

Applied Spatial Data Analysis - Spatial Point and Lattice Data

Dr Sebnem Er

2023-02-21

Contents

1	Introduction	5
2	Spatial Point Pattern Analysis	7
2.1	Prerequisites	7
2.2	Datasets - Readily Available, Imported, Simulated Datasets	7
2.3	Plotting Datasets	34
2.4	Quadrat Analysis - Quadrat Counts and Tests	38
2.5	Kernel Density Smoothing	44
2.6	Kernel Smoothing with a Covariate	47
2.7	Distance Measures and Tests	53
2.8	G Function	68
2.9	F Function	70
2.10	Ripley's K Function	73
2.11	References:	76
3	Spatial Lattice Data Analysis	77
3.1	Prerequisites	77
3.2	Columbus Dataset	77
3.3	Obtain the Centroids	80
3.4	Obtain the Neighbourhood Relationship	80
3.5	Neighbours to Weights Matrix	87
3.6	Moran's I	87
3.7	Local Moran's I	91

3.8 Week 2 of Lattice	92
3.9 OLS	92
3.10 Moran's I Test of Residual	93

Chapter 1

Introduction

This book will guide you through the R codes for Spatial Point and Lattice Data Analysis.

The chapters will be made available on Tuesdays when we start a new week So please update your browser to access the codes for the relevant chapter.

Chapter 2

Spatial Point Pattern Analysis

2.1 Prerequisites

You need to have the following R packages installed and recalled into your library:

```
suppressPackageStartupMessages({  
  library(sf)  
  library(spatstat)  
  library(spatstat.data)  
  library(ggplot2)  
  library(sp)  
  library(animation)  
  library(plotrix)  
  library(tmap)})
```

2.2 Datasets - Readily Available, Imported, Simulated Datasets

2.2.1 Simple Feature Basics

```
library(sf)
```

`sf` geometry types:

- point: `st_point()`
- linestring: `st_linestring()`
- polygon: `st_polygon()`
- multipoint: `st_multipoint()`
- multilinestring: `st_multilinestring()`
- multipolygon: `st_multipolygon()`
- collection of different geometry types: `st_geometrycollection()`

2.2.1.1 sfg

- A single point and multipoint

```
# Using the sf library for simple features

# Tarih      Saat      Enlem(N)   Boylam(E) Derinlik(km) MD   ML   Mw   Yer
# -----      -----      -----       -----      -----   --  --  --  --
# 2023.02.15 14:46:46 36.8803    36.6177     8.3      --  2.8  --  ASAGIBILENLER
# 2023.02.15 14:43:05 37.0218    28.8915     8.4      --  2.7  --  OTMANLAR-KOYCA

# sf::st_crs(4326), which means x and y positions are interpreted as longitude (E) and
point1_sg = c(36.6177, 36.8803) %>% st_point() # this will create a "sfg" class Geomet
plot(point1_sg)
```

o

2.2. DATASETS - READILY AVAILABLE, IMPORTED, SIMULATED DATASETS9

```
class(point1_sg)

## [1] "XY"     "POINT"  "sfg"

point2_sg = c(28.8915, 37.0218) %>% st_point()
plot(point2_sg)
```

○

```
class(point2_sg)

## [1] "XY"     "POINT"  "sfg"

points_sg = rbind(c(36.6177, 36.8803),c(28.8915, 37.0218)) %>% st_multipoint()
plot(points_sg)
```

○ ○

```
class(points_sg)  
  
## [1] "XY"           "MULTIPOINT" "sfg"  
  
st_crs(points_sg)
```

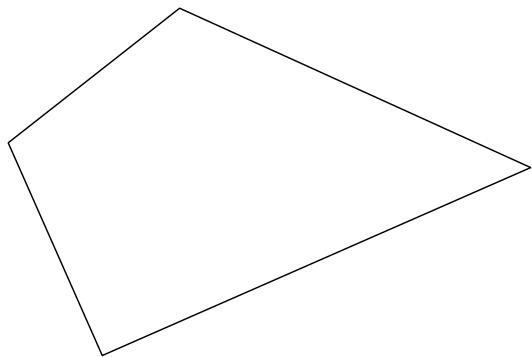
Coordinate Reference System: NA

- A single polygon and a multipolygon

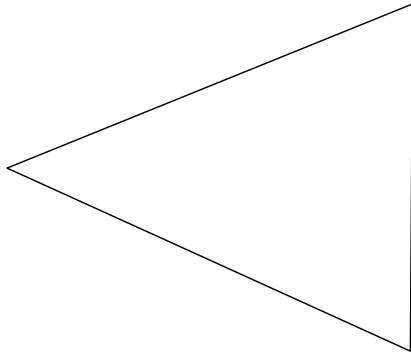
```
# polygon  
poly1 = list(rbind(c(35.8337398, 37.8864154),  
                  c(36.3281246, 36.7682256),  
                  c(38.5803218, 37.7562395),  
                  c(36.7346187, 38.5939762),  
                  c(35.8337398, 37.8864154)))  
  
poly1_sg = poly1 %>% st_polygon()  
  
poly2 = list(rbind(c(36.7346187, 38.5939762),  
                  c(38.5913081, 39.3456266),  
                  c(38.5803218, 37.7562395),  
                  c(36.7346187, 38.5939762)))
```

2.2. DATASETS - READILY AVAILABLE, IMPORTED, SIMULATED DATASETS11

```
poly2_sg = poly2 %>% st_polygon()  
plot(poly1_sg)
```



```
plot(poly2_sg)
```



```
# multipolygon
polygons <- list(poly1, poly2)
polygons_sg <- polygons %>% st_multipolygon()

st_crs(polygons_sg) # we see that the sf geometry has no CRS set up.

## Coordinate Reference System: NA
```

2.2.1.2 sfc

- We usually work with multiple simple features and want to combine them. We have a multipolygon and points and want to combine these. We will use `st_sfc()` to merge

```
points_sfc <- st_sfc(point1_sg, point2_sg) # with no CRS
st_crs(points_sfc)

## Coordinate Reference System: NA

points_sfc <- st_sfc(point1_sg, point2_sg, crs = 4326) # with appropriate CRS
st_crs(points_sfc)
```

2.2. DATASETS - READILY AVAILABLE, IMPORTED, SIMULATED DATASETS13

```
## Coordinate Reference System:  
##   User input: EPSG:4326  
##   wkt:  
## GEOCRS["WGS 84",  
##        DATUM["World Geodetic System 1984",  
##               ELLIPSOID["WGS 84",6378137,298.257223563,  
##                           LENGTHUNIT["metre",1]]],  
##        PRIMEM["Greenwich",0,  
##                  ANGLEUNIT["degree",0.0174532925199433]],  
##        CS[ellipsoidal,2],  
##           AXIS["geodetic latitude (Lat)",north,  
##                  ORDER[1],  
##                  ANGLEUNIT["degree",0.0174532925199433]],  
##           AXIS["geodetic longitude (Lon)",east,  
##                  ORDER[2],  
##                  ANGLEUNIT["degree",0.0174532925199433]],  
##        USAGE[  
##               SCOPE["Horizontal component of 3D system."],  
##               AREA["World."],  
##               BBOX[-90,-180,90,180]],  
##               ID["EPSG",4326]]  
  
multipolygon_sfc <- st_sfc(poly1_sg, poly2_sg, crs = 4326)  
st_crs(multipolygon_sfc)
```

```
## Coordinate Reference System:  
##   User input: EPSG:4326  
##   wkt:  
## GEOCRS["WGS 84",  
##        DATUM["World Geodetic System 1984",  
##               ELLIPSOID["WGS 84",6378137,298.257223563,  
##                           LENGTHUNIT["metre",1]]],  
##        PRIMEM["Greenwich",0,  
##                  ANGLEUNIT["degree",0.0174532925199433]],  
##        CS[ellipsoidal,2],  
##           AXIS["geodetic latitude (Lat)",north,  
##                  ORDER[1],  
##                  ANGLEUNIT["degree",0.0174532925199433]],  
##           AXIS["geodetic longitude (Lon)",east,  
##                  ORDER[2],  
##                  ANGLEUNIT["degree",0.0174532925199433]],  
##        USAGE[  
##               SCOPE["Horizontal component of 3D system."],  
##               AREA["World."],  
##               BBOX[-90,-180,90,180]],  
##               ID["EPSG",4326]]
```

```
##     ID["EPSG",4326]]
```

2.2.1.3 sf

We usually have a data frame that stores the attributes of points, polygons, lines. Remember the earthquake magnitude, time, depth for the points; elevation, population, size for the polygons.

For points

```
magnitude = c(2.8, 2.7)
depth = c(8.3, 8.4)
time = c(1446, 1443)

earthquake_marks = data.frame(magnitude, depth, time)

earthquake_sf <- earthquake_marks %>% st_sf(geometry = points_sf) # sf object

plot1 = ggplot() +
  geom_sf(data = earthquake_sf[["magnitude"]], color = "black") +
  coord_sf()
plot1
```



2.2. DATASETS - READILY AVAILABLE, IMPORTED, SIMULATED DATASETS15

2.2.2 External files (csv, shp etc)

We will use two layers:

- Location of earthquakes - earthquakes
- Boundaries of Turkey as a polygon - turkeyshp

One of the files is a shp file, the other one is just a simple csv file.

- Location of earthquakes - earthquakes (from a csv)

Read in the dataset as usual

```
earthquake_csv <- read.csv("C:/Users/01438475/OneDrive - University of Cape Town/ASDA/DataSets/Ea
head(earthquake_csv)

##      Tarih      Saat     Lat     Long Derinlik_km   MD   ML   Mw
## 1 2023.02.07 13:44:51 38.2452 37.9093      13.9 -- 3.5 --
## 2 2023.02.07 13:41:44 38.1057 36.5072      5.0 -- 3.9 --
## 3 2023.02.07 13:37:24 36.4372 35.6998     13.4 -- 4.0 --
## 4 2023.02.07 13:32:27 37.2445 36.9277      5.4 -- 2.5 --
## 5 2023.02.07 13:30:10 38.1428 37.8203      2.0 -- 2.9 --
## 6 2023.02.07 13:27:30 38.3478 38.1357      6.5 -- 4.7 --
##                               Yer
## 1          OREN-AKCADAG (MALATYA)
## 2 MAHMUTBEY-GOKSUN (KAHRAMANMARAS)
## 3    ISKENDERUN KORFEZI (AKDENIZ)
## 4      NAIMLER-NURDAGI (GAZIANTEP)
## 5    FINDIKKOVY-DOGANSEHIR (MALATYA)
## 6      KUSDOGAN-YESILYURT (MALATYA)

earthquake_sf <- earthquake_csv %>%
  st_as_sf(coords = c("Long", "Lat"), crs = 4326) %>%      # convert to sf file %>%
  #Here Long comes first.
  st_transform("+proj=utm +zone=37 +datum=WGS84 +units=m +no_defs")

st_crs(earthquake_sf)

## Coordinate Reference System:
##   User input: +proj=utm +zone=37 +datum=WGS84 +units=m +no_defs
##   wkt:
##   PROJCRS["unknown",
##           BASEGEOGCRS["unknown",
```

```

##           DATUM["World Geodetic System 1984",
##                     ELLIPSOID["WGS 84",6378137,298.257223563,
##                               LENGTHUNIT["metre",1]],
##                     ID["EPSG",6326]],
##                     PRIMEM["Greenwich",0,
##                               ANGLEUNIT["degree",0.0174532925199433],
##                               ID["EPSG",8901]]],
##                     CONVERSION["UTM zone 37N",
##                               METHOD["Transverse Mercator",
##                                         ID["EPSG",9807]],
##                               PARAMETER["Latitude of natural origin",0,
##                                         ANGLEUNIT["degree",0.0174532925199433],
##                                         ID["EPSG",8801]],
##                               PARAMETER["Longitude of natural origin",39,
##                                         ANGLEUNIT["degree",0.0174532925199433],
##                                         ID["EPSG",8802]],
##                               PARAMETER["Scale factor at natural origin",0.9996,
##                                         SCALEUNIT["unity",1],
##                                         ID["EPSG",8805]],
##                               PARAMETER["False easting",500000,
##                                         LENGTHUNIT["metre",1],
##                                         ID["EPSG",8806]],
##                               PARAMETER["False northing",0,
##                                         LENGTHUNIT["metre",1],
##                                         ID["EPSG",8807]],
##                                         ID["EPSG",16037]],
##                     CS[Cartesian,2],
##                     AXIS["(E)",east,
##                           ORDER[1],
##                           LENGTHUNIT["metre",1,
##                                         ID["EPSG",9001]]],
##                     AXIS["(N)",north,
##                           ORDER[2],
##                           LENGTHUNIT["metre",1,
##                                         ID["EPSG",9001]]]]

```

- Boundaries of Turkey as a polygon - turkeyshp

```

library(sf)
turkeyshp = "C:/Users/01438475/OneDrive - University of Cape Town/ASDA/DataSets/turkey"
st_read() %>%
  st_transform("+proj=utm +zone=37 +datum=WGS84 +units=m +no_defs")

## Reading layer `tur_polbnda_adm0' from data source
##   `C:\Users\01438475\OneDrive - University of Cape Town\ASDA\DataSetsturkey_adminis

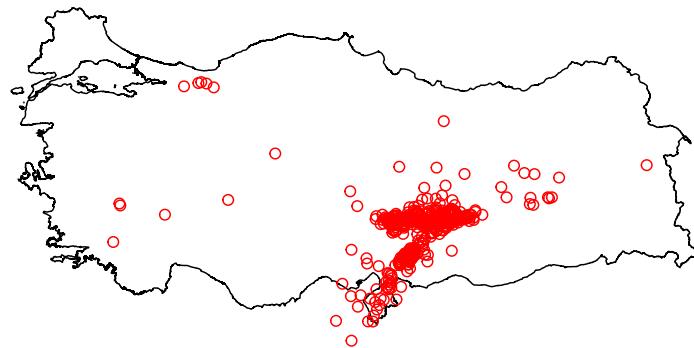
```

2.2. DATASETS - READILY AVAILABLE, IMPORTED, SIMULATED DATASETS17

```
##  using driver `ESRI Shapefile'  
## Simple feature collection with 1 feature and 5 fields  
## Geometry type: MULTIPOLYGON  
## Dimension: XY  
## Bounding box: xmin: 25.66851 ymin: 35.80842 xmax: 44.81793 ymax: 42.10479  
## Geodetic CRS: WGS 84
```

Plotting the two together in a simple plot:

```
plot(st_geometry(turkeyshp))  
plot(st_geometry(earthquake_sf), add = TRUE, col = "red")
```



or using `geom_sf()` in ggplot:

```
plot1 = ggplot() +  
  geom_sf(data = turkeyshp) +  
  geom_sf(data = earthquake_sf) +  
  theme_minimal()
```

Here you see that earthquakes outside of Turkey are also plotted. We can subset only the earthquakes that happened in Turkey:

```
earthquake_sf = earthquake_sf[turkeyshp,]
plot2 = ggplot() +
  geom_sf(data = turkeyshp) +
  geom_sf(data = earthquake_sf) +
  theme_minimal()
```

Visualisations can be done via `tmap` package as well, and can be saved as png, html files.

```
tmap_mode("view")
```

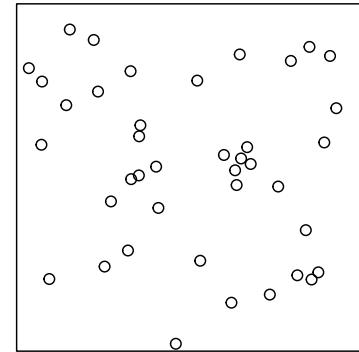
```
## tmap mode set to interactive viewing
```

```
tmap1 = tm_shape(turkeyshp)+tm_polygons() +tm_shape(earthquake_sf) + tm_dots()
tmap_save(tmap1, filename= "turkeyearthquakes.html")
```

```
## Interactive map saved to C:\Users\01438475\OneDrive - University of Cape Town\ASDA\
```

2.2.3 Simulated Datasets

```
set.seed(135)
xy_csr <- matrix(runif(80), ncol=2)
pp_csr <- as.ppp(xy_csr, c(0,1,0,1))
plot(pp_csr)
```

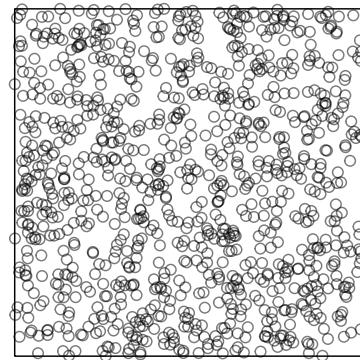
pp_csr

2.2.3.0.1 CSR Data Points

```
set.seed(135)
xy_csr <- matrix(runif(2000), ncol=2)
mark_numerical = rexp(1000)
mark_categorical = sample(0:1, size=1000, replace=TRUE)

# plot without marks
pp_csr_m <- as.ppp(xy_csr, c(0,1,0,1))
plot(pp_csr_m)
```

pp_csr_m



2.2.3.0.2 CSR Data Points with Marks

```
# plot with marks
xy_csr_withmark = as.data.frame(cbind(xy_csr,mark_numerical, mark_categorical))

# The categorical variable needs to be set as factor with clearly defined levels

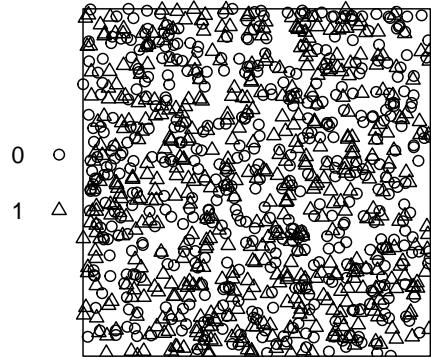
xy_csr_withmark$mark_categorical = factor(xy_csr_withmark$mark_categorical, levels = c

# set as point pattern

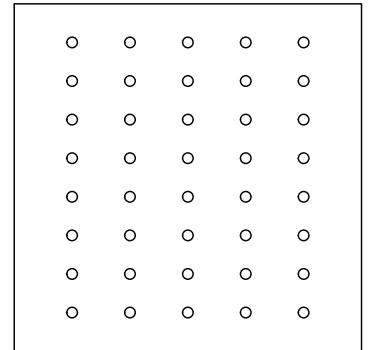
pp_csr_withmark = with(xy_csr_withmark, ppp(V1,V2,c(0,1),c(0,1),marks=xy_csr_withmark[)
plot(pp_csr_withmark)
```

2.2. DATASETS - READILY AVAILABLE, IMPORTED, SIMULATED DATASETS21

pp_csr_withmark

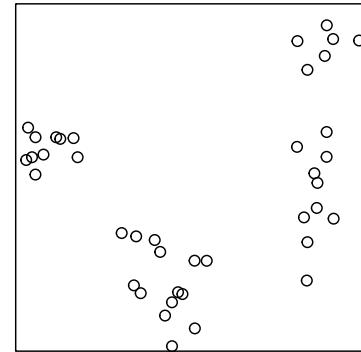


```
regular <- read.csv("C:/Users/01438475/OneDrive - University of Cape Town/ASDA/regular.csv")
xy_regular <- matrix(cbind(regular$X,regular$Y), ncol=2)
pp_regular <- as.ppp(xy_regular, c(0,1,0,1))
plot(pp_regular)
```

pp_regular

2.2.3.0.3 Regular Data Points

```
cluster <- read.csv("C:/Users/01438475/OneDrive - University of Cape Town/ASDA/cluster  
xy_cluster <- matrix(cbind(cluster$X,cluster$Y), ncol=2)  
pp_cluster <- as.ppp(xy_cluster, c(0,1,0,1))  
plot(pp_cluster)
```

pp_cluster

2.2.3.0.4 Cluster Data Points

2.2.4 Swedishpines Dataset from spatstat.data library

```

data(swedishpines)
swp = rescale.ppp(swedishpines)
swp

## Planar point pattern: 71 points
## window: rectangle = [0, 9.6] x [0, 10] metres

class(swp)

## [1] "ppp"

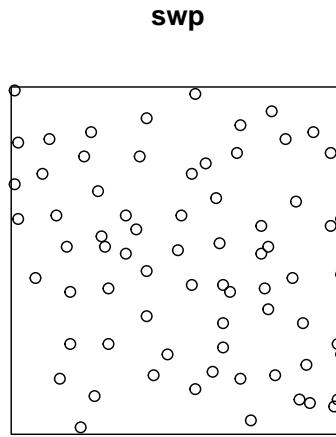
summary(swp)

## Planar point pattern: 71 points
## Average intensity 0.7395833 points per square metre
##
## Coordinates are given to 1 decimal place
## i.e. rounded to the nearest multiple of 0.1 metres
##

```

```
## Window: rectangle = [0, 9.6] x [0, 10] metres
## Window area = 96 square metres
## Unit of length: 1 metre
```

```
plot(swp)
```



2.2.5 External datasets - Turkey

Let us go back to Earthquake data that we converted to sf object.

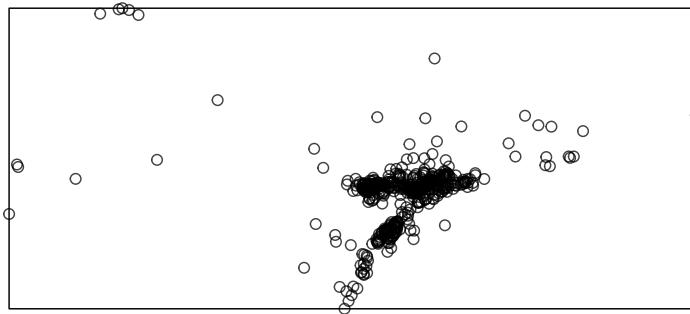
Convert the sf objects to ppp point pattern object using function `as.ppp` so that we can analyse them in R with spatstat.

```
# only the geometry
earthquake_ppp = as.ppp(st_geometry(earthquake_sf))
turkeyshp_owin = as.owin(st_geometry(turkeyshp))

plot(earthquake_ppp)
```

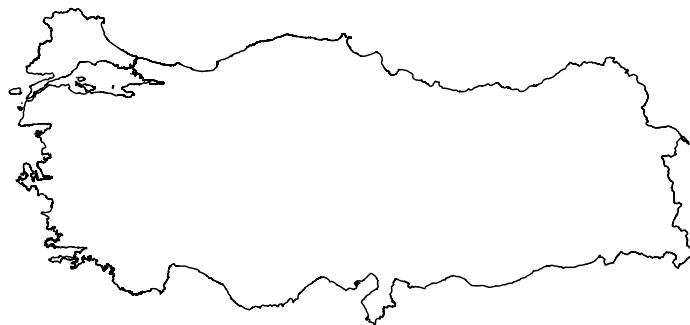
2.2. DATASETS - READILY AVAILABLE, IMPORTED, SIMULATED DATASETS25

earthquake_ppp



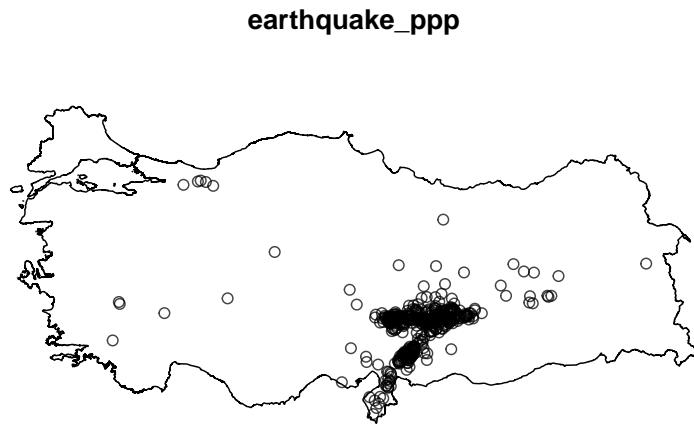
```
plot(turkeyshp_owin)
```

turkeyshp_owin



The observation window and the point pattern can be combined, so that the custom window replaces the default rectangular window:

```
earthquake_ppp = earthquake_ppp[turkeyshp_owin]  
plot(earthquake_ppp)
```

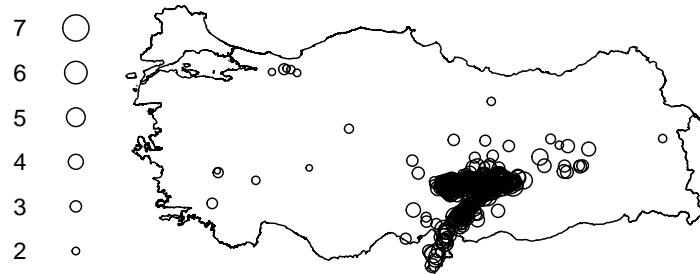


We might want to incorporate some marks to these points:

```
# if there are marks that need to be included:  
  
earthquake_ppp.marked_numeric <- ppp(earthquake_ppp$x, earthquake_ppp$y, window = turkeyshp_owin)  
  
plot(earthquake_ppp.marked_numeric, use.marks=TRUE)
```

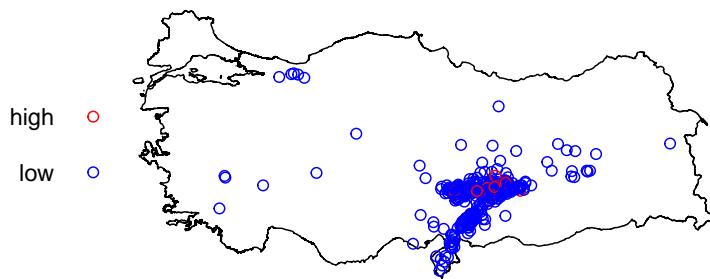
2.2. DATASETS - READILY AVAILABLE, IMPORTED, SIMULATED DATASETS27

earthquake_ppp.marked_numeric



```
earthquake_ppp.marked_categorical <- ppp(earthquake_ppp$x, earthquake_ppp$y, window = turkeyshp_c  
plot(earthquake_ppp.marked_categorical, use.marks=TRUE,  
     cols=c("red","blue","pink"),  
     markscale=0.1,  
     pch=21, cex=1)
```

earthquake_ppp.marked_categorical



<https://rspatial.org/raster/roku/Chapter5.html> <https://geobgu.xyz/r/point-pattern-analysis.html> <https://www.keene.edu/campus/maps/tool/> <https://data.humdata.org/dataset/cod-ab-tur?> <https://www.jla-data.net/eng/merging-geometry-of-sf-objects-in-r/>

2.2.5.0.1 External Datasets - Clinics Dataset SA Download the data from the following:

<https://web1.capetown.gov.za/web1/OpenDataPortal/DatasetDetail?DatasetName=Clinics>

```

library(sf)

RSA_roads = "C:/Users/01438475/OneDrive - University of Cape Town/ASDA/DataSets/zafrds8ff5a/Z
st_read() %>% st_transform(4326)

## Reading layer `ZAF_roads' from data source
## `C:\Users\01438475\OneDrive - University of Cape Town\ASDA\DataSet
## using driver `ESRI Shapefile'
## Simple feature collection with 5559 features and 5 fields
## Geometry type: MULTILINESTRING
## Dimension: XY
## Bounding box: xmin: 16.48784 ymin: -34.82747 xmax: 32.85267 ymax: -22.17693
## Geodetic CRS: WGS 84

```

2.2. DATASETS - READILY AVAILABLE, IMPORTED, SIMULATED DATASETS29

```
RSA_biome = "C:/Users/01438475/OneDrive - University of Cape Town/ASDA/DataSets/rsabiome4xkhi/RSA.biome.shp"
st_read() %>% st_transform(4326)

## Reading layer `RSA_biome` from data source
##   `C:/Users/01438475/OneDrive - University of Cape Town\ASDA\DataSets\rsabiome4xkhi\RSA.biome.shp`
##   using driver `ESRI Shapefile'
## Simple feature collection with 3127 features and 1 field
## Geometry type: POLYGON
## Dimension:      XY
## Bounding box:  xmin: 16.45296 ymin: -34.8336 xmax: 32.8928 ymax: -22.12583
## Geodetic CRS:  Hartebeesthoek94

clinics_sf = "C:/Users/01438475/OneDrive - University of Cape Town/ASDA/DataSets/Clinics/SL_CLNC.shp"
st_read() %>% st_transform(4326)

## Reading layer `SL_CLNC` from data source
##   `C:/Users/01438475/OneDrive - University of Cape Town\ASDA\DataSets\Clinics\SL_CLNC.shp`
##   using driver `ESRI Shapefile'
## Simple feature collection with 149 features and 5 fields
## Geometry type: POINT
## Dimension:      XY
## Bounding box:  xmin: 18.34268 ymin: -34.19491 xmax: 18.90847 ymax: -33.51262
## Geodetic CRS:  WGS 84

ct.wards_sf <- "C:/Users/01438475/OneDrive - University of Cape Town/ASDA/DataSets/sa/CPT/electoralwards.shp"
st_read(quiet = TRUE) %>%
  st_set_crs(4326)

st_crs(ct.wards_sf)

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

```

##           AXIS["geodetic longitude (Lon)",east,
##                  ORDER[2],
##                  ANGLEUNIT["degree",0.0174532925199433]],
##      USAGE[
##          SCOPE["Horizontal component of 3D system."],
##          AREA["World."],
##          BBOX[-90,-180,90,180]],
##          ID["EPSG",4326]]


st_crs(clinics_sf)

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


class(clinics_sf)

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

summary(clinics_sf)

##      LCTN          ATHY          NAME          CLASS
##  Length:149    Length:149    Length:149    Length:149
##  Class :character  Class :character  Class :character  Class :character
##  Mode  :character  Mode  :character  Mode  :character  Mode  :character

```

2.2. DATASETS - READILY AVAILABLE, IMPORTED, SIMULATED DATASETS31

```
##      RGN                  geometry
##  Length:149      POINT    :149
##  Class :character  epsg:4326   : 0
##  Mode   :character +proj=long...: 0
```

2.2.6 Swedishpines Dataset from spatstat.data library

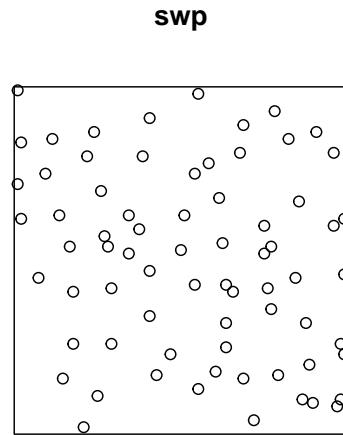
```
data(swedishpines)
swp = rescale.ppp(swedishpines)
class(swp)

## [1] "ppp"

summary(swp)

## Planar point pattern: 71 points
## Average intensity 0.7395833 points per square metre
##
## Coordinates are given to 1 decimal place
## i.e. rounded to the nearest multiple of 0.1 metres
##
## Window: rectangle = [0, 9.6] x [0, 10] metres
## Window area = 96 square metres
## Unit of length: 1 metre
```

```
plot(swp)
```



2.2.7 Clinics Dataset Using Simple Features (SF)

Download the data from the following:

<https://web1.capetown.gov.za/web1/OpenDataPortal/DatasetDetail?DatasetName=Clinics>

Extract the data frame into R:

```
library(sf)
clinics_sf = st_read("C:/Users/01438475/OneDrive - University of Cape Town/ASDA/DataSets/Clinics/SL_CLNC.shp")

## Reading layer `SL_CLNC` from data source
##   `C:/Users/01438475/OneDrive - University of Cape Town/ASDA/DataSets/Clinics/SL_CLNC.shp`
##   using driver `ESRI Shapefile'
## Simple feature collection with 149 features and 5 fields
## Geometry type: POINT
## Dimension:      XY
## Bounding box:  xmin: 18.34268 ymin: -34.19491 xmax: 18.90847 ymax: -33.51262
## Geodetic CRS:  WGS 84

clinics_sf

## Simple feature collection with 149 features and 5 fields
```

2.2. DATASETS - READILY AVAILABLE, IMPORTED, SIMULATED DATASETS33

```
## Geometry type: POINT
## Dimension: XY
## Bounding box: xmin: 18.34268 ymin: -34.19491 xmax: 18.90847 ymax: -33.51262
## Geodetic CRS: WGS 84
## First 10 features:
##                                LCTN          ATHY
## 1      C/O Adam/ Liedeman Street Mamre    PAWC
## 2      Cnr Hermes & GrosvenorAve CITY OF CAPE TOWN
## 3      Hassen Kahn Ave Rusthof Strand    PAWC
## 4      61 Central Circle, Fish Hoek CITY OF CAPE TOWN
## 5      Simon Street, Nomzamo CITY OF CAPE TOWN
## 6      C/O Musical and Hospital Street Macassar    PAWC
## 7      28 Church Street Somerset West CITY OF CAPE TOWN
## 8      Fagan Street Strand CITY OF CAPE TOWN
## 9      Karbonkel Road, CMC Building, Hout Bay    PAWC
## 10     Midmar Street Groenvallei CITY OF CAPE TOWN
##                                NAME          CLASS        RGN
## 1      MAMRE CDC Community Day Centre  Western
## 2      SAXON SEA CLINIC             Clinic  Western
## 3      GUSTROUW CDC Community Day Centre Eastern
## 4      FISH HOEK CLINIC            Clinic Southern
## 5      IKWEZI CDC Community Day Centre Eastern
## 6      MACASSAR CDC Community Day Centre Eastern
## 7      SOMERSET WEST CLINIC          Clinic  Eastern
## 8      FAGAN STREET SATELLITE       Satellite Eastern
## 9      HOUT BAY HARBOUR CDC Community Day Centre Southern
## 10     GROENVALLEI SATELLITE        Satellite Tygerberg
##                                geometry
## 1  POINT (18.47692 -33.51262)
## 2  POINT (18.48881 -33.55012)
## 3  POINT (18.85211 -34.13472)
## 4  POINT (18.42632 -34.13669)
## 5  POINT (18.86622 -34.11375)
## 6  POINT (18.76369 -34.06105)
## 7  POINT (18.84814 -34.08579)
## 8  POINT (18.82979 -34.1162)
## 9  POINT (18.34268 -34.0549)
## 10 POINT (18.66701 -33.89165)

class(clinics_sf)

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

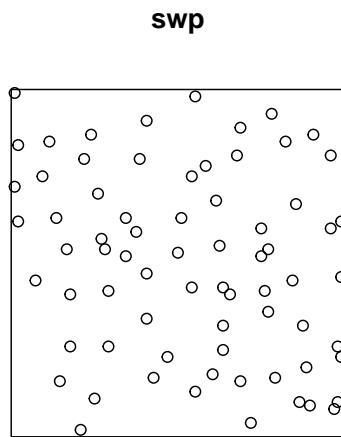
```
summary(clinics_sf)
```

```
##      LCTN          ATHY          NAME          CLASS 
##  Length:149    Length:149    Length:149    Length:149 
##  Class :character  Class :character  Class :character  Class :character 
##  Mode  :character  Mode  :character  Mode  :character  Mode  :character 
##           RGN          geometry    
##  Length:149    POINT       :149 
##  Class :character  epsg:4326   : 0 
##  Mode  :character  +proj=long...: 0
```

2.3 Plotting Datasets

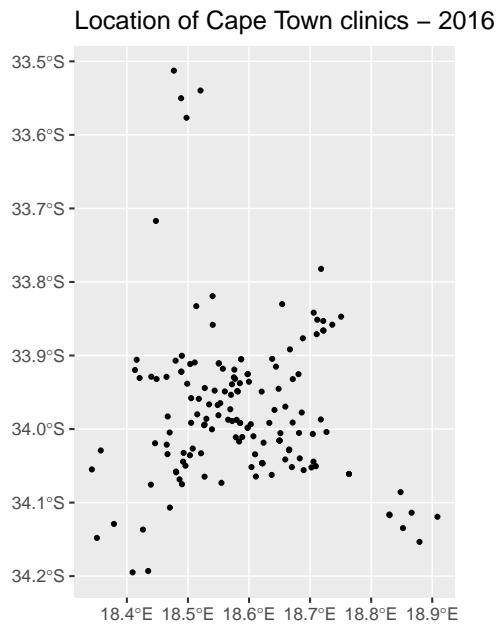
2.3.1 Basic plot() function

```
plot(swp)
```



2.3.2 Basic ggplot() function - (sf) object

```
library(ggplot2)
plot1 = ggplot() +
  geom_sf(data = clinics_sf, size = .8, color = "black") +
  ggtitle("Location of Cape Town clinics – 2016") +
  # not specifying crs here, coord_sf will use the CRS defined in the first layer = "+proj=longla"
  coord_sf()
plot1
```

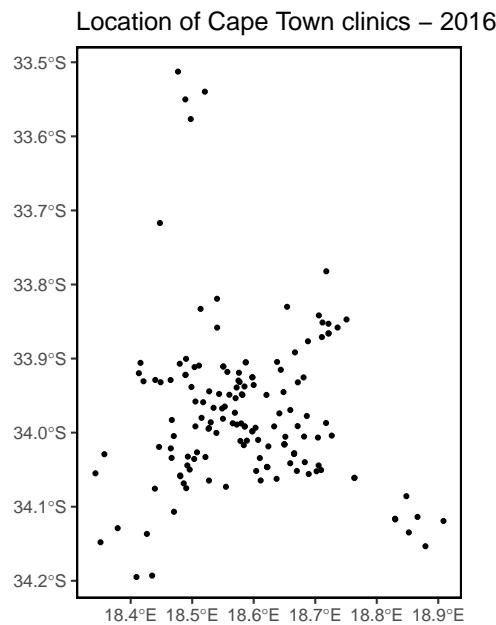


2.3.3 ggplot() function with a bounding box - (sf) object

```
library(ggplot2)
plot2 = ggplot() +
  geom_sf(data = clinics_sf, size = .8, color = "black") +
  ggtitle("Location of Cape Town clinics – 2016") +
  coord_sf(xlim = c(18.34, 18.91), ylim = c(-34.19, -33.51262)) +
  theme(panel.grid.major = element_blank(),
        panel.grid.minor = element_blank(),
        panel.background = element_rect(colour = "black", size=1, fill=NA))
```

```
## Warning: The `size` argument of `element_rect()` is deprecated as of ggplot2 3.4.0.
## i Please use the `linewidth` argument instead.
```

```
plot2
```



2.3.4 ggplot() with Electoral Wards Shape File

In order to plot using the electoral wards polygons, we need the sf data frame to be converted into Spatial Points Data Frame (sp).

```
clinics_sp <- as(clinics_sf, Class = "Spatial")
class(clinics_sp)
```

```
## [1] "SpatialPointsDataFrame"
## attr(,"package")
## [1] "sp"
```

Download the CPT electoral wards and import the shape file as follows:

```
library(sf)
ct.wards_sf = st_read("C:/Users/01438475/OneDrive - University of Cape Town/ASDA/DataS
```

```

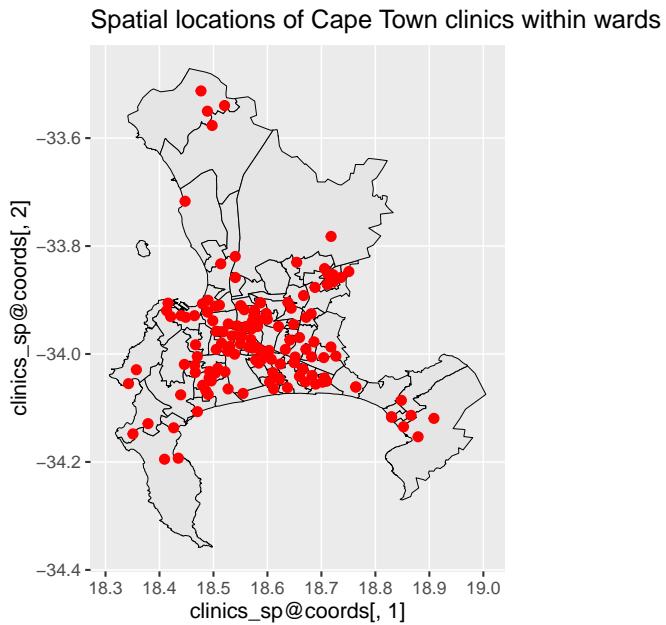
## Reading layer `electoral wards for cpt' from data source
##   `C:\Users\01438475\OneDrive - University of Cape Town\ASDA\DataSets\sa\CPT\electoral wards f
##   using driver `ESRI Shapefile'
## Simple feature collection with 111 features and 9 fields
## Geometry type: MULTIPOLYGON
## Dimension:      XY
## Bounding box:  xmin: 18.30722 ymin: -34.35834 xmax: 19.00467 ymax: -33.47128
## CRS:           NA

st_geometry_type(ct.wards_sf)

## [1] MULTIPOLYGON MULTIPOLYGON MULTIPOLYGON MULTIPOLYGON MULTIPOLYGON
## [6] MULTIPOLYGON MULTIPOLYGON MULTIPOLYGON MULTIPOLYGON MULTIPOLYGON
## [11] MULTIPOLYGON MULTIPOLYGON MULTIPOLYGON MULTIPOLYGON MULTIPOLYGON
## [16] MULTIPOLYGON MULTIPOLYGON MULTIPOLYGON MULTIPOLYGON MULTIPOLYGON
## [21] MULTIPOLYGON MULTIPOLYGON MULTIPOLYGON MULTIPOLYGON MULTIPOLYGON
## [26] MULTIPOLYGON MULTIPOLYGON MULTIPOLYGON MULTIPOLYGON MULTIPOLYGON
## [31] MULTIPOLYGON MULTIPOLYGON MULTIPOLYGON MULTIPOLYGON MULTIPOLYGON
## [36] MULTIPOLYGON MULTIPOLYGON MULTIPOLYGON MULTIPOLYGON MULTIPOLYGON
## [41] MULTIPOLYGON MULTIPOLYGON MULTIPOLYGON MULTIPOLYGON MULTIPOLYGON
## [46] MULTIPOLYGON MULTIPOLYGON MULTIPOLYGON MULTIPOLYGON MULTIPOLYGON
## [51] MULTIPOLYGON MULTIPOLYGON MULTIPOLYGON MULTIPOLYGON MULTIPOLYGON
## [56] MULTIPOLYGON MULTIPOLYGON MULTIPOLYGON MULTIPOLYGON MULTIPOLYGON
## [61] MULTIPOLYGON MULTIPOLYGON MULTIPOLYGON MULTIPOLYGON MULTIPOLYGON
## [66] MULTIPOLYGON MULTIPOLYGON MULTIPOLYGON MULTIPOLYGON MULTIPOLYGON
## [71] MULTIPOLYGON MULTIPOLYGON MULTIPOLYGON MULTIPOLYGON MULTIPOLYGON
## [76] MULTIPOLYGON MULTIPOLYGON MULTIPOLYGON MULTIPOLYGON MULTIPOLYGON
## [81] MULTIPOLYGON MULTIPOLYGON MULTIPOLYGON MULTIPOLYGON MULTIPOLYGON
## [86] MULTIPOLYGON MULTIPOLYGON MULTIPOLYGON MULTIPOLYGON MULTIPOLYGON
## [91] MULTIPOLYGON MULTIPOLYGON MULTIPOLYGON MULTIPOLYGON MULTIPOLYGON
## [96] MULTIPOLYGON MULTIPOLYGON MULTIPOLYGON MULTIPOLYGON MULTIPOLYGON
## [101] MULTIPOLYGON MULTIPOLYGON MULTIPOLYGON MULTIPOLYGON MULTIPOLYGON
## [106] MULTIPOLYGON MULTIPOLYGON MULTIPOLYGON MULTIPOLYGON MULTIPOLYGON
## [111] MULTIPOLYGON
## 18 Levels: GEOMETRY POINT LINESTRING POLYGON MULTIPOINT ... TRIANGLE

library(ggplot2)
ggplot() +
  geom_sf(data = ct.wards_sf, size = .5, color = "black") +
  geom_point(aes(x = clinics_sp@coords[,1], y = clinics_sp@coords[,2]),
             data = clinics_sp@data, alpha = 1, size=2, color = "red")+
  ggtitle("Spatial locations of Cape Town clinics within wards") +
  coord_sf()

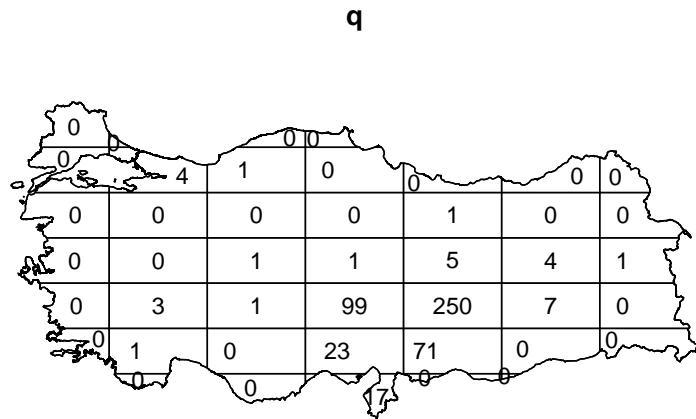
```



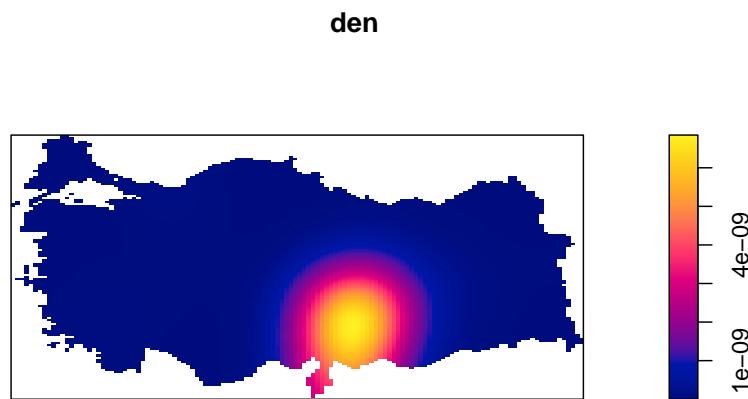
2.4 Quadrat Analysis - Quadrat Counts and Tests

2.4.1 Turkey Earthquake

```
q = quadratcount(earthquake_ppp, nx = 7, ny = 7)
plot(q)
```



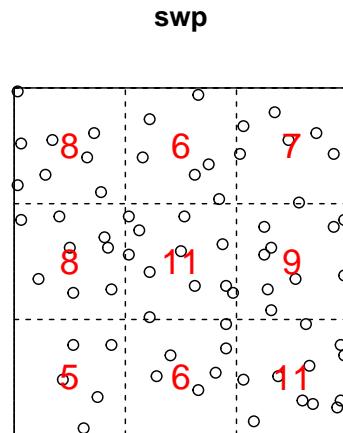
```
den = density(earthquake_ppp)  
plot(den)
```



2.4.2 swp dataset

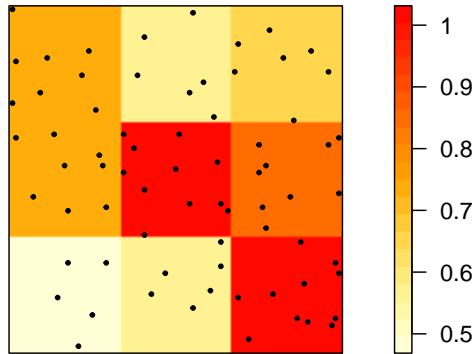
Quadrat counts:

```
Q3x3 = quadratcount(swp, nx=3, ny=3)
plot(swp)
plot(Q3x3, add=TRUE, col="red", cex=1.5, lty=2)
```



```
# Plot the density
cl <- interp.colours(c("lightyellow", "orange", "red"), 20)

plot( intensity(Q3x3, image=TRUE), las=1, col=cl, main=NULL)
plot(swp, pch=20, cex=0.6, col="black", add=TRUE) # Add points
```



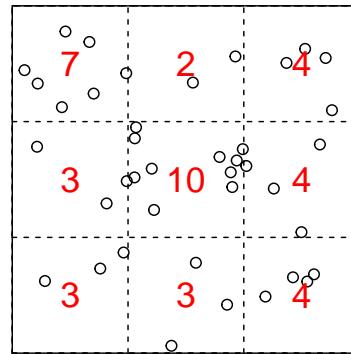
Quadrat test

```
Q3x3test = quadrat.test(swp, 3,3)
Q3x3test
```

```
##
## Chi-squared test of CSR using quadrat counts
##
## data: swp
## X2 = 4.6761, df = 8, p-value = 0.4169
## alternative hypothesis: two.sided
##
## Quadrats: 3 by 3 grid of tiles
```

2.4.3 Simulated CSR Pattern

```
Q3x3_csr = quadratcount(pp_csr, nx=3, ny=3)
plot(pp_csr)
plot(Q3x3_csr, add=TRUE, col="red", cex=1.5, lty=2)
```

pp_csr

Test:

```
Q3x3test_csr = quadrat.test(pp_csr, 3,3)

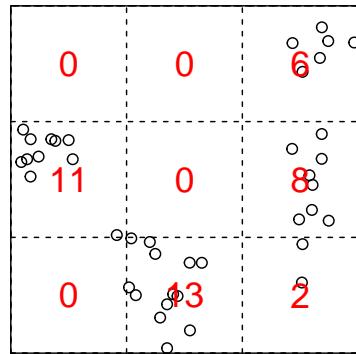
## Warning: Some expected counts are small; chi^2 approximation may be inaccurate

Q3x3test_csr

##
## Chi-squared test of CSR using quadrat counts
##
## data: pp_csr
## X2 = 11.3, df = 8, p-value = 0.3705
## alternative hypothesis: two.sided
##
## Quadrats: 3 by 3 grid of tiles
```

2.4.4 Simulated Cluster Pattern

```
Q3x3_cluster = quadratcount(pp_cluster, nx=3, ny=3)
plot(pp_cluster)
plot(Q3x3_cluster, add=TRUE, col="red", cex=1.5, lty=2)
```

pp_cluster

Test:

```
Q3x3test_cluster = quadrat.test(pp_cluster, 3,3)

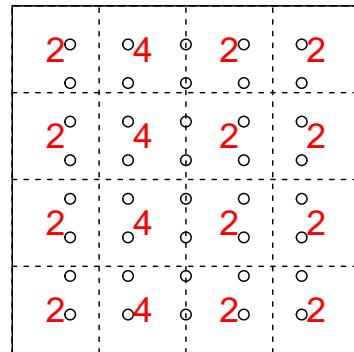
## Warning: Some expected counts are small; chi^2 approximation may be inaccurate

Q3x3test_cluster

##
## Chi-squared test of CSR using quadrat counts
##
## data: pp_cluster
## X2 = 48.65, df = 8, p-value = 1.484e-07
## alternative hypothesis: two.sided
##
## Quadrats: 3 by 3 grid of tiles
```

2.4.5 Simulated Regular Pattern

```
Q3x3_regular = quadratcount(pp_regular, nx=4, ny=4)
plot(pp_regular)
plot(Q3x3_regular, add=TRUE, col="red", cex=1.5, lty=2)
```

pp_regular

Test:

```
Q3x3test_regular = quadrat.test(pp_regular, 3,3)

## Warning: Some expected counts are small; chi^2 approximation may be inaccurate

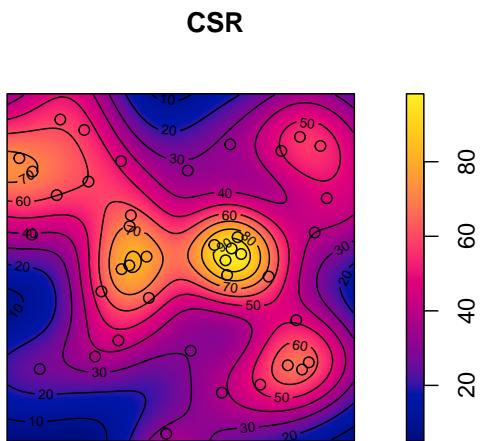
Q3x3test_regular

##
## Chi-squared test of CSR using quadrat counts
##
## data: pp_regular
## X2 = 4.55, df = 8, p-value = 0.3912
## alternative hypothesis: two.sided
##
## Quadrats: 3 by 3 grid of tiles
```

2.5 Kernel Density Smoothing

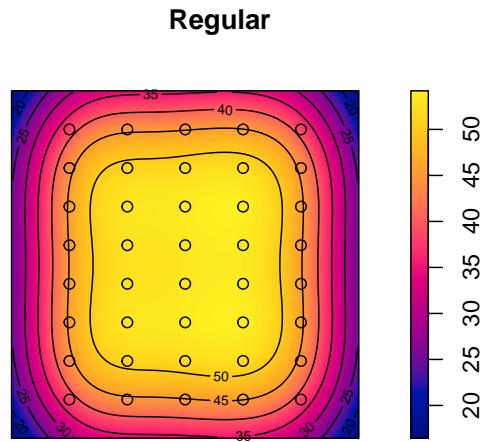
2.5.1 CSR Pattern

```
den <- density(pp_csr, sigma = .1)
plot(den, main = "CSR")
plot(pp_csr, add=TRUE)
contour(den, add = TRUE)
```



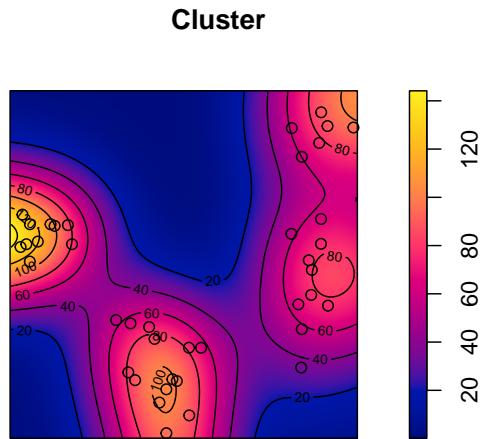
2.5.2 Regular Pattern

```
den <- density(pp_regular)
plot(den, main = "Regular")
plot(pp_regular, add=TRUE)
contour(den, add=TRUE)
```



2.5.3 Cluster Pattern

```
den <- density(pp_cluster)
plot(den, main = "Cluster")
plot(pp_cluster, add=TRUE)
contour(den, add=TRUE)
```



2.6 Kernel Smoothing with a Covariate

2.6.1 Tropical rain forest trees dataset

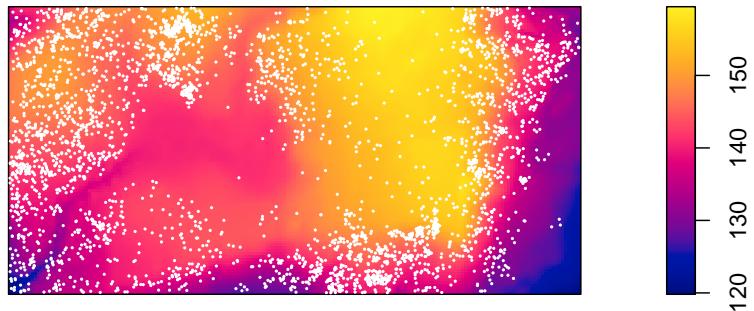
```
data("bei")
```

Assign the elevation covariate to a variable elev by typing

```
elev <- bei.extra$elev
```

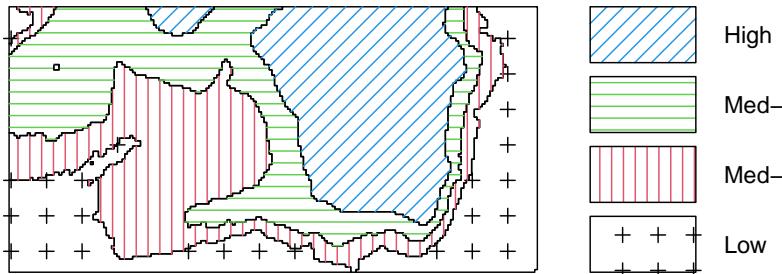
Plot the trees on top of an image of the elevation covariate.

```
plot(elev, main = "")  
plot(bi, add = TRUE, cex = 0.3, pch = 16, cols = "white")
```



For the tropical rainforest data bei, it might be useful to split the study region into several sub-regions according to the terrain elevation:

```
b <- quantile(elev, probs=(0:4)/4, type=2)
Zcut <- cut(elev, breaks=b, labels=c("Low", "Med-Low", "Med-High", "High"))
textureplot(Zcut, main = "")
```



Convert the image from above to a tessellation, count the number of points in each region using quadratcount, and plot the quadrat counts.

```
V <- tess(image=Zcut)
qc <- quadratcount(bei, tess = V)
qc
```

```
## tile
##      Low  Med-Low  Med-High      High
##    714     883     1344     663
```

The output shows the number of trees in each region. Since the four regions have equal area, the counts should be approximately equal if there is a uniform density of trees. Obviously they are not equal; there appears to be a strong preference for higher elevations (dropping off for the highest elevations).

Estimate the intensity in each of the four regions.

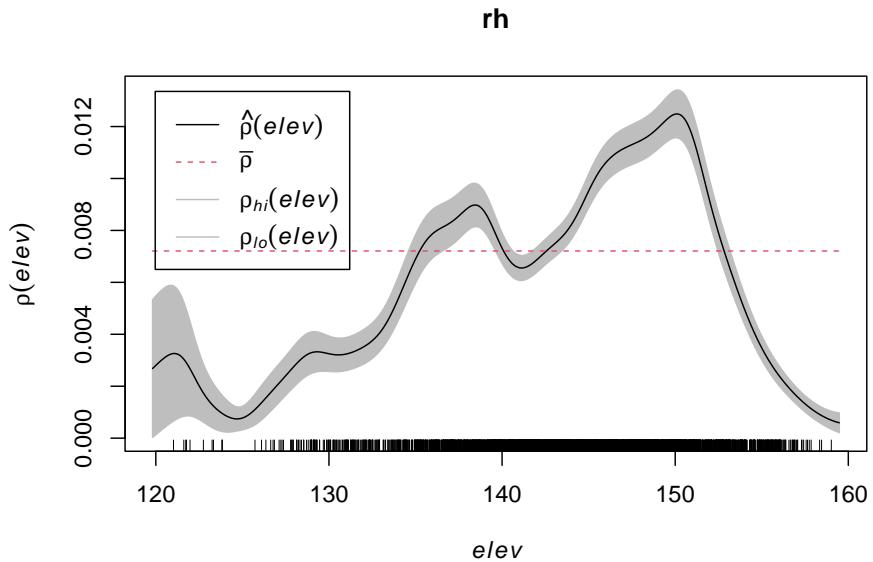
```
intensity(qc)
```

```
## tile
##      Low  Med-Low  Med-High      High
## 0.005623154 0.006960978 0.010593103 0.005228707
```

Assume that the intensity of trees is a function $(u) = (e(u))$ where $e(u)$ is the terrain elevation at location u .

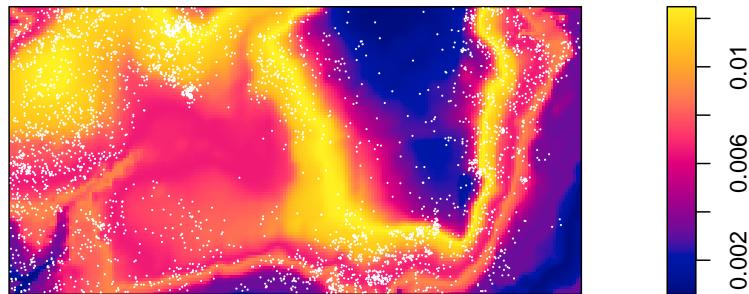
Compute a nonparametric estimate of the function $()$ and plot it by

```
rh <- rhohat(bei, elev)
plot(rh)
```



Compute the predicted intensity based on this estimate of $()$.

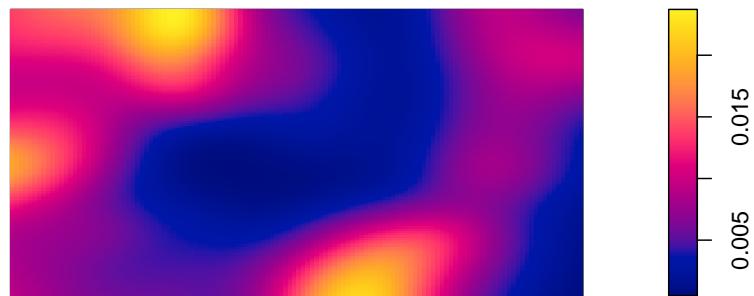
```
predictedrho <- predict(rh)
plot(predictedrho, main = "")
plot(bei, add = TRUE, cols = "white", cex = .2, pch = 16)
```



Compute a non-parametric estimate by kernel smoothing and compare with the predicted intensity above.

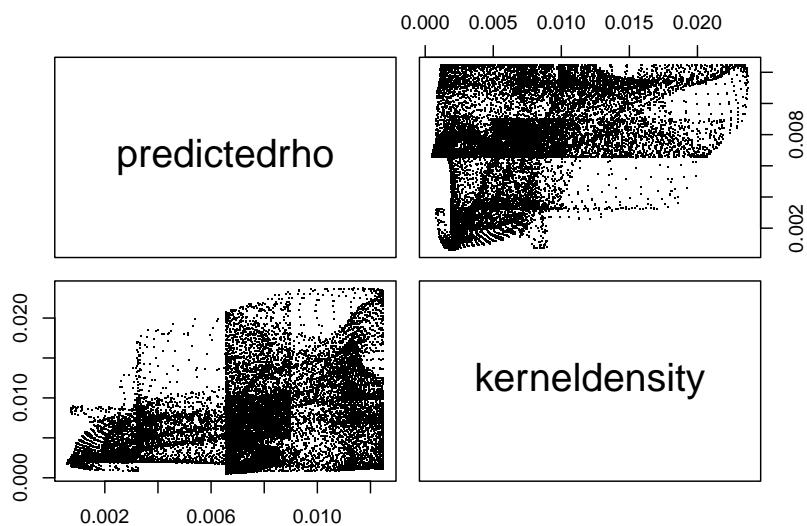
The kernel density estimate of the points is computed and plotted with the following code:

```
kerneldensity <- density(besi, sigma = bw.scott)
plot(kerneldensity, main = "")  
plot(kerneldensity, add = TRUE, cols = "white", cex = .2, pch = 16)
```



Compare the two

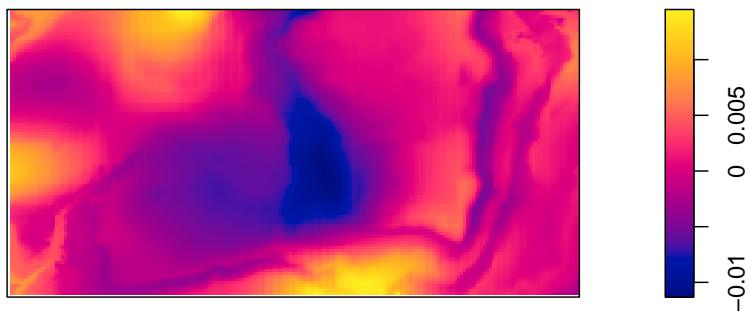
```
pairs(predictedrho, kerneldensity)
```



```
plot(eval.im(kerneldensity-predictedrho))

## Warning: the images 'kerneldensity' and 'predictedrho' were not compatible

eval.im(kerneldensity - predictedrho)
```



Which seems to be quite different form the predicted intensity.

2.7 Distance Measures and Tests

2.7.1 e2e Distances

2.7.1.1 swp dataset

```
PD = pairdist(swp)
class(PD)

## [1] "matrix" "array"

dm <- as.matrix(PD)
dm[1:5, 1:5]
```

```

##          [,1]      [,2]      [,3]      [,4]      [,5]
## [1,] 0.000000 2.700000 3.701351 1.503330 5.433231
## [2,] 2.700000 0.000000 1.004988 1.204159 2.765863
## [3,] 3.701351 1.004988 0.000000 2.200000 1.772005
## [4,] 1.503330 1.204159 2.200000 0.000000 3.931921
## [5,] 5.433231 2.765863 1.772005 3.931921 0.000000

```

```

diag(dm) <- NA
#dm[1:5, 1:5]
wdmin <- apply(dm, 1, which.min)

dmin <- apply(dm, 1, min, na.rm=TRUE)
head(dmin)

```

```

## [1] 1.5033296 0.8544004 1.0049876 0.9055385 1.0770330 0.8544004

```

```

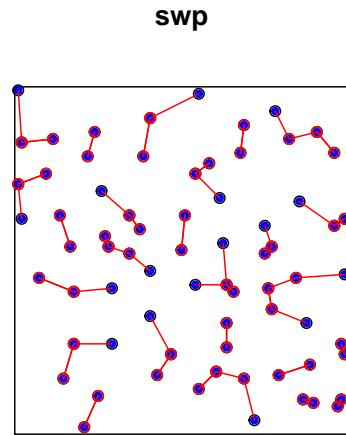
# which is the same as nndist e2e=nndist(swp)

dmin = nndist(swp)

plot(swp)
xy = cbind(swp$x, swp$y)

ord <- rev(order(dmin))
far25 <- ord[1:71]
neighbors <- wdmin[far25]
points(xy[far25, ], col='blue', pch=20)
points(xy[neighbors, ], col='red')
# drawing the lines, easiest via a loop
for (i in far25) {
  lines(rbind(xy[i, ], xy[wdmin[i], ]), col='red')
}

```



2.7.1.2 Simulated CSR Pattern

```
e2e_csr = nnndist(pp_csr)
e2e_csr
```

```
## [1] 0.11056220 0.05419105 0.08163249 0.05574520 0.19907565 0.03191166
## [7] 0.10456756 0.15049858 0.16325765 0.02841510 0.05044346 0.05419105
## [13] 0.10456756 0.07525211 0.02471890 0.08651227 0.03700818 0.06471165
## [19] 0.12663659 0.08163249 0.06471165 0.07525211 0.14362670 0.03195288
## [25] 0.09669706 0.03195288 0.09731910 0.04284774 0.11297958 0.03191166
## [31] 0.13454666 0.02841510 0.02471890 0.06711512 0.10924574 0.04269836
## [37] 0.09947882 0.03815278 0.14362670 0.10235020
```

```
PD = pairdist(pp_csr)
class(PD)
```

```
## [1] "matrix" "array"
```

```
dm <- as.matrix(PD)
dm[1:5, 1:5]
```

```

##          [,1]      [,2]      [,3]      [,4]      [,5]
## [1,] 0.0000000 0.2930205 0.5163115 0.2844957 0.7955061
## [2,] 0.2930205 0.0000000 0.5975953 0.4633681 0.8993362
## [3,] 0.5163115 0.5975953 0.0000000 0.2546708 0.3017974
## [4,] 0.2844957 0.4633681 0.2546708 0.0000000 0.5130861
## [5,] 0.7955061 0.8993362 0.3017974 0.5130861 0.0000000

```

```

diag(dm) <- NA
#dm[1:5, 1:5]
wdmin <- apply(dm, 1, which.min)

dmin <- apply(dm, 1, min, na.rm=TRUE)
head(dmin)

```

```

## [1] 0.11056220 0.05419105 0.08163249 0.05574520 0.19907565 0.03191166

```

```

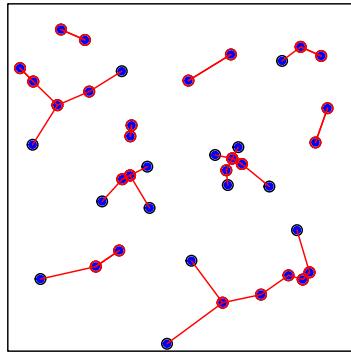
# which is the same as nndist e2e=nndist(swp)

dmin = nndist(pp_csr)

plot(pp_csr)
xy = cbind(pp_csr$x, pp_csr$y)

ord <- rev(order(dmin))
far25 <- ord[1:40]
neighbors <- wdmin[far25]
points(xy[far25, ], col='blue', pch=20)
points(xy[neighbors, ], col='red')
# drawing the lines, easiest via a loop
for (i in far25) {
  lines(rbind(xy[i, ], xy[wdmin[i], ]), col='red')
}

```

pp_csr

2.7.1.3 Simulated Cluster Pattern

```
e2e_cluster = nnndist(pp_cluster)
e2e_cluster

## [1] 0.01854666 0.03502091 0.01854666 0.04969353 0.03502091 0.03390187
## [7] 0.01268286 0.05624330 0.01268286 0.03830134 0.04275255 0.02983313
## [13] 0.04275255 0.02983313 0.03774271 0.03774271 0.03391843 0.01376367
## [19] 0.01376367 0.03500058 0.08344093 0.06403124 0.05422191 0.08344093
## [25] 0.03500058 0.08988091 0.04304986 0.07202173 0.04586212 0.02906394
## [31] 0.08760076 0.11053898 0.04586212 0.02906394 0.05896243 0.07149045
## [37] 0.04361429 0.04361429 0.05745563 0.07483198

PD_cluster = pairdist(pp_cluster)
class(PD_cluster)

## [1] "matrix" "array"

dm_cluster <- as.matrix(PD_cluster)
dm_cluster[1:5, 1:5]
```

```

##          [,1]      [,2]      [,3]      [,4]      [,5]
## [1,] 0.00000000 0.09345138 0.01854666 0.04969353 0.07093497
## [2,] 0.09345138 0.00000000 0.08597921 0.13688899 0.03502091
## [3,] 0.01854666 0.08597921 0.00000000 0.05097721 0.05847323
## [4,] 0.04969353 0.13688899 0.05097721 0.00000000 0.10757315
## [5,] 0.07093497 0.03502091 0.05847323 0.10757315 0.00000000

```

```

diag(dm_cluster) <- NA
wadmin_cluster <- apply(dm_cluster, 1, which.min)

dmin_cluster <- apply(dm_cluster, 1, min, na.rm=TRUE)
head(dmin-cluster)

```

```

##          X      Y
## 1  0.080388175 -0.4397788
## 2  0.018557146 -0.5894417
## 3  0.034750612 -0.4767600
## 4 -0.001070259 -0.4526473
## 5  0.141971489 -0.4168896
## 6 -0.048015500 -0.5340536

```

```

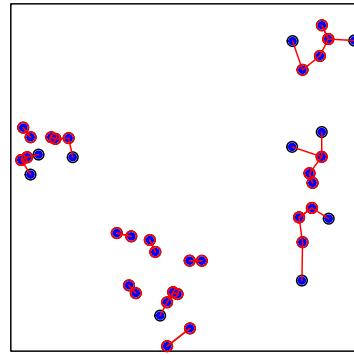
# which is the same as nndist e2e=nndist(swp)

dmin_cluster = nndist(pp_cluster)

plot(pp_cluster)
xy_cluster = cbind(pp_cluster$x, pp_cluster$y)

ord <- rev(order(dmin_cluster))
far25 <- ord[1:40]
neighbors <- wadmin_cluster[far25]
points(xy_cluster[far25, ], col='blue', pch=20)
points(xy_cluster[neighbors, ], col='red')
# drawing the lines, easiest via a loop
for (i in far25) {
  lines(rbind(xy_cluster[i, ], xy_cluster[wadmin_cluster[i], ]), col='red')
}

```

pp_cluster

2.7.1.4 Simulated Regular Pattern

```
e2e_regular = nnndist(pp_regular)
e2e_regular

## [1] 0.1111111 0.1111111 0.1111111 0.1111111 0.1111111 0.1111111 0.1111111 0.1111111
## [8] 0.1111111 0.1111111 0.1111111 0.1111111 0.1111111 0.1111111 0.1111111 0.1111111
## [15] 0.1111111 0.1111111 0.1111111 0.1111111 0.1111111 0.1111111 0.1111111 0.1111111
## [22] 0.1111111 0.1111111 0.1111111 0.1111111 0.1111111 0.1111111 0.1111111 0.1111111
## [29] 0.1111111 0.1111111 0.1111111 0.1111111 0.1111111 0.1111111 0.1111111 0.1111111
## [36] 0.1111111 0.1111111 0.1111111 0.1111111 0.1111111 0.1111111 0.1111111 0.1111111

PD_regular = pairdist(pp_regular)
class(PD_regular)

## [1] "matrix" "array"

dm_regular <- as.matrix(PD_regular)
dm_regular[1:5, 1:5]

## [,1]      [,2]      [,3]      [,4]      [,5]
```

```

## [1,] 0.0000000 0.1666667 0.3333333 0.5000000 0.6666667
## [2,] 0.1666667 0.0000000 0.1666667 0.3333333 0.5000000
## [3,] 0.3333333 0.1666667 0.0000000 0.1666667 0.3333333
## [4,] 0.5000000 0.3333333 0.1666667 0.0000000 0.1666667
## [5,] 0.6666667 0.5000000 0.3333333 0.1666667 0.0000000

```

```

diag(dm_regular) <- NA
wadmin_regular <- apply(dm_regular, 1, which.min)

dmin_regular <- apply(dm_regular, 1, min, na.rm=TRUE)
head(dmin-regular)

```

```

##           X           Y
## 1 -0.05610447 -0.0005489143
## 2 -0.27914228 -0.0569200607
## 3 -0.41836751 -0.0294786165
## 4 -0.61092147 -0.0553659112
## 5 -0.63425768  0.0879645408
## 6 -0.13475501 -0.1903105666

```

```

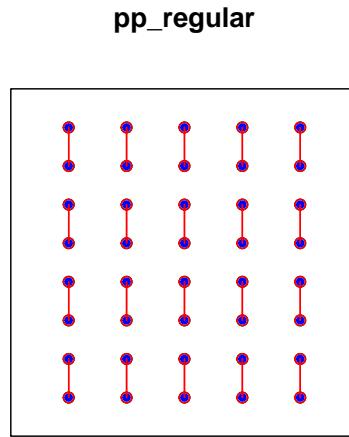
# which is the same as nndist e2e=nndist(swp)

dmin_regular = nndist(pp_regular)

plot(pp_regular)
xy_regular = cbind(pp_regular$x, pp_regular$y)

ord <- rev(order(dmin_regular))
far25 <- ord[1:40]
neighbors <- wadmin_regular[far25]
points(xy_regular[far25, ], col='blue', pch=20)
points(xy_regular[neighbors, ], col='red')
# drawing the lines, easiest via a loop
for (i in far25) {
  lines(rbind(xy_regular[i, ], xy_regular[wadmin_regular[i], ]), col='red')
}

```



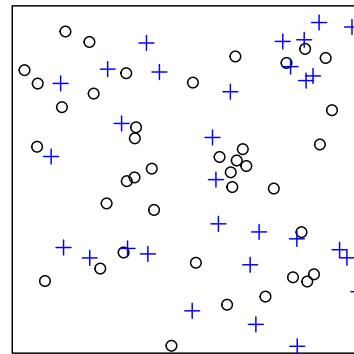
2.7.2 p2e Distances

Generate Random points

```
set.seed(23)
randompoints = matrix(runif(60), ncol=2)
#randompoints = matrix(runif(250), ncol=2)
```

2.7.2.1 CSR Pattern

```
plot(pp_csr)
points(randompoints, col = "blue", pch=3)
```

pp_csr

```

p2e_distances_csr = NULL
mins_csr = NULL
xy = cbind(pp_csr$x, pp_csr$y)

# sqrt((xy[2,1]-randompoints[1,1])^2+(xy[2,2]-randompoints[1,2])^2)
# sqrt((xy[1,1]-randompoints[2,1])^2+(xy[1,2]-randompoints[2,2])^2)

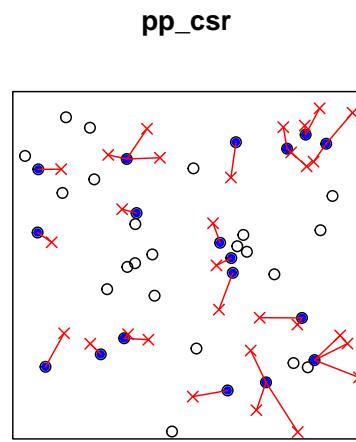
for(i in 1:dim(randompoints)[1]){
  dist1 = matrix(pairdist(rbind(randompoints[i,],xy)),41)

  p2e_distances_csr = c(p2e_distances_csr,min(dist1[2:41,1]))
  mins_csr = c(mins_csr,which.min(dist1[2:41,1]))
}

plot(pp_csr)
ord <- rev(order(p2e_distances_csr))
far25 <- 1:dim(randompoints)[1]
neighbors <- mins_csr
points(randompoints, col='red', pch=4)
points(xy[mins_csr, ], col='blue', pch=20)
# drawing the lines, easiest via a loop
for (i in far25) {

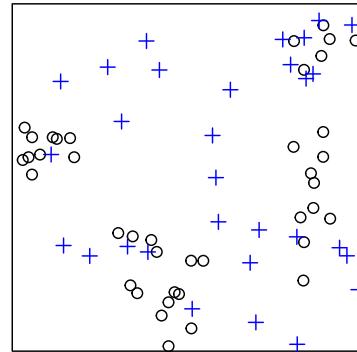
```

```
    lines(rbind(xy[mins_csr[i], ], randompoints[i, ]), col='red')
}
```



2.7.2.2 Cluster Pattern

```
plot(pp_cluster)
points(randompoints, col = "blue", pch=3)
```

pp_cluster

```

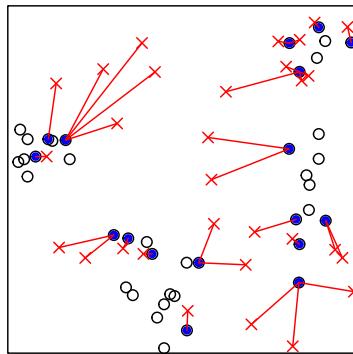
p2e_distances_cluster = NULL
mins_cluster = NULL
xy_cluster = cbind(pp_cluster$x, pp_cluster$y)

for(i in 1:dim(randompoints)[1]){
  dist1 = matrix(pairdist(rbind(randompoints[i,],xy_cluster)),41)

  p2e_distances_cluster = c(p2e_distances_cluster,min(dist1[2:41,1]))
  mins_cluster = c(mins_cluster,which.min(dist1[2:41,1]))
}

plot(pp_cluster)
ord <- rev(order(p2e_distances_cluster))
far25 <- 1:dim(randompoints)[1]
neighbors <- mins_cluster
points(randompoints, col='red', pch=4)
points(xy_cluster[mins_cluster, ], col='blue', pch=20)
# drawing the lines, easiest via a loop
for (i in far25) {
  lines(rbind(xy_cluster[mins_cluster[i], ], randompoints[i, ]), col='red')
}

```

pp_cluster

2.7.2.3 Regular Pattern

```

p2e_distances_regular = NULL
p2e_mins_regular = NULL
xy_regular = cbind(pp_regular$x, pp_regular$y)

for(i in 1:dim(randompoints)[1]){
dist1 = matrix(pairdist(rbind(randompoints[i,],xy_regular)),41)

p2e_distances_regular = c(p2e_distances_regular,min(dist1[2:41,1]))
p2e_mins_regular = c(p2e_mins_regular,which.min(dist1[2:41,1]))
}

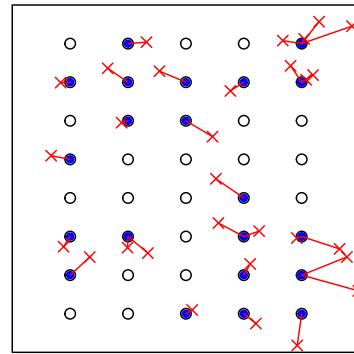
plot(pp_regular)
ord <- rev(order(p2e_distances_regular))
far25 <- 1:dim(randompoints)[1]
neighbors <- p2e_mins_regular
points(randompoints, col='red', pch=4)
points(xy_regular[p2e_mins_regular, ], col='blue', pch=20)
# drawing the lines, easiest via a loop
for (i in far25) {

```

```

    lines(rbind(xy_regular[p2e_mins_regular[i], ], randompoints[i, ]), col='red')
}

```

pp_regular

2.7.3 Clark and Evans Index and Test

2.7.3.1 CSR Pattern

```
clarkevans(pp_csr)
```

```

##      naive  Donnelly      cdf
## 1.0135515 0.9443703 0.9719128

```

```
clarkevans.test(pp_csr)
```

```

##
##  Clark-Evans test
##  No edge correction
##  Z-test
##
## data: pp_csr
## R = 1.0136, p-value = 0.8698
## alternative hypothesis: two-sided

```

2.7.3.2 Cluster Pattern

```
clarkevans(pp_cluster)

##      naive Donnelly      cdf
## 0.5852722 0.5453237 0.5621148

clarkevans.test(pp_cluster)

##
## Clark-Evans test
## No edge correction
## Z-test
##
## data: pp_cluster
## R = 0.58527, p-value = 5.224e-07
## alternative hypothesis: two-sided
```

2.7.3.3 Regular Pattern

```
clarkevans(pp_regular)

##      naive Donnelly      cdf
## 1.405457 1.309526 1.398362

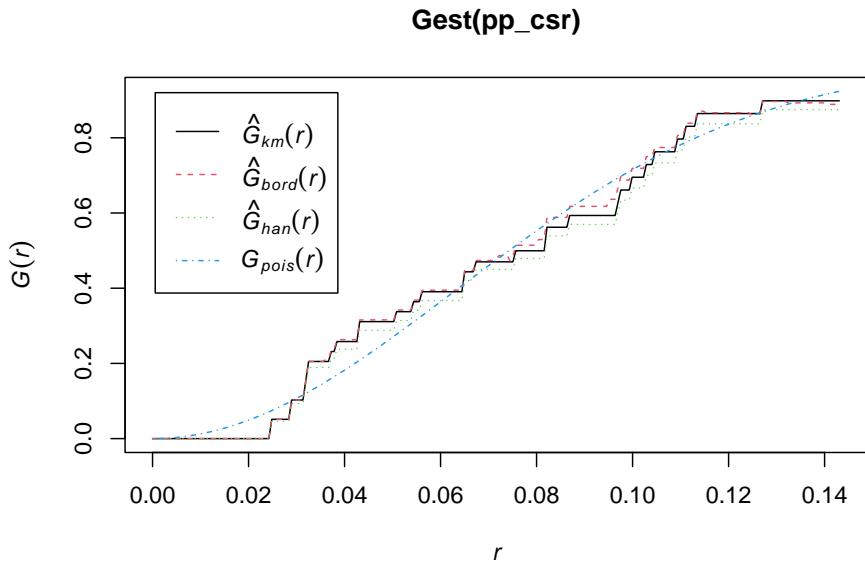
clarkevans.test(pp_regular)

##
## Clark-Evans test
## No edge correction
## Z-test
##
## data: pp_regular
## R = 1.4055, p-value = 9.309e-07
## alternative hypothesis: two-sided
```

2.8 G Function

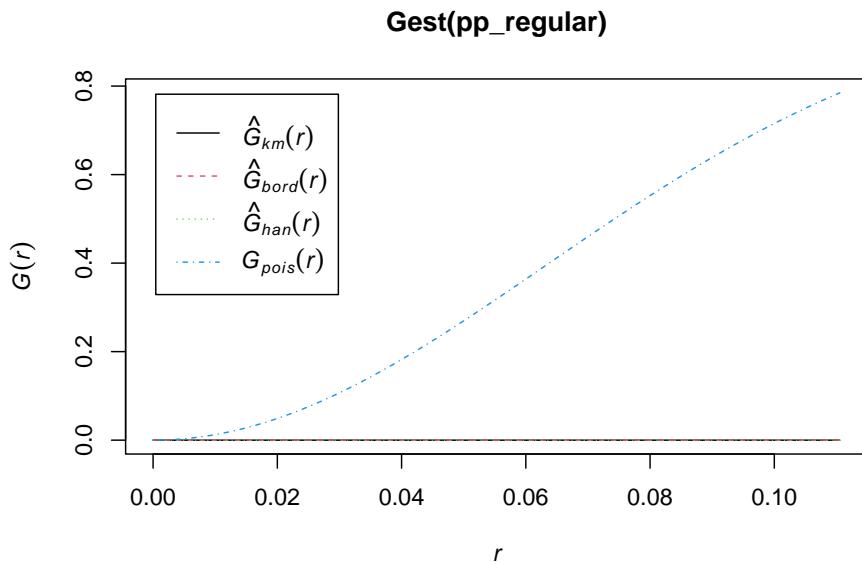
2.8.1 Simulated CSR Pattern

```
plot(Gest(pp_csr))
```



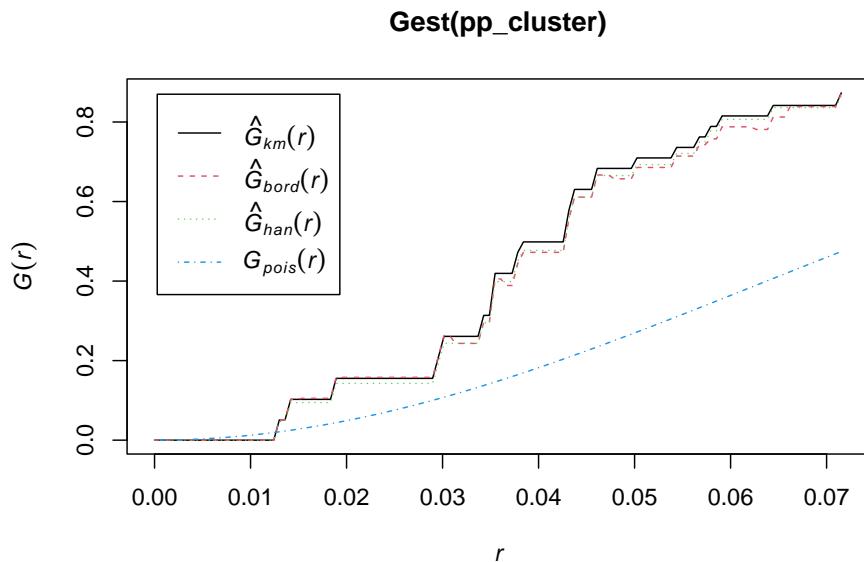
2.8.2 Simulated Regular Pattern

```
plot(Gest(pp_regular))
```



2.8.3 Simulated Cluster Pattern

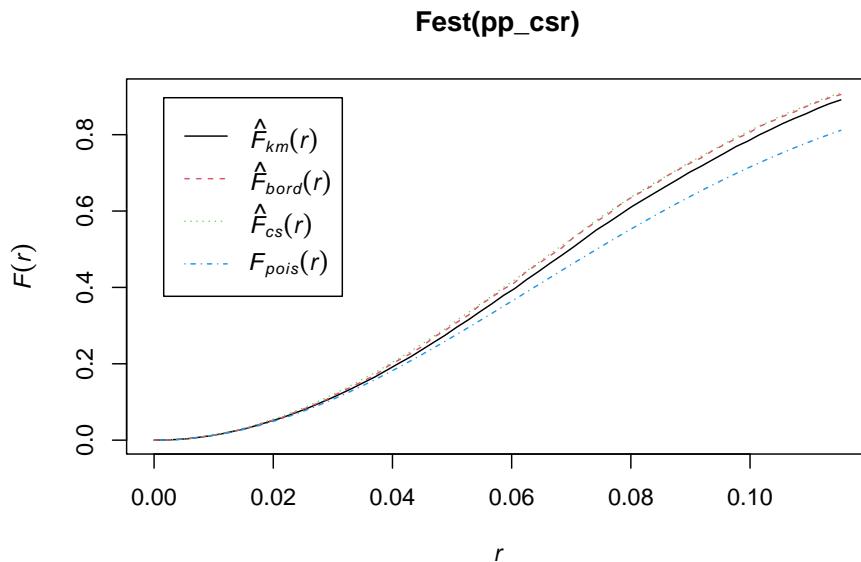
```
plot(Gest(pp_cluster))
```



2.9 F Function

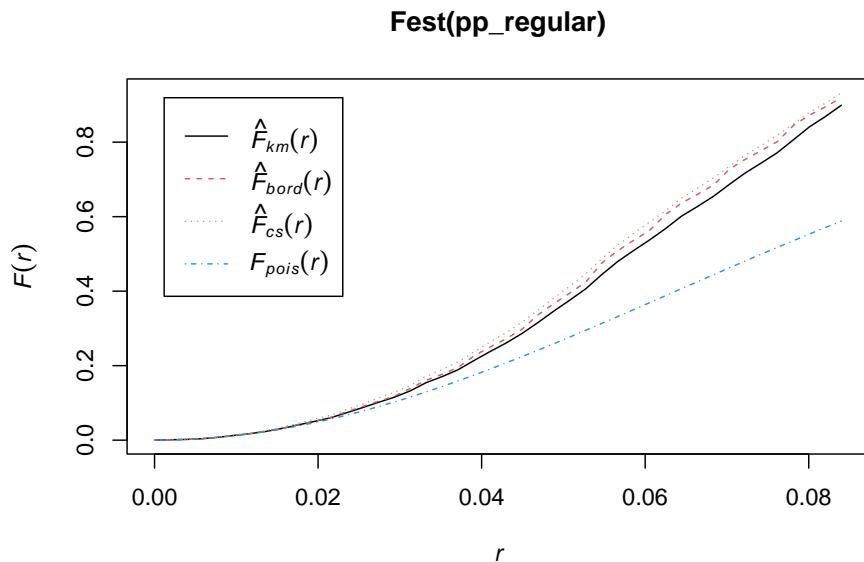
2.9.1 Simulated CSR Pattern

```
plot(Fest(pp_csr))
```



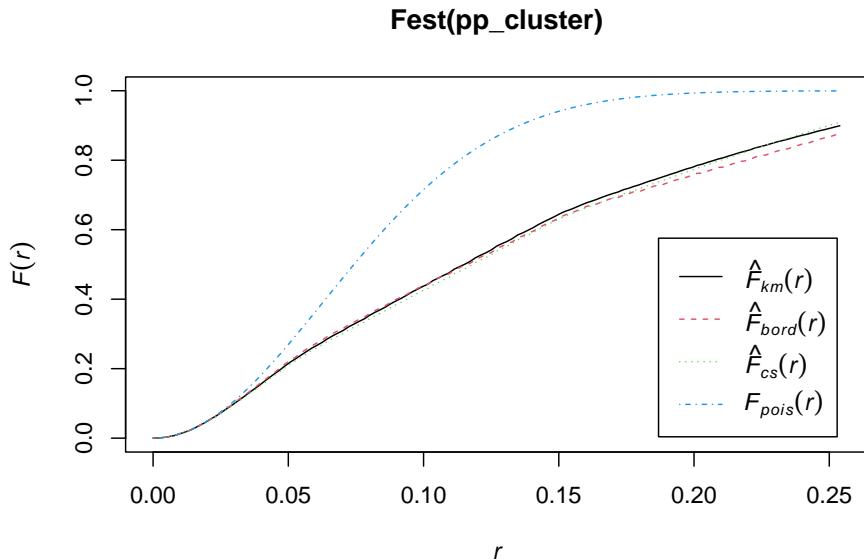
2.9.2 Simulated Regular Pattern

```
plot(Fest(pp_regular))
```



2.9.3 Simulated Cluster Pattern

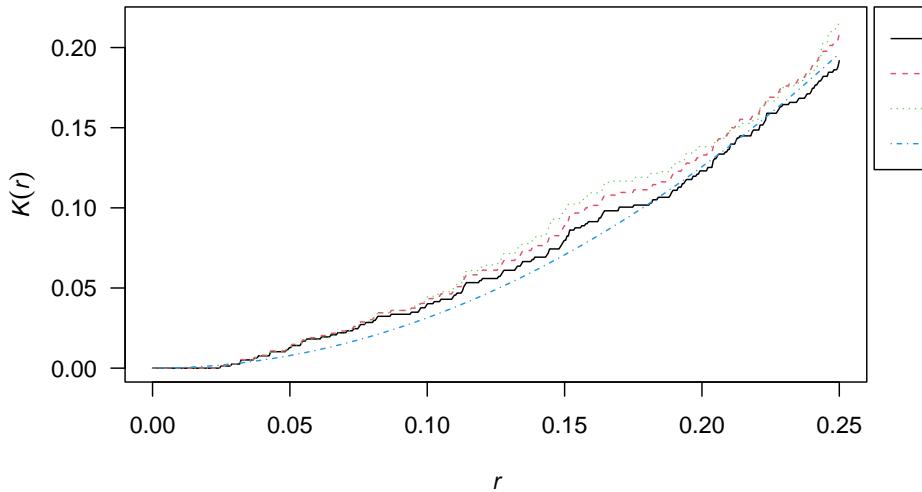
```
plot(Fest(pp_cluster))
```



2.10 Ripley's K Function

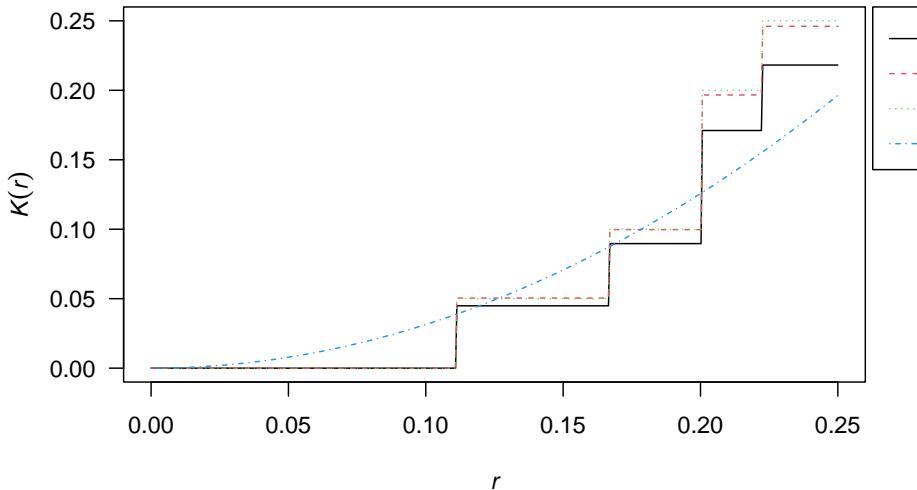
2.10.1 Simulated CSR Pattern

```
K <- Kest(pp_csr)
plot(K, main=NULL, las=1, legendargs=list(cex=0.8, xpd=TRUE, inset=c(1.01, 0) ))
## Warning in min(D[scaledlegbox]): no non-missing arguments to min; returning Inf
```



2.10.2 Simulated Regular Pattern

```
K <- Kest(pp_regular)
plot(K, main=NULL, las=1, legendargs=list(cex=0.8, xpd=TRUE, inset=c(1.01, 0) ))
## Warning in min(D[scaledlegbox]): no non-missing arguments to min; returning Inf
```



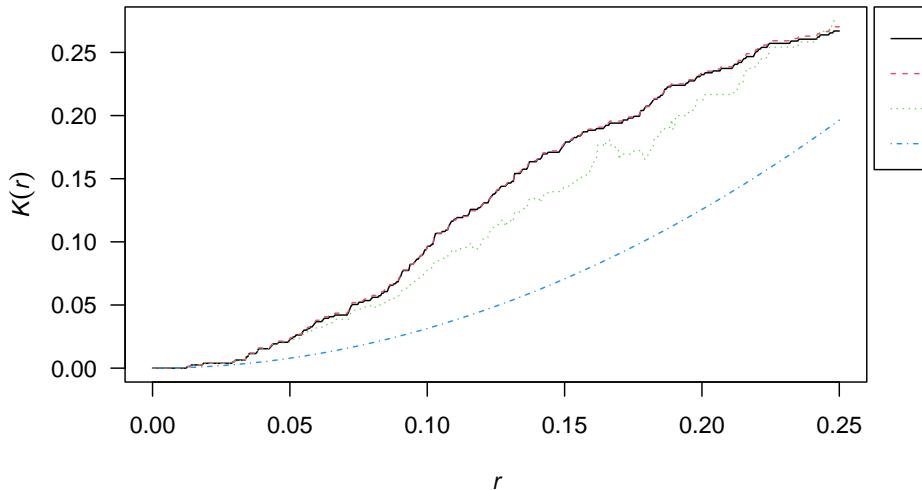
2.10.3 Simulated Cluster Pattern

```

K <- Kest(pp_cluster)
plot(K, main=NULL, las=1, legendargs=list(cex=0.8, xpd=TRUE, inset=c(1.01, 0) ))

## Warning in min(D[scaledlegbox]): no non-missing arguments to min; returning Inf

```



2.11 References:

- Point Pattern Analysis by Yongsung Lee
- An Introduction to Spatial Analysis and Mapping in R 2nd edition by Chris Brunsdon and Lex Comber
- SPP in R by Manuel Gimond
- Animation
- Animation
- Baddeley
- Spatstat

Chapter 3

Spatial Lattice Data Analysis

3.1 Prerequisites

```
library(tmap)
library(spdep)

## Loading required package: spData

## To access larger datasets in this package, install the spDataLarge
## package with: `install.packages('spDataLarge',
## repos='https://nowosad.github.io/drat/', type='source')` 

library(maptools)

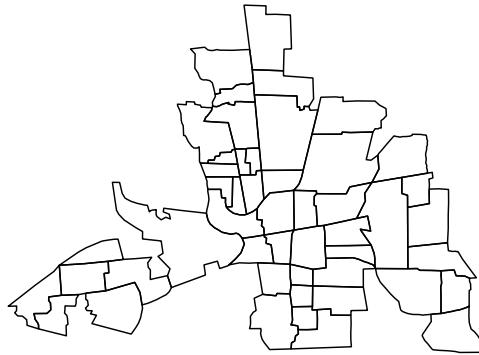
## Checking rgeos availability: TRUE
## Please note that 'maptools' will be retired by the end of 2023,
## plan transition at your earliest convenience;
## some functionality will be moved to 'sp'.
```

3.2 Columbus Dataset

```
example(columbus)
```

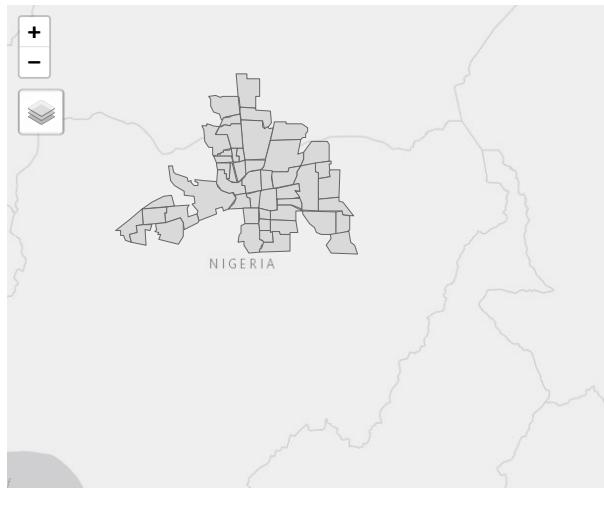
```
##  
## colmbs> columbus <- st_read(system.file("shapes/columbus.shp", package="spData")[1]  
##  
## colmbs> col.gal.nb <- read.gal(system.file("weights/columbus.gal", package="spData"))
```

```
plot(columbus$geometry)
```



```
library(tmap)  
tm_shape(columbus) + tm_polygons()
```

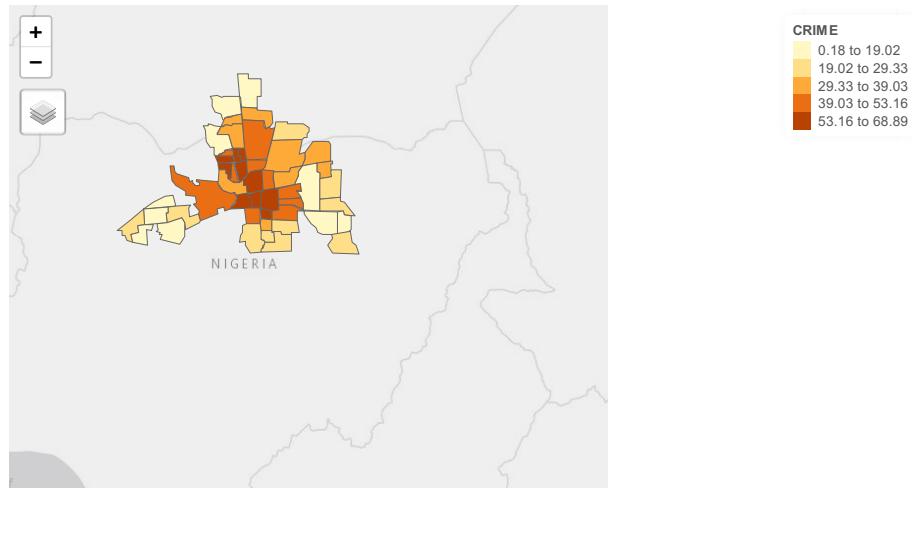
```
## Warning: Current projection of shape columbus unknown. Long-lat (WGS84) is  
## assumed.
```



3.2.1 Plot of Crime Variable

```
tm_shape(columbus) + tm_polygons(style="quantile", col = "CRIME") +  
  tm_legend(outside = TRUE, text.size = .8)
```

```
## Warning: Current projection of shape columbus unknown. Long-lat (WGS84) is  
## assumed.
```



Leaflet | Tiles © Esri — Esri, DeLorme, NAVTEQ

3.3 Obtain the Centroids

```
#columbus <- st_read(system.file("shapes/columbus.shp", package="spData")[1], quiet=TRUE)
#col.gal.nb <- read.gal(system.file("weights/columbus.gal", package="spData")[1])
coords <- st_coordinates(st_centroid(st_geometry(columbus)))
```

3.4 Obtain the Neighbourhood Relationship

3.4.1 Contiguity Based Neighbours

3.4.1.1 Rook style NB

```
nbRook <- poly2nb(as(columbus, "Spatial"), queen = FALSE)
class(nbRook)
```

```
## [1] "nb"
```

```
nbRook
```

```
## Neighbour list object:  
## Number of regions: 49  
## Number of nonzero links: 200  
## Percentage nonzero weights: 8.329863  
## Average number of links: 4.081633
```

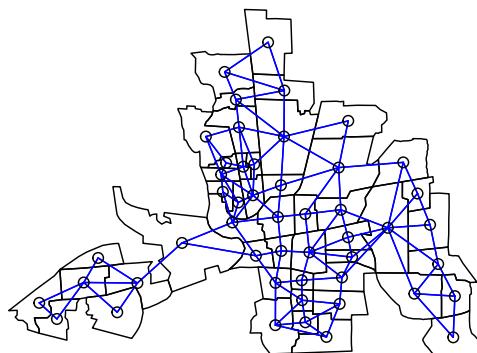
As you can see this is an “nb” class object and for each polygon (in the Columbus dataset we have 49 polygons) in our polygon object, nbRook lists all neighboring polygons. For example, to see the neighbors for the first polygon in the object, type:

```
nbRook[[1]]
```

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

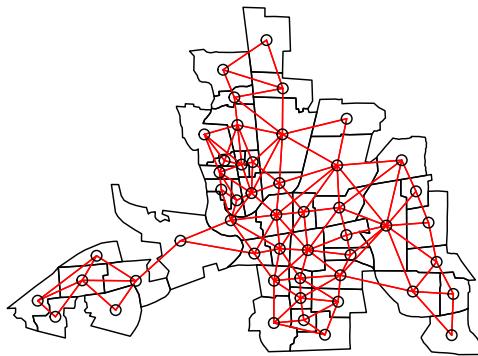
Polygon 1 has 2 neighbours according to Queen Style Contiguity Based Neighbour definition. These 2 neighbours are Poly2 and Poly3.

```
plot(st_geometry(columbus))  
plot(nbRook, coords, add=TRUE, col="blue")
```



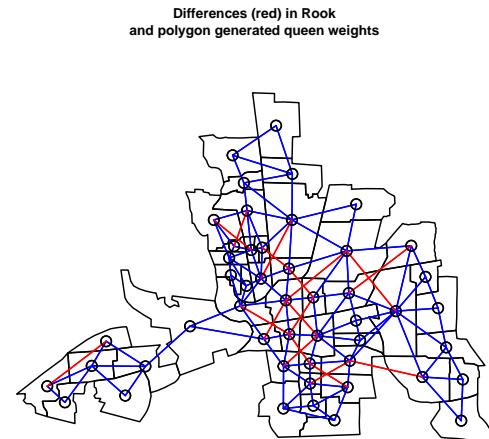
3.4.1.2 Queen style NB

```
nbQueen <- poly2nb(as(columbus, "Spatial"), queen = TRUE)
plot(st_geometry(columbus))
plot(nbQueen, coords, add=TRUE, col="red")
```



3.4.1.3 Difference between Rook and Queen

```
dQueenRook <- diffnb(nbQueen, nbRook)
plot(st_geometry(columbus))
plot(nbRook, coords, add = TRUE, col = "blue")
plot(dQueenRook, coords, add = TRUE, col = "red")
title(main=paste("Differences (red) in Rook",
"and polygon generated queen weights", sep="\n"), cex.main=0.6)
```

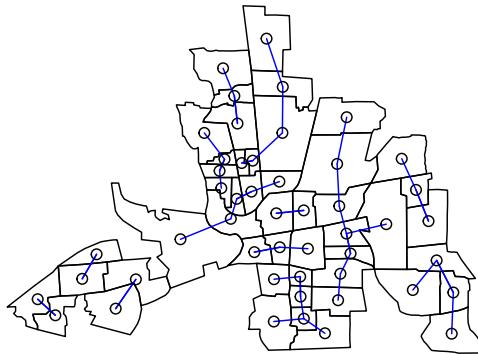


```
# poly2nb with sf sfc_MULTIPOLYGON objects
```

3.4.2 Distance Based Neighbours

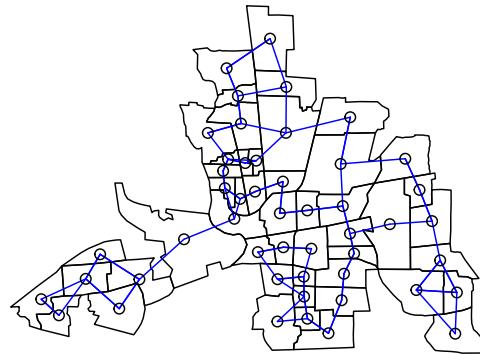
3.4.2.1 1st Nearest Neighbour

```
whoisfirstnear=knearneigh(coords,k=1,latlong=TRUE)
knn1columbus=knn2nb(whoisfirstnear)
plot(st_geometry(columbus))
plot(knn1columbus, coords, add=TRUE, col="blue")
```



3.4.2.2 Two Nearest Neighbour

```
whois2near=knearestneigh(coords,k=2,longlat=TRUE)
knn2columbus=knn2nb(whois2near)
plot(st_geometry(columbus))
plot(knn2columbus, coords, add=TRUE, col="blue")
```



3.4.2.3 Critical Cut-off

```

distBetwNeigh1=nbdists(knn1columbus,coords,longlat=TRUE)
all.linkedTresh=max(unlist(distBetwNeigh1))
all.linkedTresh

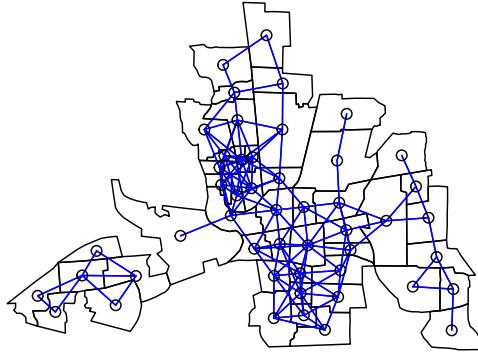
## [1] 67.50447

dnbTresh1=dnearneigh(coords,0,68,longlat=TRUE)
summary(dnbTresh1)

## Neighbour list object:
## Number of regions: 49
## Number of nonzero links: 252
## Percentage nonzero weights: 10.49563
## Average number of links: 5.142857
## Link number distribution:
##
##    1   2   3   4   5   6   7   8   9  10  11 
##    4   8   6   2   5   8   6   2   6   1   1 
## 4 least connected regions:
## 6 10 21 47 with 1 link

```

```
## 1 most connected region:  
## 28 with 11 links  
  
plot(st_geometry(columbus))  
plot(dnbTresh1, coords, add=TRUE, col="blue")
```



If you examine this neighbourhood list object a bit more closely you will see that, Polygon1 is neighbours with Poly2 and Poly3; Polygon2 is neighbours with Poly1 and Poly4; Polygon3 is neighbours with Poly1, Poly4 and Poly5 and so on:

```
dnbTresh1[1:3]  
  
## [[1]]  
## [1] 2 3  
##  
## [[2]]  
## [1] 1 4  
##  
## [[3]]  
## [1] 1 4 5
```

In calculation of the weights that will be used to create the spatially lagged variables, only the neighbours values are going to be used. For example for the 1st Polygon, the values of neighbouring Poly2 and Poly3 will be used. Before doing this the nb values need to be standardized, so that the row

3.5 Neighbours to Weights Matrix

Here we will work with the Critical Cut-off nearest neighbours

```
dnbTresh1.listw=nb2listw(dnbTresh1, style="W", zero.policy=FALSE)
class(dnbTresh1.listw)

## [1] "listw" "nb"

dnbTresh1.listw$weights[1:5]

## [[1]]
## [1] 0.5 0.5
##
## [[2]]
## [1] 0.5 0.5
##
## [[3]]
## [1] 0.3333333 0.3333333 0.3333333
##
## [[4]]
## [1] 0.25 0.25 0.25 0.25
##
## [[5]]
## [1] 0.2 0.2 0.2 0.2 0.2
```

In matrix format rather than a list:

```
listw2mat(dnbTresh1.listw)[1:5,1:5]

##      [,1] [,2] [,3]      [,4]      [,5]
## 1 0.0000000 0.50 0.50 0.0000000 0.0000000
## 2 0.5000000 0.00 0.00 0.5000000 0.0000000
## 3 0.3333333 0.00 0.00 0.3333333 0.3333333
## 4 0.0000000 0.25 0.25 0.0000000 0.0000000
## 5 0.0000000 0.00 0.20 0.0000000 0.0000000
```

3.6 Moran's I

3.6.1 Manual Calculation of Moran's I Statistic with Matrix Operations

We can manually calculate the Moran's I statistic using the lag operation with our chosen weights matrix, nearest neighbours within 68kms of radius.

```

weightsmatrix_s = listw2mat(dnbTresh1.listw)
Y_s = columbus$CRIME - mean(columbus$CRIME)

t(Y_s) %*% weightsmatrix_s %*% Y_s / (t(Y_s) %*% Y_s)

##           [,1]
## [1,] 0.5518257

```

The spatially lagged values of Y variable are obtained with W^*Y as in your Moran's I test statistic. Moran's I is measuring the dependency of your Y values to the neighbouring Y values after all.

3.6.2 Moran's I Test with `moran.test()` function

The same value can be simply obtained using the `moran.test()` function.

```

moran.test(columbus$CRIME,dnbTresh1.listw)

##
## Moran I test under randomisation
##
## data: columbus$CRIME
## weights: dnbTresh1.listw
##
## Moran I statistic standard deviate = 5.6206, p-value = 9.514e-09
## alternative hypothesis: greater
## sample estimates:
## Moran I statistic      Expectation      Variance
##          0.55182569     -0.02083333     0.01038068

```

Examine the following:

```

lagY_s = weightsmatrix_s %*% Y_s
head(lagY_s)

```

```

##           [,1]
## 1 -10.414556
## 2 -11.071954
## 3 -2.180407
## 4 -13.120658
## 5 12.195025
## 6 -4.612907

```

This is the spatially lagged value for the Y variable. Technically it is the average Y values of the neighbouring polygons around each and every single polygon. For example take the 1st polygon, the lagged value of the 1st polygon is the average of the values for 2nd and 3rd polygons:

```
mean(c(Y_s[2], Y_s[3]))
```

```
## [1] -10.41456
```

*# We can automatically create spatially lagged values of Y variable, this corresponds to $W*Y$ in your notes*

```
lagY_s <- lag.listw(dnbTresh1.listw, Y_s)
```

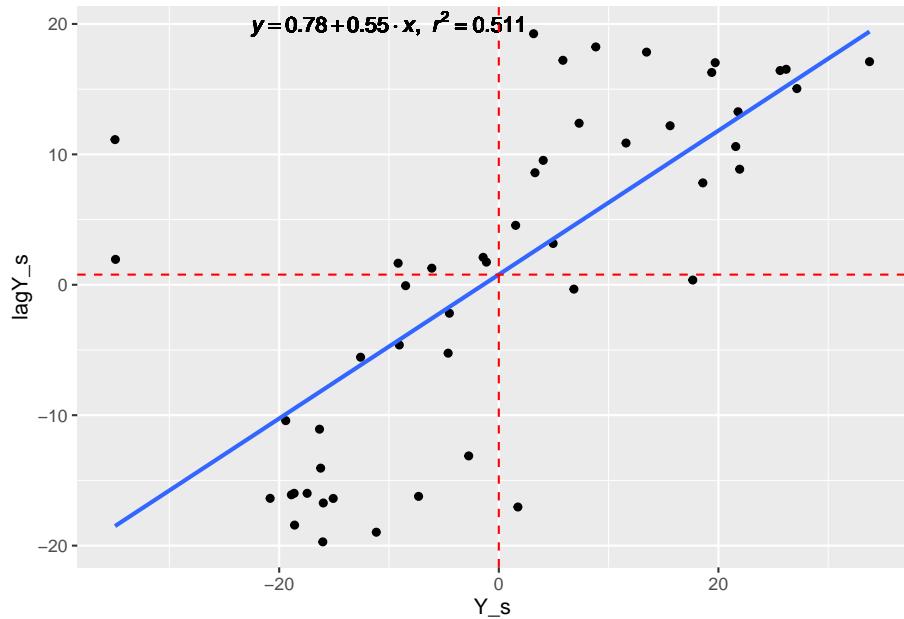
```
lm_eqn <- function(df, y, x){
  m <- lm(y ~ x, df);
  eq <- substitute(italic(y) == a + b %.% italic(x)*", "~~italic(r)^2~"=~r2,
    list(a = format(unname(coef(m)[1]), digits = 2),
        b = format(unname(coef(m)[2]), digits = 2),
        r2 = format(summary(m)$r.squared, digits = 3)))
  as.character(as.expression(eq));
}

dataMoransI = data.frame(Y_s, lagY_s)

library(ggplot2)

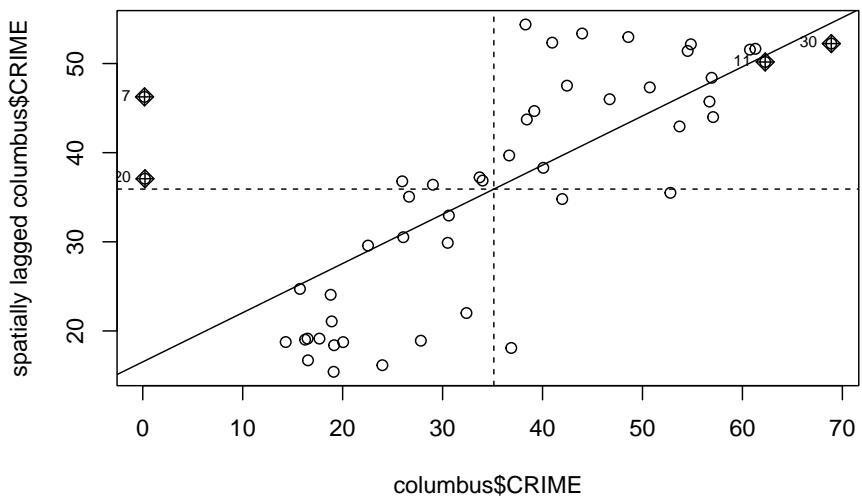
ggplot(dataMoransI, aes(x = Y_s, y = lagY_s)) + geom_point() + geom_smooth(method = "lm", se=FALSE)

## `geom_smooth()` using formula = 'y ~ x'
```



Here the slope of the lm is the Moran's I value. The same plot can be obtained using the `moran.plot()` function as follows:

```
moran.plot(columbus$CRIME, dnbTresh1.listw)
```



3.7 Local Moran's I

```
localmoranstatistics = localmoran(Y_s,dnbTresh1.listw)
head(localmoranstatistics)

##           Ii      E.Ii     Var.Ii      Z.Ii Pr(z != E(Ii))
## 1 0.73681849 -0.0285985420 0.666144891 0.9378077 0.34834327
## 2 0.65915397 -0.0202502140 0.475741297 0.9850149 0.32461676
## 3 0.03579329 -0.0015396867 0.024041007 0.2407777 0.80972741
## 4 0.13113818 -0.0005707572 0.006541757 1.6284260 0.10343458
## 5 0.69380337 -0.0184931910 0.162742823 1.7656716 0.07745096
## 6 0.15242670 -0.0062384562 0.303777355 0.2878749 0.77344247

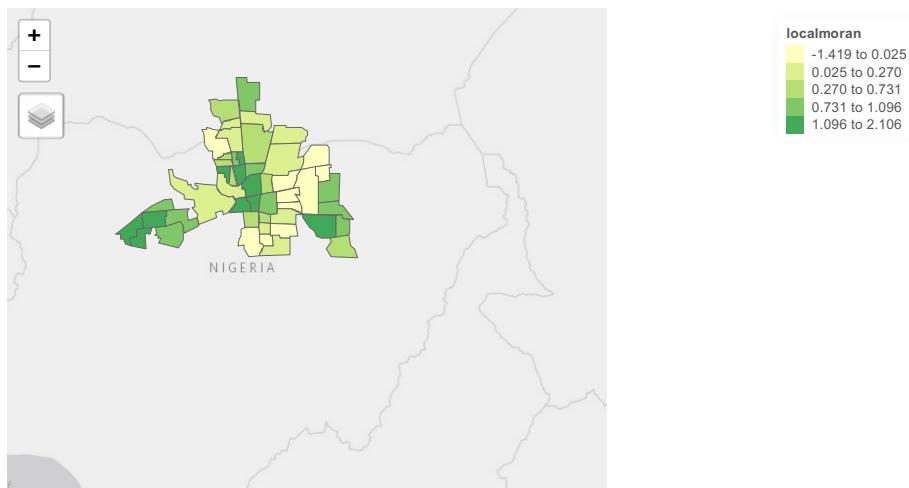
columbus$localmoran = localmoranstatistics[,1]
```

Plot of the Local Moran Statistics

```
tm_shape(columbus) + tm_polygons(style="quantile", col = "localmoran") +
  tm_legend(outside = TRUE, text.size = .8)

## Warning: Current projection of shape columbus unknown. Long-lat (WGS84) is
## assumed.

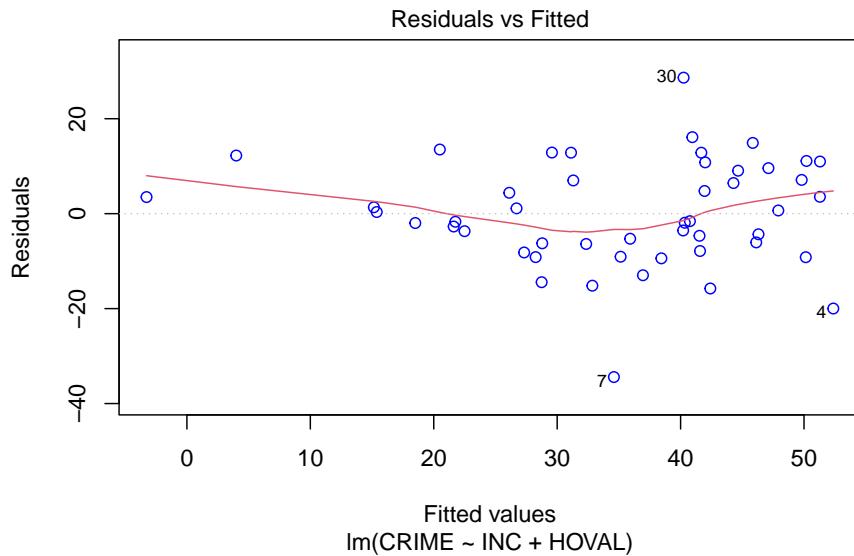
## Variable(s) "localmoran" contains positive and negative values, so midpoint is set to 0. Set m
```



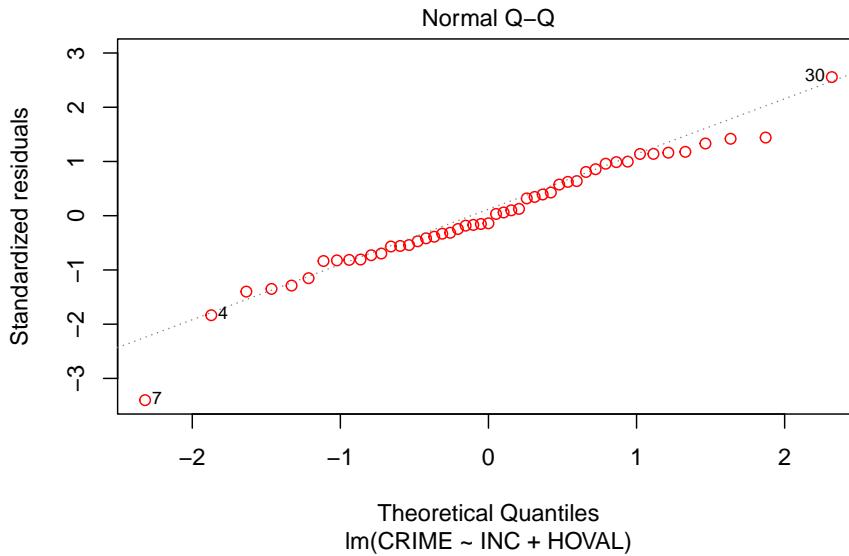
3.8 Week 2 of Lattice

3.9 OLS

```
OLScolumbus <- lm(CRIME ~ INC + HOVAL, data=columbus)
yhat <- predict(OLScolumbus)
resids = resid(OLScolumbus)
plot(OLScolumbus, which=1, col=c("blue")) # Residuals vs Fitted Plot
```



```
plot(OLScolumbus, which=2, col=c("red")) # Q-Q Plot
```



3.10 Moran's I Test of Residual

```
ols.resid.moran <- lm.morantest(OLScolumbus, dnbTresh1.listw)
summary(ols.resid.moran)
```

```
##          Length Class  Mode
## statistic     1   -none- numeric
## p.value       1   -none- numeric
## estimate      3   -none- numeric
## method        1   -none- character
## alternative   1   -none- character
## data.name     1   -none- character

ols.resid.moran

##
## Global Moran I for regression residuals
##
## data:
## model: lm(formula = CRIME ~ INC + HOVAL, data = columbus)
## weights: dnbTresh1.listw
```

```
##  
## Moran I statistic standard deviate = 3.0562, p-value = 0.001121  
## alternative hypothesis: greater  
## sample estimates:  
## Observed Moran I      Expectation      Variance  
##          0.266348450     -0.032344498    0.009551988
```