WFS:https://geodata.nationaalgeoregister.nl/bag/wfs/v1_1?service=WFS&version
=1.1.0&request=GetFeature&typeName=bag:verblijfsobject&srsName=EPSG:28992&bbo
x=     0,400000,110000,500000" -nln addressess
ogr2ogr -update -append -f PostgreSQL PG:"user=alessandro password=password
dbname=thesis"
"WFS:https://geodata.nationaalgeoregister.nl/bag/wfs/v1_1?service=WFS&version
=1.1.0&request=GetFeature&typeName=bag:verblijfsobject&srsName=EPSG:28992&bbo
x=160000,500000,210000,625000" -nln addressess
ogr2ogr -update -append -f PostgreSQL PG:"user=alessandro password=password
dbname=thesis"
"WFS:https://geodata.nationaalgeoregister.nl/bag/wfs/v1_1?service=WFS&version
=1.1.0&request=GetFeature&typeName=bag:verblijfsobject&srsName=EPSG:28992&bbo
```

```
x=210000,500000,300000,625000" -nln addressess
ogr2ogr -update -append -f PostgreSQL PG:"user=alessandro password=password
dbname=thesis"
"WFS:https://geodata.nationaalgeoregister.nl/bag/wfs/v1_1?service=WFS&version
=1.1.0&request=GetFeature&typeName=bag:verblijfsobject&srsName=EPSG:28992&bbo
x= 90000,500000,160000,625000" -nln addressess
ogr2ogr -update -append -f PostgreSQL PG:"user=alessandro password=password
dbname=thesis"
"WFS:https://geodata.nationaalgeoregister.nl/bag/wfs/v1_1?service=WFS&version
=1.1.0&request=GetFeature&typeName=bag:verblijfsobject&srsName=EPSG:28992&bbo
x=180000,400000,300000,500000" -nln addressess
```

Unfortunately, there is a limit of 1000 features downloadable per request, thus, one call cannot retrieve all the features into the boundaries. One solution was to create a loop whit $n/1000$ requests with n the number of features in the bounding box. However, I did not find a way to count the features in the layer from a WFS (Not even after having refreshed my memory with ogc/gdal and geoserver manuals). The best trial was with the following line, the result was again 1000.

https://geodata.nationaalgeoregister.nl/bag/wfs/v1_1?service=WFS&version=1.1.0&request=GetFeature&typeName=bag:verblijfsobject&srsName=EPSG:28992&bbox=100000,300000,220000,400000&resultType=hits

Therefore, the following attempt was just making a loop big enough to contain all the possible features. There are less than 17M inhabitants in the Netherlands and not everyone lives alone. 5000 per box should be sufficient to cover all the features.

```
for x in {1..5000}; do ogr2ogr -update -append -f PostgreSQL
PG:"user=alessandro password=password dbname=thesis"
"WFS:https://geodata.nationaalgeoregister.nl/bag/wfs/v1_1?service=WFS&version
=1.1.0&request=GetFeature&typeName=bag:verblijfsobject&srsName=EPSG:28992&bbo
x=100000,300000,220000,400000" -nln addresses; done
```

After many hours the result was disappointing. Contrary to the output obtained with the first tests made with a limited number of loops, the server started to reload the same features once they reached a certain number, although the total number was not covered yet.

Time to change my approach

The dataset can be downloaded directly from pdok. The page to download the data is here but it has some problems and the link opens the metadata instead of the download link that is this one. Hoping that it is the same since it is huge and my network is very slow. After half an hour the server close the connection, thus, *option 1*: find and install a download manager capable to restore the download from the interrupted point; *option 2*: download it with a modern network located in a server wisely placed at your sister's house and then conveniently download the file from that server taking all the time it takes..

```
curl
<https://geodata.nationaalgeoregister.nl/inspireadressen/extract/inspireadres
```

```
sen.zip> --output huge_file.zip

rsync -va --progress --rsh='ssh -p22'
alessandro@remote.address:/home/alessandro/Downloads/huge_file.zip .
```

loading the addresses in the postgis database turned out to be pretty tough. The data came in an unusual GML format written in xml standard. It is structured to contain multiple geometries together but is not so clear how to load it in the database. The website NLExtract explains how to do it in Dutch using the application. Even using google translate remains unclear. In the meantime the loop is downloading the points for box 1. This method is a dead truck.

Ultimately, loading the WFS (data accessed the 04 April 2021) on Qgis and from it copying the loaded table in a geopackage worked. Clearly, the number of features and the performance of the network connection deeply affected the time requested to load the dataset. After approximately 22 hours of work eventually the task was accomplished. However, it is better to have it on PostGIS, thus, the data must be copied there. Some features are multipoint while others are point, this is a problem. Again, Qgis worked better than the ogr2ogr program (but here I should have added the option for move from multipoint to point)

```
ogr2ogr -f PostgreSQL "PG:user=alessandro password=password dbname=thesis"
data.gpkg|layername=addresses
```

> Warning 1: Geometry to be inserted is of type Multi Point, whereas the layer geometry type is Point. Insertion is likely to fail ERROR 1: COPY statement failed. ERROR: Geometry type (MultiPoint) does not match column type (Point)

However, Qgis worked well and by using the tool from multi to single part it has been possible to convert and load the data into the postgis database in one step only.

Checking for duplicates

The an excerpt of the table *addresses*:

```
SELECT
id,oppervlakte,status,gebruiksdoel,openbare_ruimte,huisnummer,postcode,woonpl
aats,pandstatus FROM addresses LIMIT 5;
```

The table contain 9450967 feature,

```
--SELECT count(*) FROM addresses;
```

but only…

```
--SELECT COUNT(DISTINCT geom) FROM addresses;
```

8458574 are unique values. Thus, $9'450'967 - 8'458'574 = 992'393$ are **duplicated**. Some points have the same identification numbers.

```
SELECT fid,  identificatie, count(*) FROM addresses GROUP BY
fid,identificatie HAVING count(*)>1;
```

However, the identificatie and fid do not like to be unique, while pandidentificatie with the same geometry are in group of 2.

```
SELECT geom, count(*) AS duplicated FROM addresses GROUP BY geom HAVING
count(*)>1;
```

Moreover, the number of unique records grouped by geometry and grouped by pandidentificatie are discordant.

Before deleting the duplicate features it is wise making a DUMP of the database (the command will be executed once only).

```
pg_dump thesis | xz > thesis_db_backup_06apr2021.xz
```

NOTE: The following query is TOO SLOW!!!

```
DELETE FROM addresses a
                USING (
                      SELECT MIN(ctid) AS ctid, geom
                      FROM addresses
                      GROUP BY geom HAVING COUNT(*) > 1
                      ) b
       WHERE a.geom = b.geom  AND a.ctid <> b.ctid;
```

If the process of deleting worked than the number of features must be 8458574

```
SELECT count(*) FROM addresses;
```

R

Removing the duplicates remains a controversial point, some geometries are duplicated because the same location is referred to multiple addresses while others are distributed in a small surface forming a group of points where each one is one single address. Considering the quality of the source, keeping the duplicated geometries is perhaps the best choice.

The hypothesis is that the geocoded process generates a new space, different from the previous one which was continuous. Consequently, the test for complete spatial randomness must be adapted to the new space.

The ordinary test compares the given point pattern with a number of random point patterns, usually generated by the MonteCarlo method. For each random sample of points an average distance between the points is calculated and all average distances are naturally distributed. If the observed distribution of points has an average distance comparable with the distance distribution generated by the test, then the distribution is completely random and cannot contain clusters generated by autocorrelation or by correlation with another variable. However, the grid of points corresponding to the addresses has an average distance that cannot be compared with that of a continuous space. A commonly adopted

compensation to overcome this problem is the use of a population density map, so that more densely populated places are more likely to be counted than uninhabited ones. If the hypothesis is correct this countermeasure will prove insufficient.

The *first experiment* to verify this hypothesis consists in a simulation in which the expected outcome is a false negative. The Netherlands address list (provided by PDOK) represents all possible points available in the country space. The scale is constant, because the point grid is established, but the width of the observed area can change.

Step 1 Select an area of limited size, such as the city center. In this space, select a random sample of points. Perform a CSR test on the random sample, if the hypothesis is correct the test should consider the random distribution as non-random.

Step 2 Repeat the previous test using the population density map (retrieved from The Humanitarian data exchange) In the case of limited surfaces, the population density map should not significantly affect the outcome of the experiment.

Step 3 Apply the modified test on the same random sample. The selection of the points will therefore not be with the Monte Carlo method but will be randomly selected from the list of points present in the address layer.

Step 4 Manually select a sample by pretending to be random. Human selection, although done in such a way that it can replicate a random one, is unlikely to be. In this case, both tests should recognize the distribution as non-random.

Step 5 Repeat the test 99 times more. Step 6. Change the area and repeat the steps from 1 to 5.

Loading the required libraries

Some libraries are not used yet.

```
### load required libraries
library(spatstat)
library(ggmap)    #Loading required package: ggplot2
library(maptools) #Loading required package: sp
library(raster)
library(rasterVis)#Loading required package: lattice, latticeExtra
library(rgdal)
library(rpostgis) # Loading required package: RPostgreSQL, DBI
library(scales)
library(spatial)  # Functions for Kriging and Point Pattern Analysis
library(spdep)    # Loading required package: spData
library(splines)
library(viridis)  # Loading required package: viridisLite
library(tmap)
library(lubridate)
#
```

Load the data in R

Load the municipality of Groningen into the variable *test_area*. This will be the boundary. Load all the addresses of the city into the variable *test_ppa*, where ppa stands for Point Pattern Analysis. Also make a boundbox from the *test_area*

```
test_area   <- pgGetGeom(db, query="SELECT id,geom FROM municipalities_2018
WHERE gemeentena = 'Groningen'")
test_ppa    <- pgGetGeom(db, query="SELECT a.id,a.geom FROM addresses a JOIN
municipalities_2018 m ON ST_intersects(a.geom,m.geom) WHERE m.gemeentena =
'Groningen'")
test_bbox <- st_bbox(test_area)
```

Summary and print the loaded data

```
test_area
plot(test_area)
plot(test_ppa, pch=1, add=T)
```

Loading the raster from the DB turned out to be quite problematic. The following SQL code generates a VIEW of the *population_density* for the city of Groningen.

```
CREATE VIEW pd_selection AS
            SELECT r.rid,r.rast FROM population_density AS r
            JOIN municipalities_2018 AS m
            ON ST_Intersects(r.rast,m.geom)
            WHERE m.gemeentena = 'Groningen';
```

However, R refused to load it, thus, I made an attempt using the entire raster but limited by the bounding box, and also this is not working.

```
test_pop  <- pgGetRast(db, name='population_density', rast = "rast", bands=1,
boundary = c(226868.9, 243821.1, 577499.7, 587116.3))
```

Even if it was working, dropping the view after the use is a good practice!

```
DROP VIEW pd_selection;
```

To bypass the problem just move one moment out of R and make an excerpt of the population_density in Qgis and load it as tiff into R.

Print the loaded population density map.

```
plot(test_pop)
plot(test_area, add=T)
```

Preparing the sample

Let us go back to the geocoding issue.

Extract a random sample of 10'000 points from a list of 126'928

With this code the *sample* function extract 10'000 random id

```
test_ppa_temp <- test_ppa[ , "id", drop = FALSE]
test_random_sample_ppa <- test_ppa_temp[sample(nrow(test_ppa_temp), 10000), ]
rm(test_ppa_temp)
```

Plot the random sample and a summary.

```
summary(test_random_sample_ppa)
plot(test_pop)
plot(test_area, add=T)
plot(test_random_sample_ppa, add=T)
```

Prepare the geometries for being used by the *spatstat* library

```
test_area_owin <- as.owin(test_area) # make boundaries of the analyses
test_random_sample_ppp <- as.ppp(test_random_sample_ppa)         # make
test_sample_ppa a point pattern file

# spatial analyses focused only on patterns; is suggested remove all marks
(aka: attribute information, variables)
# in any case can be useful made a copy of the map without marks to obtain
better plots
marks(test_random_sample_ppp) <- NULL

# SET the Window of the points
Window(test_random_sample_ppp) <- test_area_owin

#for the following steps
test_pop_im <- as.im(as.integer(na.exclude(test_pop)))
Window(test_pop_im) <- test_area_owin # reduce the surface of test_pop to the
groningen extent
test_pop_im_l <- log(test_pop_im)
```

I'm not using a logarithmic distribution of the population, but the following is just to see if it follows the usual shape. The histogram of the logarithmic population_density raster is ruined by a large number of null values, though.

```
hist(test_pop_im_l)
```

Histogram of the population_density raster not log transformed. Most of the values score 7. This is not a big problem here because I need a pattern for the random generator. Clearly, a more detailed one would be better. NOTE. 7 here represent the number of inhabitants in a square of $24m$ per edge. Therefore the density is 7 inhabitants per $600m^2$

```
hist(test_pop)
```

Hypothesis tests

```
# Average nearest neighbor analysis
test_ann <- mean(nndist(test_random_sample_ppp, k=1)) # Approximate Nearest
Neighbor
test_ann   # The observed average nearest neighbor distance in meters
```

The average nearest neighbor of the sample is $22.14m$

```
ANN <- apply(nndist(test_random_sample_ppp, k=1:100),2,FUN=mean)
plot(ANN ~ eval(1:100), type="b", main=NULL, las=1)
## K and L functions
K <- Kest(test_random_sample_ppp)
plot(K, main=NULL, las=1, legendargs=list(cex=0.8, xpd=TRUE, inset=c(1.01, 0)
))


L <- Lest(test_random_sample_ppp, main=NULL)
plot(L, main=NULL, las=1, legendargs=list(cex=0.8, xpd=TRUE, inset=c(1.01, 0)
))
```

Test for clustering/dispersion

The code below is a Monte Carlo engine that shuffles the points, in this case of the sample, in order to test a number of possible alternative location configurations. Here it tests 999 combinations. In this first step I'm considering the whole surface of the municipality including impossible places like water, grassland, etc. This is not unusual because what is a realistic location depends on the case study, there are buildings, thus, the next step that takes account of the population density will improve the reliability of the test. The final vector will be a list of 999 average nearest neighbors (ANN) that for the central limit theorem and for the properties of the Poisson distribution will be normally distributed. If the ANN of the sample fall in the range we cannot reject the null hypothesis of complete spatial randomness (CSR)

```
# Distribution of expected ANN values given a homogeneous (CSR/IRP)
# point process using Monte Carlo methods. This is our null model.
n <- 999L                    # Number of simulations
# ------------------------------------------------------------------
test_NN_random   <- vector(length= n) # Create an empty object to be used to
store simulated ANN values
for (i in 1:n){
random_points   <- rpoint(n=test_random_sample_ppp$n, win=test_area_owin)  #
Generate random disposition of the location
test_NN_random[i]    <- mean(nndist(random_points, k=1))           # Tally
the ANN values
} # montecarlo
test_csr_1 <- test_NN_random
```

```
last_random_combination_csr_1 <- random_points
rm(random_points)
plot(last_random_combination_csr_1, pch=16, main=NULL, cols=rgb(0,0,0,0.5))
```

The histogram of the ANN combination is far from the sample's ANN. According to this classic test we can reject the null hypothesis of CSR for our sample. Although, we know it was random from the beginning.

```
# Print the histogram
hist(test_csr_1 , main=NULL, las=1, breaks=20, col="bisque",
xlim=range(test_ann, test_csr_1))
abline(v=test_ann, col="blue")
```

Because our simulation events are located in buildings, it is very important to consider the population density distribution. This computation will take a cup of tea-time.

```
## Monte Carlo taking account of the population density distribution.
# NOTE: 'f' must be either a function or an 'im' object. Because test_pop is
raster, we convert it ("on fly") as im.

n     <- 999L
test_NN_random <- vector(length=n)
for (i in 1:n){
  random_points   <- rpoint(n=test_random_sample_ppp$n, f=as.im(test_pop)) #
NOTE: test_pop is NOT logarithmic
  test_NN_random[i] <- mean(nndist(random_points, k=1))
}
test_csr_2 <- test_NN_random
last_random_combination_csr_2 <- random_points
rm(random_points)
```

Plotting the last combination of locations we can see that now it resembles the population distribution very closely. However, it is not enough to recognize the random sample as it is. Once again, we must reject the null hypothesis of CSR for the sample. And we know that this is a false negative.

```
plot(test_pop)
plot(test_area, add=T)
plot(last_random_combination_csr_2, pch=16, main=NULL, cols=rgb(0,0,0,0.5))

hist(test_csr_2 , main=NULL, las=1, breaks=20, col="bisque",
xlim=range(test_ann, test_csr_2))
abline(v=test_ann, col="blue")
```

My Monte Carlo. Testing the random sample considering the limits given by the lattice of points

Finally, we can consider the sample of points in its "natural environment", comparing not the random location but the random combinations between all the possible existing addresses. Clearly, it can be argued that a building can be realized everywhere in case it is

needed, it is up to the scholar to assess how realistic this is compared to the object of the study. These methods are frequently used in epidemiology, where the events are when someone has been infected and this happens in existing residential buildings.

NOTE. Although this method inherently accounts for the population distribution, it is not the same as having a precise census of the inhabitants of each address. Here, each address counts by one and there is no distinction between residential or business locations. However, the latter is not a problem here because the sample is selected without distinction of any sort.

```
test_ppa_temp <- test_ppa[ , "id", drop = FALSE]
n <- 999L                    # Number of simulations
# ------------------------------------------------------------------
test_NN_random   <- vector(length= n) # Create an empty object to be used to
store simulated ANN values
# montecarlo
for (i in 1:n){
random_points <- test_ppa_temp[sample(nrow(test_ppa_temp), 10000), ]
random_points <- as.ppp(random_points)
marks(random_points) <- NULL #clean from all non geometrical attribute
test_NN_random[i]    <- mean(nndist(random_points, k=1))            # Tally
the ANN values
} # end of montecarlo
test_csr_3 <- test_NN_random
last_random_combination_csr_3 <- random_points
rm(test_ppa_temp, random_points) # clean from temporary tables
```

Print the output

```
plot(test_pop)
plot(test_area, add=T)
plot(test_area)
plot(last_random_combination_csr_3, pch=16, main=NULL, cols=rgb(0,0,0,0.5),
add=T)
```

As can be seen from the histogram of the ANN, the sample is one of random realizations of the grid. Not surprisingly, it is also one of the most frequent. Therefore, we cannot reject the null hypothesis of CSR for the sample.

```
hist(test_csr_3 , main=NULL, las=1, breaks=40, col="bisque",
xlim=range(test_ann, test_csr_3))
abline(v=test_ann, col="blue")
```

Understanding the projection issue

During the loading process and annoying warning about the datum popup. This is due to the following proj issue. The old PROJ4 format is outdated and must be updated to the 6 or newer. The source is 6 but the software changed it to 4 and felt the need to bother me with its announcements. A detailed explanation can be seen here: "Migration to PROJ6/GDAL3"

```
getClass("CRS")
```

```
showSRID("EPSG:28992", "PROJ")
```

The projection in WKT2 version

```
cat(rgdal::showSRID("+init=epsg:28992", format="WKT2_2018", multiline="YES",
prefer_proj=FALSE), "\n")
```

Testing the case study

The case study is a research about clusterization of “sustainable” restaurants. The original paper performs an Average Nearest Neighbor (ANN) using ESRI and later grouping the points in two grids of squares with dimension defined according to *that field literature*, resulting in one grid of 18km edge and one 24km edge. The service used to geocode the points is not specified, it is possible is ESRI. Nevertheless, to properly apply the proposed method, the source of the points have to be known, thus, it can be any open-data service. Because the paper analyzed the restaurant in the Netherlands, the choice has fallen in PDOK.

The steps for the comparison are the following:

1. Replicate the same analysis using the same software and R. This is necessary to check the correspondence of the data and the verify that the closed source software is doing what we do.

1. The given database of points contains a group of outliers. This is due to the process of geocoding. The software stores these points piled all in the same location; these points should have been deleted. Therefore, in this step we will check how much their presence will affect the statistical test.

2. Moving from the analytical method to the empirical using Monte Carlo. Here, we calculate the ANN of the point pattern and compare the result with the distribution of 1000 random ANN sampled into the boundaries of the Netherlands considering the population distribution. This will be done after having removed the outliers.

3. The proposed Monte Carlo method. The point pattern will be geocoded using the open-data provided by PDOK and tested according to this distribution of points. Thus, each point distribution is a realization of the original points. A cluster or a dispersed distribution will be represented by an ANN located in one of the two extremes of the gaussian curve. (We should not see an ANN located far away from

the curve.)

4. Make a sub-selection of the addresses where a business activity can be located.

The theory

According to ESRI:

"The Average Nearest Neighbor is given as

$$ANN = \frac{\bar{D}_O}{\bar{D}_E}$$

Where $\bar{D}_O$ *is the observed mean distance between each feature and its nearest neighbor*

$$\bar{D}_O = \frac{\sum_{i=1}^{n} d_i}{n}$$

And $\bar{D}_E$ *is the expected mean distance for the features given in a random pattern:*

$$\bar{D}_E = \frac{0.5}{\sqrt{n/A}}$$

In the above equations, d_i *equals the distance between feature i and its nearest neighboring feature, n corresponds to the total number of features, and A is the area of a minimum enclosing rectangle around all features, or it is a user-specified area value.*

The average nearest neighbor z-score for the statistic is calculated as:

$$z = \frac{\bar{D}_O - \bar{D}_E}{SE}$$

where":

$$SE = \frac{0.26136}{\sqrt{n^2/A}}$$

Prepare the datasets to be joined

- Make all missing values NULL

```
UPDATE paper_restaurants_list SET toev= NULL WHERE toev=' ';
UPDATE paper_restaurants_list SET huisletter= NULL WHERE huisletter=' ';
```

- Make all the letters upper case

```
UPDATE paper_restaurants_list SET toev= UPPER (toev);
UPDATE paper_restaurants_list SET huisletter= UPPER (huisletter);
UPDATE addresses_cleaned      SET toevoeging= UPPER (toevoeging);
UPDATE addresses_cleaned      SET huisletter= UPPER (huisletter);
```

- Create a view that joins the tables, eventually it can be copied in a table to make the whole process faster.

```
CREATE TABLE paper_restaurants AS
SELECT a.id,
       r.id rid,
       a.geom,
       r.naam,
       a.openbare_ruimte AS street,
       r.straat          AS r_street,
       a.huisnummer      AS house_number,
       a.huisletter      AS house_letter,
       a.toevoeging,
       a.postcode
FROM addresses_cleaned AS a
JOIN paper_restaurants_list AS r
ON
(QUOTE_NULLABLE(a.huisnummer)||QUOTE_NULLABLE(a.huisletter)||QUOTE_NULLABLE(a
.toevoeging)||QUOTE_NULLABLE(a.postcode))=    (QUOTE_NULLABLE(r.huisnr)
||QUOTE_NULLABLE(r.huisletter)||QUOTE_NULLABLE(r.toev)
||QUOTE_NULLABLE(r.postcode));
```

Now the fastest way to remove the duplicated is to load the layer in a gis software and use a "remove duplicated" tool. Here is done with QGIS. After that the table can be deleted and the new one loaded. This can be done from QGIS too.

```
CREATE OR REPLACE VIEW paper_my_hub_geocoding AS
SELECT a.id,
    a.geom,
    a.openbare_ruimte AS street,
    a.huisnummer AS house_number,
    a.huisletter AS house_letter,
    a.toevoeging,
    a.postcode,
    h.hub_distance AS distance
FROM addresses_cleaned a
JOIN paper_adjusted_original_db h
  ON a.id::text = h.hub_name::text;
```

Average Nearest Neighbour analysis with different softwares

The differences between QGIS and ArcMap could be due to the projection or due to a different implementation of the program code.

QGIS ANN analysis:	original_dataset
Observed Mean Distance:	173.8018835601438 Meters
Expected Mean Distance:	1045.9206545682018 Meters

Nearest Neighbor Ratio:	0.16617119358054577
z-score:	-314.26121765427735
p-value:	0.000000
Point count:	38812
Study Area:	

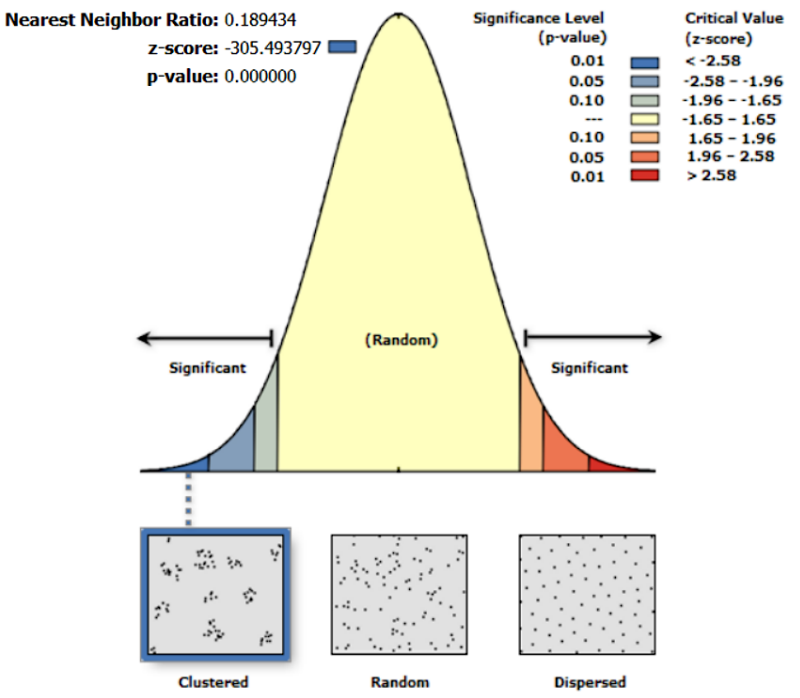
ArcMap 10.5 ANN analysis:	original_dataset
Observed Mean Distance:	173.7942 Meters
Expected Mean Distance:	917.4406 Meters
Nearest Neighbor Ratio:	0.189434
z-score:	-305.493797
p-value:	0.000000
Point count:	38812
Study Area:	130671818770.172928

ArcMap 10.5 ANN analysis:	adjusted_original_dataset
Observed Mean Distance:	178.6267 Meters
Expected Mean Distance:	643.8703 Meters
Nearest Neighbor Ratio:	0.277427
z-score:	-268.621258
p-value:	0.000000
Study Area:	62619814533.049225

paper_restaurant represents the LISA list of restaurants geocoded on the PDOK addresses. Therefore, this list is missing the secondary set of restaurants added from different sources.

QGIS ANN analysis:	paper_restaurant
Observed Mean Distance:	196.38296969206087 Meters
Expected Mean Distance:	848.5928777888188 Meters
Nearest Neighbor Ratio:	0.23142189244361397
z-score:	-245.529803917235
p-value:	0.000000
Point count:	27885
Study Area:	

Average Nearest Neighbor Summary



Given the z-score of -305.493797232, there is a less than 1% likelihood that this clustered pattern could be the result of random chance.

Average Nearest Neighbor Summary

Observed Mean Distance:	173.7942 Meters
Expected Mean Distance:	917.4406 Meters
Nearest Neighbor Ratio:	0.189434
z-score:	-305.493797
p-value:	0.000000

Dataset Information

Input Feature Class:	lisa_restaurants
Distance Method:	EUCLIDEAN
Study Area:	130671818770.172928
Selection Set:	False

From ArcMap 10.5.1

Following this strategy, the number of geocoded features are:

Alternatively, the existing geocoded dataset of points can be adjusted to fit the given addresses list. From the original dataset there are 1050 points located all in the same place, this is an error (blunder), here I deleted them.

Test the ANN of the original dataset with R

Load the data

```
paper_original     <- pgGetGeom(db, query="SELECT id,geom FROM
paper_original_db")
paper_adj_original <- pgGetGeom(db, query="SELECT id,geom FROM
paper_adjusted_original_db")
#
paper_my_hub_geocoding  <- pgGetGeom(db, query="SELECT id,geom FROM
paper_my_hub_geocoding")
paper_my_pdok_geocoding <- pgGetGeom(db, query="SELECT id,geom FROM
paper_my_pdok_geocoding")
#
paper_bbox      <- st_bbox(paper_original)
paper_adj_bbox  <- st_bbox(paper_adj_original)
paper_hub_bbox  <- st_bbox(paper_my_hub_geocoding)
paper_pdok_bbox <- st_bbox(paper_my_pdok_geocoding)
```

Loading the raster of population, the Netherlands boundaries and the list of addresses. The list contains duplicates, the table *addresses_cleaned*, despite the name and my initial enthusiasm, is the same as the original. However, it came out that not all the duplicated geometries have to be deleted or removed, because the PDOK does not constantly follow the same procedure. Sometimes the points representing buildings are stacked with different address letters, other times are grouped in dense clusters. The duplicates have NOT to be removed!

Summary of the loaded data

```
# Average nearest neighbor analysis
Do_ann <- mean(nndist(paper_original_ppp, k=1))    # Average Nearest Neighbor
A <- area(as.rectangle(Window(paper_original_ppp)))
A_ESRI <- 130671818770.172928 # ESRI calculates the smallest area, the
smallest rectangle is called envelope (tool: minimum bounding geometry).
n <- npoints.ppp(paper_original_ppp)
De_ann <- 0.5/sqrt(n/A)
DE <- 0.5/sqrt(n/A_ESRI)
z <- (Do_ann - De_ann)/(0.26136/sqrt(n^2/A))
z_ESRI <- (Do_ann - DE)/(0.26136/sqrt(n^2/A_ESRI))
#
Do_ann
De_ann
DE
z
z_ESRI
```

Both ESRI and QGIS are the same, as expected. So now it can be applied to the adjusted point patterns.

```
# Average nearest neighbor analysis
Do_ann <- mean(nndist(paper_adj_original_ppp, k=1))    # Average Nearest
```

```
Neighbor
A <- area(as.rectangle(Window(paper_adj_original_ppp)))
n <- npoints(paper_adj_original_ppp)
De_ann <- 0.5/sqrt(n/A)
z <- (Do_ann - De_ann)/(0.26136/sqrt(n^2/A))
# Results:
n
Do_ann
De_ann
z
```

As expected the z-score is always far away from the range of $\pm$ 3 standard deviations from the mean.

Test the ANN with the proposed method

Before moving to Monte Carlo to calculate the expected value the ANN can be calculated also for the new dataset. This has to be done because the Monte Carlo will be calculated on the list of addresses given by PDOK, thus, also the list of events have to be geocoded with this dataset.

```
# Average nearest neighbor analysis
Do_ann <- mean(nndist(paper_my_hub_geocoding_ppp, k=1))    # Average Nearest
Neighbor
A <- area(as.rectangle(Window(paper_my_hub_geocoding_ppp)))
n <- npoints(paper_my_hub_geocoding_ppp)
De_ann <- 0.5/sqrt(n/A)
z <- (Do_ann - De_ann)/(0.26136/sqrt(n^2/A))
# Results:
n
Do_ann
De_ann
z
```

Let us try with Monte Carlo

First the original dataset

```
netherlands_owin <- as.owin(netherlands) # make boundaries of the analyses
Window(paper_original_ppp) <- netherlands_owin # SET the Window of the points

plot(population_density)
plot(netherlands_owin, add=T)
```

Monte Carlo

```
n     <- 999L
test_NN_random <- vector(length=n)
for (i in 1:n){
  random_points   <- rpoint(n=paper_original_ppp$n,
f=(population_density_im))
  test_NN_random[i] <- mean(nndist(random_points, k=1))
```

```
}
De_MC            <- test_NN_random # Expected Value
last_realization <- random_points
rm(random_points)
```

Plotting the last combination of locations we can see that now it resembles the population distribution very closely. However, it is not enough to recognize the random sample as it is. Once again, we must reject the null hypothesis of CSR for the sample. And we know that this is a false negative.

```
plot(population_density)
plot(netherlands_owin, add=T)
plot(last_realization, pch=16, main=NULL, cols=rgb(0,0,0,0.5))

# Print the histogram
Do     <- mean(nndist(paper_original_ppp, k=1))     # Original Average
Nearest Neighbor
Do_adj <- mean(nndist(paper_adj_original_ppp, k=1)) # Adjusted Average
Nearest Neighbor
hist(De_MC , main=NULL, las=1, breaks=20, col="bisque", xlim=range(Do_adj,
De_MC))
abline(v=Do, col="blue")
abline(v=Do_adj, col="red")
```

Although the expected value was lower compared to the analytic, ~ 350 the Monte Carlo and 917. 4406 the D_EThe difference remains highly significant and we must reject the null hypothesis. In order to compare the pattern of events (restaurants) with the existing list of addresses only the given database can be used. Therefore, the test will not be performed on the original list of addresses because it is geocoded with an unknown provider and has to be adapted to the available one, the PDOK.

Second the adjusted dataset with the proposed method

This Monte Carlo compares the pattern of events with 1000 possible realizations of the same number of events. However, this time the events can fall only in one of the existing locations given by the list of addresses.

```
# set the new window for the "paper_my_hub_geocoding_ppp"
Window(paper_my_hub_geocoding_ppp) <- netherlands_owin
```

Monte Carlo

```
n <- 999L                                          # Number of simulations
nevents <- npoints(paper_my_hub_geocoding_ppp)     # number of events
test_ppa_temp <- pdok_addresses[ , "id", drop = FALSE]   # list of id to
select the existing addresses

# ------------------------------------------------------------------
test_NN_random   <- vector(length= n) # Create an empty object to be used to
store simulated ANN values
```

```
# montecarlo
for (i in 1:n){
random_points <- test_ppa_temp[sample(nrow(test_ppa_temp), nevents), ]
random_points <- as.ppp(random_points)
marks(random_points) <- NULL #clean from all non geometrical attribute
test_NN_random[i]    <- mean(nndist(random_points, k=1))            # Tally
the ANN values
} # end of monte carlo
MC                   <- test_NN_random # Expected Value
last_realization_mc <- random_points
rm(test_ppa_temp, random_points) # clean from temporary tables

#plot(population_density)
plot(netherlands_owin)
plot(last_realization_mc, pch=1, main=NULL, cols=rgb(0.5,0.2,0.6,0.2), add=T)
plot(paper_my_hub_geocoding_ppp, pch=3, main=NULL, cols=rgb(0,0,0,0.1),
add=T)

# Print the histogram
Do     <- mean(nndist(paper_original_ppp, k=1))         # Original Average
Nearest Neighbor
Do_mc  <- mean(nndist(paper_my_hub_geocoding_ppp, k=1)) # Average Nearest
Neighbor
hist(MC , main=NULL, las=1, breaks=20, col="bisque", xlim=range(Do, MC))
abline(v=Do, col="blue")
abline(v=Do_mc, col="red")
```

Although the ANN is once again closer to the expected value, the result is significant and we must reject the null hypothesis of complete spatial randomness for the pattern of events. One last improvement we could insert is the subsection of addresses. Depending on the field of application and the phenomenon observed, the scholar must argue if the space selected is an appropriate environment to test the pattern of events. In this case, we could guess the restaurants cannot be located in every possible building, apartment or room, thus, we could consider reducing the list of addresses only to the places that can potentially host a restaurant. Once again, here it is not discussed the validity of the case study but the proposed method. Reducing the list of addresses only to the business activities cannot be a wise method in the field of the given paper used as example. Even if the results have been strengthened so far, this does not mean that the validity of the research is improved or decreased, this can be done only if the results are argued with a valid literature and interpreted in the proper context.

NOTE: the list of addresses still contain duplicate geometries. However, not all the geometries have the same address, sometimes it differs by letter, others not.

Third the adjusted dataset with the proposed method

```
pdok_addresses_firms <- pgGetGeom(db, query="SELECT id,geom FROM addresses
WHERE (gebruiksdoel LIKE 'industriefunctie%') OR (gebruiksdoel LIKE
'winkelfunctie%') OR (gebruiksdoel LIKE 'bijeenkomstfunctie%')")
```

```
# set the new window for the "paper_my_hub_geocoding_ppp"
Window(paper_my_hub_geocoding_ppp) <- netherlands_owin
```

Monte Carlo

```
n <- 1000L                                              # Number of simulations
nevents <- npoints(paper_my_hub_geocoding_ppp)     # number of events
test_ppa_temp <- pdok_addresses_firms[ , "id", drop = FALSE]   # list of id
to select the existing addresses

# ------------------------------------------------------------------
test_NN_random   <- vector(length= n) # Create an empty object to be used to
store simulated ANN values
# montecarlo
for (i in 1:n){
random_points <- test_ppa_temp[sample(nrow(test_ppa_temp), nevents), ]
random_points <- as.ppp(random_points)
marks(random_points) <- NULL #clean from all non geometrical attribute
test_NN_random[i]    <- mean(nndist(random_points, k=1))            # Tally
the ANN values
} # end of monte carlo
MC                  <- test_NN_random # Expected Value
last_realization_mc <- random_points
rm(test_ppa_temp, random_points) # clean from temporary tables

#plot(population_density)
plot(netherlands_owin)
plot(last_realization_mc, pch=1, main=NULL, cols=rgb(0.5,0.2,0.6,0.2), add=T)
plot(paper_my_hub_geocoding_ppp, pch=3, main=NULL, cols=rgb(0,0,0,0.1),
add=T)

# Print the histogram
Do     <- mean(nndist(paper_original_ppp, k=1))         # Original Average
Nearest Neighbor
Do_mc  <- mean(nndist(paper_my_hub_geocoding_ppp, k=1)) # Average Nearest
Neighbor
X <-unique(paper_my_hub_geocoding_ppp,rule="deldir")  # remove duplicate
geometries
Do_X  <- mean(nndist(X, k=1))
hist(MC , main=NULL, las=1, breaks=20, col="bisque", xlim=range(Do, MC))
abline(v=Do, col="blue")
abline(v=Do_mc, col="red")
abline(v=Do_X, col="black")
npoints(paper_my_hub_geocoding_ppp)
npoints(X)
```

Further analysis

To further confirm the validity of the result compared to the previous methods, we can compare a random realization with the same characteristics of the pattern of events. Consequently, the number of points and the surface of the random selection, will be the

same as the random pattern. Despite both methods resulting in the same attribution, this test should give strength to the proposed method.

Step 1. Select a random pattern 2. Test it with ordinary ANN 3. Test it with ordinary Monte Carlo 4. Test it with the proposed Monte Carlo 5. Compare the results

Generate a random pattern

This code will generate a random pattern of the same size of the restaurant pattern

```
nevents <- npoints(paper_my_hub_geocoding_ppp)     # number of events
test_ppa_temp <- pdok_addresses[ , "id", drop = FALSE]   # list of id to
which select the existing addresses
random_sample_ppa <- test_ppa_temp[sample(nrow(test_ppa_temp), nevents), ]
#random_sample_ppa <- test_ppa_temp[sample(nrow(test_ppa_temp), 1000), ] #
use this to manually set the sample number
rm(test_ppa_temp)
```

Make it as a ppp object and clean…

```
random_sample_ppa <- as.ppp(random_sample_ppa) # make test_sample_ppa a point
pattern file
marks(random_sample_ppa) <- NULL               # clean
Window(random_sample_ppa) <- netherlands_owin  # SET the Window of the points
```

Test the ordinary ANN for the random pattern

```
# Average nearest neighbor analysis
Do_rand <- mean(nndist(random_sample_ppa, k=1))    # Average Nearest Neighbor
A <- area(as.rectangle(Window(random_sample_ppa)))
n <- npoints(random_sample_ppa)
De_rand <- 0.5/sqrt(n/A)
z <- (Do_rand - De_rand)/(0.26136/sqrt(n^2/A))
# Results:
n
Do_rand
De_rand
z
```

As expected, the ordinary ANN test is not able to recognize a true random pattern.

Test it with ordinary Monte Carlo

Because it is an ordinary Monte Carlo, the random sample will be compared with other random realizations, considering the population density and the Netherlands' boundaries.

In order to make a proper comparison the random sample used will be the same generated in the previous step *random_sample_ppa*. The window is the same as well.

Monte Carlo

```
n     <- 999L
test_NN_random <- vector(length=n)
```

```
for (i in 1:n){
  random_points     <- rpoint(n=random_sample_ppa$n,
f=as.im(population_density)) # NOTE: test_pop is NOT logarithmic
  test_NN_random[i] <- mean(nndist(random_points, k=1))
}
De_randMC             <- test_NN_random # Expected Value
last_realization_rand <- random_points
rm(random_points)
```

Usual plots…

```
plot(population_density)
plot(netherlands_owin, add=T)
plot(last_realization_rand, pch=16, main=NULL, cols=rgb(0,0,0,0.5))

# Print the histogram
Do      <- mean(nndist(paper_original_ppp, k=1))     # Original Average
Nearest Neighbor
Do_adj  <- mean(nndist(paper_adj_original_ppp, k=1)) # Adjusted Average
Nearest Neighbor
Do_rand <- mean(nndist(random_sample_ppa, k=1)) # Adjusted Average Nearest
Neighbor
hist(De_randMC , main=NULL, las=1, breaks=20, col="bisque", xlim=range(Do,
De_randMC))
abline(v=Do, col="blue")
abline(v=Do_adj, col="red")
abline(v=Do_rand, col="purple")
summary(pdok_addresses)
```

Speed test!

```
library(foreach)
library(doParallel)
#
parallel::detectCores()
n.cores <- parallel::detectCores() - 1
n.cores

#create the cluster
my.cluster <- parallel::makeCluster(
  n.cores,
  type = "FORK"
  )

#check cluster definition (optional)
print(my.cluster)

#register it to be used by %dopar%
doParallel::registerDoParallel(cl = my.cluster)

#check if it is registered (optional)
foreach::getDoParRegistered()
```

```
#how many workers are available? (optional)
foreach::getDoParWorkers()

x <- foreach(
  i = 1:10,
  .combine = 'c'
) %dopar% {
    sqrt(i)
  }
x

n     <- 6L
test_NN_random <- vector(length=n)
x <- foreach(i = 1:n
        ) %dopar% {
  random_points     <- rpoint(random_sample_ppa$n, f=(population_density_im))
  test_NN_random[i] <- mean(nndist(random_points, k=1))
}
test_De             <- test_NN_random # Expected Value
x

parallel::stopCluster(cl = my.cluster)
```

Usual plots…

```
contour(population_density_im, axes=TRUE)
plot(netherlands_owin, add=T)
hist(population_density_im)

plot(netherlands_owin)
plot(test_last_random, pch=16, main=NULL, cols=rgb(0.3,0.1,0.4,0.5), add=T)
test_last_random
```