Spatial Analysis Notes

Francisco Rowe & Dani Arribas-Bel

2020-12-23

Contents

Chapter 1

Spatial Analysis Notes

This book contains computational illustrations on spatial analytical approaches using R.

Chapter 2

Overview

2.1 Computational Environment

Dependencies + Landing page for guides

2.1.1 Dependency list

List of libraries used in this book:

```
deps <- list(
    "arm",
    "car",
    "corrplot",
    "FRK",
    "gghighlight",
    "ggplot2",
    "ggmap",
    "GISTools",
    "gstat",
    "jtools",
    "kableExtra",
    "knitr",
    "lme4",
    "lmtest",
    "lubridate",
    "MASS",
    "merTools",
    "plyr",
    "RColorBrewer",
    "rgdal",
    "sf",
```

```
    "sjPlot",
    "sp",
    "spgwr",
    "spatialreg",
    "spacetime",
    "stargazer",
    "tidyverse",
    "tmap",
    "viridis"
)
```

And we can load them all to make sure they are installed:

```
for(lib in deps){library(lib, character.only = TRUE)}
```

```
## Loading required package: MASS
## Loading required package: Matrix
## Loading required package: lme4
##
## arm (Version 1.11-2, built: 2020-7-27)
## Working directory is /home/rstudio/Documents
## Loading required package: carData
## Registered S3 methods overwritten by 'car':
##   method                          from
##   influence.merMod                lme4
##   cooks.distance.influence.merMod lme4
##   dfbeta.influence.merMod         lme4
##   dfbetas.influence.merMod        lme4
##
## Attaching package: 'car'
## The following object is masked from 'package:arm':
##
##     logit
## corrplot 0.84 loaded
##
## Attaching package: 'corrplot'
## The following object is masked from 'package:arm':
##
##     corrplot
## Loading required package: ggplot2
```

```
## Google's Terms of Service: https://cloud.google.com/maps-platform/terms/.

## Please cite ggmap if you use it! See citation("ggmap") for details.

## Loading required package: maptools

## Loading required package: sp

## Checking rgeos availability: TRUE

## Loading required package: RColorBrewer

## Loading required package: rgeos

## rgeos version: 0.5-5, (SVN revision 640)
##  GEOS runtime version: 3.8.0-CAPI-1.13.1
##  Linking to sp version: 1.4-4
##  Polygon checking: TRUE

##
## Attaching package: 'jtools'

## The following object is masked from 'package:arm':
##
##     standardize

## Loading required package: zoo

##
## Attaching package: 'zoo'

## The following objects are masked from 'package:base':
##
##     as.Date, as.Date.numeric

##
## Attaching package: 'lubridate'

## The following objects are masked from 'package:rgeos':
##
##     intersect, setdiff, union

## The following objects are masked from 'package:base':
##
##     date, intersect, setdiff, union

## Registered S3 methods overwritten by 'broom':
##   method            from
##   tidy.glht         jtools
##   tidy.summary.glht jtools

## Registered S3 method overwritten by 'broom.mixed':
##   method      from
##   tidy.gamlss broom
```

```
## rgdal: version: 1.5-18, (SVN revision 1082)
## Geospatial Data Abstraction Library extensions to R successfully loaded
## Loaded GDAL runtime: GDAL 3.0.4, released 2020/01/28
## Path to GDAL shared files: /usr/share/gdal
## GDAL binary built with GEOS: TRUE
## Loaded PROJ runtime: Rel. 6.3.1, February 10th, 2020, [PJ_VERSION: 631]
## Path to PROJ shared files: /usr/share/proj
## Linking to sp version:1.4-4
## To mute warnings of possible GDAL/OSR exportToProj4() degradation,
## use options("rgdal_show_exportToProj4_warnings"="none") before loading rgdal.

## Linking to GEOS 3.8.0, GDAL 3.0.4, PROJ 6.3.1

## Loading required package: spData

## NOTE: This package does not constitute approval of GWR
## as a method of spatial analysis; see example(gwr)

## Registered S3 methods overwritten by 'spatialreg':
##   method                   from
##   residuals.stsls          spdep
##   deviance.stsls           spdep
##   coef.stsls               spdep
##   print.stsls              spdep
##   summary.stsls            spdep
##   print.summary.stsls      spdep
##   residuals.gmsar          spdep
##   deviance.gmsar           spdep
##   coef.gmsar               spdep
##   fitted.gmsar             spdep
##   print.gmsar              spdep
##   summary.gmsar            spdep
##   print.summary.gmsar      spdep
##   print.lagmess            spdep
##   summary.lagmess          spdep
##   print.summary.lagmess    spdep
##   residuals.lagmess        spdep
##   deviance.lagmess         spdep
##   coef.lagmess             spdep
##   fitted.lagmess           spdep
##   logLik.lagmess           spdep
##   fitted.SFResult          spdep
##   print.SFResult           spdep
##   fitted.ME_res            spdep
##   print.ME_res             spdep
##   print.lagImpact          spdep
##   plot.lagImpact           spdep
##   summary.lagImpact        spdep
```

```
##   HPDinterval.lagImpact    spdep
##   print.summary.lagImpact  spdep
##   print.sarlm              spdep
##   summary.sarlm            spdep
##   residuals.sarlm          spdep
##   deviance.sarlm           spdep
##   coef.sarlm               spdep
##   vcov.sarlm               spdep
##   fitted.sarlm             spdep
##   logLik.sarlm             spdep
##   anova.sarlm              spdep
##   predict.sarlm            spdep
##   print.summary.sarlm      spdep
##   print.sarlm.pred         spdep
##   as.data.frame.sarlm.pred spdep
##   residuals.spautolm       spdep
##   deviance.spautolm        spdep
##   coef.spautolm            spdep
##   fitted.spautolm          spdep
##   print.spautolm           spdep
##   summary.spautolm         spdep
##   logLik.spautolm          spdep
##   print.summary.spautolm   spdep
##   print.WXImpact           spdep
##   summary.WXImpact         spdep
##   print.summary.WXImpact   spdep
##   predict.SLX              spdep

##
## Please cite as:

##  Hlavac, Marek (2018). stargazer: Well-Formatted Regression and Summary Statistics Tables.

##  R package version 5.2.2. https://CRAN.R-project.org/package=stargazer

## -- Attaching packages ------------------------------------------------------------------- tid

## v tibble  3.0.4     v dplyr   1.0.2
## v tidyr   1.1.2     v stringr 1.4.0
## v readr   1.4.0     v forcats 0.5.0
## v purrr   0.3.4

## -- Conflicts ---------------------------------------------------------------------- tidyverse
## x dplyr::arrange()         masks plyr::arrange()
## x lubridate::as.difftime() masks base::as.difftime()
## x purrr::compact()         masks plyr::compact()
## x dplyr::count()           masks plyr::count()
## x lubridate::date()        masks base::date()
## x tidyr::expand()          masks Matrix::expand()
```

```
## x dplyr::failwith()        masks plyr::failwith()
## x dplyr::filter()          masks stats::filter()
## x dplyr::group_rows()      masks kableExtra::group_rows()
## x dplyr::id()              masks plyr::id()
## x lubridate::intersect()   masks rgeos::intersect(), base::intersect()
## x dplyr::lag()             masks stats::lag()
## x dplyr::mutate()          masks plyr::mutate()
## x tidyr::pack()            masks Matrix::pack()
## x dplyr::recode()          masks car::recode()
## x dplyr::rename()          masks plyr::rename()
## x dplyr::select()          masks MASS::select()
## x lubridate::setdiff()     masks rgeos::setdiff(), base::setdiff()
## x purrr::some()            masks car::some()
## x dplyr::summarise()       masks plyr::summarise()
## x dplyr::summarize()       masks plyr::summarize()
## x lubridate::union()       masks rgeos::union(), base::union()
## x tidyr::unpack()          masks Matrix::unpack()
```

```
## Loading required package: viridisLite
```

Chapter 3

Spatial Data

Chapter 4

Data Wrangling

This chapter[1] introduces R Notebooks, basic functions and data types. These are all important concepts that we will use during the module.

If you are already familiar with R, R notebooks and data types, you may want to jump to Section Read Data and start from there. This section describes how to read and manipulate data using `sf` and `tidyverse` functions, including `mutate()`, `%>%` (known as pipe operator), `select()`, `filter()` and specific packages and functions how to manipulate spatial data.

The chapter is based on:

- Grolemund and Wickham (2019), this book illustrates key libraries, including tidyverse, and functions for data manipulation in R
- Xie et al. (2019), excellent introduction to R markdown!
- Williamson (2018), some examples from the first lecture of ENVS450 are used to explain the various types of random variables.
- Lovelace et al. (2020), a really good book on handling spatial data and historical background of the evolution of R packages for spatial data analysis.

4.1 Dependencies

This tutorial uses the libraries below. Ensure they are installed on your machine[2] before loading them executing the following code chunk:

[1] This chapter is part of Spatial Analysis Notes Introduction – R Notebooks + Basic Functions + Data Types by Francisco Rowe is licensed under a Creative Commons Attribution-NonCommercial-ShareAlike 4.0 International License.

[2] You can install package `mypackage` by running the command `install.packages("mypackage")` on the R prompt or through the `Tools --> Install`

```
# Data manipulation, transformation and visualisation
library(tidyverse)
# Nice tables
library(kableExtra)
# Simple features (a standardised way to encode vector data ie. points, lines, polygon
library(sf)
# Spatial objects conversion
library(sp)
# Thematic maps
library(tmap)
# Colour palettes
library(RColorBrewer)
# More colour palettes
library(viridis)
```

4.2 Introducing R

R is a freely available language and environment for statistical computing and graphics which provides a wide variety of statistical and graphical techniques. It has gained widespread use in academia and industry. R offers a wider array of functionality than a traditional statistics package, such as SPSS and is composed of core (base) functionality, and is expandable through libraries hosted on CRAN. CRAN is a network of ftp and web servers around the world that store identical, up-to-date, versions of code and documentation for R.

Commands are sent to R using either the terminal / command line or the R Console which is installed with R on either Windows or OS X. On Linux, there is no equivalent of the console, however, third party solutions exist. On your own machine, R can be installed from here.

Normally RStudio is used to implement R coding. RStudio is an integrated development environment (IDE) for R and provides a more user-friendly front-end to R than the front-end provided with R.

To run R or RStudio, just double click on the R or RStudio icon. Throughout this module, we will be using RStudio:

If you would like to know more about the various features of RStudio, watch this video

4.3 Setting the working directory

Before we start any analysis, ensure to set the path to the directory where we are working. We can easily do that with `setwd()`. Please replace in the following

`Packages...` menu in RStudio.

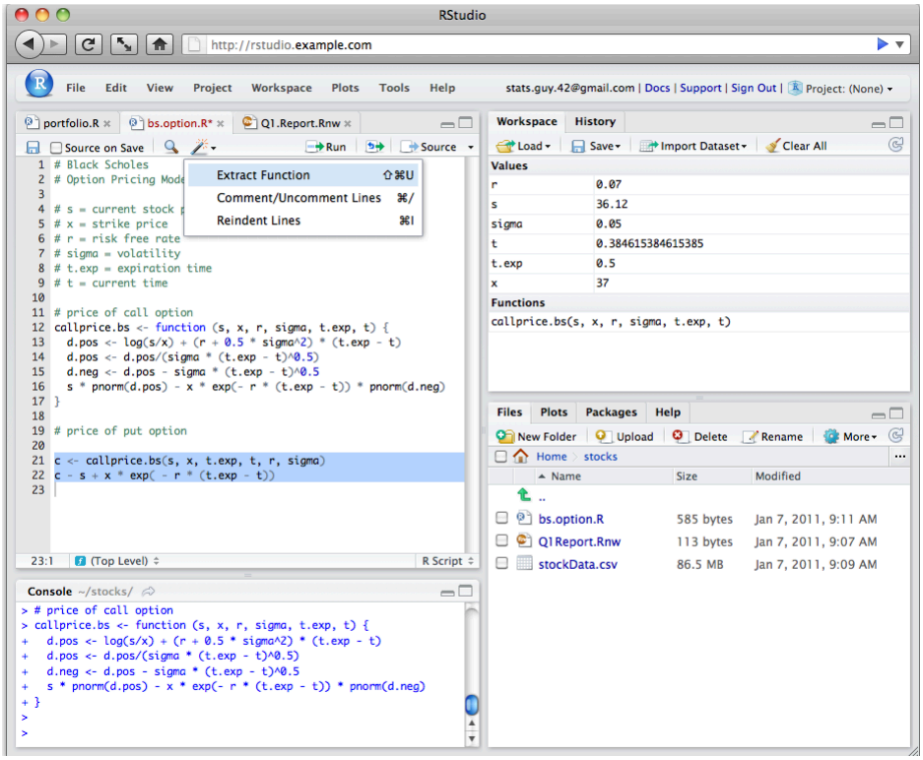


Figure 4.1: Fig. 1. RStudio features.

line the path to the folder where you have placed this file -and where the `data` folder lives.

```
#setwd('../data/sar.csv')
#setwd('.')
```

Note: It is good practice to not include spaces when naming folders and files. Use *underscores* or *dots*.

You can check your current working directory by typing:

```
getwd()
```

```
## [1] "/home/rstudio/Documents"
```

4.4 R Scripts and Notebooks

An *R script* is a series of commands that you can execute at one time and help you save time. So you don't repeat the same steps every time you want to execute the same process with different datasets. An R script is just a plain text file with R commands in it.

To create an R script in RStudio, you need to

- Open a new script file: *File* > *New File* > *R Script*
- Write some code on your new script window by typing eg. `mtcars`
- Run the script. Click anywhere on the line of code, then hit *Ctrl + Enter* (Windows) or *Cmd + Enter* (Mac) to run the command or select the code chunk and click *run* on the right-top corner of your script window. If do that, you should get:

```
mtcars
```

```
##                    mpg cyl  disp  hp drat    wt  qsec vs am gear carb
## Mazda RX4         21.0   6 160.0 110 3.90 2.620 16.46  0  1    4    4
## Mazda RX4 Wag     21.0   6 160.0 110 3.90 2.875 17.02  0  1    4    4
## Datsun 710        22.8   4 108.0  93 3.85 2.320 18.61  1  1    4    1
## Hornet 4 Drive    21.4   6 258.0 110 3.08 3.215 19.44  1  0    3    1
## Hornet Sportabout 18.7   8 360.0 175 3.15 3.440 17.02  0  0    3    2
## Valiant           18.1   6 225.0 105 2.76 3.460 20.22  1  0    3    1
## Duster 360        14.3   8 360.0 245 3.21 3.570 15.84  0  0    3    4
## Merc 240D         24.4   4 146.7  62 3.69 3.190 20.00  1  0    4    2
## Merc 230          22.8   4 140.8  95 3.92 3.150 22.90  1  0    4    2
## Merc 280          19.2   6 167.6 123 3.92 3.440 18.30  1  0    4    4
## Merc 280C         17.8   6 167.6 123 3.92 3.440 18.90  1  0    4    4
## Merc 450SE        16.4   8 275.8 180 3.07 4.070 17.40  0  0    3    3
## Merc 450SL        17.3   8 275.8 180 3.07 3.730 17.60  0  0    3    3
## Merc 450SLC       15.2   8 275.8 180 3.07 3.780 18.00  0  0    3    3
```

```
## Cadillac Fleetwood  10.4   8 472.0 205 2.93 5.250 17.98  0  0    3    4
## Lincoln Continental 10.4   8 460.0 215 3.00 5.424 17.82  0  0    3    4
## Chrysler Imperial   14.7   8 440.0 230 3.23 5.345 17.42  0  0    3    4
## Fiat 128            32.4   4  78.7  66 4.08 2.200 19.47  1  1    4    1
## Honda Civic         30.4   4  75.7  52 4.93 1.615 18.52  1  1    4    2
## Toyota Corolla      33.9   4  71.1  65 4.22 1.835 19.90  1  1    4    1
## Toyota Corona       21.5   4 120.1  97 3.70 2.465 20.01  1  0    3    1
## Dodge Challenger    15.5   8 318.0 150 2.76 3.520 16.87  0  0    3    2
## AMC Javelin         15.2   8 304.0 150 3.15 3.435 17.30  0  0    3    2
## Camaro Z28          13.3   8 350.0 245 3.73 3.840 15.41  0  0    3    4
## Pontiac Firebird    19.2   8 400.0 175 3.08 3.845 17.05  0  0    3    2
## Fiat X1-9           27.3   4  79.0  66 4.08 1.935 18.90  1  1    4    1
## Porsche 914-2       26.0   4 120.3  91 4.43 2.140 16.70  0  1    5    2
## Lotus Europa        30.4   4  95.1 113 3.77 1.513 16.90  1  1    5    2
## Ford Pantera L      15.8   8 351.0 264 4.22 3.170 14.50  0  1    5    4
## Ferrari Dino        19.7   6 145.0 175 3.62 2.770 15.50  0  1    5    6
## Maserati Bora       15.0   8 301.0 335 3.54 3.570 14.60  0  1    5    8
## Volvo 142E          21.4   4 121.0 109 4.11 2.780 18.60  1  1    4    2
```

- Save the script: *File* > *Save As*, select your required destination folder, and enter any filename that you like, provided that it ends with the file extension *.R*

An *R Notebook* is an R Markdown document with descriptive text and code chunks that can be executed independently and interactively, with output visible immediately beneath a code chunk - see Xie et al. (2019).

To create an R Notebook, you need to:

- Open a new script file: *File* > *New File* > *R Notebook*

```
---
title: "My Notebook"
output: html_notebook
---
```

Figure 4.2: Fig. 2. YAML metadata for notebooks.

- Insert code chunks, either:

1) use the *Insert* command on the editor toolbar;
2) use the keyboard shortcut *Ctrl + Alt + I* or *Cmd + Option + I* (Mac); or,
3) type the chunk delimiters ```` ```{r} ```` and ```` ``` ````

In a chunk code you can produce text output, tables, graphics and write code! You can control these outputs via chunk options which are provided inside the curly brackets eg.

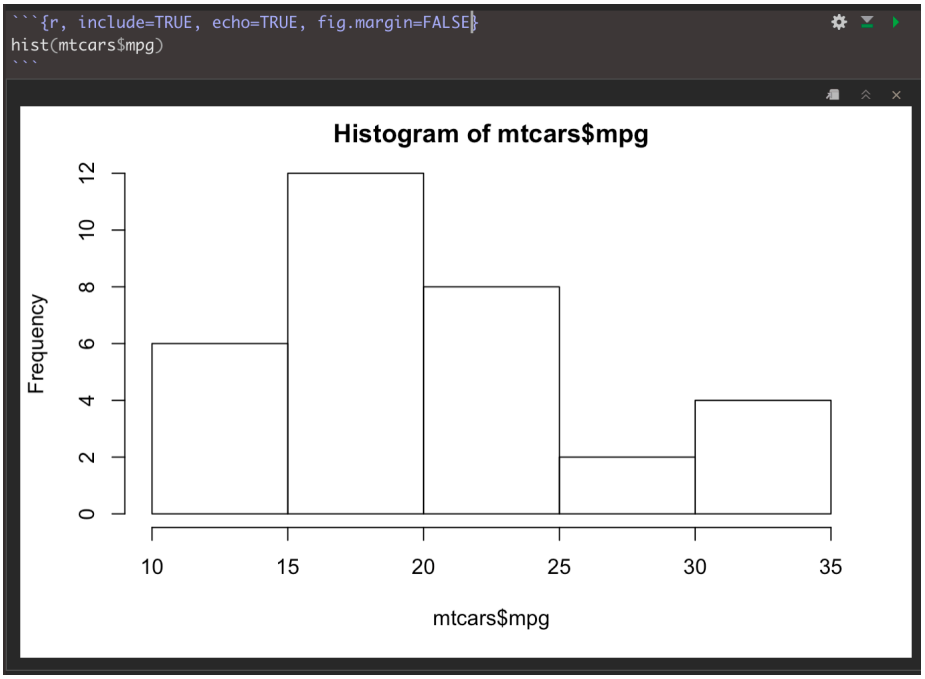


Figure 4.3: Fig. 3. Code chunk example. Details on the various options: https://rmarkdown.rstudio.com/lesson-3.html

- Execute code: hit *"Run Current Chunk"*, *Ctrl + Shift + Enter* or *Cmd + Shift + Enter* (Mac)
- Save an R notebook: *File > Save As*. A notebook has a `*.Rmd` extension and when it is saved a `*.nb.html` file is automatically created. The latter is a self-contained HTML file which contains both a rendered copy of the notebook with all current chunk outputs and a copy of the *.Rmd file itself.

Rstudio also offers a *Preview* option on the toolbar which can be used to create pdf, html and word versions of the notebook. To do this, choose from the drop-down list menu `knit to ...`

4.5 Getting Help

You can use `help` or `?` to ask for details for a specific function:

```
help(sqrt) #or ?sqrt
```

And using `example` provides examples for said function:

```
example(sqrt)
```

```
##
## sqrt> require(stats) # for spline
##
## sqrt> require(graphics)
##
## sqrt> xx <- -9:9
##
## sqrt> plot(xx, sqrt(abs(xx)),  col = "red")
```

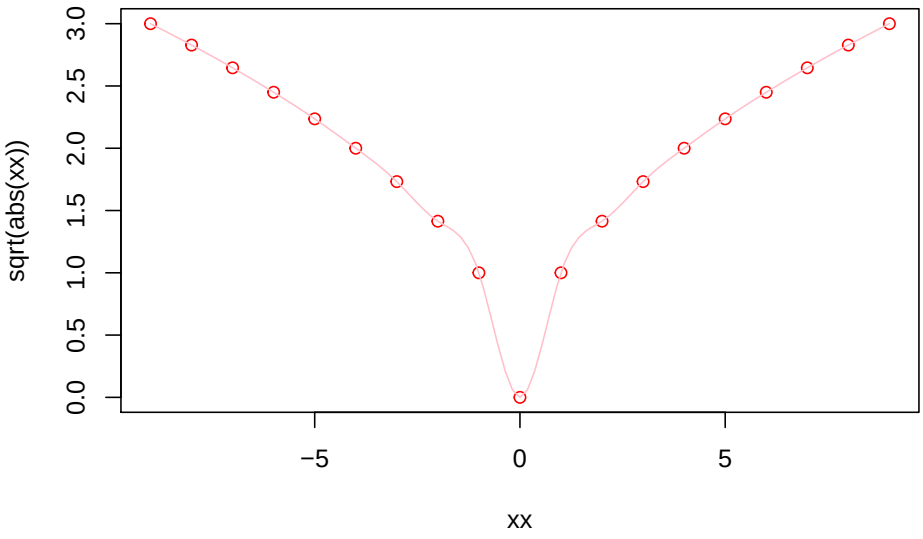


Figure 4.4: Example sqrt

```
##
## sqrt> lines(spline(xx, sqrt(abs(xx)), n=101), col = "pink")
```

4.6 Variables and objects

An *object* is a data structure having attributes and methods. In fact, everything in R is an object!

A *variable* is a type of data object. Data objects also include list, vector, matrices and text.

- Creating a data object

In R a variable can be created by using the symbol `<-` to assign a value to a variable name. The variable name is entered on the left `<-` and the value on the right. Note: Data objects can be given any name, provided that they start with a letter of the alphabet, and include only letters of the alphabet, numbers and the characters `.` and `_`. Hence AgeGroup, Age_Group and Age.Group are all valid names for an R data object. Note also that R is case-sensitive, to agegroup and AgeGroup would be treated as different data objects.

To save the value *28* to a variable (data object) labelled *age*, run the code:

```
age <- 28
```

- Inspecting a data object

To inspect the contents of the data object *age* run the following line of code:

```
age
```

```
## [1] 28
```

Find out what kind (class) of data object *age* is using:

```
class(age)
```

```
## [1] "numeric"
```

Inspect the structure of the *age* data object:

```
str(age)
```

```
##  num 28
```

- The *vector* data object

What if we have more than one response? We can use the `c( )` function to combine multiple values into one data vector object:

```
age <- c(28, 36, 25, 24, 32)
age
```

```
## [1] 28 36 25 24 32
```

```
class(age) #Still numeric..
```

```
## [1] "numeric"
```

```
str(age) #..but now a vector (set) of 5 separate values
```

```
##  num [1:5] 28 36 25 24 32
```

Note that on each line in the code above any text following the `#` character is ignored by R when executing the code. Instead, text following a `#` can be used

to add comments to the code to make clear what the code is doing. Two marks of good code are a clear layout and clear commentary on the code.

4.6.1 Basic Data Types

There are a number of data types. Four are the most common. In R, **numeric** is the default type for numbers. It stores all numbers as floating-point numbers (numbers with decimals). This is because most statistical calculations deal with numbers with up to two decimals.

- Numeric

```
num <- 4.5 # Decimal values
class(num)
```

```
## [1] "numeric"
```

- Integer

```
int <- as.integer(4) # Natural numbers. Note integers are also numerics.
class(int)
```

```
## [1] "integer"
```

- Character

```
cha <- "are you enjoying this?" # text or string. You can also type `as.character("are you enjoyi
class(cha)
```

```
## [1] "character"
```

- Logical

```
log <- 2 < 1 # assigns TRUE or FALSE. In this case, FALSE as 2 is greater than 1
log
```

```
## [1] FALSE
```

```
class(log)
```

```
## [1] "logical"
```

4.6.2 Random Variables

In statistics, we differentiate between data to capture:

- *Qualitative attributes* categorise objects eg.gender, marital status. To measure these attributes, we use *Categorical* data which can be divided into:
  - *Nominal* data in categories that have no inherent order eg. gender
  - *Ordinal* data in categories that have an inherent order eg. income bands

- *Quantitative attributes*:
  - *Discrete* data: count objects of a certain category eg. number of kids, cars
  - *Continuous* data: precise numeric measures eg. weight, income, length.

In R these three types of random variables are represented by the following types of R data object:

variables	objects
nominal	factor
ordinal	ordered factor
discrete	numeric
continuous	numeric

We have already encountered the R data type *numeric*. The next section introduces the *factor* data type.

4.6.2.1 Factor

What is a factor?

A factor variable assigns a numeric code to each possible category (*level*) in a variable. Behind the scenes, R stores the variable using these numeric codes to save space and speed up computing. For example, compare the size of a list of `10,000` *males* and *females* to a list of `10,000` `1s` and `0s`. At the same time R also saves the category names associated with each numeric code (level). These are used for display purposes.

For example, the variable *gender*, converted to a factor, would be stored as a series of `1s` and `2s`, where `1 = female` and `2 = male`; but would be displayed in all outputs using their category labels of *female* and *male*.

Creating a factor

To convert a numeric or character vector into a factor use the `factor( )` function. For instance:

```
gender <- c("female","male","male","female","female") # create a gender variable
gender <- factor(gender) # replace character vector with a factor version
gender
```

```
## [1] female male   male   female female
## Levels: female male
```

```
class(gender)
```

```
## [1] "factor"
```

```
str(gender)
```

```
##  Factor w/ 2 levels "female","male": 1 2 2 1 1
```

Now *gender* is a factor and is stored as a series of `1s` and `2s`, with `1s` representing `females` and `2s` representing `males`. The function `levels( )` lists the levels (categories) associated with a given factor variable:

```
levels(gender)
```

```
## [1] "female" "male"
```

The categories are reported in the order that they have been numbered (starting from `1`). Hence from the output we can infer that `females` are coded as `1`, and `males` as 2.

4.7 Data Frames

R stores different types of data using different types of data structure. Data are normally stored as a *data.frame*. A data frames contain one row per observation (e.g. wards) and one column per attribute (eg. population and health).

We create three variables wards, population (`pop`) and people with good health (`ghealth`). We use 2011 census data counts for total population and good health for wards in Liverpool.

```
wards <- c("Allerton and Hunts Cross","Anfield","Belle Vale","Central","Childwall","Church","Club

pop <- c(14853,14510,15004,20340,13908,13974,15272,14045,14503,
                14561,14782,16786,16132,15377,16115,13312,13816,15047,
                16461,17009,17104,18422,12991,20300,16489,16481,14772,
                14382,12921,16746)

ghealth <- c(7274,6124,6129,11925,7219,7461,6403,5930,7094,6992,
                 5517,7879,8990,6495,6662,5981,7322,6529,7192,7953,
                 7636,9001,6450,8973,7302,7521,7268,7013,6025,7717)
```

Note that `pop` and `ghealth` and `wards` contains characters.

4.7.1 Creating A Data Frame

We can create a data frame and examine its structure:

```
df <- data.frame(wards, pop, ghealth)
df # or use view(data)
```

```
##                        wards   pop ghealth
## 1  Allerton and Hunts Cross 14853    7274
## 2                    Anfield 14510    6124
## 3                Belle Vale 15004    6129
## 4                    Central 20340   11925
```

```
## 5                 Childwall 13908    7219
## 6                    Church 13974    7461
## 7                  Clubmoor 15272    6403
## 8                    County 14045    5930
## 9                Cressington 14503    7094
## 10                  Croxteth 14561    6992
## 11                   Everton 14782    5517
## 12                Fazakerley 16786    7879
## 13                 Greenbank 16132    8990
## 14 Kensington and Fairfield 15377    6495
## 15                  Kirkdale 16115    6662
## 16                Knotty Ash 13312    5981
## 17              Mossley Hill 13816    7322
## 18              Norris Green 15047    6529
## 19                  Old Swan 16461    7192
## 20                    Picton 17009    7953
## 21              Princes Park 17104    7636
## 22                 Riverside 18422    9001
## 23              St Michael's 12991    6450
## 24             Speke-Garston 20300    8973
## 25 Tuebrook and Stoneycroft 16489    7302
## 26                  Warbreck 16481    7521
## 27                 Wavertree 14772    7268
## 28                West Derby 14382    7013
## 29                   Woolton 12921    6025
## 30                  Yew Tree 16746    7717
```

```
str(df) # or use glimpse(data)
```

```
## 'data.frame':    30 obs. of  3 variables:
##  $ wards  : chr  "Allerton and Hunts Cross" "Anfield" "Belle Vale" "Central" ...
##  $ pop    : num  14853 14510 15004 20340 13908 ...
##  $ ghealth: num  7274 6124 6129 11925 7219 ...
```

4.7.2 Referencing Data Frames

Throughout this module, you will need to refer to particular parts of a dataframe - perhaps a particular column (an area attribute); or a particular subset of respondents. Hence it is worth spending some time now mastering this particular skill.

The relevant R function, `[ ]`, has the format `[row,col]` or, more generally, `[set of rows, set of cols]`.

Run the following commands to get a feel of how to extract different slices of the data:

```
df # whole data.frame
df[1, 1] # contents of first row and column
df[2, 2:3] # contents of the second row, second and third columns
df[1, ] # first row, ALL columns [the default if no columns specified]
df[ ,1:2] # ALL rows; first and second columns
df[c(1,3,5), ] # rows 1,3,5; ALL columns
df[ , 2] # ALL rows; second column (by default results containing only
             #one column are converted back into a vector)
df[ , 2, drop=FALSE] # ALL rows; second column (returned as a data.frame)
```

In the above, note that we have used two other R functions:

- `1:3` The colon operator tells R to produce a list of numbers including the named start and end points.
- `c(1,3,5)` Tells R to combine the contents within the brackets into one list of objects

Run both of these fuctions on their own to get a better understanding of what they do.

Three other methods for referencing the contents of a data.frame make direct use of the variable names within the data.frame, which tends to make for easier to read/understand code:

```
df[,"pop"] # variable name in quotes inside the square brackets
df$pop # variable name prefixed with $ and appended to the data.frame name
# or you can use attach
attach(df)
pop # but be careful if you already have an age variable in your local workspace
```

Want to check the variables available, use the `names( )`:

```
names(df)
```

```
## [1] "wards"   "pop"     "ghealth"
```

4.8 Read Data

Ensure your memory is clear

```
rm(list=ls()) # rm for targeted deletion / ls for listing all existing objects
```

There are many commands to read / load data onto R. The command to use will depend upon the format they have been saved. Normally they are saved in *csv* format from Excel or other software packages. So we use either:

- `df <- read.table("path/file_name.csv", header = FALSE, sep =",")`
- `df <- read("path/file_name.csv", header = FALSE)`

- `df <- read.csv2("path/file_name.csv", header = FALSE)`

To read files in other formats, refer to this useful DataCamp tutorial

```
census <- read.csv("data/census/census_data.csv")
head(census)
```

```
##        code                      ward pop16_74 higher_managerial   pop ghealth
## 1 E05000886 Allerton and Hunts Cross    10930              1103 14853    7274
## 2 E05000887                 Anfield    10712               312 14510    6124
## 3 E05000888               Belle Vale    10987               432 15004    6129
## 4 E05000889                  Central    19174              1346 20340   11925
## 5 E05000890                Childwall    10410              1123 13908    7219
## 6 E05000891                   Church    10569              1843 13974    7461
```

```
# NOTE: always ensure your are setting the correct directory leading to the data.
# It may differ from your existing working directory
```

4.8.1 Quickly inspect the data

1. What class?
2. What R data types?
3. What data types?

```
# 1
class(census)
# 2 & 3
str(census)
```

Just interested in the variable names:

```
names(census)
```

```
## [1] "code"              "ward"              "pop16_74"
## [4] "higher_managerial" "pop"               "ghealth"
```

or want to view the data:

```
View(census)
```

4.9 Manipulation Data

4.9.1 Adding New Variables

Usually you want to add / create new variables to your data frame using existing variables eg. computing percentages by dividing two variables. There are many ways in which you can do this i.e. referecing a data frame as we have done above, or using `$` (e.g. `census$pop`). For this module, we'll use `tidyverse`:

```
census <- census %>% mutate(per_ghealth = ghealth / pop)
```

Note we used a *pipe operator* %>%, which helps make the code more efficient and readable - more details, see Grolemund and Wickham (2019). When using the pipe operator, recall to first indicate the data frame before %>%.

Note also the use a variable name before the = sign in brackets to indicate the name of the new variable after `mutate`.

4.9.2 Selecting Variables

Usually you want to select a subset of variables for your analysis as storing to large data sets in your R memory can reduce the processing speed of your machine. A selection of data can be achieved by using the `select` function:

```
ndf <- census %>% select(ward, pop16_74, per_ghealth)
```

Again first indicate the data frame and then the variable you want to select to build a new data frame. Note the code chunk above has created a new data frame called `ndf`. Explore it.

4.9.3 Filtering Data

You may also want to filter values based on defined conditions. You may want to filter observations greater than a certain threshold or only areas within a certain region. For example, you may want to select areas with a percentage of good health population over 50%:

```
ndf2 <- census %>% filter(per_ghealth < 0.5)
```

You can use more than one variables to set conditions. Use "`,`" to add a condition.

4.9.4 Joining Data Drames

When working with spatial data, we often need to join data. To this end, you need a common unique `id variable`. Let's say, we want to add a data frame containing census data on households for Liverpool, and join the new attributes to one of the existing data frames in the workspace. First we will read the data frame we want to join (ie. `census_data2.csv`).

```
# read data
census2 <- read.csv("data/census/census_data2.csv")
# visualise data structure
str(census2)
```

```
## 'data.frame':    30 obs. of  3 variables:
##  $ geo_code              : chr  "E05000886" "E05000887" "E05000888" "E05000889" ...
##  $ households            : int  6359 6622 6622 7139 5391 5884 6576 6745 6317 6024 ...
```

```
##  $ socialrented_households: int  827 1508 2818 1311 374 178 2859 1564 1023 1558 ...
```

The variable `geo_code` in this data frame corresponds to the `code` in the existing data frame and they are unique so they can be automatically matched by using the `merge()` function. The `merge()` function uses two arguments: `x` and `y`. The former refers to data frame 1 and the latter to data frame 2. Both of these two data frames must have a `id` variable containing the same information. Note they can have different names. Another key argument to include is `all.x=TRUE` which tells the function to keep all the records in `x`, but only those in `y` that match in case there are discrepancies in the `id` variable.

```
# join data frames
join_dfs <- merge(census, census2, by.x="code", by.y="geo_code", all.x = TRUE)
# check data
head(join_dfs)
```

```
##        code                      ward pop16_74 higher_managerial   pop ghealth
## 1 E05000886 Allerton and Hunts Cross    10930              1103 14853    7274
## 2 E05000887                 Anfield    10712               312 14510    6124
## 3 E05000888               Belle Vale    10987               432 15004    6129
## 4 E05000889                  Central    19174              1346 20340   11925
## 5 E05000890                Childwall    10410              1123 13908    7219
## 6 E05000891                   Church    10569              1843 13974    7461
##   per_ghealth households socialrented_households
## 1   0.4897327       6359                    827
## 2   0.4220538       6622                   1508
## 3   0.4084911       6622                   2818
## 4   0.5862832       7139                   1311
## 5   0.5190538       5391                    374
## 6   0.5339201       5884                    178
```

4.9.5 Saving Data

It may also be convinient to save your R projects. They contains all the objects that you have created in your workspace by using the `save.image( )` function:

```
save.image("week1_envs453.RData")
```

This creates a file labelled "week1_envs453.RData" in your working directory. You can load this at a later stage using the `load( )` function.

```
load("week1_envs453.RData")
```

Alternatively you can save / export your data into a `csv` file. The first argument in the function is the object name, and the second: the name of the csv we want to create.

```
write.csv(join_dfs, "join_censusdfs.csv")
```

4.10 Using Spatial Data Frames

A core area of this module is learning to work with spatial data in R. R has various purposedly designed **packages** for manipulation of spatial data and spatial analysis techniques. Various R packages exist in CRAN eg. `spatial`, `sgeostat`, `splancs`, `maptools`, `tmap`, `rgdal`, `spand` and more recent development of `sf` - see Lovelace et al. (2020) for a great description and historical context for some of these packages.

During this session, we will use `sf`.

We first need to import our spatial data. We will use a shapefile containing data at Output Area (OA) level for Liverpool. These data illustrates the hierarchical structure of spatial data.

4.10.1 Read Spatial Data

```
oa_shp <- st_read("data/census/Liverpool_OA.shp")
```

```
## Reading layer `Liverpool_OA' from data source `/home/rstudio/Documents/data/census/Liverpool_O
## Simple feature collection with 1584 features and 18 fields
## geometry type:  MULTIPOLYGON
## dimension:      XY
## bbox:           xmin: 332390.2 ymin: 379748.5 xmax: 345636 ymax: 397980.1
## projected CRS:  Transverse_Mercator
```

Examine the input data. A spatial data frame stores a range of attributes derived from a shapefile including the **geometry** of features (e.g. polygon shape and location), **attributes** for each feature (stored in the .dbf), projection and coordinates of the shapefile's bounding box - for details, execute:

```
?st_read
```

You can employ the usual functions to visualise the content of the created data frame:

```
# visualise variable names
names(oa_shp)
```

```
##  [1] "OA_CD"    "LSOA_CD"  "MSOA_CD"  "LAD_CD"   "pop"      "H_Vbad"
##  [7] "H_bad"    "H_fair"   "H_good"   "H_Vgood"  "age_men"  "age_med"
## [13] "age_60"   "S_Rent"   "Ethnic"   "illness"  "unemp"    "males"
## [19] "geometry"
```

```
# data structure
str(oa_shp)
```

```
## Classes 'sf' and 'data.frame':   1584 obs. of  19 variables:
##  $ OA_CD   : chr  "E00176737" "E00033515" "E00033141" "E00176757" ...
##  $ LSOA_CD : chr  "E01033761" "E01006614" "E01006546" "E01006646" ...
```

```
##  $ MSOA_CD : chr  "E02006932" "E02001358" "E02001365" "E02001369" ...
##  $ LAD_CD  : chr  "E08000012" "E08000012" "E08000012" "E08000012" ...
##  $ pop     : int  185 281 208 200 321 187 395 320 316 214 ...
##  $ H_Vbad  : int  1 2 3 7 4 4 5 9 5 4 ...
##  $ H_bad   : int  2 20 10 8 10 25 19 22 25 17 ...
##  $ H_fair  : int  9 47 22 17 32 70 42 53 55 39 ...
##  $ H_good  : int  53 111 71 52 112 57 131 104 104 53 ...
##  $ H_Vgood : int  120 101 102 116 163 31 198 132 127 101 ...
##  $ age_men : num  27.9 37.7 37.1 33.7 34.2 ...
##  $ age_med : num  25 36 32 29 34 53 23 30 34 29 ...
##  $ age_60  : num  0.0108 0.1637 0.1971 0.1 0.1402 ...
##  $ S_Rent  : num  0.0526 0.176 0.0235 0.2222 0.0222 ...
##  $ Ethnic  : num  0.3514 0.0463 0.0192 0.215 0.0779 ...
##  $ illness : int  185 281 208 200 321 187 395 320 316 214 ...
##  $ unemp   : num  0.0438 0.121 0.1121 0.036 0.0743 ...
##  $ males   : int  122 128 95 120 158 123 207 164 157 94 ...
##  $ geometry:sfc_MULTIPOLYGON of length 1584; first list element: List of 1
##   ..$ :List of 1
##   .. ..$ : num [1:14, 1:2] 335106 335130 335164 335173 335185 ...
##   ..- attr(*, "class")= chr [1:3] "XY" "MULTIPOLYGON" "sfg"
##  - attr(*, "sf_column")= chr "geometry"
##  - attr(*, "agr")= Factor w/ 3 levels "constant","aggregate",..: NA NA NA NA NA NA
##   ..- attr(*, "names")= chr [1:18] "OA_CD" "LSOA_CD" "MSOA_CD" "LAD_CD" ...
```

```
# see first few observations
head(oa_shp)
```

```
## Simple feature collection with 6 features and 18 fields
## geometry type:  MULTIPOLYGON
## dimension:      XY
## bbox:           xmin: 335071.6 ymin: 389876.7 xmax: 339426.9 ymax: 394479
## projected CRS:  Transverse_Mercator
##       OA_CD   LSOA_CD   MSOA_CD    LAD_CD pop H_Vbad H_bad H_fair H_good
## 1 E00176737 E01033761 E02006932 E08000012 185      1     2      9     53
## 2 E00033515 E01006614 E02001358 E08000012 281      2    20     47    111
## 3 E00033141 E01006546 E02001365 E08000012 208      3    10     22     71
## 4 E00176757 E01006646 E02001369 E08000012 200      7     8     17     52
## 5 E00034050 E01006712 E02001375 E08000012 321      4    10     32    112
## 6 E00034280 E01006761 E02001366 E08000012 187      4    25     70     57
##   H_Vgood  age_men age_med     age_60     S_Rent     Ethnic illness      unemp
## 1     120 27.94054      25 0.01081081 0.05263158 0.35135135     185 0.04379562
## 2     101 37.71174      36 0.16370107 0.17600000 0.04626335     281 0.12101911
## 3     102 37.08173      32 0.19711538 0.02352941 0.01923077     208 0.11214953
## 4     116 33.73000      29 0.10000000 0.22222222 0.21500000     200 0.03597122
## 5     163 34.19003      34 0.14018692 0.02222222 0.07788162     321 0.07428571
## 6      31 56.09091      53 0.44919786 0.88524590 0.11764706     187 0.44615385
```

```
##   males                       geometry
## 1   122 MULTIPOLYGON (((335106.3 38...
## 2   128 MULTIPOLYGON (((335810.5 39...
## 3    95 MULTIPOLYGON (((336738 3931...
## 4   120 MULTIPOLYGON (((335914.5 39...
## 5   158 MULTIPOLYGON (((339325 3914...
## 6   123 MULTIPOLYGON (((338198.1 39...
```

TASK:

- What are the geographical hierarchy in these data?
- What is the smallest geography?
- What is the largest geography?

4.10.2 Basic Mapping

Again, many functions exist in CRAN for creating maps:

- `plot` to create static maps
- `tmap` to create static and interactive maps
- `leaflet` to create interactive maps
- `mapview` to create interactive maps
- `ggplot2` to create data visualisations, including static maps
- `shiny` to create web applications, including maps

Here this notebook demonstrates the use of `plot` and `tmap`. First `plot` is used to map the spatial distribution of non-British-born population in Liverpool. First we only map the geometries on the right,

4.10.2.1 Using `plot`

```
# mapping geometry
plot(st_geometry(oa_shp))
```

and then:

```
# map attributes, adding intervals
plot(oa_shp["Ethnic"], key.pos = 4, axes = TRUE, key.width = lcm(1.3), key.length = 1.,
     breaks = "jenks", lwd = 0.1, border = 'grey')
```

TASK:

- What is the key pattern emerging from this map?

4.10.2.2 Using `tmap`

Similar to `ggplot2`, `tmap` is based on the idea of a 'grammar of graphics' which involves a separation between the input data and aesthetics (i.e. the way data are visualised). Each data set can be mapped in various different ways, including

Figure 4.5: OAs of Livepool

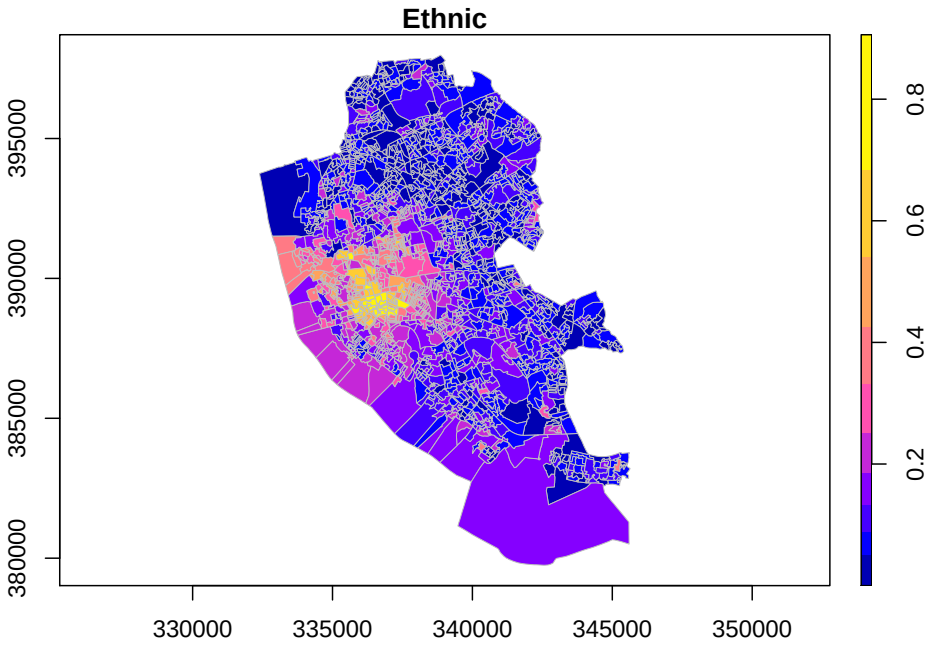
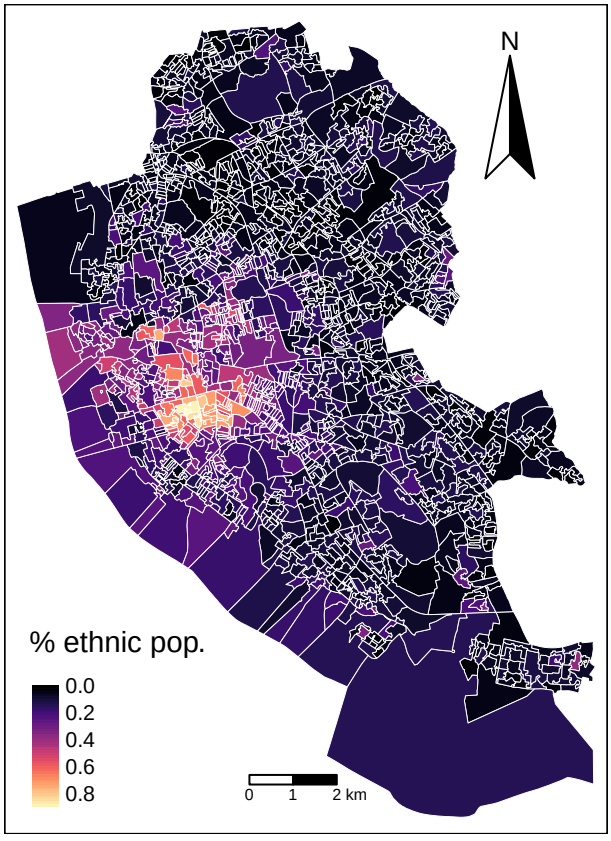


Figure 4.6: Spatial distribution of ethnic groups, Liverpool

location as defined by its geometry, colour and other features. The basic building block is `tm_shape()` (which defines input data), followed by one or more layer elements such as `tm_fill()` and `tm_dots()`.

```
# ensure geometry is valid
oa_shp = sf::st_make_valid(oa_shp)

# map
legend_title = expression("% ethnic pop.")
map_oa = tm_shape(oa_shp) +
  tm_fill(col = "Ethnic", title = legend_title, palette = magma(256), style = "cont") + # add fil
  tm_borders(col = "white", lwd = .01)  + # add borders
  tm_compass(type = "arrow", position = c("right", "top") , size = 4) + # add compass
  tm_scale_bar(breaks = c(0,1,2), text.size = 0.5, position =  c("center", "bottom")) # add scale
map_oa
```



Note that the operation `+` is used to add new layers. You can set style themes by `tm_style`. To visualise the existing styles use `tmap_style_catalogue()`, and you can also evaluate the code chunk below if you would like to create an interactive map.

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

TASK:

- Try mapping other variables in the spatial data frame. Where do population aged 60 and over concentrate?

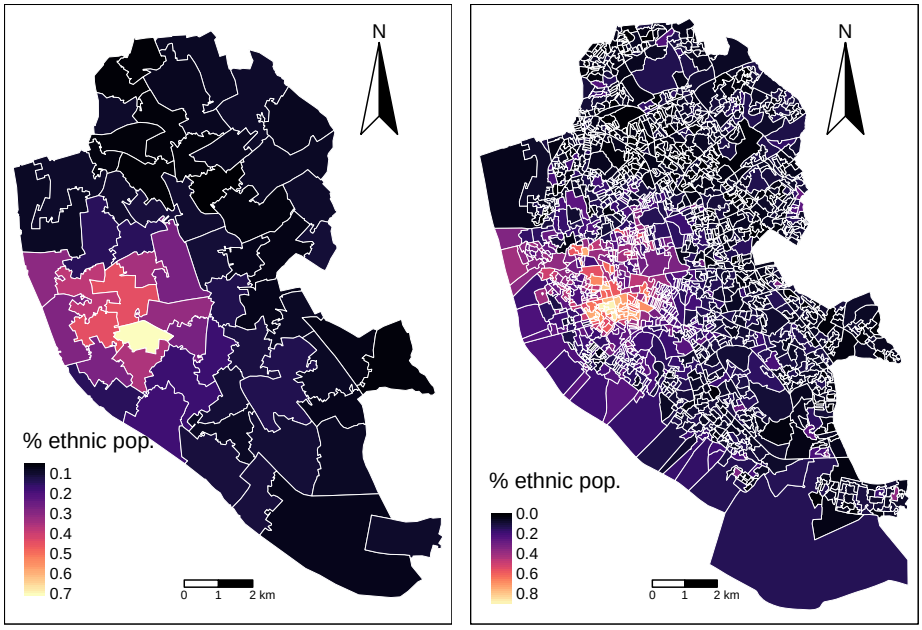
4.10.3 Comparing geographies

If you recall, one of the key issues of working with spatial data is the modifiable area unit problem (MAUP) - see lecture notes. To get a sense of the effects of MAUP, we analyse differences in the spatial patterns of the ethnic population in Liverpool between Middle Layer Super Output Areas (MSOAs) and OAs. So we map these geographies together.

```
# read data at the msoa level
msoa_shp <- st_read("data/census/Liverpool_MSOA.shp")
```

```
## Reading layer `Liverpool_MSOA' from data source `/home/rstudio/Documents/data/census
## Simple feature collection with 61 features and 16 fields
## geometry type:  MULTIPOLYGON
## dimension:      XY
## bbox:           xmin: 333086.1 ymin: 381426.3 xmax: 345636 ymax: 397980.1
## projected CRS:  Transverse_Mercator
```

```
# ensure geometry is valid
msoa_shp = sf::st_make_valid(msoa_shp)

# create a map
map_msoa = tm_shape(msoa_shp) +
  tm_fill(col = "Ethnic", title = legend_title, palette = magma(256), style = "cont") +
  tm_borders(col = "white", lwd = .01)  +
  tm_compass(type = "arrow", position = c("right", "top") , size = 4) +
  tm_scale_bar(breaks = c(0,1,2), text.size = 0.5, position =  c("center", "bottom"))

# arrange maps
tmap_arrange(map_msoa, map_oa)
```



TASK:

- What differences do you see between OAs and MSOAs?
- Can you identify areas of spatial clustering? Where are they?

4.11 Useful Functions

Function	Description
read.csv()	read csv files into data frames
str()	inspect data structure
mutate()	create a new variable
filter()	filter observations based on variable values
%>%	pipe operator - chain operations
select()	select variables
merge()	join dat frames
st_read	read spatial data (ie. shapefiles)
plot()	create a map based a spatial data set
tm_shape(), tm_fill(), tm_borders()	create a map using tmap functions
tm_arrange	display multiple maps in a single “metaplot”

Chapter 5

Points

This chapter[1] is based on the following references, which are great follow-up's on the topic:

- Lovelace and Cheshire (2014) is a great introduction.
- Chapter 6 of Brunsdon and Comber (2015), in particular subsections 6.3 and 6.7.
- Bivand et al. (2013) provides an in-depth treatment of spatial data in R.

This chapter is part of Spatial Analysis Notes, a compilation hosted as a GitHub repository that you can access it in a few ways:

- As a download of a `.zip` file that contains all the materials.
- As an html website.
- As a pdf document
- As a GitHub repository.

5.1 Dependencies

This tutorial relies on the following libraries that you will need to have installed on your machine to be able to interactively follow along[2]. Once installed, load them up with the following commands:

```
# For pretty table
library(knitr)
```

[1] This chapter is part of Spatial Analysis Notes Points – Kernel Density Estimation and Spatial interpolation by Dani Arribas-Bel is licensed under a Creative Commons Attribution-NonCommercial-ShareAlike 4.0 International License.

[2] You can install package `mypackage` by running the command `install.packages("mypackage")` on the R prompt or through the `Tools --> Install Packages...` menu in RStudio.

```
# Spatial Data management
library(rgdal)
```

```
## Loading required package: sp
```

```
## rgdal: version: 1.5-18, (SVN revision 1082)
## Geospatial Data Abstraction Library extensions to R successfully loaded
## Loaded GDAL runtime: GDAL 3.0.4, released 2020/01/28
## Path to GDAL shared files: /usr/share/gdal
## GDAL binary built with GEOS: TRUE
## Loaded PROJ runtime: Rel. 6.3.1, February 10th, 2020, [PJ_VERSION: 631]
## Path to PROJ shared files: /usr/share/proj
## Linking to sp version:1.4-4
## To mute warnings of possible GDAL/OSR exportToProj4() degradation,
## use options("rgdal_show_exportToProj4_warnings"="none") before loading rgdal.
```

```
# Pretty graphics
library(ggplot2)
# Thematic maps
library(tmap)
# Pretty maps
library(ggmap)
```

```
## Google's Terms of Service: https://cloud.google.com/maps-platform/terms/.
```

```
## Please cite ggmap if you use it! See citation("ggmap") for details.
```

```
# Various GIS utilities
library(GISTools)
```

```
## Loading required package: maptools
```

```
## Checking rgeos availability: TRUE
```

```
## Loading required package: RColorBrewer
```

```
## Loading required package: MASS
```

```
## Loading required package: rgeos
```

```
## rgeos version: 0.5-5, (SVN revision 640)
##  GEOS runtime version: 3.8.0-CAPI-1.13.1
##  Linking to sp version: 1.4-4
##  Polygon checking: TRUE
```

```
# For all your interpolation needs
library(gstat)
# For data manipulation
library(plyr)
```

Before we start any analysis, let us set the path to the directory where we are

working. We can easily do that with `setwd()`. Please replace in the following line the path to the folder where you have placed this file -and where the `house_transactions` folder with the data lives.

```
setwd('.')
```

5.2 Data

For this session, we will use a subset of residential property transaction data for the city of Liverpool. These are provided by the Land Registry (as part of their Price Paid Data) but have been cleaned and re-packaged by Dani Arribas-Bel.

Let us start by reading the data, which comes in a shapefile:

```
db <- readOGR(dsn = 'data/house_transactions', layer = 'liv_house_trans')
```

```
## Warning in OGRSpatialRef(dsn, layer, morphFromESRI = morphFromESRI, dumpSRS =
## dumpSRS, : Discarded datum OSGB_1936 in CRS definition: +proj=tmerc +lat_0=49
## +lon_0=-2 +k=0.9996012717 +x_0=400000 +y_0=-100000 +ellps=airy +units=m +no_defs
```

```
## OGR data source with driver: ESRI Shapefile
## Source: "/home/rstudio/Documents/data/house_transactions", layer: "liv_house_trans"
## with 6324 features
## It has 18 fields
## Integer64 fields read as strings:  price
```

Before we forget, let us make sure `price` is considered a number, not a factor:

```
db@data$price <- as.numeric(as.character((db@data$price)))
```

The dataset spans the year 2014:

```
# Format dates
dts <- as.Date(db@data$trans_date)
# Set up summary table
tab <- summary(dts)
tab
```

```
##         Min.      1st Qu.       Median         Mean      3rd Qu.         Max.
## "2014-01-02" "2014-04-11" "2014-07-09" "2014-07-08" "2014-10-03" "2014-12-30"
```

We can then examine the elements of the object with the `summary` method:

```
summary(db)
```

```
## Object of class SpatialPointsDataFrame
## Coordinates:
##              min    max
## coords.x1 333536 345449
## coords.x2 382684 397833
```

```
## Is projected: TRUE
## proj4string :
## [+proj=tmerc +lat_0=49 +lon_0=-2 +k=0.9996012717 +x_0=400000
## +y_0=-100000 +ellps=airy +units=m +no_defs]
## Number of points: 6324
## Data attributes:
##      pcds                id                price           trans_date
##  Length:6324        Length:6324        Min.   :    1000   Length:6324
##  Class :character   Class :character   1st Qu.:   70000   Class :character
##  Mode  :character   Mode  :character   Median :  110000   Mode  :character
##                                        Mean   :  144310
##                                        3rd Qu.:  160000
##                                        Max.   :26615720
##      type               new              duration            paon
##  Length:6324        Length:6324        Length:6324        Length:6324
##  Class :character   Class :character   Class :character   Class :character
##  Mode  :character   Mode  :character   Mode  :character   Mode  :character
##
##
##
##      saon               street            locality            town
##  Length:6324        Length:6324        Length:6324        Length:6324
##  Class :character   Class :character   Class :character   Class :character
##  Mode  :character   Mode  :character   Mode  :character   Mode  :character
##
##
##
##    district            county            ppd_cat             status
##  Length:6324        Length:6324        Length:6324        Length:6324
##  Class :character   Class :character   Class :character   Class :character
##  Mode  :character   Mode  :character   Mode  :character   Mode  :character
##
##
##
##     lsoa11            LSOA11CD
##  Length:6324        Length:6324
##  Class :character   Class :character
##  Mode  :character   Mode  :character
##
##
##
```

See how it contains several pieces, some relating to the spatial information, some relating to the tabular data attached to it. We can access each of the separately if we need it. For example, to pull out the names of the columns in the `data.frame`, we can use the `@data` appendix:

```
colnames(db@data)
```

```
##  [1] "pcds"       "id"         "price"      "trans_date" "type"
##  [6] "new"        "duration"   "paon"       "saon"       "street"
## [11] "locality"   "town"       "district"   "county"     "ppd_cat"
## [16] "status"     "lsoa11"     "LSOA11CD"
```

The rest of this session will focus on two main elements of the shapefile: the spatial dimension (as stored in the point coordinates), and the house price values contained in the `price` column. To get a sense of what they look like first, let us plot both. We can get a quick look at the non-spatial distribution of house values with the following commands:

```
# Create the histogram
hist <- qplot(data=db@data,x=price)
hist
```

```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```

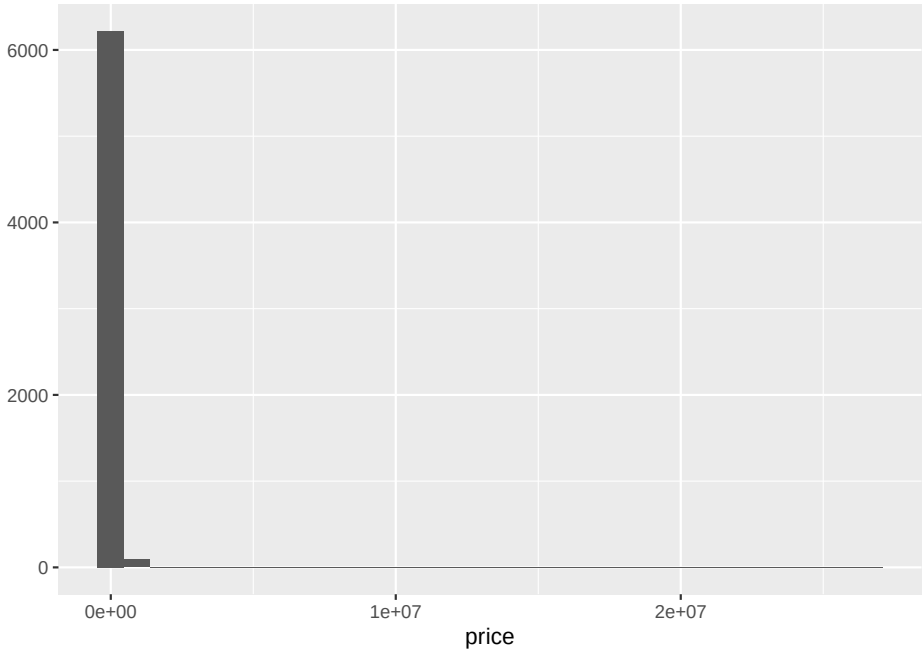


Figure 5.1: Raw house prices in Liverpool

This basically shows there is a lot of values concentrated around the lower end of the distribution but a few very large ones. A usual transformation to *shrink* these differences is to take logarithms:

```
# Create log and add it to the table
logpr <- log(as.numeric(db@data$price))
db@data['logpr'] <- logpr
# Create the histogram
hist <- qplot(x=logpr)
hist
```

```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```

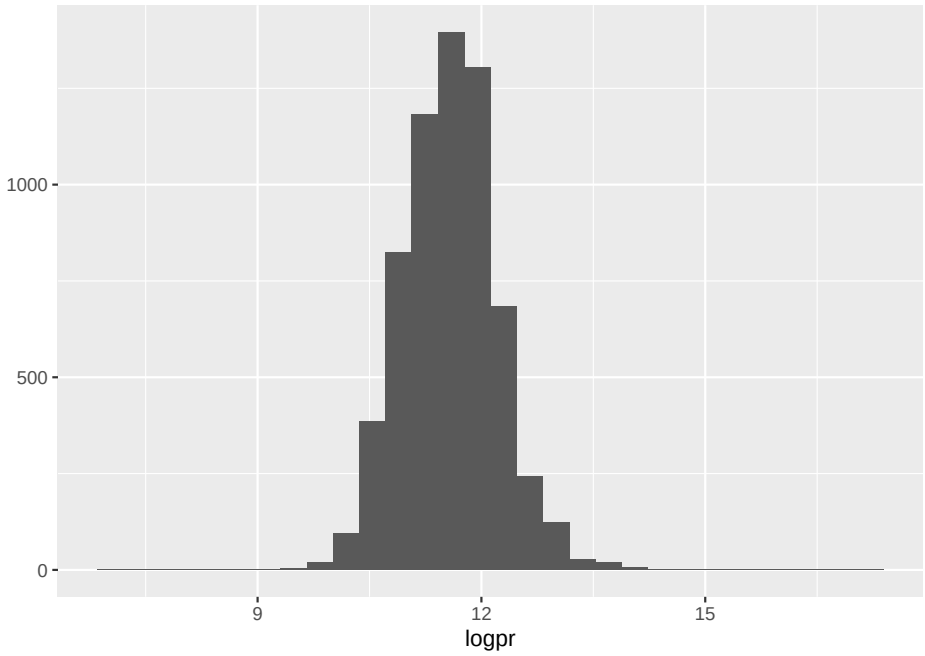


Figure 5.2: Log of house price in Liverpool

To obtain the spatial distribution of these houses, we need to turn away from the `@data` component of `db`. The easiest, quickest (and also dirtiest) way to get a sense of what the data look like over space is using `plot`:

```
plot(db)
```

5.3 KDE

Kernel Density Estimation (KDE) is a technique that creates a *continuous* representation of the distribution of a given variable, such as house prices. Although theoretically it can be applied to any dimension, usually, KDE is applied to either one or two dimensions.

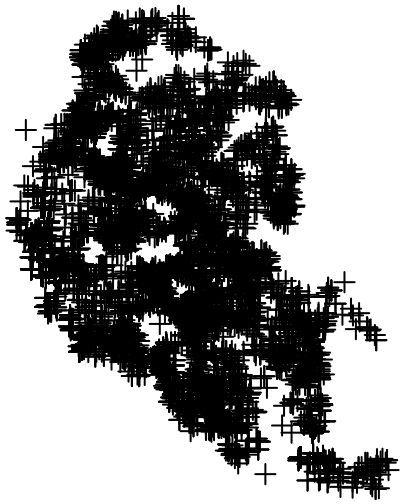


Figure 5.3: Spatial distribution of house transactions in Liverpool

5.3.1 One-dimensional KDE

KDE over a single dimension is essentially a contiguous version of a histogram. We can see that by overlaying a KDE on top of the histogram of logs that we have created before:

```
# Create the base
base <- ggplot(db@data, aes(x=logpr))
# Histogram
hist <- base +
  geom_histogram(bins=50, aes(y=..density..))
# Overlay density plot
kde <- hist +
  geom_density(fill="#FF6666", alpha=0.5, colour="#FF6666")
kde
```

The key idea is that we are smoothing out the discrete binning that the histogram involves. Note how the histogram is exactly the same as above shapewise, but it has been rescalend on the Y axis to reflect probabilities rather than counts.

5.3.2 Two-dimensional (spatial) KDE

Geography, at the end of the day, is usually represented as a two-dimensional space where we locate objects using a system of dual coordinates, `X` and `Y` (or latitude and longitude). Thanks to that, we can use the same technique as above to obtain a smooth representation of the distribution of a two-dimensional variable. The crucial difference is that, instead of obtaining a curve as the

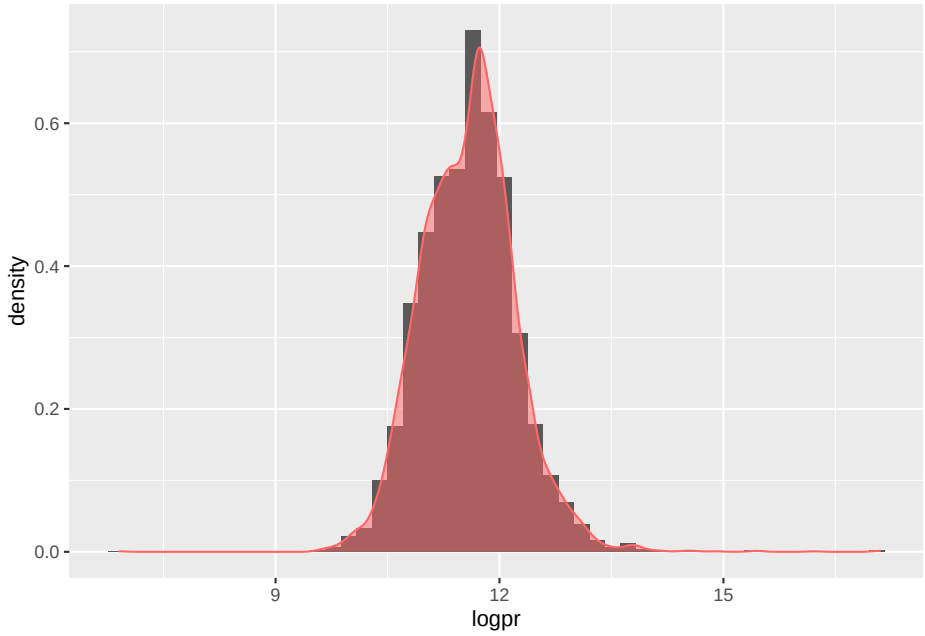


Figure 5.4: Histogram and KDE of the log of house prices in Liverpool

output, we will create a *surface*, where intensity will be represented with a color gradient, rather than with the second dimension, as it is the case in the figure above.

To create a spatial KDE in R, there are several ways. If you do not want to necessarily acknowledge the spatial nature of your data, or you they are not stored in a spatial format, you can plot them using `ggplot2`. Note we first need to convert the coordinates (stored in the spatial part of `db`) into columns of X and Y coordinates, then we can plot them:

```
# Attach XY coordinates
db@data['X'] <- db@coords[, 1]
db@data['Y'] <- db@coords[, 2]
# Set up base layer
base <- ggplot(data=db@data, aes(x=X, y=Y))
# Create the KDE surface
kde <- base + stat_density2d(aes(x = X, y = Y, alpha = ..level..),
               size = 0.01, bins = 16, geom = 'polygon') +
           scale_fill_gradient()
kde
```

Or, we can use a package such as the `GISTools`, which allows to pass a spatial object directly:

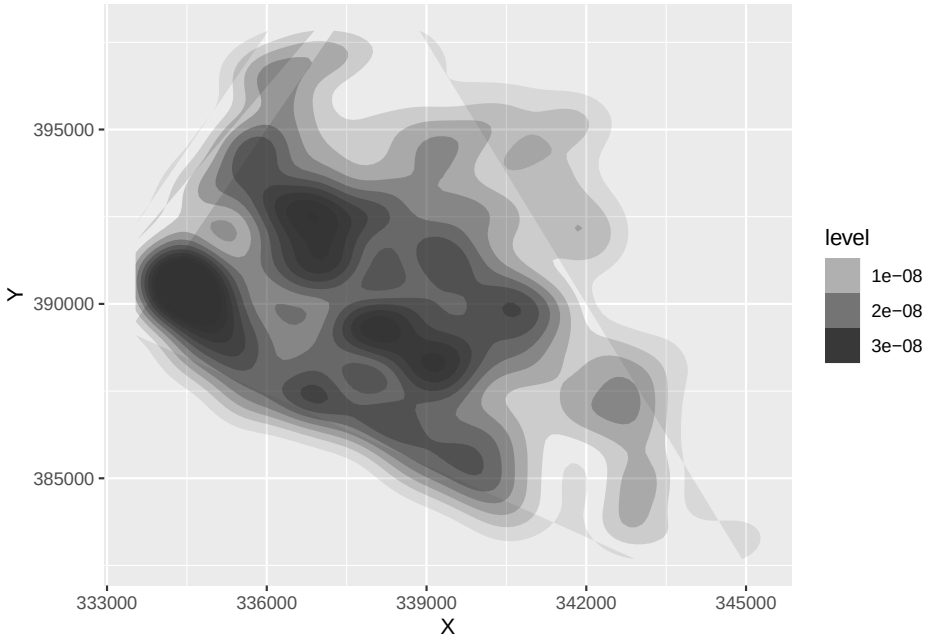


Figure 5.5: KDE of house transactions in Liverpool

```
# Compute the KDE
kde <- kde.points(db)
```

```
## Warning in proj4string(pts): CRS object has comment, which is lost in output
```

```
## Warning in showSRID(uprojargs, format = "PROJ", multiline = "NO", prefer_proj
## = prefer_proj): Discarded datum Unknown based on Airy 1830 ellipsoid in CRS
## definition
```

```
# Plot the KDE
level.plot(kde)
```

Either of these approaches generate a surface that represents the density of dots, that is an estimation of the probability of finding a house transaction at a given coordinate. However, without any further information, they are hard to interpret and link with previous knowledge of the area. To bring such context to the figure, we can plot an underlying basemap, using a cloud provider such as Google Maps or, as in this case, OpenStreetMap. To do it, we will leverage the library `ggmap`, which is designed to play nicely with the `ggplot2` family (hence the seemingly counterintuitive example above). Before we can plot them with the online map, we need to reproject them though.

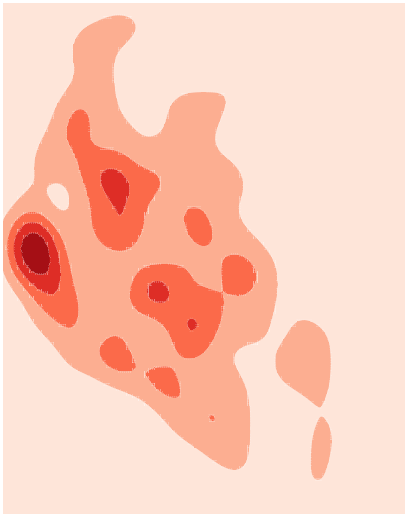


Figure 5.6: KDE of house transactions in Liverpool

```
# Reproject coordinates
wgs84 <- CRS("+proj=longlat +datum=WGS84 +ellps=WGS84 +towgs84=0,0,0")
db_wgs84 <- spTransform(db, wgs84)
db_wgs84@data['lon'] <- db_wgs84@coords[, 1]
db_wgs84@data['lat'] <- db_wgs84@coords[, 2]
xys <- db_wgs84@data[c('lon', 'lat')]
# Bounding box
liv <- c(left = min(xys$lon), bottom = min(xys$lat),
         right = max(xys$lon), top = max(xys$lat))
# Download map tiles
basemap <- get_stamenmap(liv, zoom = 12,
                         maptype = "toner-lite")
```

```
## Source : http://tile.stamen.com/toner-lite/12/2013/1325.png
## Source : http://tile.stamen.com/toner-lite/12/2014/1325.png
## Source : http://tile.stamen.com/toner-lite/12/2015/1325.png
## Source : http://tile.stamen.com/toner-lite/12/2013/1326.png
## Source : http://tile.stamen.com/toner-lite/12/2014/1326.png
## Source : http://tile.stamen.com/toner-lite/12/2015/1326.png
## Source : http://tile.stamen.com/toner-lite/12/2013/1327.png
## Source : http://tile.stamen.com/toner-lite/12/2014/1327.png
## Source : http://tile.stamen.com/toner-lite/12/2015/1327.png
```

```
# Overlay KDE
final <- ggmap(basemap, extent = "device",
               maprange=FALSE) +
  stat_density2d(data = db_wgs84@data,
                 aes(x = lon, y = lat,
                     alpha=..level..,
                     fill = ..level..),
                 size = 0.01, bins = 16,
                 geom = 'polygon',
                 show.legend = FALSE) +
  scale_fill_gradient2("Transaction\nDensity",
                       low = "#ffffff8",
                       high = "#8da0cb")
final
```

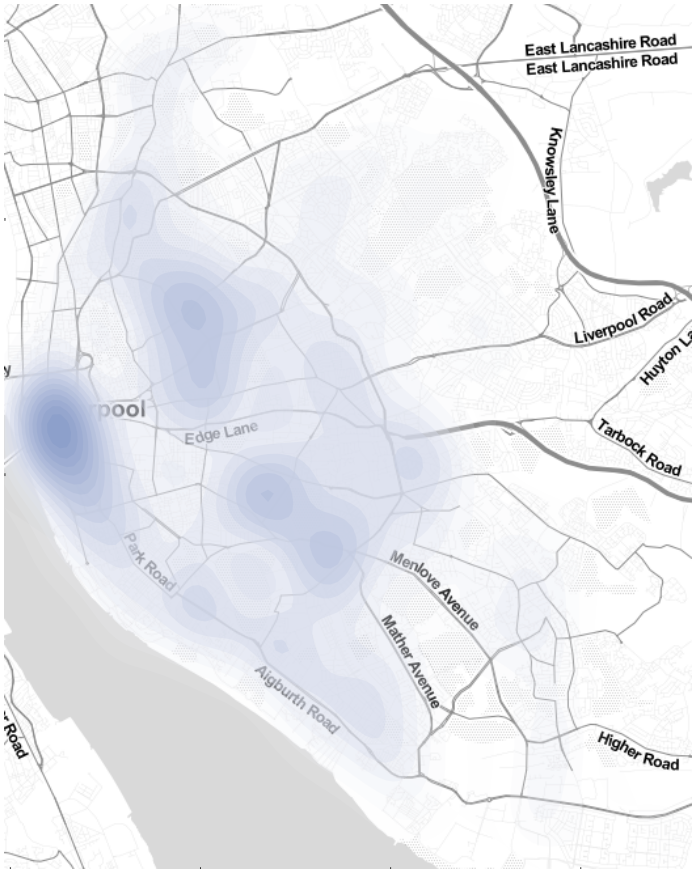


Figure 5.7: KDE of house transactions in Liverpool

The plot above[3] allows us to not only see the distribution of house transactions, but to relate it to what we know about Liverpool, allowing us to establish many more connections than we were previously able. Mainly, we can easily see that the area with a highest volume of houses being sold is the city centre, with a "hole" around it that displays very few to no transactions and then several pockets further away.

5.4 Spatial Interpolation

The previous section demonstrates how to visualize the distribution of a set of spatial objects represented as points. In particular, given a bunch of house transactions, it shows how one can effectively visualize their distribution over space and get a sense of the density of occurrences. Such visualization, because it is based on KDE, is based on a smooth continuum, rather than on a discrete approach (as a choropleth would do, for example).

Many times however, we are not particularly interested in learning about the density of occurrences, but about the distribution of a given value attached to each location. Think for example of weather stations and temperature: the location of the stations is no secret and rarely changes, so it is not of particular interest to visualize the density of stations; what we are usually interested instead is to know how temperature is distributed over space, given we only measure it in a few places. One could argue the example we have been working with so far, house price transactions, fits into this category as well: although where house are sold may be of relevance, more often we are interested in finding out what the "surface of price" looks like. Rather than *where are most houses being sold?* we usually want to know *where the most expensive or most affordable* houses are located.

In cases where we are interested in creating a surface of a given value, rather than a simple density surface of occurrences, KDE cannot help us. In these cases, what we are interested in is *spatial interpolation*, a family of techniques that aim at exactly that: creating continuous surfaces for a particular phenomenon (e.g. temperature, house prices) given only a finite sample of observations. Spatial interpolation is a large field of research that is still being actively developed and that can involve a substantial amount of mathematical complexity in order to obtain the most accurate estimates possible[4]. In this session, we will introduce the simplest possible way of interpolating values, hoping this will give you a general understanding of the methodology and, if you are interested, you can check out further literature. For example, Banerjee et al. (2014) or Cressie (2015) are hard but good overviews.

[3] **EXERCISE** The map above uses the Stamen map `toner-lite`. Explore additional tile styles on their website and try to recreate the plot above.

[4] There is also an important economic incentive to do this: some of the most popular applications are in the oil and gas or mining industries. In fact, the very creator of this technique, Danie G. Krige, was a mining engineer. His name is usually used to nickname spatial interpolation as *kriging*.

5.4.1 Inverse Distance Weight (IDW) interpolation

The technique we will cover here is called *Inverse Distance Weighting*, or IDW for convenience. Brunsdon and Comber (2015) offer a good description:

> In the *inverse distance weighting* (IDW) approach to interpolation, to estimate the value of z at location x a weighted mean of nearby observations is taken [...]. To accommodate the idea that observations of z at points closer to x should be given more importance in the interpolation, greater weight is given to these points [...]
>
> — Page 204

The math[5] is not particularly complicated and may be found in detail elsewhere (the reference above is a good starting point), so we will not spend too much time on it. More relevant in this context is the intuition behind. Essentially, the idea is that we will create a surface of house price by smoothing many values arranged along a regular grid and obtained by interpolating from the known locations to the regular grid locations. This will give us full and equal coverage to soundly perform the smoothing.

Enough chat, let's code.

From what we have just mentioned, there are a few steps to perform an IDW spatial interpolation:

1. Create a regular grid over the area where we have house transactions.
2. Obtain IDW estimates for each point in the grid, based on the values of k nearest neighbors.
3. Plot a smoothed version of the grid, effectively representing the surface of house prices.

Let us go in detail into each of them[6]. First, let us set up a grid:

```
liv.grid <- spsample(db, type='regular', n=25000)
```

That's it, we're done! The function `spsample` hugely simplifies the task by taking a spatial object and returning the grid we need. Not a couple of additional arguments we pass: `type` allows us to get a set of points that are *uniformly* distributed over space, which is handy for the later smoothing; `n` controls how many points we want to create in that grid.

On to the IDW. Again, this is hugely simplified by `gstat`:

```
idw.hp <- idw(price ~ 1, locations=db, newdata=liv.grid)
```

[5] Essentially, for any point x in space, the IDW estimate for value z is equivalent to $\hat{z}(x) = \frac{\sum_i w_i z_i}{\sum_i w_i}$ where i are the observations for which we do have a value, and w_i is a weight given to location i based on its distance to x.

[6] For the relevant calculations, we will be using the `gstat` library.

Boom! We've got it. Let us pause for a second to see how we just did it. First, we pass `price ~ 1`. This specifies the formula we are using to model house prices. The name on the left of `~` represents the variable we want to explain, while everything to its right captures the *explanatory* variables. Since we are considering the simplest possible case, we do not have further variables to add, so we simply write `1`. Then we specify the original locations for which we do have house prices (our original `db` object), and the points where we want to interpolate the house prices (the `liv.grid` object we just created above). One more note: by default, `idw.hp` uses all the available observations, weighted by distance, to provide an estimate for a given point. If you want to modify that and restrict the maximum number of neighbors to consider, you need to tweak the argument `nmax`, as we do above by using the 150 neares observations to each point[7].

The object we get from `idw` is another spatial table, just as `db`, containing the interpolated values. As such, we can inspect it just as with any other of its kind. For example, to check out the top of the estimated table:

```
head(idw.hp@data)
```

The column we will pay attention to is `var1.pred`. And to see the locations for which those correspond:

```
head(idw.hp@coords)
```

So, for a hypothetical house sold at the location in the first row of `idw.hp@coords` (expressed in the OSGB coordinate system), the price we would guess it would cost, based on the price of houses sold nearby, is the first element of column `var1.pred` in `idw.hp@data`.

5.4.2 A surface of housing prices

Once we have the IDW object computed, we can plot it to explore the distribution, not of house transactions in this case, but of house price over the geography of Liverpool. The easiest way to do this is by quickly calling the command `spplot`:

```
spplot(idw.hp['var1.pred'])
```

However, this is not entirely satisfactory for a number of reasons. Let us get an equivalen plot with the package `tmap`, which streamlines some of this and makes more aesthetically pleasant maps easier to build as it follows a "ggplot-y" approach.

```
# Load up the layer
liv.otl <- readOGR('data/house_transactions', 'liv_outline')
```

The shape we will overlay looks like this:

[7] Have a play with this because the results do change significantly. Can you reason why?

```
qtm(liv.otl)
```

Now let's give it a first go!

```
#
p = tm_shape(liv.otl) + tm_fill(col='black', alpha=1) +
  tm_shape(idw.hp) +
  tm_symbols(col='var1.pred', size=0.1, alpha=0.25,
             border.lwd=0., palette='YlGn')
p
```

The last two plots, however, are not really a surface, but a representation of the points we have just estimated. To create a surface, we need to do an interim transformation to convert the spatial object `idw.hp` into a table that a "surface plotter" can understand.

```
xyz <- data.frame(x=coordinates(idw.hp)[, 1],
                  y=coordinates(idw.hp)[, 2],
                  z=idw.hp$var1.pred)
```

Now we are ready to plot the surface as a contour:

```
base <- ggplot(data=xyz, aes(x=x, y=y))
surface <- base + geom_contour(aes(z=z))
surface
```

Which can also be shown as a filled contour:

```
base <- ggplot(data=xyz, aes(x=x, y=y))
surface <- base + geom_raster(aes(fill=z))
surface
```

The problem here, when compared to the KDE above for example, is that a few values are extremely large:

```
qplot(data=xyz, x=z, geom='density')
```

Let us then take the logarithm before we plot the surface:

```
xyz['lz'] <- log(xyz$z)
base <- ggplot(data=xyz, aes(x=x, y=y))
surface <- base +
           geom_raster(aes(fill=lz),
                       show.legend = F)
surface
```

Now this looks better. We can start to tell some patterns. To bring in context, it would be great to be able to add a basemap layer, as we did for the KDE. This is conceptually very similar to what we did above, starting by reprojecting the points and continuing by overlaying them on top of the basemap. However,

technically speaking it is not possible because `ggmap` –the library we have been using to display tiles from cloud providers– does not play well with our own rasters (i.e. the price surface). At the moment, it is surprisingly tricky to get this to work, so we will park it for now. However, developments such as the `sf` project promise to make this easier in the future[8].

5.4.3 *"What should the next house's price be?"*

The last bit we will explore in this session relates to prediction for new values. Imagine you are a real state data scientist and your boss asks you to give an estimate of how much a new house going into the market should cost. The only information you have to make such a guess is the location of the house. In this case, the IDW model we have just fitted can help you. The trick is realizing that, instead of creating an entire grid, all we need is to obtain an estimate of a single location.

Let us say, the house is located on the coordinates `x=340000, y=390000` as expressed in the GB National Grid coordinate system. In that case, we can do as follows:

```
pt <- SpatialPoints(cbind(x=340000, y=390000),
                    proj4string = db@proj4string)
idw.one <- idw(price ~ 1, locations=db, newdata=pt)
idw.one
```

And, as show above, the estimated value is GBP157,099[9].

Using this predictive logic, and taking advantage of Google Maps and its geocoding capabilities, it is possible to devise a function that takes an arbitrary address in Liverpool and, based on the transactions occurred throughout 2014, provides an estimate of what the price for a property in that location could be.

```
how.much.is <- function(address, print.message=TRUE){
  # Convert the address into Lon/Lat coordinates
  # NOTE: this now requires an API key
  # https://github.com/dkahle/ggmap#google-maps-and-credentials
  ll.pt <- geocode(address)
  # Process as spatial table
  wgs84 <- CRS("+proj=longlat +datum=WGS84 +ellps=WGS84 +towgs84=0,0,0")
  ll.pt <- SpatialPoints(cbind(x=ll.pt$lon, y=ll.pt$lat),
                         proj4string = wgs84)
  # Transform Lon/Lat into OSGB
  pt <- spTransform(ll.pt, db@proj4string)
  # Obtain prediction
  idw.one <- idw(price ~ 1, locations=db, newdata=pt)
```

[8] **BONUS** if you can figure out a way to do it yourself!

[9] **PRO QUESTION** Is that house expensive or cheap, as compared to the other houses sold in this dataset? Can you figure out where the house is?

```
  price <- idw.one@data$var1.pred
  # Return predicted price
  if(print.message==T){
    writeLines(paste("\n\nBased on what surrounding properties were sold",
                     "for in 2014 a house located at", address, "would",
                     "cost",  paste("GBP", round(price), ".", sep=''), "\n\n"))
  }
  return(price)
}
```

Ready to test!

```
address <- "74 Bedford St S, Liverpool, L69 7ZT, UK"
#p <- how.much.is(address)
```

Chapter 6

Flows

This chapter[1] covers spatial interaction flows. Using open data from the city of San Francisco about trips on its bikeshare system, we will estimate spatial interaction models that try to capture and explain the variation in the amount of trips on each given route. After visualizing the dataset, we begin with a very simple model and then build complexity progressively by augmenting it with more information, refined measurements, and better modeling approaches. Throughout the chapter, we explore different ways to grasp the predictive performance of each model. We finish with a prediction example that illustrates how these models can be deployed in a real-world application.

Content is based on the following references, which are great follow-up's on the topic:

- Singleton (2017), an online short course on R for Geographic Data Science and Urban Analytics. In particular, the section on mapping flows is specially relevant here.
- The predictive checks section draws heavily from Gelman and Hill (2006a), in particular Chapters 6 and 7.

This tutorial is part of Spatial Analysis Notes, a compilation hosted as a GitHub repository that you can access in a few ways:

- As a download of a `.zip` file that contains all the materials.
- As an html website.
- As a pdf document
- As a GitHub repository.

[1] This chapter is part of Spatial Analysis Notes Flows – Exploring flows visually and through spatial interaction by Dani Arribas-Bel is licensed under a Creative Commons Attribution-NonCommercial-ShareAlike 4.0 International License.

6.1 Dependencies

This tutorial relies on the following libraries that you will need to have installed on your machine to be able to interactively follow along[2]. Once installed, load them up with the following commands:

```
# Spatial Data management
library(rgdal)
```

```
## Loading required package: sp
```

```
## rgdal: version: 1.5-18, (SVN revision 1082)
## Geospatial Data Abstraction Library extensions to R successfully loaded
## Loaded GDAL runtime: GDAL 3.0.4, released 2020/01/28
## Path to GDAL shared files: /usr/share/gdal
## GDAL binary built with GEOS: TRUE
## Loaded PROJ runtime: Rel. 6.3.1, February 10th, 2020, [PJ_VERSION: 631]
## Path to PROJ shared files: /usr/share/proj
## Linking to sp version:1.4-4
## To mute warnings of possible GDAL/OSR exportToProj4() degradation,
## use options("rgdal_show_exportToProj4_warnings"="none") before loading rgdal.
```

```
# Pretty graphics
library(ggplot2)
# Thematic maps
library(tmap)
# Pretty maps
library(ggmap)
```

```
## Google's Terms of Service: https://cloud.google.com/maps-platform/terms/.
```

```
## Please cite ggmap if you use it! See citation("ggmap") for details.
```

```
# Simulation methods
library(arm)
```

```
## Loading required package: MASS
```

```
## Loading required package: Matrix
```

```
## Loading required package: lme4
```

```
##
## arm (Version 1.11-2, built: 2020-7-27)
```

```
## Working directory is /home/rstudio/Documents
```

Before we start any analysis, let us set the path to the directory where we are working. We can easily do that with `setwd()`. Please replace in the follow-

[2] You can install package `mypackage` by running the command `install.packages("mypackage")` on the R prompt or through the `Tools --> Install Packages...` menu in RStudio.

ing line the path to the folder where you have placed this file -and where the `sf_bikes` folder with the data lives.

```
setwd('.')
```

6.2 Data

In this note, we will use data from the city of San Francisco representing bike trips on their public bike share system. The original source is the SF Open Data portal (link) and the dataset comprises both the location of each station in the Bay Area as well as information on trips (station of origin to station of destination) undertaken in the system from September 2014 to August 2015 and the following year. Since this note is about modeling and not data preparation, a cleanly reshaped version of the data, together with some additional information, has been created and placed in the `sf_bikes` folder. The data file is named `flows.geojson` and, in case you are interested, the (Python) code required to created from the original files in the SF Data Portal is also available on the `flows_prep.ipynb` notebook [url], also in the same folder.

Let us then directly load the file with all the information necessary:

```
db <- readOGR('./data/sf_bikes/flows.geojson')
```

```
## OGR data source with driver: GeoJSON
## Source: "/home/rstudio/Documents/data/sf_bikes/flows.geojson", layer: "flows"
## with 1722 features
## It has 9 fields
```

```
rownames(db@data) <- db$flow_id
db@data$flow_id <- NULL
```

Note how the interface is slightly different since we are reading a `GeoJSON` file instead of a shapefile.

The data contains the geometries of the flows, as calculated from the Google Maps API, as well as a series of columns with characteristics of each flow:

```
head(db@data)
```

```
##       dest orig straight_dist street_dist total_down total_up trips15 trips16
## 39-41   41   39      1452.201   1804.1150  11.205753 4.698162      68      68
## 39-42   42   39      1734.861   2069.1557  10.290236 2.897886      23      29
## 39-45   45   39      1255.349   1747.9928  11.015596 4.593927      83      50
## 39-46   46   39      1323.303   1490.8361   3.511543 5.038044     258     163
## 39-47   47   39       715.689    769.9189   0.000000 3.282495     127      73
## 39-48   48   39      1996.778   2740.1290  11.375186 3.841296      81      56
```

where `orig` and `dest` are the station IDs of the origin and destination, `street/straight_dist` is the distance in metres between stations measured

along the street network or as-the-crow-flies, `total_down/up` is the total downhil and climb in the trip, and `tripsXX` contains the amount of trips undertaken in the years of study.

6.3 "*Seeing*" flows

The easiest way to get a quick preview of what the data looks like spatially is to make a simple plot:

```
plot(db)
```

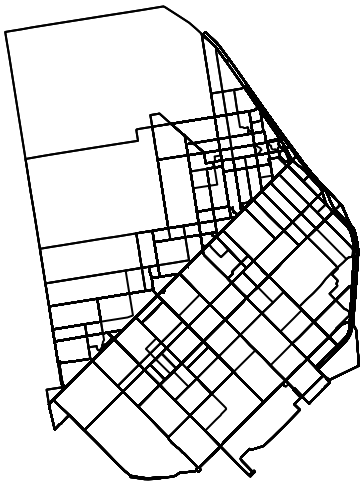


Figure 6.1: Potential routes

Equally, if we want to visualize a single route, we can simply subset the table. For example, to get the shape of the trip from station `39` to station `48`, we can:

```
one39to48 <- db[ which(
          db@data$orig == 39 & db@data$dest == 48
          ) , ]
plot(one39to48)
```

or, for the most popular route, we can:

```
most_pop <- db[ which(
          db@data$trips15 == max(db@data$trips15)
          ) , ]
plot(most_pop)
```

These however do not reveal a lot: there is no geographical context (*why are there so many routes along the NE?*) and no sense of how volumes of bikers are allocated along different routes. Let us fix those two.

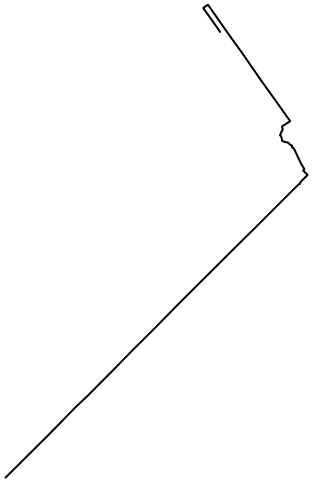


Figure 6.2: Trip from station 39 to 48

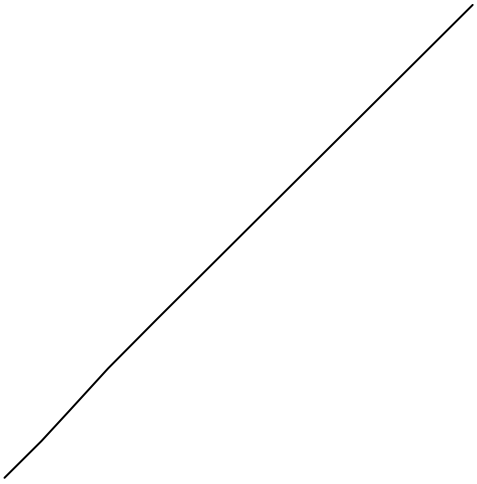


Figure 6.3: Most popular trip

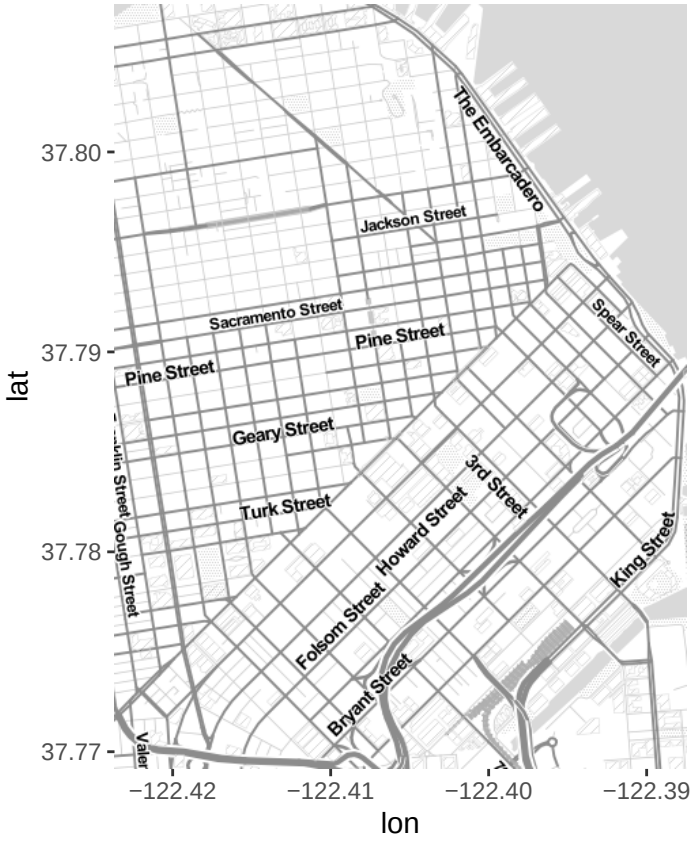
The easiest way to bring in geographical context is by overlaying the routes on top of a background map of tiles downloaded from the internet. Let us download this using `ggmap`:

```
sf_bb <- c(left=db@bbox['x', 'min'],
           right=db@bbox['x', 'max'],
           bottom=db@bbox['y', 'min'],
           top=db@bbox['y', 'max'])
SanFran <- get_stamenmap(sf_bb,
                         zoom = 14,
                         maptype = "toner-lite")
```

```
## Source : http://tile.stamen.com/toner-lite/14/2620/6330.png

## Source : http://tile.stamen.com/toner-lite/14/2621/6330.png

## Source : http://tile.stamen.com/toner-lite/14/2622/6330.png

## Source : http://tile.stamen.com/toner-lite/14/2620/6331.png

## Source : http://tile.stamen.com/toner-lite/14/2621/6331.png

## Source : http://tile.stamen.com/toner-lite/14/2622/6331.png

## Source : http://tile.stamen.com/toner-lite/14/2620/6332.png

## Source : http://tile.stamen.com/toner-lite/14/2621/6332.png

## Source : http://tile.stamen.com/toner-lite/14/2622/6332.png

## Source : http://tile.stamen.com/toner-lite/14/2620/6333.png

## Source : http://tile.stamen.com/toner-lite/14/2621/6333.png

## Source : http://tile.stamen.com/toner-lite/14/2622/6333.png
```

and make sure it looks like we intend it to look:

```
ggmap(SanFran)
```



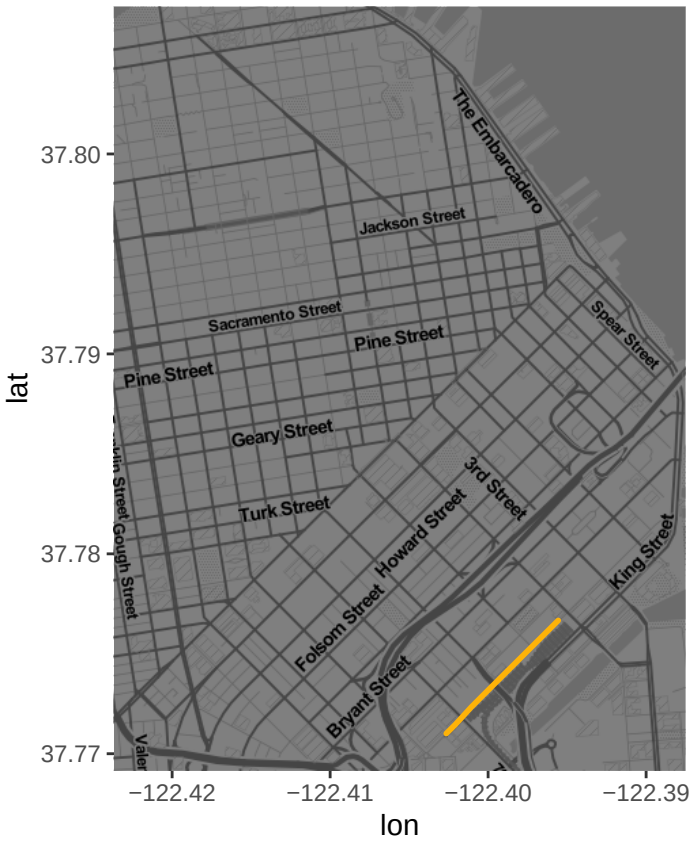
Now to combine tiles and routes, we need to pull out the coordinates that make up each line. For the route example above, this would be:

```
xys1 <- as.data.frame(coordinates(most_pop))
```

Now we can plot the route[3] (note we also dim down the background to focus the attention on flows):

```
ggmap(SanFran, darken=0.5) +
  geom_path(aes(x=X1, y=X2),
            data=xys1,
            size=1,
            color=rgb(0.996078431372549, 0.7019607843137254, 0.03137254901960784),
            lineend='round')
```

[3] **EXERCISE**: *can you plot the route for the largest climb?*



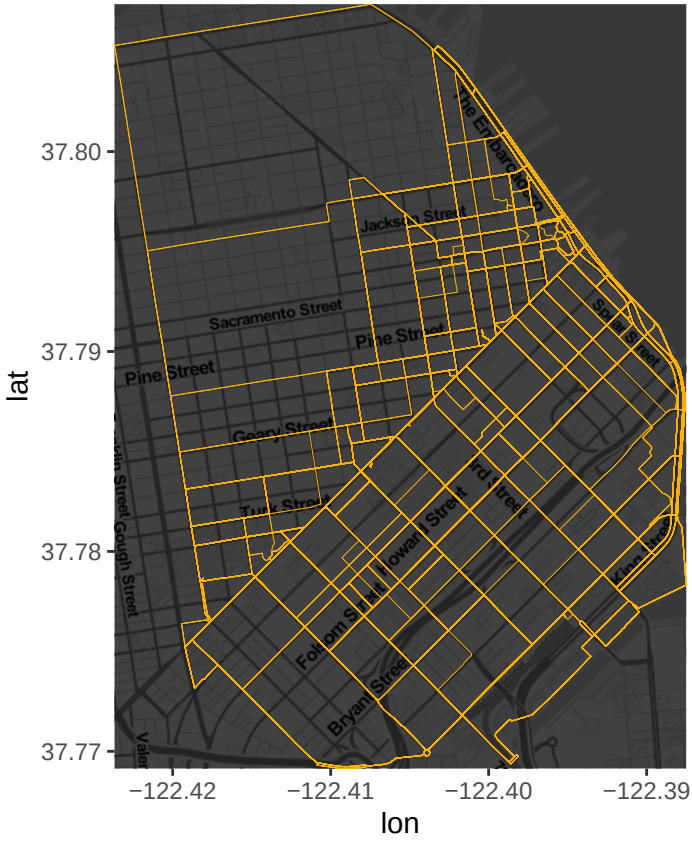
Now we can plot all of the lines by using a short `for` loop to build up the table:

```
# Set up shell data.frame
lines <- data.frame(lat = numeric(0),
                    lon = numeric(0),
                    trips = numeric(0),
                    id = numeric(0)
                    )
# Run loop
for(x in 1:nrow(db)){
  # Pull out row
  r <- db[x, ]
  # Extract lon/lat coords
  xys <- as.data.frame(coordinates(r))
  names(xys) <- c('lon', 'lat')
  # Insert trips and id
  xys['trips'] <- r@data$trips15
  xys['id'] <- x
  # Append them to `lines`
```

```
  lines <- rbind(lines, xys)
}
```

Now we can go on and plot all of them:

```
ggmap(SanFran, darken=0.75) +
  geom_path(aes(
                x=lon,
                y=lat,
                group=id
                ),
            data=lines,
            size=0.1,
            color=rgb(0.996078431372549, 0.7019607843137254, 0.03137254901960784),
            lineend='round')
```



Finally, we can get a sense of the distribution of the flows by associating a color gradient to each flow based on its number of trips:

```
ggmap(SanFran, darken=0.75) +
  geom_path(aes(
                x=lon,
                y=lat,
                group=id,
                colour=trips
                ),
            data=lines,
            size=log1p(lines$trips / max(lines$trips)),
            lineend='round') +
  scale_colour_gradient(low='grey',
                        high='#07eda0') +
  theme(axis.text.x = element_blank(),
      axis.text.y = element_blank(),
      axis.ticks = element_blank()
      )
```

Note how we transform the size so it's a proportion of the largest trip and then it is compressed with a logarithm.

6.4 Modelling flows

Now we have an idea of the spatial distribution of flows, we can begin to think about modeling them. The core idea in this section is to fit a model that can capture the particular characteristics of our variable of interest (the volume of trips) using a set of predictors that describe the nature of a given flow. We will start from the simplest model and then progressively build complexity until we get to a satisfying point. Along the way, we will be exploring each model using concepts from Gelman and Hill (2006a) such as predictive performance checks[4] (PPC)

Before we start running regressions, let us first standardize the predictors so we can interpret the intercept as the average flow when all the predictors take the average value, and so we can interpret the model coefficients as changes in standard deviation units:

```
# Scale all the table
db_std <- as.data.frame(scale(db@data))
# Reset trips as we want the original version
db_std$trips15 <- db@data$trips15
db_std$trips16 <- db@data$trips16
# Reset origin and destination station and express them as factors
db_std$orig <- as.factor(db@data$orig)
db_std$dest <- as.factor(db@data$dest)
```

Baseline model

One of the simplest possible models we can fit in this context is a linear model that explains the number of trips as a function of the straight distance between the two stations and total amount of climb and downhill. We will take this as the baseline on which we can further build later:

```
m1 <- lm('trips15 ~ straight_dist + total_up + total_down', data=db_std)
summary(m1)
```

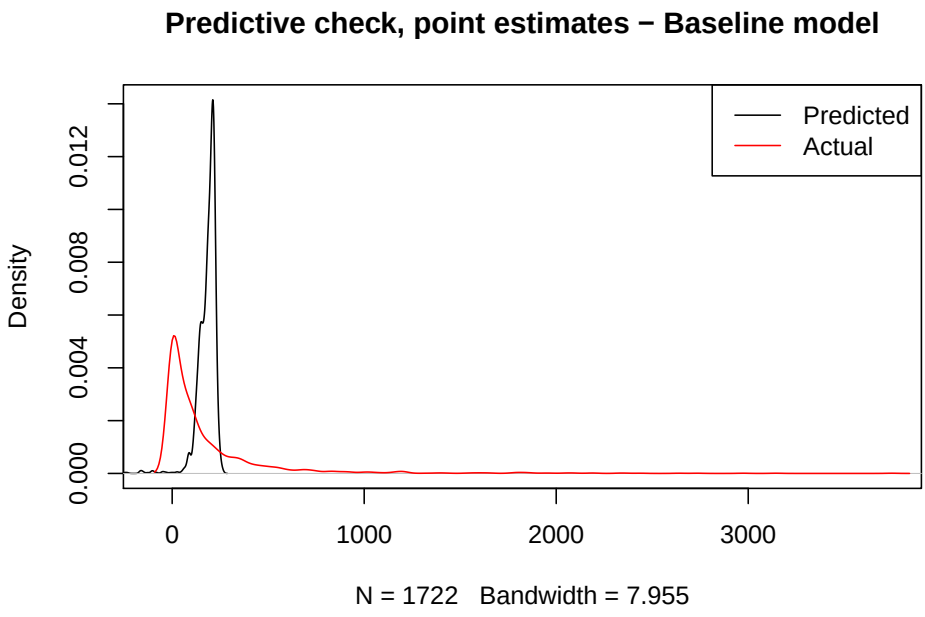
```
##
## Call:
## lm(formula = "trips15 ~ straight_dist + total_up + total_down",
##     data = db_std)
##
## Residuals:
##    Min     1Q Median     3Q    Max
## -261.9 -168.3 -102.4   30.8 3527.4
##
## Coefficients:
##               Estimate Std. Error t value Pr(>|t|)
## (Intercept)    182.070      8.110  22.451  < 2e-16 ***
```

[4] For a more elaborate introduction to PPC, have a look at Chapters 7 and 8.

```
## straight_dist   17.906      9.108   1.966   0.0495 *
## total_up       -44.100      9.353  -4.715 2.61e-06 ***
## total_down     -20.241      9.229  -2.193   0.0284 *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 336.5 on 1718 degrees of freedom
## Multiple R-squared:  0.02196,    Adjusted R-squared:  0.02025
## F-statistic: 12.86 on 3 and 1718 DF,  p-value: 2.625e-08
```

To explore how good this model is, we will be comparing the predictions the model makes about the number of trips each flow should have with the actual number of trips. A first approach is to simply plot the distribution of both variables:

```
plot(density(m1$fitted.values),
     xlim=c(-100, max(db_std$trips15)),
     main='')
lines(density(db_std$trips15),
      col='red',
      main='')
legend('topright',
       c('Predicted', 'Actual'),
       col=c('black', 'red'),
       lwd=1)
title(main="Predictive check, point estimates - Baseline model")
```



The plot makes pretty obvious that our initial model captures very few aspects of the distribution we want to explain. However, we should not get too attached to this plot just yet. What it is showing is the distribution of predicted *point* estimates from our model. Since our model is not deterministic but inferential, there is a certain degree of uncertainty attached to its predictions, and that is completely absent from this plot.

Generally speaking, a given model has two sources of uncertainty: *predictive*, and *inferential*. The former relates to the fact that the equation we fit does not capture all the elements or in the exact form they enter the true data generating process; the latter has to do with the fact that we never get to know the true value of the model parameters only guesses (estimates) subject to error and uncertainty. If you think of our linear model above as

$$T_{ij} = X_{ij}\beta + \epsilon_{ij}$$

where T_{ij} represents the number of trips undertaken between station i and j, X_{ij} is the set of explanatory variables (length, climb, descent, etc.), and ϵ_{ij} is an error term assumed to be distributed as a normal distribution $N(0, \sigma)$; then predictive uncertainty comes from the fact that there are elements to some extent relevant for y that are not accounted for and thus subsummed into ϵ_{ij}. Inferential uncertainty comes from the fact that we never get to know β but only an estimate of it which is also subject to uncertainty itself.

Taking these two sources into consideration means that the black line in the plot above represents only the behaviour of our model we expect if the error term is absent (no predictive uncertainty) and the coefficients are the true estimates (no inferential uncertainty). However, this is not necessarily the case as our estimate for the uncertainty of the error term is certainly not zero, and our estimates for each parameter are also subject to a great deal of inferential variability. we do not know to what extent other outcomes would be just as likely. Predictive checking relates to simulating several feasible scenarios under our model and use those to assess uncertainty and to get a better grasp of the quality of our predictions.

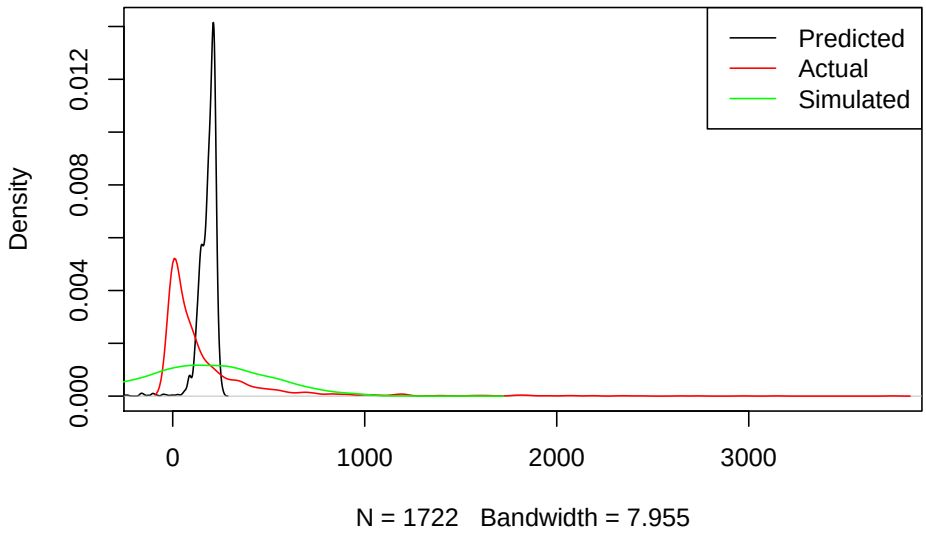
Technically speaking, to do this, we need to build a mechanism to obtain a possible draw from our model and then repeat it several times. The first part of those two steps can be elegantly dealt with by writing a short function that takes a given model and a set of predictors, and produces a possible random draw from such model:

```
generate_draw <- function(m){
  # Set up predictors matrix
  x <- model.matrix(m)
  # Obtain draws of parameters (inferential uncertainty)
  sim_bs <- sim(m, 1)
  # Predicted value
  mu <- x %*% sim_bs@coef[1, ]
```

```
  # Draw
  n <- length(mu)
  y_hat <- rnorm(n, mu, sim_bs@sigma[1])
  return(y_hat)
}
```

This function takes a model `m` and the set of covariates `x` used and returns a random realization of predictions from the model. To get a sense of how this works, we can get and plot a realization of the model, compared to the expected one and the actual values:

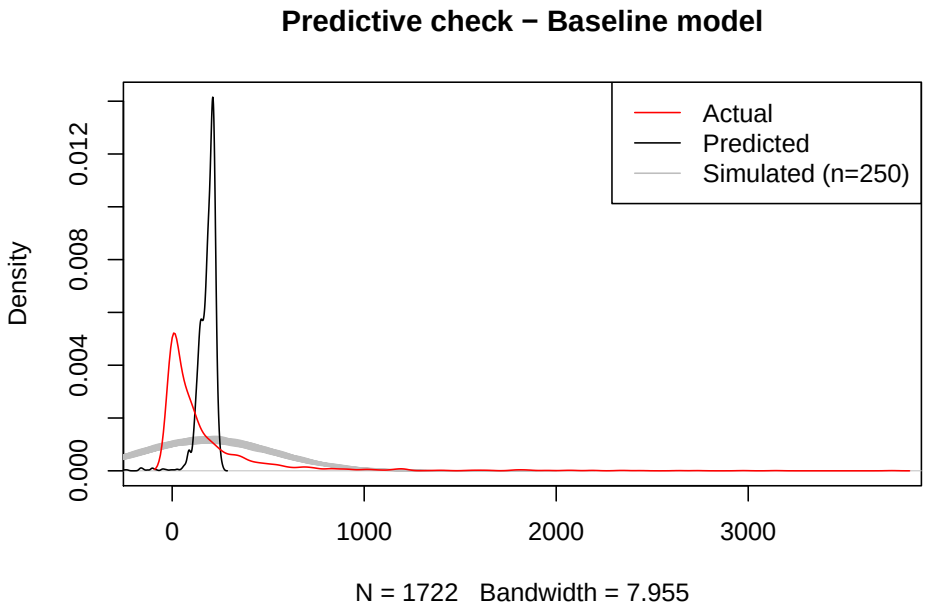
```
new_y <- generate_draw(m1)

plot(density(m1$fitted.values),
     xlim=c(-100, max(db_std$trips15)),
     ylim=c(0, max(c(
                     max(density(m1$fitted.values)$y),
                     max(density(db_std$trips15)$y)
                     )
                   )
            ),
     col='black',
     main='')
lines(density(db_std$trips15),
      col='red',
      main='')
lines(density(new_y),
      col='green',
      main='')
legend('topright',
       c('Predicted', 'Actual', 'Simulated'),
       col=c('black', 'red', 'green'),
       lwd=1)
```



Once we have this "draw engine", we can set it to work as many times as we want using a simple `for` loop. In fact, we can directly plot these lines as compared to the expected one and the trip count:

```
plot(density(m1$fitted.values),
     xlim=c(-100, max(db_std$trips15)),
     ylim=c(0, max(c(
                  max(density(m1$fitted.values)$y),
                  max(density(db_std$trips15)$y)
                  )
                )
        ),
     col='white',
     main='')
# Loop for realizations
for(i in 1:250){
  tmp_y <- generate_draw(m1)
  lines(density(tmp_y),
        col='grey',
        lwd=0.1
        )
}
#
lines(density(m1$fitted.values),
      col='black',
      main='')
lines(density(db_std$trips15),
      col='red',
      main='')
```

```
legend('topright',
       c('Actual', 'Predicted', 'Simulated (n=250)'),
       col=c('red', 'black', 'grey'),
       lwd=1)
title(main="Predictive check - Baseline model")
```



The plot shows there is a significant mismatch between the fitted values, which are much more concentrated around small positive values, and the realizations of our "inferential engine", which depict a much less concentrated distribution of values. This is likely due to the combination of two different reasons: on the one hand, the accuracy of our estimates may be poor, causing them to jump around a wide range of potential values and hence resulting in very diverse predictions (inferential uncertainty); on the other hand, it may be that the amount of variation we are not able to account for in the model[5] is so large that the degree of uncertainty contained in the error term of the model is very large, hence resulting in such a flat predictive distribution.

It is important to keep in mind that the issues discussed in the paragraph above relate only to the uncertainty behind our model, not to the point predictions derived from them, which are a mechanistic result of the minimization of the squared residuals and hence are not subject to probability or inference. That allows them in this case to provide a fitted distribution much more accurate apparently (black line above). However, the lesson to take from this model is that, even if the point predictions (fitted values) are artificially accurate[6], our

[5] The R^2 of our model is around 2%

[6] which they are not really, in light of the comparison between the black and red lines.

capabilities to infer about the more general underlying process are fairly limited.

Improving the model

The bad news from the previous section is that our initial model is not great at explaining bike trips. The good news is there are several ways in which we can improve this. In this section we will cover three main extensions that exemplify three different routes you can take when enriching and augmenting models in general, and spatial interaction ones in particular[7]. These three routes are aligned around the following principles:

1. Use better approximations to model your dependent variable.
2. Recognize the structure of your data.
3. Get better predictors.

- **Use better approximations to model your dependent variable**

Standard OLS regression assumes that the error term and, since the predictors are deterministic, the dependent variable are distributed following a normal (gaussian) distribution. This is usually a good approximation for several phenomena of interest, but maybe not the best one for trips along routes: for one, we know trips cannot be negative, which the normal distribution does not account for[8]; more subtly, their distribution is not really symmetric but skewed with a very long tail on the right. This is common in variables that represent counts and that is why usually it is more appropriate to fit a model that relies on a distribution different from the normal.

One of the most common distributions for this cases is the Poisson, which can be incorporated through a general linear model (or GLM). The underlying assumption here is that instead of $T_{ij} \sim N(\mu_{ij}, \sigma)$, our model now follows:

$$T_{ij} \sim Poisson(\exp^{X_{ij}\beta})$$

As usual, such a model is easy to run in R:

```
m2 <- glm('trips15 ~ straight_dist + total_up + total_down',
          data=db_std,
          family=poisson,
          )
```

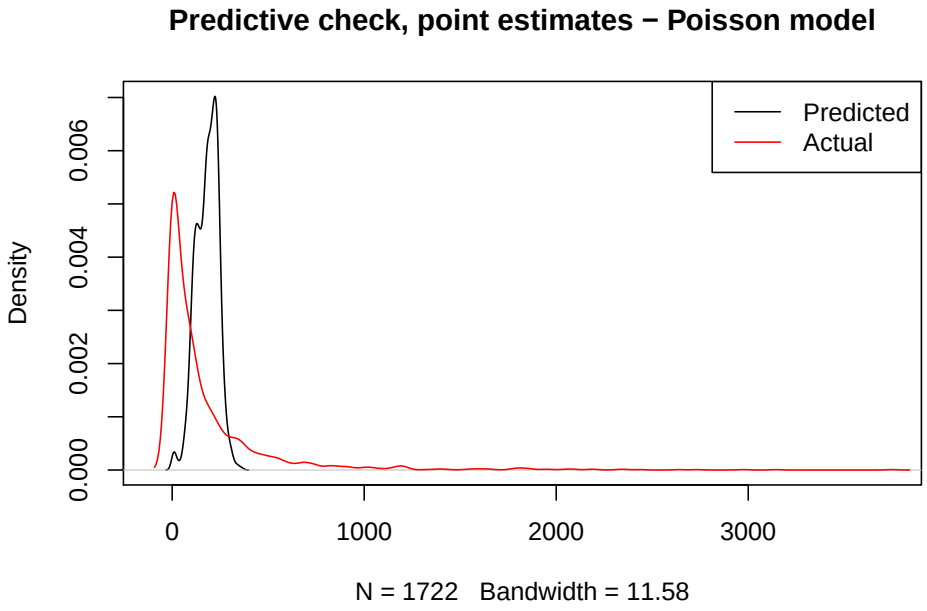
Now let's see how much better, if any, this approach is. To get a quick overview, we can simply plot the point predictions:

```
plot(density(m2$fitted.values),
     xlim=c(-100, max(db_std$trips15)),
```

[7] These principles are general and can be applied to pretty much any modeling exercise you run into. The specific approaches we take in this note relate to spatial interaction models

[8] For an illustration of this, consider the amount of probability mass to the left of zero in the predictive checks above.

```
     ylim=c(0, max(c(
                  max(density(m2$fitted.values)$y),
                  max(density(db_std$trips15)$y)
                  )
                )
      ),
     col='black',
     main='')
lines(density(db_std$trips15),
      col='red',
      main='')
legend('topright',
       c('Predicted', 'Actual'),
       col=c('black', 'red'),
       lwd=1)
title(main="Predictive check, point estimates - Poisson model")
```



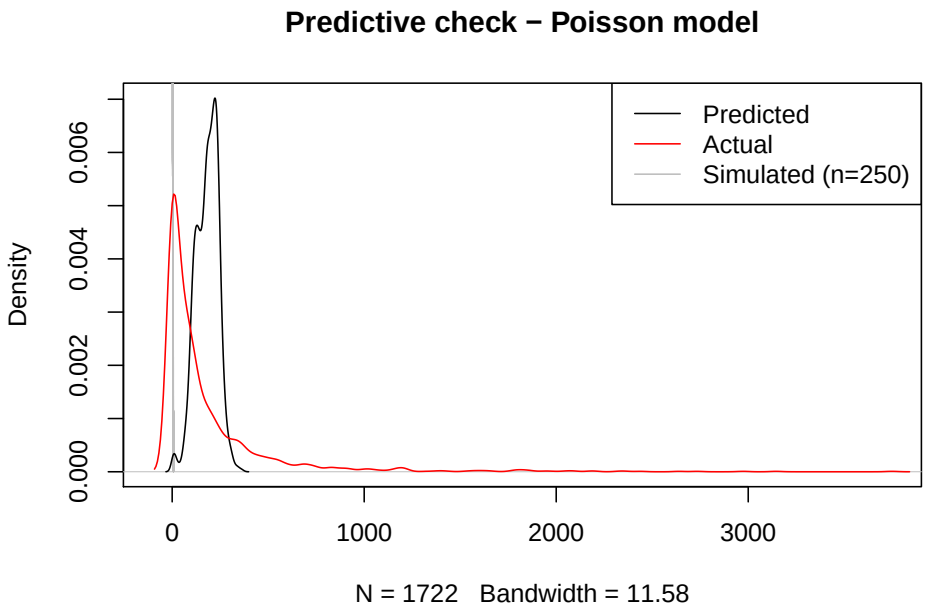
To incorporate uncertainty to these predictions, we need to tweak our `generate_draw` function so it accommodates the fact that our model is not linear anymore.

```
generate_draw_poi <- function(m){
  # Set up predictors matrix
  x <- model.matrix(m)
  # Obtain draws of parameters (inferential uncertainty)
  sim_bs <- sim(m, 1)
```

```
  # Predicted value
  xb <- x %*% sim_bs@coef[1, ]
  #xb <- x %*% m$coefficients
  # Transform using the link function
  mu <- exp(xb)
  # Obtain a random realization
  y_hat <- rpois(n=length(mu), lambda=mu)
  return(y_hat)
}
```

And then we can examine both point predictions an uncertainty around them:

```
plot(density(m2$fitted.values),
     xlim=c(-100, max(db_std$trips15)),
     ylim=c(0, max(c(
                  max(density(m2$fitted.values)$y),
                  max(density(db_std$trips15)$y)
                  )
                )
      ),
     col='white',
     main='')
# Loop for realizations
for(i in 1:250){
  tmp_y <- generate_draw_poi(m2)
  lines(density(tmp_y),
        col='grey',
        lwd=0.1
        )
}
#
lines(density(m2$fitted.values),
      col='black',
      main='')
lines(density(db_std$trips15),
      col='red',
      main='')
legend('topright',
       c('Predicted', 'Actual', 'Simulated (n=250)'),
       col=c('black', 'red', 'grey'),
       lwd=1)
title(main="Predictive check - Poisson model")
```



Voila! Although the curve is still a bit off, centered too much to the right of the actual data, our predictive simulation leaves the fitted values right in the middle. This speaks to a better fit of the model to the actual distribution othe original data follow.

- **Recognize the structure of your data**

So far, we've treated our dataset as if it was flat (i.e. comprise of fully independent realizations) when in fact it is not. Most crucially, our baseline model does not account for the fact that every observation in the dataset pertains to a trip between two stations. This means that all the trips from or to the same station probably share elements which likely help explain how many trips are undertaken between stations. For example, think of trips to an from a station located in the famous Embarcadero, a popular tourist spot. Every route to and from there probably has more trips due to the popularity of the area and we are currently not acknowledging it in the model.

A simple way to incorporate these effects into the model is through origin and destination fixed effects. This approach shares elements with both spatial fixed effects and multilevel modeling and essentially consists of including a binary variable for every origin and destination station. In mathematical notation, this equates to:

$$T_{ij} = X_{ij}\beta + \delta_i + \delta_j + \epsilon_{ij}$$

where δ_i and δ_j are origin and destination station fixed effects[9], and the rest

[9] In this session, δ_i and δ_j are estimated as independent variables so their estimates are

is as above. This strategy accounts for all the unobserved heterogeneity associated with the location of the station. Technically speaking, we simply need to introduce `orig` and `dest` in the the model:

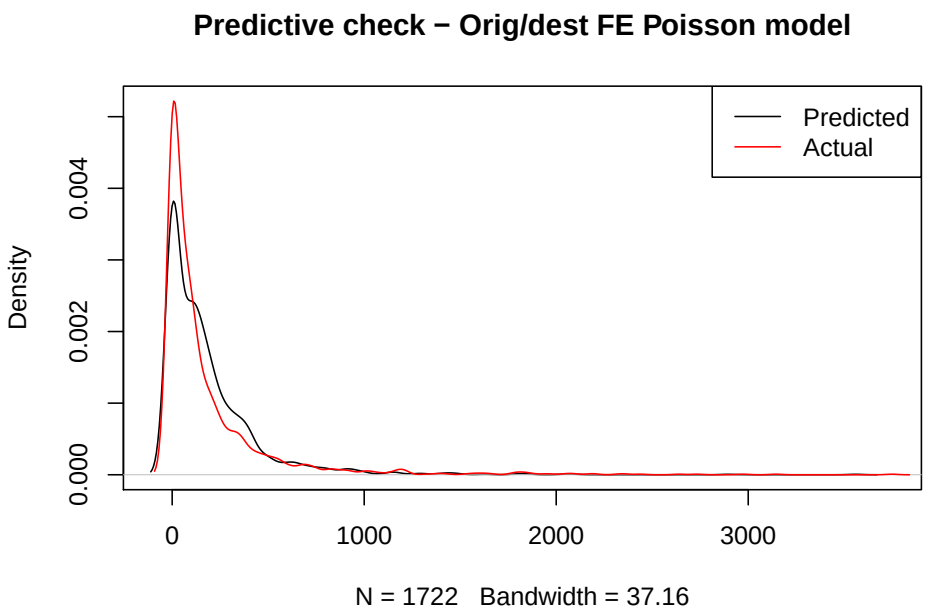
```
m3 <- glm('trips15 ~ straight_dist + total_up + total_down + orig + dest',
          data=db_std,
          family=poisson)
```

And with our new model, we can have a look at how well it does at predicting the overall number of trips[10]:

```
plot(density(m3$fitted.values),
     xlim=c(-100, max(db_std$trips15)),
     ylim=c(0, max(c(
                  max(density(m3$fitted.values)$y),
                  max(density(db_std$trips15)$y)
                  )
                )
      ),
     col='black',
     main='')
lines(density(db_std$trips15),
      col='red',
      main='')
legend('topright',
       c('Predicted', 'Actual'),
       col=c('black', 'red'),
       lwd=1)
title(main="Predictive check - Orig/dest FE Poisson model")
```

similar to interpret to those in β. An alternative approach could be to model them as random effects in a multilevel framework.

[10] Although, theoretically, we could also include simulations of the model in the plot to get a better sense of the uncertainty behind our model, in practice this seems troublesome. The problems most likely arise from the fact that many of the origin and destination binary variable coefficients are estimated with a great deal of uncertainty. This causes some of the simulation to generate extreme values that, when passed through the exponential term of the Poisson link function, cause problems. If anything, this is testimony of how a simple fixed effect model can sometimes lack accuracy and generate very uncertain estimates. A potential extension to work around these problems could be to fit a multilevel model with two specific levels beyond the trip-level: one for origin and another one for destination stations.



That looks significantly better, doesn't it? In fact, our model now better accounts for the long tail where a few routes take a lot of trips. This is likely because the distribution of trips is far from random across stations and our origin and destination fixed effects do a decent job at accounting for that structure. However our model is still notably underpredicting less popular routes and overpredicting routes with above average number of trips. Maybe we should think about moving beyond a simple linear model.

- **Get better predictors**

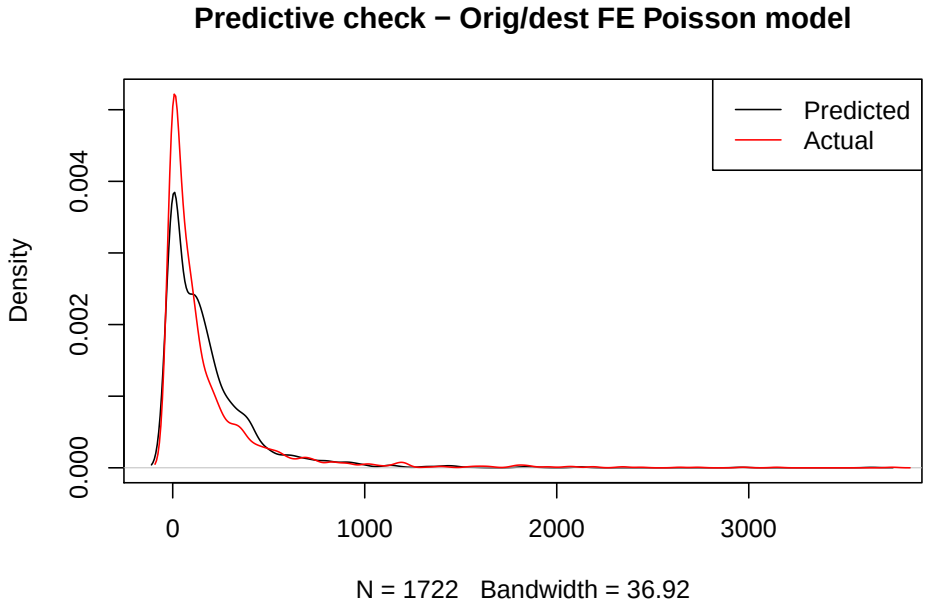
The final extension is, in principle, always available but, in practice, it can be tricky to implement. The core idea is that your baseline model might not have the best measurement of the phenomena you want to account for. In our example, we can think of the distance between stations. So far, we have been including the distance measured "as the crow flies" between stations. Although in some cases this is a good approximation (particularly when distances are long and likely route taken is as close to straight as possible), in some cases like ours, where the street layout and the presence of elevation probably matter more than the actual final distance pedalled, this is not necessarily a safe assumption.

As an exampe of this approach, we can replace the straight distance measurements for more refined ones based on the Google Maps API routes. This is very easy as all we need to do (once the distances have been calculated!) is to swap `straight_dist` for `street_dist`:

```
m4 <- glm('trips15 ~ street_dist + total_up + total_down + orig + dest',
          data=db_std,
          family=poisson)
```

And we can similarly get a sense of our predictive fitting with:

```
plot(density(m4$fitted.values),
     xlim=c(-100, max(db_std$trips15)),
     ylim=c(0, max(c(
                  max(density(m4$fitted.values)$y),
                  max(density(db_std$trips15)$y)
                  )
                )
      ),
     col='black',
     main='')
lines(density(db_std$trips15),
      col='red',
      main='')
legend('topright',
       c('Predicted', 'Actual'),
       col=c('black', 'red'),
       lwd=1)
title(main="Predictive check - Orig/dest FE Poisson model")
```



Hard to tell any noticeable difference, right? To see if there is any, we can have a look at the estimates obtained:

```
summary(m4)$coefficients['street_dist', ]
```

```
##       Estimate     Std. Error        z value       Pr(>|z|)
##  -9.961619e-02   2.688731e-03  -3.704952e+01  1.828096e-300
```

And compare this to that of the straight distances in the previous model:

```
summary(m3)$coefficients['straight_dist', ]
```

```
##       Estimate     Std. Error        z value       Pr(>|z|)
##  -7.820014e-02   2.683052e-03  -2.914596e+01  9.399407e-187
```

As we can see, the differences exist but ar not massive. Let's use this example to learn how to interpret coefficients in a Poisson model[11]. Effectively, these estimates can be understood as multiplicative effects. Since our model fits

$$T_{ij} \sim Poisson(\exp^{X_{ij}\beta})$$

we need to transform β through an exponential in order to get a sense of the effect of distance on the number of trips. This means that for the street distance, our original estimate is $\beta_{street} = -0.0996$, but this needs to be translated through the exponential into $e^{-0.0996} = 0.906$. In other words, since distance is expressed in standard deviations[12], we can expect a 10% decrease in the number of trips for an increase of one standard deviation (about 1Km) in the distance between the stations. This can be compared with $e^{-0.0782} = 0.925$ for the straight distances, or a reduction of about 8% the number of trips for every increase of a standard deviation (about 720m).

6.5 Predicting flows

So far we have put all of our modeling efforts in understanding the model we fit and improving such model so it fits our data as closely as possible. This is essential in any modelling exercise but should be far from a stopping point. Once we're confident our model is a decent representation of the data generating process, we can start exploiting it. In this section, we will cover one specific case that showcases how a fitted model can help: out-of-sample forecasts.

It is August 2015, and you have just started working as a data scientist for the bikeshare company that runs the San Francisco system. You join them as they're planning for the next academic year and, in order to plan their operations (re-allocating vans, station maintenance, etc.), they need to get a sense of how many people are going to be pedalling across the city and, crucially, *where* they are going to be pedalling through. What can you do to help them?

The easiest approach is to say "well, a good guess for how many people will be going between two given stations this coming year is how many went through last year, isn't it?". This is one prediction approach. However, you could see how, even if the same process governs over both datasets (2015 and 2016), each year will probably have some idiosyncracies and thus looking too closely into

[11] See section 6.2 of Gelman and Hill (2006a) for a similar treatment of these.
[12] Remember the transformation at the very beginning.

one year might not give the best possible answer for the next one. Ideally, you want a good stylized synthesis that captures the bits that stay constant over time and thus can be applied in the future and that ignores those aspects that are too particular to a given point in time. That is the rationale behind using a fitted model to obtain predictions.

However good any theory though, the truth is in the pudding. So, to see if a modeling approach is better at producing forecasts than just using the counts from last year, we can put them to a test. The way this is done when evaluating the predictive performance of a model (as this is called in the literature) relies on two basic steps: a) obtain predictions from a given model and b) compare those to the actual values (in our case, with the counts for 2016 in `trips16`) and get a sense of "how off" they are. We have essentially covered a) above; for b), there are several measures to use. We will use one of the most common ones, the root mean squared error (RMSE), which roughly gives a sense of the average difference between a predicted vector and the real deal:

$$RMSE = \sqrt{\sum_{ij} (\hat{T_{ij}} - T_{ij})^2}$$

where $\hat{T_{ij}}$ is the predicted amount of trips between stations i and j. RMSE is straightforward in R and, since we will use it a couple of times, let's write a short function to make our lives easier:

```
rmse <- function(t, p){
  se <- (t - p)^2
  mse <- mean(se)
  rmse <- sqrt(mse)
  return(rmse)
}
```

where `t` stands for the vector of true values, and `p` is the vector of predictions. Let's give it a spin to make sure it works:

```
rmse_m4 <- rmse(db_std$trips16, m4$fitted.values)
rmse_m4
```

```
## [1] 256.2197
```

That means that, on average, predictions in our best model `m4` are 256 trips off. Is this good? Bad? Worse? It's hard to say but, being practical, what we can say is whether this better than our alternative. Let us have a look at the RMSE of the other models as well as that of simply plugging in last year's counts:[13]

[13] **EXERCISE**: can you create a single plot that displays the distribution of the predicted values of the five different ways to predict trips in 2016 and the actual counts of trips?

```
rmses <- data.frame(model=c('OLS', 'Poisson', 'Poisson + FE',
                            'Poisson + FE + street dist.',
                            'Trips-2015'
                            ),
                    RMSE=c(rmse(db_std$trips16,
                                m1$fitted.values),
                           rmse(db_std$trips16,
                                m2$fitted.values),
                           rmse(db_std$trips16,
                                m3$fitted.values),
                           rmse(db_std$trips16,
                                m4$fitted.values),
                           rmse(db_std$trips16,
                                db_std$trips15)
                           )
                      )
rmses
```

```
##                         model     RMSE
## 1                         OLS 323.6135
## 2                     Poisson 320.8962
## 3                 Poisson + FE 254.4468
## 4 Poisson + FE + street dist. 256.2197
## 5                   Trips-2015 131.0228
```

The table is both encouraging and disheartning at the same time. On the one hand, all the modeling techniques covered above behave as we would expect: the baseline model displays the worst predicting power of all, and every improvement (except the street distances!) results in notable decreases of the RMSE. This is good news. However, on the other hand, all of our modelling efforts fall short of given a better guess than simply using the previous year's counts. *Why? Does this mean that we should not pay attention to modeling and inference?* Not really. Generally speaking, a model is as good at predicting as it is able to mimic the underlying process that gave rise to the data in the first place. The results above point to a case where our model is not picking up all the factors that determine the amount of trips undertaken in a give route. This could be improved by enriching the model with more/better predictors, as we have seen above. Also, the example above seems to point to a case where those idiosyncracies in 2015 that the model does not pick up seem to be at work in 2016 as well. This is great news for our prediction efforts this time, but we have no idea why this is the case and, for all that matters, it could change the coming year. Besides the elegant quantification of uncertainty, the true advantage of a modeling approach in this context is that, if well fit, it is able to pick up the fundamentals that apply over and over. This means that, if next year we're not as lucky as this one and previous counts are not good predictors but the variables we used in our model continue to have a role in determining

the outcome, the data scientist should be luckier and hit a better prediction.

6.6 References

Chapter 7

Spatial Econometrics

This chapter[1] is based on the following references, which are good follow-up's on the topic:

- Session III of Arribas-Bel (2014). Check the "Related readings" section on the session page for more in-depth discussions.
- Anselin (2007), freely available to download [`pdf`].
- The second part of this tutorial assumes you have reviewed Lecture 5 of Arribas-Bel (2019). [html]
- A similar coverage of topics is presented in Chapter 11 of the upcoming book "Geographic Data Science with Python", which is currently in progress. [`url`]

This tutorial is part of Spatial Analysis Notes, a compilation hosted as a GitHub repository that you can access it in a few ways:

- As a download of a `.zip` file that contains all the materials.
- As an html website.
- As a pdf document
- As a GitHub repository.

7.1 Dependencies

The illustration below relies on the following libraries that you will need to have installed on your machine to be able to interactively follow along[2]. Once installed, load them up with the following commands:

[1] Spatial Econometrics by Dani Arribas-Bel is licensed under a Creative Commons Attribution-NonCommercial-ShareAlike 4.0 International License.

[2] You can install package `mypackage` by running the command `install.packages("mypackage")` on the R prompt or through the `Tools --> Install Packages...` menu in RStudio.

```
# Layout
library(tufte)
# For pretty table
library(knitr)
# Spatial Data management
library(rgdal)
```

```
## Loading required package: sp
```

```
## rgdal: version: 1.5-18, (SVN revision 1082)
## Geospatial Data Abstraction Library extensions to R successfully loaded
## Loaded GDAL runtime: GDAL 3.0.4, released 2020/01/28
## Path to GDAL shared files: /usr/share/gdal
## GDAL binary built with GEOS: TRUE
## Loaded PROJ runtime: Rel. 6.3.1, February 10th, 2020, [PJ_VERSION: 631]
## Path to PROJ shared files: /usr/share/proj
## Linking to sp version:1.4-4
## To mute warnings of possible GDAL/OSR exportToProj4() degradation,
## use options("rgdal_show_exportToProj4_warnings"="none") before loading rgdal.
```

```
# Pretty graphics
library(ggplot2)
# Pretty maps
library(ggmap)
```

```
## Google's Terms of Service: https://cloud.google.com/maps-platform/terms/.
```

```
## Please cite ggmap if you use it! See citation("ggmap") for details.
```

```
# Various GIS utilities
library(GISTools)
```

```
## Loading required package: maptools
```

```
## Checking rgeos availability: TRUE
```

```
## Loading required package: RColorBrewer
```

```
## Loading required package: MASS
```

```
## Loading required package: rgeos
```

```
## rgeos version: 0.5-5, (SVN revision 640)
##  GEOS runtime version: 3.8.0-CAPI-1.13.1
##  Linking to sp version: 1.4-4
##  Polygon checking: TRUE
```

```
# For all your interpolation needs
library(gstat)
# For data manipulation
library(plyr)
```

```
# Spatial regression
library(spdep)
```

```
## Loading required package: spData
```

```
## Loading required package: sf
```

```
## Linking to GEOS 3.8.0, GDAL 3.0.4, PROJ 6.3.1
```

Before we start any analysis, let us set the path to the directory where we are working. We can easily do that with `setwd()`. Please replace in the following line the path to the folder where you have placed this file -and where the `house_transactions` folder with the data lives.

```
#setwd('/media/dani/baul/AAA/Documents/teaching/u-lvl/2016/envs453/code/GIT/kde_idw_r/')
setwd('.')
```

7.2 Data

To explore ideas in spatial regression, we will be using house price data for the municipality of Liverpool. Our main dataset is provided by the Land Registry (as part of their Price Paid Data) but has been cleaned and re-packaged into a shapefile by Dani Arribas-Bel.

Let us load it up first of all:

```
hst <- readOGR(dsn = 'data/house_transactions', layer = 'liv_house_trans')
```

```
## Warning in OGRSpatialRef(dsn, layer, morphFromESRI = morphFromESRI, dumpSRS =
## dumpSRS, : Discarded datum OSGB_1936 in CRS definition: +proj=tmerc +lat_0=49
## +lon_0=-2 +k=0.9996012717 +x_0=400000 +y_0=-100000 +ellps=airy +units=m +no_defs
```

```
## OGR data source with driver: ESRI Shapefile
## Source: "/home/rstudio/Documents/data/house_transactions", layer: "liv_house_trans"
## with 6324 features
## It has 18 fields
## Integer64 fields read as strings:  price
```

The tabular component of the spatial frame contains the followig variables:

```
names(hst)
```

```
##  [1] "pcds"       "id"         "price"      "trans_date" "type"
##  [6] "new"        "duration"   "paon"       "saon"       "street"
## [11] "locality"   "town"       "district"   "county"     "ppd_cat"
## [16] "status"     "lsoa11"     "LSOA11CD"
```

The meaning for most of the variables can be found in the original Land Registry documentation. The dataset contains transactions that took place during 2,014:

```
# Format dates
dts <- as.Date(hst@data$trans_date)
# Set up summary table
tab <- summary(dts)
tab
```

```
##         Min.      1st Qu.       Median         Mean      3rd Qu.         Max.
## "2014-01-02" "2014-04-11" "2014-07-09" "2014-07-08" "2014-10-03" "2014-12-30"
```

Although the original Land Registry data contain some characteristics of the house, all of them are categorical: *is the house newly built? What type of property is it?* To bring in a richer picture and illustrate how continuous variables can also be included in a spatial setting, we will augment the original transaction data with Deprivation indices from the CDRC at the Lower Layer Super Output Area (LSOA) level.

Let us read the `csv` in:

```
imd <- read.csv('data/house_transactions/E08000012.csv')
```

The table contains not only the overall IMD score and rank, but some of the component scores, as well as the `LSOA` code:

```
names(imd)
```

```
##  [1] "LSOA11CD"   "imd_rank"   "imd_score"  "income"     "employment"
##  [6] "education"  "health"     "crime"      "housing"    "living_env"
## [11] "idaci"      "idaopi"
```

That bit of information, `LSOA11CD`, is crucial to be able to connect it to each house transaction. To "join" both tables, we can use the base command `merge`, which will assign values from `imd` into `hst` making sure that each house transaction get the IMD data for the LSOA where it is located:

```
db <- merge(hst, imd)
```

The resulting table, `db`, contains variables from both original tables:

```
names(db)
```

```
##  [1] "LSOA11CD"   "pcds"       "id"         "price"      "trans_date"
##  [6] "type"       "new"        "duration"   "paon"       "saon"
## [11] "street"     "locality"   "town"       "district"   "county"
## [16] "ppd_cat"    "status"     "lsoa11"     "imd_rank"   "imd_score"
## [21] "income"     "employment" "education"  "health"     "crime"
## [26] "housing"    "living_env" "idaci"      "idaopi"
```

Since we will heavily rely on `price`, we need to turn it into a numeric column, rather than as a factor, which is how it is picked up:

```
db@data$price <- as.numeric(as.character(db@data$price))
```

For some of our analysis, we will need the coarse postcode of each house, rather than the finely specified one in the original data. This means using the available one

```
head(db@data['pcds'])
```

```
##      pcds
## 62 L1 0AB
## 63 L1 0AB
## 64 L1 0AB
## 65 L1 0AB
## 66 L1 0AB
## 67 L1 0AB
```

to create a new column that only contains the first bit of the postcode (`L1` in the examples above). The following lines of code will do that for us:

```
db$pc <- as.character(lapply(strsplit(as.character(db$pcds), split=" "), "[", 1))
```

Given there are 6,324 transactions in the dataset, a simple plot of the point coordinates implicitly draws the shape of the Liverpool municipality:

```
plot(db)
```

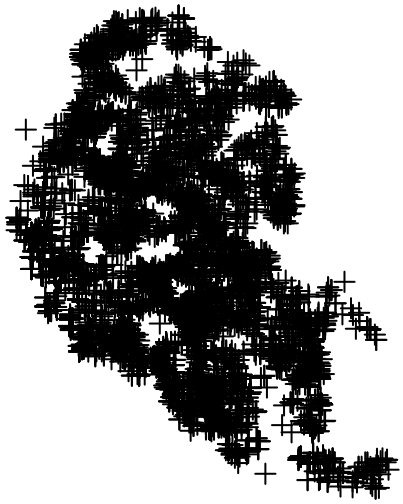


Figure 7.1: Spatial distribution of house transactions in Liverpool

7.3 Non-spatial regression, a refresh

Before we discuss how to explicitly include space into the linear regression framework, let us show how basic regression can be carried out in R, and how you can begin to interpret the results. By no means is this a formal and complete introduction to regression so, if that is what you are looking for, I suggest the first part of Gelman and Hill (2006a), in particular chapters 3 and 4.

The core idea of linear regression is to explain the variation in a given (*dependent*) variable as a linear function of a series of other (*explanatory*) variables. For example, in our case, we may want to express/explain the price of a house as a function of whether it is new and the degree of deprivation of the area where it is located. At the individual level, we can express this as:

$$P_i = \alpha + \beta_1 NEW_i + \beta_2 IMD_i + \epsilon_i$$

where P_i is the price of house i, NEW_i is a binary variable that takes one if the house is newly built or zero otherwise and IMD_i is the IMD score of the LSOA where i is located. The parameters β_1, β_2, and β_3 give us information about in which way and to what extent each variable is related to the price, and α, the constant term, is the average house price when all the other variables are zero. The term ϵ_i is usually referred to as "error" and captures elements that influence the price of a house but are not whether the house is new or the IMD score of its area. We can also express this relation in matrix form, excluding subindices for i[3].

Essentially, a regression can be seen as a multivariate extension of simple bivariate correlations. Indeed, one way to interpret the β_k coefficients in the equation above is as the degree of correlation between the explanatory variable k and the dependent variable, *keeping all the other explanatory variables constant*. When you calculate simple bivariate correlations, the coefficient of a variable is picking up the correlation between the variables, but it is also subsuming into it variation associated with other correlated variables –also called confounding factors[4]. Regression allows you to isolate the distinct effect that a single variable has on the dependent one, once we *control* for those other variables.

[3] In this case, the equation would look like

$$P = \alpha + \beta_1 NEW + \beta_2 IMD + \epsilon$$

and would be interpreted in terms of vectors and matrices instead of scalar values.

[4] **EXAMPLE** Assume that new houses tend to be built more often in areas with low deprivation. If that is the case, then NEW and IMD will be correlated with each other (as well as with the price of a house, as we are hypothesizing in this case). If we calculate a simple correlation between P and IMD, the coefficient will represent the degree of association between both variables, but it will also include some of the association between IMD and NEW. That is, part of the obtained correlation coefficient will be due not to the fact that higher prices tend to be found in areas with low IMD, but to the fact that new houses tend to be more expensive. This is because (in this example) new houses tend to be built in areas with low deprivation and simple bivariate correlation cannot account for that.

Practically speaking, running linear regressions in R is straightforward. For example, to fit the model specified in the equation above, we only need one line of code:

```
m1 <- lm('price ~ new + imd_score', db)
```

We use the command `lm`, for linear model, and specify the equation we want to fit using a string that relates the dependent variable (`price`) with a set of explanatory ones (`new` and `price`) by using a tilde `~` that is akin the = symbol in the mathematical equation. Since we are using names of variables that are stored in a table, we need to pass the table object (`db`) as well.

In order to inspect the results of the model, the quickest way is to call `summary`:

```
summary(m1)
```

```
##
## Call:
## lm(formula = "price ~ new + imd_score", data = db)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
##  -184254   -59948   -29032    11430 26434741
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)   235596      13326  17.679  < 2e-16 ***
## newY            4926      19104   0.258    0.797
## imd_score      -2416        308  -7.843 5.12e-15 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 509000 on 6321 degrees of freedom
## Multiple R-squared:  0.009712,   Adjusted R-squared:  0.009398
## F-statistic: 30.99 on 2 and 6321 DF,  p-value: 4.027e-14
```

A full detailed explanation of the output is beyond the scope of this note, so we will focus on the relevant bits for our main purpose. This is concentrated on the `Coefficients` section, which gives us the estimates for the β_k coefficients in our model. Or, in other words, the coefficients are the raw equivalent of the correlation coefficient between each explanatory variable and the dependent one, once the polluting effect of confounding factors has been accounted for[5]. Results are as expected for the most part: houses tend to be significantly more expensive in areas with lower deprivation (an average of GBP2,416 for every additional score); and a newly built house is on average GBP4,926 more expensive, although this association cannot be ruled out to be random (probably due

[5]Keep in mind that regression is no magic. We are only discounting the effect of other confounding factors that we include in the model, not of *all* potentially confounding factors.

to the small relative number of new houses).

Finally, before we jump into introducing space in our models, let us modify our equation slightly to make it more useful when it comes to interpretating it. Many house price models in the literature is estimated in log-linear terms:

$$\log P_i = \alpha + \beta_1 NEW_i + \beta_2 IMD_i + \epsilon_i$$

This allows to interpret the coefficients more directly: as the percentual change induced by a unit increase in the explanatory variable of the estimate. To fit such a model, we can specify the logarithm of a given variable directly in the formula.

```
m2 <- lm('log(price) ~ new + imd_score', db)
summary(m2)
```

```
##
## Call:
## lm(formula = "log(price) ~ new + imd_score", data = db)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -4.3172 -0.3231 -0.0149  0.3063  5.2769
##
## Coefficients:
##               Estimate Std. Error t value Pr(>|t|)
## (Intercept) 12.2037784  0.0138634  880.28   <2e-16 ***
## newY         0.2456446  0.0198740   12.36   <2e-16 ***
## imd_score   -0.0169702  0.0003204  -52.96   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.5295 on 6321 degrees of freedom
## Multiple R-squared:  0.3101, Adjusted R-squared:  0.3099
## F-statistic:  1421 on 2 and 6321 DF,  p-value: < 2.2e-16
```

Looking at the results we can see a couple of differences with respect to the original specification. First, the estimates are substantially different numbers. This is because, although they consider the same variable, the look at it from different angles, and provide different interpretations. For example, the coefficient for the IMD, instead of being interpretable in terms of GBP, the unit of the dependent variable, it represents a percentage: a unit increase in the degree of deprivation is associated with a 0.2% decrease in the price of a house.[6]

[6] **EXERCISE** *How does the type of a house affect the price at which it is sold, given whether it is new and the level of deprivation of the area where it is located?* To answer this, fit a model as we have done but including additionally the variable `type`. In order to interpret the codes, check the reference at the Land Registry documentation.

Second, the variable `new` is significant in this case. This is probably related to the fact that, by taking logs, we are also making the dependent variable look more normal (Gaussian) and that allows the linear model to provide a better fit and, hence, more accurate estimates. In this case, a house being newly built, as compared to an old house, is overall 25% more expensive.

7.4 Spatial regression: a (very) first dip

Spatial regression is about *explicitly* introducing space or geographical context into the statistical framework of a regression. Conceptually, we want to introduce space into our model whenever we think it plays an important role in the process we are interested in, or when space can act as a reasonable proxy for other factors we cannot but should include in our model. As an example of the former, we can imagine how houses at the seafront are probably more expensive than those in the second row, given their better views. To illustrate the latter, we can think of how the character of a neighborhood is important in determining the price of a house; however, it is very hard to identify and quantify "character" perse, although it might be easier to get at its spatial variation, hence a case of space as a proxy.

Spatial regression is a large field of development in the econometrics and statistics literatures. In this brief introduction, we will consider two related but very different processes that give rise to spatial effects: spatial heterogeneity and spatial dependence. For more rigorous treatments of the topics introduced here, the reader is referred to Anselin (2003), Anselin and Rey (2014), and Gibbons et al. (2014).

7.5 Spatial heterogeneity

Spatial heterogeneity (SH) arises when we cannot safely assume the process we are studying operates under the same "rules" throughout the geography of interest. In other words, we can observe SH when there are effects on the outcome variable that are intrinsically linked to specific locations. A good example of this is the case of seafront houses above: we are trying to model the price of a house and, the fact some houses are located under certain conditions (i.e. by the sea), makes their price behave differently[7].

This somewhat abstract concept of SH can be made operational in a model in several ways. We will explore the following two: spatial fixed-effects (FE); and spatial regimes, which is a generalization of FE.

Spatial FE

Let us consider the house price example from the previous section to introduce a

[7] **QUESTION** How would you incorporate this into a regression model that extends the log-log equation of the previous section?

more general illustration that relates to the second motivation for spatial effects ("space as a proxy"). Given we are only including two explanatory variables in the model, it is likely we are missing some important factors that play a role at determining the price at which a house is sold. Some of them, however, are likely to vary systematically over space (e.g. different neighborhood characteristics). If that is the case, we can control for those unobserved factors by using traditional dummy variables but basing their creation on a spatial rule. For example, let us include a binary variable for every two-digit postcode in Liverpool, indicating whether a given house is located within such area (`1`) or not (`0`). Mathematically, we are now fitting the following equation:

$$\log P_i = \alpha_r + \beta_1 NEW_i + \beta_2 IMD_i + \epsilon_i$$

where the main difference is that we are now allowing the constant term, α, to vary by postcode r, α_r.

Programmatically, this is straightforward to estimate:

```
# Include `-1` to eliminate the constant term and include a dummy for every area
m3 <- lm('log(price) ~ pc + new + imd_score - 1', db)
summary(m3)
```

```
##
## Call:
## lm(formula = "log(price) ~ pc + new + imd_score - 1", data = db)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -4.2680 -0.2833 -0.0235  0.2599  5.6714
##
## Coefficients:
##            Estimate Std. Error t value Pr(>|t|)
## pcL1      11.697166   0.032137  363.98   <2e-16 ***
## pcL10     11.893524   0.076471  155.53   <2e-16 ***
## pcL11     11.902319   0.043194  275.55   <2e-16 ***
## pcL12     12.065776   0.029692  406.37   <2e-16 ***
## pcL13     11.865523   0.034893  340.06   <2e-16 ***
## pcL14     11.920922   0.044480  268.01   <2e-16 ***
## pcL15     12.039916   0.028555  421.64   <2e-16 ***
## pcL16     12.295535   0.034950  351.81   <2e-16 ***
## pcL17     12.275339   0.027978  438.75   <2e-16 ***
## pcL18     12.396475   0.024534  505.27   <2e-16 ***
## pcL19     12.162630   0.029697  409.56   <2e-16 ***
## pcL2      12.061443   0.100805  119.65   <2e-16 ***
## pcL20     11.928142   0.226932   52.56   <2e-16 ***
## pcL24     11.868363   0.045785  259.22   <2e-16 ***
```

```
## pcL25      12.234462   0.028557  428.42   <2e-16 ***
## pcL27      12.035241   0.092832  129.65   <2e-16 ***
## pcL28      11.438267   0.206323   55.44   <2e-16 ***
## pcL3       11.954453   0.029561  404.40   <2e-16 ***
## pcL4       11.718609   0.039575  296.11   <2e-16 ***
## pcL5       12.037267   0.055952  215.14   <2e-16 ***
## pcL6       11.898506   0.042918  277.24   <2e-16 ***
## pcL7       11.855374   0.044681  265.33   <2e-16 ***
## pcL8       12.034093   0.039801  302.36   <2e-16 ***
## pcL9       11.759056   0.033610  349.87   <2e-16 ***
## newY        0.310829   0.020947   14.84   <2e-16 ***
## imd_score -0.012008   0.000517  -23.23   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.5007 on 6298 degrees of freedom
## Multiple R-squared:  0.9981, Adjusted R-squared:  0.9981
## F-statistic: 1.305e+05 on 26 and 6298 DF,  p-value: < 2.2e-16
```

Econometrically speaking, what the postcode FE we have introduced imply is that, instead of comparing all house prices across Liverpool as equal, we only derive variation from within each postcode[8]. Remember that the interpretation of a β_k coefficient is the effect of variable k, *given all the other explanatory variables included remain constant*. By including a single variable for each area, we are effectively forcing the model to compare as equal only house prices that share the same value for each variable; in other words, only houses located within the same area. Introducing FE affords you a higher degree of isolation of the effects of the variables you introduce in your model because you can control for unobserved effects that align spatially with the distribution of the FE you introduce (by postcode, in our case).

Spatial regimes

At the core of estimating spatial FEs is the idea that, instead of assuming the dependent variable behaves uniformly over space, there are systematic effects following a geographical pattern that affect its behaviour. In other words, spatial FEs introduce econometrically the notion of spatial heterogeneity. They do this in the simplest possible form: by allowing the constant term to vary geographically. The other elements of the regression are left untouched and hence apply uniformly across space. The idea of spatial regimes (SRs) is to generalize the spatial FE approach to allow not only the constant term to vary but also any other explanatory variable. This implies that the equation we will

[8] Additionally, estimating spatial FE in our particular example also gives you an indirect measure of area *desirability*: since they are simple dummies in a regression explaining the price of a house, their estimate tells us about how much people are willing to pay to live in a given area. However, this interpretation does not necessarily apply in other contexts where introducing spatial FEs does make sense. **EXERCISE** *What is the most desired area to live in Liverpool?*

be estimating is:

$$\log P_i = \alpha_r + \beta_{1r} NEW_i + \beta_{2r} IMD_i + \epsilon_i$$

where we are not only allowing the constant term to vary by region (α_r), but also every other parameter (β_{kr}).

In R terms, this is more straightforward to estimate if `new` is expressed as `0` and `1`, rather than as factors:

```
# Create a new variable `newB` to store the binary form of `new`
db@data$newB <- 1
db[db@data$new=='N', 'newB'] <- 0
```

Also, given we are going to allow *every* coefficient to vary by regime, we will need to explicitly set a constant term that we can allow to vary:

```
db$one <- 1
```

Then, the estimation leverages the capabilities in model description of R formulas:

```
# `:` notation implies interaction variables
m4 <- lm('log(price) ~ 0 +(one + newB + imd_score):(pc)', db)
summary(m4)
```

```
##
## Call:
## lm(formula = "log(price) ~ 0 +(one + newB + imd_score):(pc)",
##     data = db)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -4.2051 -0.2506 -0.0182  0.2198  5.3507
##
## Coefficients: (8 not defined because of singularities)
##                  Estimate Std. Error t value Pr(>|t|)
## one:pcL1        11.983460   0.098675 121.444  < 2e-16 ***
## one:pcL10       11.911281   0.504210  23.624  < 2e-16 ***
## one:pcL11       11.774865   0.160864  73.198  < 2e-16 ***
## one:pcL12       12.157627   0.068165 178.356  < 2e-16 ***
## one:pcL13       11.821576   0.103946 113.728  < 2e-16 ***
## one:pcL14       11.821895   0.115040 102.763  < 2e-16 ***
## one:pcL15       12.317524   0.052051 236.645  < 2e-16 ***
## one:pcL16       12.761711   0.098709 129.287  < 2e-16 ***
## one:pcL17       12.240652   0.065074 188.103  < 2e-16 ***
## one:pcL18       12.695103   0.056510 224.654  < 2e-16 ***
## one:pcL19       12.489604   0.054177 230.533  < 2e-16 ***
```

```
## one:pcL2         12.164848   0.321558  37.831  < 2e-16 ***
## one:pcL20        11.069199   0.211083  52.440  < 2e-16 ***
## one:pcL24        10.876938   0.135326  80.376  < 2e-16 ***
## one:pcL25        12.427878   0.051147 242.982  < 2e-16 ***
## one:pcL27        11.022835   0.435730  25.297  < 2e-16 ***
## one:pcL28        11.300143   1.126387  10.032  < 2e-16 ***
## one:pcL3         11.977708   0.054781 218.647  < 2e-16 ***
## one:pcL4         11.653464   0.109113 106.802  < 2e-16 ***
## one:pcL5         11.544085   0.277000  41.675  < 2e-16 ***
## one:pcL6         11.574617   0.158022  73.247  < 2e-16 ***
## one:pcL7         11.617544   0.128402  90.478  < 2e-16 ***
## one:pcL8         11.586224   0.085036 136.251  < 2e-16 ***
## one:pcL9         11.837544   0.080452 147.139  < 2e-16 ***
## newB:pcL1        -0.298430   0.051078  -5.843 5.40e-09 ***
## newB:pcL10             NA         NA      NA       NA
## newB:pcL11        0.672479   0.089894   7.481 8.40e-14 ***
## newB:pcL12        0.977903   0.114281   8.557  < 2e-16 ***
## newB:pcL13       -0.659910   0.473773  -1.393 0.163705
## newB:pcL14        0.610482   0.132911   4.593 4.45e-06 ***
## newB:pcL15        0.744844   0.139732   5.331 1.01e-07 ***
## newB:pcL16             NA         NA      NA       NA
## newB:pcL17       -0.094580   0.095188  -0.994 0.320449
## newB:pcL18       -0.197115   0.122109  -1.614 0.106521
## newB:pcL19        0.393901   0.089607   4.396 1.12e-05 ***
## newB:pcL2              NA         NA      NA       NA
## newB:pcL20             NA         NA      NA       NA
## newB:pcL24        0.735316   0.072739  10.109  < 2e-16 ***
## newB:pcL25        0.196018   0.144527   1.356 0.175063
## newB:pcL27        0.356678   0.189946   1.878 0.060457 .
## newB:pcL28             NA         NA      NA       NA
## newB:pcL3        -0.261461   0.055228  -4.734 2.25e-06 ***
## newB:pcL4         1.087380   0.079152  13.738  < 2e-16 ***
## newB:pcL5         0.519945   0.094850   5.482 4.38e-08 ***
## newB:pcL6         0.695856   0.062461  11.141  < 2e-16 ***
## newB:pcL7              NA         NA      NA       NA
## newB:pcL8         0.334517   0.059368   5.635 1.83e-08 ***
## newB:pcL9              NA         NA      NA       NA
## imd_score:pcL1   -0.010489   0.003383  -3.101 0.001938 **
## imd_score:pcL10  -0.012413   0.011381  -1.091 0.275485
## imd_score:pcL11  -0.010637   0.003005  -3.540 0.000404 ***
## imd_score:pcL12  -0.017296   0.002625  -6.588 4.81e-11 ***
## imd_score:pcL13  -0.011004   0.002185  -5.036 4.89e-07 ***
## imd_score:pcL14  -0.010432   0.002469  -4.225 2.42e-05 ***
## imd_score:pcL15  -0.020737   0.001413 -14.681  < 2e-16 ***
## imd_score:pcL16  -0.050522   0.007703  -6.559 5.85e-11 ***
## imd_score:pcL17  -0.009763   0.002214  -4.410 1.05e-05 ***
```

```
## imd_score:pcL18 -0.032289   0.003799  -8.499  < 2e-16 ***
## imd_score:pcL19 -0.024629   0.001860 -13.242  < 2e-16 ***
## imd_score:pcL2  -0.016601   0.013654  -1.216 0.224084
## imd_score:pcL20         NA         NA      NA       NA
## imd_score:pcL24  0.004090   0.002391   1.711 0.087139 .
## imd_score:pcL25 -0.020461   0.001981 -10.327  < 2e-16 ***
## imd_score:pcL27  0.006626   0.007561   0.876 0.380911
## imd_score:pcL28 -0.009473   0.020367  -0.465 0.641850
## imd_score:pcL3  -0.006935   0.001747  -3.970 7.28e-05 ***
## imd_score:pcL4  -0.012032   0.001731  -6.952 3.96e-12 ***
## imd_score:pcL5  -0.006051   0.004009  -1.509 0.131280
## imd_score:pcL6  -0.008324   0.002367  -3.517 0.000440 ***
## imd_score:pcL7  -0.007517   0.002342  -3.209 0.001337 **
## imd_score:pcL8  -0.004064   0.001561  -2.603 0.009255 **
## imd_score:pcL9  -0.014124   0.002052  -6.882 6.47e-12 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.472 on 6260 degrees of freedom
## Multiple R-squared:  0.9984, Adjusted R-squared:  0.9983
## F-statistic: 5.966e+04 on 64 and 6260 DF,  p-value: < 2.2e-16
```

As we can see, there are a few `NA` values (e.g. `pcL10`). This has to do with the fact that there are not that many new houses, so some of the buckets in which the regimes split the data to estimate each parameter are empty. This can be readily seen by obtaining a quick cross tabulation of `pc` and `new`:

```
table(db$pc, db$new)
```

```
##
##         N   Y
##   L1  161 189
##   L10  47   0
##   L11 192  34
##   L12 326  18
##   L13 387   1
##   L14 144  19
##   L15 466  12
##   L16 212   0
##   L17 398  27
##   L18 439  16
##   L19 329  32
##   L2   25   0
##   L20   5   0
##   L24  97  84
##   L25 357  11
##   L27  22  10
```

```
##   L28   6   0
##   L3  272 101
##   L4  437  41
##   L5   97  46
##   L6  319  78
##   L7  201   0
##   L8  227 110
##   L9  329   0
```

To illustrate a correct regime estimation, we can focus only on `imd_score`[9]:

```
# `:` notation implies interaction variables
m5 <- lm('log(price) ~ 0 + (one + imd_score):pc', db)
summary(m5)
```

```
##
## Call:
## lm(formula = "log(price) ~ 0 + (one + imd_score):pc", data = db)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -4.4952 -0.2779 -0.0221  0.2477  5.4973
##
## Coefficients: (1 not defined because of singularities)
##                   Estimate Std. Error t value Pr(>|t|)
## one:pcL1        11.8965065  0.1030279 115.469  < 2e-16 ***
## one:pcL10       11.9112807  0.5325447  22.367  < 2e-16 ***
## one:pcL11       11.6203268  0.1684973  68.964  < 2e-16 ***
## one:pcL12       12.2106579  0.0716973 170.309  < 2e-16 ***
## one:pcL13       11.8099973  0.1094358 107.917  < 2e-16 ***
## one:pcL14       12.1229804  0.0998496 121.412  < 2e-16 ***
## one:pcL15       12.3753226  0.0537697 230.154  < 2e-16 ***
## one:pcL16       12.7617106  0.1042555 122.408  < 2e-16 ***
## one:pcL17       12.2248054  0.0666348 183.460  < 2e-16 ***
## one:pcL18       12.7033951  0.0594382 213.724  < 2e-16 ***
## one:pcL19       12.4767606  0.0571384 218.360  < 2e-16 ***
## one:pcL2        12.1648479  0.3396283  35.818  < 2e-16 ***
## one:pcL20       11.0691989  0.2229447  49.650  < 2e-16 ***
## one:pcL24       11.5214275  0.1260747  91.386  < 2e-16 ***
## one:pcL25       12.4351502  0.0537240 231.464  < 2e-16 ***
## one:pcL27       11.3770205  0.4148632  27.424  < 2e-16 ***
## one:pcL28       11.3001428  1.1896847   9.498  < 2e-16 ***
## one:pcL3        11.8873513  0.0542344 219.185  < 2e-16 ***
## one:pcL4        11.4097020  0.1137109 100.340  < 2e-16 ***
## one:pcL5        10.9973874  0.2729459  40.291  < 2e-16 ***
```

[9]Note this still returns a `NA` for the IMD estimate in `L20`. This is most likely due to the little amount of houses (five) sold in that area. The regression nevertheless serves the illustration

```
## one:pcL6        11.1816689  0.1626918  68.729  < 2e-16 ***
## one:pcL7        11.6175445  0.1356173  85.664  < 2e-16 ***
## one:pcL8        11.5086161  0.0886286 129.852  < 2e-16 ***
## one:pcL9        11.8375435  0.0849726 139.310  < 2e-16 ***
## imd_score:pcL1  -0.0131286  0.0035407  -3.708 0.000211 ***
## imd_score:pcL10 -0.0124128  0.0120210  -1.033 0.301837
## imd_score:pcL11 -0.0058373  0.0031009  -1.882 0.059824 .
## imd_score:pcL12 -0.0173734  0.0027727  -6.266 3.95e-10 ***
## imd_score:pcL13 -0.0107902  0.0023022  -4.687 2.83e-06 ***
## imd_score:pcL14 -0.0160870  0.0022604  -7.117 1.23e-12 ***
## imd_score:pcL15 -0.0219205  0.0014734 -14.878  < 2e-16 ***
## imd_score:pcL16 -0.0505217  0.0081354  -6.210 5.63e-10 ***
## imd_score:pcL17 -0.0093980  0.0023061  -4.075 4.65e-05 ***
## imd_score:pcL18 -0.0333885  0.0039476  -8.458  < 2e-16 ***
## imd_score:pcL19 -0.0228257  0.0019160 -11.913  < 2e-16 ***
## imd_score:pcL2  -0.0166013  0.0144213  -1.151 0.249707
## imd_score:pcL20         NA         NA      NA       NA
## imd_score:pcL24 -0.0020546  0.0024420  -0.841 0.400163
## imd_score:pcL25 -0.0205241  0.0020921  -9.810  < 2e-16 ***
## imd_score:pcL27  0.0020943  0.0075687   0.277 0.782014
## imd_score:pcL28 -0.0094733  0.0215112  -0.440 0.659671
## imd_score:pcL3  -0.0061811  0.0018374  -3.364 0.000773 ***
## imd_score:pcL4  -0.0066455  0.0017803  -3.733 0.000191 ***
## imd_score:pcL5   0.0039338  0.0037726   1.043 0.297115
## imd_score:pcL6  -0.0004620  0.0023860  -0.194 0.846480
## imd_score:pcL7  -0.0075165  0.0024737  -3.039 0.002387 **
## imd_score:pcL8  -0.0006921  0.0015228  -0.455 0.649469
## imd_score:pcL9  -0.0141241  0.0021676  -6.516 7.79e-11 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.4985 on 6277 degrees of freedom
## Multiple R-squared:  0.9982, Adjusted R-squared:  0.9982
## F-statistic: 7.282e+04 on 47 and 6277 DF,  p-value: < 2.2e-16
```

This allows us to get a separate constant term and estimate of the impact of IMD on the price of a house *for every postcode*[10].

7.6 Spatial dependence

As we have just discussed, SH is about effects of phenomena that are *explicitly linked* to geography and that hence cause spatial variation and clustering of val-

[10] **PRO EXERCISE** *How does the effect of IMD vary over space?* You can answer this by looking at the coefficients of `imd_score` over postcodes, but it would be much clearer if you could create a choropleth of the house locations where each dot is colored based on the value of the `imd_score` estimated for that postcode.

ues. This encompasses many of the kinds of spatial effects we may be interested in when we fit linear regressions. However, in other cases, our interest is on the effect of the *spatial configuration* of the observations, and the extent to which that has an effect on the outcome we are considering. For example, we might think that the price of a house not only depends on the level of deprivation where the house is located, but also whether is close to other highly deprived areas. This kind of spatial effect is fundamentally different from SH in that is it not related to inherent characteristics of the geography but relates to the characteristics of the observations in our dataset and, specially, to their spatial arrangement. We call this phenomenon by which the values of observations are related to each other through distance *spatial dependence* (Anselin, 1988).

Spatial Weights

There are several ways to introduce spatial dependence in an econometric framework, with varying degrees of econometric sophistication (see Anselin, 2003, for a good overview). Common to all of them however is the way space is formally encapsulated: through *spatial weights matrices (W)*[11]. These are NxN matrices with zero diagonals and every w_{ij} cell with a value that represents the degree of spatial connectivity/interaction between observations i and j. If they are not connected at all, $w_{ij} = 0$, otherwise $w_{ij} > 0$ and we call i and j neighbors. The exact value in the latter case depends on the criterium we use to define neighborhood relations. These matrices also tend to be row-standardized so the sum of each row equals to one.

A related concept to spatial weight matrices is that of *spatial lag*. This is an operator that multiplies a given variable y by a spatial weight matrix:

$$y_{lag} = Wy$$

If W is row-standardized, y_{lag} is effectively the average value of y in the neighborhood of each observation. The individual notation may help clarify this:

$$y_{lag-i} = \sum_j w_{ij} y_j$$

where y_{lag-i} is the spatial lag of variable y at location i, and j sums over the entire dataset. If W is row-standardized, y_{lag-i} becomes an average of y weighted by the spatial criterium defined in W.

Given that spatial weights matrices are not the focus of this tutorial, we will stick to a very simple case. Since we are dealing with points, we will use K-nn weights, which take the k nearest neighbors of each observation as neighbors and assign a value of one, assigning everyone else a zero. We will use $k = 150$ to get a good degree of variation and sensible results. If your computer is struggles

[11] If you need to refresh your knowledge on spatial weight matrices, check Lecture 5 of **?**

to compute the following lines of code, you can replace 50 by a lowed number. Technically speaking is the same thing, but the probability that you will pick up only houses in the same LSOA (and hence with exactly the same IMD score) will be higher.

```
# Because some rows are different units on the same house, slightly
# jitter the locations to break ties
xy.jit <- jitter(db@coords)
# Create knn list of each house
hnn <- knearneigh(xy.jit, k=50)
# Create nb object
hnb <- knn2nb(hnn)
# Create spatial weights matrix (note it row-standardizes by default)
hknn <- nb2listw(hnb)
```

We can inspect the weights created by simply typing the name of the object:

```
hknn
```

```
## Characteristics of weights list object:
## Neighbour list object:
## Number of regions: 6324
## Number of nonzero links: 316200
## Percentage nonzero weights: 0.7906388
## Average number of links: 50
## Non-symmetric neighbours list
##
## Weights style: W
## Weights constants summary:
##      n       nn   S0       S1      S2
## W 6324 39992976 6324 230.5488 25810.7
```

Exogenous spatial effects

Let us come back to the house price example we have been working with. So far, we have hypothesized that the price of a house sold in Liverpool can be explained using information about whether it is newly built, the level of deprivation of the area where it is located, and its postcode. However, it is also reasonable to think that prospective house owners care about the larger area around a house, not only about its immediate surroundings, and would be willing to pay more for a house that was close to nicer areas, everything else being equal. How could we test this idea?

The most straightforward way to introduce spatial dependence in a regression is by considering not only a given explanatory variable, but also its spatial lag. In our example case, in addition to including the level of deprivation in the area of the house, we will include its spatial lag. In other words, we will be saying that it is not only the level of deprivation of the area where a house is located but also that of the surrounding locations that helps explain the final price at which

a house is sold. Mathematically, this implies estimating the following model:

$$\log P_i = \alpha + \beta_1 NEW_i + \beta_2 IMD_i + \beta_3 IMD_{lag-i} + \epsilon_i$$

Let us first compute the spatial lag of imd_score:

```
db@data$w_imd_score <- lag.listw(hknn, db@data$imd_score)
```

And then we can include it in our previous specification. Note that we apply the log to the lag, not the reverse:

```
# `:` notation implies interaction variables
m6 <- lm('log(price) ~ new + imd_score + w_imd_score', db)
summary(m6)
```

```
##
## Call:
## lm(formula = "log(price) ~ new + imd_score + w_imd_score", data = db)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -4.2907 -0.3014 -0.0151  0.2819  5.2631
##
## Coefficients:
##               Estimate Std. Error t value Pr(>|t|)
## (Intercept) 12.2812400  0.0145293  845.27  < 2e-16 ***
## newY         0.2475108  0.0195190   12.68  < 2e-16 ***
## imd_score   -0.0042200  0.0008921   -4.73 2.29e-06 ***
## w_imd_score -0.0148007  0.0009690  -15.27  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.5201 on 6320 degrees of freedom
## Multiple R-squared:  0.3347, Adjusted R-squared:  0.3344
## F-statistic:  1060 on 3 and 6320 DF,  p-value: < 2.2e-16
```

As we can see, the lag is not only significative and negative (as expected), but its effect seems to be even larger that that of the house itself. Taken literally, this would imply that prospective owners value more the area of the surrounding houses than that of the actual house they buy. However, it is important to remember how these variables have been constructed and what they really represent. Because the IMD score is not exactly calculated at the house level, but at the area level, many of the surrounding houses will share that so, to some extent, the IMD of neighboring houses is that of the house itself[12]. This is likely to be affecting the final parameter, and it is a reminder

[12] **EXERCISE** *How do results change if you modify the number of neighbors included to compute the K-nn spatial weight matrix?* Replace the originak k used and re-run the

and an illustration that we cannot take model results as universal truth but we need to use them as tools to inform analysis, couple with theory and what we know about the particular question of analysis. Nevertheless, the example does illustrate how to introduce spatial dependence in a regression framework in a fairly straight forward way.

A note on more advanced spatial regression

Introducing a spatial lag of an explanatory variable, as we have just seen, is the most straightforward way of incorporating the notion of spatial dependence in a linear regression framework. It does not require additional changes, it can be estimated with OLS, and the interpretation is rather similar to interpreting non-spatial variables. The field of spatial econometrics however is a much broader one and has produced over the last decades many techniques to deal with spatial effects and spatial dependence in different ways. Although this might be an over simplification, one can say that most of such efforts for the case of a single cross-section are focused on two main variations: the spatial lag and the spatial error model. Both are similar to the case we have seen in that they are based on the introduction of a spatial lag, but they differ in the component of the model they modify and affect.

The spatial lag model introduces a spatial lag of the *dependent* variable. In the example we have covered, this would translate into:

$$\log P_i = \alpha + \rho \log P_{lag-i} + \beta_1 NEW_i + \beta_2 IMD_i + \epsilon_i$$

Although it might not seem very different from the previous equation, this model violates the exogeneity assumption, crucial for OLS to work.

Equally, the spatial error model includes a spatial lag in the *error* term of the equation:

$$\log P_i = \alpha + \beta_{1r} NEW_i + \beta_{2r} IMD_i + u_i$$

$$u_i = u_{lag-i} + \epsilon_i$$

Again, although similar, one can show this specification violates the assumptions about the error term in a classical OLS model.

Both the spatial lag and error model violate some of the assumptions on which OLS relies and thus render the technique unusable. Much of the efforts have thus focused on coming up with alternative methodologies that allow unbiased, robust, and efficient estimation of such models. A survey of those is beyond the scope of this note, but the interested reader is referred to Anselin (1988), Anselin (2003), and Anselin and Rey (2014) for further reference.

regressions. Try to interpret the results and the (potential) differences with the original ones.

7.7 Predicting house prices

So far, we have seen how exploit the output of a regression model to evaluate the role different variables play in explaining another one of interest. However, once fit, a model can also be used to obtain predictions of the dependent variable given a new set of values for the explanatory variables. We will finish this session by dipping our toes in predicting with linear models.

The core idea is that once you have estimates for the way in which the explanatory variables can be combined to explain the dependent one, you can plug new values on the explanatory side of the model and combine them following the model estimates to obtain predictions. In the example we have worked with, you can imagine this application would be useful to obtain valuations of a house, given we know the IMD of the area where the house is located and whether it is a newly built house or not.

Conceptually, predicting in linear regression models involves using the estimates of the parameters to obtain a value for the dependent variable:

$$\bar{\log P_i} = \bar{\alpha} + \bar{\beta_{1r}} NEW_i^* + \bar{\beta_{2r}} IMD_i^*$$

where $\bar{\log P_i}$ is our predicted value, and we include the $\bar{}$ sign to note that it is our estimate obtained from fitting the model. We use the * sign to note that those can be new values for the explanatory variables, not necessarily those used to fit the model.

Technically speaking, prediction in linear models is fairly streamlined in R. Suppose we are given data for a new house which is to be put in the market. We know it is been newly built on an area with an IMD score of 75, but surrounded by areas that, on average, have a score of 50. Let us record the data first:

```
new.house <- data.frame(new='Y', imd_score=75, w_imd_score=50)
```

To obtain the prediction for its price, we can use the `predict` method:

```
new.price <- predict(m6, new.house)
new.price
```

```
##        1
## 11.47221
```

Now remember we were using the log of the price as dependent variable. If we want to recover the actual price of the house, we need to take its exponent:

```
exp(new.price)
```

```
##       1
## 96010.3
```

According to our model, the house would be worth GBP96,060.29[13].

7.8 References

[13] **EXERCISE** *How would the price change if the surrounding houses did not have an average of 50 but of 80?* Obtain a new prediction and compare it with the original one.

Chapter 8

Multilevel Modelling - Part 1

This chapter[1] provides an introduction to multi-level data structures and multi-level modelling.

The content of this chapter is based on:

- Gelman and Hill (2006b) provides an excellent and intuitive explanation of multilevel modelling and data analysis in general. Read Part 2A for a really good explanation of multilevel models.
- for Multilevel Modelling (nd) is an useful online resource on multilevel modelling and is free!

This Chapter is part of Spatial Analysis Notes, a compilation hosted as a GitHub repository that you can access it in a few ways:

- As a download of a `.zip` file that contains all the materials.
- As an html website.
- As a pdf document
- As a GitHub repository.

8.1 Dependencies

This chapter uses the following libraries: Ensure they are installed on your machine[2] before loading them executing the following code chunk:

[1] This note is part of Spatial Analysis Notes Multilevel Modelling – Random Intercept Multilevel Model by Francisco Rowe is licensed under a Creative Commons Attribution-NonCommercial-ShareAlike 4.0 International License.

[2] You can install package `mypackage` by running the command `install.packages("mypackage")` on the R prompt or through the `Tools --> Install`

```
# Data manipulation, transformation and visualisation
library(tidyverse)
# Nice tables
library(kableExtra)
# Simple features (a standardised way to encode vector data ie. points, lines, polygon
library(sf)
# Spatial objects conversion
library(sp)
# Thematic maps
library(tmap)
# Colour palettes
library(RColorBrewer)
# More colour palettes
library(viridis) # nice colour schemes
# Fitting multilevel models
library(lme4)
# Tools for extracting information generated by lme4
library(merTools)
# Exportable regression tables
library(jtools)
library(stargazer)
library(sjPlot)
```

8.2 Data

For this chapter, we will data for Liverpool from England's 2011 Census. The original source is the Office of National Statistics and the dataset comprises a number of selected variables capturing demographic, health and socio-economic attributes of the local resident population at four geographic levels: Output Area (OA), Lower Super Output Area (LSOA), Middle Super Output Area (MSOA) and Local Authority District (LAD). The variables include population counts and percentages. For a description of the variables, see the readme file in the mlm data folder.[3]

Let us read the data:

```
# clean workspace
rm(list=ls())
# read data
oa_shp <- st_read("data/mlm/OA.shp")
```

We can now attach and visualise the structure of the data.

`Packages...` menu in RStudio.

[3] Read the file in R by executing `read_tsv("data/mlm/readme.txt")`

```
# attach data frame
attach(oa_shp)

# sort data by oa
oa_shp <- oa_shp[order(oa_cd),]
head(oa_shp)
```

```
## Simple feature collection with 6 features and 19 fields
## geometry type:  MULTIPOLYGON
## dimension:      XY
## bbox:           xmin: 335056 ymin: 389163 xmax: 336155 ymax: 389642
## projected CRS:  Transverse_Mercator
##       oa_cd   lsoa_cd   msoa_cd    lad_cd      ward_nm  dstrt_nm    cnty_nm
## 1 E00032987 E01006515 E02001383 E08000012    Riverside Liverpool Merseyside
## 2 E00032988 E01006514 E02001383 E08000012 Princes Park Liverpool Merseyside
## 3 E00032989 E01033768 E02001383 E08000012 Princes Park Liverpool Merseyside
## 4 E00032990 E01033768 E02001383 E08000012 Princes Park Liverpool Merseyside
## 5 E00032991 E01033768 E02001383 E08000012 Princes Park Liverpool Merseyside
## 6 E00032992 E01033768 E02001383 E08000012 Princes Park Liverpool Merseyside
##   cntry_nm pop     age_60     unemp      lat      long    males   lt_ill
## 1  England 198 0.11616162 0.1130435 53.39821 -2.976786 46.46465 19.19192
## 2  England 348 0.16954023 0.1458333 53.39813 -2.969072 58.33333 33.62069
## 3  England 333 0.09009009 0.1049724 53.39778 -2.965290 64.26426 23.72372
## 4  England 330 0.15151515 0.1329787 53.39802 -2.963597 59.69697 23.03030
## 5  England 320 0.04687500 0.1813725 53.39706 -2.968030 60.62500 25.00000
## 6  England 240 0.05833333 0.2519685 53.39679 -2.966494 57.91667 28.33333
##     Bhealth VBhealth  no_qual   manprof                       geometry
## 1  6.565657 1.515152 24.69136  7.643312 MULTIPOLYGON (((335187 3894...
## 2 10.344828 1.436782 14.84848 13.375796 MULTIPOLYGON (((335834 3895...
## 3  6.606607 2.102102 15.38462 10.204082 MULTIPOLYGON (((335975.2 38...
## 4  5.151515 2.424242 17.91531 15.224913 MULTIPOLYGON (((336030.8 38...
## 5  8.750000 2.187500 12.58278 11.333333 MULTIPOLYGON (((335804.9 38...
## 6  6.666667 2.916667 27.47748  5.479452 MULTIPOLYGON (((335804.9 38...
```

The data are hierarchically structured: OAs nested within LSOAs; LSOAs nested within MSOAs; and, MSOAs nested within LADs. Observations nested within higher geographical units may be correlated.

This is one type of hierarchical structure. There is a range of data structures:

- Strict nested data structures eg. an individual unit is nested within only one higher unit
- Repeated measures structures eg. various measurements for an individual unit
- Crossed classified structures eg. individuals may work and live in different neighbourhoods

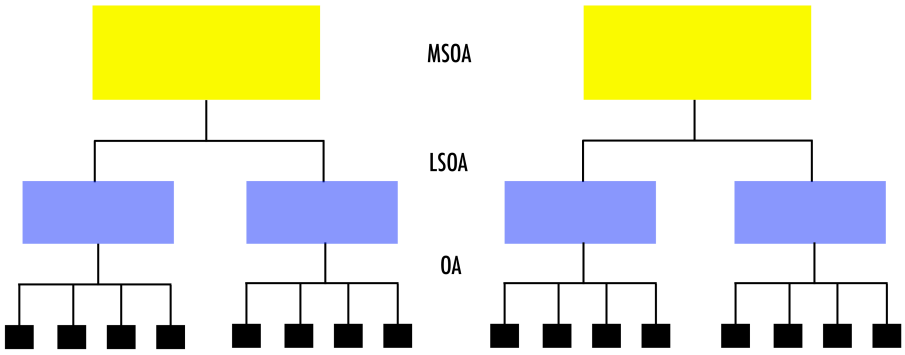


Figure 8.1: Fig. 1. Data Structure.

- Multiple membership structure eg. individuals may have two different work places

Why should we care about the structure of the data?

- *Draw correct statistical inference*: Failing to recognise hierarchical structures will lead to underestimated standard errors of regression coefficients and an overstatement of statistical significance. Standard errors for the coefficients of higher-level predictor variables will be the most affected by ignoring grouping.
- *Link context to individual units*: We can link and understand the extent of group effects on individual outcomes eg. how belonging to a certain socio-economic group influences on future career opportunities.
- *Spatial dependency*: Recognising the hierarchical structure of data may help mitigate the effects of severe spatial autocorrelation.

Quickly, let us get a better idea about the data and look at the number of OAs nested within LSOAs and MSOAs

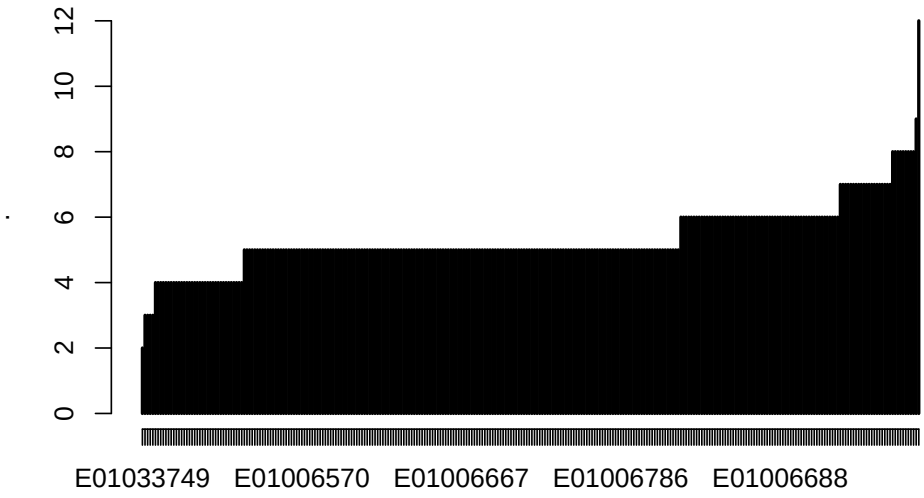
```
# mean of nested OAs within LSOAs and MSOAs
lsoa_cd %>% table() %>%
  mean() %>%
  round(, 2)
```

```
## [1] 5
```

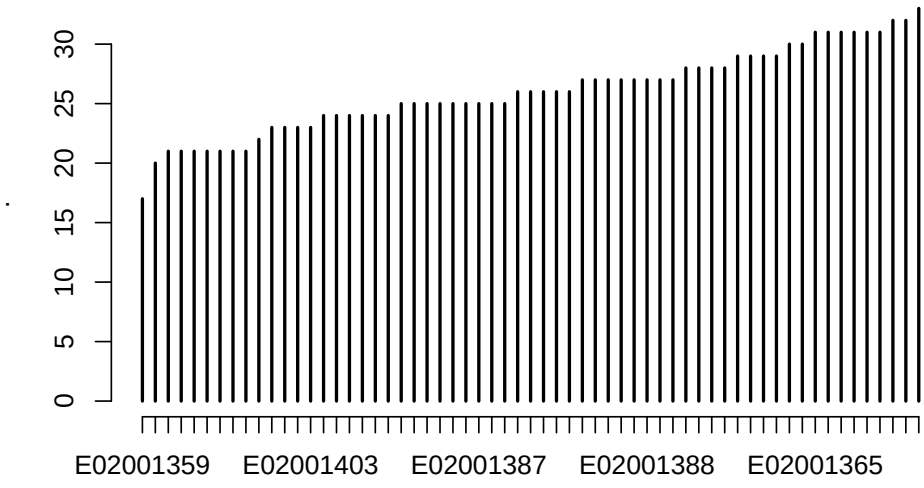
```
msoa_cd %>% table() %>%
  mean() %>%
  round(, 2)
```

```
## [1] 26
```

```
# number of OAs nested within LSOAs and MSOAs
lsoa_cd %>% table() %>%
  sort() %>%
  plot()
```




```
msoa_cd %>% table() %>%
  sort() %>%
  plot()
```

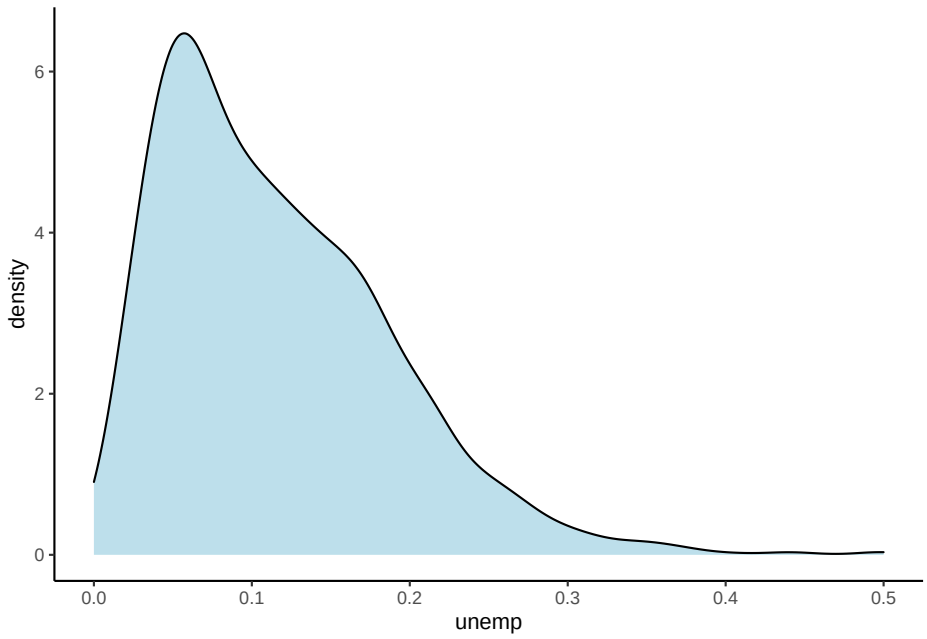


8.3 Modelling

We should now be persuaded that ignoring the hierarchical structure of data may be a major issue. Let us now use a simple example to understand the intuition of multilevel model using the census data. We will seek to understand the spatial distribution of the proportion of population in unemployment in Liverpool, particularly why and where concentrations in this proportion occur. To illustrate the advantages of taking a multilevel modelling approach, we will start by estimating a linear regression model and progressively building complexity. We will first estimate a model and then explain the intuition underpinning the process. We will seek to gain a general understanding of multilevel modelling. If you are interested in the statistical and mathemathical formalisation of the underpinning concepts, please refer to Gelman and Hill (2006b).

We first need to want to understand our dependent variable: its density ditribution;

```
ggplot(data = oa_shp) +
geom_density(alpha=0.8, colour="black", fill="lightblue", aes(x = unemp)) +
   theme_classic()
```




```
summary(unemp)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
## 0.00000 0.05797 0.10256 0.11581 0.16129 0.50000
```

and, its spatial distribution:

```
# ensure geometry is valid
oa_shp = sf::st_make_valid(oa_shp)

# create a map
legend_title = expression("% unemployment")
map_oa = tm_shape(oa_shp) +
  tm_fill(col = "unemp", title = legend_title, palette = magma(256, begin = 0.25, end = 1), style
  tm_borders(col = "white", lwd = .01)  +
  tm_compass(type = "arrow", position = c("right", "top") , size = 4) +
  tm_scale_bar(breaks = c(0,1,2), text.size = 0.5, position =  c("center", "bottom"))
map_oa
```

Let us look at those areas:

```
# high %s
oa_shp %>% filter(unemp > 0.2) %>%
  dplyr::select(oa_cd, pop, unemp)
```

```
## Simple feature collection with 203 features and 3 fields
```

```
## geometry type:  GEOMETRY
## dimension:      XY
## bbox:           xmin: 333993.8 ymin: 379748.5 xmax: 345600.2 ymax: 397681.5
## projected CRS:  Transverse_Mercator
## First 10 features:
##        oa_cd pop     unemp                       geometry
## 1  E00032992 240 0.2519685 MULTIPOLYGON (((335804.9 38...
## 2  E00033008 345 0.2636364 MULTIPOLYGON (((335080 3885...
## 3  E00033074 299 0.2075472 MULTIPOLYGON (((336947.3 38...
## 4  E00033075 254 0.2288136 MULTIPOLYGON (((336753.6 38...
## 5  E00033080 197 0.2647059 MULTIPOLYGON (((338196 3870...
## 6  E00033103 298 0.2148148 MULTIPOLYGON (((340484 3854...
## 7  E00033116 190 0.2156863 MULTIPOLYGON (((341960.7 38...
## 8  E00033134 190 0.2674419 MULTIPOLYGON (((337137 3930...
## 9  E00033137 289 0.2661290 MULTIPOLYGON (((337363.8 39...
## 10 E00033138 171 0.3561644 MULTIPOLYGON (((337481.5 39...
```

8.3.1 Baseline Linear Regression Model

Now let us estimate a simple linear regression model with the intercept only:

```
# specify a model equation
eq1 <- unemp ~ 1
model1 <- lm(formula = eq1, data = oa_shp)

# estimates
summary(model1)
```

```
##
## Call:
## lm(formula = eq1, data = oa_shp)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -0.11581 -0.05784 -0.01325  0.04548  0.38419
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) 0.115812   0.001836   63.09   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.07306 on 1583 degrees of freedom
```

To understand the differences between the linear regression model and multilevel models, let us consider the model we have estimated:

$$y_i = \beta_0 + e_i$$

where y_i represents the proportion of the unemployed resident population in the OA i; β_0 is the regression intercept and measures the average proportion of the unemployed resident population across OAs; and, e_i is the error term. But how do we deal with the hierarchical structure of the data?

8.3.1.1 Limitations

Before looking at the answer, let's first understand some of the key limitations of the linear regression model to handle the hierarchical structure of data. A key limitation of the linear regression model is that it only captures average relationships in the data. It does not capture variations in the relationship between variables across areas or groups. Another key limitation is that the linear regression model can capture associations at either macro or micro levels, but it does not simultaneously measure their interdependencies.

To illustrate this, let us consider the regression intercept. It indicates that the average percentage of unemployed population at the OA level is 0.12 but this model ignores any spatial clustering ie. the percentage of unemployed population tends to be similar across OAs nested within a same LSOA or MSOA. A side effect of ignoring this is that our standard errors are biased, and thus claims about statistical significance based on them would be misleading. Additionally, this situation also means we cannot explore variations in the percentage of unemployed population across LSOAs or MSOAs ie. how the percentage of unemployed population may be dependent on various contextual factors at these geographical scales.

8.3.1.2 Fixed Effect Approach

An alternative approach is to adopt a fixed effects approach, or no-pooling model; that is, adding dummy variables indicating the group classification into the regression model eg. the way OAs is nested within LSOAs (or MSOAs). This approach has limitations. First, there is high risk of overfitting. The number of groups may be too large, relative to the number of observations. Second, the estimation of multiple parameters may be required so that measuring differences between groups may be challenging. Third, a fixed effects approach does not allow including group-level explanatory variables. You can try fitting a linear regression model extending our estimated model to include dummy variables for individual LSOAs (and/or MSOAs) so you can compare this to the multilevel model below.

An alternative is fitting separate linear regression models for each group. This approach is not always possible if there are groups with small sizes.

8.4 Multilevel Modelling: Random Intercept Model

We use multilevel modelling to account for the hierarchical nature of the data by explicitly recognising that OAs are nested within LSOAs and MSOAs. Multilevel models can easily be estimated using in R using the package `lme4`. We implement an two-level model to allow for variation across LSOAs. We estimate an only intercept model allowing for variation across LSOAs. In essence, we are estimating a model with varying intercept coefficient by LSOA. As you can see in the code chunk below, the equation has an additional component. This is the group component or LSOA effect. The `(1 | lsoa_cd)` means that we are allowing the intercept, represented by 1, to vary by LSOA.

```
# specify a model equation
eq2 <- unemp ~ 1 + (1 | lsoa_cd)
model2 <- lmer(eq2, data = oa_shp)

# estimates
summary(model2)
```

```
## Linear mixed model fit by REML ['lmerMod']
## Formula: unemp ~ 1 + (1 | lsoa_cd)
##    Data: oa_shp
##
## REML criterion at convergence: -4382.6
##
## Scaled residuals:
##     Min      1Q  Median      3Q     Max
## -2.8741 -0.5531 -0.1215  0.4055  5.8207
##
## Random effects:
##  Groups   Name        Variance Std.Dev.
##  lsoa_cd  (Intercept) 0.002701 0.05197
##  Residual             0.002575 0.05074
## Number of obs: 1584, groups:  lsoa_cd, 298
##
## Fixed effects:
##             Estimate Std. Error t value
## (Intercept) 0.114316   0.003277   34.89
```

We can estimate a three-level model by adding `(1 | msoa_cd)` to allow the intercept to also vary by MSOAs and account for the nesting structure of LSOAs within MSOAs.

```
# specify a model equation
eq3 <- unemp ~ 1 + (1 | lsoa_cd) + (1 | msoa_cd)
model3 <- lmer(eq3, data = oa_shp)
```

```
# estimates
summary(model3)
```

```
## Linear mixed model fit by REML ['lmerMod']
## Formula: unemp ~ 1 + (1 | lsoa_cd) + (1 | msoa_cd)
##    Data: oa_shp
##
## REML criterion at convergence: -4529.3
##
## Scaled residuals:
##     Min      1Q  Median      3Q     Max
## -2.5624 -0.5728 -0.1029  0.4228  6.1363
##
## Random effects:
##  Groups   Name        Variance  Std.Dev.
##  lsoa_cd  (Intercept) 0.0007603 0.02757
##  msoa_cd  (Intercept) 0.0020735 0.04554
##  Residual             0.0025723 0.05072
## Number of obs: 1584, groups:  lsoa_cd, 298; msoa_cd, 61
##
## Fixed effects:
##             Estimate Std. Error t value
## (Intercept) 0.115288   0.006187   18.64
```

We see two sets of coefficients: *fixed effects* and *random effects*. *Fixed effects* correspond to the standard linear regression coefficients. Their interpretation is as usual. *Random effects* are the novelty. It is a term in multilevel modelling and refers to varying coefficients i.e. the randomness in the probability of the model for the group-level coefficients. Specifically they relate to estimates of the average variance and standard deviation within groups (i.e. LSOAs or MSOAs). Intiutively, variance and standard deviation indicate the extent to which the intercept, on average, varies by LSOAs and MSOAs.

More formally, we first estimated the simplest regression model which is an intercept-only model and equivalent to the sample mean (i.e. the *fixed* part of the model):

$$y_{ijk} = \mu + e_{ijk}$$

and then we made the *random* part of the model (e_{ijk}) more complex to account for the hierarchical structure of the data by estimating the following three-level regression model:

$$y_{ijk} = \mu + u_{i..} + u_{ij.} + e_{ijk}$$

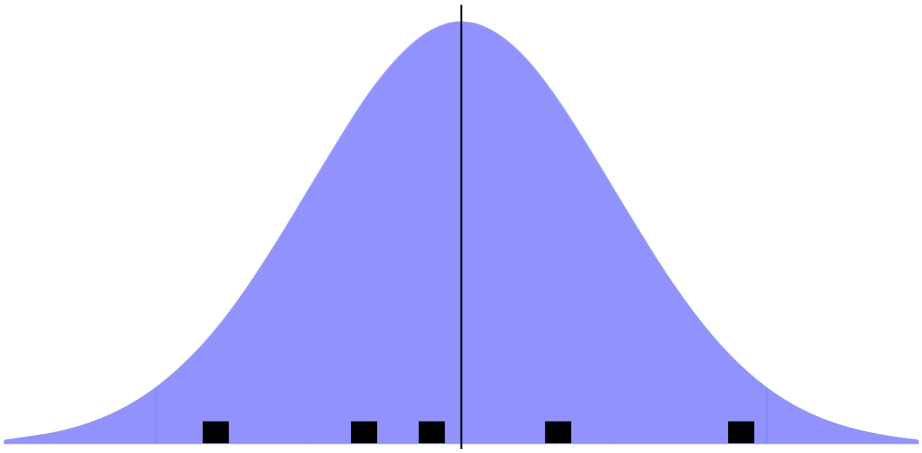


Figure 8.2: Fig. 2. Variation of observations around their level 1 group mean.

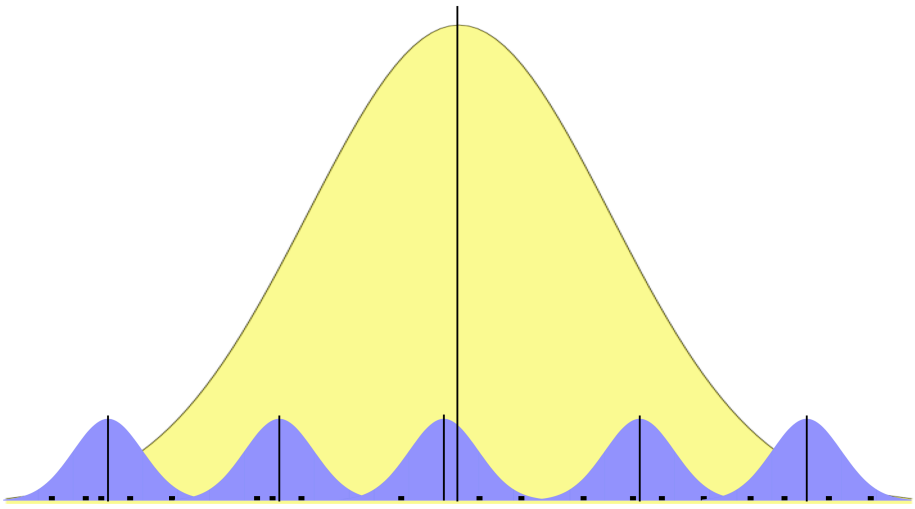


Figure 8.3: Fig. 3. Variation of level 1 group mean around their level 2 group mean.

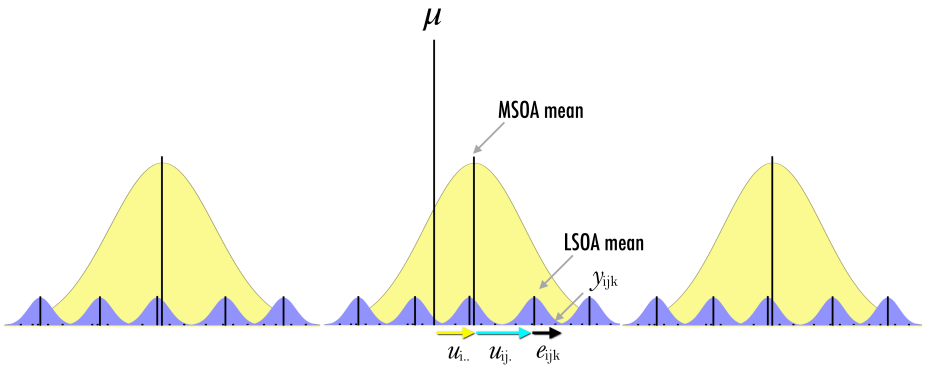


Figure 8.4: Fig. 4. Grand mean.

where y_{ijk} represents the proportion of unemployed population in OA i nested within LSOA j and MSOA k; μ represents the sample mean and the *fixed* part of the model; e_{ijk} is the deviation of an observation from its LSOA mean; $u_{ij.}$ is the deviation of the LSOA mean from its MSOA mean; $u_{i..}$ is the deviation of the MSOA mean from the fixed part of the model μ. Conceptually, this model is decomposing the variance of the model in terms of the hierarchical structure of the data. It is partitioning the observation's residual into three parts or *variance components*. These components measure the relative extent of variation of each hierarchical level ie. LSOA, MSOA and grand means. To estimate the set of residuals, they are assumed to follow a normal distribution and are obtained after fitting the model and are based on the estimates of the model parameters (i.e. intercept and variances of the random parameters).

Let's now return to our three-level model (reported again below), we see that the intercept or fixed part of the model is the same as for the linear regression. The multilevel model reports greater standard errors. Multilevel models capture the hierarchical structure of the data and thus more precisely estimate the standard errors for our parameters.

```
# report model 3
summary(model3)
```

```
## Linear mixed model fit by REML ['lmerMod']
## Formula: unemp ~ 1 + (1 | lsoa_cd) + (1 | msoa_cd)
##    Data: oa_shp
##
## REML criterion at convergence: -4529.3
##
## Scaled residuals:
##     Min      1Q  Median      3Q     Max
```

```
## -2.5624 -0.5728 -0.1029  0.4228  6.1363
##
## Random effects:
##  Groups   Name        Variance  Std.Dev.
##  lsoa_cd  (Intercept) 0.0007603 0.02757
##  msoa_cd  (Intercept) 0.0020735 0.04554
##  Residual             0.0025723 0.05072
## Number of obs: 1584, groups:  lsoa_cd, 298; msoa_cd, 61
##
## Fixed effects:
##             Estimate Std. Error t value
## (Intercept) 0.115288   0.006187   18.64
```

8.4.1 Interpretation

Fixed effects

We start by examining the fixed effects or estimated model averaging over LSOAs and MSOAs, $y_{ijk} = 0.115288$ which can also be called by executing:

```
fixef(model3)
```

```
## (Intercept)
##   0.1152881
```

Th estimated intercept indicates that the overall mean taken across LSOAs and MSOAs is estimated as `0.115288` and is statistically significant at `5%` significance.

Random effects

The set of random effects contains three estimates of variance and standard deviation and refer to the variance components discussed above. The `lsoa_cd`, `msoa_cd` and `Residual` estimates indicate that the extent of estimated LSOA-, MSOA- and individual-level variance is `0.0007603`, `0.0020735` and `0.0025723`, respectively.

8.4.2 Variance Partition Coefficient (VPC)

The purpose of multilevel models is to partition variance in the outcome between the different groupings in the data. We thus often want to know the percentage of variation in the dependent variable accounted by differences across groups i.e. what proportion of the total variance is attributable to variation within-groups, or how much is found between-groups. The statistic to obtain this is termed the variance partition coefficient (VPC), or intraclass correlation.[4] For

[4]The VPC is equal to the intra-class correlation coefficient which is the correlation between the observations of the dependent variable selected randomly from the same group. For instance, if the VPC is 0.1, we would say that 10% of the variation is between groups and 90% within. The correlation between randomly chosen pairs of observations belonging to the

our case, the VPC at the LSOA level indicates that 14% of the variation in percentage of unemployed resident population across OAs can be explained by differences across LSOAs. What is the VPC at the MSOA level?

```
vpc_lsoa <- 0.0007603 / (0.0007603 + 0.0020735 + 0.0025723)
vpc_lsoa * 100
```

```
## [1] 14.06374
```

You can also obtain the VPC by executing:

```
#summ(model3)
```

8.4.3 Uncertainty of Estimates

You may have noticed that `lme4` does not provide p-values, because of various reasons as explained by Doug Bates, one of the author of `lme4`. These explanations mainly refer to the complexity of dealing with varying sample sizes at a given hierarchical level. The number of observations at each hierarchical level varies across individual groupings (i.e. LSOA or MSOA). It may even be one single observation. This has implications for the distributional assumptions, denominator degrees of freedom and how to approximate a "best" solution. Various approaches exist to compute the statistical significance of estimates. We use the `confint` function available within `lme4` to obtain confidence intervals.

```
confint(model3, level = 0.95)
```

```
## Computing profile confidence intervals ...
```

```
##                   2.5 %     97.5 %
## .sig01      0.02360251 0.03189046
## .sig02      0.03707707 0.05562307
## .sigma      0.04882281 0.05273830
## (Intercept) 0.10307341 0.12751103
```

`.sig01` refers to the LSOA level; `.sig02` refers to the MSOA level; and, `.sigma` refers to the OA level.

8.4.4 Assessing Group-level Variation

Estimated regression coefficients

In multilevel modelling, our primary interest is in knowing differences across groups. To visualise the estimated model within each group (ie. LSOA and MSOA), we type:

```
coef_m3 <- coef(model3)
head(coef_m3$lsoa_cd,5)
```

same group is 0.1.

```
##           (Intercept)
## E01006512  0.09915456
## E01006513  0.09889615
## E01006514  0.09297051
## E01006515  0.09803754
## E01006518  0.09642939
```

The results indicate that the estimated regression line is $y = 0.09915456$ for LSOA `E01006512`; $y = 0.09889615$ for LSOA `E01006513` and so forth. Try getting the estimated model within each MSOA.

Random effects

We can look at the estimated group-level (or LSOA-level and MSOA-level) errors; that is, *random effects*:

```
ranef_m3 <- ranef(model3)
head(ranef_m3$lsoa_cd, 5)
```

```
##           (Intercept)
## E01006512 -0.01613353
## E01006513 -0.01639194
## E01006514 -0.02231758
## E01006515 -0.01725055
## E01006518 -0.01885870
```

Group-level errors indicate how much the intercept is shifted up or down in particular groups (ie. LSOAs or MSOAs). Thus, for example, in LSOA `E01006512`, the estimated intercept is `-0.01613353` lower than average, so that the regression line is `(0.1152881 - 0.01613353) = 0.09915457` which is what we observed from the call to `coef()`.

We can also obtain group-level errors (*random effects*) by using a simulation approach, labelled "Empirical Bayes" and discussed here. To this end, we run:

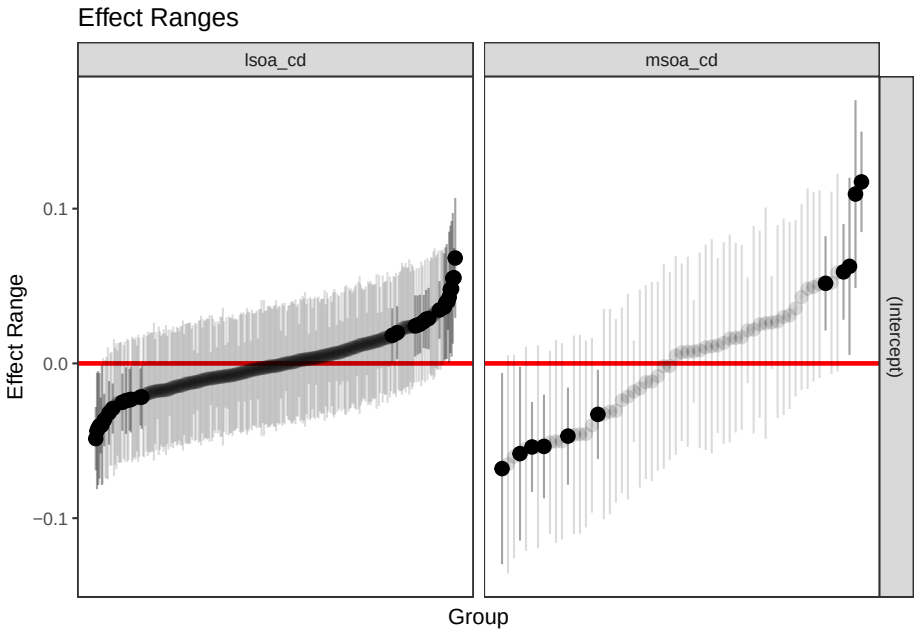
```
# obtain estimates
REsim(model3) %>% head(10)
```

```
##    groupFctr   groupID        term         mean       median         sd
## 1    lsoa_cd E01006512 (Intercept) -0.016457937 -0.017014712 0.01883432
## 2    lsoa_cd E01006513 (Intercept) -0.019391596 -0.018908412 0.02320422
## 3    lsoa_cd E01006514 (Intercept) -0.024226265 -0.023668911 0.02031985
## 4    lsoa_cd E01006515 (Intercept) -0.017634251 -0.017009473 0.02121148
## 5    lsoa_cd E01006518 (Intercept) -0.018827797 -0.019912609 0.02122950
## 6    lsoa_cd E01006519 (Intercept) -0.015741329 -0.015076692 0.01066756
## 7    lsoa_cd E01006520 (Intercept) -0.024220433 -0.024230257 0.02048487
## 8    lsoa_cd E01006521 (Intercept)  0.004949023  0.003082787 0.02140118
## 9    lsoa_cd E01006522 (Intercept)  0.019968183  0.019923845 0.02047809
## 10   lsoa_cd E01006523 (Intercept)  0.005570575  0.005581077 0.01992987
```

The results contain the estimated mean, median and standard deviation for the intercept within each group (e.g. LSOA). The mean estimates are similar to those obtained from `ranef` with some small differences due to rounding.

To gain an undertanding of the general pattern of the *random effects*, we can use caterpillar plots via `plotREsim` - reported below. The plot on the right shows the estimated random effects for each MSOA and their respective interval estimate. Note that random effects are on average zero, represented by the red horizontal line. Intervals that do not include zero are in bold. Also note that the width of the confidence interval depends on the standard error of the respective residual estimate, which is inversely related to the size of the sample. The residuals represent an observation departures from the grand mean, so an observation whose confidence interval does not overlap the line at zero (representing the mean proportion of unemployed population across all areas) is said to differ significantly from the average at the 5% level.

```
# plot
plotREsim(REsim(model3))
```



Focusing on the plot on the right, we see MSOAs whose mean proportion of unemployed population, assuming no explanatory variables, is lower than average. On the right-hand side of the plot, you will see MSOAs whose mean proportion is higher than average. The MSOAs with the smallest residuals include the districts of Allerton and Hunt Cross, Church, Childwall, Wavertree and Woolton. What districts do we have at the other extreme?

```
re <- REsim(model3)
oa_shp %>% dplyr::select(msoa_cd, ward_nm, unemp) %>%
    filter(as.character(msoa_cd) == "E02001387" | as.character(msoa_cd) == "E02001393")
```

```
## Simple feature collection with 49 features and 3 fields
## geometry type:  MULTIPOLYGON
## dimension:      XY
## bbox:           xmin: 339178.6 ymin: 386244.2 xmax: 341959.9 ymax: 389646.7
## projected CRS:  Transverse_Mercator
## First 10 features:
##      msoa_cd                  ward_nm      unemp                       geometry
## 1  E02001393 Allerton and Hunts Cross 0.03246753 MULTIPOLYGON (((341333.6 38...
## 2  E02001393 Allerton and Hunts Cross 0.03684211 MULTIPOLYGON (((340658.2 38...
## 3  E02001393                  Church 0.04098361 MULTIPOLYGON (((339908.1 38...
## 4  E02001393 Allerton and Hunts Cross 0.05982906 MULTIPOLYGON (((340306 3865...
## 5  E02001393                  Church 0.01212121 MULTIPOLYGON (((339974.2 38...
## 6  E02001393                  Church 0.09219858 MULTIPOLYGON (((340181.4 38...
## 7  E02001393                  Church 0.01986755 MULTIPOLYGON (((340301.2 38...
## 8  E02001393                  Church 0.04615385 MULTIPOLYGON (((340375.9 38...
## 9  E02001393 Allerton and Hunts Cross 0.04117647 MULTIPOLYGON (((340435.3 38...
## 10 E02001393 Allerton and Hunts Cross 0.02272727 MULTIPOLYGON (((340681.7 38...
```

We can also map the MSOA-level *random effects*. To this end, we first need to read a shapefile containing data at the MSOA level and merge it with the *random effects* estimates.

```
# read data
msoa_shp <- st_read("data/mlm/MSOA.shp")
```

```
## Reading layer `MSOA' from data source `/home/rstudio/Documents/data/mlm/MSOA.shp' us
## Simple feature collection with 61 features and 17 fields
## geometry type:  MULTIPOLYGON
## dimension:      XY
## bbox:           xmin: 333086.1 ymin: 381426.3 xmax: 345636 ymax: 397980.1
## projected CRS:  Transverse_Mercator
```

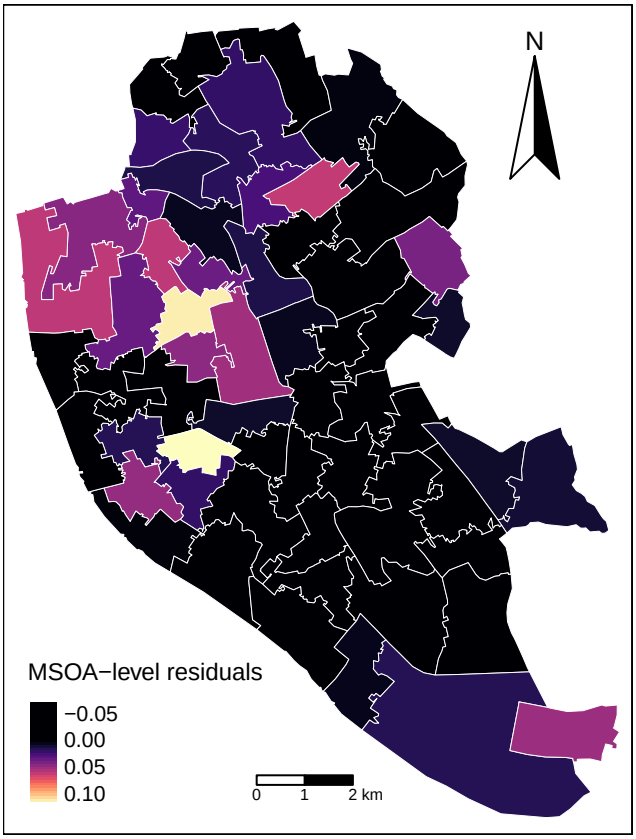
```
# create a dataframe for MSOA-level random effects
re_msoa <- re %>% filter(groupFctr == "msoa_cd")
str(re_msoa)
```

```
## 'data.frame':    61 obs. of  6 variables:
##  $ groupFctr: chr  "msoa_cd" "msoa_cd" "msoa_cd" "msoa_cd" ...
##  $ groupID  : chr  "E02001347" "E02001348" "E02001349" "E02001350" ...
##  $ term     : chr  "(Intercept)" "(Intercept)" "(Intercept)" "(Intercept)" ...
##  $ mean     : num  -0.00853 -0.02338 -0.02854 0.00473 0.02104 ...
##  $ median   : num  -0.00538 -0.02263 -0.02846 0.00577 0.0223 ...
##  $ sd       : num  0.0354 0.036 0.032 0.0313 0.0162 ...
```

```
# merge data
msoa_shp <- merge(x = msoa_shp, y = re_msoa, by.x = "MSOA_CD", by.y = "groupID")
```

Now we can create our map:

```
# ensure geometry is valid
msoa_shp = sf::st_make_valid(msoa_shp)

# create a map
legend_title = expression("MSOA-level residuals")
map_msoa = tm_shape(msoa_shp) +
  tm_fill(col = "mean", title = legend_title, palette = magma(256, begin = 0, end = 1), style = '
  tm_borders(col = "white", lwd = .01)  +
  tm_compass(type = "arrow", position = c("right", "top") , size = 4) +
  tm_scale_bar(breaks = c(0,1,2), text.size = 0.5, position =  c("center", "bottom"))
map_msoa
```



8.4.5 Adding Individual-level Predictors

In this example, μ represents the sample mean but it could include a collection of independent variables or predictors. To explain the logic, we will assume that unemployment is strongly associated to long-term illness. We could expect that long-term illness (`lt_ill`) will reduce the chances of working and therefore being unemployed. Note that our focus is on the relationship, not on establishing causation. Specifically we want to estimate the relationship between unemployment and long-term illness and we are interested in variations in OA-level unemployment by MSOAs so we will estimate the following two-level model:

OA-level:

$$y_{ij} = \beta_{0j} + \beta_1 x_{ij} + e_{ij}$$

MSOA-level:

$$\beta_{0j} = \beta_0 + u_{0j}$$

Replacing the first equation into the second, we have:

$$y_{ij} = (\beta_0 + u_{0j}) + \beta_1 x_{ij} + e_{ij}$$

where y the proportion of unemployed population in OA i within MSOA j; β_0 is the fixed intercept (averaging over all MSOAs); u_{0j} represents the MSOA-level residuals or *random effects*; β_0 and u_{0j} together represent the varying-intercept; β_1 is the slope coefficient; x_{ij} represents the percentage of long-term illness population; and, e_{ij} is the individual-level residuals.

We estimate the model executing:

```
# change to proportion
oa_shp$lt_ill <- lt_ill/100

# specify a model equation
eq4 <- unemp ~ lt_ill + (1 | msoa_cd)
model4 <- lmer(eq4, data = oa_shp)

# estimates
summary(model4)
```

```
## Linear mixed model fit by REML ['lmerMod']
## Formula: unemp ~ lt_ill + (1 | msoa_cd)
##    Data: oa_shp
##
## REML criterion at convergence: -4711.9
##
## Scaled residuals:
```

```
##     Min      1Q  Median      3Q     Max
## -5.1941 -0.5718 -0.0906  0.4507  5.9393
##
## Random effects:
##  Groups   Name        Variance Std.Dev.
##  msoa_cd  (Intercept) 0.001421 0.03769
##  Residual             0.002674 0.05171
## Number of obs: 1584, groups:  msoa_cd, 61
##
## Fixed effects:
##             Estimate Std. Error t value
## (Intercept)  0.04682    0.00625   7.492
## lt_ill       0.29588    0.01615  18.317
##
## Correlation of Fixed Effects:
##        (Intr)
## lt_ill -0.600
```

Fixed effects: model averaging over MSOAs

```
fixef(model4)
```

```
## (Intercept)      lt_ill
##  0.04681959  0.29588110
```

yields an estimated regression line in an average McSOA: $y = 0.04681959 + 0.29588110x$

Random effects: MSOA-level errors

```
ranef_m4 <- ranef(model4)
head(ranef_m4$msoa_cd, 5)
```

```
##            (Intercept)
## E02001347 -0.017474815
## E02001348 -0.021203807
## E02001349 -0.022469313
## E02001350 -0.003539869
## E02001351  0.008502813
```

yields an estimated intercept for MSOA `E02001347` which is `0.017474815` lower than the average with a regression line: `(0.04681959 - 0.017474815) + 0.29588110x = 0.02934478 + 0.29588110x`. You can confirm this by looking at the estimated model within each MSOA by executing (remove the `#` sign):

```
#coef(model4)
```

Fixed effect correlations

In the bottom of the output, we have the correlations between the fixed-effects

estimates. In our example, it refers to the correlation between β_0 and β_1. It is negative indicating that in MSOAs where the relationship between unemployment and long-term illness is greater, as measured by β_1, the average proportion of unemployed people tends to be smaller, as captured by β_0.

8.4.6 Adding Group-level Predictors

We can also add group-level predictors. We use the formulation:

OA-level:

$$y_{ij} = \beta_{0j} + \beta_1 x_{ij} + e_{ij}$$

MSOA-level:

$$\beta_{0j} = \beta_0 + \gamma_1 m_j + u_{0j}$$

where x_{ij} is the OA-level proportion of population suffering long-term illness and m_j is the MSOA-level proportion of male population. We first need to create this group-level predictor:

```
# detach OA shp and attach MSOA shp
detach(oa_shp)
attach(msoa_shp)

# group-level predictor
msoa_shp$pr_male <- males/pop

# remove geometries
msoa_df <- `st_geometry<-`(msoa_shp, NULL)

# select variables
msoa_df <- msoa_df %>% dplyr::select(MSOA_CD, pop, pr_male)

# merge data sets
oa_shp <- merge(x=oa_shp, y=msoa_df, by.x = "msoa_cd", by.y="MSOA_CD")

# inspect data
head(oa_shp[1:10, c("msoa_cd", "oa_cd", "unemp", "pr_male")])
```

```
## Simple feature collection with 6 features and 4 fields
## geometry type:  MULTIPOLYGON
## dimension:      XY
## bbox:           xmin: 337693.5 ymin: 396068.2 xmax: 339430.9 ymax: 397790
## projected CRS:  Transverse_Mercator
##     msoa_cd     oa_cd      unemp   pr_male                       geometry
```

```
## 1 E02001347 E00033730 0.10322581 0.4775905 MULTIPOLYGON (((338376 3970...
## 2 E02001347 E00033722 0.06306306 0.4775905 MULTIPOLYGON (((337929.4 39...
## 3 E02001347 E00033712 0.09090909 0.4775905 MULTIPOLYGON (((338830 3960...
## 4 E02001347 E00033739 0.09401709 0.4775905 MULTIPOLYGON (((339140.3 39...
## 5 E02001347 E00033719 0.05855856 0.4775905 MULTIPOLYGON (((338128.8 39...
## 6 E02001347 E00033711 0.12195122 0.4775905 MULTIPOLYGON (((339163.2 39...
```

We can now estimate our model:

```
detach(msoa_shp)
attach(oa_shp)

# specify a model equation
eq5 <- unemp ~ lt_ill + pr_male + (1 | msoa_cd)
model5 <- lmer(eq5, data = oa_shp)

# estimates
summary(model5)
```

```
## Linear mixed model fit by REML ['lmerMod']
## Formula: unemp ~ lt_ill + pr_male + (1 | msoa_cd)
##    Data: oa_shp
##
## REML criterion at convergence: -4712.3
##
## Scaled residuals:
##     Min      1Q  Median      3Q     Max
## -5.2162 -0.5696 -0.0929  0.4549  5.9370
##
## Random effects:
##  Groups   Name        Variance Std.Dev.
##  msoa_cd  (Intercept) 0.001391 0.03729
##  Residual             0.002674 0.05171
## Number of obs: 1584, groups:  msoa_cd, 61
##
## Fixed effects:
##             Estimate Std. Error t value
## (Intercept) -0.07746    0.08768  -0.883
## lt_ill       0.29781    0.01620  18.389
## pr_male      0.25059    0.17642   1.420
##
## Correlation of Fixed Effects:
##         (Intr) lt_ill
## lt_ill  -0.118
## pr_male -0.997  0.075
```

This model includes the proportion of males and intercepts that vary by MSOA.

The `lmer()` function only accepts predictors at the individual level, so we have included data on the proportion of male population at this level. Explore and interpret the model running the functions below:

```
# fixed effects
fixef(model5)
```

```
## (Intercept)      lt_ill     pr_male
##  -0.0774607   0.2978084   0.2505913
```

```
# random effects
ranef_m5 <- ranef(model5)
head(ranef_m5$msoa_cd, 5)
```

```
##             (Intercept)
## E02001347 -0.013625261
## E02001348 -0.019757846
## E02001349 -0.023709992
## E02001350  0.003003861
## E02001351  0.003508477
```

Adding group-level predictors tends to improve inferences for group coefficients. Examine the confidence intervals, in order to evalute how the precision of our estimates of the MSOA intercepts have changed. *Have confidence intervals for the intercepts of Model 4 and 5 increased or reduced?* Hint: look at how to get the confidence intervals above.

8.5 Useful Functions

Function	Description
lmer()	fit linear mixed-effects models
fixef()	obtain estimated fixed effects or model averaging over groups
ranef()	obtain estimated random effects or group-level residuals
REsim()	obtain estimated random effects or group-level residuals based on simulation
plotREsim()	create a caterpillar plot of estimated random effects
coef()	obtain coefficients within each group
anova()	provide regression model diagnostics

Chapter 9

Multilevel Modelling - Part 2

This chapter[1] provides an introduction to multi-level data structures and multi-level modelling.

The content of this chapter is based on:

- Gelman and Hill (2006b) provides an excellent and intuitive explanation of multilevel modelling and data analysis in general. Read Part 2A for a really good explanation of multilevel models.
- for Multilevel Modelling (nd) is an useful online resource on multilevel modelling and is free!

This Chapter is part of Spatial Analysis Notes, a compilation hosted as a GitHub repository that you can access it in a few ways:

- As a download of a `.zip` file that contains all the materials.
- As an html website.
- As a pdf document
- As a GitHub repository.

9.1 Dependencies

This chapter uses the following libraries: Ensure they are installed on your machine[2] before loading them executing the following code chunk:

[1] This note is part of Spatial Analysis Notes Multilevel Modelling – Random Slope Model by Francisco Rowe is licensed under a Creative Commons Attribution-NonCommercial-ShareAlike 4.0 International License.

[2] You can install package `mypackage` by running the command `install.packages("mypackage")` on the R prompt or through the `Tools --> Install`

```
# Data manipulation, transformation and visualisation
library(tidyverse)
# Nice tables
library(kableExtra)
# Simple features (a standardised way to encode vector data ie. points, lines, polygon
library(sf)
# Spatial objects conversion
library(sp)
# Thematic maps
library(tmap)
# Colour palettes
library(RColorBrewer)
# More colour palettes
library(viridis) # nice colour schemes
# Fitting multilevel models
library(lme4)
# Tools for extracting information generated by lme4
library(merTools)
# Exportable regression tables
library(jtools)
library(stargazer)
library(sjPlot)
```

9.2 Data

For this chapter, we will data for Liverpool from England's 2011 Census. The original source is the Office of National Statistics and the dataset comprises a number of selected variables capturing demographic, health and socio-economic of the local resident population at four geographic levels: Output Area (OA), Lower Super Output Area (LSOA), Middle Super Output Area (MSOA) and Local Authority District (LAD). The variables include population counts and percentages. For a description of the variables, see the readme file in the mlm data folder.[3]

Let us read the data:

```
# clean workspace
rm(list=ls())
# read data
oa_shp <- st_read("data/mlm/OA.shp")
```

`Packages...` menu in RStudio.

[3] Read the file in R by executing `read_tsv("data/mlm/readme.txt")`

9.3 Conceptual Overview

So far, we have estimated varying-intercept models; that is, when the intercept (β_0) is allowed to vary by group (eg. geographical area) - as shown in Fig. 1(a). The strength of the relationship between y (i.e. unemployment rate) and x (long-term illness) has been assumed to be the same across groups (i.e. MSOAs), as captured by the regression slope (β_1). Yet it can also vary by group as shown in Fig. 1(b), or we can observe group variability for both intercepts and slopes as represented in Fig. 1(c).

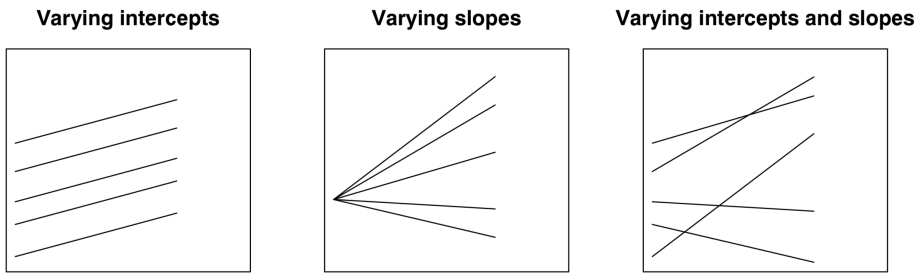


Figure 9.1: Fig. 1. Linear regression model with (a) varying intercepts, (b) varying slopes, and (c) both. Source: Gelman and Hill (2006b) p.238.

9.3.1 Exploratory Analysis: Varying Slopes

Let's then explore if there is variation in the relationship between unemployment rate and the share of population in long-term illness. We do this by selecting the 8 MSOAs containing OAs with the highest unemployment rates in Liverpool.

```
# Sort data
oa_shp <- oa_shp %>% arrange(-unemp)
oa_shp[1:9, c("msoa_cd", "unemp")]
```

```
## Simple feature collection with 9 features and 2 fields
## geometry type:  MULTIPOLYGON
## dimension:      XY
## bbox:           xmin: 335032 ymin: 387777 xmax: 338576.1 ymax: 395022.4
## projected CRS:  Transverse_Mercator
##     msoa_cd     unemp                       geometry
## 1 E02001354 0.5000000 MULTIPOLYGON (((337491.2 39...
## 2 E02001369 0.4960630 MULTIPOLYGON (((335272.3 39...
## 3 E02001366 0.4461538 MULTIPOLYGON (((338198.1 39...
## 4 E02001365 0.4352941 MULTIPOLYGON (((336572.2 39...
## 5 E02001370 0.4024390 MULTIPOLYGON (((336328.3 39...
## 6 E02001390 0.3801653 MULTIPOLYGON (((335833.6 38...
## 7 E02001354 0.3750000 MULTIPOLYGON (((337403 3949...
```

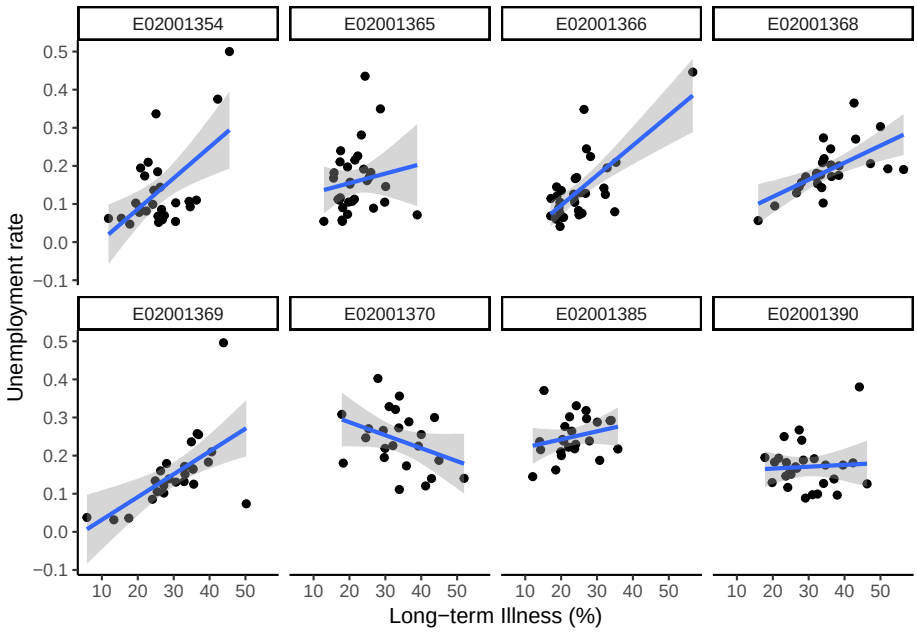
```
## 8 E02001385 0.3707865 MULTIPOLYGON (((336251.6 38...
## 9 E02001368 0.3648649 MULTIPOLYGON (((335209.3 39...
```

```
# Select MSOAs
s_t8 <- oa_shp %>% dplyr::filter(
    as.character(msoa_cd) %in% c(
      "E02001354",
      "E02001369",
      "E02001366",
      "E02001365",
      "E02001370",
      "E02001390",
      "E02001368",
      "E02001385")
    )
```

And then we generate a set of scatter plots and draw regression lines for each MSOA.

```
ggplot(s_t8, aes(x = lt_ill, y = unemp)) +
  geom_point() +
  geom_smooth(method = "lm") +
  facet_wrap(~ msoa_cd, nrow = 2) +
  ylab("Unemployment rate") +
  xlab("Long-term Illness (%)") +
  theme_classic()
```

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



We can observe great variability in the relationship between unemployment rates and the percentage of population in long-term illness. A strong and positive relationship exists in MSOA `E02001366` (Tuebrook and Stoneycroft), while it is negative in MSOA `E02001370` (Everton) and neutral in MSOA `E02001390` (Princes Park & Riverside). This visual inspection suggests that accounting for differences in the way unmployment rates relate to long-term illness is important. Contextual factors may differ across MSOAs in systematic ways.

9.4 Estimating Varying Intercept and Slopes Models

A way to capture for these group differences in the relationship between unemployment rates and long-term illness is to allow the relevant slope to vary by group (i.e. MSOA). We can do this estimating the following model:

OA-level:

$$y_{ij} = \beta_{0j} + \beta_{1j}x_{ij} + e_{ij}$$

MSOA-level:

$$\beta_{0j} = \beta_0 + u_{0j}$$
$$\beta_{1j} = \beta_1 + u_{1j}$$

Replacing the first equation into the second generates:

$$y_{ij} = (\beta_0 + u_{0j}) + (\beta_1 + u_{1j})x_{ij} + e_{ij}$$

where, as in the previous Chapter, y the proportion of unemployed population in OA i within MSOA j; β_0 is the fixed intercept (averaging over all MSOAs); u_{0j} represents the MSOA-level residuals, or *random effects*, of the intercept; e_{ij} is the individual-level residuals; and, x_{ij} represents the percentage of long-term illness population. *But* now we have a varying slope represented by β_1 and u_{1j}: β_1 is estimated average slope - fixed part of the model; and, u_{1j} is the estimated group-level errors of the slope.

To estimate such model, we add `lt_ill` in the bracket with a `+` sign between `1` and `|` i.e. `(1 + lt_ill | msoa_cd)`.

```
# attach df
attach(oa_shp)

# change to proportion
oa_shp$lt_ill <- lt_ill/100

# specify a model equation
eq6 <- unemp ~ lt_ill + (1 + lt_ill | msoa_cd)
model6 <- lmer(eq6, data = oa_shp)

# estimates
summary(model6)
```

```
## Linear mixed model fit by REML ['lmerMod']
## Formula: unemp ~ lt_ill + (1 + lt_ill | msoa_cd)
##    Data: oa_shp
##
## REML criterion at convergence: -4762.8
##
## Scaled residuals:
##     Min      1Q  Median      3Q     Max
## -3.6639 -0.5744 -0.0873  0.4565  5.4876
##
## Random effects:
##  Groups   Name        Variance Std.Dev. Corr
##  msoa_cd  (Intercept) 0.003428 0.05855
##           lt_ill      0.029425 0.17154  -0.73
##  Residual             0.002474 0.04974
## Number of obs: 1584, groups:  msoa_cd, 61
##
## Fixed effects:
##             Estimate Std. Error t value
```

```
## (Intercept) 0.047650   0.008635   5.519
## lt_ill      0.301259   0.028162  10.697
##
## Correlation of Fixed Effects:
##        (Intr)
## lt_ill -0.786
```

In this model, the estimated standard deviation of the unexplained within-MSOA variation is 0.04974, and the estimated standard deviation of the MSOA intercepts is 0.05855. But, additionally, we also have estimates of standard deviation of the MSOA slopes (0.17154) and correlation between MSOA-level residuals for the intercept and slope (-0.73). While the former measures the extent of average deviation in the slopes across MSOAs, the latter indicates that the intercept and slope MSOA-level residuals are negatively associated; that is, MSOAs with large slopes have relatively smaller intercepts and *vice versa*. We will come back to this in Section Interpreting Correlations Between Group-level Intercepts and Slopes.

Similarly, the correlation of fixed effects indicates a negative relationship between the intercept and slope of the average regression model; that is, as the average model intercept tends to increase, the average strength of the relationship between unemployment rate and long-term illness decreases and *vice versa*.

We then explore the estimated average coefficients (*fixed effects*):

```
fixef(model6)
```

```
## (Intercept)      lt_ill
##  0.04765009  0.30125875
```

yields an estimated regression line in an average LSOA: $y = 0.04764998 + 0.30125916x$. The fixed intercept indicates that the average unemployment rate is 0.05 if the percentage of population with long-term illness is zero.The fixed slope indicates that the average relationship between unemployment rate and long-term illness is positive across MSOAs i.e. as the percentage of population with long-term illness increases by 1 percentage point, the unemployment rate increases by 0.3.

We look the estimated MSOA-level errors (*random effects*):

```
ranef_m6 <- ranef(model6)
head(ranef_m6$msoa_cd, 5)
```

```
##            (Intercept)      lt_ill
## E02001347 -0.026561345  0.02718102
## E02001348  0.001688245 -0.11533102
## E02001349 -0.036084817  0.05547075
## E02001350  0.032240842 -0.14298734
## E02001351  0.086214137 -0.28130162
```

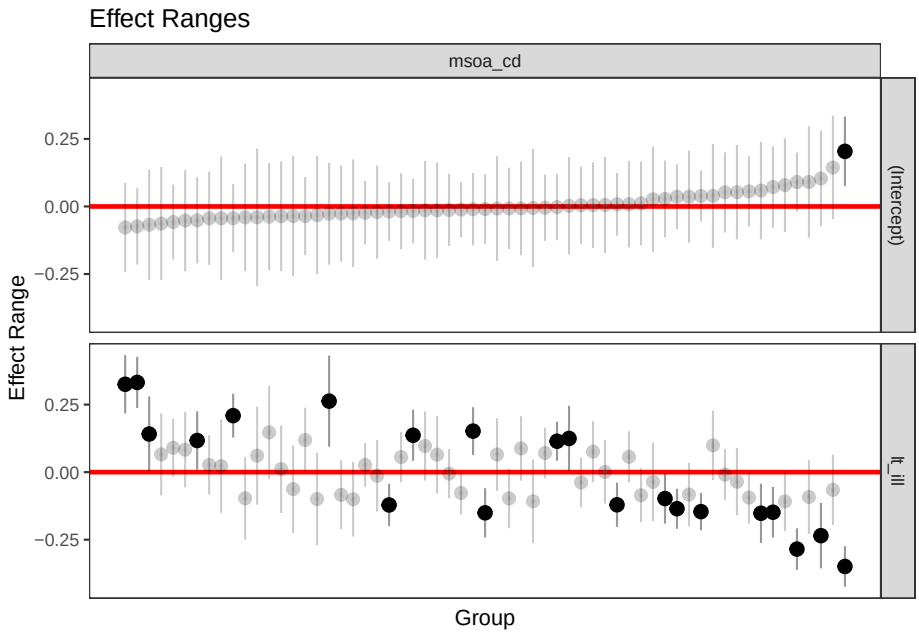
Recall these estimates indicate the extent of deviation of the MSOA-specific intercept and slope from the estimated model average captured by the fixed model component.

We can also regain the estimated intercept and slope for each county by adding the estimated MSOA-level errors to the estimated average coefficients; or by executing:

```
#coef(model6)
```

We are normally more interested in identifying the extent of deviation and its significance. To this end, we create a caterpillar plot:

```
# plot
plotREsim(REsim(model6))
```



These plots reveal some interesting patterns. First, only one MSOA, containing wards such as Tuebrook and Stoneycroft, Anfield & Everton, seems to have a statistically significantly different intercept, or average unemployment rate. Confidence intervals overlap zero for all other 60 MSOAs. Despite this, note that when a slope is allowed to vary by group, it generally makes sense for the intercept to also vary. Second, significant variability exists in the association between unemployment rate and long-term illness across MSOAs. Ten MSOAs display a significant positive association, while 12 exhibit a significantly negative relationship. Third, these results reveal that geographical differences in the relationship between unemployment rate and long-term illness can explain the significant differences in average unemployment rates in the varying intercept

only model.

Let's try to get a better understanding of the varying relationship between unemployment rate and long-term illness by mapping the relevant MSOA-level errors.

```
# read data
msoa_shp <- st_read("data/mlm/MSOA.shp")
```

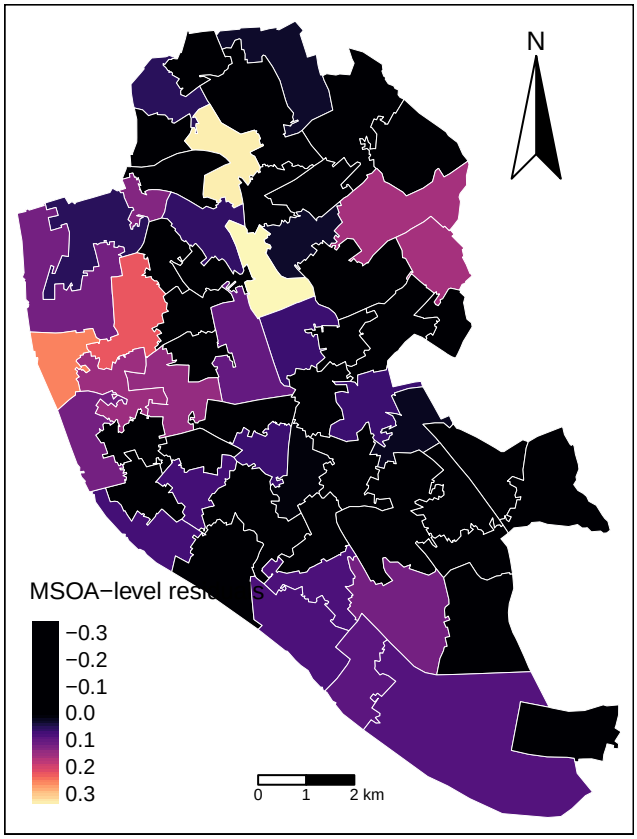
```
## Reading layer `MSOA' from data source `/home/rstudio/Documents/data/mlm/MSOA.shp' using driver
## Simple feature collection with 61 features and 17 fields
## geometry type:  MULTIPOLYGON
## dimension:      XY
## bbox:           xmin: 333086.1 ymin: 381426.3 xmax: 345636 ymax: 397980.1
## projected CRS:  Transverse_Mercator
```

```
# create a dataframe for MSOA-level random effects
re_msoa_m6 <- REsim(model6) %>% filter(groupFctr == "msoa_cd") %>%
  filter(term == "lt_ill")
str(re_msoa_m6)
```

```
## 'data.frame':    61 obs. of  6 variables:
##  $ groupFctr: chr  "msoa_cd" "msoa_cd" "msoa_cd" "msoa_cd" ...
##  $ groupID  : chr  "E02001347" "E02001348" "E02001349" "E02001350" ...
##  $ term     : chr  "lt_ill" "lt_ill" "lt_ill" "lt_ill" ...
##  $ mean     : num  0.0288 -0.1169 0.0613 -0.1427 -0.2851 ...
##  $ median   : num  0.0282 -0.118 0.0577 -0.1425 -0.285 ...
##  $ sd       : num  0.0469 0.0738 0.0883 0.0372 0.0395 ...
```

```
# merge data
msoa_shp <- merge(x = msoa_shp, y = re_msoa_m6, by.x = "MSOA_CD", by.y = "groupID")
```

```
# ensure geometry is valid
msoa_shp = sf::st_make_valid(msoa_shp)

# create a map
legend_title = expression("MSOA-level residuals")
map_msoa = tm_shape(msoa_shp) +
  tm_fill(col = "median", title = legend_title, palette = magma(256, begin = 0, end = 1), style =
  tm_borders(col = "white", lwd = .01)  +
  tm_compass(type = "arrow", position = c("right", "top") , size = 4) +
  tm_scale_bar(breaks = c(0,1,2), text.size = 0.5, position =  c("center", "bottom"))
map_msoa
```



The map indicates that the relationship between unemployment rate and long-term illness is tends to stronger and positive in northern MSOAs; that is, the percentage of population with long-term illness explains a greater share of the variation in unemployment rates in these locations. As expected, a greater share of population in long-term illness is associated with higher local unemployment. In contrast, the relationship between unemployment rate and long-term illness tends to operate in the reverse direction in north-east and middle-southern MSOAs. In these MSOAs, OAs tend to have a higher unemployment rate relative the share of population in long-term illness. You can confirm this examining the data for specific MSOA executing:

```
oa_shp %>% dplyr::select(msoa_cd, ward_nm, unemp, lt_ill) %>%
    filter(as.character(msoa_cd) == "E02001370")
```

```
## Simple feature collection with 23 features and 4 fields
## geometry type:  MULTIPOLYGON
## dimension:      XY
## bbox:           xmin: 335885 ymin: 391134.2 xmax: 337596.3 ymax: 392467
## projected CRS:  Transverse_Mercator
## First 10 features:
```

```
##      msoa_cd                  ward_nm     unemp    lt_ill
## 1  E02001370                  Everton 0.4024390 0.2792793
## 2  E02001370 Tuebrook and Stoneycroft 0.3561644 0.3391813
## 3  E02001370                  Everton 0.3285714 0.3106383
## 4  E02001370                  Everton 0.3209877 0.3283019
## 5  E02001370                  Anfield 0.3082707 0.1785714
## 6  E02001370                  Everton 0.3000000 0.4369501
## 7  E02001370                  Everton 0.2886598 0.3657143
## 8  E02001370                  Everton 0.2727273 0.3375000
## 9  E02001370                  Everton 0.2705882 0.2534247
## 10 E02001370 Tuebrook and Stoneycroft 0.2661290 0.2941176
##                            geometry
## 1  MULTIPOLYGON (((336328.3 39...
## 2  MULTIPOLYGON (((337481.5 39...
## 3  MULTIPOLYGON (((336018.5 39...
## 4  MULTIPOLYGON (((336475.7 39...
## 5  MULTIPOLYGON (((337110.6 39...
## 6  MULTIPOLYGON (((336516.3 39...
## 7  MULTIPOLYGON (((336668.6 39...
## 8  MULTIPOLYGON (((336173.8 39...
## 9  MULTIPOLYGON (((336870 3917...
## 10 MULTIPOLYGON (((337363.8 39...
```

Now try adding a group-level predictor and an individual-level predictor to the model. Unsure, look at the Sections Adding Group-level Predictors and Adding Individual-level Predictors in the previous Chapter.

9.5 Interpreting Correlations Between Group-level Intercepts and Slopes

Correlations of random effects are confusing to interpret. Key for their appropriate interpretation is to recall they refer to group-level residuals i.e. deviation of intercepts and slopes from the average model intercept and slope. A strong *negative* correlation indicates that groups with high intercepts have relatively low slopes, and *vice versa*. A strong *positive* correlation indicates that groups with high intercepts have relatively high slopes, and *vice versa*. A correlation close to *zero* indicate little or no systematic between intercepts and slopes. Note that a high correlation between intercepts and slopes is not a problem, but it makes the interpretation of the estimated intercepts more challenging. For this reason, a suggestion is to center predictors ($x's$); that is, substract their average value ($z = x - \bar{x}$). For a more detailed discussion, see for Multilevel Modelling (nd).

To illustrate this, let's reestimate our model adding an individual-level predictor: the share of population with no educational qualification.

```r
# centering to the mean
oa_shp$z_no_qual <- no_qual/100 - mean(no_qual/100)
oa_shp$z_lt_ill <- lt_ill - mean(lt_ill)

# specify a model equation
eq7 <- unemp ~ z_lt_ill + z_no_qual + (1 + z_lt_ill | msoa_cd)
model7 <- lmer(eq7, data = oa_shp)

# estimates
summary(model7)
```

```
## Linear mixed model fit by REML ['lmerMod']
## Formula: unemp ~ z_lt_ill + z_no_qual + (1 + z_lt_ill | msoa_cd)
##    Data: oa_shp
##
## REML criterion at convergence: -4940.7
##
## Scaled residuals:
##     Min      1Q  Median      3Q     Max
## -3.6830 -0.5949 -0.0868  0.4631  6.3556
##
## Random effects:
##  Groups   Name        Variance  Std.Dev. Corr
##  msoa_cd  (Intercept) 8.200e-04 0.02864
##           z_lt_ill    2.161e-06 0.00147  -0.04
##  Residual             2.246e-03 0.04739
## Number of obs: 1584, groups:  msoa_cd, 61
##
## Fixed effects:
##               Estimate Std. Error t value
## (Intercept)  0.1163682  0.0039201   29.68
## z_lt_ill    -0.0003130  0.0003404   -0.92
## z_no_qual    0.3245811  0.0221347   14.66
##
## Correlation of Fixed Effects:
##           (Intr) z_lt_l
## z_lt_ill  -0.007
## z_no_qual -0.015 -0.679
```

How do you interpret the random effect correlation?

9.6 Model building

Now we know how to estimate multilevel regression models in *R*. The question that remains is: *When does multilevel modeling make a difference?* The

short answer is: when there is little group-level variation. When there is very little group-level variation, the multilevel modelling reduces to classical linear regression estimates *with no group indicators*. Inversely, when group-level coefficients vary greatly (compared to their standard errors of estimation), multilevel modelling reduces to classical regression *with group indicators* Gelman and Hill (2006b).

How do you go about building a model?

We generally start simple by fitting simple linear regressions and then work our way up to a full multilevel model - see Gelman and Hill (2006b) p. 270.

How many groups are needed?

As an absolute minimum, more than two groups are required. With only one or two groups, a multilevel model reduces to a linear regression model.

How many observations per group?

Two observations per group is sufficient to fit a multilevel model.

9.6.1 Model Comparison

How we assess different candidate models? We can use the function `anova()` and assess various statistics: The Akaike Information Criterion (AIC), the Bayesian Information Criterion (BIC), Loglik and Deviance. Generally, we look for lower scores for all these indicators. We can also refer to the *Chisq* statistic below. It tests the hypothesis of whether additional predictors improve model fit. Particularly it tests the *Null Hypothesis* whether the coefficients of the additional predictors equal 0. It does so comparing the deviance statistic and determining if changes in the deviance are statistically significant. Note that a major limitation of the deviance test is that it is for nested models i.e. a model being compared must be nested in the other. Below we compare our two models. The results indicate that adding an individual-level predictor (i.e. the share of population with no qualification) provides a model with better.

```
anova(model6, model7)
```

```
## refitting model(s) with ML (instead of REML)
```

```
## Data: oa_shp
## Models:
## model6: unemp ~ lt_ill + (1 + lt_ill | msoa_cd)
## model7: unemp ~ z_lt_ill + z_no_qual + (1 + z_lt_ill | msoa_cd)
##        npar     AIC     BIC logLik deviance  Chisq Df Pr(>Chisq)
## model6    6 -4764.7 -4732.5 2388.3  -4776.7
## model7    7 -4956.5 -4918.9 2485.2  -4970.5 193.76  1  < 2.2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Chapter 10

Geographically Weighted Regression

This chapter[1] provides an introduction to geographically weighted regression models.

The content of this chapter is based on:

- Fotheringham et al. (2002), a must-go book if you are working or planning to start working on geographically weighted regression modelling.
- Comber et al. (2020)'s recently published preprint which provides a roadmap to approach various practical issues in the application of GWR.

This Chapter is part of Spatial Analysis Notes, a compilation hosted as a GitHub repository that you can access in a few ways:

- As a download of a `.zip` file that contains all the materials.
- As an html website.
- As a pdf document
- As a GitHub repository.

10.1 Dependencies

This chapter uses the following libraries. Ensure they are installed on your machine[2] before loading them by executing the following code chunk:

[1] This note is part of Spatial Analysis Notes Geographically Weighted Regression – Spatial Nonstationarity by Francisco Rowe is licensed under a Creative Commons Attribution-NonCommercial-ShareAlike 4.0 International License.

[2] You can install package `mypackage` by running the command `install.packages("mypackage")` on the R prompt or through the `Tools --> Install Packages...` menu in RStudio.

```
# Data manipulation, transformation and visualisation
library(tidyverse)
# Nice tables
library(kableExtra)
# Simple features (a standardised way to encode vector data ie. points, lines, polygon
library(sf)
# Spatial objects conversion
library(sp)
# Thematic maps
library(tmap)
# Colour palettes
library(RColorBrewer)
# More colour palettes
library(viridis) # nice colour schemes
# Fitting geographically weighted regression models
library(spgwr)
# Obtain correlation coefficients
library(corrplot)
# Exportable regression tables
library(jtools)
library(stargazer)
library(sjPlot)
# Assess multicollinearity
library(car)
```

10.2 Data

For this chapter, we will use data on:

- cumulative COVID-19 confirmed cases from 1st January, 2020 to 14th April, 2020 from Public Health England via the GOV.UK dashboard;
- resident population characteristics from the 2011 census, available from the Office of National Statistics; and,
- 2019 Index of Multiple Deprivation (IMD) data from GOV.UK and published by the Ministry of Housing, Communities & Local Government.

The data used for this Chapter are organised at the ONS Upper Tier Local Authority (UTLA) level - also known as Counties and Unitary Authorities. They are the geographical units used to report COVID-19 data.

If you use the dataset utilised in this chapter, make sure cite this book. For a full list of the variables included in the data set used in this Chapter, see the readme file in the gwr data folder.[3]

[3] Read the file in R by executing `read_tsv("data/gwr/readme.txt")`

Let's read the data:

```
# clean workspace
rm(list=ls())
# read data
utla_shp <- st_read("data/gwr/Covid19_total_cases_geo.shp") %>%
  select(objct, cty19c, ctyu19nm, long, lat, st_rs, st_ln, X2020.04.14, I.PL1, IMD20, IMD2., Rsdr

# replace nas with 0s
utla_shp[is.na(utla_shp)] <- 0
# explore data
str(utla_shp)
```

10.3 Recap: Spatial Effects

To this point, we have implicitly discussed three distinctive spatial effects:

- *Spatial heterogeneity* refers to the uneven distribution of a variable's values across space
- *Spatial dependence* refers to the spatial relationship of a variable's values for a pair of locations at a certain distance apart, so that they are more similar (or less similar) than expected for randomly associated pairs of observations
- *Spatial nonstationarity* refers to variations in the relationship between an outcome variable and a set of predictor variables across space

In previous sessions, we considered multilevel models to deal with spatial nonstationarity, recognising that the strength and direction of the relationship between an outcome y and a set of predictors x may vary over space. Here we consider a different approach, namely geographically weighted regression (GWR).

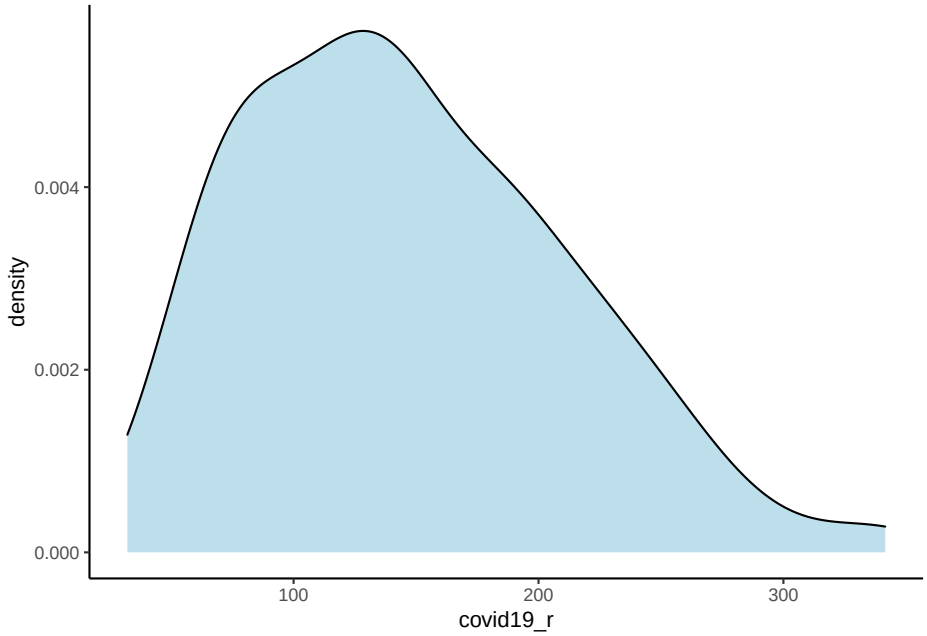
10.4 Exploratory Analysis

We will explore this technique through an empirical analysis considering the current global COVID-19 outbreak. Specifically we will seek to identify potential contextual factors that may be related to an increased risk of local infection. Population density, overcrowded housing, vulnerable individuals and critical workers have all been linked to a higher risk of COVID-19 infection.

First, we will define and develop some basic understanding of our variable of interest. We define the risk of COVID-19 infection by the cumulative number of confirmed positive cases COVID-19 per 100,000 people:

```
# risk of covid-19 infection
utla_shp$covid19_r <- (utla_shp$X2020.04.14 / utla_shp$Rsdnt) * 100000
```

```
# histogram
ggplot(data = utla_shp) +
geom_density(alpha=0.8, colour="black", fill="lightblue", aes(x = covid19_r)) +
   theme_classic()
```




```
# distribution in numbers
summary(utla_shp$covid19_r)
```

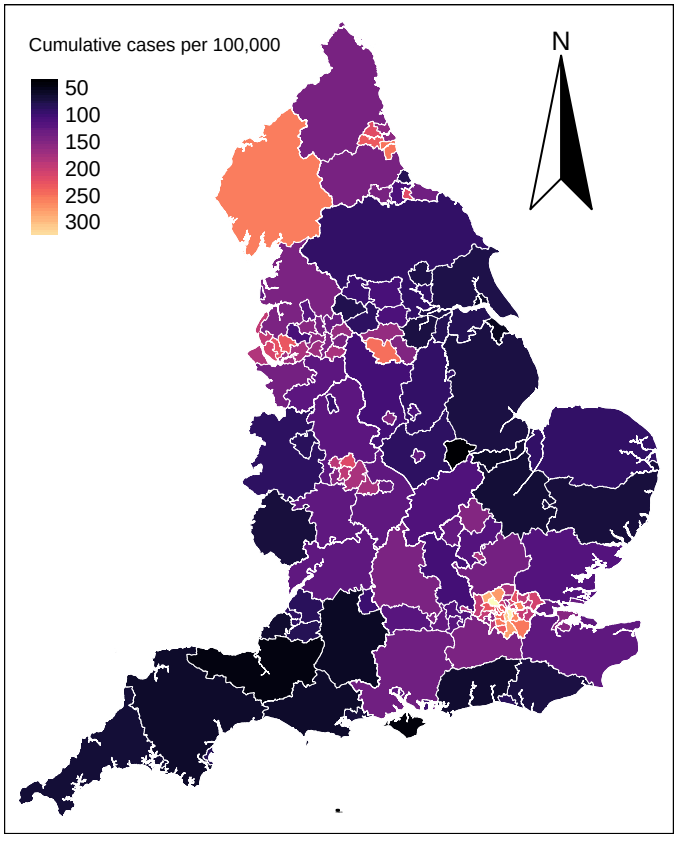
```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   32.11   92.81  140.15  146.66  190.96  341.56
```

The results indicate a wide variation in the risk of infection across UTLAs in England, ranging from 31 to 342 confirmed positive cases of COVID-19 per 100,000 people with a median of 147. We map the cases to understand their spatial structure.

```
# read region boundaries for a better looking map
reg_shp <- st_read("data/gwr/Regions_December_2019_Boundaries_EN_BGC.shp")
```

```
## Reading layer `Regions_December_2019_Boundaries_EN_BGC' from data source `/home/rstu
## Simple feature collection with 9 features and 9 fields
## geometry type:  MULTIPOLYGON
## dimension:      XY
## bbox:           xmin: 82672 ymin: 5342.7 xmax: 655653.8 ymax: 657536
## projected CRS:  OSGB 1936 / British National Grid
```

```
# ensure geometry is valid
utla_shp = sf::st_make_valid(utla_shp)
reg_shp = sf::st_make_valid(reg_shp)

# map
legend_title = expression("Cumulative cases per 100,000")
map_utla = tm_shape(utla_shp) +
  tm_fill(col = "covid19_r", title = legend_title, palette = magma(256), style = "cont") + # add
  tm_borders(col = "white", lwd = .1)  + # add borders
  tm_compass(type = "arrow", position = c("right", "top") , size = 5) + # add compass
  tm_scale_bar(breaks = c(0,1,2), text.size = 0.7, position =  c("center", "bottom")) + # add sc
  tm_layout(bg.color = "white") # change background colour
map_utla + tm_shape(reg_shp) + # add region boundaries
  tm_borders(col = "white", lwd = .5) # add borders
```



The map shows that concentrations of high incidence of infections in the metropolitan areas of London, Liverpool, Newcastle, Sheffield, Middlesbrough and Birmingham. Below we list the UTLAs in these areas in descending order.

```
hotspots <- utla_shp %>% select(ctyu19nm, covid19_r) %>%
  filter(covid19_r > 190)
hotspots[order(-hotspots$covid19_r),]
```

```
## Simple feature collection with 38 features and 2 fields
## geometry type:  MULTIPOLYGON
## dimension:      XY
## bbox:           xmin: 293941.4 ymin: 155850.8 xmax: 561956.7 ymax: 588517.4
## projected CRS:  Transverse_Mercator
## First 10 features:
##     ctyu19nm covid19_r                       geometry
## 14     Brent  341.5645 MULTIPOLYGON (((520113.1 19...
## 32 Southwark  337.1687 MULTIPOLYGON (((532223 1805...
## 27   Lambeth  305.5238 MULTIPOLYGON (((531189.5 18...
## 22    Harrow  286.1254 MULTIPOLYGON (((517363.8 19...
## 17   Croydon  276.8467 MULTIPOLYGON (((531549.3 17...
## 12    Barnet  274.1410 MULTIPOLYGON (((524645.2 19...
## 28  Lewisham  261.3408 MULTIPOLYGON (((536691.6 17...
## 30    Newham  258.4550 MULTIPOLYGON (((542600.7 18...
## 38   Cumbria  255.6726 MULTIPOLYGON (((357012.9 58...
## 15   Bromley  253.7234 MULTIPOLYGON (((542252.7 17...
```

Challenge 1: How does Liverpool ranked in this list?

10.5 Global Regression

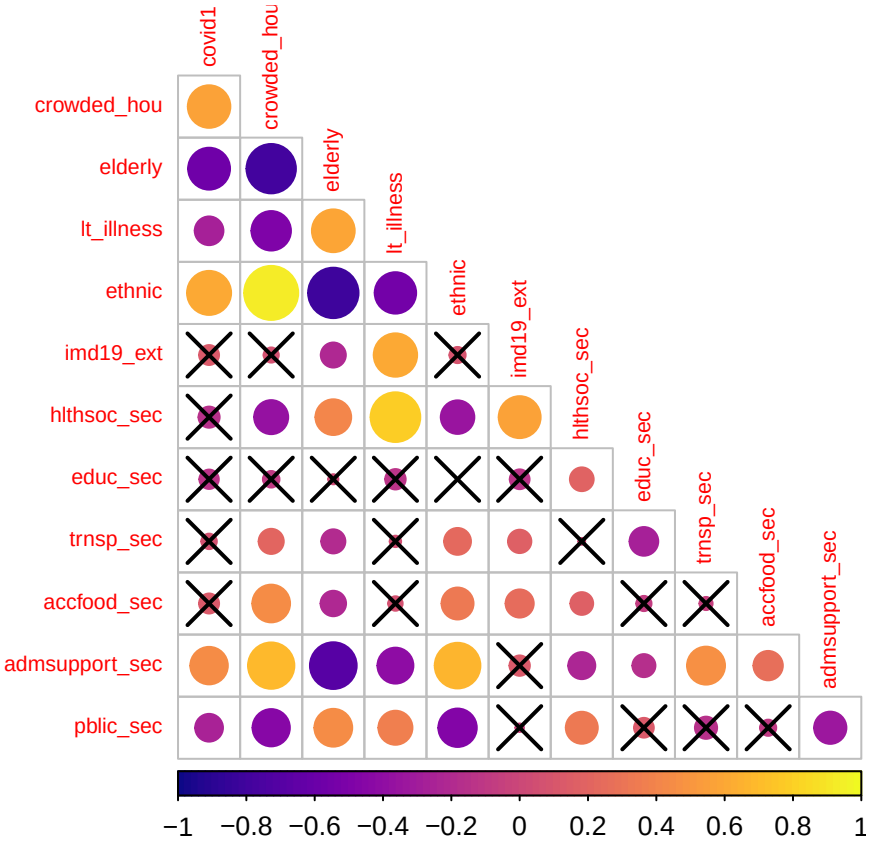
To provide an intuitive understanding of GWR, a useful start is to explore the data using an ordinary least squares (OLS) linear regression model. The key issue here is to understand if high incidence of COVID-19 is linked to structural differences across UTLAs in England. As indicated above, confirmed positive cases of COVID-19 have been associated with overcrowded housing, vulnerable populations - including people in elderly age groups, economically disadvantaged groups and those suffering from chronic health conditions - ethnic minorities, critical workers in the health & social work, education, accommodation & food, transport, and administrative & support sectors. So, let's create a set of variables to approximate these factors.

```
# define predictors
utla_shp <- utla_shp %>% mutate(
  crowded_hou = Crwd_ / Hshld, # share of crowded housing
  elderly = (A_65_ + Ag_85) / Rsdnt, # share of population aged 65+
  lt_illness = Lng__ / Rsdnt, # share of population in long-term illness
  ethnic = (Mixed + Indin + Pkstn + Bngld + Chins + Oth_A + Black + Othr_t) / Rsdnt, #
  imd19_ext = IMD20, # proportion of a larger area's population living in the most dep
  hlthsoc_sec = H____ / E_16_, # share of workforce in the human health & social work
  educ_sec = Edctn / E_16_, # share of workforce in the education sector
```

```
  trnsp_sec= Trn__ / E_16_, # share of workforce in the Transport & storage sector
  accfood_sec = Ac___ / E_16_, # share of workforce in the accommodation & food service sector
  admsupport_sec = Adm__ /  E_16_, # share of workforce in the administrative & support sector
  pblic_sec = Pb___ / E_16_ # share of workforce in the public administration & defence sector
)
```

Let's quickly examine how they correlate to our outcome variable i.e. incidence rate of COVID-19 using correlation coefficients and correlograms.

```
# obtain a matrix of Pearson correlation coefficients
df_sel <- st_set_geometry(utla_shp[,37:48], NULL) # temporary data set removing geometries
cormat <- cor(df_sel, use="complete.obs", method="pearson")

# significance test
sig1 <- corrplot::cor.mtest(df_sel, conf.level = .95)

# creta a correlogram
corrplot::corrplot(cormat, type="lower",
                   method = "circle",
                   order = "original",
                   tl.cex = 0.7,
                   p.mat = sig1$p, sig.level = .05,
                   col = viridis::viridis(100, option = "plasma"),
                   diag = FALSE)
```



The correlogram shows the strength and significance of the linear relationship between our set of variables. The size of the circle reflects the strength of the relationships as captured by the Pearson correlation coefficient, and crosses indicate statistically insignificant relationships at the 95% level of confidence. The colour indicate the direction of the relationship with dark (light) colours indicating a negative (positive) association.

The results indicate that the incidence of COVID-19 is significantly and positively related to the share of overcrowded housing, nonwhite ethnic minorities and administrative & support workers. Against expectations, the incidence of COVID-19 appears to be negatively correlated with the share of elderly population, of population suffering from long-term illness and of administrative & support workers, and displays no significant association with the share of the population living in deprived areas as well as the share of public administration & defence workers, and health & social workers. The latter probably reflects the effectiveness of the protective measures undertaken to prevent infection among these population groups, but it may also reflect the partial coverage of COVID-19 testing and underreporting. It may also reveal the descriptive limitations of correlation coefficients as they show the relationship between a pairs of variables,

not controlling for others. Correlation coefficients can thus produce spurious relationships resulting from confounded variables. We will return to this point below.

The results also reveal high collinearity between particular pairs of variables, notably between the share of crowded housing and of nonwhite ethnic population, the share of crowded housing and of elderly population, the share of overcrowded housing and of administrative & support workers, the share of elderly population and of population suffering from long-term illness. A more refined analysis of multicollinearity is needed. Various diagnostics for multicollinearity in a regression framework exist, including matrix condition numbers (CNs), predictor variance inflation factors (VIFs) and variance decomposition factors (VDPs). Rules of thumb (CNs > 30, VIFs > 10 and VDPs > 0.5) to indicate worrying levels of collinearity can be found in Belsley et al. (2005). To avoid problems of multicollinearity, often a simple strategy is to remove highly correlated predictors. The difficultly is in deciding which predictor(s) to remove, especially when all are considered important. Keep this in mind when specifying your model.

> Challenge 2: Analyse the relationship of all the variables executing `pairs(df_sel)`. How accurate would a linear regression be in capturing the relationships for our set of variables?

10.5.1 Global Regression Results

To gain a better understanding of these relationships, we can regress the incidence rate of COVID-19 on a series of factors capturing differences across areas. To focus on the description of GWR, we keep our analysis simple and study the incidence rate of COVID-19 as a function of the share of nonwhite ethnic population and of population suffering from long-term illness by estimating the following OLS linear regression model:

```
# attach data
attach(utla_shp)

# specify a model equation
eq1 <- covid19_r ~ ethnic + lt_illness
model1 <- lm(formula = eq1, data = utla_shp)

# estimates
summary(model1)
```

```
##
## Call:
## lm(formula = eq1, data = utla_shp)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
```

```
## -109.234  -38.386   -4.879   29.284  143.786
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)    63.77      30.13   2.117    0.036 *
## ethnic        271.10      30.65   8.845 2.64e-15 ***
## lt_illness    216.20     151.88   1.424    0.157
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 51 on 147 degrees of freedom
## Multiple R-squared:  0.3926, Adjusted R-squared:  0.3844
## F-statistic: 47.52 on 2 and 147 DF,  p-value: < 2.2e-16
```

We also compute the VIFs for the variables in the model:

```
vif(model1)
```

```
##     ethnic lt_illness
##    1.43015    1.43015
```

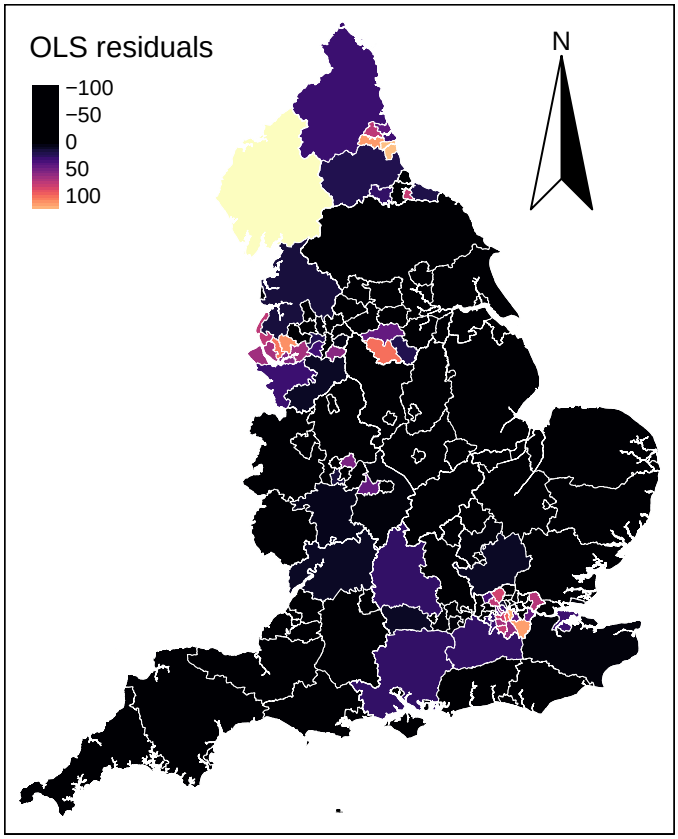
The regression results indicate a positive relationship exists between the share of nonwhite population and an increased risk of COVID-19 infection. A one percentage point increase in the share of nonwhite population returns a 271 rise in the cumulative count of COVID-19 infection per 100,000 people, everything else constant. The results also reveal a positive (albeit statistically insignificant) relationship between the share of population suffering from long-term illness and an increased risk of COVID-19 infection, after controlling for the share of nonwhite population, thereby confirming our suspicion about the limitations of correlation coefficients; that is, once differences in the share of nonwhite population are taken into account, the association between the share of population suffering from long-term illness and an increased risk of COVID-19 infection becomes positive. We also test for multicollinearity. The VIFs are below 10 indicating that multicollinearity is not highly problematic.

The R^2 value for the OLS regression is 0.393 indicating that our model explains only 39% of the variance in the rate of COVID-19 infection. This leaves 71% of the variance unexplained. Some of this unexplained variance can be because we have only included two explanatory variables in our model, but also because the OLS regression model assumes that the relationships in the model are constant over space; that is, it assumes a stationary process. Hence, an OLS regression model is considered to capture global relationships. However, relationships may vary over space. Suppose, for instance, that there are intrinsic behavioural variations across England and that people have adhered more strictly to self-isolation and social distancing measures in some areas than in others, or that ethnic minorities are less exposed to contracting COVID-19 in certain parts of England. If such variations in associations exist over space, our estimated OLS model will be a misspecification of reality because it assumes these relationships

to be constant.

To better understand this potential misspecification, we investigate the model residuals which show high variability (see below). The distribution is non-random displaying large positive residuals in the metropolitan areas of London, Liverpool, Newcastle (in light colours) and the Lake District and large negative residuals across much of England (in black). This conforms to the spatial pattern of confirmed COVID-19 cases with high concentration in a limited number of metropolitan areas (see above). While our residual map reveals that there is a problem with the OLS model, it does not indicate which, if any, of the parameters in the model might exhibit spatial nonstationarity. A simple way of examining if the relationships being modelled in our global OLS model are likely to be stationary over space would be to estimate separate OLS model for each UTLA in England. But this would require higher resolution i.e. data within UTLA, and we only have one data point per UTLA. (2002) (2002, p.40-44) discuss alternative approaches and their limitations.

```
utla_shp$res_m1 <- residuals(model1)

# map
legend_title = expression("OLS residuals")
map_utla = tm_shape(utla_shp) +
  tm_fill(col = "res_m1", title = legend_title, palette = magma(256), style = "cont") + # add fil
  tm_borders(col = "white", lwd = .1)  + # add borders
  tm_compass(type = "arrow", position = c("right", "top") , size = 5) + # add compass
  tm_scale_bar(breaks = c(0,1,2), text.size = 0.7, position =  c("center", "bottom")) + # add sca
  tm_layout(bg.color = "white") # change background colour
map_utla + tm_shape(reg_shp) + # add region boundaries
  tm_borders(col = "white", lwd = .5) # add borders
```



10.6 Fitting a Geographically Weighted Regression

GWR overcomes the limitation of the OLS regression model of generating a global set of estimates. The basic idea behind GWR is to examine the way in which the relationships between a dependent variable and a set of predictors might vary over space. GWR operates by moving a search window from one regression point to the next, working sequentially through all the existing regression points in the dataset. A set of regions is then defined around each regression point and within the search window. A regression model is then fitted to all data contained in each of the identified regions around a regression point, with data points closer to the sample point being weighted more heavily than are those farther away. This process is repeated for all samples points in the dataset. For a data set of 150 observations GWR will fit 150 weighted regression models. The resulting local estimates can then be mapped at the locations of the regression points to view possible variations in the relationships between variables.

Graphically, GWR involves fitting a spatial kernel to the data as described in the Fig. 1. For a given regression point X, the weight (W) of a data point is at a maximum at the location of the regression point. The weight decreases gradually as the distance between two points increases. A regression model is thus calibrated locally by moving the regression point across the area under study. For each location, the data are weighted differently so that the resulting estimates are unique to a particular location.

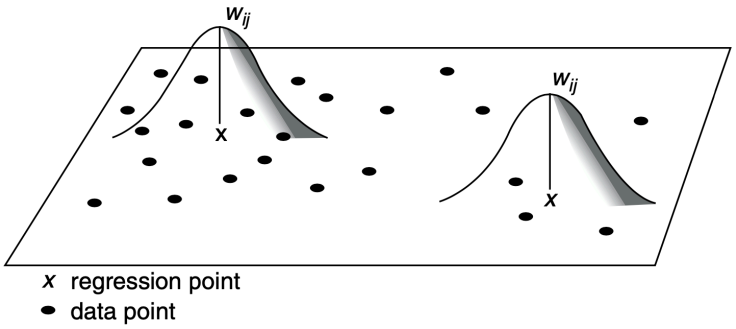


Figure 10.1: Fig. 1. GWR with fixed spatial kernel. Source: Fotheringham et al. (2002, p.45).

10.6.1 Fixed or Adaptive Kernel

A key issue is to decide between two options of spatial kernels: a fixed kernel or an adaptive kernel. Intuitively, a fixed kernel involves using a fixed bandwidth to define a region around all regression points as displayed in Fig. 1. The extent of the kernel is determined by the distance to a given regression point, with the kernel being identical at any point in space. An adaptive kernel involves using varying bandwidth to define a region around regression points as displayed in Fig. 2. The extent of the kernel is determined by the number of nearest neighbours from a given regression point. The kernels have larger bandwidths where the data are sparse.

10.6.2 Optimal Bandwidth

A second issue is to define the extent of geographical area (i.e. *optimal bandwidth*) of the spatial kernel. The bandwidth is the distance beyond which a value of zero is assigned to weight observations. Larger bandwidths include a larger number of observations receiving a non-zero weight and more observations are used to fit a local regression.

To determine the optimal bandwidth, a cross-validation approach is applied; that is, for a location, a local regression is fitted based on a given bandwidth and used to predict the value of the dependent variable. The resulting predicted

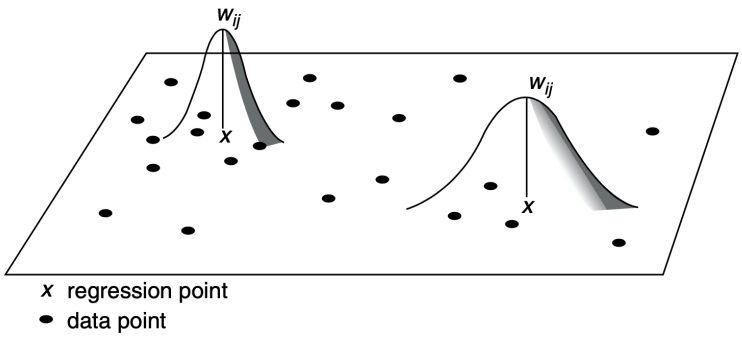


Figure 10.2: Fig. 2. GWR with adaptive spatial kernel. Source: Fotheringham et al. (2002, p.47).

value is used to compute the residuals of the model. Residuals are compared using a series of bandwidth and the bandwidth returning the smallest local residuals are selected.

Variance and Bias Trade off

Choosing an optimal bandwidth involves a compromise between bias and precision. For example, a larger bandwidth will involve using a larger number of observations to fit a local regression, and hence result in reduced variance (or increased precision) but high bias of estimates. On the other hand, too small bandwidth involves using a very small number of observations resulting in increased variance but small bias. An optimal bandwidth offers a compromise between bias and variance.

10.6.3 Shape of Spatial Kernel

Two general set of kernel functions can be distinguished: continuous kernels and kernels with compact support. Continuous kernels are used to weight all observations in the study area and includes uniform, Gaussian and Exponential kernel functions. Kernel with compact support are used to assign a nonzero weight to observations within a certain distance and a zero weight beyond it. The shape of the kernel has been reported to cause small changes to resulting estimates (Brunsdon et al., 1998).

10.6.4 Selecting a Bandwidth

Let's now implement a GWR model. The first key step is to define the optimal bandwidth. We first illustrate the use of a fixed spatial kernel.

10.6.4.1 Fixed Bandwidth

Cross-validation is used to search for the optimal bandwidth. Recall that this procedure compares the model residuals based on different bandwidths and chooses the optimal solution i.e. the bandwidth returning the smallest model residuals based on a given model specification. A key parameter here is the shape of the geographical weight function (`gweight`). We set it to be a Gaussian function which is the default. A bi-square function is recommended to reduce computational time. Since we have a simple model, a Gaussian function should not take that long. Note that we set the argument `longlat` to `TRUE` and use latitude and longitude for coordinates (`coords`). When `longlat` is set to `TRUE`, distances are measured in kilometres.

```
# find optimal kernel bandwidth using cross validation
fbw <- gwr.sel(eq1,
               data = utla_shp,
               coords=cbind( long, lat),
               longlat = TRUE,
               adapt=FALSE,
               gweight = gwr.Gauss,
               verbose = FALSE)

# view selected bandwidth
fbw
```

```
## [1] 29.30417
```

The result indicates that the optimal bandwidth is 39.79 kms. This means that neighbouring UTLAs within a fixed radius of 39.79 kms will be taken to estimate local regressions. To estimate a GWR, we execute the code below in which the optimal bandwidth above is used as an input in the argument `bandwidth`.

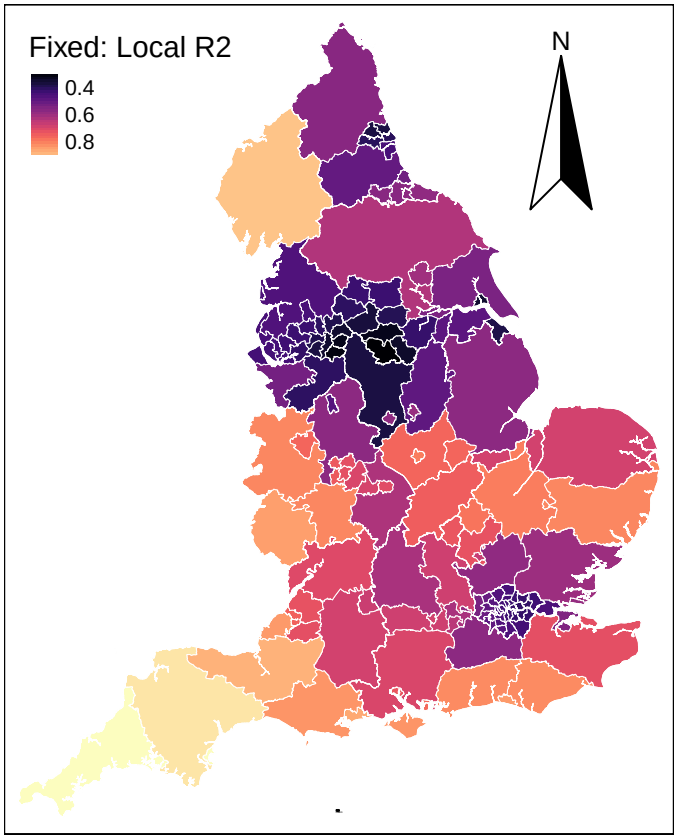
```
# fit a gwr based on fixed bandwidth
fb_gwr <- gwr(eq1,
              data = utla_shp,
              coords=cbind( long, lat),
              longlat = TRUE,
              bandwidth = fbw,
              gweight = gwr.Gauss,
              hatmatrix=TRUE,
              se.fit=TRUE)

fb_gwr
```

```
## Call:
## gwr(formula = eq1, data = utla_shp, coords = cbind(long, lat),
##     bandwidth = fbw, gweight = gwr.Gauss, hatmatrix = TRUE, longlat = TRUE,
##     se.fit = TRUE)
```

```
## Kernel function: gwr.Gauss
## Fixed bandwidth: 29.30417
## Summary of GWR coefficient estimates at data points:
##                   Min.   1st Qu.    Median   3rd Qu.      Max.  Global
## X.Intercept.  -187.913   -42.890    93.702   211.685   792.989  63.768
## ethnic        -785.938   104.813   194.609   254.717  1078.854 271.096
## lt_illness   -2599.119  -563.128   128.176   690.603  1507.024 216.198
## Number of data points: 150
## Effective number of parameters (residual: 2traceS - traceS'S): 57.11019
## Effective degrees of freedom (residual: 2traceS - traceS'S): 92.88981
## Sigma (residual: 2traceS - traceS'S): 38.34777
## Effective number of parameters (model: traceS): 44.65744
## Effective degrees of freedom (model: traceS): 105.3426
## Sigma (model: traceS): 36.00992
## Sigma (ML): 30.17717
## AICc (GWR p. 61, eq 2.33; p. 96, eq. 4.21): 1580.349
## AIC (GWR p. 96, eq. 4.22): 1492.465
## Residual sum of squares: 136599.2
## Quasi-global R2: 0.7830537
```

We will skip the interpretation of the results for now and consider them in the next section. Now, we want to focus on the overall model fit and will map the results of the R^2 for the estimated local regressions. To do this, we extract the model results stored in a Spatial Data Frame (SDF) and add them to our spatial data frame `utla_shp`. Note that the Quasi-global R^2 is very high (0.77) indicating a high in-sample prediction accuracy.

```
# write gwr output into a data frame
fb_gwr_out <- as.data.frame(fb_gwr$SDF)

utla_shp$fmb_localR2 <- fb_gwr_out$localR2

# map
  # Local R2
legend_title = expression("Fixed: Local R2")
map_fbgwr1 = tm_shape(utla_shp) +
  tm_fill(col = "fmb_localR2", title = legend_title, palette = magma(256), style = "co
  tm_borders(col = "white", lwd = .1)  + # add borders
  tm_compass(type = "arrow", position = c("right", "top") , size = 5) + # add compass
  tm_scale_bar(breaks = c(0,1,2), text.size = 0.7, position =  c("center", "bottom")) +
  tm_layout(bg.color = "white") # change background colour
map_fbgwr1 + tm_shape(reg_shp) + # add region boundaries
  tm_borders(col = "white", lwd = .5) # add borders
```



The map shows very high in-sample model predictions of up to 80% in relatively large UTLAs (i.e. Cornwall, Devon and Cumbria) but poor predictions in Linconshire and small UTLAs in the North West and Yorkshire & The Humber Regions and the Greater London. The spatial distribution of this pattern may reflect a potential problem that arise in the application of GWR with fixed spatial kernels. The use of fixed kernels implies that local regressions for small spatial units may be calibrated on a large number of dissimilar areas, while local regressions for large areas may be calibrated on very few data points, giving rise to estimates with large standard errors. In extreme cases, generating estimates might not be possible due to insufficient variation in small samples. In practice, this issue is relatively common if the number of geographical areas in the dataset is small.

10.6.4.2 Adaptive Bandwidth

To reduce these problems, adaptive spatial kernels can be used. These kernels adapt in size to variations in the density of the data so that the kernels have larger bandwidths where the data are sparse and have smaller bandwidths where the data are plentiful. As above, we first need to search for the optimal

bandwidth before estimating a GWR.

```
# find optimal kernel bandwidth using cross validation
abw <- gwr.sel(eq1,
               data = utla_shp,
               coords=cbind( long, lat),
               longlat = TRUE,
               adapt = TRUE,
               gweight = gwr.Gauss,
               verbose = FALSE)

# view selected bandwidth
abw
```

```
## [1] 0.03126972
```

The optimal bandwidth is 0.03 indicating the proportion of observations (or k-nearest neighbours) to be included in the weighting scheme. In this example, the optimal bandwidth indicates that for a given UTLA, 3% of its nearest neighbours should be used to calibrate the relevant local regression; that is about 5 UTLAs. The search window will thus be variable in size depending on the extent of UTLAs. Note that here the optimal bandwidth is defined based on a data point's k-nearest neighbours. It can also be defined by geographical distance as done above for the fixed spatial kernel. We next fit a GWR based on an adaptive bandwidth.

```
# fit a gwr based on adaptive bandwidth
ab_gwr <- gwr(eq1,
              data = utla_shp,
              coords=cbind( long, lat),
              longlat = TRUE,
              adapt = abw,
              gweight = gwr.Gauss,
              hatmatrix=TRUE,
              se.fit=TRUE)

ab_gwr
```

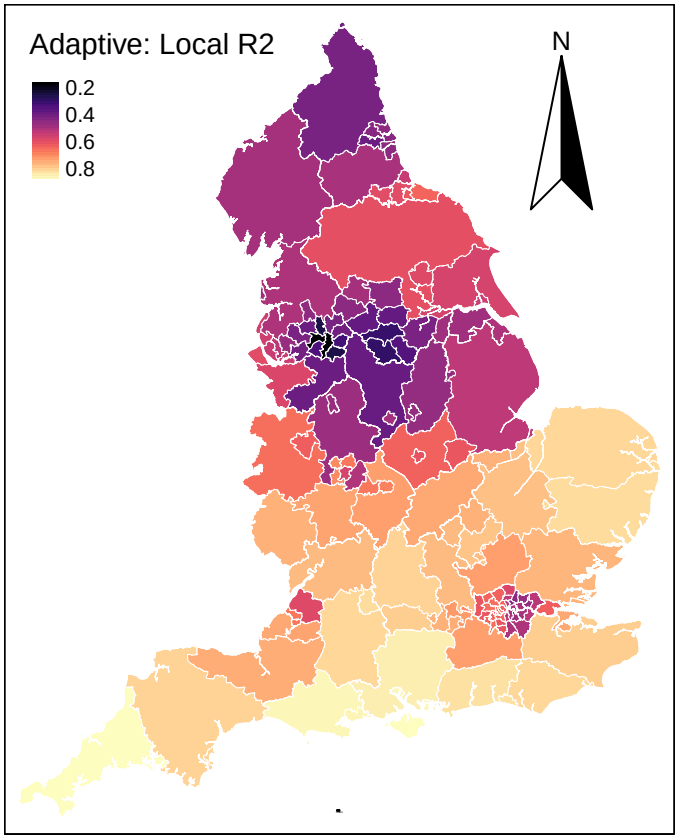
```
## Call:
## gwr(formula = eq1, data = utla_shp, coords = cbind(long, lat),
##     gweight = gwr.Gauss, adapt = abw, hatmatrix = TRUE, longlat = TRUE,
##     se.fit = TRUE)
## Kernel function: gwr.Gauss
## Adaptive quantile: 0.03126972 (about 4 of 150 data points)
## Summary of GWR coefficient estimates at data points:
##                    Min.   1st Qu.    Median   3rd Qu.      Max.  Global
## X.Intercept.  -198.790   -28.398   113.961   226.437   346.510  63.768
## ethnic        -121.872   106.822   229.591   283.739  1162.123 271.096
```

```
## lt_illness   -1907.098  -746.468  -125.855   798.875  1496.549 216.198
## Number of data points: 150
## Effective number of parameters (residual: 2traceS - traceS'S): 48.59361
## Effective degrees of freedom (residual: 2traceS - traceS'S): 101.4064
## Sigma (residual: 2traceS - traceS'S): 36.57493
## Effective number of parameters (model: traceS): 36.04378
## Effective degrees of freedom (model: traceS): 113.9562
## Sigma (model: traceS): 34.50222
## Sigma (ML): 30.07257
## AICc (GWR p. 61, eq 2.33; p. 96, eq. 4.21): 1546.029
## AIC (GWR p. 96, eq. 4.22): 1482.809
## Residual sum of squares: 135653.9
## Quasi-global R2: 0.7845551
```

10.6.5 Model fit

Assessing the global fit of the model, marginal improvements are observed. The *AIC* and *Residual sum of squares* experienced marginal reductions, while the R^2 increased compared to the GRW based on a fixed kernel. To gain a better understanding of these changes, as above, we map the R^2 values for the estimated local regressions.

```
# write gwr output into a data frame
ab_gwr_out <- as.data.frame(ab_gwr$SDF)

utla_shp$amb_ethnic <- ab_gwr_out$ethnic
utla_shp$amb_lt_illness <- ab_gwr_out$lt_illness
utla_shp$amb_localR2 <- ab_gwr_out$localR2

# map
  # Local R2
legend_title = expression("Adaptive: Local R2")
map_abgwr1 = tm_shape(utla_shp) +
  tm_fill(col = "amb_localR2", title = legend_title, palette = magma(256), style = "cont") + # ad
  tm_borders(col = "white", lwd = .1)  + # add borders
  tm_compass(type = "arrow", position = c("right", "top") , size = 5) + # add compass
  tm_scale_bar(breaks = c(0,1,2), text.size = 0.7, position =  c("center", "bottom")) + # add sca
  tm_layout(bg.color = "white") # change background colour
map_abgwr1 + tm_shape(reg_shp) + # add region boundaries
  tm_borders(col = "white", lwd = .5) # add borders
```



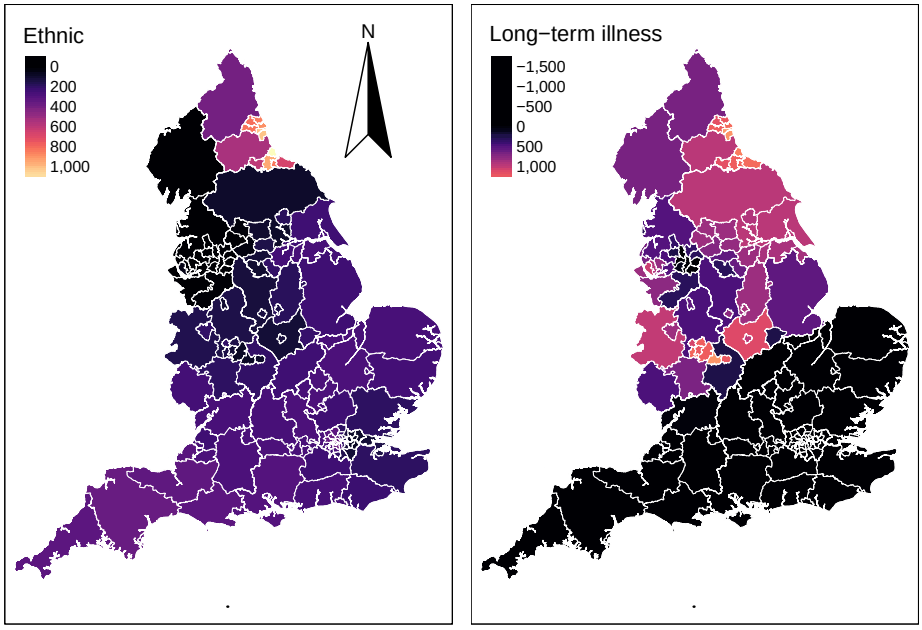
The map reveals notable improvements in local estimates for UTLAs within West and East Midlands, the South East, South West and East of England. Estimates are still poor in hot spot UTLAs concentrating confirmed cases of COVID-19, such as the Greater London, Liverpool and Newcastle areas.

10.6.6 Interpretation

The key strength of GWR models is in identifying patterns of spatial variation in the associations between pairs of variables. The results reveal how these coefficients vary across the 150 UTLAs of England. To examine this variability, let's first focus on the adaptive GWR output reported in Section 8.6.4.2. The output includes a summary of GWR coefficient estimates at various data points. The last column reports the global estimates which are the same as the coefficients from the OLS regression we fitted at the start of our analysis. For our variable nonwhite ethnic population, the GWR outputs reveals that local coefficients range from a minimum value of -148.41 to a maximum value of 1076.84, indicating that one percentage point increase in the share of nonwhite ethnic population is associated with a a reduction of 148.41 in the number of cumulative confirmed cases of COVID-19 per 100,000 people in some UTLAs

and an increase of 1076.84 in others. For half of the UTLAs in the dataset, as the share of nonwhite ethnic population increases by one percentage point, the rate of COVID-19 will increase between 106.29 and 291.24 cases; that is, the inter-quartile range between the 1st Qu and the 3rd Qu. To analyse the spatial structure, we next map the estimated coefficients obtained from the adaptive kernel GWR.

```
  # Ethnic
legend_title = expression("Ethnic")
map_abgwr2 = tm_shape(utla_shp) +
  tm_fill(col = "amb_ethnic", title = legend_title, palette = magma(256), style = "cont") + # ado
  tm_borders(col = "white", lwd = .1)  + # add borders
  tm_compass(type = "arrow", position = c("right", "top") , size = 5) + # add compass
  tm_scale_bar(breaks = c(0,1,2), text.size = 0.7, position =  c("center", "bottom")) + # add sca
  tm_layout(bg.color = "white") # change background colour
map_abgwr2 = map_abgwr2 + tm_shape(reg_shp) + # add region boundaries
  tm_borders(col = "white", lwd = .5) # add borders

  # Long-term Illness
legend_title = expression("Long-term illness")
map_abgwr3 = tm_shape(utla_shp) +
  tm_fill(col = "amb_lt_illness", title = legend_title, palette = magma(256), style = "cont") + #
  tm_borders(col = "white", lwd = .1)  + # add borders
  tm_scale_bar(breaks = c(0,1,2), text.size = 0.7, position =  c("center", "bottom")) + # add sca
  tm_layout(bg.color = "white") # change background colour
map_abgwr3 = map_abgwr3 + tm_shape(reg_shp) + # add region boundaries
  tm_borders(col = "white", lwd = .5) # add borders

tmap_arrange(map_abgwr2, map_abgwr3)
```



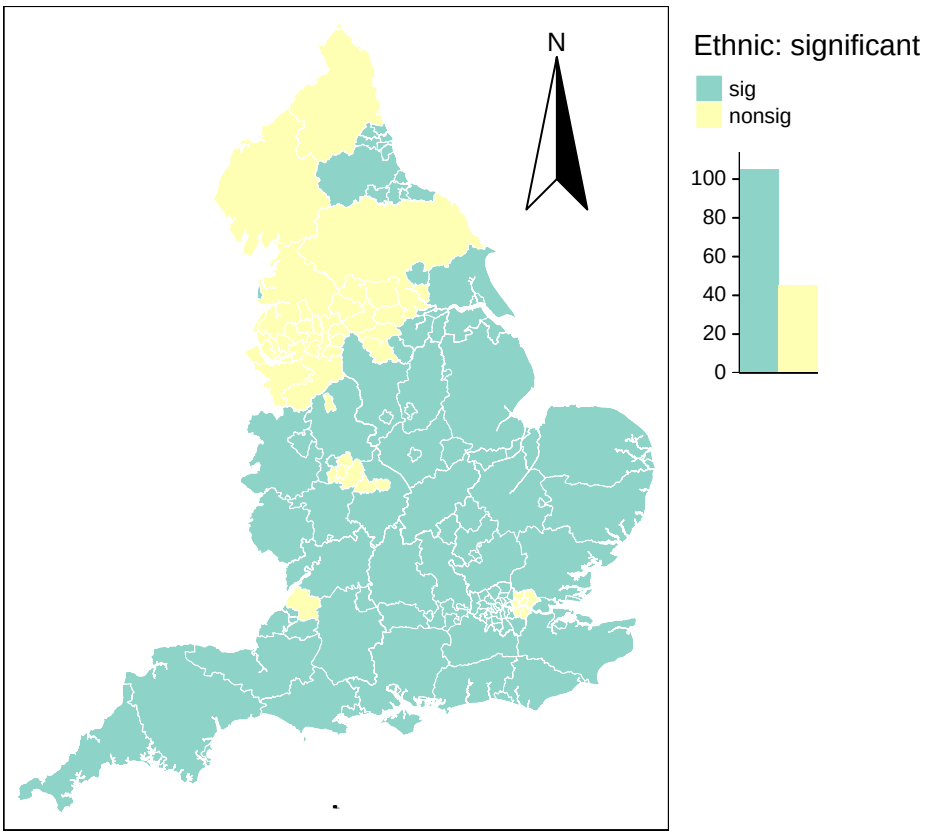
Analysing the map for long-term illness, a clear North-South divide can be identified. In the North we observed the expected positive relationship between COVID-19 and long-term illness i.e. as the share of the local population suffering from long-term illness rises, the cumulative number of positive COVID-19 cases is expected to increase. In the South, we observe the inverse pattern i.e. as the share of local population suffering from long-term illness rises, the cumulative number of positive COVID-19 cases is expected to drop. This pattern is counterintuitive but may be explained by the wider socio-economic disadvantages between the North and the South of England. The North is usually characterised by a persistent concentration of more disadvantaged neighbourhoods than the South where affluent households have tended to cluster for the last 40 years (Rowe et al., 2020).

10.6.7 Assessing statistical significance

While the maps above offer valuable insights to understand the spatial pattering of relationships, they do not identify whether these associations are statistically significant. They may not be. Roughly, if a coefficient estimate has an absolute value of t greater than 1.96 and the sample is sufficiently large, then it is statistically significant. Our sample has only 150 observations, so we are more conservative and considered a coefficient to be statistically significant if it has an absolute value of t larger than 2. Note also that p-values could be computed - see Lu et al. (2014).

```
# compute t statistic
utla_shp$t_ethnic = ab_gwr_out$ethnic / ab_gwr_out$ethnic_se
```

```
# categorise t values
utla_shp$t_ethnic_cat <- cut(utla_shp$t_ethnic,
                      breaks=c(min(utla_shp$t_ethnic), -2, 2, max(utla_shp$t_ethnic)),
                      labels=c("sig","nonsig", "sig"))

# map statistically significant coefs for ethnic
legend_title = expression("Ethnic: significant")
map_sig = tm_shape(utla_shp) +
  tm_fill(col = "t_ethnic_cat", title = legend_title, legend.hist = TRUE, midpoint = NA, textNA =
  tm_borders(col = "white", lwd = .1)  + # add borders
  tm_compass(type = "arrow", position = c("right", "top") , size = 5) + # add compass
  tm_scale_bar(breaks = c(0,1,2), text.size = 0.7, position =  c("center", "bottom")) + # add sc
  tm_layout(bg.color = "white", legend.outside = TRUE) # change background colour & place legend

map_sig + tm_shape(reg_shp) + # add region boundaries
  tm_borders(col = "white", lwd = .5) # add borders
```



```
# utla count
table(utla_shp$t_ethnic_cat)
```

```
##
##    sig nonsig
##    105     45
```

For the share of nonwhite population, 67% of all local coefficients are statistically significant and these are largely in the South of England. Coefficients in the North tend to be insignificant. Through outliers exist in both regions. In the South, nonsignificant coefficients are observed in the metropolitan areas of London, Birmingham and Nottingham, while significant coefficients exist in the areas of Newcastle and Middlesbrough in the North.

> Challenge 3 Compute the t values for the intercept and estimated coefficient for long-term illness and create maps of their statistical significance. How many UTLAs report statistically significant coefficients?

10.6.8 Collinearity in GWR

An important final note is: collinearity tends to be problematic in GWR models. It can be present in the data subsets to estimate local coefficients even when not observed globally Wheeler and Tiefelsdorf (2005). Collinearity can be highly problematic in the case of compositional, categorical and ordinal predictors, and may result in exact local collinearity making the search for an optimal bandwidth impossible. A recent paper suggests potential ways forward (Comber et al., 2020).

Chapter 11

Spatio-Temporal Analysis

This chapter[1] provides an introduction to the complexities of spatio-temporal data and modelling. For modelling, we consider the Fixed Rank Kriging (FRK) framework developed by Cressie and Johannesson (2008). It enables constructing a spatial random effects model on a discretised spatial domain. Key advantages of this approach comprise the capacity to: (1) work with large data sets, (2) be scaled up; (3) generate predictions based on sparse linear algebraic techniques, and (4) produce fine-scale resolution uncertainty estimates.

The content of this chapter is based on:

- Wikle et al. (2019), a recently published book which provides a good overview of existing statistical approaches to spatio-temporal modelling and R packages.

- Zammit-Mangion and Cressie (2017), who introduce the statistical framework and R package for modelling spatio-temporal used in this Chapter.

This Chapter is part of Spatial Analysis Notes, a compilation hosted as a GitHub repository that you can access in a few ways:

- As a download of a `.zip` file that contains all the materials.
- As an html website.
- As a pdf document
- As a GitHub repository.

[1] This note is part of Spatial Analysis Notes Space-Time Analysis – Spatio-temporal modelling by Francisco Rowe is licensed under a Creative Commons Attribution-NonCommercial-ShareAlike 4.0 International License.

11.1 Dependencies

This chapter uses the following libraries: Ensure they are installed on your machine[2] before loading them executing the following code chunk:

```
# Data manipulation, transformation and visualisation
library(tidyverse)
# Nice tables
library(kableExtra)
# Simple features (a standardised way to encode vector data ie. points, lines, polygon
library(sf)
# Spatial objects conversion
library(sp)
# Thematic maps
library(tmap)
# Nice colour schemes
library(viridis)
# Obtain correlation coefficients
library(corrplot)
# Highlight data on plots
library(gghighlight)
# Analysing spatio-temporal data
#library(STRbook)
library(spacetime)
# Date parsing and manipulation
library(lubridate)
# Applied statistics
library(MASS)
# Statistical tests for linear regression models
library(lmtest)
# Fit spatial random effects models
library(FRK)
# Exportable regression tables
library(jtools)
```

11.2 Data

For this chapter, we will use data on:

- COVID-19 confirmed cases from 30th January, 2020 to 21st April, 2020 from Public Health England via the GOV.UK dashboard;
- resident population characteristics from the 2011 census, available from

[2] You can install package `mypackage` by running the command `install.packages("mypackage")` on the R prompt or through the `Tools --> Install Packages...` menu in RStudio.

the Office of National Statistics; and,

- 2019 Index of Multiple Deprivation (IMD) data from GOV.UK and published by the Ministry of Housing, Communities & Local Government. The data are at the ONS Upper Tier Local Authority (UTLA) level - also known as Counties and Unitary Authorities.

For a full list of the variables included in the data sets used in this chapter, see the readme file in the sta data folder.[3]. Before we get our hands on the data, there are some important concepts that need to be introduced. They provide a useful framework to understand the complex structure of spatio-temporal data. Let's start by first highlighling the importance of spatio-temporal analysis.

11.3 Why Spatio-Temporal Analysis?

Investigating the spatial patterns of human processes as we have done so far in this book only offers a partial incomplete representation of these processes. It does not allow understanding of the temporal evolution of these processes. Human processes evolve in space and time. Human mobility is a inherent geographical process which changes over the course of the day, with peaks at rush hours and high concentration towards employment, education and retail centres. Exposure to air pollution changes with local climatic conditions, and emission and concentration of atmospheric pollutants which fluctuate over time. The rate of disease spread varies over space and may significantly change over time as we have seen during the current outbreak, with flattened or rapid declining trends in Australia, New Zealand and South Korea but fast proliferation in the United Kingdom and the United States. Only by considering time and space together we can address how geographic entities change over time and why they change. A large part of how and why of such change occurs is due to interactions across space and time, and multiple processes. It is essential to understand the past to inform our understanding of the present and make predictions about the future.

11.3.1 Spatio-temporal Data Structures

A first key element is to understand the structure of spatio-temporal data. Spatio-temporal data incorporate two dimensions. At one end, we have the temporal dimension. In quantitative analysis, time-series data are used to capture geographical processes at regular or irregular intervals; that is, in a continuous (daily) or discrete (only when a event occurs) temporal scale. At another end, we have the spatial dimension. We often use spatial data as temporal aggregations or temporally frozen states (or 'snapshots') of a geographical process - this is what we have done so far. Recall that spatial data can be capture in different geographical units, such as areal or lattice, points, flows or trajectories - refer to the introductory lecture in Week 1. Relatively few ways exist to formally integrate temporal and spatial data in consistent analytical framework. Two

[3]Read the file in R by executing `read_tsv("data/sta/readme.txt")`

notable exceptions in R are the packages `TraMiner` (Gabadinho et al., 2009) and `spacetime` (Pebesma et al., 2012). We use the class definitions defined in the R package `spacetime`. These classes extend those used for spatial data in `sp` and time-series data in `xts`. Next a brief introduction to concepts that facilitate thinking about spatio-temporal data structures.

11.3.1.1 Type of Table

Spatio-temporal data can be conceptualised as three main different types of tables:

- time-wide: a table in which columns correspond to different time points
- space-wide: a table in which columns correspond to different spatial location
- long formats: a table in which each row-column pair corresponds to a specific time and spatial location (or space coordinate)

  Note that data in long format are space inefficient because spatial coordinates and time attributes are required for each data point. Yet, data in this format are relatively easy to manipulate via packages such as `dplyr` and `tidyr`, and visualise using `ggplot2`. These packages are designed to work with data in long format.

11.3.1.2 Type of Spatio-Temporal Object

To integrate spatio-temporal data, spatio-temporal objects are needed. We consider four different spatio-temporal frames (STFs) or objects which can be defined via the package `spacetime`:

- Full grid (STF): an object containing data on all possible locations in all time points in a sequence of data;
- Sparse grid (STS): an object similar to STF but only containing non-missing space-time data combinations;
- Irregular (STI): an object with an irregular space-time data structure, where each point is allocated a spatial coordinate and a time stamp;
- Simple Trajectories (STT): an object containig a sequence of space-time points that form trajectories.

More details on these spatio-temporal structures, construction and manipulation, see Pebesma et al. (2012). Enough theory, let's code!

11.4 Data Wrangling

This section illustrates the complexities of handling spatio-temporal data. It discusses good practices in data manipulation and construction of a Space Time

Irregular Data Frame (STIDF) object. Three key requirements to define a STFDF object are:

1. Have a data frame in long format i.e. a location-time pair data frame
2. Define a time stamp
3. Construct the spatio-temporal object of class STIDF by indicating the spatial and temporal coordinates

Let's now read all the required data. While we can have all data in a single data frame, you will find helpful to have separate data objects to identify:

- spatial locations
- temporal units
- data

These data objects correspond to `locs`, `time`, and `covid19` and `censusimd` below. Throughout the chapter you will notice that we switch between the various data frames when convinient, depending on the operation.

```
# clear workspace
rm(list=ls())

# read ONS UTLA shapefile
utla_shp <- st_read("data/sta/ons_utla.shp")
```

```
## Reading layer `ons_utla' from data source `/home/rstudio/Documents/data/sta/ons_utla.shp' usir
## Simple feature collection with 150 features and 11 fields
## geometry type:  MULTIPOLYGON
## dimension:      XY
## bbox:           xmin: 134112.4 ymin: 11429.67 xmax: 655653.8 ymax: 657536
## projected CRS:  Transverse_Mercator
```

```
# create table of locations
locs <- utla_shp %>% as.data.frame() %>%
  dplyr::select(objct, cty19c, ctyu19nm, long, lat, st_rs)

# read time data frame
time <- read_csv("data/sta/reporting_dates.csv")

# read COVID-19 data in long format
covid19 <- read_csv("data/sta/covid19_cases.csv")

# read census and IMD data
censusimd <- read_csv("data/sta/2011census_2019imd_utla.csv")
```

If we explore the structure of the data via `head` and `str`, we can see we have data on daily and cumulative new COVID-19 cases for 150 spatial units (i.e. UTLAs)

over 71 time points from January 30th to April 21st. We also have census and IMD data for a range of attributes.

```
head(covid19, 3)
```

```
## # A tibble: 3 x 6
##   Area.name  Area.code Area.type   date       Daily.lab.confi~ Cumulative.lab.c~
##   <chr>      <chr>     <chr>       <date>                <dbl>             <dbl>
## 1 Barking a~ E09000002 Upper tier~ 2020-01-30                0                 0
## 2 Barnet     E09000003 Upper tier~ 2020-01-30                0                 0
## 3 Barnsley   E08000016 Upper tier~ 2020-01-30                0                 0
```

Once we have understood the structure of the data, we first need to confirm if the `covid19` data are in wide or long format. Luckily they are in long format; otherwise, we would have needed to transform the data from wide to long format. Useful functions to achieve this include `pivot_longer` (`pivot_longer`) which has superseded `gather` (`spread`) in the `tidyr` package. Note that the `covid19` data frame has 10,650 observations (i.e. rows); that is, 150 UTLAs * 71 daily observations.

We then define a regular time stamp for our temporal data. We use the `lubridate` package to do this. A key advantage of `lubridate` is that it automatically recognises the common separators used when recording dates ("-", "/", ":", and ""). As a result, you only need to focus on specifying the order of the date elements to determine the parsing function applied. Below we check the structure of our time data, define a time stamp and create separate variables for days, weeks, months and year.

> Note that working with dates can be a complex task. A good discussion of these complexities is provided here.

```
# check the time structure used for reporting covid cases
head(covid19$date, 5)
```

```
## [1] "2020-01-30" "2020-01-30" "2020-01-30" "2020-01-30" "2020-01-30"
```

```
# parsing data into a time stamp
covid19$date <- ymd(covid19$date)
class(covid19$date)
```

```
## [1] "Date"
```

```
# separate date variable into day,week, month and year variables
covid19$day <- day(covid19$date)
covid19$week <- week(covid19$date) # week of the year
covid19$month <- month(covid19$date)
covid19$year <- year(covid19$date)
```

Once defined the time stamp, we need to add the spatial information contained in our shapefile to create a spatio-temporal data frame.

```
# join dfs
covid19_spt <- left_join(utla_shp, covid19, by = c("ctyu19nm" = "Area.name"))
```

We now have all the components to build a spatio-temporal object of class STIDF using `STIDF` from the `spacetime` package:

```
# identifying spatial fields
spat_part <- as(dplyr::select(covid19_spt, -c(bng_e, bng_n, Area.code, Area.type, Daily.lab.confi
```

```
## Warning in showSRID(uprojargs, format = "PROJ", multiline = "NO", prefer_proj
## = prefer_proj): Discarded datum Unknown based on Airy 1830 ellipsoid in CRS
## definition
```

```
## Warning in showSRID(SRS_string, format = "PROJ", multiline = "NO", prefer_proj =
## prefer_proj): Discarded datum D_unknown in CRS definition
```

```
# identifying temporal fields
temp_part <- covid19_spt$date

# identifying data
covid19_data <- covid19_spt %>% dplyr::select(c(Area.code, Area.type, date, Daily.lab.confirmed.c
  as.data.frame()

# construct STIDF object
covid19_stobj <- STIDF(sp = spat_part, # spatial fields
                time = temp_part, # time fields
                data = covid19_data) # data

class(covid19_stobj)
```

```
## [1] "STIDF"
## attr(,"package")
## [1] "spacetime"
```

We now add census and IMD variables. For the purposes of this Chapter, we only add total population and long-term sick or disabled population counts. You can add more variables by adding their names in the `select` function.

```
# select pop data
pop <- censusimd %>% dplyr::select("UTLA19NM", "Residents", "Longterm_sick_or_disabled")
# join dfs
covid19_spt <- left_join(covid19_spt, pop,
                         by = c("ctyu19nm" = "UTLA19NM"))
covid19 <- left_join(covid19, pop, by = c("Area.name" = "UTLA19NM"))
```

11.5 Exploring Spatio-Temporal Data

We now have all the required data in place. In this section various methods of data visualisation are illustrated before key dimensions of the data are explored. Both of these types of exploration can be challenging as one or more dimensions in space and one in time need to be interrogated.

11.5.1 Visualisation

In the context spatio-temporal data, a first challenge is data visualization. Visualising more than two dimensions of spatio-temporal data, so it is helpful to slice or aggregate the data over a dimension, use color, or build animations through time. Before exploring the data, we need to define our key variable of interest; that is, the number of confirmed COVID-19 cases per 100,000 people. We also compute the cumulative number of confirmed COVID-19 cases per 100,000 people as it may be handy in some analyses.

Fisrt create variable to be analysed:

```
# rate of new covid-19 infection
covid19_spt$n_covid19_r <- round( (covid19_spt$Daily.lab.confirmed.cases / covid19_spt$
covid19$n_covid19_r <- round( (covid19$Daily.lab.confirmed.cases / covid19$Residents) *

# risk of cumulative covid-19 infection
covid19_spt$c_covid19_r <- round( (covid19_spt$Cumulative.lab.confirmed.cases / covid19
covid19$c_covid19_r <- round( (covid19$Cumulative.lab.confirmed.cases / covid19$Residen
```

11.5.1.1 Spatial Plots

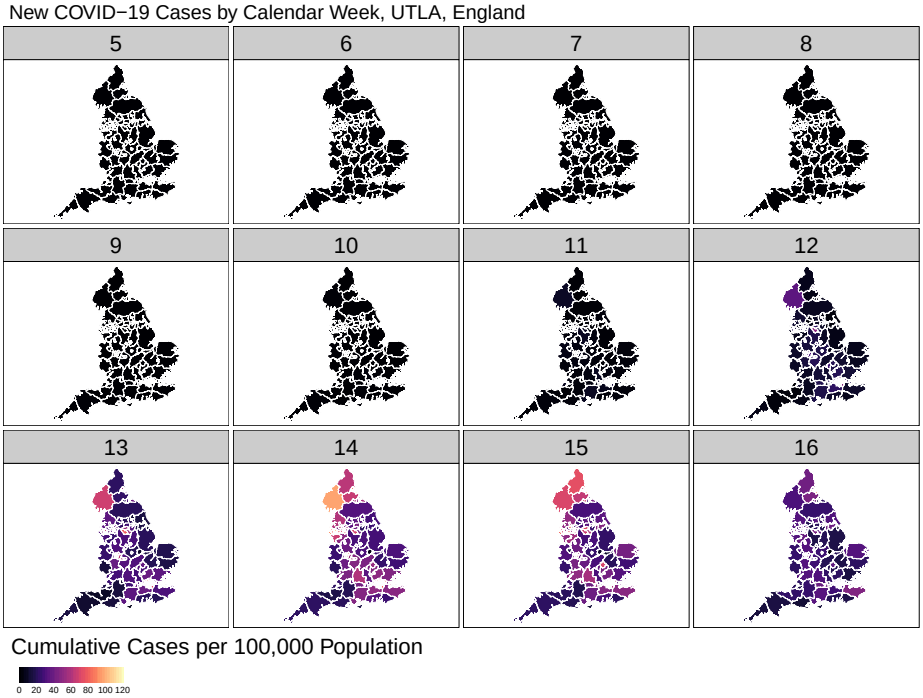
One way to visualise the data is using spatial plots; that is, snapshots of a geographic process for a given time period. Data can be mapped in different ways using clorepleth, countour or surface plots. The key aim of these maps is to understand how the overall extent of spatial variation and local patterns of spatial concentration change over time. Below we visualise the weekly number of confirmed COVID-19 cases per 100,000 people.

> Note that Weeks range from 5 to 16 as they refer to calendar weeks. Calendar week 5 is when the first COVID-19 case in England was reported.

```
# create data frame for new cases by week
daycases_week <- covid19_spt %>% group_by(week, ctyu19nm, as.character(cty19c), Residen
  summarise(n_daycases = sum(Daily.lab.confirmed.cases))

## `summarise()` regrouping output by 'week', 'ctyu19nm', 'as.character(cty19c)' (overr

# weekly rate of new covid-19 infection
daycases_week$wn_covid19_r <- (daycases_week$n_daycases / daycases_week$Residents) * 10
```

```
# map
legend_title = expression("Cumulative Cases per 100,000 Population")
tm_shape(daycases_week) +
  tm_fill("wn_covid19_r", title = legend_title, palette = magma(256), style ="cont", legend.hist=
  tm_facets(by = "week", ncol = 4) +
  tm_borders(col = "white", lwd = .1)  + # add borders +
  tm_layout(bg.color = "white", # change background colour
            legend.outside = TRUE, # legend outside
            legend.outside.position = "bottom",
            legend.stack = "horizontal",
            legend.title.size = 2,
            legend.width = 1,
            legend.height = 1,
            panel.label.size = 3,
            main.title = "New COVID-19 Cases by Calendar Week, UTLA, England")
```

```
## Warning in pre_process_gt(x, interactive = interactive, orig_crs =
## gm$shape.orig_crs): legend.width controls the width of the legend within a map.
## Please use legend.outside.size to control the width of the outside legend
```



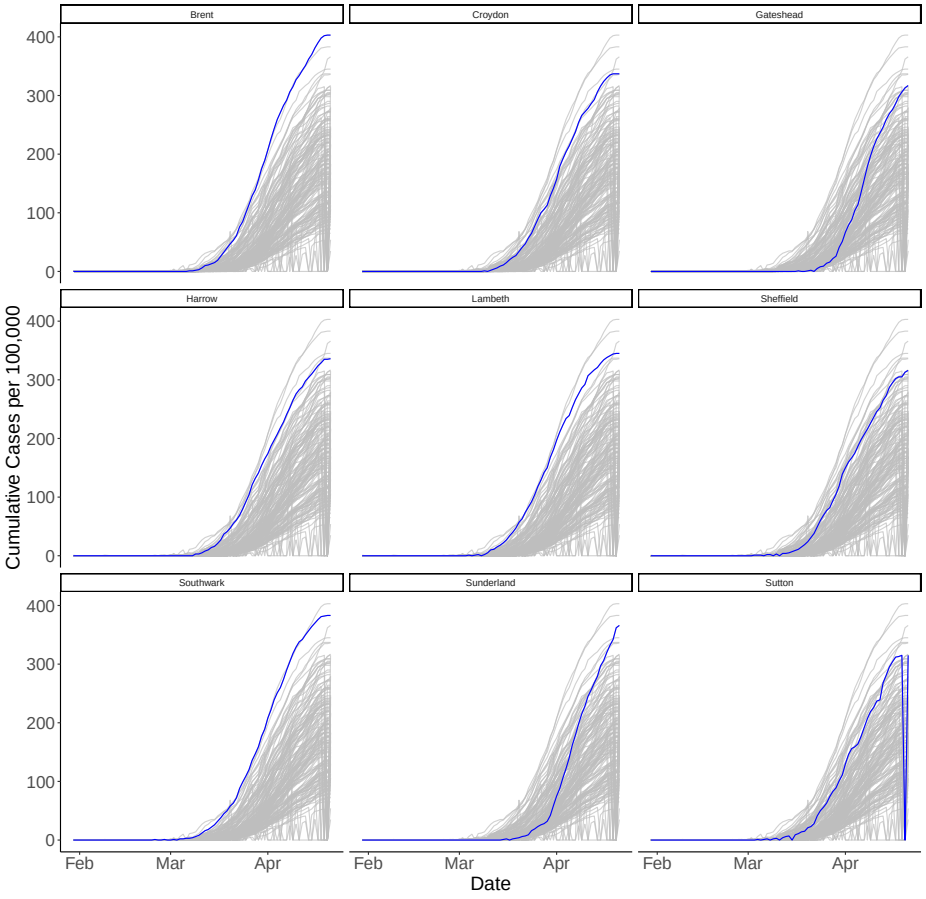
The series of maps reveal a stable pattern of low reported cases from calendar weeks 5 to 11. From week 12 a number of hot spots emerged, notably in London, Birmingham, Cumbria and subsequently around Liverpool. The intensity of new

cases seem to have started to decline from week 15; yet, it is important to note that week 16 display reported cases for only two days.

11.5.1.2 Time-Series Plots

Time-series plots can be used to capture a different dimension of the process in analysis. They can be used to better understand changes in an observation location, an aggregation of observations, or multiple locations simultaneously over time. We plot the cumulative number of COVID-19 cases per 100,000 people for UTLAs reporting over 310 cases. The plots identify the UTLAs in London, Newcastle and Sheffield reporting the largest numbers of COVID-19 cases. The plots also reveal that there has been a steady increase in the number of cases, with some differences. While cases have steadily increase in Brent and Southwark since mid March, the rise has been more sudden in Sunderland. The plots also reveal a possible case of misreporting in Sutton towards the end of the series.

```
tsp <- ggplot(data = covid19_spt,
              mapping = aes(x = date, y = c_covid19_r,
                            group = ctyu19nm))
tsp + geom_line(color = "blue") +
    gghighlight(max(c_covid19_r) > 310, use_direct_label = FALSE) +
    labs(title= paste(" "), x="Date", y="Cumulative Cases per 100,000") +
    theme_classic() +
    theme(plot.title=element_text(size = 20)) +
    theme(axis.text=element_text(size=16)) +
    theme(axis.title.y = element_text(size = 18)) +
    theme(axis.title.x = element_text(size = 18)) +
    theme(plot.subtitle=element_text(size = 16)) +
    theme(axis.title=element_text(size=20, face="plain")) +
    facet_wrap(~ ctyu19nm)
```

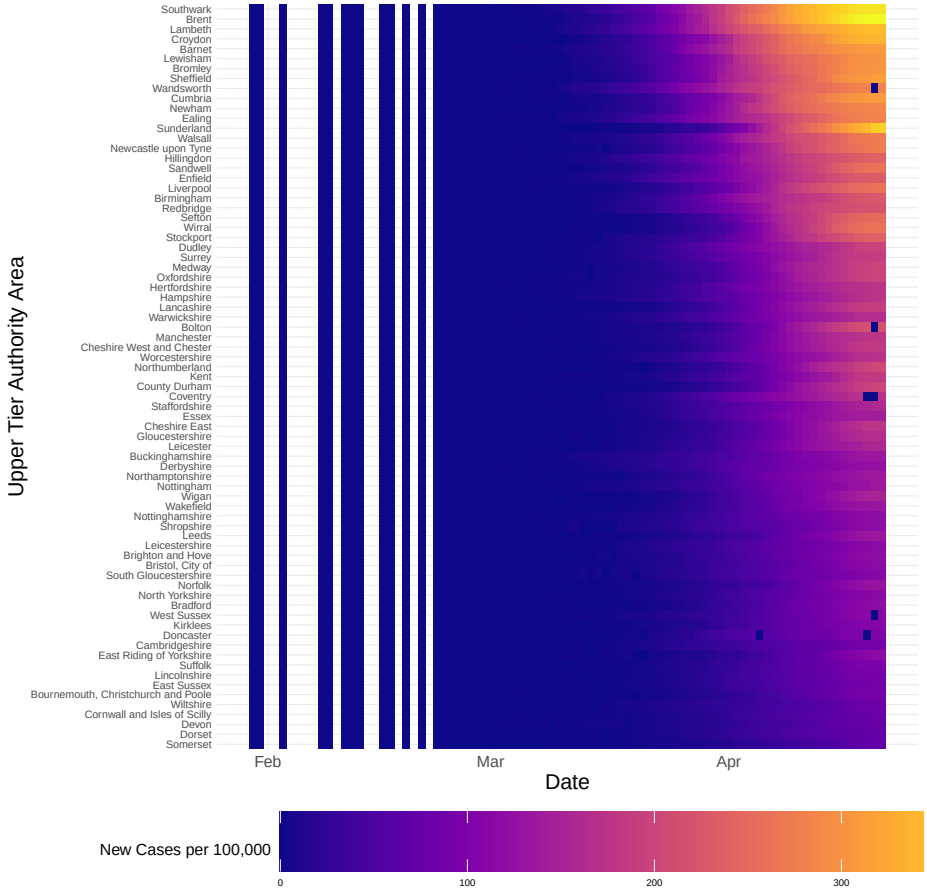


11.5.1.3 Hovmöller Plots

An alternative visualisation is a Hovmöller plot - sometimes known as heatmap. It is a two-dimensional space-time representation in which space is collapsed onto one dimension against time. Hovmöller plots can easily be generated if the data are arranged on a space-time grid; however, this is rarely the case. Luckily we have `ggplot!` which can do magic rearranging the data as needed. Below we produce a Hovmöller plot for UTLAs with resident populations over 260,000. The plot makes clear that the critical period of COVID-19 spread has been during April despite the implementation of a series of social distancing measures by the government.

```
ggplot(data = dplyr::filter(covid19_spt, Residents > 260000),
           mapping = aes(x= date, y= reorder(ctyu19nm, c_covid19_r), fill= c_covid19_r)) +
  geom_tile() +
  scale_fill_viridis(name="New Cases per 100,000", option ="plasma", begin = 0, end = 1, directio
  theme_minimal() +
```

```
  labs(title= paste(" "), x="Date", y="Upper Tier Authority Area") +
  theme(legend.position = "bottom") +
  theme(legend.title = element_text(size=15)) +
  theme(axis.text.y = element_text(size=10)) +
  theme(axis.text.x = element_text(size=15)) +
  theme(axis.title=element_text(size=20, face="plain")) +
  theme(legend.key.width = unit(5, "cm"), legend.key.height = unit(2, "cm"))
```



11.5.1.4 Interactive Plots

Interactive visualisations comprise very effective ways to understand spatio-temporal data and they are now fairly accessible. Interactive visualisations allow for a more data-immersive experience, and enable exploration of the data without having to resort to scripting. Here is when the use of `tmap` shines as it does not only enables easily creating nice static maps but also interactive maps! Below an interactive map for a time snapshot of the data (i.e. `2020-04-14`)

is produced, but with a bit of work layers can be added to display multiple temporal slices of the data.

```
# map
legend_title = expression("Cumulative Cases per 100,000 Population")
imap = tm_shape(dplyr::filter(covid19_spt[,c("ctyu19nm", "date", "c_covid19_r")], as.character(da
  tm_fill("c_covid19_r", title = legend_title, palette = magma(256), style ="cont", legend.is.por
  tm_borders(col = "white") +
  #tm_text("ctyu19nm", size = .4) +
  tm_layout(bg.color = "white", # change background colour
            legend.outside = TRUE, # legend outside
            legend.title.size = 1,
            legend.width = 1)
```

To view the map on your local machines, execute the code chunk below removing the # sign.

```
#tmap_mode("view")
#imap
```

Alternative data visualisation tools are animations, telliscope and shiny. Animations can be constructed by plotting spatial data frame-by-frame, and then stringing them together in sequence. A useful R packages `gganimate` and `tmap`! See Lovelace et al. (2020). Note that the creation of animations may require external dependencies; hence, they have been included here. Both `telliscope` and `shiny` are useful ways for visualising large spatio-temporal data sets in an interactive ways. Some effort is required to deploy these tools.

11.5.2 Exploratory Analysis

In addition to visualising data, we often want to obtain numerical summaries of the data. Again, innovative ways to reduce the inherent dimensionality of the data and examine dependence structures and potential relationships in time and space are needed. We consider visualisations of empirical spatial and temporal means, dependence structures and some basic time-series analysis.

11.5.2.1 Means

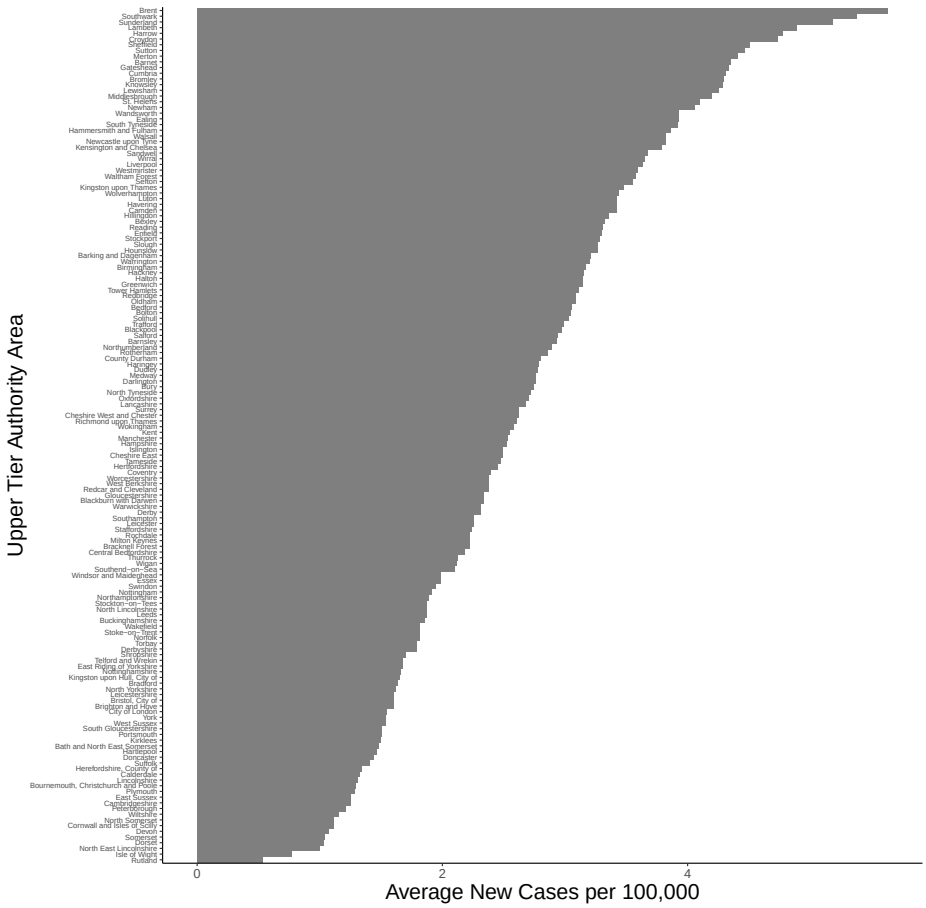
Empirical Spatial Mean

The empirical spatial mean for a data set can be obtained by averaging over time points for one location. In our case, we can compute the empirical spatial mean by averaging the daily rate of new COVID-19 cases for UTLAs between January 30th and April 21st. It reveals that Brent, Southwark and Sunderland report an average daily infection rate of over 5 new cases per 100,000 people, whereas Rutland and Isle of Wight display an average of less than 1.

```
# compute empirical spatial mean
sp_av <- covid19_spt %>% group_by(ctyu19nm) %>% # group by spatial unit
  summarise(sp_mu_emp = mean(n_covid19_r))
```

```
## `summarise()` ungrouping output (override with `.groups` argument)
```

```
# plot empirical spatial mean
ggplot(data=sp_av) +
  geom_col( aes( y = reorder(ctyu19nm, sp_mu_emp), x = sp_mu_emp) , fill = "grey50") +
  theme_classic() +
  labs(title= paste(" "), x="Average New Cases per 100,000", y="Upper Tier Authority Ar
  theme(legend.position = "bottom") +
  theme(axis.text.y = element_text(size=7)) +
  theme(axis.text.x = element_text(size=12)) +
  theme(axis.title=element_text(size=20, face="plain"))
```

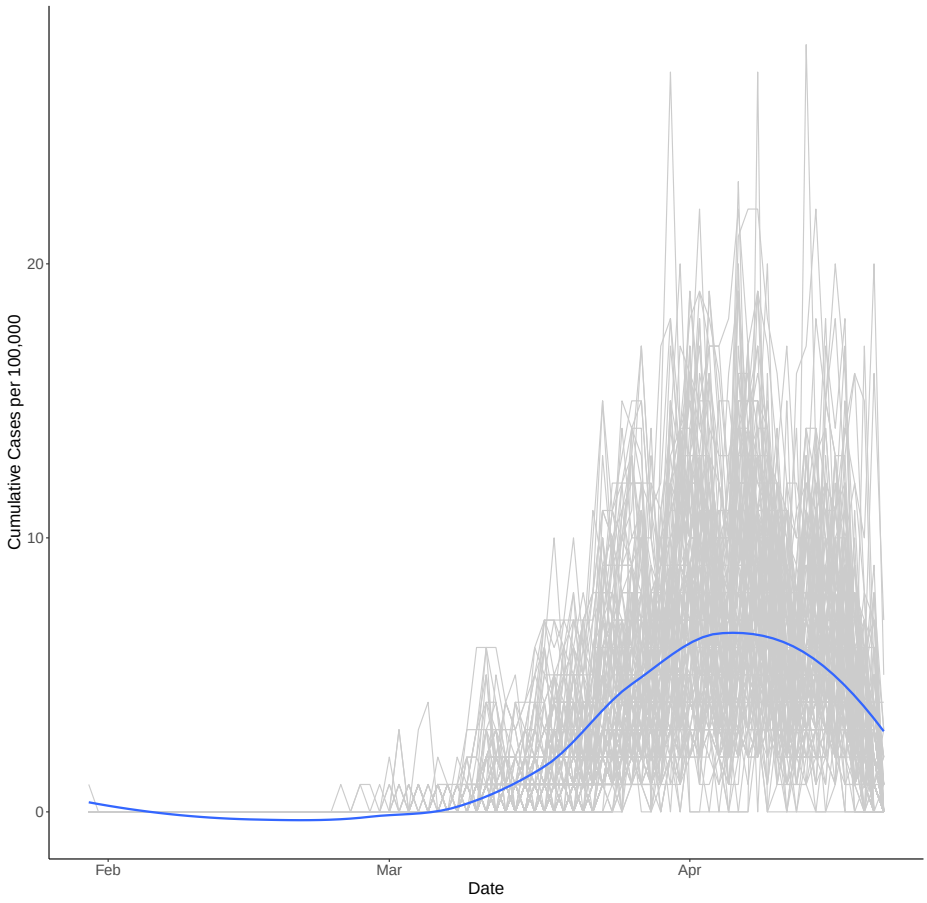


Empirical Temporal Mean

The empirical temporal mean for a data set can be obtained by averaging across spatial locations for a time point. In our case, we can compute the empirical temporal mean by averaging the rate of new COVID-19 cases over UTLAs by day. The empirical temporal mean is plotted below revealing a peak of 8.32 number of new cases per 100,000 people the 7th of April, steadily decreasing to 0.35 for the last reporting observation in our data; that is, April 21st.

Note the empirical temporal mean is smoothed via local polynomial regression fitting; hence below zero values are reported between February and March.

```
# compute temporal mean
tm_av <- covid19 %>% group_by(date) %>%
  summarise(tm_mu_emp = mean(n_covid19_r))

# plot temporal mean + trends for all spatial units
ggplot() +
  geom_line(data = covid19, mapping = aes(x =date, y = n_covid19_r,
                          group = Area.name), color = "gray80") +
   theme_classic() +
  geom_smooth(data = tm_av, mapping = aes(x =date, y = tm_mu_emp),
              alpha = 0.5,
              se = FALSE) +
    labs(title= paste(" "), x="Date", y="Cumulative Cases per 100,000") +
    theme_classic() +
    theme(plot.title=element_text(size = 18)) +
    theme(axis.text=element_text(size=14)) +
    theme(axis.title.y = element_text(size = 16)) +
    theme(axis.title.x = element_text(size = 16)) +
    theme(plot.subtitle=element_text(size = 16)) +
    theme(axis.title=element_text(size=18, face="plain"))
```



11.5.2.2 Dependence

Spatial Dependence

As we know spatial dependence refers to the spatial relationship of a variable's values for a pairs of locations at a certain distance apart, so that are more similar (or less similar) than expected for randomly associated pairs of observations. Patterns of spatial dependence may change over time. In the case of a disease outbreak patterns of spatial dependence can change very quickly as new cases emerge and social distancing measures are implemented. Chapter 6 illustrates how to measure spatial dependence in the context of spatial data.

> Challenge 1: Measure how spatial dependence change over time. Hint: compute the Moran's I on the rate of new COVID-19 cases (i.e. `n_covid19_r` in the `covid19` data frame) at multiple time points.
>
> Note: recall that the problem of ignoring the dependence in the er-

rors when doing OLS regression is that the resulting standard errors and prediction standard errors are inappropriate. In the case of positive dependence, which is the most common case in spatio-temporal data (recall Tobler's law), the standard errors and prediction standard errors are underestimated. This is if dependence is ignored, resulting in a false sense of how good the estimates and predictions really are.

Temporal Dependence

As for spatial data, dependence can also exists in temporal data. Temporal dependence or temporal autocorrelation exists when a variable's value at time t is dependent on its value(s) at $t-1$. More recent observations are often expected to have a greater influence on present observations. A key difference between temporal and spatial dependence is that temporal dependence is unidirectional (i.e. past observations can only affect present or future observations but not inversely), while spatial dependence is multidirectional (i.e. an observation in a spatial unit can influence and be influenced by observations in multiple spatial units).

Before measuring the temporal dependence is our time-series, a time-series object needs to be created with a time stamp and given cycle frequency. A cycle frequency refers to when a seasonal pattern is repeated. We consider a time series of the total number of new COVID-19 cases per 100,000 (i.e. we sum cases over UTLAs by day) and the frequency set to 7 to reflect weekly cycles. So we end up with a data frame of length 71.

```
# create a time series object
total_cnt <- covid19 %>% group_by(date) %>%
  summarise(new_cases = sum(n_covid19_r))
```

```
## `summarise()` ungrouping output (override with `.groups` argument)
```

```
total_cases_ts <- ts(total_cnt$new_cases,
                     start = 1,
                     frequency =7)
```

There are various ways to test for temporal autocorrelation. An easy way is to compute the correlation coefficient between a time series measured at time t and its lag measured at time $t-1$. Below we measure the temporal autocorrelation in the rate of new COVID-19 cases per 100,000 people. A correlation of 0.97 is returned indicating high positive autocorrelation; that is, high (low) past numbers of new COVID-19 cases per 100,000 people tend to correlate with higher (lower) present numbers of new COVID-19 cases. The Durbin-Watson test is often used to test for autocorrelation in regression models.

```
# create lag term t-1
lag_new_cases <- total_cnt$new_cases[-1]
total_cnt <- cbind(total_cnt[1:70,], lag_new_cases)
```

```
cor(total_cnt[,2:3])
```

```
##               new_cases lag_new_cases
## new_cases      1.000000      0.974284
## lag_new_cases  0.974284      1.000000
```

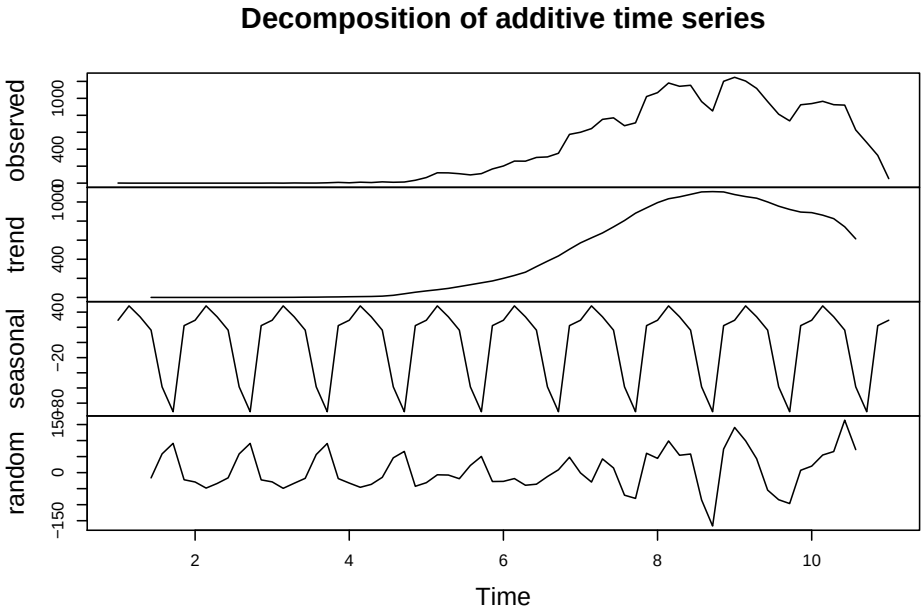
Time Series Components

In addition to temporal autocorrelation, critical to the analysis of time-series are its constituent components. A time-series is generally defined by three key components:

- Trend: A trend exists when there is a long-term increase or decrease in the data.
- Seasonal: A seasonal pattern exists when a time series is affected by seasonal factors and is of a fixed and known frequency. Seasonal cycles can occur at various time intervals such as the time of the day or the time of the year.
- Cyclic (random): A cycle exists when the data exhibit rises and falls that are not of a fixed frequency.

To understand and model a time series, these components need to be identified and appropriately incorporated into a regression model. We illustrate these components by decomposing our time series for total COVID-19 cases below. The top plot shows the observed data. Subsequent plots display the trend, seasonal and random components of the total number of COVID-19 cases on a weekly periodicity. They reveal a clear inverted U-shape trend and seasonal pattern. This idea that we can decompose data to extract information and understand temporal processes is key to understand the concept of basis functions to model spatio-temporal data, which will be introduced in the next section.

```
# decompose time series
dec_ts <- decompose(total_cases_ts)
# plot time series components
plot(dec_ts)
```



For a good introduction to time-series analysis in R, refer to Hyndman and Athanasopoulos (2018) and DataCamp.

11.6 Spatio-Temporal Data Modelling

Having some understanding of the spatio-temporal patterns of COVID-19 spread through data exploration, we are ready to start further examining structural relationships between the rate of new infections and local contextual factors via regression modelling across UTLAs. Specifically we consider the number of new cases per 100,000 people to capture the rate of new infections and only one contextual factor; that is, the share of population suffering from long-term sickness or disabled. We will consider some basic statistical models, of the form of linear regression and generalized linear models, to account for spatio-temporal dependencies in the data. Note that we do not consider more complex structures based on hierarchical models or spatio-temporal weighted regression models which would be the natural step moving forward.

As any modelling approach, spatio-temporal statistical modelling has three principal goals:

1. predicting values of a given outcome variable at some location in space within the time span of the observations and offering information about the uncertainty of those predictions;

2. performing statistical inference about the influence of predictors on an outcome variable in the presence of spatio-temporal dependence; and,

3. forecasting future values of an outcome variable at some location, offering information about the uncertainty of the forecast.

11.6.1 Intuition

The key idea on what follows is to use a basic statistical regression model to understand the relationship between the share of new COVID-19 infections and the share of population suffering from long-term illness, accounting for spatio-temporal dependencies. We will consider what is known as a trend-surface regression model which assumes that spatio-temporal dependencies can be accounted for by "trend" components and incorporate as predictors in the model. Formally we consider the regression model below which seeks to account for spatial and temporal trends.

$$y(s_i, t_j) = \beta_0 + \beta_k x(s_i, t_j) + e(s_i, t_j)$$

where β_0 is the intercept and β_k represents a set of regression coefficients associated with $x(s_i, t_j)$; the k indicates the number of covariates at spatial location s_i and time t_j; e represents the regression errors which are assumed to follow a normal distribution. The key difference to aproaches considered in previous chapters is the incorporation of space and time together. As we learnt from the previous section, this has implications are we now have two sources of dependence: spatial and temporal autocorrelation, as well as seasonal and trend components. This has implications for modelling as we now need to account for all of these components if we are to establish any relationship between y and x.

A key implication is how we consider the set of covariates represented by x. Three key types can be identified:

- spatial-variant, temporal-invariant covariates: these are attributes which may vary across space but be temporally invariant, such as geographical distances;
- spatial-invariant, temporal-variant covariates: these are attributes which do not vary across space but change over time; and,
- spatial-variant, temporal-variant covariates: these are attributes which vary over both space and time;

  Note that what is variant or invariant will depend on the spatial and temporal scale of the analysis.

We can also consider spatio-temporal "basis functions". Note that this is an important concept for the rest of the Chapter. What are basis functions then? If you think that spatio-temporal data represent a complex set of curves or surfaces in space, basis functions represent the components into which this set of curves can be decomposed. In this sense, basis functions operate in a similar fashion as the decomposition of time series considered above i.e. time series data

can be decomposed into a trend, seasonal and random components and their sum can be used to represent the observed temporal trajectory. Basis functions offer an effective way to incorporate spatio-temporal dependencies. Thus, basis functions have the key goal of accounting for spatio-temporal dependencies as spatial weight matrices or temporal lags help accounting spatial autocorrelation in spatial models and temporal autocorrelation in time series analysis.

As standard regression coefficients, basis functions are related to y via coefficients (or weights). The difference is that we typically assume that basis functions are known while coefficients are random. Examples of basis functions include polynomials, splines, wavelets, sines and cosines so various linear combinations that can be used to infer potential spatio-temporal dependencies in the data. This is similar to deep learning models in which cases you provide, for example, an image and the model provides a classification of pixels. But you normally do not know what the classification represents (hence they are known as black boxes!) so further analysis on the classification is needed to understand what the model has attempted to capture. Basically basis functions are smoother functions to represent the observed data, and their objective to capture the spatial and temporal variability in the data as well as their dependence.

For our application, we start by considering a basic OLS regression model with the following basis functions to account spatial-temporal structures:

- overall mean;
- linear in lon-coordinate;
- linear in lat-coordinate;
- linear time daily trend;
- additional spatio-temporal basis functions which are presented below; and,

These basis functions are incorporated as independent variables in the regression model. Additionally, we also include the share of population suffering from long-term illness as we know it is highly correlated to the cumulative number of COVID-19 cases. The share of population suffering long-term illness is incorporated as a spatial-variant, temporal-invariant covariates given that rely in 2011 census data.

11.6.2 Fitting Spatio-Temporal Models

As indicated at the start of this Chapter, we use the FRK framework developed by Cressie and Johannesson (2008). It provides a scalable, relies on the use a spatial random effects model (with which we have some familiarity) and can be easily implemented in R by the use of the `FRK` package (Zammit-Mangion and Cressie, 2017). In this framework, a spatially correlated errors can be decomposed using a linear combination of spatial basis functions, effectively addressing issues of spatial-temporal dependence and nonstationarity. The specification of spatio-temporal basis functions is a key component of the model and they can be generated automatically or by the user via the `FRK` package. We will use

the automatically generated functions. While as we will notice they are difficult to interpret, user generated functions require greater understanding of the spatio-temporal structure of COVID-19 which is beyond the scope of this Chapter.

Prepare Data

The first step to create a data frame with the variables that we will consider for the analysis. We first remove the geometries to convert `covid19_spt` from a simple feature object to a data frame and then compute the share of long-term illness population.

```
# remove geometries
st_geometry(covid19_spt) <- NULL

# share of population in long-term illness
covid19_spt <- covid19_spt %>% mutate(
 lt_illness = Longterm_sick_or_disabled / Residents
)
```

Construct Basis Functions

We now build the set of basis functions. The can be constructed by using the function `auto_basis` from the `FRK` package. The function takes as arguments: data, nres (which is the number of "resolutions" or aggregation to construct); and type of basis function to use. We consider a single resolution of the default Gaussian radial basis function.

```
# build basis functions
G <- auto_basis(data = covid19_spt[,c("long","lat")] %>%
                       SpatialPoints(),           # To sp obj
                nres = 1,                         # One resolution
                type = "Gaussian")                # Gaussian BFs
# basis functions evaluated at data locations are then the covariates
S <- eval_basis(basis = G,                       # basis functions
                s = covid19_spt[,c("long","lat")] %>%
                     as.matrix()) %>%            # conv. to matrix
     as.matrix()                                 # conv. to matrix
colnames(S) <- paste0("B", 1:ncol(S)) # assign column names
```

Add Basis Functions to Data Frame

We then prepare a data frame for the regression model, adding the weights extracted from the basis functions. These weights enter as covariates in our model. Note that the resulting number of basis functions is nine. Explore by executing `colnames(S)`. Below we select only relevant variables for our model.

```
# selecting variables
reg_df <- cbind(covid19_spt, S) %>%
  dplyr::select(ctyu19nm, c_covid19_r, long, lat, day, lt_illness, B1:B9)
```

Fit Linear Regression

We now fit a linear model using `lm` including as covariates longitude, latitude, day, share of long-term ill population and the nine basis functions.

> Recall that latitude refers to north/south from the equator and longitude refers to west/east from Greenwich. Further up north means a higher latitude score. Further west means higher longitude score. Scores for Liverpool (53.4084° N, 2.9916° W) are thus higher than for London (51.5074° N, 0.1278° W). This will be helpful for interpretation.

```
eq1 <- c_covid19_r ~ long + lat + day + lt_illness + .
lm_m <- lm(formula = eq1,
           data = dplyr::select(reg_df, -ctyu19nm))
lm_m %>% summary()
```

```
##
## Call:
## lm(formula = eq1, data = dplyr::select(reg_df, -ctyu19nm))
##
## Residuals:
##    Min     1Q Median     3Q    Max
## -87.55 -48.31 -27.51  27.68 338.54
##
## Coefficients:
##               Estimate Std. Error t value Pr(>|t|)
## (Intercept) -3.772e+03  4.218e+02  -8.943  < 2e-16 ***
## long        -3.738e+01  9.054e+00  -4.128 3.68e-05 ***
## lat          6.652e+01  8.169e+00   8.144 4.27e-16 ***
## day         -6.662e-01  7.993e-02  -8.335  < 2e-16 ***
## lt_illness   7.094e+02  8.857e+01   8.009 1.28e-15 ***
## B1           2.367e+02  7.896e+01   2.997  0.00273 **
## B2           4.438e+01  3.525e+01   1.259  0.20802
## B3           4.002e+02  4.947e+01   8.090 6.61e-16 ***
## B4          -6.687e+01  6.858e+01  -0.975  0.32960
## B5           6.585e+01  5.585e+01   1.179  0.23835
## B6           8.324e+00  6.249e+01   0.133  0.89403
## B7           4.503e+01  9.222e+01   0.488  0.62534
## B8          -6.151e+00  7.006e+01  -0.088  0.93005
## B9           3.862e+01  6.359e+01   0.607  0.54368
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 71.08 on 10636 degrees of freedom
## Multiple R-squared:  0.05544,    Adjusted R-squared:  0.05429
## F-statistic: 48.02 on 13 and 10636 DF,  p-value: < 2.2e-16
```

Coefficients for explicitly specified spatial and temporal variables and the share of long-term ill population are all statistically significant. The interpretation of the regression coefficients is as usual; that is, one unit increase in a covariate relates to one unit increase in the dependent variable. For instance, the coefficient for long-term illness population indicates that UTLAs with a larger share of long-term ill population in one percentage point tend to have 709 more new COVID-19 cases per 100,000 people! on average. The coefficient for day reveals a strong negative temporal dependence with smaller number of new cases per 100,000 people as we move over time. The coefficient for latutide indicates as we move north the number of new COVID-19 cases per 100,000 people tends to be higher but lower if we move west.

While overall the model provides some understanding of the spatio-temporal structure of the spread of COVID-19, the overall fit of the model is relatively poor. The R^2 reveals that the model explains only 5% of the variability of the spread of COVID-19 cases. Also, except for one, the coefficients associated to the basis functions are statistically insignificant. A key issue that we have ignored so far is the fact that our dependent variable is a count and is highly skewed - refer back to Section [8.4 Exploratory Analysis].

> Challenge 2: Explore a model with only spatial components (i.e. `long` and `lat`) or only temporal components (`day`). What model returns the largest R^2?

Poisson Regression

A Poisson regression model provides a more appropriate framework to address these issues. We do this fitting a general linear model (or GLM) specifying the family function to be a Poisson.

```
# estimate a poisson model
poisson_m1 <- glm(eq1,
                  family = poisson("log"), # Poisson + log link
                  data = dplyr::select(reg_df, -ctyu19nm))
poisson_m1 %>% summary()
```

```
##
## Call:
## glm(formula = eq1, family = poisson("log"), data = dplyr::select(reg_df,
##     -ctyu19nm))
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -14.457   -9.386   -6.380    3.995   29.573
##
## Coefficients:
##               Estimate Std. Error z value Pr(>|z|)
## (Intercept) -9.726e+01  1.010e+00 -96.313  < 2e-16 ***
```

```
## long        -9.789e-01  2.242e-02 -43.666  < 2e-16 ***
## lat          1.767e+00  1.902e-02  92.869  < 2e-16 ***
## day         -1.441e-02  1.664e-04 -86.581  < 2e-16 ***
## lt_illness   1.442e+01  1.851e-01  77.923  < 2e-16 ***
## B1           6.198e+00  2.017e-01  30.719  < 2e-16 ***
## B2           3.616e-01  7.889e-02   4.583 4.59e-06 ***
## B3           1.136e+01  1.418e-01  80.147  < 2e-16 ***
## B4          -1.976e+00  1.450e-01 -13.627  < 2e-16 ***
## B5           2.813e+00  1.255e-01  22.414  < 2e-16 ***
## B6          -1.334e+00  1.380e-01  -9.662  < 2e-16 ***
## B7           7.696e-01  2.117e-01   3.635 0.000278 ***
## B8          -7.094e-01  1.546e-01  -4.589 4.46e-06 ***
## B9           1.809e+00  1.616e-01  11.196  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for poisson family taken to be 1)
##
##     Null deviance: 1032743  on 10649  degrees of freedom
## Residual deviance:  959940  on 10636  degrees of freedom
## AIC: 993314
##
## Number of Fisher Scoring iterations: 6
```

The model seems to provide a better fit to the data as the median of deviance residuals (-6.3) is smaller than for the linear regression model (-27.51). And, all coefficients are positive and statistically significant. Yet, the Poisson model assumes that the mean and variance of the COVID-19 cases is the same. But, given the distribution of our dependent variable, its variance is likely to be greater than the mean. That means the data exhibit "overdispersion". How do we know this? An estimate of the dispersion is given by the ratio of the deviance to the total degrees of freedom (the number of data points minus the number of covariates). In this case the dispersion estimate is:

```
poisson_m1$deviance / poisson_m1$df.residual
```

```
## [1] 90.25383
```

which is clearly greater than 1! i.e. the data are overdispersed.

Quasipoisson Regression

An approach to account for overdispersion is to use quasipoisson when calling `glm`. The quasi-Poisson model assumes that the variance is proportional to the mean, and that the constant of the proportionality is the over-dispersion parameter.

```
# estimate a quasipoisson model
qpoisson_m1 <- glm(eq1,
```

```
                family = quasipoisson("log"), # QuasiPoisson + log link
                data = dplyr::select(reg_df, -ctyu19nm))
qpoisson_m1 %>% summary()
```

```
##
## Call:
## glm(formula = eq1, family = quasipoisson("log"), data = dplyr::select(reg_df,
##     -ctyu19nm))
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -14.457   -9.386   -6.380    3.995   29.573
##
## Coefficients:
##               Estimate Std. Error t value Pr(>|t|)
## (Intercept) -97.261261  10.069209  -9.659  < 2e-16 ***
## long         -0.978924   0.223534  -4.379 1.20e-05 ***
## lat           1.766675   0.189682   9.314  < 2e-16 ***
## day          -0.014405   0.001659  -8.683  < 2e-16 ***
## lt_illness   14.420396   1.845247   7.815 6.02e-15 ***
## B1            6.197505   2.011637   3.081  0.00207 **
## B2            0.361556   0.786651   0.460  0.64580
## B3           11.363202   1.413688   8.038 1.01e-15 ***
## B4           -1.975530   1.445504  -1.367  0.17176
## B5            2.813267   1.251510   2.248  0.02460 *
## B6           -1.333746   1.376430  -0.969  0.33257
## B7            0.769599   2.110944   0.365  0.71544
## B8           -0.709375   1.541475  -0.460  0.64539
## B9            1.809393   1.611435   1.123  0.26153
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for quasipoisson family taken to be 99.42153)
##
##     Null deviance: 1032743  on 10649  degrees of freedom
## Residual deviance:  959940  on 10636  degrees of freedom
## AIC: NA
##
## Number of Fisher Scoring iterations: 6
```

Negative Binomial Regression

The model output indicates major improvement in terms of model fit as the residual deviance (959940) and median of deviance residuals (-6.380) remain unchanged. An alternative approach is a Negative Binomial Model (NBM). This models relaxes the assumption of equality between the mean and variance.

We estimate a NBM by using the function `glm.nb` from the `MASS` package.

```
# estimate a negative binomial model
nb_m1 <- glm.nb(eq1,
       data = dplyr::select(reg_df, -ctyu19nm))
nb_m1
```

```
##
## Call:  glm.nb(formula = eq1, data = dplyr::select(reg_df, -ctyu19nm),
##     init.theta = 11051670.9, link = log)
##
## Coefficients:
## (Intercept)         long          lat          day   lt_illness           B1
##   -97.26122     -0.97892      1.76667     -0.01441     14.42040      6.19749
##          B2           B3           B4           B5           B6           B7
##     0.36155     11.36319     -1.97553      2.81326     -1.33374      0.76959
##          B8           B9
##    -0.70938      1.80938
##
## Degrees of Freedom: 10649 Total (i.e. Null);  10636 Residual
## Null Deviance:       1033000
## Residual Deviance: 959900    AIC: 993300
```

Including Interactions

Similarly the model output does not suggest any major improvement in explaining the spatio-temporal variability in the spread of COVID-19. We may need a different strategy then. Let's try running a NBM including interaction terms between spatial and temporal terms (i.e. longitude, latitude and day). We can do this by estimating the following model `c_covid19_r ~ (long + lat + day)^2 + lt_illness + .`

```
# new model specification
eq2 <- c_covid19_r ~ (long + lat + day)^2 + lt_illness + .
# estimate a negative binomial model
nb_m2 <- glm.nb(eq2,
       data = dplyr::select(reg_df, -ctyu19nm))
nb_m2
```

```
##
## Call:  glm.nb(formula = eq2, data = dplyr::select(reg_df, -ctyu19nm),
##     init.theta = 300465.7453, link = log)
##
## Coefficients:
## (Intercept)         long          lat          day   lt_illness           B1
##  -2.642e+02   -7.873e+01    4.936e+00    6.423e-02    1.335e+01    1.145e+01
##          B2           B3           B4           B5           B6           B7
##  -7.235e-01    2.132e+01   -6.598e+00    9.543e+00   -1.075e+01    2.157e+01
```

```
##          B8          B9     long:lat     long:day      lat:day
##  -5.503e+00   9.127e-01   1.533e+00   8.504e-05  -1.499e-03
##
## Degrees of Freedom: 10649 Total (i.e. Null);  10633 Residual
## Null Deviance:       1033000
## Residual Deviance: 953600    AIC: 987000
```

This model leads to a better model by returning a slight reduction in the residual deviance and AIC score. Interestingly it also returns highly statisticaly significant coeficients for the interaction terms between longitude and latitude (`long:lat`) and latitude and day (`lat:day`). The former indicates that as we move one degree north and west, the number of new cases tend to increase in 2 cases. The latter indicates UTLAs on the west of England tend to report a lower number of cases as time passes.

You can report the output for all models estimated above by executing (after removing `#`):

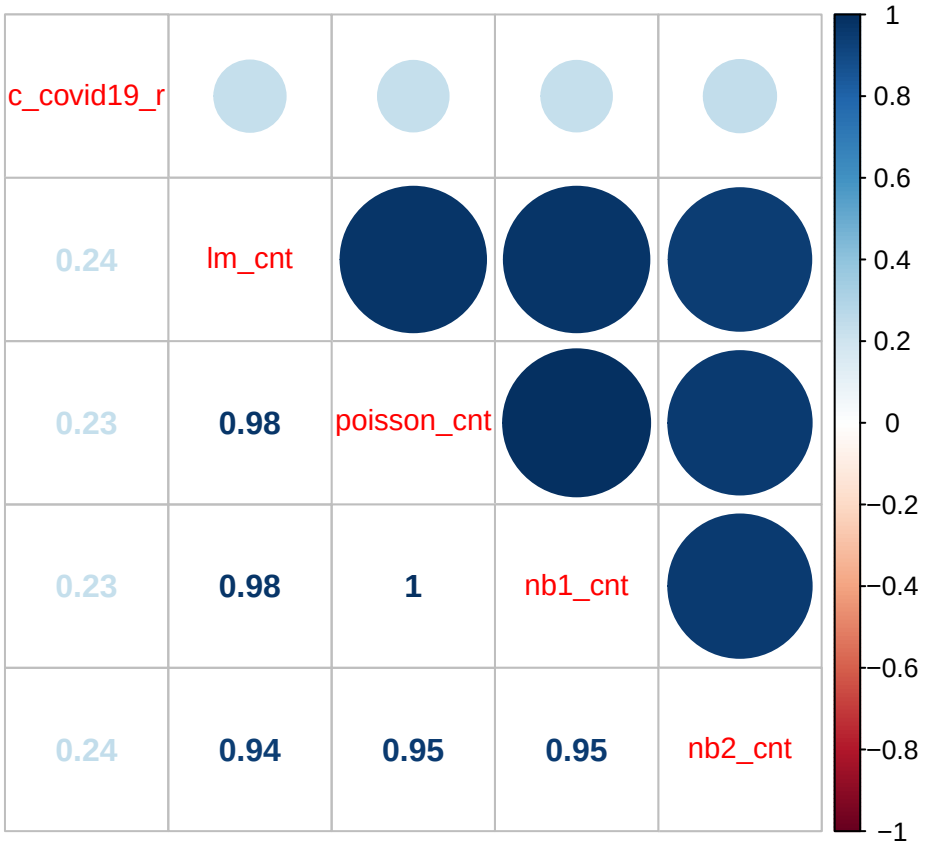
```
# export_summs(lm_m, poisson_m, qpoisson_m1, nb_m1, nb_m2)
```

11.6.2.1 Model Comparision

To compare regression models based on different specifications and assumptions, like those reported above, you may want to consider the cross-validation approach used in Chapter 4 Flows. An easy approach if you would like to get a quick sense of model fit, you can look at the correlation between observed and predicted values of the dependent variable. For our models, we can achieve this by executing:

```
# computing predictions for all models
lm_cnt <- predict(lm_m)
poisson_cnt <- predict(poisson_m1)
nb1_cnt <- predict(nb_m1)
nb2_cnt <- predict(nb_m2)
reg_df <- cbind(reg_df, lm_cnt, poisson_cnt, nb1_cnt, nb2_cnt)

# computing correlation coefficients
cormat <- cor(reg_df[, c("c_covid19_r", "lm_cnt", "poisson_cnt", "nb1_cnt", "nb2_cnt")]
              use="complete.obs",
              method="pearson")

# significance test
sig1 <- corrplot::cor.mtest(reg_df[, c("c_covid19_r", "lm_cnt", "poisson_cnt", "nb1_cnt
                            conf.level = .95)

# create a correlogram
corrplot::corrplot.mixed(cormat,
                         number.cex = 1,
```

```
                    tl.pos = "d",
                    tl.cex = 0.9)
```



None of the models does a great job at predicting the observed count of new COVID-19 cases. They display correlation coefficients between 0.23 and 0.24 and high degree of correlation between them. Part of the assignment will be finding ways to improve this initial models. They should just be considered as a starting point.

> Challenge 3: Find ways to achive better model fit. Hint: There are a potentially few easy ways to make some considerable improvement. *One option* is to remove all zeros from the dependent variable `c_covid19_r`. They are likely to be affecting the ability of the model to predict positive values which are of main interest if we want to understand the spatio-temporal patterns of the outbreak. *A second idea* is to remove all zeros from the dependent variable and additionally use its log for the regression model. *A third idea* is to include more explanatory variables. Look for factors which can explain the spatial-temporal variability of the current COVID-19

outbreak. Look for hypotheses / anecdotal evidence from the newspapers and social media. *A fourth idea* is to check for collinearity. Collinearity is likely to be an issue given the way basis functions are created. Checking for collinearity of course will not improve the fit of the existing model but it is important to remove collinear terms if statistical inference is a key goal - which in this case is. Over to you now!

Bibliography

Anselin, L. (1988). *Spatial econometrics: methods and models*, volume 4. Springer Science & Business Media.

Anselin, L. (2003). Spatial externalities, spatial multipliers, and spatial econometrics. *International regional science review*, 26(2):153–166.

Anselin, L. (2007). Spatial regression analysis in r–a workbook.

Anselin, L. and Rey, S. J. (2014). *Modern spatial econometrics in practice: a guide to GeoDa, GeoDaSpace and PySAL*. GeoDa Press LLC.

Arribas-Bel, D. (2014). Spatial data, analysis, and regression-a mini course. *REGION*, 1(1):R1.

Arribas-Bel, D. (2019). Geographic data science'19.

Banerjee, S., Carlin, B. P., and Gelfand, A. E. (2014). *Hierarchical modeling and analysis for spatial data*. Crc Press.

Belsley, D. A., Kuh, E., and Welsch, R. E. (2005). *Regression diagnostics: Identifying influential data and sources of collinearity*, volume 571. John Wiley & Sons.

Bivand, R. S., Pebesma, E., and Gómez-Rubio, V. (2013). *Applied Spatial Data Analysis with R*. Springer New York.

Brunsdon, C. and Comber, L. (2015). *An introduction to R for spatial analysis & mapping*. Sage.

Brunsdon, C., Fotheringham, S., and Charlton, M. (1998). Geographically weighted regression. *Journal of the Royal Statistical Society: Series D (The Statistician)*, 47(3):431–443.

Comber, A., Brunsdon, C., Charlton, M., Dong, G., Harris, R., Lu, B., Lü, Y., Murakami, D., Nakaya, T., Wang, Y., et al. (2020). The gwr route map: a guide to the informed application of geographically weighted regression. *arXiv preprint arXiv:2004.06070*.

Cressie, N. (2015). *Statistics for spatial data*. John Wiley & Sons.

Cressie, N. and Johannesson, G. (2008). Fixed rank kriging for very large spatial data sets. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 70(1):209–226.

for Multilevel Modelling, C. (n.d.). Introduction to multilevel modelling.

Fotheringham, S., Brunsdon, C., and Charlton, M. (2002). *Geographically Weighted Regression*. John Wiley & Sons.

Gabadinho, A., Ritschard, G., Studer, M., and Müller, N. S. (2009). Mining sequence data in r with the traminer package: A user's guide. *Geneva: Department of Econometrics and Laboratory of Demography, University of Geneva*.

Gelman, A. and Hill, J. (2006a). *Data analysis using regression and multilevel/hierarchical models*. Cambridge University Press.

Gelman, A. and Hill, J. (2006b). *Data analysis using regression and multilevel/hierarchical models. Cambridge university press.* Cambridge University Press.

Gibbons, S., Overman, H. G., and Patacchini, E. (2014). Spatial methods.

Grolemund, G. and Wickham, H. (2019). *R for Data Science*. O'Reilly, US.

Hyndman, R. J. and Athanasopoulos, G. (2018). *Forecasting: principles and practice*. OTexts.

Lovelace, R. and Cheshire, J. (2014). Introduction to visualising spatial data in R. *National Centre for Research Methods Working Papers*, 14(03).

Lovelace, R., Nowosad, J., and Muenchow, J. (2020). *An introduction to statistical learning*, volume 112. CRC Press, R Series.

Lu, B., Harris, P., Charlton, M., and Brunsdon, C. (2014). The gwmodel r package: further topics for exploring spatial heterogeneity using geographically weighted models. *Geo-spatial Information Science*, 17(2):85–101.

Pebesma, E. et al. (2012). spacetime: Spatio-temporal data in r. *Journal of statistical software*, 51(7):1–30.

Rowe, F., Patias, N., and Arribas-Bel, D. (2020). Policy brief: Neighbourhood change and trajectories of inequality in britain, 1971-2011.

Singleton, A. (2017). Geographic data science for urban analytics. Online course.

Wheeler, D. and Tiefelsdorf, M. (2005). Multicollinearity and correlation among local regression coefficients in geographically weighted regression. *Journal of Geographical Systems*, 7(2):161–187.

Wikle, C. K., Zammit-Mangion, A., and Cressie, N. (2019). *Spatio-temporal Statistics with R*. CRC Press.

Williamson, P. (2018). Survey analysis.

Xie, Y., Allaire, J., and Grolemund, G. (2019). *R Markdown: The Definitive Guide*. CRC Press, Taylor & Francis, Chapman & Hall Book.

Zammit-Mangion, A. and Cressie, N. (2017). Frk: An r package for spatial and spatio-temporal prediction with large datasets. *arXiv preprint arXiv:1705.08105*.