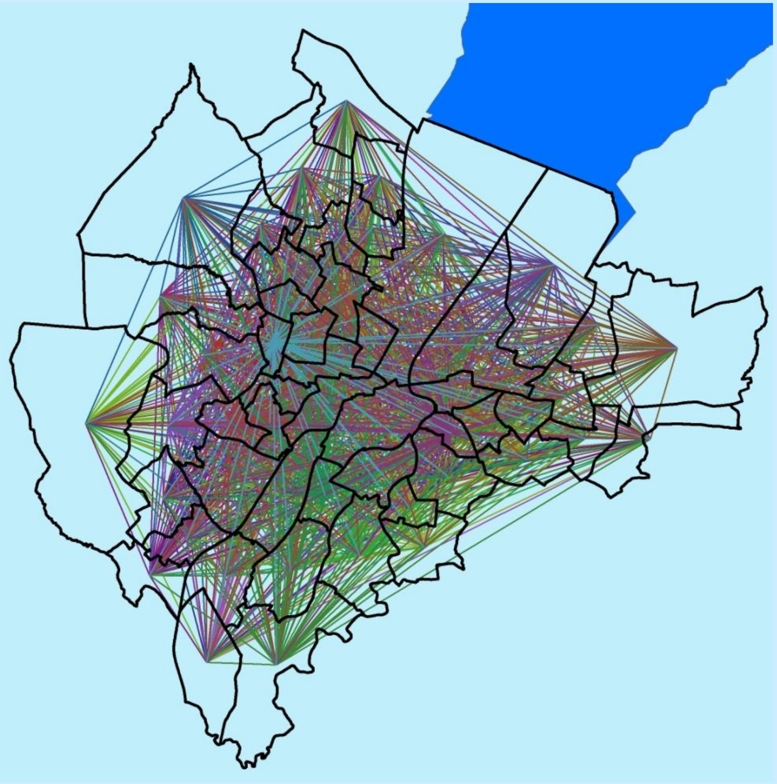
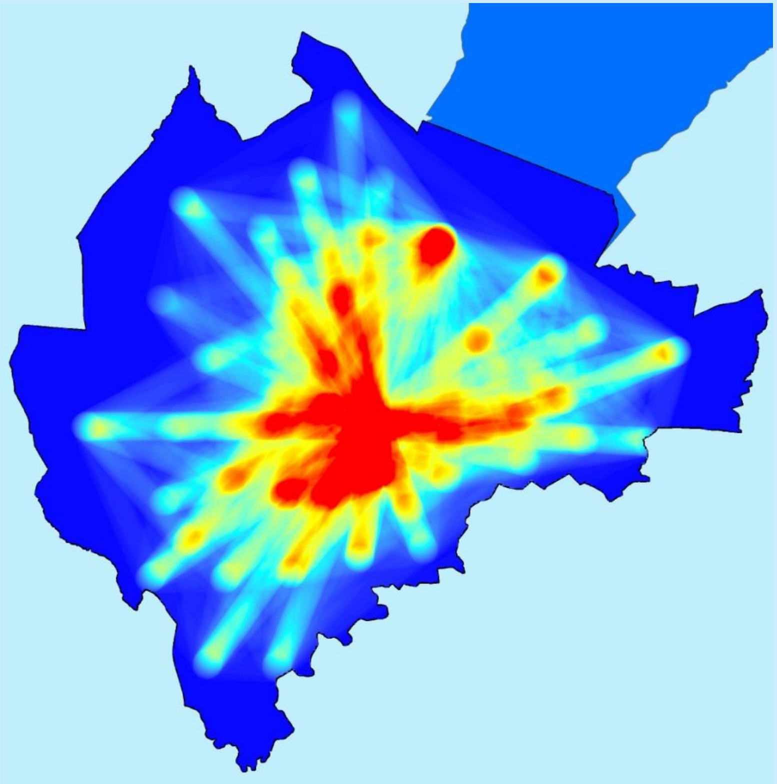
Open Geospatial Data Science for Modelling Commuter Movements and Demographics



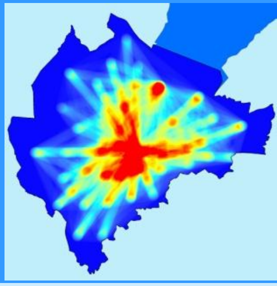


Lorraine Barry

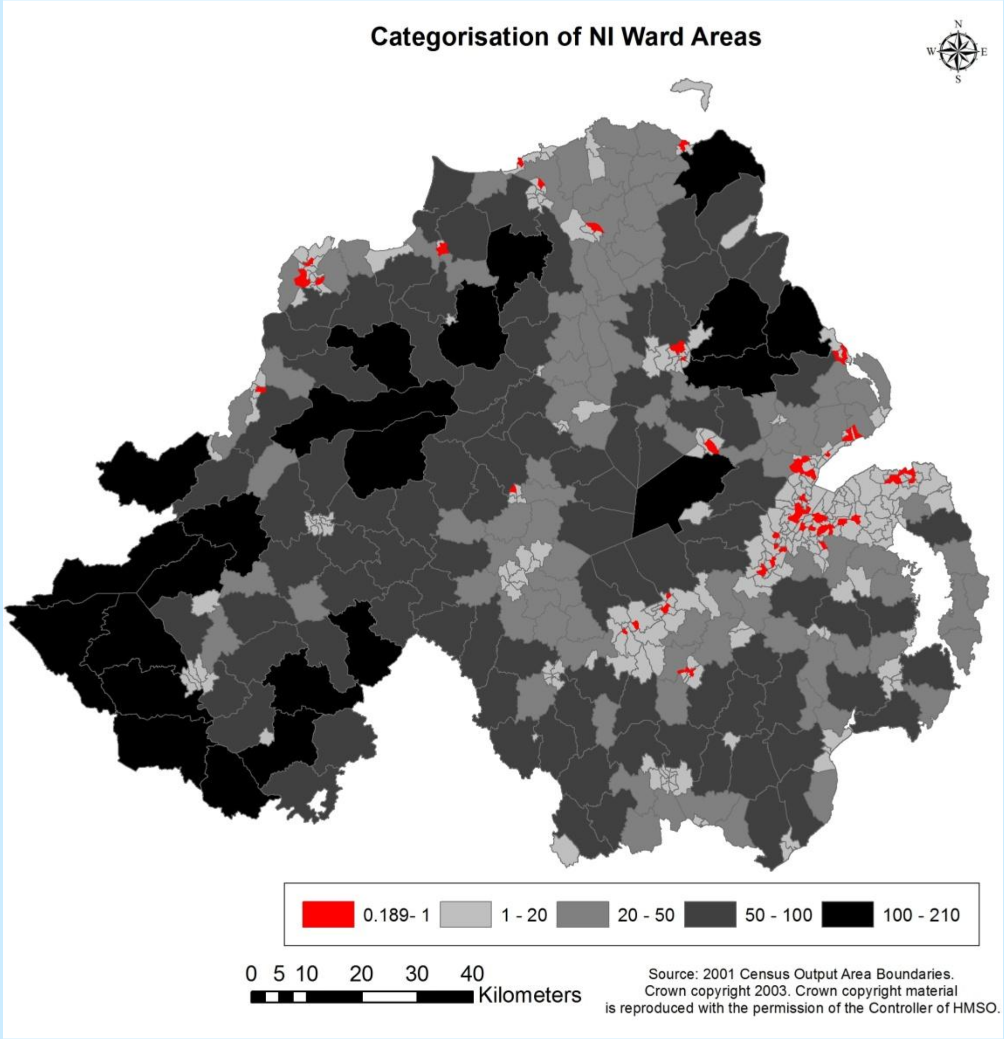
Centre for GIS and Geomatics, School of Natural and Built Environment,
Queen's University Belfast   l.barry@qub.ac.uk

Presentation at OSGeo Ireland Symposium 2017

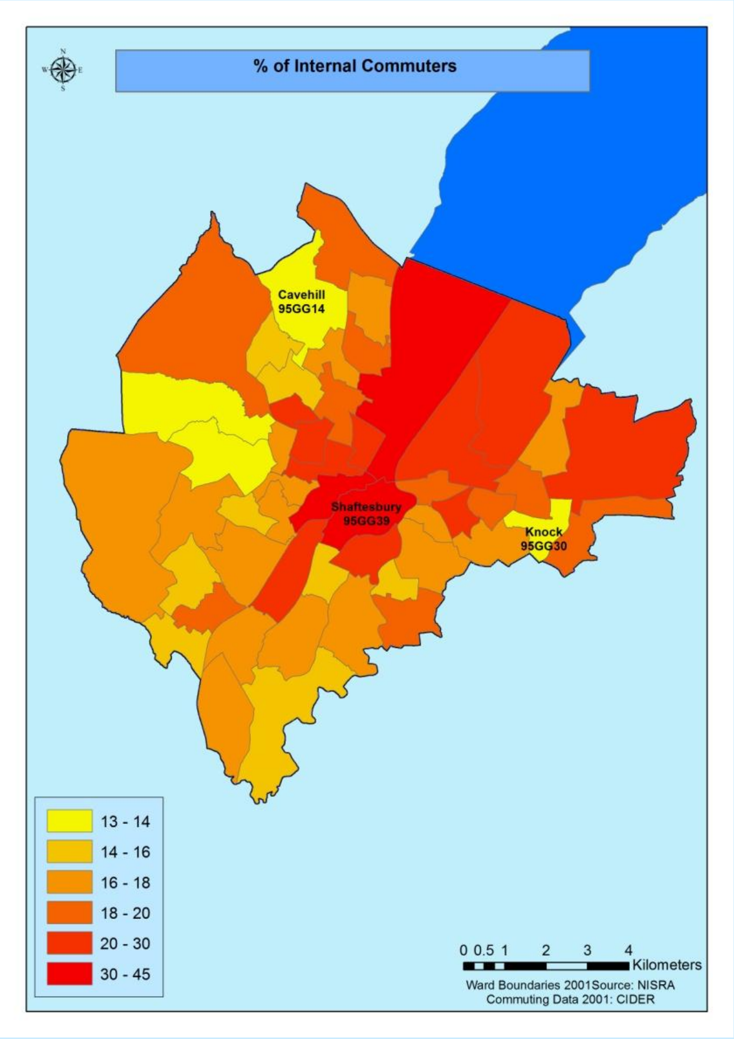
Modelling Commuter Movements and Demographics: A Northern Ireland Case Study



Northern Ireland Wards – Variation in Size and Internal Flows

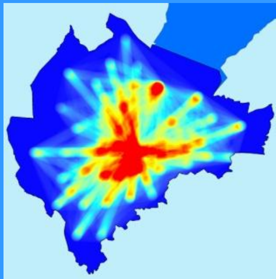


Source: 2001 Census Output Area Boundaries.
Crown copyright 2003. Crown copyright material
is reproduced with the permission of the Controller of HMSO.



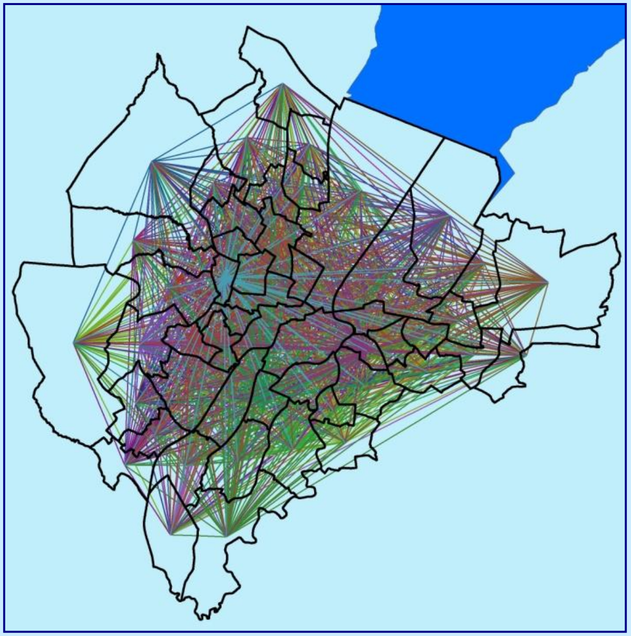
Ward Boundaries 2001Source: NISRA
Commuting Data 2001: CIDER

Modelling Commuter Movements and Demographics: A Northern Ireland Case Study



Are patterns of commuter movement influenced by distance or employment opportunities only?

Or are gender, religion, age demographics influencing patterns?



Objectives:

To create functional regions for interaction data

To investigate population variables which influence patterns of movement

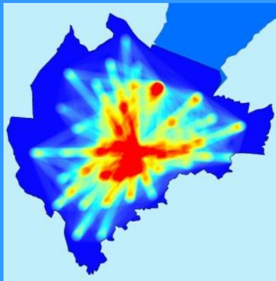
To investigate interactions at both regional and local scales

To efficiently display flows and patterns

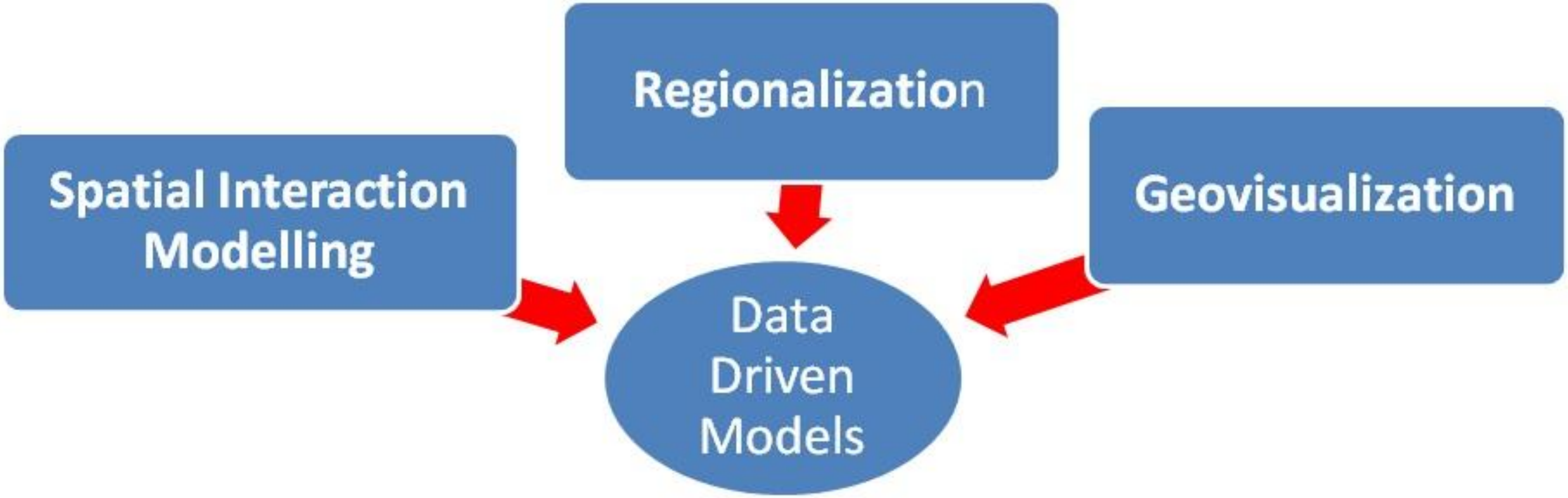
To demonstrate applicability to policy and industry

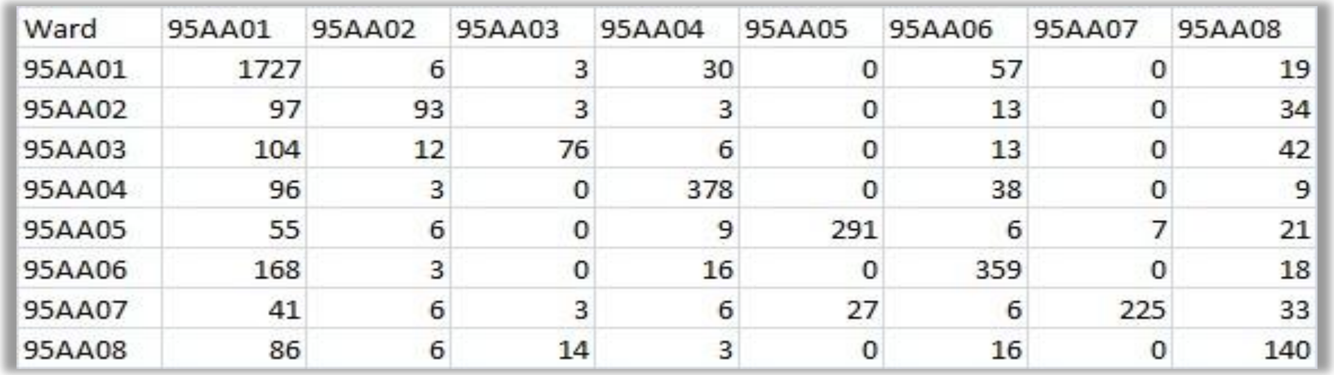
Idea of Optimal Zoning? Geography of consistency

Modelling Commuter Movements and Demographics: A Northern Ireland Case Study



Goal to obtain geography of consistent internal flows for modelling

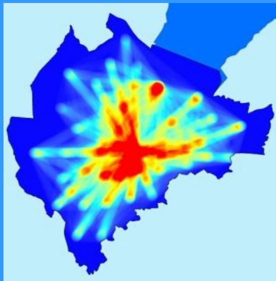




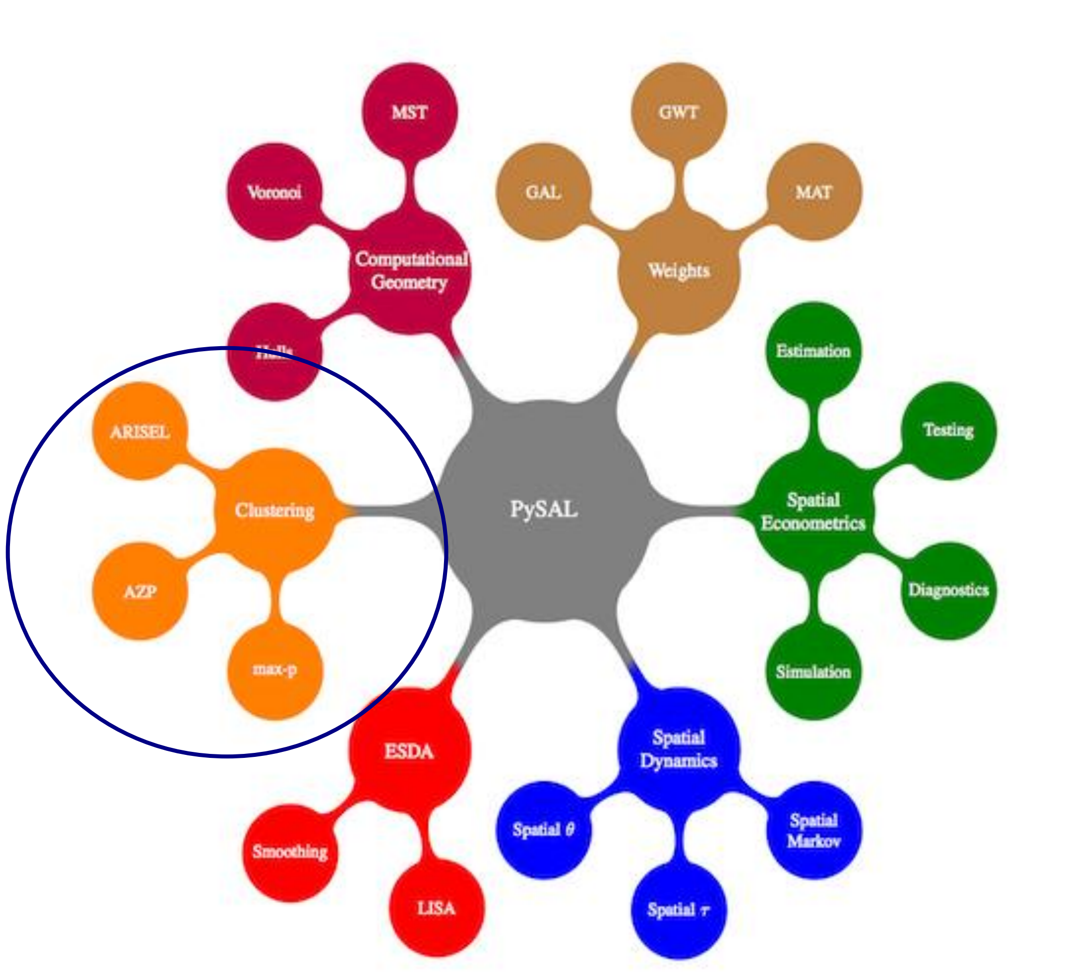
Ward	95AA01	95AA02	95AA03	95AA04	95AA05	95AA06	95AA07	95AA08
95AA01	1727	6	3	30	0	57	0	19
95AA02	97	93	3	3	0	13	0	34
95AA03	104	12	76	6	0	13	0	42
95AA04	96	3	0	378	0	38	0	9
95AA05	55	6	0	9	291	6	7	21
95AA06	168	3	0	16	0	359	0	18
95AA07	41	6	3	6	27	6	225	33
95AA08	86	6	14	3	0	16	0	140

1991
2001
2011
2021

Modelling Commuter Movements and Demographics: Python and PySAL



Regionalisation



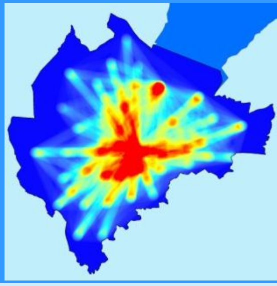
Source: Rey & Anselin, 2007

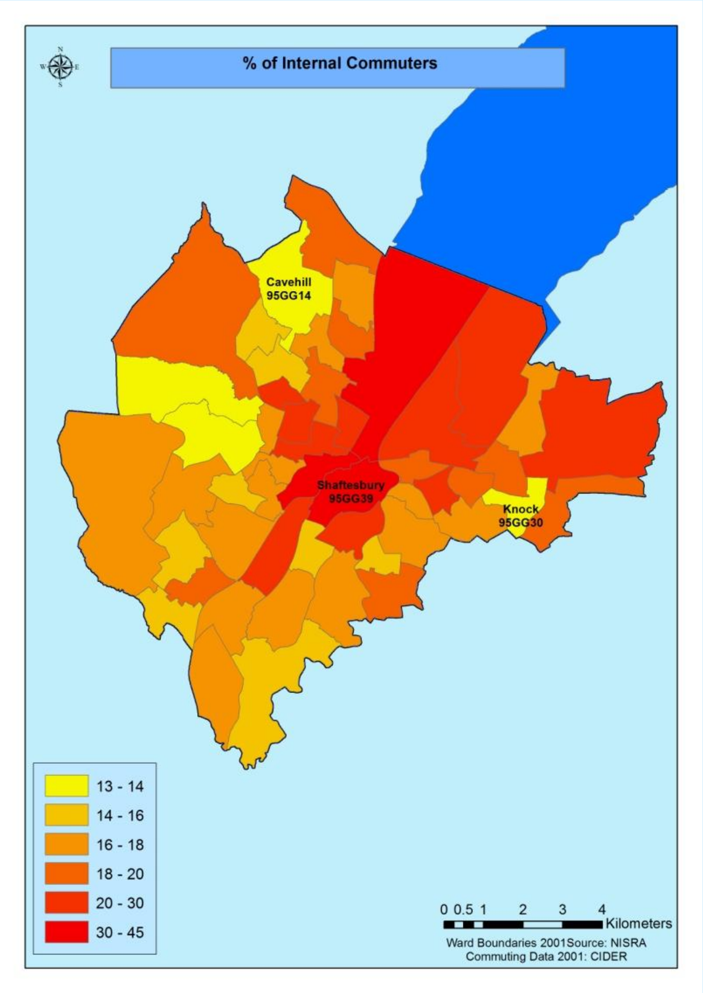
python

•Easily to code
•Object Orientated language
•Combine with database systems
•Easily work with other python packages – numpy, scipy

MaxpTabu:
•Maximum number of regions
•platform independent
•Modification is possible
•Creates maximum number of regions

Modelling Commuter Movements and Demographics: Regionalisation





Ward Boundaries 2001Source: NISRA
Commuting Data 2001: CIDER

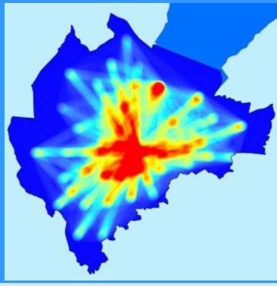
Regionalisation

Regionalisation is the clustering or grouping of spatial units into spatially contiguous regions whilst maximising a particular objective function (Guo, 2009).

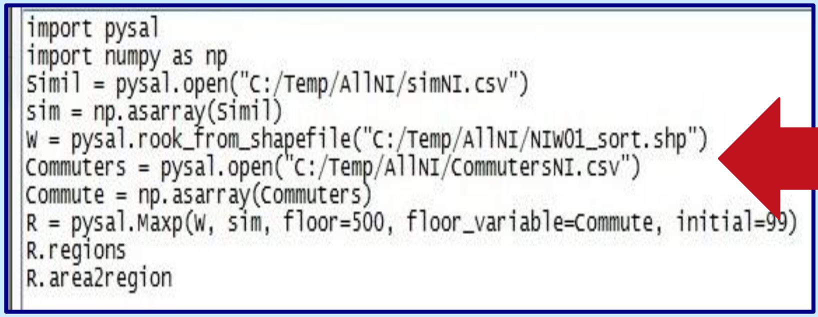
- •Creation of purpose-specific zones
- •Zones of consistent internal flows
- •Generalisations from voluminous datasets
- •Diminish effects of irregular zones

Termed: districting, redistricting, zonation, zone design systems, functional regions, functional regionalization and spatial clustering
(Duque et al 2011, Alvanides et al 2000, Konjar et al 2010, Koo 2010, Srinivas et al 2011).

Modelling Commuter Movements and Demographics: Regionalisation

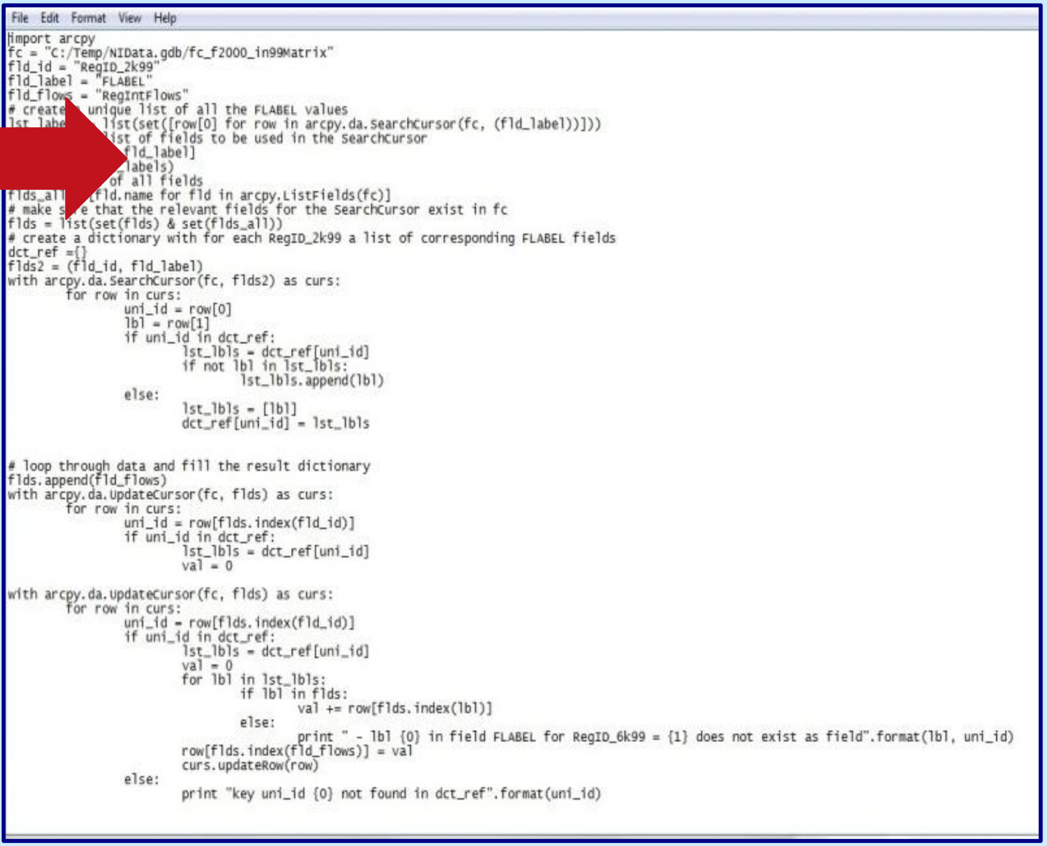


Region Building Code



```
import pysal
import numpy as np
Simil = pysal.open("C:/Temp/AllNI/simNI.csv")
sim = np.asarray(Simil)
W = pysal.rook_from_shapefile("C:/Temp/AllNI/NIW01_sort.shp")
Commuters = pysal.open("C:/Temp/AllNI/CommutersNI.csv")
Commute = np.asarray(Commuters)
R = pysal.Maxp(W, sim, floor=500, floor_variable=Commute, initial=99)
R.regions
R.area2region
```

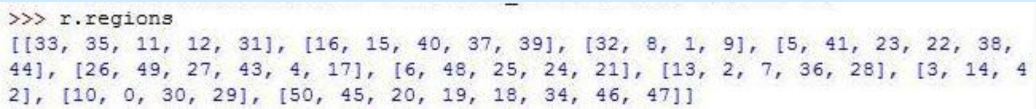
Region Internal Flow Calculation Code



```
import arcpy
fc = "C:/Temp/NIData.gdb/fc_f2000_in99Matrix"
fld_id = "RegID_2k99"
fld_label = "FLABEL"
fld_flows = "RegIntFlows"
# create   unique list of all the FLABEL values
lst_labe    list(set([row[0] for row in arcpy.da.SearchCursor(fc, (fld_label))]))
              ist of fields to be used in the SearchCursor
                fld_label]
                labels)
              of all fields
flds_all    [fld.name for fld in arcpy.ListFields(fc)]
# make s  e that the relevant fields for the SearchCursor exist in fc
flds = list(set(flds) & set(flds_all))
# create a dictionary with for each RegID_2k99 a list of corresponding FLABEL fields
dct_ref ={}
flds2 = (fld_id, fld_label)
with arcpy.da.SearchCursor(fc, flds2) as curs:
        for row in curs:
                uni_id = row[0]
                lbl = row[1]
                if uni_id in dct_ref:
                        lst_lbls = dct_ref[uni_id]
                        if not lbl in lst_lbls:
                                lst_lbls.append(lbl)
                else:
                        lst_lbls = [lbl]
                        dct_ref[uni_id] = lst_lbls


# loop through data and fill the result dictionary
flds.append(fld_flows)
with arcpy.da.UpdateCursor(fc, flds) as curs:
        for row in curs:
                uni_id = row[flds.index(fld_id)]
                if uni_id in dct_ref:
                        lst_lbls = dct_ref[uni_id]
                        val = 0

with arcpy.da.UpdateCursor(fc, flds) as curs:
        for row in curs:
                uni_id = row[flds.index(fld_id)]
                if uni_id in dct_ref:
                        lst_lbls = dct_ref[uni_id]
                        val = 0
                        for lbl in lst_lbls:
                                if lbl in flds:
                                        val += row[flds.index(lbl)]
                                else:
                                        print " - lbl {0} in field FLABEL for RegID_6k99 = {1} does not exist as field".format(lbl, uni_id)
                        row[flds.index(fld_flows)] = val
                        curs.updateRow(row)
                else:
                        print "key uni_id {0} not found in dct_ref".format(uni_id)
```

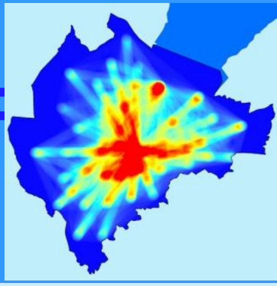
Integration of code through a WHILE LOOP

- **JOIN** region output to feature class
- **SAVE** join
- **DISSOLVE** wards based on regions
- **CALCULATE** % internal flows



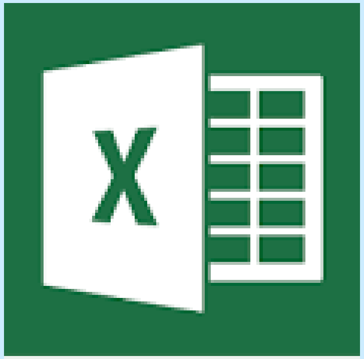
```
>>> r.regions
[[33, 35, 11, 12, 31], [16, 15, 40, 37, 39], [32, 8, 1, 9], [5, 41, 23, 22, 38,
44], [26, 49, 27, 43, 4, 17], [6, 48, 25, 24, 21], [13, 2, 7, 36, 28], [3, 14, 4
2], [10, 0, 30, 29], [50, 45, 20, 19, 18, 34, 46, 47]]
```

Modelling Commuter Movements and Demographics :
Why Open Source?



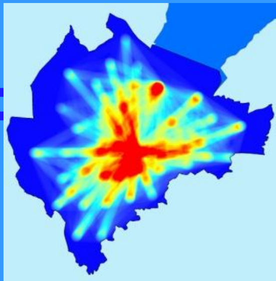
Shapefile, ESRI Geodatabase, ArcPy

Open Python and Data Formats



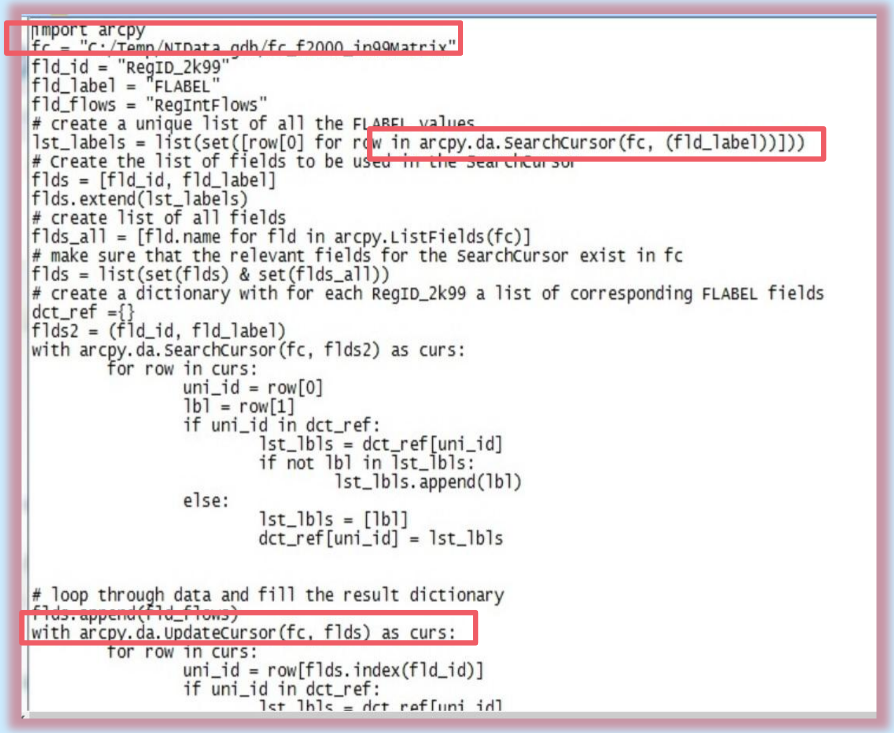
Flexible
Interactive
Open
Customisable
Powerful

Modelling Commuter Movements and Demographics :
Why Open Source?

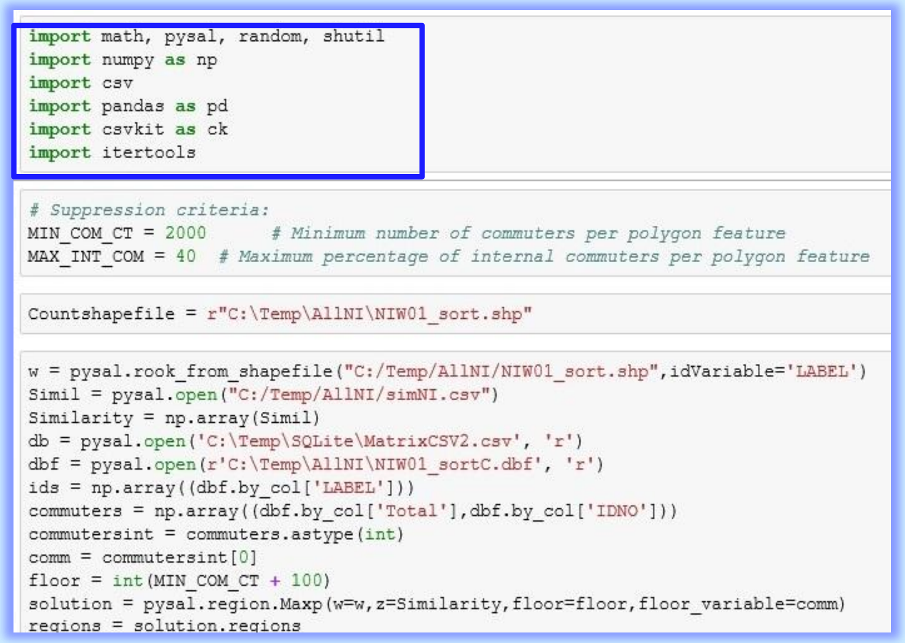


Shapefile, ESRI Geodatabase, ArcPy

Open Python and Data Formats



```
import arcpy
fc = "C:/Temp/NIData.gdb/fc_f2000_in99Matrix"
fld_id = "RegID_2k99"
fld_label = "FLABEL"
fld_flows = "RegIntFlows"
# create a unique list of all the FLABEL values
lst_labels = list(set([row[0] for row in arcpy.da.SearchCursor(fc, (fld_label))]))
# Create the list of fields to be used in the SearchCursor
flds = [fld_id, fld_label]
flds.extend(lst_labels)
# create list of all fields
flds_all = [fld.name for fld in arcpy.ListFields(fc)]
# make sure that the relevant fields for the SearchCursor exist in fc
flds = list(set(flds) & set(flds_all))
# create a dictionary with for each RegID_2k99 a list of corresponding FLABEL fields
dct_ref ={}
flds2 = (fld_id, fld_label)
with arcpy.da.SearchCursor(fc, flds2) as curs:
        for row in curs:
                uni_id = row[0]
                lbl = row[1]
                if uni_id in dct_ref:
                        lst_lbls = dct_ref[uni_id]
                        if not lbl in lst_lbls:
                                lst_lbls.append(lbl)
                else:
                        lst_lbls = [lbl]
                        dct_ref[uni_id] = lst_lbls


# loop through data and fill the result dictionary
flds.append(fld_flows)
with arcpy.da.UpdateCursor(fc, flds) as curs:
        for row in curs:
                uni_id = row[flds.index(fld_id)]
                if uni_id in dct_ref:
                        lst_lbls = dct_ref[uni_id]
```



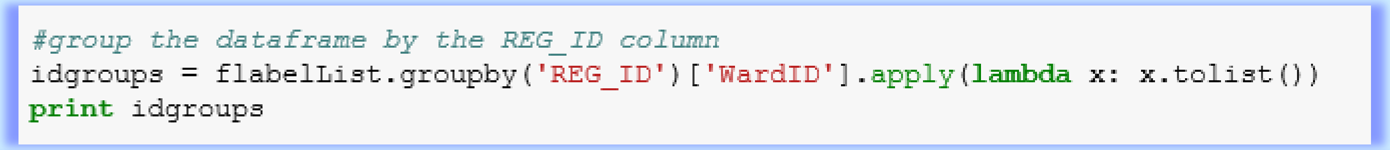
```
import math, pysal, random, shutil
import numpy as np
import csv
import pandas as pd
import csvkit as ck
import itertools
```

```
# Suppression criteria:
MIN_COM_CT = 2000       # Minimum number of commuters per polygon feature
MAX_INT_COM = 40  # Maximum percentage of internal commuters per polygon feature
```

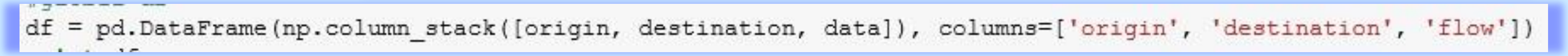
```
Countshapefile = r"C:\Temp\AllNI\NIW01_sort.shp"
```

```
w = pysal.rook_from_shapefile("C:/Temp/AllNI/NIW01_sort.shp",idVariable='LABEL')
Simil = pysal.open("C:/Temp/AllNI/simNI.csv")
Similarity = np.array(Simil)
db = pysal.open('C:\Temp\SQLite\MatrixCSV2.csv', 'r')
dbf = pysal.open(r'C:\Temp\AllNI\NIW01_sortC.dbf', 'r')
ids = np.array((dbf.by_col['LABEL']))
commuters = np.array((dbf.by_col['Total'],dbf.by_col['IDNO']))
commutersint = commuters.astype(int)
comm = commutersint[0]
floor = int(MIN_COM_CT + 100)
solution = pysal.region.Maxp(w=w,z=Similarity,floor=floor,floor_variable=comm)
regions = solution.regions
```

Open Data Formats for Grouping and indexing

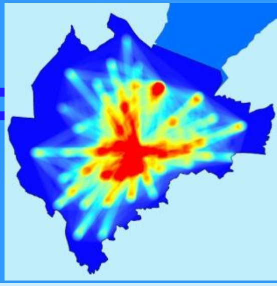


```
#group the dataframe by the REG_ID column
idgroups = flabelList.groupby('REG_ID')['WardID'].apply(lambda x: x.tolist())
print idgroups
```



```
df = pd.DataFrame(np.column_stack([origin, destination, data]), columns=['origin', 'destination', 'flow'])
```

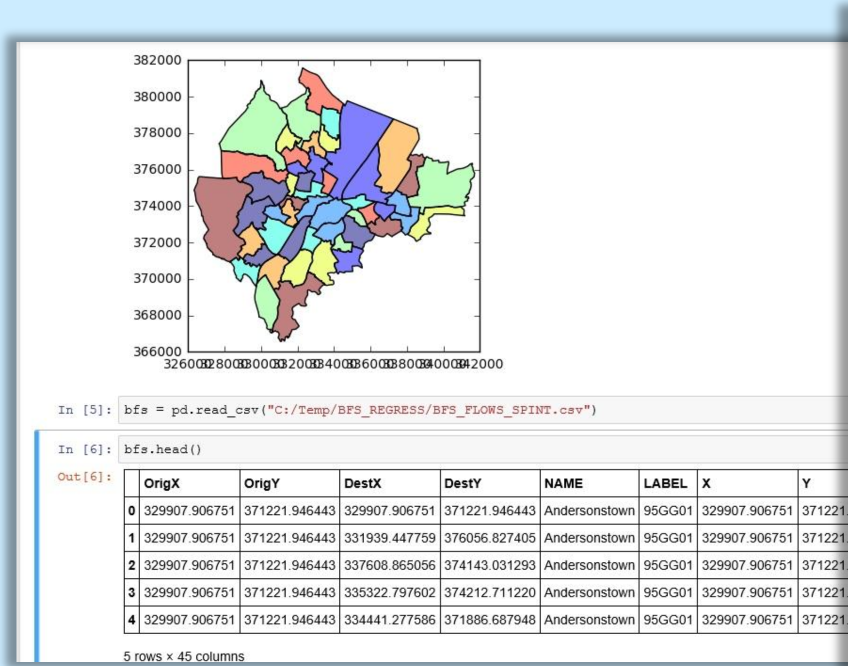
Modelling Commuter Movements and Demographics : SIM



Spatial Interaction Modelling:

a technique to evaluate the patterns between volume of flows and the underlying socio-economic tendencies of the origin and destination zones.

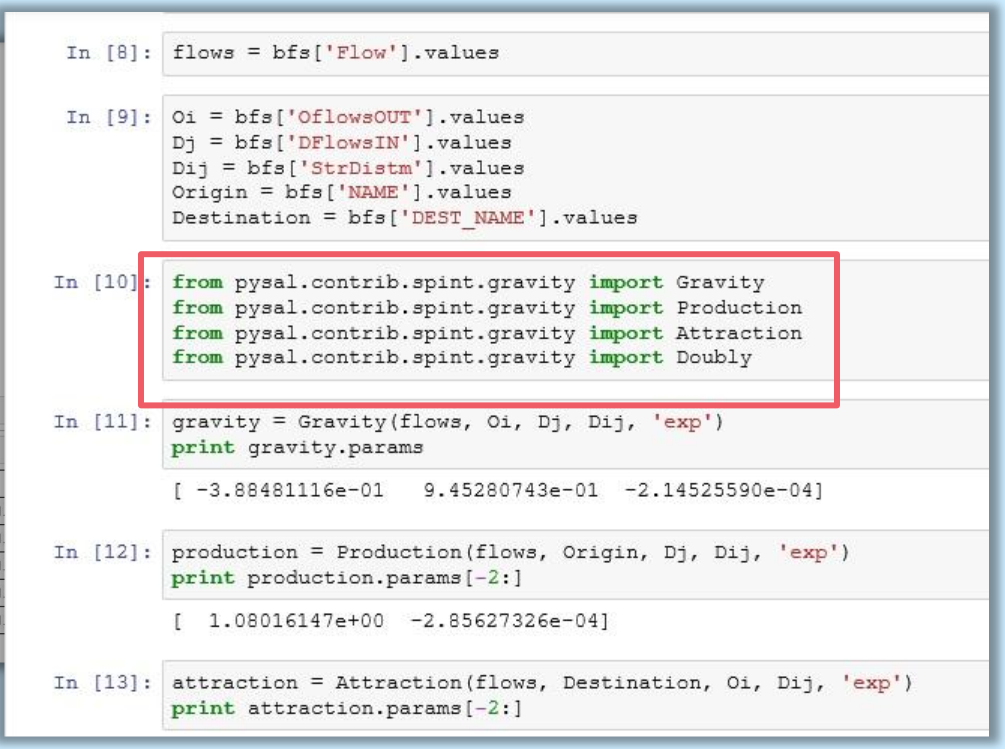
(Lloyd et al, 2011)



```
In [5]: bfs = pd.read_csv("C:/Temp/BFS_REGRESS/BFS_FLOWS_SPINT.csv")

In [6]: bfs.head()

Out[6]:
```

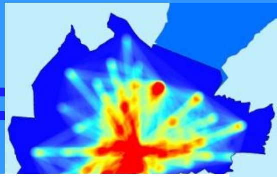
	OrigX	OrigY	DestX	DestY	NAME	LABEL	X	Y
0	329907.906751	371221.946443	329907.906751	371221.946443	Andersonstown	95GG01	329907.906751	371221.
1	329907.906751	371221.946443	331939.447759	376056.827405	Andersonstown	95GG01	329907.906751	371221.
2	329907.906751	371221.946443	337608.865056	374143.031293	Andersonstown	95GG01	329907.906751	371221.
3	329907.906751	371221.946443	335322.797602	374212.711220	Andersonstown	95GG01	329907.906751	371221.
4	329907.906751	371221.946443	334441.277586	371886.687948	Andersonstown	95GG01	329907.906751	371221.

5 rows × 45 columns



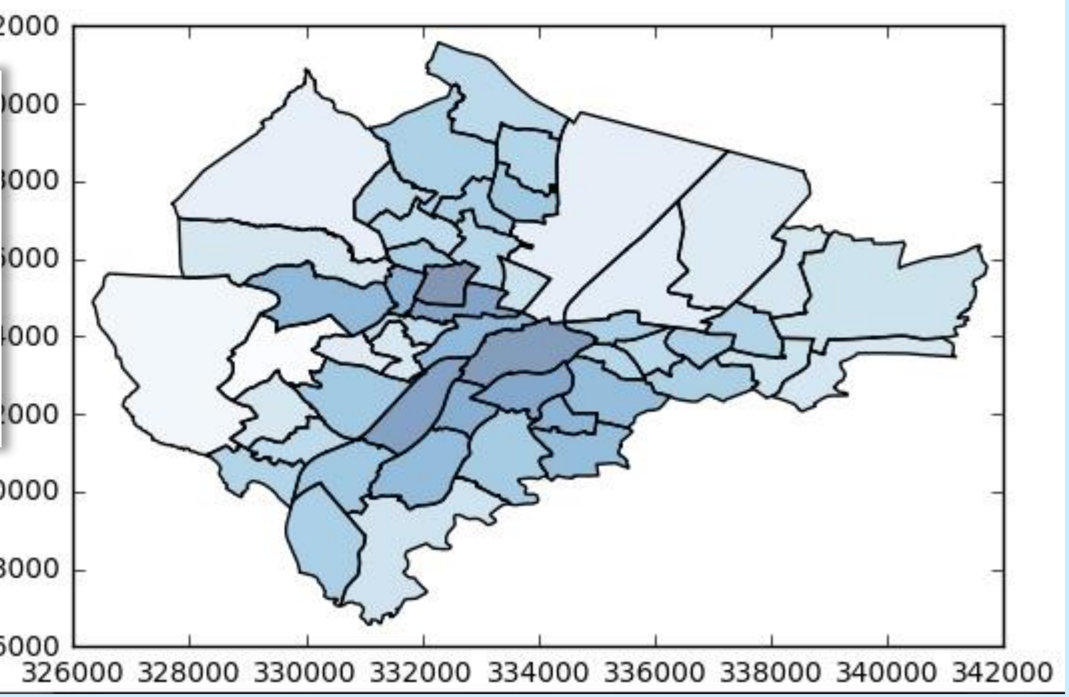
```
In [8]:  flows = bfs['Flow'].values

In [9]:  Oi = bfs['OflowsOUT'].values
         Dj = bfs['DFlowsIN'].values
         Dij = bfs['StrDistm'].values
         Origin = bfs['NAME'].values
         Destination = bfs['DEST_NAME'].values

In [10]: from pysal.contrib.spint.gravity import Gravity
         from pysal.contrib.spint.gravity import Production
         from pysal.contrib.spint.gravity import Attraction
         from pysal.contrib.spint.gravity import Doubly

In [11]: gravity = Gravity(flows, Oi, Dj, Dij, 'exp')
         print gravity.params

         [ -3.88481116e-01   9.45280743e-01  -2.14525590e-04]

In [12]: production = Production(flows, Origin, Dj, Dij, 'exp')
         print production.params[-2:]

         [  1.08016147e+00  -2.85627326e-04]

In [13]: attraction = Attraction(flows, Destination, Oi, Dij, 'exp')
         print attraction.params[-2:]
```

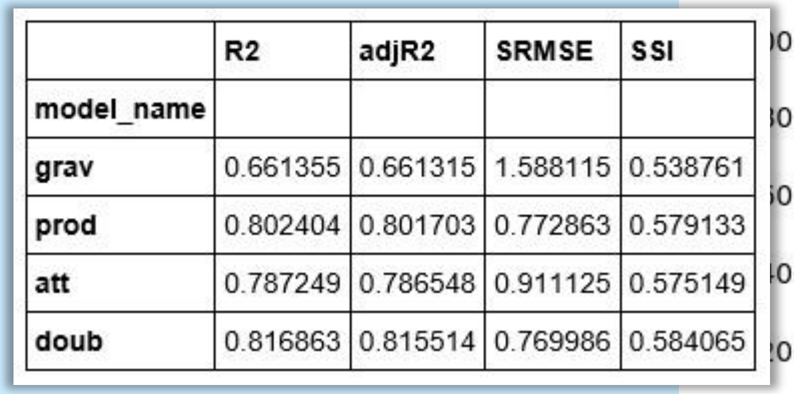
Modelling Commuter Movements and Demographics : SIM



```
local_vals = pd.DataFrame({'betas': local_gravity['param2'],
                           'Dest':np.unique(Origin),
                           'pseudoR2': local_gravity['pseudoR2']})
local_vals = pd.merge(local_vals, bfs_shp[['NAME', 'geometry']],
                      left_on='Dest', right_on='NAME')
local_vals = gp.GeoDataFrame(local_vals)


#plot betas - use inverse so the most negative values are "higher"
fig = plt.figure()
ax = fig.add_subplot(111)
local_vals.plot('betas', cmap='Blues', ax=ax)
```

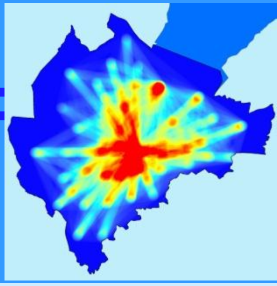
```
<matplotlib.axes._subplots.AxesSubplot at 0x266a82b0>
```



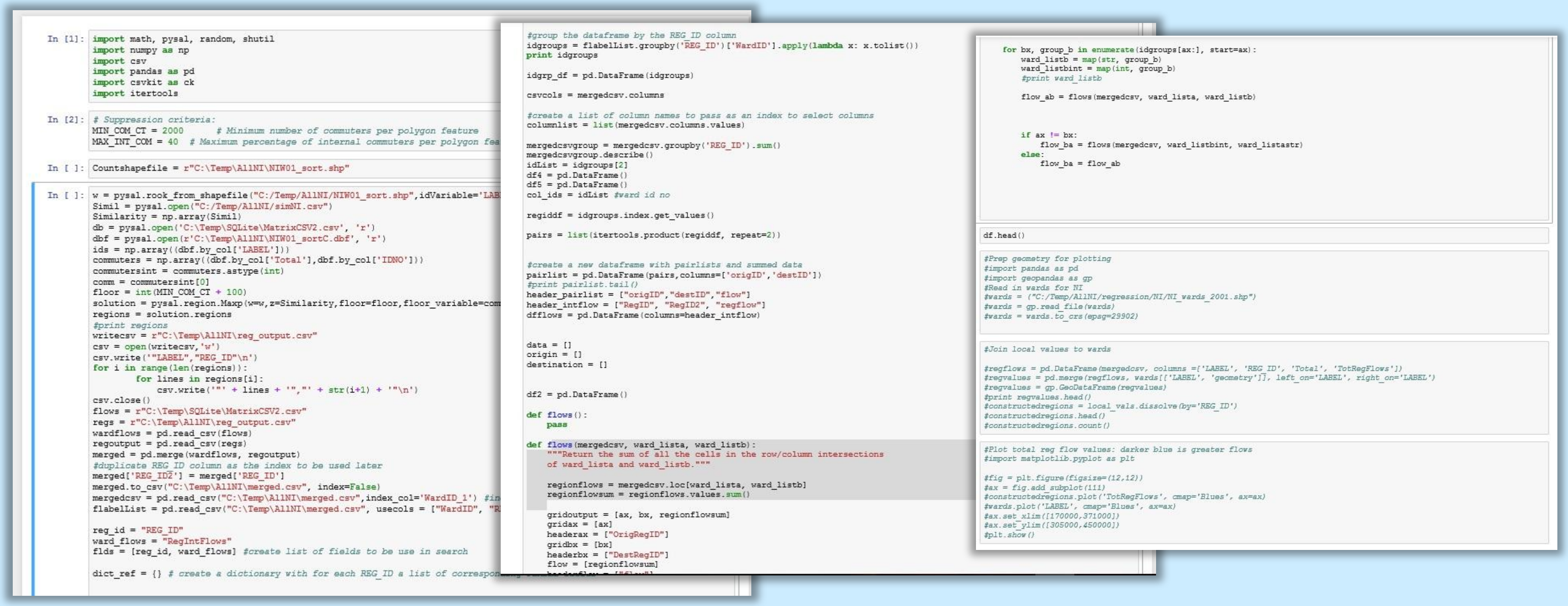


	R2	adjR2	SRMSE	SSI
model_name				
grav	0.661355	0.661315	1.588115	0.538761
prod	0.802404	0.801703	0.772863	0.579133
att	0.787249	0.786548	0.911125	0.575149
doub	0.816863	0.815514	0.769986	0.584065

Modelling Commuter Movements and Demographics : SIM



Regionalization and Spatial Interaction Modelling:



```
In [1]: import math, pysal, random, shutil
        import numpy as np
        import csv
        import pandas as pd
        import csvkit as ck
        import itertools

In [2]: # Suppression criteria:
        MIN_COM_CT = 2000      # Minimum number of commuters per polygon feature
        MAX_INT_COM = 40  # Maximum percentage of internal commuters per polygon fea

In [ ]: Countshapefile = r"C:\Temp\AllNI\NIW01_sort.shp"

In [ ]: w = pysal.rook_from_shapefile("C:/Temp/AllNI/NIW01_sort.shp",idVariable='LAB
        Simil = pysal.open("C:/Temp/AllNI/simNI.csv")
        Similarity = np.array(Simil)
        db = pysal.open('C:\Temp\SQLite\MatrixCSV2.csv', 'r')
        dbf = pysal.open(r'C:\Temp\AllNI\NIW01_sortC.dbf', 'r')
        ids = np.array((dbf.by_col['LABEL']))
        commuters = np.array((dbf.by_col['Total'],dbf.by_col['IDNO']))
        commutersint = commuters.astype(int)
        comm = commutersint[0]
        floor = int(MIN_COM_CT + 100)
        solution = pysal.region.Maxp(w=w,z=Similarity,floor=floor,floor_variable=com
        regions = solution.regions
        #print regions
        writecsv = r"C:\Temp\AllNI\reg_output.csv"
        csv = open(writecsv,'w')
        csv.write('"LABEL","REG_ID"\n')
        for i in range(len(regions)):
                for lines in regions[i]:
                    csv.write('"' + lines + '","' + str(i+1) + '"\n')
        csv.close()
        flows = r"C:\Temp\SQLite\MatrixCSV2.csv"
        regs = r"C:\Temp\AllNI\reg_output.csv"
        wardflows = pd.read_csv(flows)
        regoutput = pd.read_csv(regs)
        merged = pd.merge(wardflows, regoutput)
        #duplicate REG_ID column as the index to be used later
        merged['REG_ID2'] = merged['REG_ID']
        merged.to_csv("C:\Temp\AllNI\merged.csv", index=False)
        mergedcsv = pd.read_csv("C:\Temp\AllNI\merged.csv",index_col='WardID_1') #in
        flabelList = pd.read_csv("C:\Temp\AllNI\merged.csv", usecols = ["WardID", "R

        reg_id = "REG_ID"
        ward_flows = "RegIntFlows"
        flds = [reg_id, ward_flows] #create list of fields to be use in search

        dict_ref = {} # create a dictionary with for each REG_ID a list of correspon
```

```
#group the dataframe by the REG_ID column
idgroups = flabelList.groupby('REG_ID')['WardID'].apply(lambda x: x.tolist())
print idgroups

idgrp_df = pd.DataFrame(idgroups)

csvcols = mergedcsv.columns

#create a list of column names to pass as an index to select columns
columnlist = list(mergedcsv.columns.values)

mergedcsvgroup = mergedcsv.groupby('REG_ID').sum()
mergedcsvgroup.describe()
idList = idgroups[2]
df4 = pd.DataFrame()
df5 = pd.DataFrame()
col_ids = idList #ward id no

regiddf = idgroups.index.get_values()

pairs = list(itertools.product(regiddf, repeat=2))


#create a new dataframe with pairlists and summed data
pairlist = pd.DataFrame(pairs,columns=['origID','destID'])
#print pairlist.tail()
header_pairlist = ["origID","destID","flow"]
header_intflow = ["RegID", "RegID2", "regflow"]
dfflows = pd.DataFrame(columns=header_intflow)


data = []
origin = []
destination = []


df2 = pd.DataFrame()

def flows():
    pass

def flows(mergedcsv, ward_lista, ward_listb):
    """Return the sum of all the cells in the row/column intersections
    of ward_lista and ward_listb."""

    regionflows = mergedcsv.loc[ward_lista, ward_listb]
    regionflowsum = regionflows.values.sum()

    gridoutput = [ax, bx, regionflowsum]
    gridax = [ax]
    headerax = ["OrigRegID"]
    gridbx = [bx]
    headerbx = ["DestRegID"]
    flow = [regionflowsum]
    headerflow = ["flow"]
```

```
    for bx, group_b in enumerate(idgroups[ax:], start=ax):
        ward_listb = map(str, group_b)
        ward_listbint = map(int, group_b)
        #print ward_listb

        flow_ab = flows(mergedcsv, ward_lista, ward_listb)



        if ax != bx:
            flow_ba = flows(mergedcsv, ward_listbint, ward_listastr)
        else:
            flow_ba = flow_ab
```

```
df.head()
```

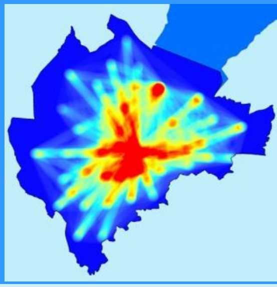
```
#Prep geometry for plotting
#import pandas as pd
#import geopandas as gp
#Read in wards for NI
#wards = ("C:/Temp/AllNI/regression/NI/NI_wards_2001.shp")
#wards = gp.read_file(wards)
#wards = wards.to_crs(epsg=29902)
```

```
#Join local values to wards

#regflows = pd.DataFrame(mergedcsv, columns =['LABEL', 'REG_ID', 'Total', 'TotRegFlows'])
#regvalues = pd.merge(regflows, wards[['LABEL', 'geometry']], left_on='LABEL', right_on='LABEL')
#regvalues = gp.GeoDataFrame(regvalues)
#print regvalues.head()
#constructedregions = local_vals.dissolve(by='REG_ID')
#constructedregions.head()
#constructedregions.count()
```

```
#Plot total reg flow values: darker blue is greater flows
#import matplotlib.pyplot as plt

#fig = plt.figure(figsize=(12,12))
#ax = fig.add_subplot(111)
#constructedregions.plot('TotRegFlows', cmap='Blues', ax=ax)
#wards.plot('LABEL', cmap='Blues', ax=ax)
#ax.set_xlim([170000,371000])
#ax.set_ylim([305000,450000])
#plt.show()
```

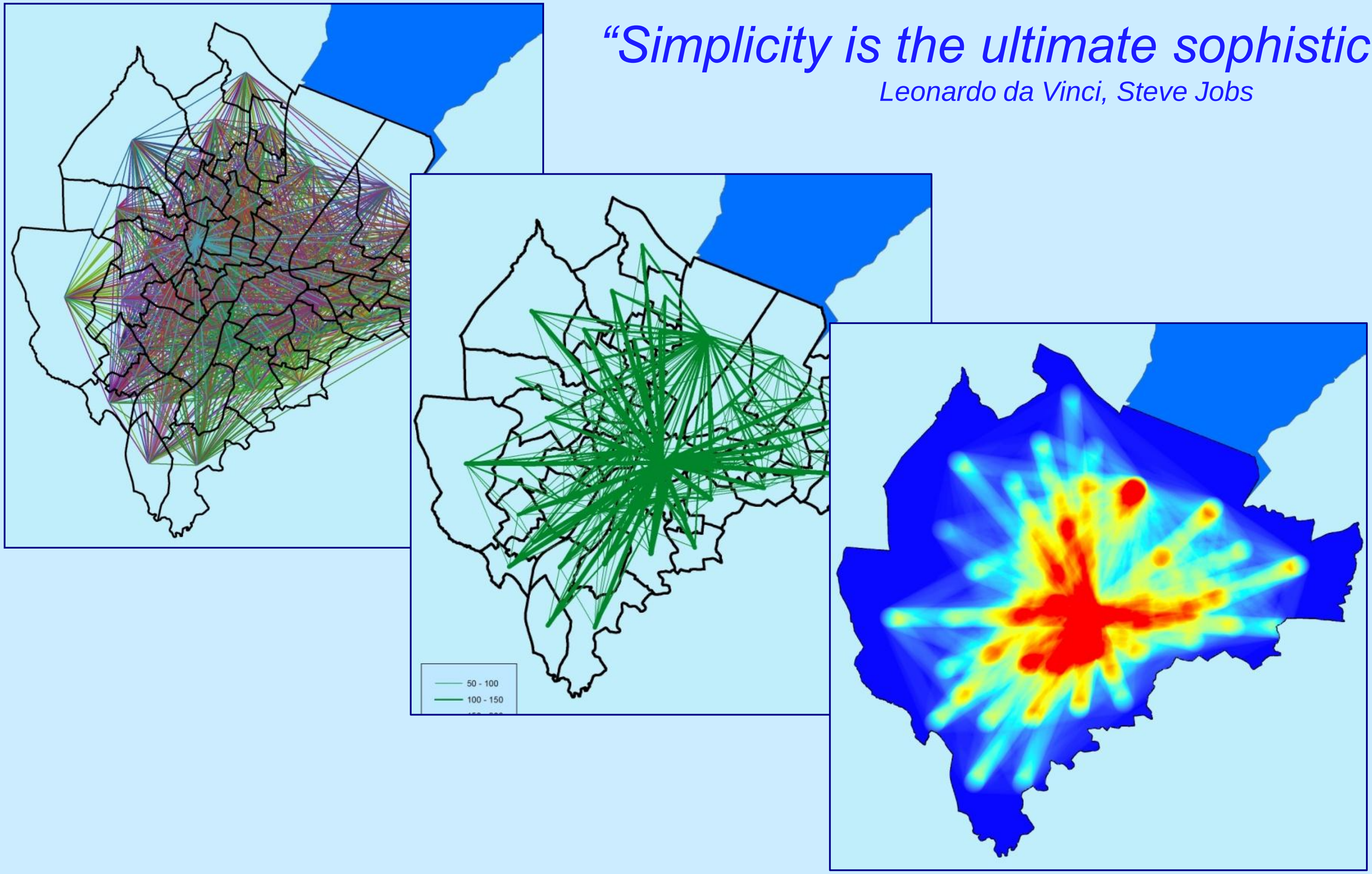
Create new fit for purpose regions
Test fit based on internal commuter flows
Rework regions if necessary
Run spatial interaction based on new regions and demographics
Evaluations and Visualisations

Modelling Commuter Movements and Demographics: Geovisualization

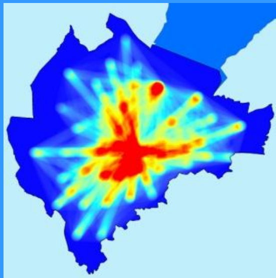


“Simplicity is the ultimate sophistication”

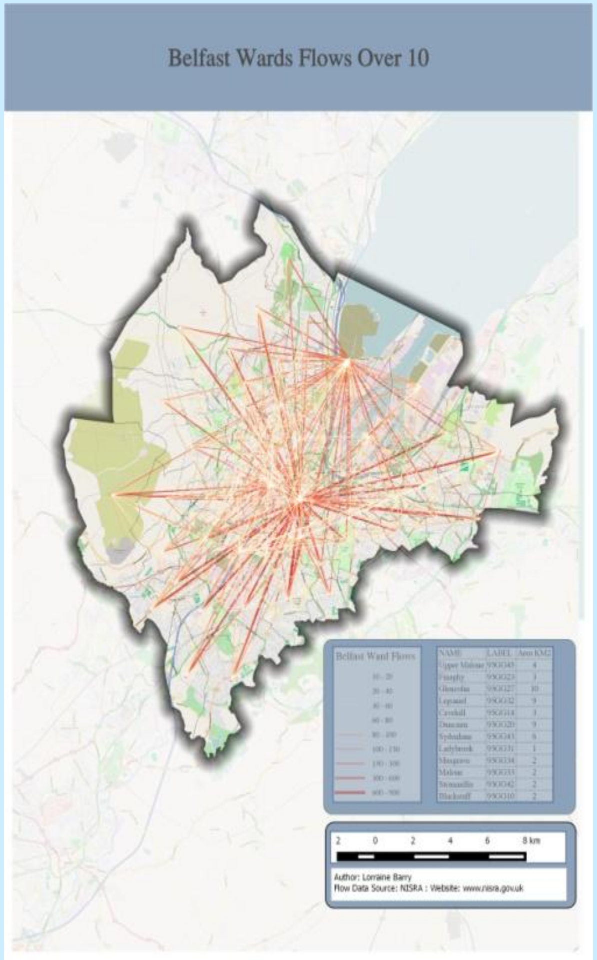
Leonardo da Vinci, Steve Jobs



Modelling Commuter Movements and Demographics: Geovisualization



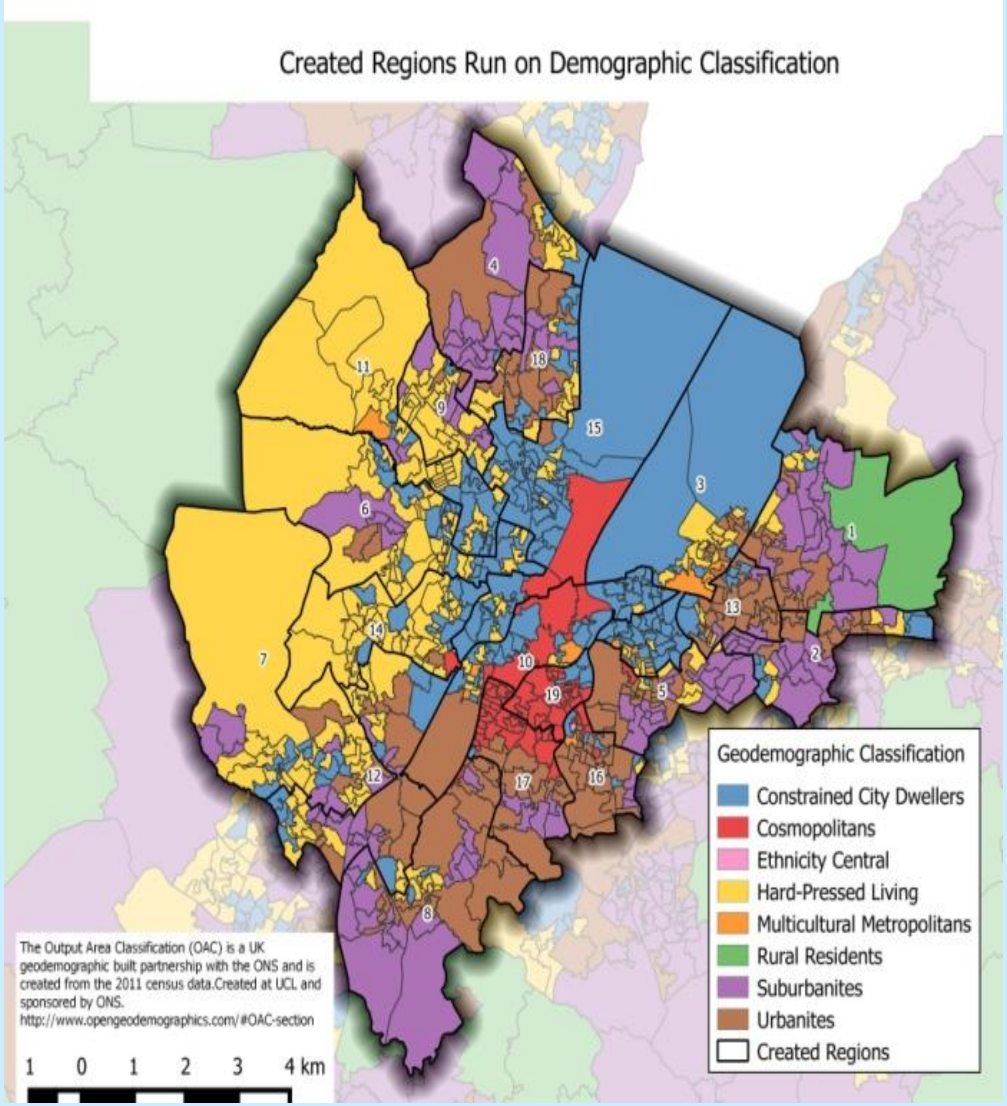
QGIS



Belfast Wards Flows Over 10

NAME	LABEL	Area KM2
Upper Malone	95GG45	4
Finaghy	95GG23	3
Glencolin	95GG27	10
Ligoniel	95GG32	9
Cavehill	95GG14	3
Duncairn	95GG20	9
Sydenham	95GG43	6
Ladybrook	95GG31	1
Musgrave	95GG34	2
Malone	95GG33	2
Stranmillis	95GG42	2
Blackstaff	95GG10	2

Author: Lorraine Barry
Flow Data Source: NISRA : Website: www.nisra.gov.uk



Created Regions Run on Demographic Classification

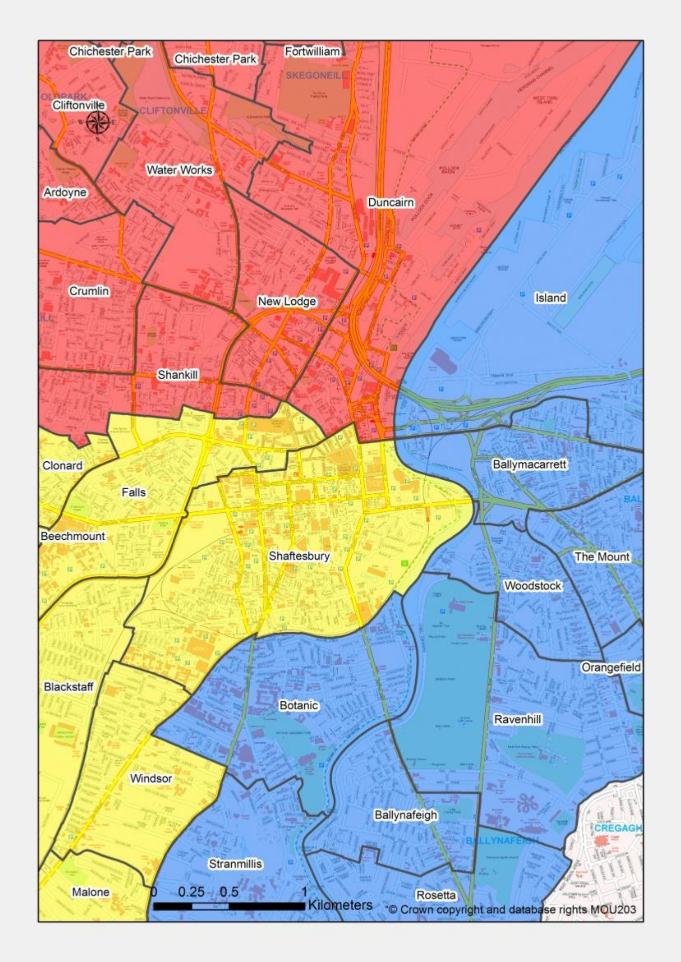
Geodemographic Classification

- 🟦 Constrained City Dwellers
- 🟥 Cosmopolitans
- 🟪 Ethnicity Central
- 🟨 Hard-Pressed Living
- 🟧 Multicultural Metropolitans
- 🟩 Rural Residents
- 🟪 Suburbanites
- 🟫 Urbanites
- □ Created Regions

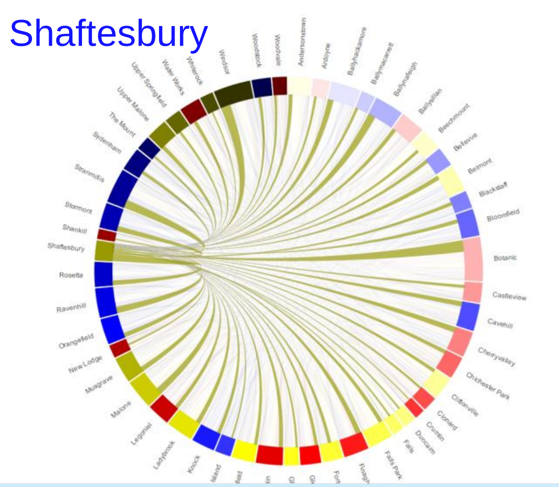
The Output Area Classification (OAC) is a UK geodemographic built partnership with the ONS and is created from the 2011 census data.Created at UCL and sponsored by ONS.
http://www.opengeodemographics.com/#OAC-section

Modelling Commuter Movements and Demographics: Geovisualization

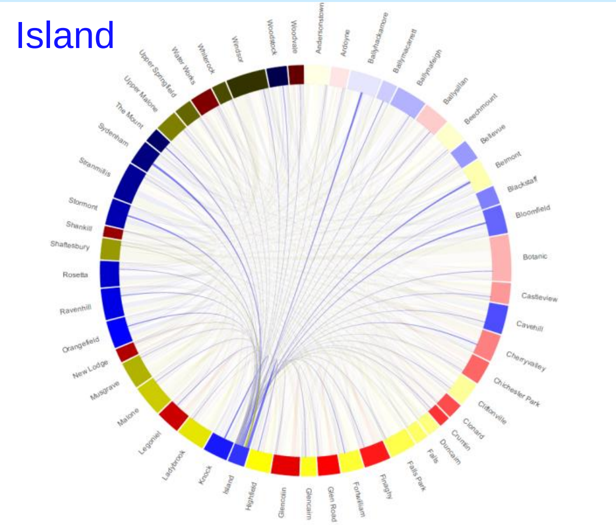
D3 JavaScript

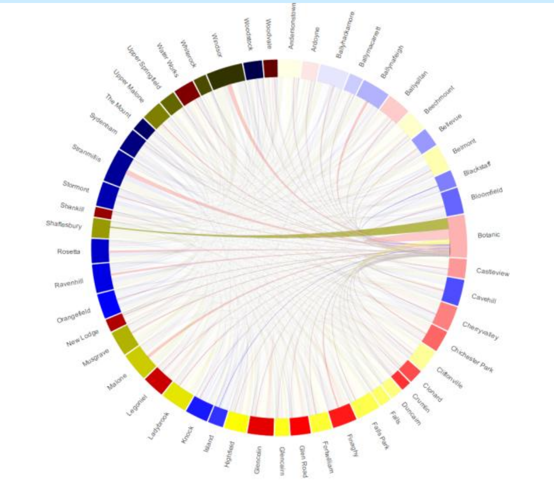


Shaftesbury



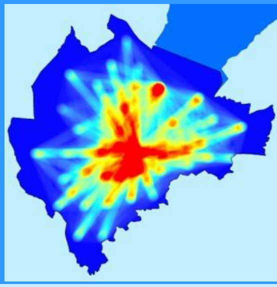
Island





Botanic

Modelling Commuter Movements and Demographics:



Near Future Work

•**Evaluation of demographics on commuting patterns**.

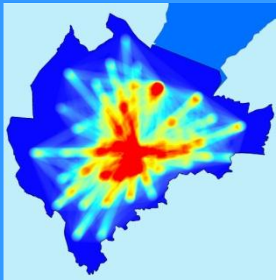
•**Comparison of 1991, 2001 and 2011**. Comparison over time would allow for evaluations to be made on the change of rates in commuting or migration and evaluate the effect on commuting patterns of a changing economical and social setting..

•Application of methodology to **other interaction Origin Destination** data

•Evaluation and examination of the applicability of this research to wider interactions **applications of goods and services**.

•Emphasis on this fundamental importance of Open Source Geospatial Data Science

Open Source Geospatial Data Science



Open Source Conferences:

- FOSS4G- NA, Free and Open Source for Geospatial, North America, March 2015
- FOSS4G, Free and Open Source for Geospatial, Bonn, August 2016
- Awarded OSGeo Student Poster Prize at FOSS4G Bonn

Presentation at BelFOSS, Queen's University Belfast, January 2016 and January 2017

Centre for GIS and Geomatics
School of Natural and Built Environment
Queen's University Belfast

Lorraine Barry **l.barry@qub.ac.uk** **@lorraine__barry**

Supervisors: Dr. Ian Shuttleworth, Dr. Jennifer McKinley.
Advice from Dr Chris Lloyd (University of Liverpool)