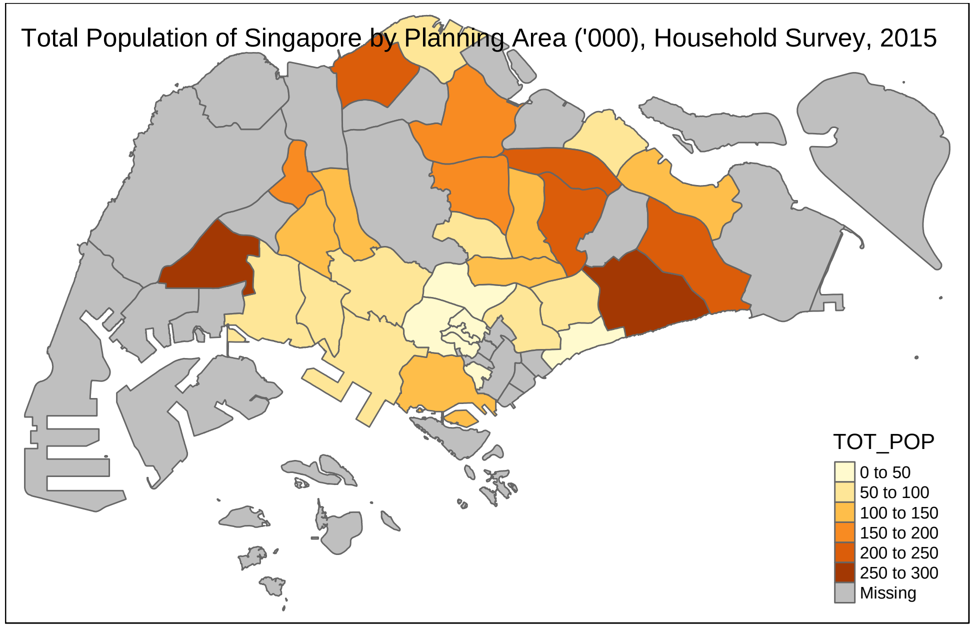
ECON6027 6a

Areal units and neighbours

Required R packages

- sf
- spdep

Areal/lattice/regional data



- These are spatial data where the **domain D is "fixed and discrete"** (non-random and countable).
  - Eg: postal codes, GRCs, planning areas, remotely sensed data reported by pixels (such as data coming from satellites).
- Spatial locations with areal data are often referred to as "**sites**" or "**areal units**".
- One of the main differences between point data and areal data is that, in practice areal data are **spatially aggregated** over areal regions. (Mathematically this refers to an integration of a continuous spatial attribute).
  - yield measures on an agricultural plot
  - event counts (such as deaths, crimes, voter turnout, etc.) for various sites (such as postal codes, regions, states, etc.)
- Spatial aggregation is becoming increasingly common due to the growing need to confidentiality and privacy of data records.

Areal/lattice/regional data

- If areal units are **irregular**, a more precise term would be "regional data".
- Given the discrete nature of the collection of sites, areal data can be **exhaustive** (another differentiating feature compared to point data or geospatial data).
  - For example voter turnout data provide the number for every electoral unit and the issue of predicting the number for any other are does not arise.

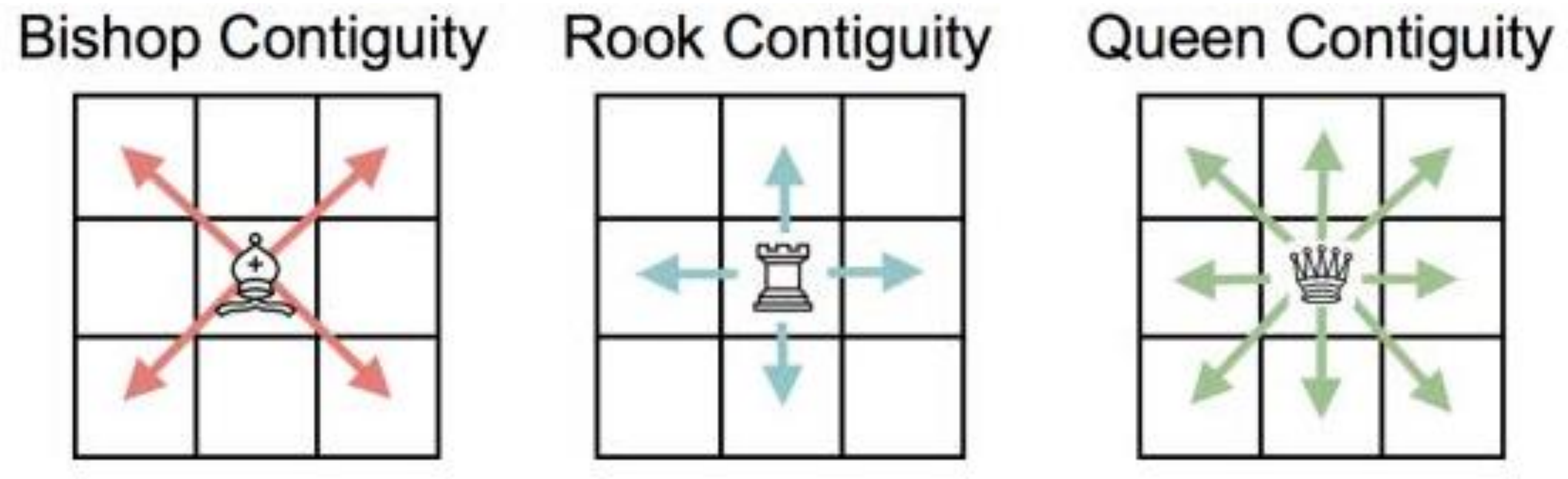
Spatial autocorrelation and neighbours

- Spatial relationships are best modelled based on the principle of spatial neighbours.
- We assume that the influence of spatial neighbours among *n* spatial units can be quantified using a ***spatial weight***.

Neighbours…

- Once you have divided the complete domain into the areal units, next step is to generate a matrix (n X n) that indicates how each areal unit is related to one another.
- This is the so-called **"spatial weights"** matrix (aka connectivity matrix).
- Two areal units may be neighbours of each other based on
  - Distance (geographic, economic or social distance)
  - Nearest neighbour
  - Contiguity (sharing borders)
  - Or a combination

$$W_n = \begin{bmatrix} w_{11} & \cdots & w_{n1} \\ \cdots & w_{ij} & \\ w_{1n} & & w_{nn} \end{bmatrix} \qquad \text{where } w_{ij} = \begin{cases} 1 & if\ j \in N(i) \\ 0 & o/w \end{cases}$$

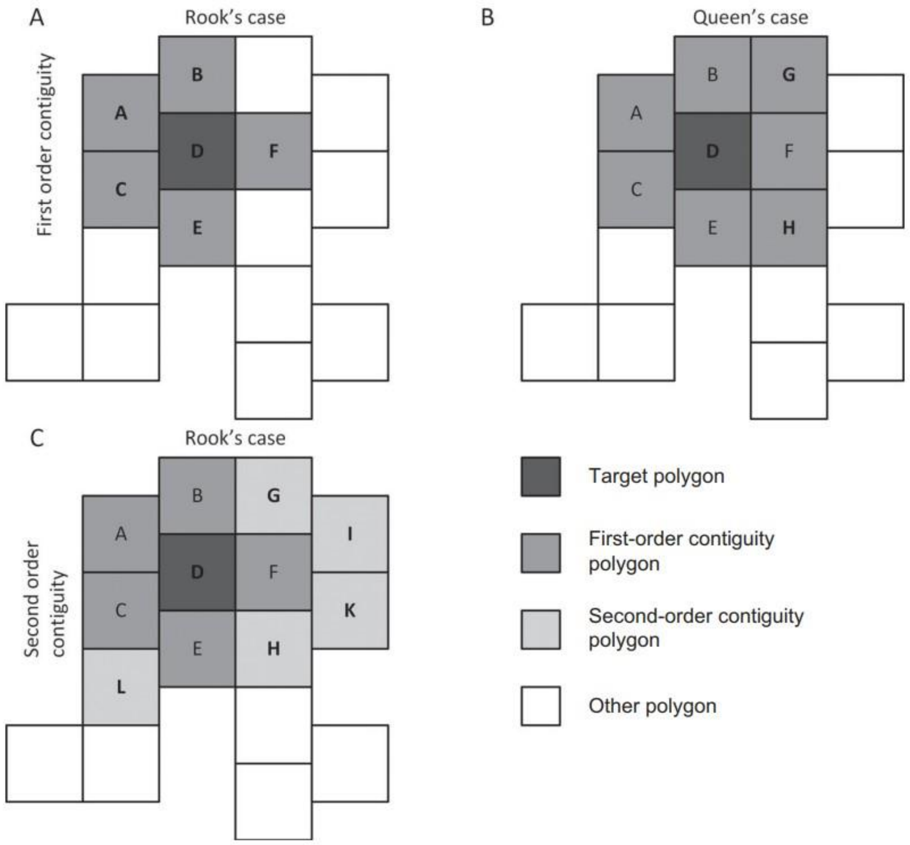


Contiguity criteria on a "regular" lattice

If data are observed on a regular rectangular lattice, the contiguity neighbours can be defined using,

- Rook criterion (cardinal neighbours)
- Queen criterion (cardinal and ordinal neighbours)
- Bishop criterion (less popular)
- Circular
- Group interaction

Contiguity criteria on regional data



However, in practice we must deal with irregularly spaced areal units such as planning areas, regions, countries etc.

W matrix example

- See uk.xlsx for an example of a W matrix for UK regions based on contiguity.
- The fact that northern Ireland doesn't have any contiguity poses a (computational) problem since the rank(W) < n.
- The given W matrix is a symmetric matrix which means neighbours are mutual, however, this need not be the case.
- It is normal for weights matrices to be row standardised in applications (especially in spatial econometrics) as:

$$w_{ij}^* = \frac{w_{ij}}{\sum_j w_{ij}}$$

- The implication of row standardisation is that,

$$\sum_i \sum_j w_{ij}^* = n$$

Where n is the number of areal units.

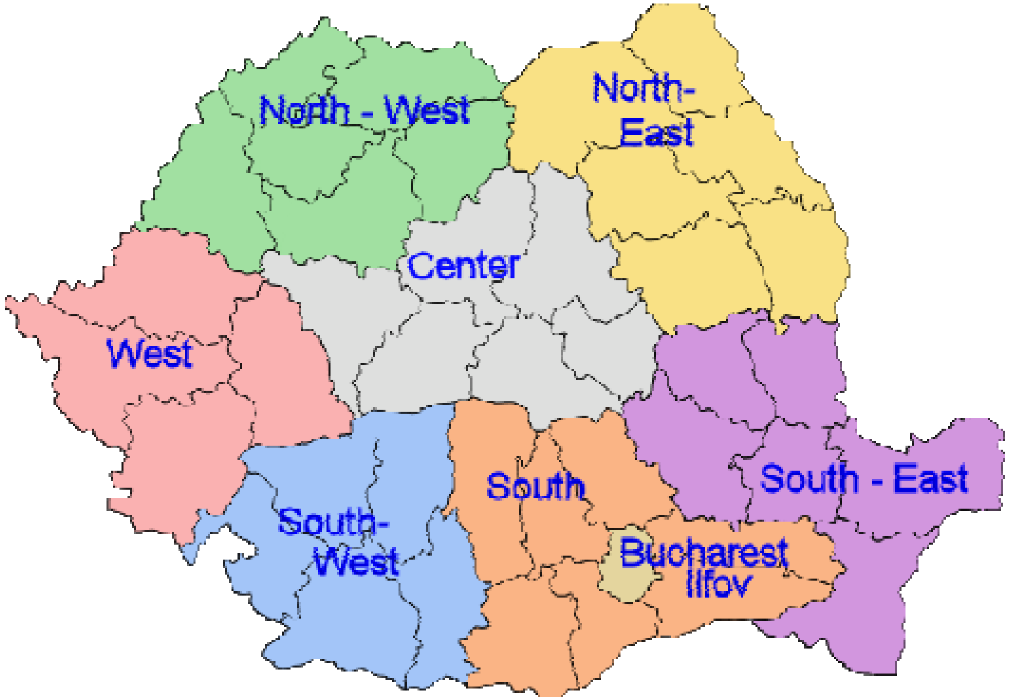
GAL file

- The portable form of a weights matrix that indicate information on neighbouring areal units are *usually* (but not necessarily) saved as a GAL file.
- A .gal file is a file produced by the software GeoDa.
- A .gal file is useful since it allows flexibility in the creation and the updating of neighbours, however, may not be feasible for large n.
- For small datasets, we can create a .gal file from scratch ourselves without having the need to use GeoDa.

Create your own GAL file (without GeoDa)

- You can save the information on neighbours and save this text file with the extension .GAL.
- A .GAL file must have the following format:
  - Header: line 1, starts with a mandatory 0 and the number of areal units (and optional column name)
  - A region identifier will be assigned to each areal unit from 1 to n.
  - Line 2: the region identifier of region 1 followed by the number of neighbouring regions of regions 1.
  - Line 3: region identifiers of the neighbours of region 1.
  - And so on…
  - You need to **leave an empty line at the end**.

GAL file example for regions of Romania



The figure shows the boundaries of the 8 Romanian NUTS2 regions.

1. North-west
2. Centre
3. North-east
4. South-east
5. South
6. Bucharest
7. South-west
8. West

Copy and paste the following data in a text file and save as a *.GAL file (you need to leave an empty line at the end.).
Instructions on how to change the file extension to *.gal: https://www.wikihow.com/Change-a-File-Extension
(Windows is preferred)

```
0 8 romania rom_regions
1 3
3 2 8
2 6
1 3 4 5 7 8
3 3
1 2 4
4 3
2 3 5
5 4
2 4 6 7
6 1
5
7 3
2 5 8
8 3
1 2 7
```

Activity A

Create a GAL file for the 8 South Asian countries.

- Two countries are to be considered close neighbours if they share a common land border or if the shortest distance separated by water is less than 100km.
- Use the indexing found here: https://en.wikipedia.org/wiki/South_Asia

Read neighbours from an existing GAL file

```
> (uk_nb = read.gal("uk_cont.gal"))
```

Neighbour list object:

- Number of regions: 12
- Number of nonzero links: 42
- Percentage nonzero weights: 29.16667
- Average number of links: 3.5

R class “nb”

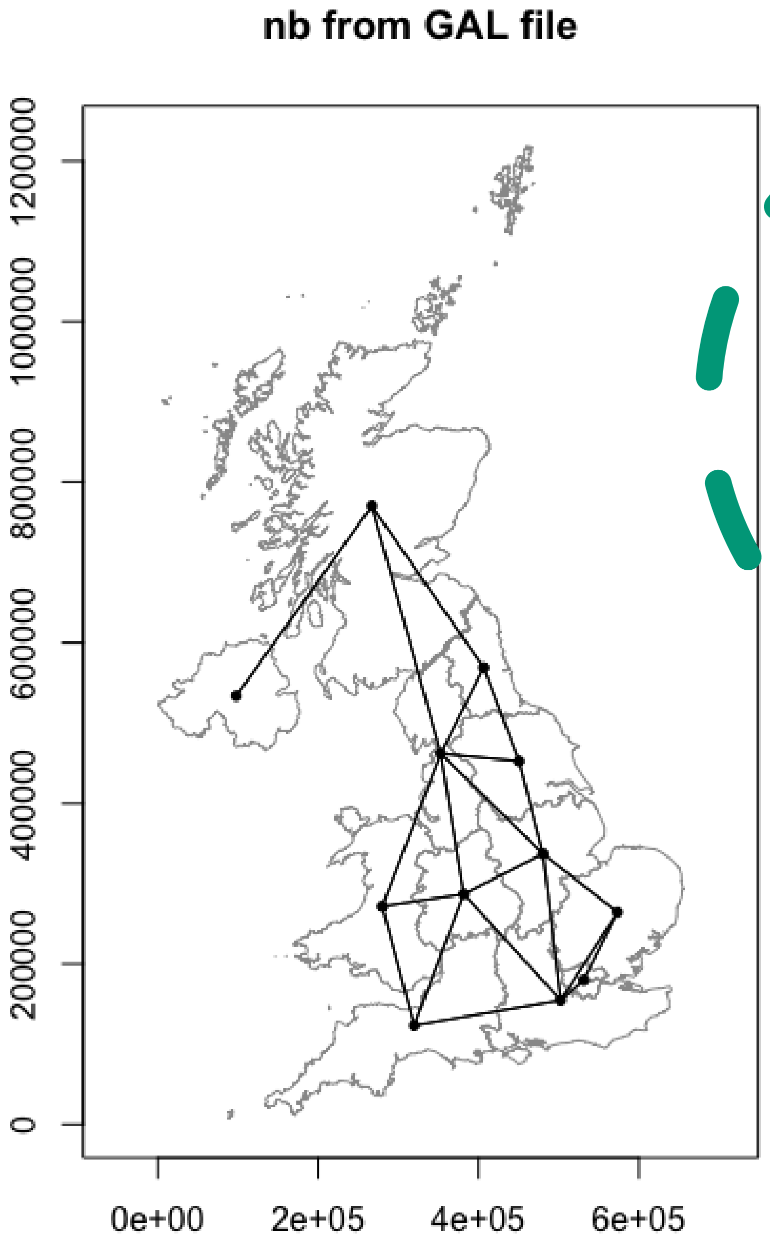
```
> class(UK_nb)
[1] "nb"
```

nb is a neighbour list object which is how neighbourhood information is handled in the R ecosystem.

We also need a shapefile to anchor the external “nb” object to…

```
> (uk_boundaries =
st_read("NUTS_Level_1__January_2018__Boundaries.
shp"))
```

- Make sure the areal units are in the same order as the “nb” object.
- For your convenience, in this case, I have made sure that the object ID of the shapefile and the gal file are the same.



Plot nb

```
> UK_coords = st_centroid(st_geometry(UK_boundaries))

> plot(st_geometry(UK_boundaries), border="grey60", axes=T, reset = F, main="nb from GAL file")

> plot(UK_nb, UK_coords, pch=19, cex=0.6, add=T)
```

Create GAL file from nb

```
> write.nb.gal(UK_nb,
"UK_nb_new.gal", oldstyle=F)
```

oldstyle: if TRUE, first line of file contains only number of spatial units, if FALSE (recommended), uses newer GeoDa style.

Create neighbours based on spatial polygons

We will work with Singapore planning areas

```
> SG = st_read("MySingapura.shp")
```

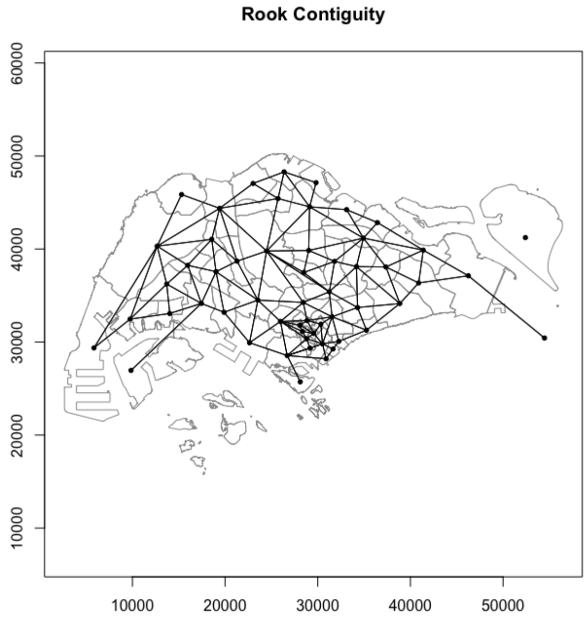
1. Creating contiguity neighbours from spatial polygons

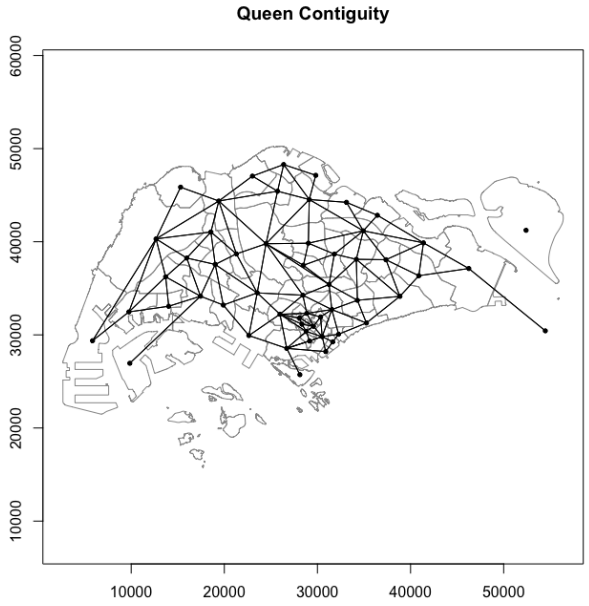
Create **queen contiguity** neighbours from SG.

```
> (SG1_nb = poly2nb(SG))
```

In "poly2nb", queen=T by default, if we make queen=F, then we get the **rook contiguity**.

```
> (SG2_nb = poly2nb(SG, queen=F))
```





Plot…

```
SG_coords =
st_point_on_surface(st_geometry(SG))

plot(st_geometry(SG), border="grey60", axes=T,
main="Queen Contiguity")
plot(SG1_nb, SG_coords, pch=19, cex=0.6, add=T)

plot(st_geometry(SG), border="grey60", axes=T,
main="Rook Contiguity")
plot(SG2_nb, SG_coords, pch=19, cex=0.6, add=T)
```

No difference …

```
> isTRUE(all.equal(SG1_nb, SG2_nb, check.attributes = F))
```

[1] TRUE

For Singapore planning areas queen contiguity is the same as rook contiguity.

2. "k" Nearest neighbours

```
> IDs = row.names(SG)
```

k-nearest neighbours

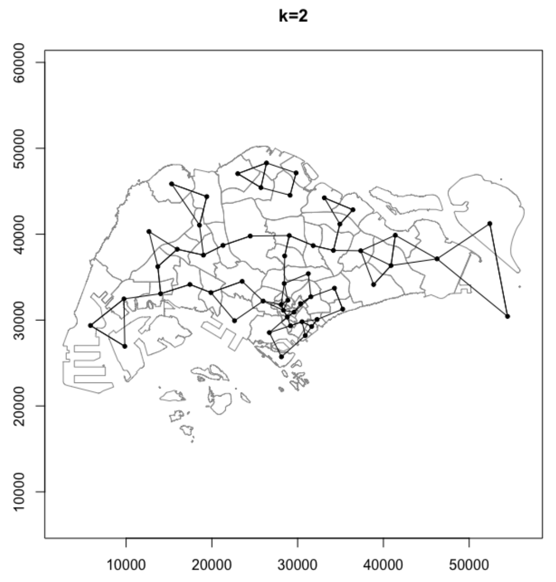
```
SG3_nb =
knn2nb(knearneigh(SG_coords, k =
1), row.names = IDs)
SG4_nb =
knn2nb(knearneigh(SG_coords, k =
2), row.names = IDs)
SG5_nb =
knn2nb(knearneigh(SG_coords, k =
3), row.names = IDs)
```

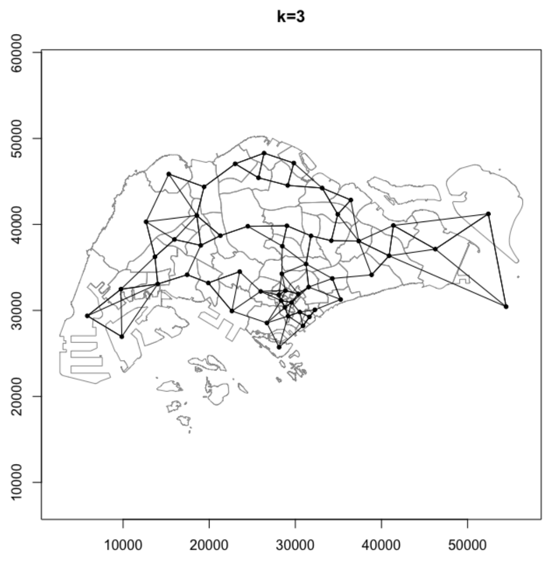
Plot…

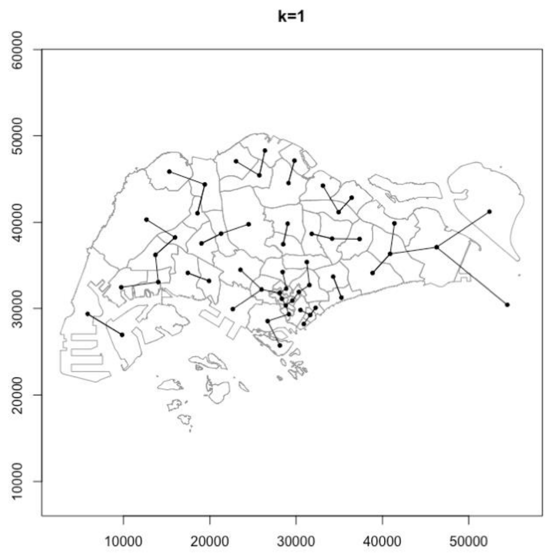
```
plot(st_geometry(SG), border="grey60",
axes=T,  main="k=1")
plot(SG3_nb, SG_coords, pch=19, cex=0.6,
add=T)
```

```
plot(st_geometry(SG), border="grey60",
axes=T,  main="k=2")
plot(SG4_nb, SG_coords, pch=19, cex=0.6,
add=T)
```

```
plot(st_geometry(SG), border="grey60",
axes=T,  main="k=3")
plot(SG5_nb, SG_coords, pch=19, cex=0.6,
add=T)
```


k=2
60000
50000
40000
30000
20000
10000
10000
20000
30000
40000
50000


k=3
60000
50000
40000
30000
20000
10000
10000
20000
30000
40000
50000


k=1
60000
50000
40000
30000
20000
10000
10000
20000
30000
40000
50000

3. Neighbours within a certain metric distance

```
> st_crs(SG)$units
[1] "m"
```

```
SG6_nb = dnearneigh(SG_coords, d1 = 0, d2 = 1000, row.names = IDs)
SG7_nb = dnearneigh(SG_coords, d1 = 0, d2 = 5000, row.names = IDs)
SG8_nb = dnearneigh(SG_coords, d1 = 0, d2 = 10000, row.names = IDs)
```

Note that distance-based neighbours can leave some spatial units neighbour-less and the degree of connectedness can exponentially increase as you increase the distance.

Plot...

```
plot(st_geometry(SG), border="grey60",
axes=T,  main="Within 1km")

plot(SG6_nb, SG_coords, pch=19, cex=0.6,
add=T)


plot(st_geometry(SG), border="grey60",
axes=T,  main="Within 5km")

plot(SG7_nb, SG_coords, pch=19, cex=0.6,
add=T)


plot(st_geometry(SG), border="grey60",
axes=T,  main="Within 10km")

plot(SG8_nb, SG_coords, pch=19, cex=0.6,
add=T)
```


Within 5km
10000
20000
30000
40000
Within 10km
10000
20000
30000
40000
Within 1km
10000
20000
30000
40000
50000

Activity B

- Create nb files based on
  - Contiguity criterion
  - k-nearest neighbours and
  - Distance based (use three appropriate distances)

  Using the UK shapefile
- Create a GAL file using the "nb" object based on queen contiguity criterion. Coerce a relationship between Scotland and Northern Ireland by manipulating the GAL file. Read the GAL file back into your R session and inspect the new "nb" object.

W Matrix

Create a weights list object

- The **nb list** object that we created before contains information on how areal units are related to one another
- It must be **converted to a weights matrix before it can be used in statistical analyses**.
  - See the “uk.xlxs” file
- We can do this by suing the function `nb2listw()`
- This function converts the nb list to a weights list object that contains information about the type of the weights matrix to be used in the analysis.

Example of Weights list object

```
> (UK_lw = nb2listw(UK_nb))
Characteristics of weights list object:
Neighbour list object:
Number of regions: 12
Number of nonzero links: 42
Percentage nonzero weights: 29.16667
Average number of links: 3.5

Weights style: W (default)
Weights constants summary:
   n  nn S0        S1        S2
W 12 144 12 7.751111 50.16889
```

Different weights "styles"

- Style options
  - style="B" is the basic binary coding,
  - style="**W" is row standardised/normalised** (sums over all row links to n),
  - style="C" is globally standardised (sums over all links to n),
  - style="U" is C divided by the number of neighbours (sums over all links to unity).
- Row normalised W matrix
  - Row standardising can turn a symmetric weights matrix into an asymmetric one.
  - There is also the chance of boosting the weightage of the units near the boundary.
  - In many econometric applications it is the norm to use style="W" matrix as it has the property of *sum(W)=n* which is quite useful in numerical optimisations and other computations related to estimation of parameters and test statistics. (See next chapter).

Inspect weights list

```
> names(UK_lw)
[1] "style"      "neighbours" "weights"
> names(attributes(UK_lw))
[1] "names"  "class"  "region.id" "call"   "GeoDa"
> summary(unlist(UK_lw$weights))
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
 0.1667  0.2000  0.2667  0.2857  0.3333  1.0000
```

Zero policy

By default, zero.policy=F, which means there cannot be “loners”.

You can disable this by setting zer.policy=T but in many econometric applications such a W matrix raises more problems than can be answered, for example, the effective sample size, a.k.a degrees of freedom in tests model identification issues, etc.

Zero Policy example

```
> (SG1_lw = nb2listw(SG1_nb))
Error in nb2listw(SG1_nb) : Empty neighbour sets found

> (SG1_lw = nb2listw(SG1_nb, zero.policy=T))
Error in print.listw(x) : regions with no neighbours
found, use zero.policy=TRUE

> print(SG1_lw, zero.policy=T)
```

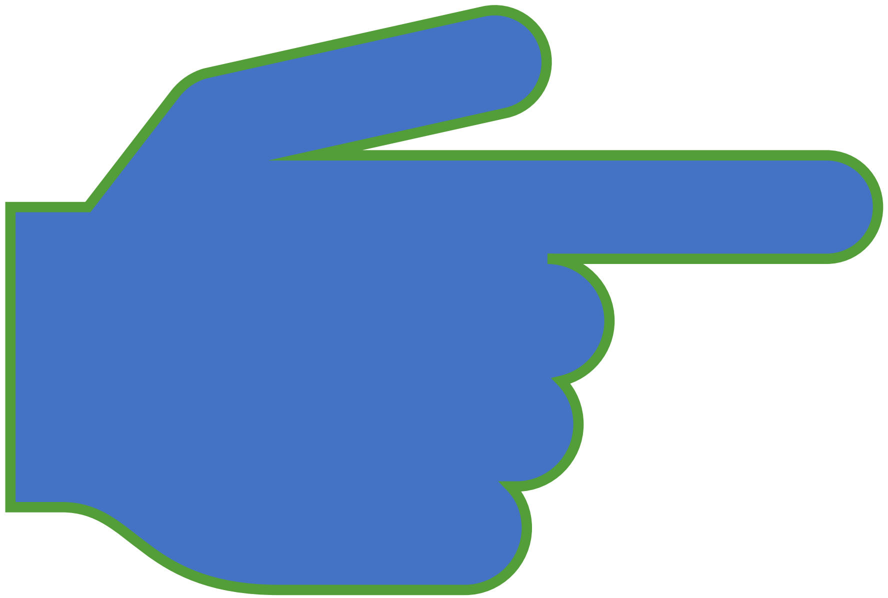
Characteristics of weights list object:
Neighbour list object:
Number of regions: 55
Number of nonzero links: 258
Percentage nonzero weights: 8.528926
Average number of links: 4.690909
1 region with no links: 38

Weights style: W
Weights constants summary:

```
  n   nn S0       S1       S2
W 54 2916 54 26.00677 229.5735
```

Take home points

- Areal data and it's properties
- Neighbours and weights matrix
  - Distance-based
  - Contiguity
- .GAL file: portable weights matrix
- Create neighbours from spatial polygons
- Convert nb list to weights list



Important R functions

- read.gal()
- write.nb.gal()
- poly2nb()
- knearneigh()
- dnearneigh()
- nb2listw()

References

- ***Spatial Analysis*** by Tonny Oyana 2nd edition, Chapter 7.
- ***Applied Spatial Data Analysis with R*** by Roger S. Bivand, Edzer Pebesma, and Virgilio Gómez-Rubio, 2nd edition, (2013), Chapter 9.
- `> vignette(package="spdep")`
- https://r-spatial.github.io/spdep/articles/nb_sf.html