Spatial Modelling for Data Scientists

Francisco Rowe, Dani Arribas-Bel

2024-02-16

Table of contents

Welcome

This is the website for **Spatial Modeling for Data Scientists 2023/24**. This is a course taught by Professor Francisco Rowe at the University of Liverpool, United Kingdom. The course materials were developed by Professor Francisco Rowe and Professor Dani Arribas-Bel. You will learn how to analyse and model different types of spatial data as well as gaining an understanding of the various challenges arising from manipulating such data.

 A compilation of this web course is hosted as a GitHub repository that you can access:

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

Contact

Francisco Rowe - `F.Rowe-Gonzalez [at] liverpool.ac.uk`
Professor of Population Data Science
Office 507, Roxby Building,
University of Liverpool - 74 Bedford St S,
Liverpool, L69 7ZT,
United Kingdom.

1 Overview

Access to all materials, including lecture notes, computational notebooks and datasets, is centralised through the use of the course website available in the following url:

https://gdsl-ul.github.io/san/

The module handbook, including the assessment description, criteria and module programme, and videos for each teaching week can be accessed via the module Canvas site:

ENS453 Spatial Modelling for Data Scientists

1.1 Aims

This module aims to provides students with a range of techniques for analysing and modelling spatial data:

- build upon the more general research training delivered via companion modules on *Data Collection and Data Analysis*, both of which have an aspatial focus;
- highlight a number of key social issues that have a spatial dimension;
- explain the specific challenges faced when attempting to analyse spatial data;
- introduce a range of analytical techniques and approaches suitable for the analysis of spatial data; and,
- enhance practical skills in using *R* software packages to implement a wide range of spatial analytical tools.

1.2 Learning Outcomes

By the end of the module, students should be able to:

- identify some key sources of spatial data and resources of spatial analysis and modelling tools;
- explain the advantages of taking spatial structure into account when analysing spatial data;
- apply a range of computer-based techniques for the analysis of spatial data, including mapping, correlation, kernel density estimation, regression, multi-level models, geographically-weighted regression, spatial interaction models and spatial econometrics;
- apply appropriate analytical strategies to tackle the key methodological challenges facing spatial analysis – spatial autocorrelation, heterogeneity, and ecological fallacy; and,
- select appropriate analytical tools for analysing specific spatial data sets to address emerging social issues facing the society.

1.3 Feedback

- *Formal assessment of two computational essays.* Written assignment-specific feedback will be provided within three working weeks of the submission deadline. Comments will offer an understanding of the mark awarded and identify areas which can be considered for improvement in future assignments.
- *Verbal face-to-face feedback.* Immediate face-to-face feedback will be provided during lecture, discussion and clinic sessions in interaction with staff. This will take place in all live sessions during the semester.
- *Online forum.* Asynchronous written feedback will be provided via an online forum maintained by the module lead. Students are encouraged to contribute by asking and answering questions relating to the module content. Staff will monitor the forum Monday to Friday 9am-5pm,

but it will be open to students to make contributions at all times.

1.4 Computational Environment

To reproduce the code in the book, you need the following software packages:

- R-4.3.1
- RStudio 2023.09.0+463
- Quarto 1.3.450
- the list of libraries in the next section

To check your version of:

- R and libraries run `sessionInfo()`
- RStudio click `help` on the menu bar and then `About`
- Quarto check the `version` file in the quarto folder on your computer.

To install and update:

- R, download the appropriate version from The Comprehensive R Archive Network (CRAN)
- RStudio, download the appropriate version from Posit
- Quarto, download the appropriate version from the Quarto website

1.4.1 Dependency list

The list of libraries used in this book is provided below:

- `arm`
- `car`
- `corrplot`
- `devtools`
- `FRK`
- `gghighlight`
- `ggplot2`
- `ggmap`

- GISTools
- gridExtra
- gstat
- hexbin
- jtools
- kableExtra
- knitr
- lme4
- lmtest
- lubridate
- MASS
- merTools
- plyr
- RColorBrewer
- rgdal
- sf
- sjPlot
- sp
- spgwr
- spatialreg
- spacetime
- stargazer
- tidyverse
- tmap
- tufte
- viridis
- basemapR

Copy, paste and run the code below in your console. Ensure all packages are installed on your computer.

```
# package names
packages <- c(
    "arm",
    "car",
    "corrplot",
    "devtools",
    "FRK",
    "gghighlight",
    "ggplot2",
    "ggmap",
```

```
    "gridExtra",
    "gstat",
    "hexbin",
    "jtools",
    "kableExtra",
    "knitr",
    "lme4",
    "lmtest",
    "lubridate",
    "MASS",
    "merTools",
    "plyr",
    "RColorBrewer",
    "sf",
    "sjPlot",
    "sp",
    "spgwr",
    "spatialreg",
    "spacetime",
    "stargazer",
    "tidyverse",
    "tmap",
    "tufte",
    "viridis"
)

# install packages not yet installed
installed_packages <- packages %in% rownames(installed.packages())
if (any(installed_packages == FALSE)) {
  install.packages(packages[!installed_packages])
}

# packages loading
invisible(lapply(packages, library, character.only = TRUE))
```

::: column-margin ::: callout-note To install the library `basemapR`, you need to install from source by running:

```
library(devtools)
install_github('Chrisjb/basemapR')
```
::: ::: column-margin

1.5 Assessment

The final module mark is composed of the *two computational essays*. Together they are designed to cover the materials introduced in the entirety of content covered during the semester. A computational essay is an essay whose narrative is supported by code and computational results that are included in the essay itself. Each teaching week, you will be required to address a set of questions relating to the module content covered in that week, and to use the material that you will produce for this purpose to build your computational essay.

Assignment 1 (50%) refer to the set of questions at the end of Chapter **??**, Chapter **??** and Chapter **??**. You are required to use your responses to build your computational essay. Each chapter provides more specific guidance of the tasks and discussion that you are required to consider in your assignment.

Assignment 2 (50%) refer to the set of questions at the end of Chapter **??**, Chapter **??**, Chapter **??** and Chapter **??**. You are required to use your responses to build your computational essay. Each chapter provides more specific guidance of the tasks and discussion that you are required to consider in your assignment.

1.5.1 Format Requirements

Both assignments will have the same requirements:

- Maximum word count: 2,000 words, excluding figures and references.
- Up to three maps, plot or figures (a figure may include more than one map and/or plot and will only count as one but needs to be integrated in the figure)
- Up to two tables.

Assignments need to be prepared in "*Quarto Document*" format (i.e. qmd extension) and then converted into a self-contained

HTML file that will then be submitted via Turnitin. The document should only display content that will be assessed. Intermediate steps do not need to be displayed. Messages resulting from loading packages, attaching data frames, or similar messages do not need to be included as output code. Useful resources to customise your R notebook can be found on Quarto's website.

Two Quarto Document templates will be available via the module Canvas site.

Submission is electronic only via Turnitin on Canvas.

1.5.2 Marking criteria

The Standard Environmental Sciences School marking criteria apply, with a stronger emphasis on evidencing the use of regression models, critical analysis of results and presentation standards. In addition to these general criteria, the code and outputs (i.e. tables, maps and plots) contained within the notebook submitted for assessment will be assessed according to the extent of documentation and evidence of expertise in changing and extending the code options illustrated in each chapter. Specifically, the following criteria will be applied:

- **0-15**: no documentation and use of default options.
- **16-39**: little documentation and use of default options.
- **40-49**: some documentation, and use of default options.
- **50-59**: extensive documentation, and edit of some of the options provided in the notebook (e.g. change north arrow location).
- **60-69**: extensive well organised and easy to read documentation, and evidence of understanding of options provided in the code (e.g. tweaking existing options).
- **70-79**: all above, plus clear evidence of code design skills (e.g. customising graphics, combining plots (or tables) into a single output, adding clear axis labels and variable names on graphic outputs, etc.).
- **80-100**: all as above, plus code containing novel contributions that extend/improve the functionality the code

was provided with (e.g. comparative model assessments, novel methods to perform the task, etc.).

2 Spatial Data

This Chapter seeks to present and describe distinctive attributes of spatial data, and discuss some of the main challenges in analysing and modelling these data. Spatial data is a term used to describe any data associating a given variable attribute to a specific location on the Earth's surface.

2.1 Spatial Data types

Different classifications of spatial data types exist. Knowing the structure of the data at hand is important as specific analytical methods would be more appropriate for particular data types. We will use a particular classification involving four data types: lattice/areal data, point data, flow data and trajectory data (Fig. 1). This is not a exhaustive list but it is helpful to motivate the analytical and modelling methods that we cover in this book.

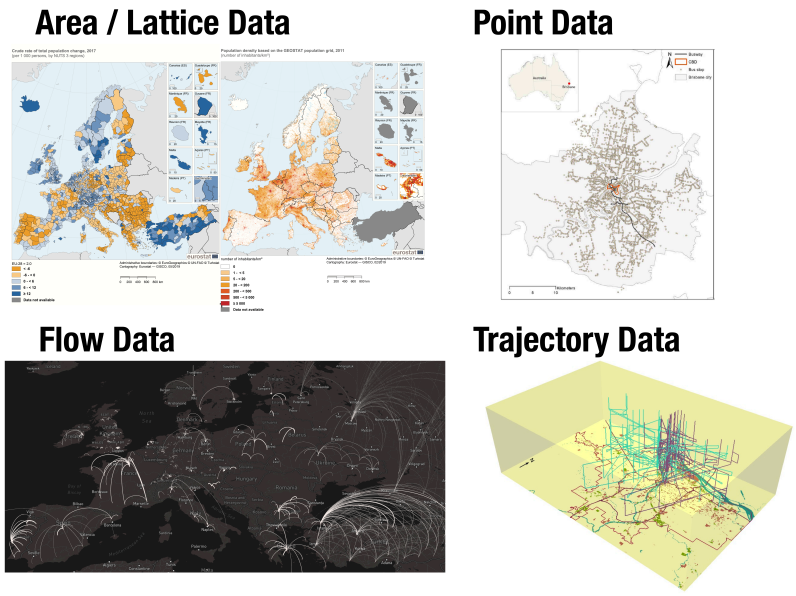


Figure 2.1: Fig. 1. Data Types. Area / Lattice data source: Önnerfors et al. (2019). Point data source: Tao et al. (2018). Flow data source: Rowe and Patias (2020). Trajectory data source: Kwan and Lee (2004).

Lattice/Areal Data. These data correspond to records of attribute values (such as population counts) for a fixed geographical area. They may comprise regular shapes (such as grids or pixels) or irregular shapes (such as states, counties or travel-to-work areas). Raster data are a common source of regular lattice/areal area, while censuses are probably the most common form of irregular lattice/areal area. Point data within an area can be aggregated to produce lattice/areal data.

Point Data. These data refer to records of the geographic location of an discrete event, or the number of occurrences of geographical process at a given location. As displayed in Fig. 1, examples include the geographic location of bus stops in a city, or the number of boarding passengers at each bus stop.

Flow Data. These data refer to records of measurements for a pair of geographic point locations. or pair of areas. These

data capture the linkage or spatial interaction between two locations. Migration flows between a place of origin and a place of destination is an example of this type of data.

Trajectory Data. These data record geographic locations of moving objects at various points in time. A trajectory is composed of a single string of data recording the geographic location of an object at various points in time and each record in the string contains a time stamp. These data are complex and can be classified into explicit trajectory data and implicit trajectory data. The former refer to well-structured data and record positions of objects continuously and intensively at uniform time intervals, such as GPS data. The latter is less structured and record data in relatively time point intervals, including sensor-based, network-based and signal-based data (Kong et al. 2018).

In this course, we cover analytical and modelling approaches for point, lattice/areal and flow data. While we do not explicitly analyse trajectory data, various of the analytical approaches described in this book can be extended to incorporate time, and can be applied to model these types of data. In Chapter **??**, we describe approaches to analyse and model spatio-temporal data. These same methods can be applied to trajectory data.

2.2 Hierarchical Structure of Data

The hierarchical organisation is a key feature of spatial data. Smaller geographical units are organised within larger geographical units. You can find the hierarchical representation of UK Statistical Geographies on the Office for National Statistics website. In the bottom part of the output below, we can observe a spatial data frame for Liverpool displaying the hierarchical structure of census data (from the smallest to the largest): Output Areas (OAs), Lower Super Output Areas (LSOAs), Middle Super Output Areas (MSOAs) and Local Authority Districts (LADs). This hierarchical structure entails that units in smaller geographies are nested within units in larger geographies, and that smaller units can be aggregated to produce large units.

```
Simple feature collection with 6 features and 4 fields
Geometry type: MULTIPOLYGON
Dimension:     XY
Bounding box:  xmin: 335071.6 ymin: 389876.7 xmax: 339426.9 ymax: 394479
Projected CRS: Transverse_Mercator
      OA_CD   LSOA_CD   MSOA_CD    LAD_CD                       geometry
1 E00176737 E01033761 E02006932 E08000012 MULTIPOLYGON (((335106.3 38...
2 E00033515 E01006614 E02001358 E08000012 MULTIPOLYGON (((335810.5 39...
3 E00033141 E01006546 E02001365 E08000012 MULTIPOLYGON (((336738 3931...
4 E00176757 E01006646 E02001369 E08000012 MULTIPOLYGON (((335914.5 39...
5 E00034050 E01006712 E02001375 E08000012 MULTIPOLYGON (((339325 3914...
6 E00034280 E01006761 E02001366 E08000012 MULTIPOLYGON (((338198.1 39...
```

Next we quickly go through the components of the output above. The first line indicates the type of feature and the number of rows (features) and columns (fields) in the data frame, except for the geometry. The second and third lines identify the type of geometry and dimension. The fourth line `bbox` stands for bounding box and display the min and max coordinates containing the Liverpool area in the data frame. The fifth line `projected CRS` indicates the coordinate reference system projection. If you would like to learn more about the various components of spatial data frames, please see the *R* `sf` package vignette on Simple Features.

2.3 Key Challenges

Major challenges exist when working with spatial data. Below we explore some of the key longstanding problems data scientists often face when working with geographical data.

2.3.1 Modifible Area Unit Problem (MAUP)

The Modifible Area Unit Problem (MAUP) represents a challenge that has troubled geographers for decades (Openshaw 1981). Two aspects of the MAUP are normally recognised in empirical analysis relating to *scale* and *zonation*. Fig. 2 illustrates these issues

- *Scale* refers to the idea that a geographical area can be divided into geographies with differing numbers of spatial units.
- *Zonation* refers to the idea that a geographical area can be divided into the same number of units in a variety of ways.

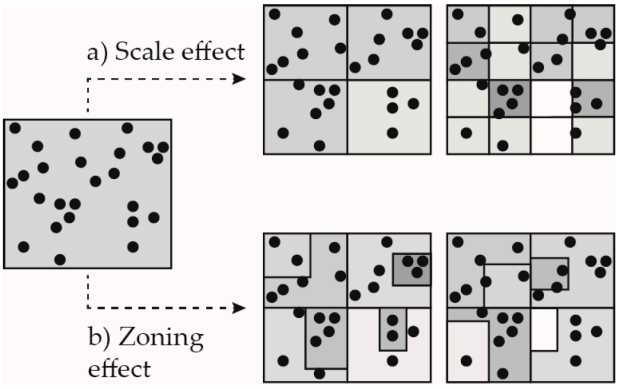


Figure 2.2: Fig. 2. MAUP effect. (a) scale effect; and, (b) zonation effect. Source: Loidl et al. (2016).

The MAUP is a critical issue as it can impact our analysis and thus any conclusions we can infer from our results (e.g. A. S. Fotheringham and Wong 1991). There is no agreed systematic approach on how to handle the effects of the MAUP. Some have suggested to perform analyses based on different existing geographical scales, and assess the consistency of the results and identify potential sources of change. The issue with such approach is that results from analysis at different scales are likely to differ because distinct dimensions of a geographic process may be captured at different scales. For example, in migration studies, smaller geographies may be more suitable to capture residential mobility over short distances, while large geographies may be more suitable to capture long-distance migration. And it is well documented that these types of moves are driven by different factors. While residential mobility tends to be driven by housing related reasons, long-distance migration is more closely related to employment-related motives (Niedomysl 2011).

An alternative approach is to use the smallest geographical system available and create random aggregations at various geographical scales, to directly quantify the extent of scale and zonation. This approach has shown promising results in applications to study internal migration flows (Stillwell, Daras, and Bell 2018). Another approach involves the production of "meaningful" or functional geographies that can more appropriately capture the process of interest. There is an active area of work defining functional labour markets (Casado-Díaz, Martínez-Bernabéu, and Rowe 2017), urban areas (Daniel Arribas-Bel, Garcia-López, and Viladecans-Marsal 2021) and various forms of geodemographic classifications (A. D. Singleton and Spielman 2013; Patias, Rowe, and Cavazzi 2019) . However there is the recognition that none of the existing approaches resolve the effects of the MAUP and recently it has been suggested that the most plausible 'solution' would be to ignore the MAUP (Wolf et al. 2020).

2.3.2 Ecological Fallacy

Ecological fallacy is an error in the interpretation of statistical data based on aggregate information. Specifically it refers to inferences made about the nature of specific individuals based solely on statistics aggregated for a given group. It is about thinking that relationships observed for groups necessarily hold for individuals. A key example is Robinson (1950) who illustrates this problem exploring the difference between ecological correlations and individual correlations. He looked at the relationship between country of birth and literacy. Robinson (1950) used the percent of foreign-born population and percent of literate population for the 48 states in the United States in 1930. The ecological correlation based on these data was 0.53. This suggests a positive association between foreign birth and literacy, and could be interpreted as foreign born individuals being more likely to be literate than native-born individuals. Yet, the correlation based on individual data was negative -0.11 which indicates the opposite. The main point emerging from this example is to carefully interpret analysis based on spatial data and avoid making inferences about individuals from these data.

2.3.3 Spatial Dependence

Spatial dependence refers to the spatial relationship of a variable's values for a pair of locations at a certain distance apart, so that these values are more similar (or less similar) than expected for randomly associated pairs of observations (Anselin 1988). For example, we could think of observed patterns of ethnic segregation in an area are a result of spillover effects of pre-existing patterns of ethnic segregation in neighbouring areas. Chapter **??** will illustrate approach to explicitly incorporate spatial dependence in regression analysis.

2.3.4 Spatial Heterogeneity

Spatial heterogeneity refers to the uneven distribution of a variable's values across space. Concentration of deprivation or unemployment across an area are good examples of spatial heterogeneity. We illustrate various ways to visualise, explore and measure the spatial distribution of data in multiple chapters. We also discuss on potential modelling approaches to capture spatial heterogeneity in Chapter **??**, Chapter **??** and Chapter **??**.

2.3.5 Spatial nonstationarity

Spatial nonstationarity refers to variations in the relationship between an outcome variable and a set of predictor variables across space. In a modelling context, it relates to a situation in which a simple "global" model is inappropriate to explain the relationships between a set of variables. The geographical nature of the model must be modified to reflect local structural relationships within the data. For example, ethinic segregation has been positively associated with employment outcomes in some countries pointing to networks in pre-existing communities facilitating access to the local labour market. Inversely ethinic segregation has been negatively associated with employment outcomes pointing to lack of integration into the broader local community. We illustrate various modelling approaches to capture spatial nonstationarity in Chapter **??** and Chapter **??**.

3 Data Wrangling

In this chapter, we will cover the fundamentals of the concepts and functions that you will need to know to navigate this book. We will introduce key concepts and functions relating to what computational notebooks are and how they work. We will also cover basic R functions and data types, including the use of factors. Additionally, we will offer a basic understanding of the manipulation and mapping of spatial data frames using commonly used libraries such as `tidyverse`, `sf`, `ggplot` and `tmap`.

If you are already familiar with R, R computational 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, Allaire, and Grolemund (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, Nowosad, and Muenchow (2019), a really good book on handling spatial data and historical background of the evolution of R packages for spatial data analysis.

3.1 Dependencies

This chapter uses the libraries below. Ensure they are installed on your machine[1] before you progress.

```
# data manipulation, transformation and visualisation
library(tidyverse)
# nice tables
library(kableExtra)
# spatial data manipulation
library(sf)
# thematic mapping
library(tmap)
# colour palettes
library(RColorBrewer)
library(viridis)
```

3.2 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 The Comprehensive R Archive Network (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.

[1] 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.

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:

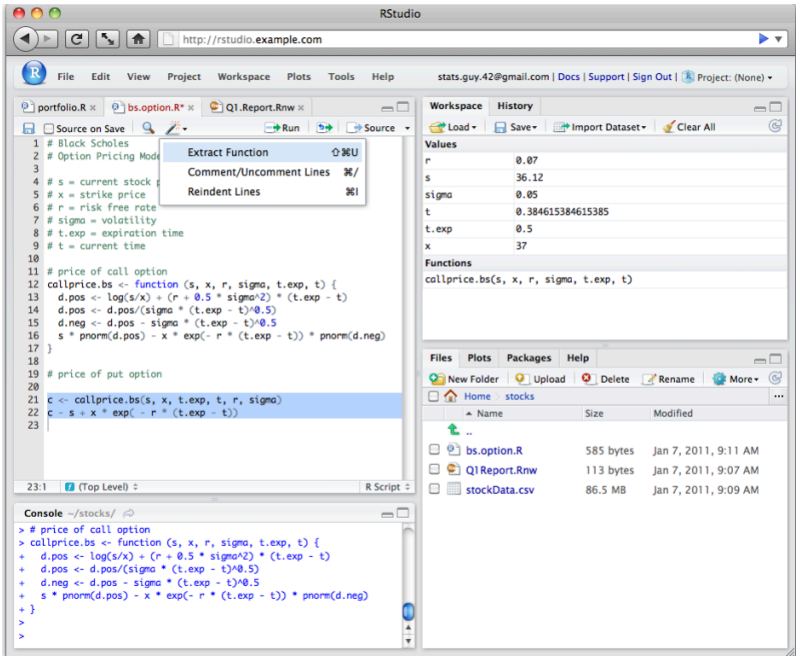


Figure 3.1: Fig. 1. RStudio features.

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

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

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

You can check your current working directory by typing:

```
getwd()
```

```
[1] "/Users/franciscorowe/Dropbox/Francisco/uol/teaching/envs453/202324/sam"
```

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

3.4 R Scripts and Computational Notebooks

An *R script* is a series of commands that you can execute at one time and help you save time. So you do not 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.

https://r4ds.hadley.nz/workflow-basics.html

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:

> **Note**
>
> To get familiar with good practices in writing your code in R, we recommend the Chapter Workflow: basics and Workflow: scripts and projects from the R in Data Science book by Wickham, Çetinkaya-Rundel, and Grolemund (2023).

```
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* or a *Quarto Document* are a Markdown options with descriptive text and code chunks that can be executed independently and interactively, with output visible immediately beneath a code chunk - see Xie, Allaire, and Grolemund (2019). A *Quarto Document* is an improved version of the original *R Notebook*. *Quarto Document* requires a package called Quarto. Quarto does not have a dependency or requirement for R. Quarto is multilingual, beginning with R, Python, Javascript, and Julia. The concept is that Quarto will work

even for languages that do not yet exist. This book was original written in *R Notebook* but later transitioned into *Quarto Documents*.

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 3.2: 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 e.g.:

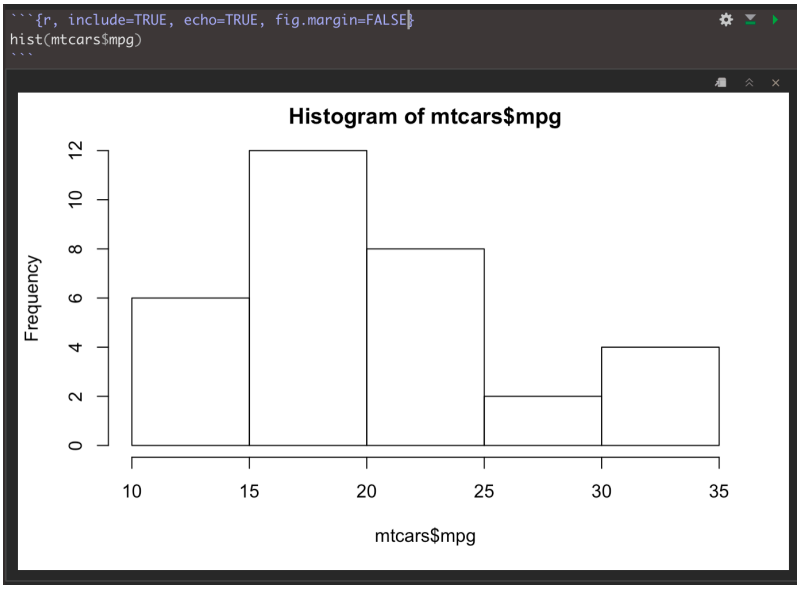


Figure 3.3: Fig. 3. Code chunk example. Details on the various options: https://rmarkdown.rstudio.com/lesson-3.html

- 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 ...`

To create a *Quarto Document*, you need to:

- Open a new script file: *File > New File > Quarto Document*

Quarto Documents work in the same way as R Notebooks with small variations. You find a comprehensive guide on the Quarto website.

3.5 Getting Help

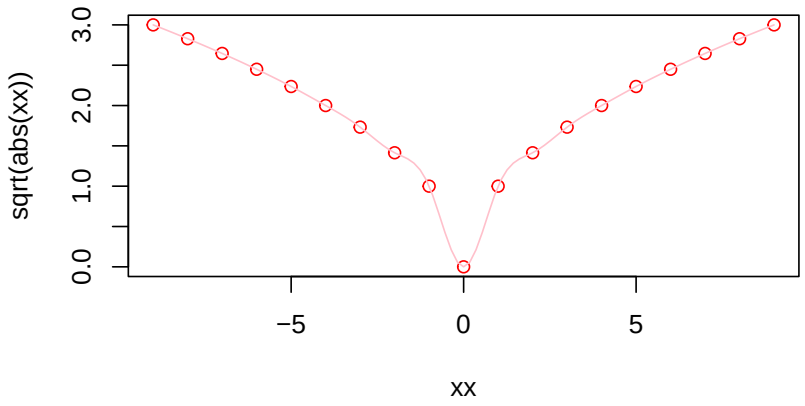
You can use `help` or `?` to ask for details for a specific function:

```
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")
```




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

3.6 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.

3.6.1 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 enj
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"
```

3.6.2 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.

3.6.2.1 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.

3.7 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","Cl

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.

3.7.1 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 ...
```

3.7.2 Referencing Data Frames

To refer to particular parts of a dataframe - say, a particular column (an area attribute), or a subset of respondents. Hence it is worth spending some time understanding how to reference dataframes.

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"
```

3.8 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

> **Note**
>
> When opening a file, ensure the correct directory set up pointing to your data. It may differ from your existing working directory.

```
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
```

3.8.1 Quickly inspect the data

Using the following questions to lead the inspection: What class? What R data types? 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)
```

3.9 Manipulation Data

3.9.1 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`.

3.9.2 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.

3.9.3 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.

3.9.4 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 (i.e. `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, # df1
                   census2, # df2
                   by.x="code", by.y="geo_code", # common ids
                   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
```

3.9.5 Saving Data

It may also be convenient 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")
```

3.10 Using Spatial Data Frames

A core area of the 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 packages exist in CRAN, including `sf` (Pebesma 2018, 2022a), `stars` (Pebesma 2022b), `terra`, `s2` (Dunnington, Pebesma, and Rubak 2023), `lwgeom` (Pebesma 2023), `gstat` (Pebesma 2004; Pebesma and Graeler 2022), `spdep` (R. Bivand 2022), `spatialreg` (R. Bivand and Piras 2022), `spatstat` (Baddeley, Rubak, and Turner 2015; Baddeley, Turner, and Rubak 2022), `tmap` (Tennekes 2018, 2022), `mapview` (Appelhans et al. 2022)

and more. A key package is this ecosystem is `sf` (Pebesma and Bivand 2023). R package `sf` provides a table format for simple features, where feature geometries are stored in a list-column. It appeared in 2016 and was developed to move spatial data analysis in R closer to standards-based approaches seen in the industry and open source projects, to build upon more modern versions of open source geospatial software stack and allow for integration of R spatial software with the `tidyverse` (Wickham et al. 2019), particularly `ggplot2`, `dplyr`, and `tidyr`. Hence, this book relies heavely on `sf` for the manipulation and analysis of the data.

To read our spatial data, we use the `st_read` function. We read a shapefile containing data at Output Area (OA) level for Liverpool. These data illustrates the hierarchical structure of spatial data.

> **Note**
>
> Lovelace, Nowosad, and Muenchow (2024) provide a helpful overview and evolution of R spatial package ecosystem.

3.10.1 Read Spatial Data

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

```
Reading layer `Liverpool_OA' from data source
  `/Users/franciscorowe/Dropbox/Francisco/uol/teaching/envs453/202324/san/data/census/Liverpool
  using driver `ESRI Shapefile'
Simple feature collection with 1584 features and 18 fields
Geometry type: MULTIPOLYGON
Dimension:     XY
Bounding box:  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 NA NA NA NA
  ..- attr(*, "names")= chr [1:18] "OA_CD" "LSOA_CD" "MSOA_CD" "LAD_CD" ...
```

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

```
Simple feature collection with 6 features and 18 fields
Geometry type: MULTIPOLYGON
Dimension:     XY
Bounding box:  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...
```

3.10.2 Basic Mapping

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

Task

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

- `ggplot2` to create data visualisations, including static maps
- `shiny` to create web applications, including maps

In this book, we will make use of `plot`, `tmap` and `ggplot`. Normally you use `plot` to get a quick inspection of the data and `tmap` and `ggplot` to get publication quality data visualisations. 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.

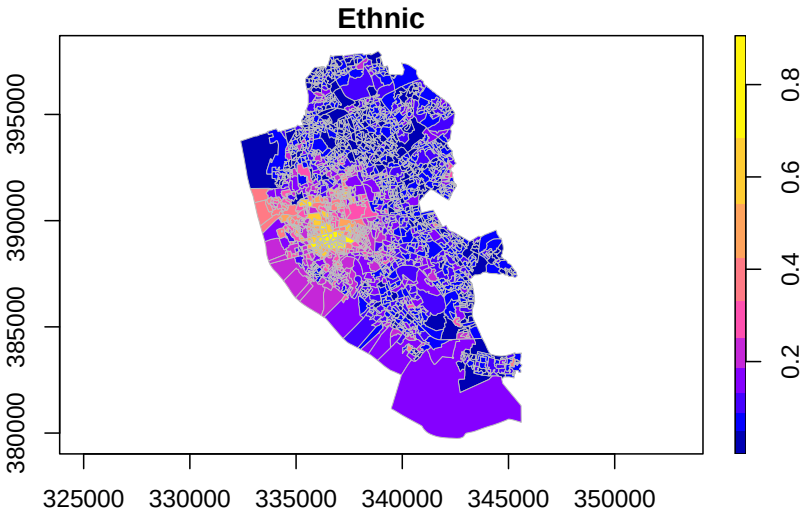
Using `plot`

We can use the base `plot` function to display the boundaries of OAs in Liverpool.

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

To visualise a column in the spatial data frame, we can run:

```
# map attributes, adding intervals
plot(oa_shp["Ethnic"], # variable to visualise
     key.pos = 4,
     axes = TRUE,
     key.width = lcm(1.3),
     key.length = 1.,
     breaks = "jenks", # algorithm to categorise the data
     lwd = 0.1,
     border = 'grey') # boundary colour
```



Let us now explore `ggplot` or `tmap`.

Using `ggplot`

We can visualise spatial data frames using `ggplot` fairly easily. `ggplot` is a generic sets of functions which was not specifically designed for spatial mapping, but it is fairly flexible that allows producing great spatial data visualisations.

Following the grammar of `ggplot`, we plot spatial data drawing layers. `ggplot` has a basic structure of three components:

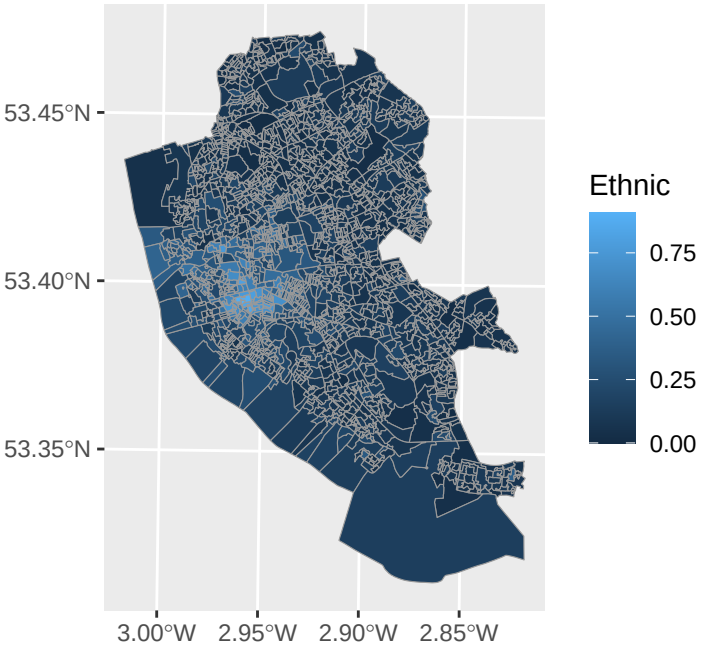
- The data i.e. `ggplot( data = *data frame*)`.
- Geometries i.e. `geom_xxx( )`.
- Aesthetic mapping i.e. `aes(x=*variable*, y=*variable*)`

We can put these three components together using `+`. This is similar to the ways we apply the pipe operator in for tidyverse. The latter component of aesthetic mapping can be added in both the `ggplot` and `geom_xxx` functions.

To map our data, we can then run:

```
ggplot(data = oa_shp, aes( fill = Ethnic) ) + # add data frame and variable to map
  geom_sf(colour = "gray60",  # colour line
          size = 0.1) # line size
```

Task



We can change the colour palette by using a different colour palette. We can use the viridis package. The power of `viridis` is that it uses color scales that visually pleasing, colorblind friendly, print well in gray scale, and can be used for both categorical and continuous data. For categorical data you can also use ColourBrewer.

So let us (1) change the colour pallete to viridis, (2) remove the colour of the boundaries, and (3) replace the theme with the theme we will be using for the book. Let's read the theme first and then implement these changes.

The `ggplot` themes for the book are in a file called `data-visualisation_theme.R` in the `style` folder. We read this file by running:

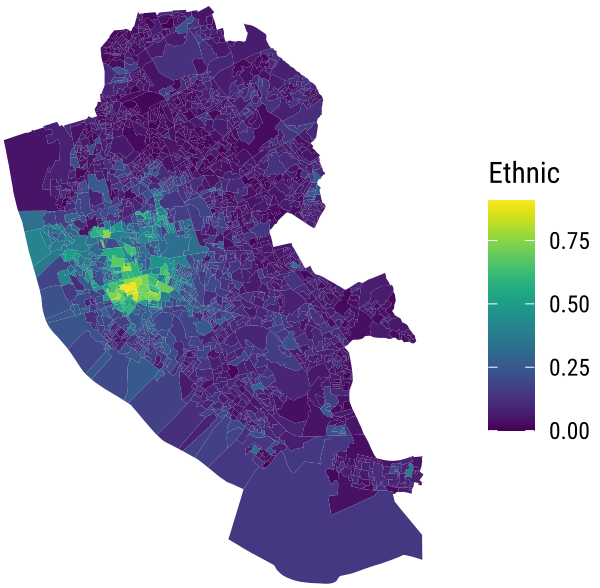
```
source("./style/data-visualisation_theme.R")
```

```
Loading required package: sysfonts
```

```
Loading required package: showtextdb
```

We can now implement our changes:

```
ggplot(data = oa_shp, aes( fill = Ethnic) ) + # add data frame and variable to map
  geom_sf(colour = "transparent") + # colour line
  scale_fill_viridis( option = "viridis" ) + # add viridis colour scheme
  theme_map_tufte()
```



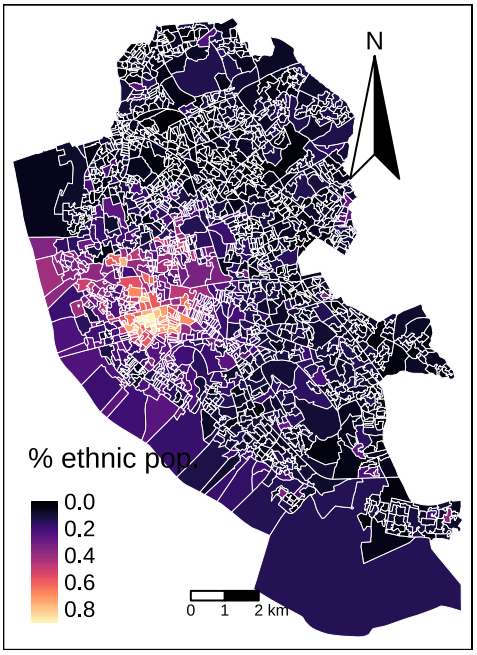
To master `ggplot`, see Wickham (2009).

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 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
  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 sc
map_oa
```



Note that the operation `+`, as for `ggplot` is used to add new layers. You can set style themes by `tm_style`. To visualise the existing styles use `tmap_style_catalogue()`. An advantage of `tmap` is that you can easily create an interactive map by running `tmap_mode`.

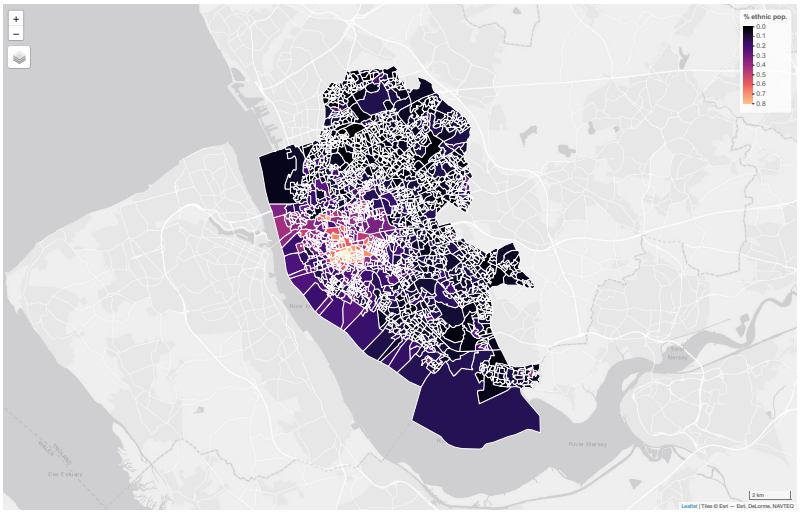
```
tmap_mode("view")
```

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

```
map_oa
```

```
Compass not supported in view mode.
```

```
Warning: In view mode, scale bar breaks are ignored.
```



3.10.3 Comparing geographies

If you recall, one of the key issues of working with spatial data is the modifiable area unit problem (MAUP) - see (**spatial_data?**). 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.

```
tmap_mode("plot")
```

```
tmap mode set to plotting
```

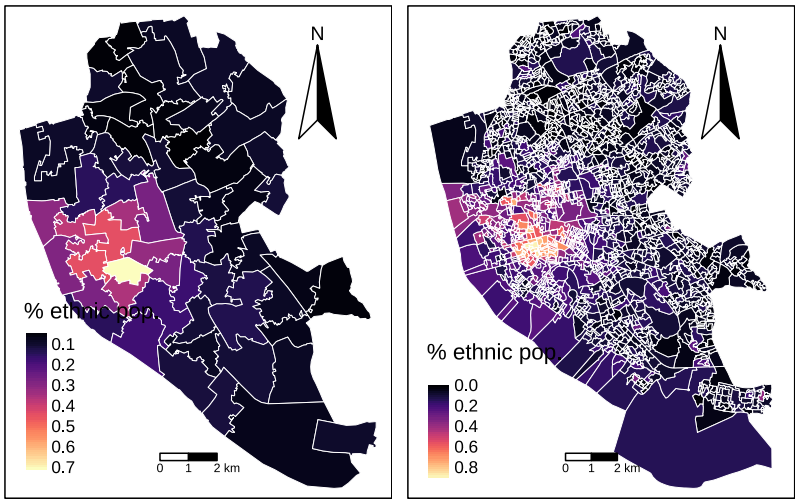
```
# read data at the msoa level
msoa_shp <- st_read("data/census/Liverpool_MSOA.shp")
```

```
Reading layer `Liverpool_MSOA' from data source
  `/Users/franciscorowe/Dropbox/Francisco/uol/teaching/envs453/202324/san/data/census/Liverpool
```

> **Note**
>
> The first line of the chunk code include `tmap_mode("plot")` which tells R that we want a static map. `tmap_mode` works like a switch to interactive and non-interactive mapping.

```
  using driver `ESRI Shapefile'
Simple feature collection with 61 features and 16 fields
Geometry type: MULTIPOLYGON
Dimension:     XY
Bounding box:  xmin: 333086.1 ymin: 381426.3 xmax: 345636 ymax: 397980.1
Projected CRS: Transverse_Mercator
```

```r
# 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 Point Data Analysis

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

- Lovelace, Nowosad, and Muenchow (2019) offer a great introduction.
- Chapter 6 of Brunsdon and Comber (2015), in particular subsections 6.3 and 6.7.
- R. S. Bivand, Pebesma, and Gómez-Rubio (2013) provides an in-depth treatment of spatial data in R.

4.1 Dependencies

We will rely on the following libraries in this section, all of them included in Section **??**:

```
# data manipulation, transformation and visualisation
library(tidyverse)
# spatial data manipulation
library(sf)
library(sp)
# data visualisation
library(gridExtra)
# basemap
library(basemapR)
# interpolation
library(gstat)
library(hexbin)
```

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.

4.2 Data

For this session, we will use the set of Airbnb properties for San Diego (US), borrowed from the "Geographic Data Science with Python" book (see here for more info on the dataset source). This covers the point location of properties advertised on the Airbnb website in the San Diego region.

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

```
db <- st_read("data/abb_sd/regression_db.geojson")
```

```
Reading layer `regression_db' from data source
  `/Users/franciscorowe/Dropbox/Francisco/uol/teaching/envs453/202324/san/data/abb_sd/regressi
  using driver `GeoJSON'
Simple feature collection with 6110 features and 19 fields
Geometry type: POINT
Dimension:     XY
Bounding box:  xmin: -117.2812 ymin: 32.57349 xmax: -116.9553 ymax: 33.08311
Geodetic CRS:  WGS 84
```

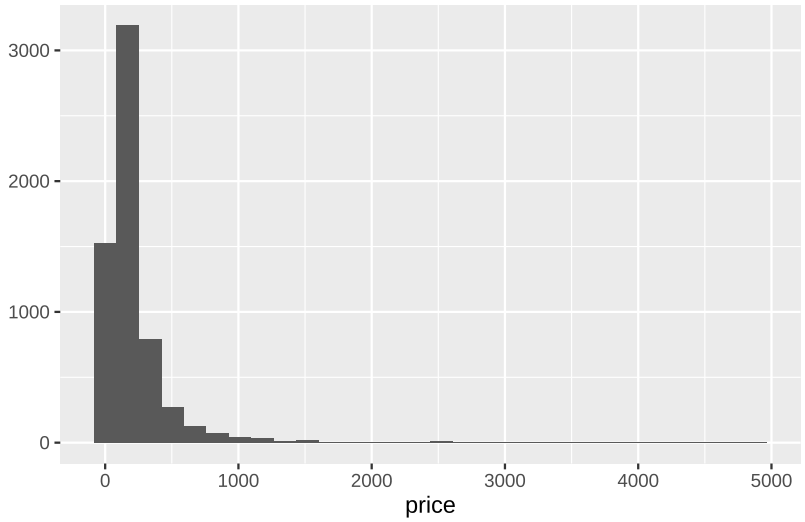
We can then examine the columns of the table with the `colnames` method:

```
colnames(db)
```

```
 [1] "accommodates"       "bathrooms"          "bedrooms"
 [4] "beds"               "neighborhood"       "pool"
 [7] "d2balboa"           "coastal"            "price"
[10] "log_price"          "id"                 "pg_Apartment"
[13] "pg_Condominium"     "pg_House"           "pg_Other"
[16] "pg_Townhouse"       "rt_Entire_home.apt" "rt_Private_room"
[19] "rt_Shared_room"     "geometry"
```

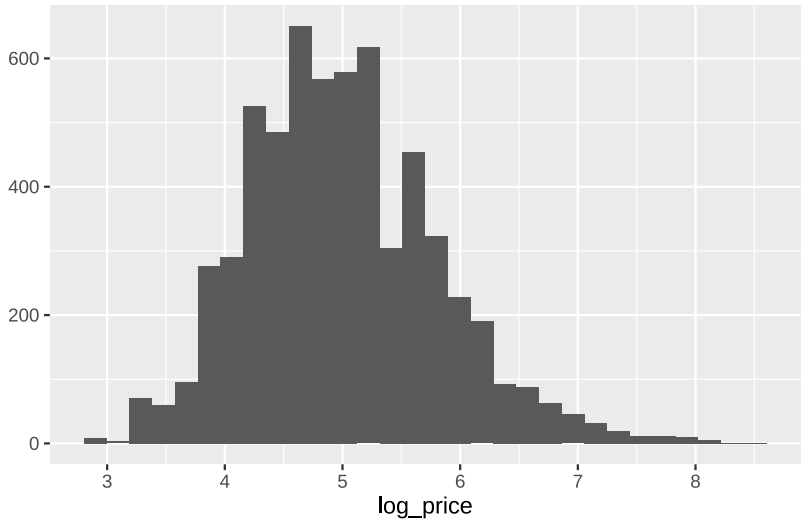
The rest of this session will focus on two main elements of the table: the spatial dimension (as stored in the point coordinates), and the nightly price values, expressed in USD and contained in the `price` column. To get a sense of what they look like first, let us plot both. We can get a quick look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

```
# Create the histogram
qplot( data = db, x = price)
```



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. The original table already contains an additional column with the logarithm of each price (`log_price`).

```
# Create the histogram
qplot( data = db, x = log_price )
```



To obtain the spatial distribution of these houses, we need to focus on the `geometry` column. The easiest, quickest (and also "dirtiest") way to get a sense of what the data look like over space is using `plot`:

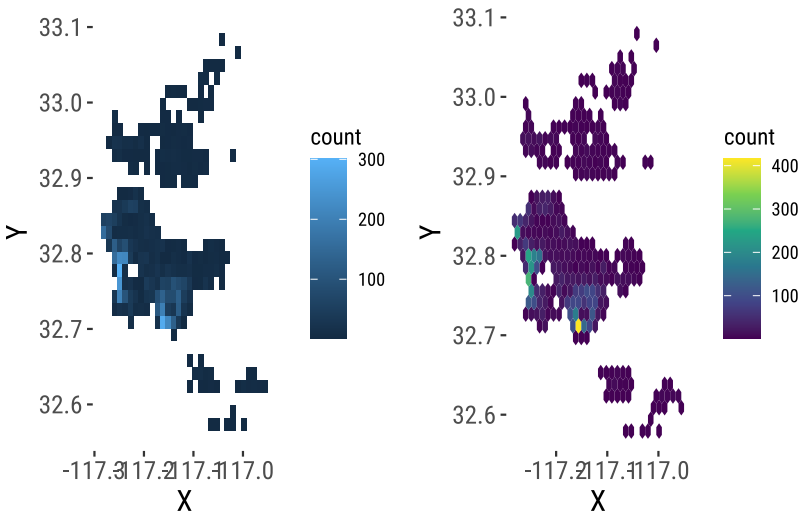
```
plot(st_geometry(db))
```

Now this has the classic problem of cluttering: some portions of the map have so many points that we can't tell what the distribution is like. To get around this issue, there are two solutions: binning and smoothing.

4.3 Binning

The two-dimensional sister of histograms are binning maps: we divide each of the two dimensions into "buckets", and count how many points fall within each bucket. Unlike histograms, we encode that count with a color gradient rather than a bar chart over an additional dimension (for that, we would need a 3D plot). These "buckets" can be squares (left) or hexagons (right):

```
      # Squared binning
# Set up plot
sqbin <- ggplot( ) +
# Add 2D binning with the XY coordinates as
# a dataframe
  geom_bin2d(
    data = as.data.frame( st_coordinates( db ) ),
    aes( x = X, y = Y)
  ) +
  # set theme
  theme_plot_tufte()
      # Hex binning
# Set up plot
hexbin <- ggplot() +
# Add hex binning with the XY coordinates as
# a dataframe
  geom_hex(
    data = as.data.frame( st_coordinates( db ) ),
    aes( x = X, y = Y)
  ) +
# Use viridis for color encoding (recommended)
  scale_fill_continuous( type = "viridis" ) +
  theme_plot_tufte()
      # Bind in subplots
grid.arrange( sqbin, hexbin, ncol = 2 )
```



4.4 KDE

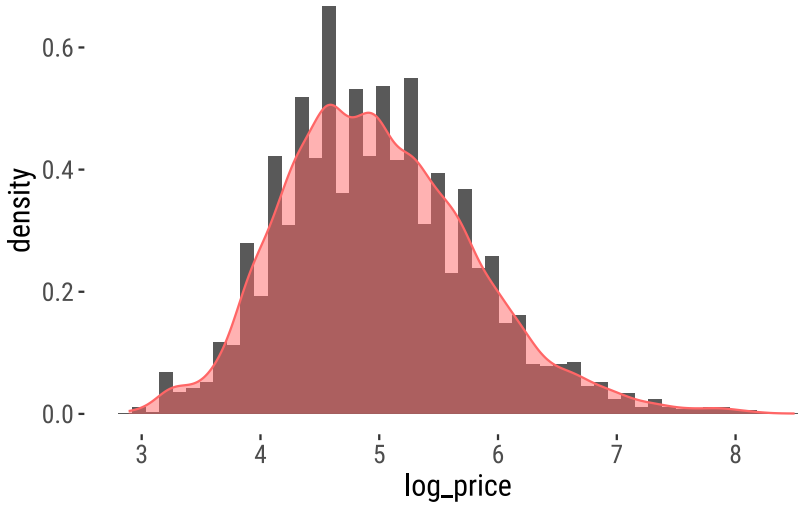
Kernel Density Estimation (KDE) is a technique that creates a *continuous* representation of the distribution of a given variable, such as house prices. Although theoretically it can be applied to any dimension, usually, KDE is applied to either one or two dimensions.

4.4.1 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, aes(x=log_price))
# Histogram
hist <- base +
  geom_histogram(bins=50, aes(y=..density..))
# Overlay density plot
kde <- hist +
  geom_density(fill="#FF6666", alpha=0.5, colour="#FF6666") +
  theme_plot_tufte()
kde
```

```
Warning: The dot-dot notation (`..density..`) was deprecated in ggplot2 3.4.0.
i Please use `after_stat(density)` instead.
```



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 shape-wise, but it has been rescalend on the Y axis to reflect probabilities rather than counts.

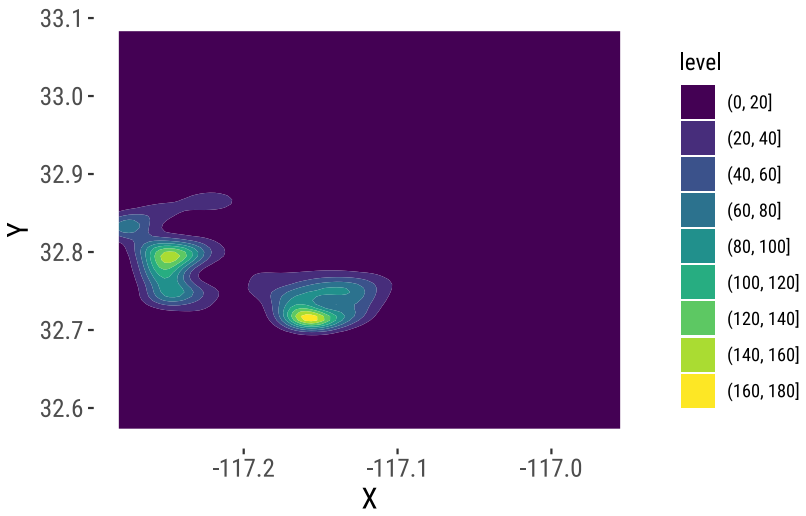
4.4.2 Two-dimensional (spatial) KDE

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 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, we can use general tooling for non-spatial points, such as the `stat_density2d_filled` method:

```
# Create the KDE surface
kde <- ggplot(data = db) +
  stat_density2d_filled(alpha = 1,
```

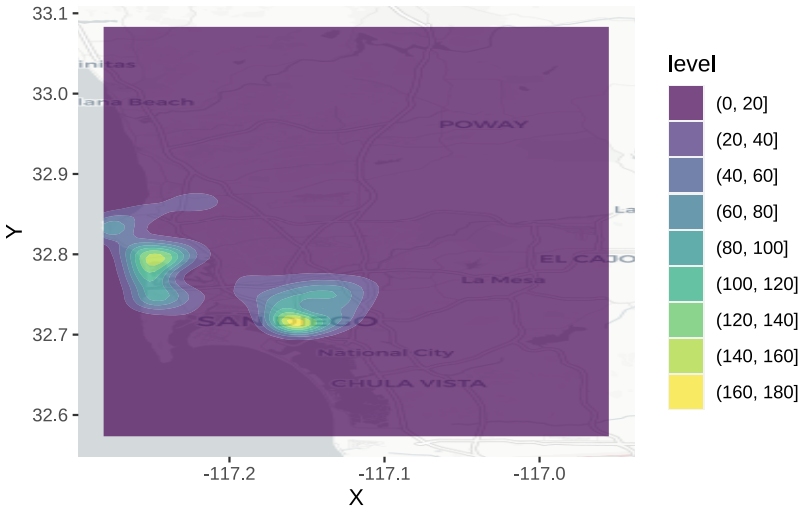
```
    data = as.data.frame(st_coordinates(db)),
    aes(x = X, y = Y),
    n = 100
  ) +
  # Tweak the color gradient
  scale_color_viridis_c() +
  # White theme
  theme_plot_tufte()
kde
```



This approach generates 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 `basemapR`, 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.

```
bbox_db <- st_bbox(db)
ggplot() +
  base_map(bbox_db, increase_zoom = 2, basemap = "positron") +
  #geom_sf(data = db, fill = NA) +
```

```
  stat_density2d_filled(alpha = 0.7,
    data = as.data.frame(st_coordinates(db)),
    aes(x = X, y = Y),
    n = 100
  )
```



4.5 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 locations, 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 prices in AirBnb, fits into this category as well: although where a house is advertised 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 advertised?* 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[1]. In this chapter, 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, Carlin, and Gelfand (2014) or Cressie (2015) are hard but good overviews.

4.5.1 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

[1] 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*.

> importance in the interpolation, greater weight is given to these points [...]
>
> — Page 204

The math[2] 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. 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[3].

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[4]. First, let us set up a grid for the extent of the bounding box of our data (not the use of pipe, `%>%`, operator to chain functions):

```
sd.grid <- db %>%
  st_bbox() %>%
  st_as_sfc() %>%
  st_make_grid(
    n = 100,
    what = "centers"
```

[2] 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.

[3] If you want a complementary view of point interpolation in R, you can read more on this fantastic blog post

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

```
  ) %>%
  st_as_sf() %>%
  cbind(., st_coordinates(.))
```

The object `sd.grid` is a regular grid with 10,000 (100×100) equally spaced cells:

```
sd.grid
```

```
Simple feature collection with 10000 features and 2 fields
Geometry type: POINT
Dimension:     XY
Bounding box:  xmin: -117.2795 ymin: 32.57604 xmax: -116.9569 ymax: 33.08056
Geodetic CRS:  WGS 84
First 10 features:
           X        Y                          x
1  -117.2795 32.57604 POINT (-117.2795 32.57604)
2  -117.2763 32.57604 POINT (-117.2763 32.57604)
3  -117.2730 32.57604  POINT (-117.273 32.57604)
4  -117.2698 32.57604 POINT (-117.2698 32.57604)
5  -117.2665 32.57604 POINT (-117.2665 32.57604)
6  -117.2632 32.57604 POINT (-117.2632 32.57604)
7  -117.2600 32.57604   POINT (-117.26 32.57604)
8  -117.2567 32.57604 POINT (-117.2567 32.57604)
9  -117.2535 32.57604 POINT (-117.2535 32.57604)
10 -117.2502 32.57604 POINT (-117.2502 32.57604)
```

Now, `sd.grid` only contain the location of points to which we wish to interpolate. That is, we now have our "target" geography for which we'd like to have AirBnb prices, but we don't have price estimates. For that, on to the IDW, which will generate estimates for locations in `sd.grid` based on the observed prices in `db`. Again, this is hugely simplified by `gstat`:

```
idw.hp <- idw(
  price ~ 1,          # Formula for IDW
  locations = db,     # Initial locations with values
  newdata=sd.grid,    # Locations we want predictions for
  nmax = 150          # Limit the number of neighbours for IDW
)
```

```
[inverse distance weighted interpolation]
```

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 `sd.grid` object we just created above). One more note: by default, `idw` 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 nearest observations to each point[5].

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)
```

```
Simple feature collection with 6 features and 2 fields
Geometry type: POINT
Dimension:     XY
Bounding box:  xmin: -117.2795 ymin: 32.57604 xmax: -117.2632 ymax: 32.57604
Geodetic CRS:  WGS 84
  var1.pred var1.var                   geometry
1  295.6100       NA POINT (-117.2795 32.57604)
2  295.1651       NA POINT (-117.2763 32.57604)
3  296.5927       NA  POINT (-117.273 32.57604)
4  288.2252       NA POINT (-117.2698 32.57604)
5  281.5522       NA POINT (-117.2665 32.57604)
6  268.3567       NA POINT (-117.2632 32.57604)
```

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

The column we will pay attention to is `var1.pred`. For a hypothetical house advertised at the location in the first row of point in `sd.grid`, the price IDW would guess it would cost, based on prices nearby, is the first element of column `var1.pred` in `idw.hp`.

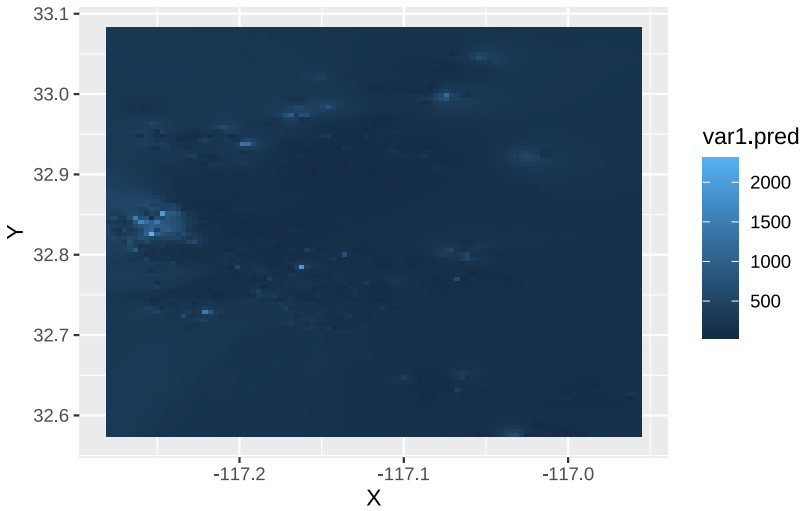
4.5.2 A surface of housing prices

Once we have the IDW object computed, we can plot it to explore the distribution, not of AirBnb locations in this case, but of house prices over the geography of San Diego. To do this using `ggplot2`, we first append the coordinates of each grid cell as columns of the table:

```
idw.hp = idw.hp %>%
  cbind(st_coordinates(.))
```

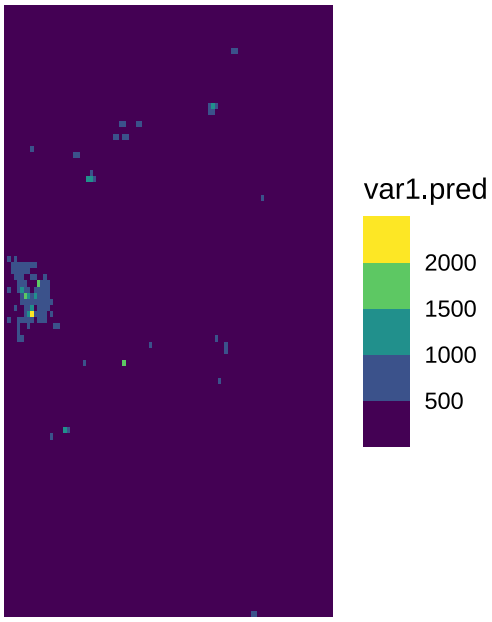
Now, we can visualise the surface using standard `ggplot2` tools:

```
ggplot(idw.hp, aes(x = X, y = Y, fill = var1.pred)) +
  geom_raster()
```



And we can "dress it up" a bit further:

```
ggplot(idw.hp, aes(x = X, y = Y, fill = var1.pred)) +
  geom_raster() +
  scale_fill_viridis_b() +
  theme_void() +
  geom_sf(alpha=0)
```



Looking at this, 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[6].

4.5.3 *"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

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

working for Airbnb 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, a new house is going to be advertised on the coordinates `X = -117.02259063720702, Y = 32.76511965117273` as expressed in longitude and latitude. In that case, we can do as follows:

```
pt <- c(X = -117.02259063720702, Y = 32.76511965117273) %>%
  st_point() %>%
  st_sfc() %>%
  st_sf(crs = "EPSG:4326") %>%
  st_transform(st_crs(db))
idw.one <- idw(price ~ 1, locations=db, newdata=pt)
```

```
[inverse distance weighted interpolation]
```

```
idw.one
```

```
Simple feature collection with 1 feature and 2 fields
Geometry type: POINT
Dimension:     XY
Bounding box:  xmin: -117.0226 ymin: 32.76512 xmax: -117.0226 ymax: 32.76512
Geodetic CRS:  WGS 84
  var1.pred var1.var                   geometry
1  171.4141       NA POINT (-117.0226 32.76512)
```

And, as show above, the estimated value is $171.4141334[7].

[7] **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 in the distribution?

4.6 Questions

We will be using the Madrid AirBnb dataset:

```
mad_abb <- st_read("data/assignment_1_madrid/madrid_abb.gpkg")
```

```
Reading layer `madrid_abb' from data source
  `/Users/franciscorowe/Dropbox/Francisco/uol/teaching/envs453/202324/san/data/assignment_1_mad
  using driver `GPKG'
Simple feature collection with 18399 features and 16 fields
Geometry type: POINT
Dimension:     XY
Bounding box:  xmin: -3.86391 ymin: 40.33243 xmax: -3.556 ymax: 40.56274
Geodetic CRS:  WGS 84
```

This is fairly similar in spirit to the one from San Diego we have relied on for the chapter, although the column set is not exactly the same:

```
colnames(mad_abb)
```

```
 [1] "price"           "price_usd"       "log1p_price_usd" "accommodates"
 [5] "bathrooms_text"  "bathrooms"       "bedrooms"        "beds"
 [9] "neighbourhood"   "room_type"       "property_type"   "WiFi"
[13] "Coffee"          "Gym"             "Parking"         "km_to_retiro"
[17] "geom"
```

For this set of questions, the only two columns we will need is `geom`, which contains the point geometries, and `price_usd`, which record the price of the AirBnb property in USD.

With this at hand, answer the following questions:

1. Create a KDE that represents the density of locations of AirBnb properties in Madrid
2. Using inverse distance weighting, create a surface of AirBnb prices

5 Spatial Interaction Modelling

This chapter 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:

- A. S. Fotheringham and O'Kelly (1989) offer a historical overview of spatial interaction models and illustration of use cases.
- Rowe, Lovelace, and Dennett (2022) provide a good overview of the existing limitations and opportunities of spatial interaction modelling.
- A. 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 (2006), in particular Chapters 6 and 7.

5.1 Dependencies

We will rely on the following libraries in this section, all of them included in Section **??**:

```
# Data management
library(tidyverse)
# Spatial Data management
library(sf)
library(sp)
# Pretty graphics
library(ggplot2)
# Thematic maps
library(tmap)
# Add basemaps
library(basemapR)
# Simulation methods
library(arm)
```

In this chapter we will show a slightly different way of managing spatial data in R. Although most of the functionality will be similar to that seen in previous chapters, we will not rely on the "`sf` stack" and we will instead show how to read and manipulate data using the more traditional `sp` stack. Although this approach is being slowly phased out, it is still important to be aware of its existence and its differences with more modern approaches.

5.2 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 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, also in the same folder.

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

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

```
Reading layer `flows' from data source
  `/Users/franciscorowe/Dropbox/Francisco/uol/teaching/envs453/202324/san/data/sf_bikes/flows.g
  using driver `GeoJSON'
Simple feature collection with 1722 features and 9 fields
Geometry type: LINESTRING
Dimension:     XY
Bounding box:  xmin: -122.4237 ymin: 37.76914 xmax: -122.3875 ymax: 37.80737
Geodetic CRS:  WGS 84
```

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)
```

```
Simple feature collection with 6 features and 9 fields
Geometry type: LINESTRING
Dimension:     XY
Bounding box:  xmin: -122.4083 ymin: 37.7838 xmax: -122.3945 ymax: 37.80097
Geodetic CRS:  WGS 84
  flow_id dest orig straight_dist street_dist total_down total_up trips15
1   39-41   41   39      1452.201   1804.1150  11.205753 4.698162      68
2   39-42   42   39      1734.861   2069.1557  10.290236 2.897886      23
3   39-45   45   39      1255.349   1747.9928  11.015596 4.593927      83
4   39-46   46   39      1323.303   1490.8361   3.511543 5.038044     258
5   39-47   47   39       715.689    769.9189   0.000000 3.282495     127
6   39-48   48   39      1996.778   2740.1290  11.375186 3.841296      81
  trips16                       geometry
1      68 LINESTRING (-122.4083 37.78...
2      29 LINESTRING (-122.4083 37.78...
3      50 LINESTRING (-122.4083 37.78...
4     163 LINESTRING (-122.4083 37.78...
```

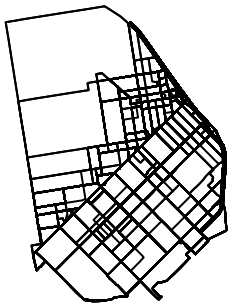
```
5      73 LINESTRING (-122.4083 37.78...
6      56 LINESTRING (-122.4083 37.78...
```

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.

5.3 "*Seeing*" flows

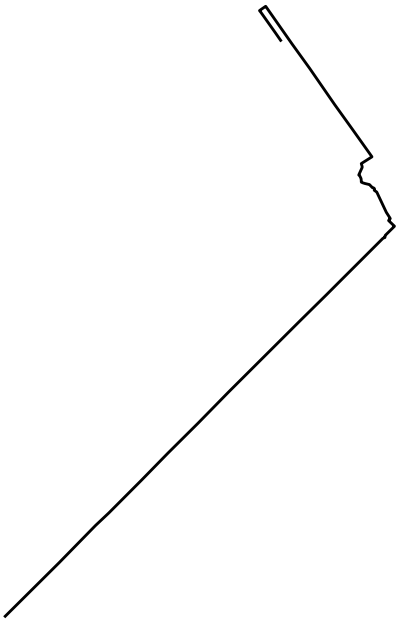
The easiest way to get a quick preview of what the data looks like spatially is to make a simple plot:

```
plot(db$geometry)
```



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:

```
db %>%
  dplyr::filter(orig == 39 & dest == 48) %>%
  ggplot(data = .) +
  geom_sf(color = "black",
          size = 0.1) +
  theme_void()
```



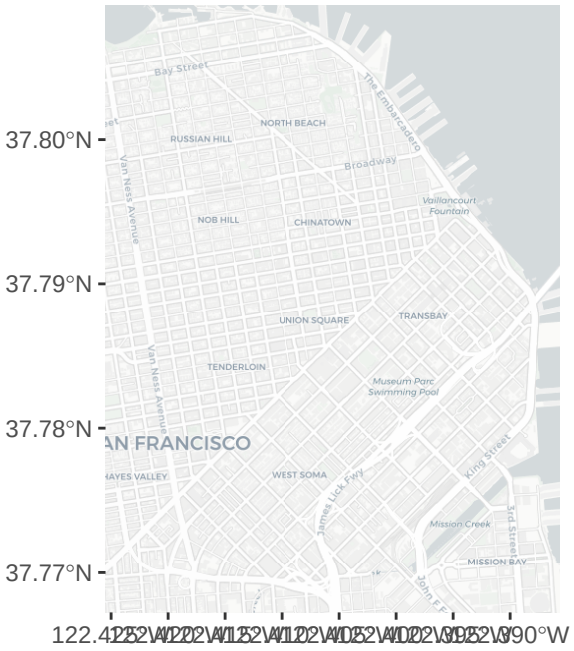
or, for the most popular route, we can:

```
most_pop <- db %>%
  dplyr::filter(trips15 == max(trips15))
```

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.

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 `basemapR`:

```
# create a bounding box
bbox_db <- st_bbox(db)
# download a basemap using ggplot and basemapR
ggplot() +
  base_map(bbox_db, increase_zoom = 2, basemap = "positron") +
  geom_sf(data = db, fill = NA, colour = "transparent")
```



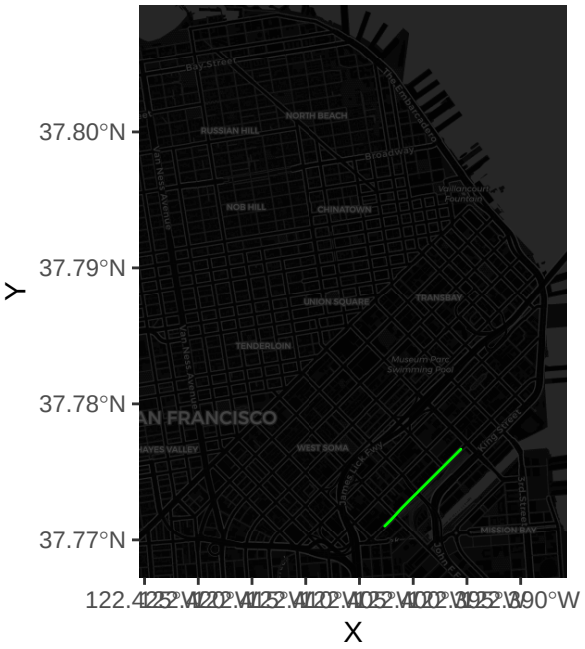
Now to combine tiles and routes, we need to pull out the coordinates that make up each line. For the route example above, this would be:

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

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

```
ggplot() +
  base_map(bbox_db, increase_zoom = 2, basemap = "dark") +
    geom_sf( data = db, fill = NA, colour = "transparent") +
  geom_path( data = xys1,
             aes( x = X, y = Y ),
             #size = 1,
             color = "green",
             lineend ='round'
             )
```

Task
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(st_coordinates(r))
  names(xys) <- c('lon', 'lat')
  # Insert trips and id
  xys['trips'] <- r$trips15
  xys['id'] <- x
  # Append them to `lines`
  lines <- rbind(lines, xys)
```

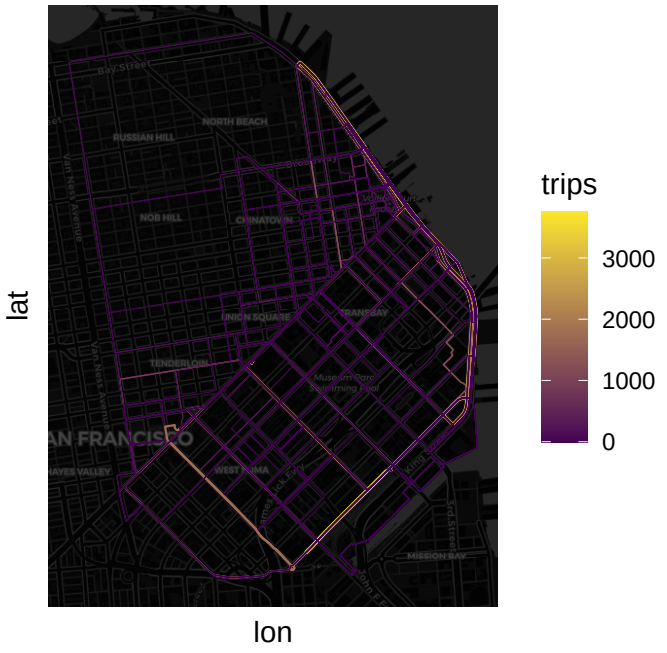
```
}
```

Now we can go on and plot all of them:

```
ggplot() +
  # call basemap
  base_map(bbox_db, increase_zoom = 2, basemap = "dark") +
  geom_sf(data = db, fill = NA, colour = "transparent") +
  # add data
  geom_path( data = lines,
             aes(x=lon, y=lat,
                 group=id
                 ),
             size = 1,
             color = "green",
             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:

```
ggplot() +
  # call basemap
  base_map(bbox_db, increase_zoom = 2, basemap = "dark") +
  geom_sf(data = db, fill = NA, colour = "transparent") +
  # add flow data
  geom_path( data = lines,
             aes(x = lon, y = lat, group = id, colour = trips ),
             size=log1p(lines$trips / max(lines$trips)),
             lineend = 'round'
    ) +
  # create a colour palette
  scale_colour_gradient(
    low='#440154FF', high='#FDE725FF'
  ) +
  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.

5.4 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 (2006) such as predictive performance checks[1] (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 <- db %>% mutate(across(where(is.numeric), scale))

# Reset trips as we want the original version
db_std$trips15 <- db$trips15
db_std$trips16 <- db$trips16

# Reset origin and destination station and express them as factors
db_std$orig <- as.factor(db$orig)
db_std$dest <- as.factor(db$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:

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

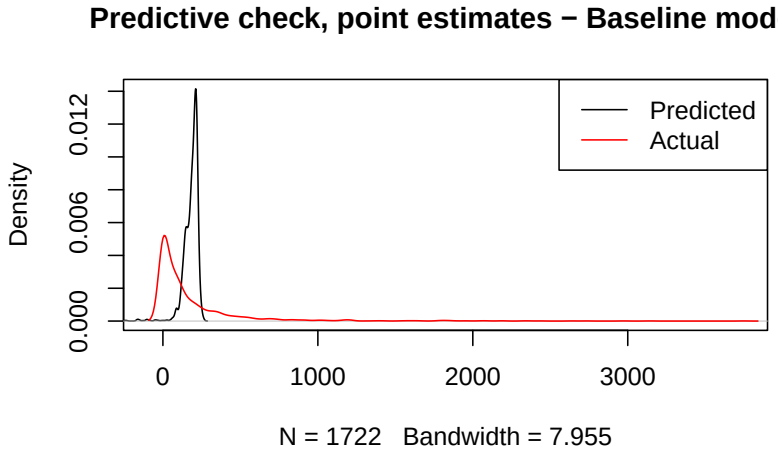
```r
m1 <- lm('trips15 ~ straight_dist + total_up + total_down', data=db_std)
summary(m1)
```

```
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 ***
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:

```r
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.

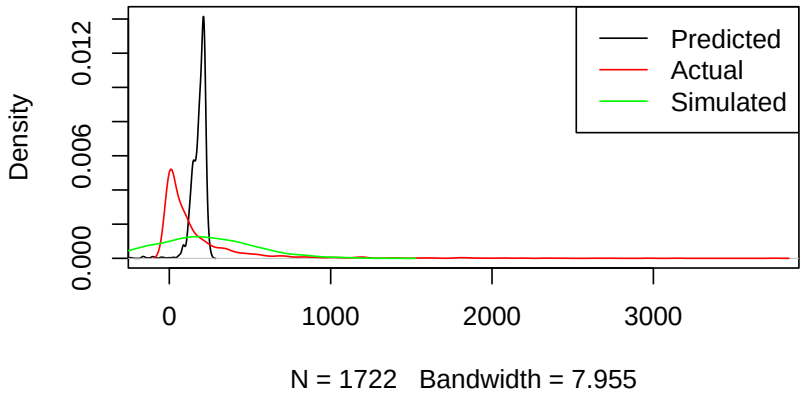
Taking these two sources into consideration means that the black line in the plot above represents only the behaviour of our model we expect if the error term is absent (no predictive uncertainty) and the coefficients are the true estimates (no inferential uncertainty). However, this is not necessarily 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:

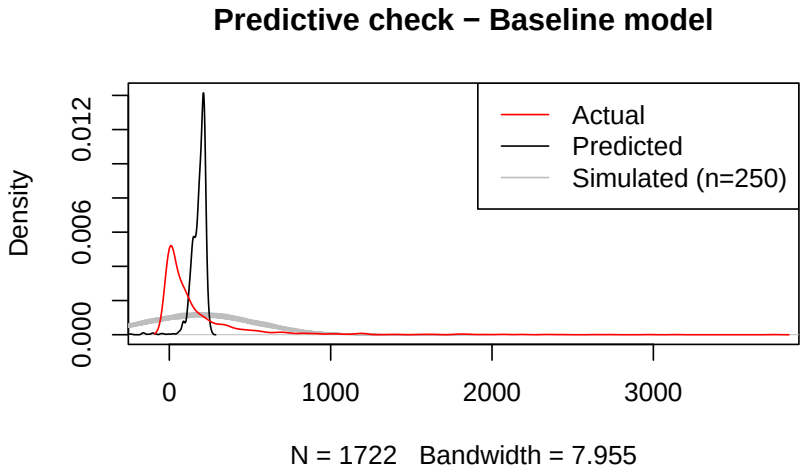
```
new_y <- generate_draw(m1)

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[2] is so large

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

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[3], our 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[4]. 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

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

[4] 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

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[5]; 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:

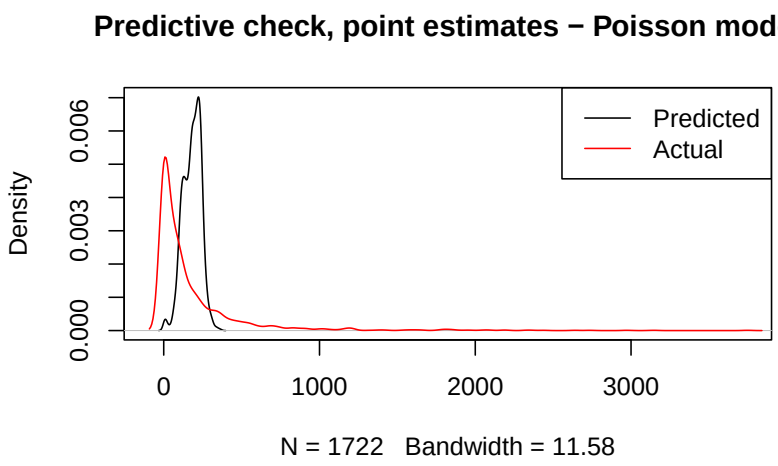
```
m2 <- glm(
  'trips15 ~ straight_dist + total_up + total_down',
  data=db_std,
  family=poisson,
)
```

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 )),
  ylim = c( 0, max(c(
               max( density( m2$fitted.values)$y ),
               max( density(db_std$trips15)$y )
               )
             )
   ),
  col = 'black',
  main = ''
```

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

```
)
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.

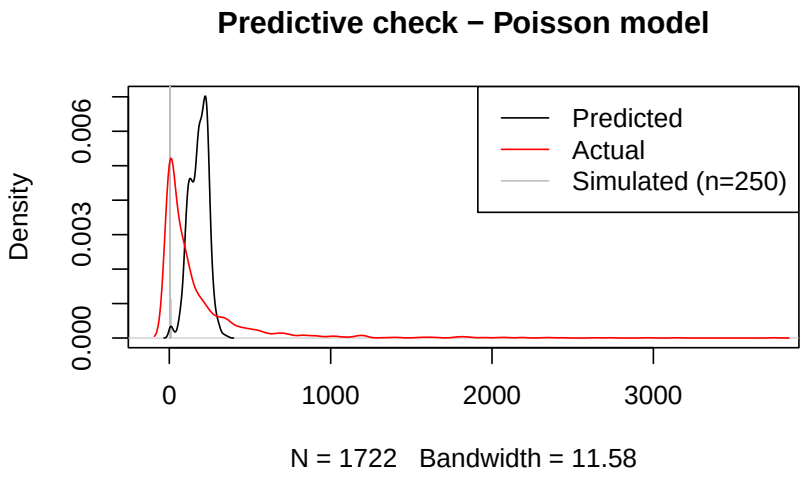
```
generate_draw_poi <- function(m){
  # Set up predictors matrix
  x <- model.matrix( m )
  # Obtain draws of parameters (inferential uncertainty)
  sim_bs <- sim( m, 1 )
  # Predicted value
  xb <- x %*% sim_bs@coef[1, ]
  #xb <- x %*% m$coefficients
  # Transform using the link function
```

```
  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 and 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[6], and the rest 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:

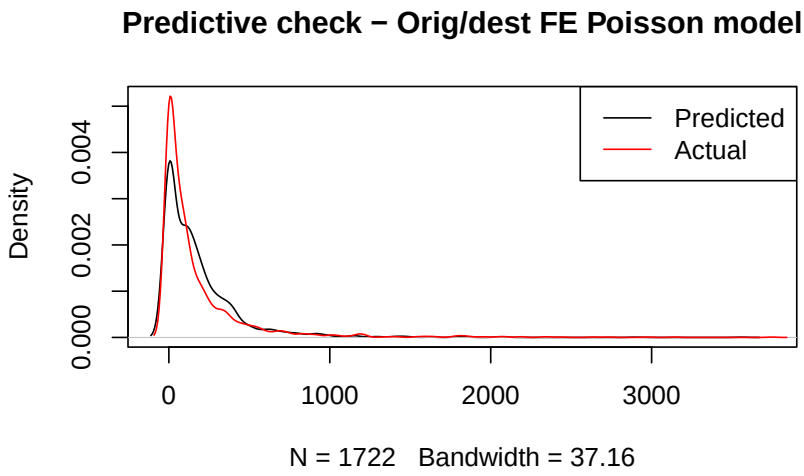
```
m3 <- glm(
  'trips15 ~ straight_dist + total_up + total_down + orig + dest',
  data = db_std,
  family = poisson
)
```

And with our new model, we can have a look at how well it does at predicting the overall number of trips[7]:

[6] In this session, δ_i and δ_j are estimated as independent variables so their estimates are similar to interpret to those in β. An alternative approach could be to model them as random effects in a multilevel framework.

[7] 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.

```
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")
```



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.

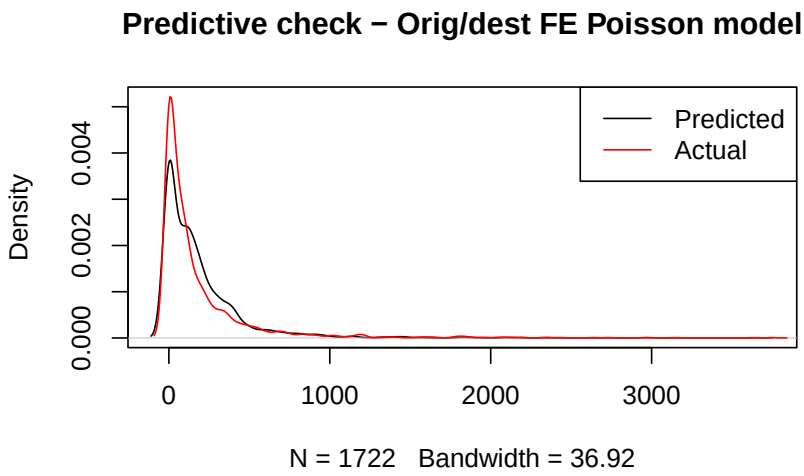
As an exampe of this approach, we can replace the straight distance measurements for more refined ones based on the Google Maps API routes. This is very easy as all we need to do (once the distances have been calculated!) is to swap `straight_dist` for `street_dist`:

```
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 they are not massive. Let's use this example to learn how to interpret coefficients in a Poisson model[8]. 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[9], 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).

5.5 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

[8] See section 6.2 of Gelman and Hill (2006) for a similar treatment of these.

[9] Remember the transformation at the very beginning.

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 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:[^05-flows-11]

```
rmses <- data.frame(
  model=c(
    'OLS',
    'Poisson',
    'Poisson + FE',
    'Poisson + FE + street dist.',
    'Trips-2015'
```

Task
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?

```
  ),
  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.

5.6 Questions

We will be using again the Madrid AirBnb dataset:

```
mad_abb <- st_read('./data/assignment_1_madrid/madrid_abb.gpkg')
```

```
Reading layer `madrid_abb' from data source
  `/Users/franciscorowe/Dropbox/Francisco/uol/teaching/envs453/202324/san/data/assignment_1_mad
  using driver `GPKG'
Simple feature collection with 18399 features and 16 fields
Geometry type: POINT
Dimension:     XY
Bounding box:  xmin: -3.86391 ymin: 40.33243 xmax: -3.556 ymax: 40.56274
Geodetic CRS:  WGS 84
```

The columns to use here are:

- `price_usd`: price expressed in USD
- `log1p_price_usd`: logarithm of the price expressed in USD
- `accommodates`: number of people the property accommodates
- `bathrooms`: number of bathrooms the property includes
- `bedrooms`: number of bedrooms the property includes
- `beds`: number of beds the property includes

With these data at hand, accomplish the following challenges:

1. Set up a baseline regression model where you explain the price of a property as a function of its characteristics:

$$P_i = \alpha + \beta_1 Acc_i + \beta_2 Bath_i + \beta_3 Bedr_i + \beta_4 Beds_i + \epsilon_i$$

2. Fit a parallel model that uses the log of price as dependent variable:

$$\log(P_i) = \alpha + \beta_1 Acc_i + \beta_2 Bath_i + \beta_3 Bedr_i + \beta_4 Beds_i + \epsilon_i$$

3. Perform a predictive check analysis of both models, discussing how they compare, which one you would prefer, and why

6 Spatial Econometrics

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

- Chapter 11 of the GDS Book, by Rey, Arribas-Bel, and Wolf (2023).
- Session III of Dani 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 the Spatial Weights Section of Dani Arribas-Bel (2019).

6.1 Dependencies

We will rely on the following libraries in this section, all of them included in Section **??**:

```
# Data management
library(tidyverse)
# Spatial Data management
library(sf)
# For all your interpolation needs
library(gstat)
# Spatial regression
library(spdep)
```

6.2 Data

To explore ideas in spatial regression, we will the set of Airbnb properties for San Diego (US), borrowed from the "Geographic

Data Science with Python" book (see here for more info on the dataset source). This covers the point location of properties advertised on the Airbnb website in the San Diego region.

Let us load the data:

```
db <- st_read('data/abb_sd/regression_db.geojson')
```

```
Reading layer `regression_db' from data source
  `/Users/franciscorowe/Dropbox/Francisco/uol/teaching/envs453/202324/san/data/abb_sd/regressi
  using driver `GeoJSON'
Simple feature collection with 6110 features and 19 fields
Geometry type: POINT
Dimension:     XY
Bounding box:  xmin: -117.2812 ymin: 32.57349 xmax: -116.9553 ymax: 33.08311
Geodetic CRS:  WGS 84
```

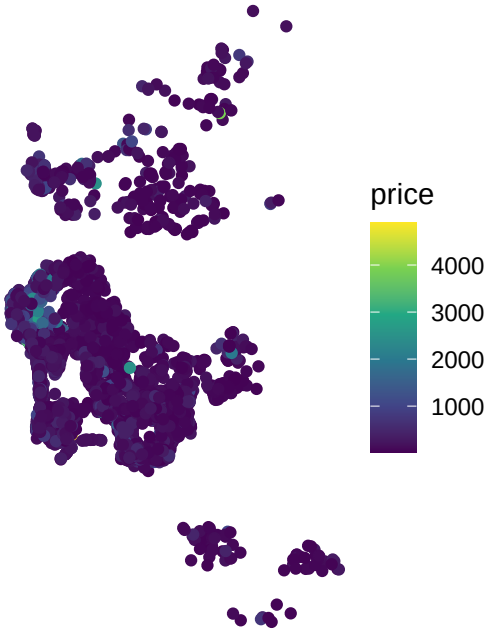
The table contains the followig variables:

```
names(db)
```

```
 [1] "accommodates"       "bathrooms"          "bedrooms"
 [4] "beds"               "neighborhood"       "pool"
 [7] "d2balboa"           "coastal"            "price"
[10] "log_price"          "id"                 "pg_Apartment"
[13] "pg_Condominium"     "pg_House"           "pg_Other"
[16] "pg_Townhouse"       "rt_Entire_home.apt" "rt_Private_room"
[19] "rt_Shared_room"     "geometry"
```

For most of this chapter, we will be exploring determinants and strategies for modelling the price of a property advertised in AirBnb. To get a first taste of what this means, we can create a plot of prices within the area of San Diego:

```
db %>%
  ggplot(aes(color = price)) +
  geom_sf() +
  scale_color_viridis_c() +
  theme_void()
```



6.3 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 interpret the results. By no means is this a formal and complete introduction to regression so, if that is what you are looking for, the first part of Gelman and Hill (2006), in particular chapters 3 and 4, are excellent places to check out.

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 property advertised on AirBnb as a function of some of its characteristics, such as the number of people it accommodates, and how many bathrooms, bedrooms and beds it features. At the individual level, we can express this as:

$$\log(P_i) = \alpha + \beta_1 Acc_i + \beta_2 Bath_i + \beta_3 Bedr_i + \beta_4 Beds_i + \epsilon_i$$

where P_i is the price of house i, Acc_i, $Bath_i$, $Bedr_i$ and $Beds_i$ are the count of people it accommodates, bathrooms, bedrooms and beds that house i has, respectively. The parameters $\beta_{1,2,3,4}$ 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 the "error" and captures elements that influence the price of a house but are not accounted for explicitly. We can also express this relation in matrix form, excluding subindices for i as:

$$\log(P) = \alpha + \beta_1 Acc + \beta_2 Bath + \beta_3 Bedr + \beta_4 Beds + \epsilon$$

where each term can be interpreted in terms of vectors instead of scalars (wit the exception of the parameters $(\alpha, \beta_{1,2,3,4})$, which *are* scalars). Note we are using the logarithm of the price, since this allows us to interpret the coefficients as roughly the percentage change induced by a unit increase in the explanatory variable of the estimate.

Remember a regression can be seen as a multivariate extension of 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[^06-spatial-econometrics-1]. Regression allows you to isolate the distinct effect that a single variable has on the dependent one, once we *control* for those other variables.

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('log_price ~ accommodates + bathrooms + bedrooms + beds', 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 (the log of the price, `log_price`) with a set of

Task
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

explanatory ones (`accommodates`, `bathrooms`, `bedrooms`, `beds`) by using a tilde `~` that is akin to the = symbol in the mathematical equation above. 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 = "log_price ~ accommodates + bathrooms + bedrooms + beds",
    data = db)

Residuals:
    Min      1Q  Median      3Q     Max
-2.8486 -0.3234 -0.0095  0.3023  3.3975

Coefficients:
             Estimate Std. Error t value Pr(>|t|)
(Intercept)  4.018133   0.013947  288.10   <2e-16 ***
accommodates 0.176851   0.005323   33.23   <2e-16 ***
bathrooms    0.150981   0.012526   12.05   <2e-16 ***
bedrooms     0.111700   0.012537    8.91   <2e-16 ***
beds        -0.076974   0.007927   -9.71   <2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 0.5366 on 6105 degrees of freedom
Multiple R-squared:  0.5583,    Adjusted R-squared:  0.558
F-statistic:  1929 on 4 and 6105 DF,  p-value: < 2.2e-16
```

A full detailed explanation of the output is beyond the scope of the chapter, but we will highlight 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. These estimates are the raw equivalent of the correlation coefficient between each explanatory variable and the dependent one,

once the "polluting" effect of the other variables included in the model has been accounted for[1]. Results are as expected for the most part: houses tend to be significantly more expensive if they accommodate more people (an extra person increases the price by 17.7%, approximately), have more bathrooms (15.1%), or bedrooms (11.2%). Perhaps counter intuitively, an extra bed available seems to decrease the price by about -7.7%. However, keep in mind that this is the case, *everything else equal*. Hence, more beds per room and bathroom (ie. a more crowded house) is a bit cheaper.

6.4 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" per se, 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 literature. 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, Overman, and Patacchini (2014).

[1] 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.

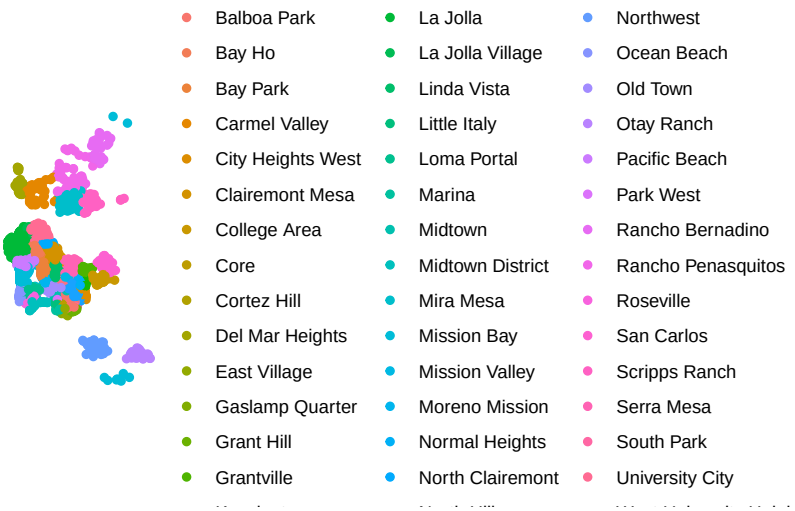
6.5 Spatial heterogeneity

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. 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 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 neighbourhood, as provided by AirBnB, indicating whether a given house is located within such area (`1`) or not (`0`). Neighbourhood membership is expressed on the `neighborhood` column:

```
db %>%
  ggplot(aes(color = neighborhood)) +
  geom_sf() +
  theme_void()
```



Mathematically, we are now fitting the following equation:

$$\log(P_i) = \alpha_r + \beta_1 Acc_i + \beta_2 Bath_i + \beta_3 Bedr_i + \beta_4 Beds_i + \epsilon_i$$

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

Programmatically, we can fit this model with `lm`:

```
# Include `-1` to eliminate the constant term and include a dummy for every area
m2 <- lm(
  'log_price ~ neighborhood + accommodates + bathrooms + bedrooms + beds - 1',
  db
)
summary(m2)
```

```
Call:
lm(formula = "log_price ~ neighborhood + accommodates + bathrooms + bedrooms + beds - 1",
    data = db)

Residuals:
    Min      1Q  Median      3Q     Max
-2.4549 -0.2920 -0.0203  0.2741  3.5323
```

Coefficients:

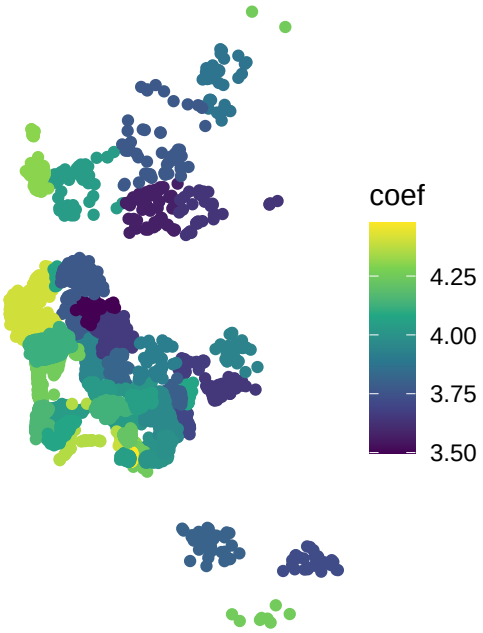
	Estimate	Std. Error	t value	Pr(>|t|)	
neighborhoodBalboa Park	3.994775	0.036539	109.33	<2e-16	***
neighborhoodBay Ho	3.780025	0.086081	43.91	<2e-16	***
neighborhoodBay Park	3.941847	0.055788	70.66	<2e-16	***
neighborhoodCarmel Valley	4.034052	0.062811	64.23	<2e-16	***
neighborhoodCity Heights West	3.698788	0.065502	56.47	<2e-16	***
neighborhoodClairemont Mesa	3.658339	0.051438	71.12	<2e-16	***
neighborhoodCollege Area	3.649859	0.064979	56.17	<2e-16	***
neighborhoodCore	4.433447	0.058864	75.32	<2e-16	***
neighborhoodCortez Hill	4.294790	0.057648	74.50	<2e-16	***
neighborhoodDel Mar Heights	4.300659	0.060912	70.61	<2e-16	***
neighborhoodEast Village	4.241146	0.032019	132.46	<2e-16	***
neighborhoodGaslamp Quarter	4.473863	0.052493	85.23	<2e-16	***
neighborhoodGrant Hill	4.001481	0.058825	68.02	<2e-16	***
neighborhoodGrantville	3.664989	0.080168	45.72	<2e-16	***
neighborhoodKensington	4.073520	0.087322	46.65	<2e-16	***
neighborhoodLa Jolla	4.400145	0.026772	164.36	<2e-16	***
neighborhoodLa Jolla Village	4.066151	0.087263	46.60	<2e-16	***
neighborhoodLinda Vista	3.817940	0.063128	60.48	<2e-16	***
neighborhoodLittle Italy	4.390651	0.052433	83.74	<2e-16	***
neighborhoodLoma Portal	4.034473	0.036173	111.53	<2e-16	***
neighborhoodMarina	4.046133	0.052178	77.55	<2e-16	***
neighborhoodMidtown	4.032038	0.030280	133.16	<2e-16	***
neighborhoodMidtown District	4.356943	0.071756	60.72	<2e-16	***
neighborhoodMira Mesa	3.570523	0.061543	58.02	<2e-16	***
neighborhoodMission Bay	4.251309	0.023318	182.32	<2e-16	***
neighborhoodMission Valley	4.012410	0.083766	47.90	<2e-16	***
neighborhoodMoreno Mission	4.028288	0.063342	63.59	<2e-16	***
neighborhoodNormal Heights	3.791895	0.054730	69.28	<2e-16	***
neighborhoodNorth Clairemont	3.498107	0.076432	45.77	<2e-16	***
neighborhoodNorth Hills	3.959403	0.026823	147.61	<2e-16	***
neighborhoodNorthwest	3.810201	0.078158	48.75	<2e-16	***
neighborhoodOcean Beach	4.152695	0.032352	128.36	<2e-16	***
neighborhoodOld Town	4.127737	0.046523	88.72	<2e-16	***
neighborhoodOtay Ranch	3.722902	0.091633	40.63	<2e-16	***
neighborhoodPacific Beach	4.116749	0.022711	181.27	<2e-16	***
neighborhoodPark West	4.216829	0.050370	83.72	<2e-16	***
neighborhoodRancho Bernadino	3.873962	0.080780	47.96	<2e-16	***
neighborhoodRancho Penasquitos	3.772037	0.068808	54.82	<2e-16	***
neighborhoodRoseville	4.070468	0.065299	62.34	<2e-16	***

```
neighborhoodSan Carlos                3.935042   0.093205   42.22   <2e-16 ***
neighborhoodScripps Ranch             3.641239   0.085190   42.74   <2e-16 ***
neighborhoodSerra Mesa                3.912127   0.066630   58.71   <2e-16 ***
neighborhoodSouth Park                3.987019   0.060141   66.30   <2e-16 ***
neighborhoodUniversity City           3.772504   0.039638   95.17   <2e-16 ***
neighborhoodWest University Heights   4.043161   0.048238   83.82   <2e-16 ***
accommodates                          0.150283   0.005086   29.55   <2e-16 ***
bathrooms                             0.132287   0.011886   11.13   <2e-16 ***
bedrooms                              0.147631   0.011960   12.34   <2e-16 ***
beds                                 -0.074622   0.007405  -10.08   <2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 0.4971 on 6061 degrees of freedom
Multiple R-squared:  0.9904,    Adjusted R-squared:  0.9904
F-statistic: 1.28e+04 on 49 and 6061 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 San Diego as equal, we only derive variation from *within* each postcode. In our particular case, 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. We can visualise this "geography of desirability" by plotting the estimates of each fixed effect on a map:

```r
# Extract neighborhood names from coefficients
nei.names <- m2$coefficients %>%
  as.data.frame() %>%
  row.names() %>%
  str_replace("neighborhood", "")
# Set up as Data Frame
nei.fes <- data.frame(
  coef = m2$coefficients,
  nei = nei.names,
  row.names = nei.names
) %>%
  right_join(
```

```
    db, by = c("nei" = "neighborhood")
)
# Plot
nei.fes %>%
  st_as_sf() %>%
  ggplot(aes(color = coef)) +
  geom_sf() +
  scale_color_viridis_c() +
  theme_void()
```



We can see how neighborhoods in the left (west) tend to have higher prices. What we can't see, but it is represented there if you are familiar with the geography of San Diego, is that the city is bounded by the Pacific ocean on the left, suggesting neighbourhoods by the beach tend to be more expensive.

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 neighbourhood, 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 be estimating is:

$$\log(P_i) = \alpha_r + \beta_{1r} Acc_i + \beta_{2r} Bath_i + \beta_{3r} Bedr_i + \beta_{4r} Beds_i + \epsilon_i$$

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

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
m3 <- lm(
  'log_price ~ (one + accommodates + bathrooms + bedrooms + beds):(neighborhood)',
  db
)
summary(m3)
```

```
Call:
lm(formula = "log_price ~ (one + accommodates + bathrooms + bedrooms + beds):(neighborhood)",
    data = db)

Residuals:
    Min      1Q  Median      3Q     Max
-2.5528 -0.2921 -0.0163  0.2586  3.1874

Coefficients: (1 not defined because of singularities)
                                                Estimate Std. Error t value
(Intercept)                                     4.012160   0.122261  32.816
one:neighborhoodBalboa Park                     0.128350   0.145826   0.880
one:neighborhoodBay Ho                         -0.202575   0.254393  -0.796
one:neighborhoodBay Park                       -0.272843   0.174512  -1.563
one:neighborhoodCarmel Valley                  -0.063356   0.164404  -0.385
one:neighborhoodCity Heights West              -0.096400   0.205758  -0.469
one:neighborhoodClairemont Mesa                -0.639595   0.183891  -3.478
one:neighborhoodCollege Area                   -0.185039   0.207335  -0.892
one:neighborhoodCore                            0.416563   0.223962   1.860
one:neighborhoodCortez Hill                     0.309752   0.193169   1.604
one:neighborhoodDel Mar Heights                 0.259677   0.182046   1.426
one:neighborhoodEast Village                    0.205331   0.147158   1.395
one:neighborhoodGaslamp Quarter                -1.156797   0.282853  -4.090
one:neighborhoodGrant Hill                     -0.077324   0.200580  -0.386
one:neighborhoodGrantville                     -0.355260   0.278718  -1.275
one:neighborhoodKensington                     -0.252743   0.248147  -1.019
one:neighborhoodLa Jolla                        0.380059   0.128014   2.969
one:neighborhoodLa Jolla Village                0.027119   0.291318   0.093
one:neighborhoodLinda Vista                    -0.448116   0.202151  -2.217
one:neighborhoodLittle Italy                    0.384100   0.188630   2.036
one:neighborhoodLoma Portal                     0.014552   0.148157   0.098
one:neighborhoodMarina                         -0.549055   0.198350  -2.768
one:neighborhoodMidtown                         0.071392   0.134620   0.530
one:neighborhoodMidtown District               -0.180003   0.219627  -0.820
one:neighborhoodMira Mesa                      -0.596573   0.205262  -2.906
one:neighborhoodMission Bay                     0.399054   0.128693   3.101
one:neighborhoodMission Valley                 -0.249279   0.288450  -0.864
one:neighborhoodMoreno Mission                  0.268901   0.222999   1.206
one:neighborhoodNormal Heights                 -0.253917   0.210697  -1.205
one:neighborhoodNorth Clairemont               -0.404436   0.221559  -1.825
```

one:neighborhoodNorth Hills	-0.103473	0.136994	-0.755
one:neighborhoodNorthwest	0.198074	0.284402	0.696
one:neighborhoodOcean Beach	0.192554	0.136493	1.411
one:neighborhoodOld Town	-0.062746	0.157153	-0.399
one:neighborhoodOtay Ranch	-0.358005	0.232826	-1.538
one:neighborhoodPacific Beach	0.113854	0.129413	0.880
one:neighborhoodPark West	0.154992	0.170398	0.910
one:neighborhoodRancho Bernadino	-0.019347	0.194468	-0.099
one:neighborhoodRancho Penasquitos	-0.401257	0.163315	-2.457
one:neighborhoodRoseville	0.120010	0.182676	0.657
one:neighborhoodSan Carlos	-0.589123	0.284463	-2.071
one:neighborhoodScripps Ranch	-0.026842	0.215965	-0.124
one:neighborhoodSerra Mesa	-0.283949	0.207741	-1.367
one:neighborhoodSouth Park	-0.181786	0.226828	-0.801
one:neighborhoodUniversity City	-0.298526	0.161655	-1.847
one:neighborhoodWest University Heights	NA	NA	NA
accommodates:neighborhoodBalboa Park	0.097052	0.024632	3.940
accommodates:neighborhoodBay Ho	-0.017961	0.089598	-0.200
accommodates:neighborhoodBay Park	0.155715	0.063773	2.442
accommodates:neighborhoodCarmel Valley	0.151990	0.049440	3.074
accommodates:neighborhoodCity Heights West	0.060303	0.063416	0.951
accommodates:neighborhoodClairemont Mesa	0.260414	0.050425	5.164
accommodates:neighborhoodCollege Area	0.084693	0.053125	1.594
accommodates:neighborhoodCore	0.106661	0.039204	2.721
accommodates:neighborhoodCortez Hill	0.217358	0.037615	5.779
accommodates:neighborhoodDel Mar Heights	0.075776	0.061335	1.235
accommodates:neighborhoodEast Village	0.162155	0.020465	7.924
accommodates:neighborhoodGaslamp Quarter	0.307007	0.053766	5.710
accommodates:neighborhoodGrant Hill	0.278114	0.056238	4.945
accommodates:neighborhoodGrantville	0.124025	0.075028	1.653
accommodates:neighborhoodKensington	0.063298	0.123887	0.511
accommodates:neighborhoodLa Jolla	0.119142	0.015162	7.858
accommodates:neighborhoodLa Jolla Village	0.207085	0.111627	1.855
accommodates:neighborhoodLinda Vista	0.172101	0.076208	2.258
accommodates:neighborhoodLittle Italy	0.230108	0.034432	6.683
accommodates:neighborhoodLoma Portal	0.090991	0.034762	2.618
accommodates:neighborhoodMarina	0.552162	0.045914	12.026
accommodates:neighborhoodMidtown	0.227617	0.023854	9.542
accommodates:neighborhoodMidtown District	0.179989	0.052007	3.461
accommodates:neighborhoodMira Mesa	0.157603	0.082351	1.914
accommodates:neighborhoodMission Bay	0.085754	0.013455	6.373

accommodates:neighborhoodMission Valley	0.099273	0.134176	0.740
accommodates:neighborhoodMoreno Mission	0.145824	0.060631	2.405
accommodates:neighborhoodNormal Heights	0.237177	0.053060	4.470
accommodates:neighborhoodNorth Clairemont	0.188941	0.084060	2.248
accommodates:neighborhoodNorth Hills	0.143390	0.022069	6.497
accommodates:neighborhoodNorthwest	0.022166	0.069235	0.320
accommodates:neighborhoodOcean Beach	0.113995	0.022845	4.990
accommodates:neighborhoodOld Town	0.239088	0.042208	5.665
accommodates:neighborhoodOtay Ranch	-0.021530	0.083481	-0.258
accommodates:neighborhoodPacific Beach	0.153379	0.015339	9.999
accommodates:neighborhoodPark West	0.224593	0.045581	4.927
accommodates:neighborhoodRancho Bernadino	0.004730	0.067968	0.070
accommodates:neighborhoodRancho Penasquitos	0.070627	0.060353	1.170
accommodates:neighborhoodRoseville	0.112686	0.060090	1.875
accommodates:neighborhoodSan Carlos	-0.140714	0.087777	-1.603
accommodates:neighborhoodScripps Ranch	0.087078	0.035562	2.449
accommodates:neighborhoodSerra Mesa	0.190589	0.075803	2.514
accommodates:neighborhoodSouth Park	0.206833	0.068093	3.038
accommodates:neighborhoodUniversity City	0.067637	0.030097	2.247
accommodates:neighborhoodWest University Heights	0.124850	0.056034	2.228
bathrooms:neighborhoodBalboa Park	0.014875	0.073585	0.202
bathrooms:neighborhoodBay Ho	0.305028	0.225044	1.355
bathrooms:neighborhoodBay Park	0.065043	0.120545	0.540
bathrooms:neighborhoodCarmel Valley	0.205847	0.065237	3.155
bathrooms:neighborhoodCity Heights West	-0.129465	0.183389	-0.706
bathrooms:neighborhoodClairemont Mesa	0.418033	0.176111	2.374
bathrooms:neighborhoodCollege Area	-0.012095	0.183917	-0.066
bathrooms:neighborhoodCore	0.464974	0.186634	2.491
bathrooms:neighborhoodCortez Hill	-0.206708	0.101521	-2.036
bathrooms:neighborhoodDel Mar Heights	0.319046	0.096172	3.317
bathrooms:neighborhoodEast Village	0.312707	0.067204	4.653
bathrooms:neighborhoodGaslamp Quarter	1.459299	0.207669	7.027
bathrooms:neighborhoodGrant Hill	0.040405	0.155867	0.259
bathrooms:neighborhoodGrantville	0.028466	0.234973	0.121
bathrooms:neighborhoodKensington	0.146585	0.188987	0.776
bathrooms:neighborhoodLa Jolla	0.242760	0.026778	9.066
bathrooms:neighborhoodLa Jolla Village	-0.296748	0.326257	-0.910
bathrooms:neighborhoodLinda Vista	0.170333	0.143441	1.187
bathrooms:neighborhoodLittle Italy	0.035532	0.111044	0.320
bathrooms:neighborhoodLoma Portal	0.188467	0.082511	2.284
bathrooms:neighborhoodMarina	0.316509	0.219948	1.439

bathrooms:neighborhoodMidtown	-0.037771	0.032333	-1.168
bathrooms:neighborhoodMidtown District	0.623515	0.197674	3.154
bathrooms:neighborhoodMira Mesa	0.215939	0.185982	1.161
bathrooms:neighborhoodMission Bay	0.176976	0.035418	4.997
bathrooms:neighborhoodMission Valley	-0.009144	0.405637	-0.023
bathrooms:neighborhoodMoreno Mission	-0.208248	0.210241	-0.991
bathrooms:neighborhoodNormal Heights	0.017303	0.201580	0.086
bathrooms:neighborhoodNorth Clairemont	-0.102553	0.212254	-0.483
bathrooms:neighborhoodNorth Hills	0.098449	0.064065	1.537
bathrooms:neighborhoodNorthwest	-0.503592	0.255300	-1.973
bathrooms:neighborhoodOcean Beach	0.082659	0.059578	1.387
bathrooms:neighborhoodOld Town	0.087046	0.096513	0.902
bathrooms:neighborhoodOtay Ranch	-0.219480	0.341525	-0.643
bathrooms:neighborhoodPacific Beach	0.116105	0.041006	2.831
bathrooms:neighborhoodPark West	0.198550	0.107749	1.843
bathrooms:neighborhoodRancho Bernadino	0.365674	0.147799	2.474
bathrooms:neighborhoodRancho Penasquitos	0.320715	0.110007	2.915
bathrooms:neighborhoodRoseville	-0.053420	0.102487	-0.521
bathrooms:neighborhoodSan Carlos	0.267797	0.214540	1.248
bathrooms:neighborhoodScripps Ranch	-0.132692	0.169684	-0.782
bathrooms:neighborhoodSerra Mesa	0.381620	0.197305	1.934
bathrooms:neighborhoodSouth Park	0.087490	0.196486	0.445
bathrooms:neighborhoodUniversity City	0.138271	0.125198	1.104
bathrooms:neighborhoodWest University Heights	0.042035	0.122097	0.344
bedrooms:neighborhoodBalboa Park	0.183347	0.065309	2.807
bedrooms:neighborhoodBay Ho	0.241200	0.170471	1.415
bedrooms:neighborhoodBay Park	0.214111	0.117198	1.827
bedrooms:neighborhoodCarmel Valley	-0.073928	0.127976	-0.578
bedrooms:neighborhoodCity Heights West	0.272272	0.176859	1.539
bedrooms:neighborhoodClairemont Mesa	-0.046650	0.122263	-0.382
bedrooms:neighborhoodCollege Area	-0.055707	0.108958	-0.511
bedrooms:neighborhoodCore	-0.069929	0.108896	-0.642
bedrooms:neighborhoodCortez Hill	0.313288	0.101770	3.078
bedrooms:neighborhoodDel Mar Heights	0.111554	0.097639	1.143
bedrooms:neighborhoodEast Village	0.088071	0.056654	1.555
bedrooms:neighborhoodGaslamp Quarter	-0.297570	0.090336	-3.294
bedrooms:neighborhoodGrant Hill	0.091300	0.114501	0.797
bedrooms:neighborhoodGrantville	0.221461	0.208950	1.060
bedrooms:neighborhoodKensington	0.450822	0.269641	1.672
bedrooms:neighborhoodLa Jolla	0.166725	0.035423	4.707
bedrooms:neighborhoodLa Jolla Village	0.845318	0.545842	1.549

bedrooms:neighborhoodLinda Vista	0.096413	0.171751	0.561
bedrooms:neighborhoodLittle Italy	0.126794	0.080939	1.567
bedrooms:neighborhoodLoma Portal	0.172480	0.067268	2.564
bedrooms:neighborhoodMarina	-0.323085	0.137046	-2.357
bedrooms:neighborhoodMidtown	0.172711	0.044927	3.844
bedrooms:neighborhoodMidtown District	0.088181	0.134514	0.656
bedrooms:neighborhoodMira Mesa	-0.029149	0.206406	-0.141
bedrooms:neighborhoodMission Bay	0.197484	0.032472	6.082
bedrooms:neighborhoodMission Valley	0.393256	0.342308	1.149
bedrooms:neighborhoodMoreno Mission	-0.081003	0.158933	-0.510
bedrooms:neighborhoodNormal Heights	0.199088	0.104514	1.905
bedrooms:neighborhoodNorth Clairemont	0.176509	0.183702	0.961
bedrooms:neighborhoodNorth Hills	0.283826	0.047159	6.018
bedrooms:neighborhoodNorthwest	0.411641	0.160558	2.564
bedrooms:neighborhoodOcean Beach	0.220345	0.055039	4.003
bedrooms:neighborhoodOld Town	0.209031	0.099966	2.091
bedrooms:neighborhoodOtay Ranch	0.772212	0.354230	2.180
bedrooms:neighborhoodPacific Beach	0.077503	0.036735	2.110
bedrooms:neighborhoodPark West	-0.127891	0.110497	-1.157
bedrooms:neighborhoodRancho Bernadino	-0.209594	0.160210	-1.308
bedrooms:neighborhoodRancho Penasquitos	0.107599	0.138183	0.779
bedrooms:neighborhoodRoseville	0.395961	0.157406	2.516
bedrooms:neighborhoodSan Carlos	0.239713	0.150420	1.594
bedrooms:neighborhoodScripps Ranch	-0.045580	0.102624	-0.444
bedrooms:neighborhoodSerra Mesa	0.202756	0.163958	1.237
bedrooms:neighborhoodSouth Park	0.080548	0.112861	0.714
bedrooms:neighborhoodUniversity City	0.124090	0.120386	1.031
bedrooms:neighborhoodWest University Heights	0.011105	0.095137	0.117
beds:neighborhoodBalboa Park	0.013322	0.045740	0.291
beds:neighborhoodBay Ho	0.037070	0.133299	0.278
beds:neighborhoodBay Park	0.011091	0.108863	0.102
beds:neighborhoodCarmel Valley	0.086393	0.076151	1.135
beds:neighborhoodCity Heights West	0.076520	0.132072	0.579
beds:neighborhoodClairemont Mesa	-0.163512	0.090244	-1.812
beds:neighborhoodCollege Area	0.228165	0.084147	2.712
beds:neighborhoodCore	-0.080211	0.078673	-1.020
beds:neighborhoodCortez Hill	-0.039031	0.042752	-0.913
beds:neighborhoodDel Mar Heights	-0.021375	0.071330	-0.300
beds:neighborhoodEast Village	-0.174883	0.040811	-4.285
beds:neighborhoodGaslamp Quarter	-0.064063	0.088752	-0.722
beds:neighborhoodGrant Hill	-0.178920	0.067398	-2.655

```
beds:neighborhoodGrantville                  -0.003795   0.126551  -0.030
beds:neighborhoodKensington                  -0.005431   0.175724  -0.031
beds:neighborhoodLa Jolla                    -0.104906   0.022470  -4.669
beds:neighborhoodLa Jolla Village            -0.385895   0.315094  -1.225
beds:neighborhoodLinda Vista                  0.032215   0.073896   0.436
beds:neighborhoodLittle Italy                -0.145613   0.029817  -4.884
beds:neighborhoodLoma Portal                 -0.013033   0.059097  -0.221
beds:neighborhoodMarina                      -0.210771   0.113913  -1.850
beds:neighborhoodMidtown                     -0.143254   0.036531  -3.921
beds:neighborhoodMidtown District            -0.132655   0.090595  -1.464
beds:neighborhoodMira Mesa                    0.097629   0.113225   0.862
beds:neighborhoodMission Bay                 -0.061315   0.017727  -3.459
beds:neighborhoodMission Valley               0.073187   0.235760   0.310
beds:neighborhoodMoreno Mission               0.195918   0.116748   1.678
beds:neighborhoodNormal Heights              -0.185827   0.062062  -2.994
beds:neighborhoodNorth Clairemont            -0.060468   0.188694  -0.320
beds:neighborhoodNorth Hills                 -0.113150   0.032768  -3.453
beds:neighborhoodNorthwest                    0.116746   0.131088   0.891
beds:neighborhoodOcean Beach                 -0.048848   0.037433  -1.305
beds:neighborhoodOld Town                    -0.158066   0.077352  -2.043
beds:neighborhoodOtay Ranch                   0.071485   0.259986   0.275
beds:neighborhoodPacific Beach               -0.025064   0.023165  -1.082
beds:neighborhoodPark West                   -0.049627   0.063967  -0.776
beds:neighborhoodRancho Bernadino             0.270293   0.092164   2.933
beds:neighborhoodRancho Penasquitos           0.011344   0.091156   0.124
beds:neighborhoodRoseville                   -0.085205   0.087365  -0.975
beds:neighborhoodSan Carlos                   0.550169   0.167389   3.287
beds:neighborhoodScripps Ranch                0.219496   0.122439   1.793
beds:neighborhoodSerra Mesa                  -0.293629   0.189261  -1.551
beds:neighborhoodSouth Park                  -0.020083   0.109799  -0.183
beds:neighborhoodUniversity City              0.124224   0.062916   1.974
beds:neighborhoodWest University Heights      0.143248   0.096150   1.490
                                            Pr(>|t|)
(Intercept)                                  < 2e-16 ***
one:neighborhoodBalboa Park                 0.378810
one:neighborhoodBay Ho                      0.425887
one:neighborhoodBay Park                    0.117997
one:neighborhoodCarmel Valley               0.699978
one:neighborhoodCity Heights West           0.639437
one:neighborhoodClairemont Mesa             0.000509 ***
one:neighborhoodCollege Area                0.372180
```

```
one:neighborhoodCore                          0.062939 .
one:neighborhoodCortez Hill                   0.108871
one:neighborhoodDel Mar Heights               0.153795
one:neighborhoodEast Village                  0.162975
one:neighborhoodGaslamp Quarter               4.38e-05 ***
one:neighborhoodGrant Hill                    0.699880
one:neighborhoodGrantville                    0.202493
one:neighborhoodKensington                    0.308471
one:neighborhoodLa Jolla                      0.003001 **
one:neighborhoodLa Jolla Village              0.925835
one:neighborhoodLinda Vista                   0.026679 *
one:neighborhoodLittle Italy                  0.041769 *
one:neighborhoodLoma Portal                   0.921764
one:neighborhoodMarina                        0.005656 **
one:neighborhoodMidtown                       0.595908
one:neighborhoodMidtown District              0.412485
one:neighborhoodMira Mesa                     0.003670 **
one:neighborhoodMission Bay                   0.001939 **
one:neighborhoodMission Valley                0.387513
one:neighborhoodMoreno Mission                0.227928
one:neighborhoodNormal Heights                0.228202
one:neighborhoodNorth Clairemont              0.067990 .
one:neighborhoodNorth Hills                   0.450091
one:neighborhoodNorthwest                     0.486169
one:neighborhoodOcean Beach                   0.158381
one:neighborhoodOld Town                      0.689710
one:neighborhoodOtay Ranch                    0.124188
one:neighborhoodPacific Beach                 0.379016
one:neighborhoodPark West                     0.363079
one:neighborhoodRancho Bernadino              0.920756
one:neighborhoodRancho Penasquitos            0.014041 *
one:neighborhoodRoseville                     0.511237
one:neighborhoodSan Carlos                    0.038402 *
one:neighborhoodScripps Ranch                 0.901090
one:neighborhoodSerra Mesa                    0.171727
one:neighborhoodSouth Park                    0.422917
one:neighborhoodUniversity City               0.064843 .
one:neighborhoodWest University Heights             NA
accommodates:neighborhoodBalboa Park          8.24e-05 ***
accommodates:neighborhoodBay Ho               0.841129
accommodates:neighborhoodBay Park             0.014647 *
```

```
accommodates:neighborhoodCarmel Valley          0.002120 **
accommodates:neighborhoodCity Heights West      0.341685
accommodates:neighborhoodClairemont Mesa        2.49e-07 ***
accommodates:neighborhoodCollege Area           0.110943
accommodates:neighborhoodCore                   0.006535 **
accommodates:neighborhoodCortez Hill            7.92e-09 ***
accommodates:neighborhoodDel Mar Heights        0.216716
accommodates:neighborhoodEast Village           2.74e-15 ***
accommodates:neighborhoodGaslamp Quarter        1.18e-08 ***
accommodates:neighborhoodGrant Hill             7.81e-07 ***
accommodates:neighborhoodGrantville             0.098376 .
accommodates:neighborhoodKensington             0.609420
accommodates:neighborhoodLa Jolla               4.61e-15 ***
accommodates:neighborhoodLa Jolla Village       0.063624 .
accommodates:neighborhoodLinda Vista            0.023963 *
accommodates:neighborhoodLittle Italy           2.56e-11 ***
accommodates:neighborhoodLoma Portal            0.008879 **
accommodates:neighborhoodMarina                  < 2e-16 ***
accommodates:neighborhoodMidtown                 < 2e-16 ***
accommodates:neighborhoodMidtown District       0.000542 ***
accommodates:neighborhoodMira Mesa              0.055697 .
accommodates:neighborhoodMission Bay            1.99e-10 ***
accommodates:neighborhoodMission Valley         0.459407
accommodates:neighborhoodMoreno Mission         0.016199 *
accommodates:neighborhoodNormal Heights         7.97e-06 ***
accommodates:neighborhoodNorth Clairemont       0.024632 *
accommodates:neighborhoodNorth Hills            8.84e-11 ***
accommodates:neighborhoodNorthwest              0.748867
accommodates:neighborhoodOcean Beach            6.21e-07 ***
accommodates:neighborhoodOld Town               1.54e-08 ***
accommodates:neighborhoodOtay Ranch             0.796490
accommodates:neighborhoodPacific Beach           < 2e-16 ***
accommodates:neighborhoodPark West              8.56e-07 ***
accommodates:neighborhoodRancho Bernadino       0.944521
accommodates:neighborhoodRancho Penasquitos     0.241960
accommodates:neighborhoodRoseville              0.060803 .
accommodates:neighborhoodSan Carlos             0.108968
accommodates:neighborhoodScripps Ranch          0.014369 *
accommodates:neighborhoodSerra Mesa             0.011955 *
accommodates:neighborhoodSouth Park             0.002396 **
accommodates:neighborhoodUniversity City        0.024659 *
```

```
accommodates:neighborhoodWest University Heights 0.025910 *
bathrooms:neighborhoodBalboa Park                 0.839806
bathrooms:neighborhoodBay Ho                      0.175338
bathrooms:neighborhoodBay Park                    0.589509
bathrooms:neighborhoodCarmel Valley               0.001611 **
bathrooms:neighborhoodCity Heights West           0.480243
bathrooms:neighborhoodClairemont Mesa             0.017643 *
bathrooms:neighborhoodCollege Area                0.947567
bathrooms:neighborhoodCore                        0.012753 *
bathrooms:neighborhoodCortez Hill                 0.041784 *
bathrooms:neighborhoodDel Mar Heights             0.000914 ***
bathrooms:neighborhoodEast Village                3.34e-06 ***
bathrooms:neighborhoodGaslamp Quarter             2.35e-12 ***
bathrooms:neighborhoodGrant Hill                  0.795468
bathrooms:neighborhoodGrantville                  0.903578
bathrooms:neighborhoodKensington                  0.437993
bathrooms:neighborhoodLa Jolla                     < 2e-16 ***
bathrooms:neighborhoodLa Jolla Village            0.363095
bathrooms:neighborhoodLinda Vista                 0.235088
bathrooms:neighborhoodLittle Italy                0.748992
bathrooms:neighborhoodLoma Portal                 0.022399 *
bathrooms:neighborhoodMarina                      0.150200
bathrooms:neighborhoodMidtown                     0.242779
bathrooms:neighborhoodMidtown District            0.001617 **
bathrooms:neighborhoodMira Mesa                   0.245659
bathrooms:neighborhoodMission Bay                 6.00e-07 ***
bathrooms:neighborhoodMission Valley              0.982016
bathrooms:neighborhoodMoreno Mission              0.321960
bathrooms:neighborhoodNormal Heights              0.931598
bathrooms:neighborhoodNorth Clairemont            0.628997
bathrooms:neighborhoodNorth Hills                 0.124417
bathrooms:neighborhoodNorthwest                   0.048594 *
bathrooms:neighborhoodOcean Beach                 0.165371
bathrooms:neighborhoodOld Town                    0.367140
bathrooms:neighborhoodOtay Ranch                  0.520477
bathrooms:neighborhoodPacific Beach               0.004650 **
bathrooms:neighborhoodPark West                   0.065420 .
bathrooms:neighborhoodRancho Bernadino            0.013384 *
bathrooms:neighborhoodRancho Penasquitos          0.003566 **
bathrooms:neighborhoodRoseville                   0.602221
bathrooms:neighborhoodSan Carlos                  0.211993
```

```
bathrooms:neighborhoodScripps Ranch              0.434248
bathrooms:neighborhoodSerra Mesa                 0.053141 .
bathrooms:neighborhoodSouth Park                 0.656138
bathrooms:neighborhoodUniversity City            0.269455
bathrooms:neighborhoodWest University Heights    0.730652
bedrooms:neighborhoodBalboa Park                 0.005011 **
bedrooms:neighborhoodBay Ho                      0.157151
bedrooms:neighborhoodBay Park                    0.067764 .
bedrooms:neighborhoodCarmel Valley               0.563507
bedrooms:neighborhoodCity Heights West           0.123740
bedrooms:neighborhoodClairemont Mesa             0.702806
bedrooms:neighborhoodCollege Area                0.609182
bedrooms:neighborhoodCore                        0.520791
bedrooms:neighborhoodCortez Hill                 0.002091 **
bedrooms:neighborhoodDel Mar Heights             0.253288
bedrooms:neighborhoodEast Village                0.120105
bedrooms:neighborhoodGaslamp Quarter             0.000993 ***
bedrooms:neighborhoodGrant Hill                  0.425266
bedrooms:neighborhoodGrantville                  0.289243
bedrooms:neighborhoodKensington                  0.094590 .
bedrooms:neighborhoodLa Jolla                    2.58e-06 ***
bedrooms:neighborhoodLa Jolla Village            0.121520
bedrooms:neighborhoodLinda Vista                 0.574579
bedrooms:neighborhoodLittle Italy                0.117274
bedrooms:neighborhoodLoma Portal                 0.010370 *
bedrooms:neighborhoodMarina                      0.018431 *
bedrooms:neighborhoodMidtown                     0.000122 ***
bedrooms:neighborhoodMidtown District            0.512137
bedrooms:neighborhoodMira Mesa                   0.887701
bedrooms:neighborhoodMission Bay                 1.26e-09 ***
bedrooms:neighborhoodMission Valley              0.250670
bedrooms:neighborhoodMoreno Mission              0.610302
bedrooms:neighborhoodNormal Heights              0.056842 .
bedrooms:neighborhoodNorth Clairemont            0.336669
bedrooms:neighborhoodNorth Hills                 1.87e-09 ***
bedrooms:neighborhoodNorthwest                   0.010377 *
bedrooms:neighborhoodOcean Beach                 6.32e-05 ***
bedrooms:neighborhoodOld Town                    0.036570 *
bedrooms:neighborhoodOtay Ranch                  0.029299 *
bedrooms:neighborhoodPacific Beach               0.034919 *
bedrooms:neighborhoodPark West                   0.247150
```

```
bedrooms:neighborhoodRancho Bernadino            0.190841
bedrooms:neighborhoodRancho Penasquitos          0.436205
bedrooms:neighborhoodRoseville                   0.011911 *
bedrooms:neighborhoodSan Carlos                  0.111073
bedrooms:neighborhoodScripps Ranch               0.656951
bedrooms:neighborhoodSerra Mesa                  0.216272
bedrooms:neighborhoodSouth Park                  0.475443
bedrooms:neighborhoodUniversity City             0.302694
bedrooms:neighborhoodWest University Heights     0.907079
beds:neighborhoodBalboa Park                     0.770871
beds:neighborhoodBay Ho                          0.780950
beds:neighborhoodBay Park                        0.918856
beds:neighborhoodCarmel Valley                   0.256631
beds:neighborhoodCity Heights West               0.562357
beds:neighborhoodClairemont Mesa                 0.070055 .
beds:neighborhoodCollege Area                    0.006717 **
beds:neighborhoodCore                            0.307987
beds:neighborhoodCortez Hill                     0.361302
beds:neighborhoodDel Mar Heights                 0.764449
beds:neighborhoodEast Village                    1.86e-05 ***
beds:neighborhoodGaslamp Quarter                 0.470435
beds:neighborhoodGrant Hill                      0.007960 **
beds:neighborhoodGrantville                      0.976078
beds:neighborhoodKensington                      0.975344
beds:neighborhoodLa Jolla                        3.10e-06 ***
beds:neighborhoodLa Jolla Village                0.220739
beds:neighborhoodLinda Vista                     0.662889
beds:neighborhoodLittle Italy                    1.07e-06 ***
beds:neighborhoodLoma Portal                     0.825463
beds:neighborhoodMarina                          0.064324 .
beds:neighborhoodMidtown                         8.90e-05 ***
beds:neighborhoodMidtown District                0.143173
beds:neighborhoodMira Mesa                       0.388579
beds:neighborhoodMission Bay                     0.000546 ***
beds:neighborhoodMission Valley                  0.756247
beds:neighborhoodMoreno Mission                  0.093375 .
beds:neighborhoodNormal Heights                  0.002763 **
beds:neighborhoodNorth Clairemont                0.748634
beds:neighborhoodNorth Hills                     0.000558 ***
beds:neighborhoodNorthwest                       0.373186
beds:neighborhoodOcean Beach                     0.191957
```

```
beds:neighborhoodOld Town                          0.041051 *
beds:neighborhoodOtay Ranch                        0.783359
beds:neighborhoodPacific Beach                     0.279326
beds:neighborhoodPark West                         0.437885
beds:neighborhoodRancho Bernadino                  0.003373 **
beds:neighborhoodRancho Penasquitos                0.900966
beds:neighborhoodRoseville                         0.329465
beds:neighborhoodSan Carlos                        0.001019 **
beds:neighborhoodScripps Ranch                     0.073072 .
beds:neighborhoodSerra Mesa                        0.120848
beds:neighborhoodSouth Park                        0.854878
beds:neighborhoodUniversity City                   0.048378 *
beds:neighborhoodWest University Heights           0.136320
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 0.478 on 5885 degrees of freedom
Multiple R-squared:  0.6622,    Adjusted R-squared:  0.6494
F-statistic: 51.51 on 224 and 5885 DF,  p-value: < 2.2e-16
```

This allows us to get a separate constant term and estimate of the impact of each variable *for every neighborhood*

6.6 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 values. 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 number of bathrooms it has but, if we take number of bathrooms as a proxy for size and status, also whether it is surrounded by other houses with many bathrooms. 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)*[2] 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.

[2] If you need to refresh your knowledge on spatial weight matrices, check Block E of Dani Arribas-Bel (2019); Chapter 4 of Rey, Arribas-Bel, and Wolf (2023); or the Spatial Weights Section of Rowe (2022).

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 $k = 50$ to get a good degree of variation and sensible results.

```
# Create knn list of each house
hnn <- db %>%
  st_coordinates() %>%
  as.matrix() %>%
  knearneigh(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: 6110
Number of nonzero links: 305500
Percentage nonzero weights: 0.8183306
Average number of links: 50
Non-symmetric neighbours list

Weights style: W
Weights constants summary:
     n       nn   S0       S1       S2
W 6110 37332100 6110 220.5032 24924.44
```

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 an

AirBnb property in San Diego can be explained using information about its own characteristics, and the neighbourhood it belongs to. However, we can hypothesise that the price of a house is also affected by the characteristics of the houses surrounding it. Considering it as a proxy for larger and more luxurious houses, we will use the number of bathrooms of neighboring houses as an additional explanatory variable. This represents the most straightforward way to introduce spatial dependence in a regression, by considering not only a given explanatory variable, but also its spatial lag.

In our example case, in addition to including the number of bathrooms of the property, we will include its spatial lag. In other words, we will be saying that it is not only the number of bathrooms in a house but also that of the surrounding properties that helps explain the final price at which a house is advertised for. Mathematically, this implies estimating the following model:

$$\log(P_i) = \alpha + \beta_1 Acc_i + \beta_2 Bath_i + \beta_3 Bedr_i + \beta_4 Beds_i + \beta_5 Bath_{lag-i} + \epsilon_i$$

Let us first compute the spatial lag of `bathrooms`:

```
db$w_bathrooms <- lag.listw(hknn, db$bathrooms)
```

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

```
m5 <- lm(
  'log_price ~ accommodates + bedrooms + beds + bathrooms + w_bathrooms',
  db
)

summary(m5)
```

```
Call:
lm(formula = "log_price ~ accommodates + bedrooms + beds + bathrooms + w_bathrooms",
    data = db)
```

```
Residuals:
    Min      1Q  Median      3Q     Max
-2.8869 -0.3243 -0.0206  0.2931  3.5132

Coefficients:
              Estimate Std. Error t value Pr(>|t|)
(Intercept)   3.579448   0.032337 110.692   <2e-16 ***
accommodates  0.173226   0.005233  33.100   <2e-16 ***
bedrooms      0.103116   0.012327   8.365   <2e-16 ***
beds         -0.075071   0.007787  -9.641   <2e-16 ***
bathrooms     0.117268   0.012507   9.376   <2e-16 ***
w_bathrooms   0.353021   0.023572  14.976   <2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 0.5271 on 6104 degrees of freedom
Multiple R-squared:  0.574, Adjusted R-squared:  0.5736
F-statistic:  1645 on 5 and 6104 DF,  p-value: < 2.2e-16
```

As we can see, the lag is not only significative and positive, but its effect seems to be even larger that that of the property itself. Taken literally, this implies that the average number of bathrooms in AirBnb's nearby has a larger effect on the final price of a given AirBnb than its own number of bathrooms. There are several ways to interpret this. One is that, if we take the spatial lag of bathrooms, as we said above, to be a proxy for the types of houses surrounding a property, this is probably a better predictor of how wealthy an area is than the number of bathrooms of a single property, which is more variable. If we also assume that the area where an AirBnb is located has a bigger effect on price than the number of bathrooms, we can start seeing an answer to the apparent puzzle.

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:

$$\log(P_i) = \alpha + \rho \log(P_i) + \beta_1 Acc_i + \beta_2 Bath_i + \beta_3 Bedr_i + \beta_4 Beds_i + \epsilon_i$$

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:

$$\log(P_i) = \alpha + \beta_1 Acc_i + \beta_2 Bath_i + \beta_3 Bedr_i + \beta_4 Beds_i + u_i$$

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

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.

6.7 Predicting house prices

So far, we have seen how to 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 its characteristics.

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

$$\log(\bar{P}_i) = \bar{\alpha} + \bar{\beta}_1 Acc_i^* + \bar{\beta}_2 Bath_i^* + \bar{\beta}_3 Bedr_i^* + \bar{\beta}_4 Beds_i^*$$

where $\log(\bar{P}_i)$ 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 relatively streamlined in R. Suppose we are given data for a new house which is to be put on the AirBnb platform. We know it accommodates four people, and has two bedrooms, three beds, and one bathroom. We also know that the surrounding properties have, on average, 1.5 bathrooms. Let us record the data first:

```
new.house <- data.frame(
  accommodates = 4,
  bedrooms = 2,
  beds = 3,
  bathrooms = 1,
```

```
  w_bathrooms = 1.5
)
```

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

```
new.price <- predict(m5, new.house)
new.price
```

```
       1
4.900168
```

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
134.3123
```

According to our model, the house would be worth $134.3123448.

6.8 Questions

We will be using again the Madrid AirBnb dataset:

```
mad_abb <- st_read('./data/assignment_1_madrid/madrid_abb.gpkg')
```

```
Reading layer `madrid_abb' from data source
  `/Users/franciscorowe/Dropbox/Francisco/uol/teaching/envs453/202324/san/data/assignment_1_mad
  using driver `GPKG'
Simple feature collection with 18399 features and 16 fields
Geometry type: POINT
Dimension:     XY
Bounding box:  xmin: -3.86391 ymin: 40.33243 xmax: -3.556 ymax: 40.56274
Geodetic CRS:  WGS 84
```

```
colnames(mad_abb)
```

```
 [1] "price"           "price_usd"       "log1p_price_usd" "accommodates"
 [5] "bathrooms_text"  "bathrooms"       "bedrooms"        "beds"
 [9] "neighbourhood"   "room_type"       "property_type"   "WiFi"
[13] "Coffee"          "Gym"             "Parking"         "km_to_retiro"
[17] "geom"
```

In addition to those we have already seen, the columns to use here are:

- `neighbourhood`: a column with the name of the neighbourhood in which the property is located

With this at hand, answer the following questions:

1. Fit a baseline model with only property characteristics explaining the log of price

$$\log(P_i) = \alpha + \beta_1 Acc_i + \beta_2 Bath_i + \beta_3 Bedr_i + \beta_4 Beds_i + \epsilon_i$$

2. Augment the model with fixed effects at the neighbourhood level

$$\log(P_i) = \alpha_r + \beta_1 Acc_i + \beta_2 Bath_i + \beta_3 Bedr_i + \beta_4 Beds_i + \epsilon_i$$

3. [Optional] Augment the model with spatial regimes at the neighbourhood level:

$$\log(P_i) = \alpha_r + \beta_{r1} Acc_i + \beta_{r2} Bath_i + \beta_{r3} Bedr_i + \beta_{r4} Beds_i + \epsilon_{ri}$$

4. Fit a model that augments the baseline in 1. with the spatial lag of a variable you consider interesting. Motivate this choice. Note that to complete this, you will need to also generate a spatial weights matrix.

In each instance, provide a brief interpretation (no more thana few lines for each) that demonstrates your understanding of theunderlying concepts behind your approach.

7 Multilevel Modelling - Part 1

This chapter provides an introduction to multi-level data structures and multi-level modelling and draws on the following references:

- Gelman and Hill (2006) 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.
- Multilevel Modelling (n.d.) is an useful online resource on multilevel modelling and is free!

7.1 Dependencies

We will use the following dependencies

```
# Data manipulation, transformation and visualisation
library(tidyverse)
# Nice tables
library(kableExtra)
# Spatial data manitulation
library(sf)
# Thematic maps
library(tmap)
# Colour palettes
library(viridis)
# Fitting multilevel models
library(lme4)
# Tools for extracting information generated by lme4
library(merTools)
# Exportable regression tables
library(jtools)
```

7.2 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.[1]

Let us read the data:

```
# read data
oa_shp <- st_read("data/mlm/OA.shp")
```

```
Reading layer `OA' from data source
  `/Users/franciscorowe/Dropbox/Francisco/uol/teaching/envs453/202324/san/data/mlm/OA.shp'
  using driver `ESRI Shapefile'
Simple feature collection with 1584 features and 19 fields
Geometry type: MULTIPOLYGON
Dimension:     XY
Bounding box:  xmin: 332390.2 ymin: 379748.5 xmax: 345636 ymax: 397980.1
Projected CRS: Transverse_Mercator
```

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

```
# attach data frame
attach(oa_shp)

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

[1] Read the file in R by executing `read_tsv("data/mlm/readme.txt")` . Ensure the library `readr` is installed before running `read_tsv`.

```
Simple feature collection with 6 features and 19 fields
Geometry type: MULTIPOLYGON
Dimension:     XY
Bounding box:  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...
```

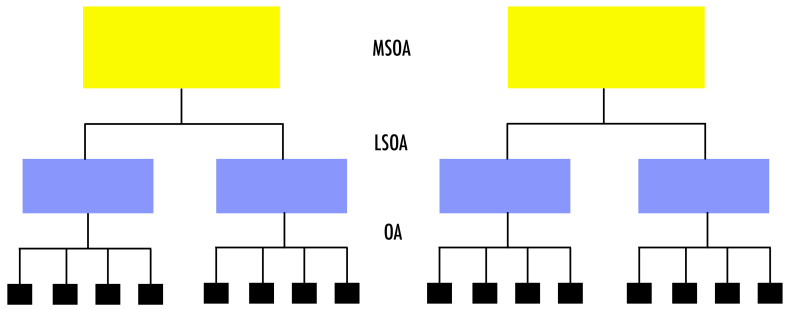


Figure 7.1: Fig. 1. Data Structure.

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
- 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

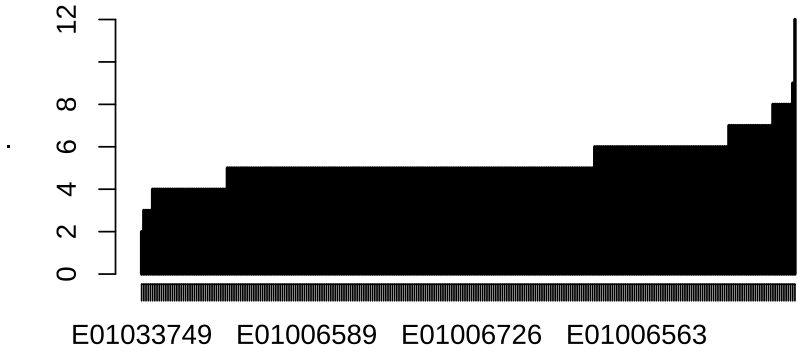
```
# mean of nested OAs within LSOAs and MSOAs
lsoa_cd %>% table() %>%
  mean() %>%
  round(, 2)
```

```
[1] 5
```

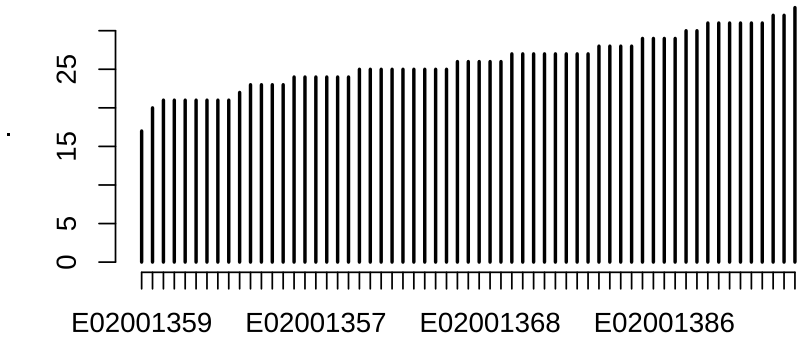
```
msoa_cd %>% table() %>%
  mean() %>%
  round(, 2)
```

```
[1] 26
```

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




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

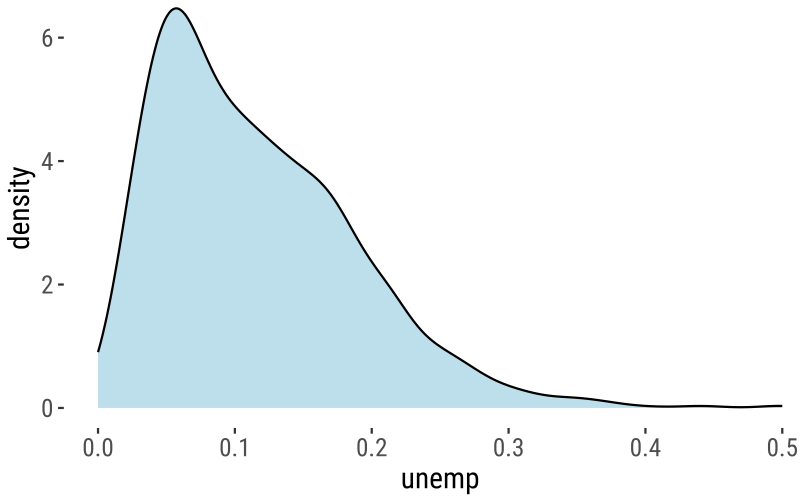


7.3 Modelling

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 (2006).

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_plot_tufte()
```

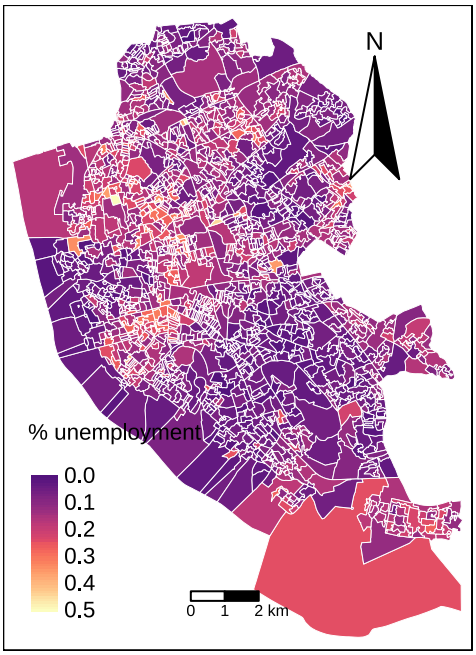


```
summary(unemp)
```

```
   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), sty
  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:

```r
# 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: POLYGON
Dimension:     XY
Bounding box:  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 POLYGON ((335804.9 389421.6...
2  E00033008 345 0.2636364 POLYGON ((335080 388528, 33...
3  E00033074 299 0.2075472 POLYGON ((336947.3 387766.7...
4  E00033075 254 0.2288136 POLYGON ((336753.6 387465.2...
5  E00033080 197 0.2647059 POLYGON ((338196 387079, 33...
6  E00033103 298 0.2148148 POLYGON ((340484 385429.6, ...
7  E00033116 190 0.2156863 POLYGON ((341960.7 386422.1...
8  E00033134 190 0.2674419 POLYGON ((337137 393089.6, ...
9  E00033137 289 0.2661290 POLYGON ((337363.8 392122.4...
10 E00033138 171 0.3561644 POLYGON ((337481.5 392166.2...
```

7.3.1 Baseline Linear Regression Model

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

```r
# 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?

7.3.1.1 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.

7.3.1.2 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.

7.4 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 replacing `(1 | lsoa_cd)` for `(1 | msoa_cd/lsoa_cd)` to allow the intercept to also vary by MSOAs and account for the nesting structure of LSOAs within MSOAs. In multilevel modelling, these types of models are formally known as *nested random effects* and

they differ from a different set of models known as *crossed random effects*.

::: column-margin ::: callout-note A crossed random effect model in our example would be expressed as follows:

```
unemp ~ 1 + (1 | lsoa_cd) + (1 | msoa_cd)
```

::: ::: column-margin

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

# estimates
summary(model3)
```

```
Linear mixed model fit by REML ['lmerMod']
Formula: unemp ~ 1 + (1 | msoa_cd/lsoa_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:msoa_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:msoa_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.

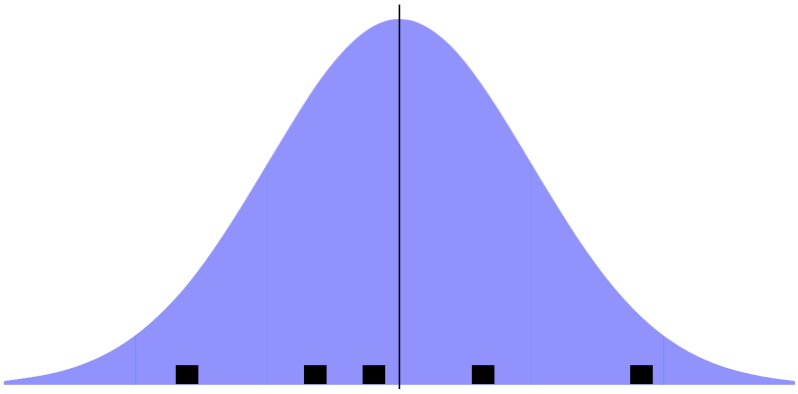


Figure 7.2: Fig. 2. Variation of observations around their level 1 group mean.

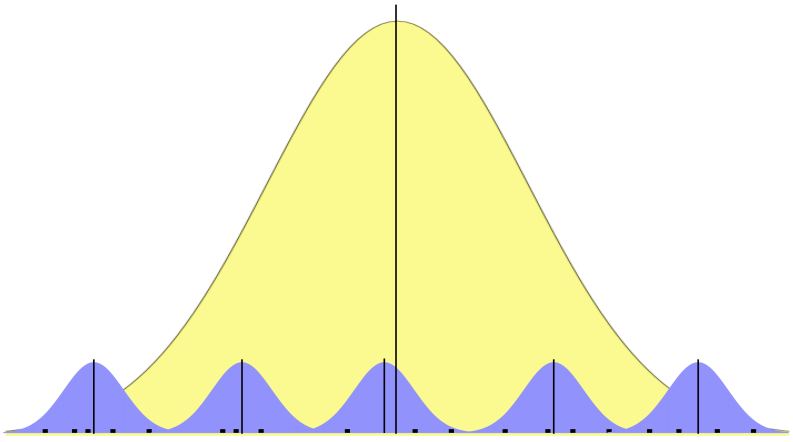


Figure 7.3: Fig. 3. Variation of level 1 group mean around their level 2 group mean.

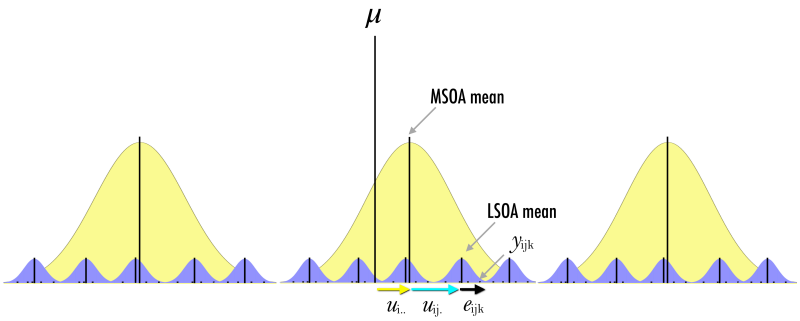


Figure 7.4: Fig. 4. Grand mean.

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:

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

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 | msoa_cd/lsoa_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:msoa_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:msoa_cd, 298; msoa_cd, 61

Fixed effects:
            Estimate Std. Error t value
(Intercept) 0.115288   0.006187   18.64
```

7.4.1 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 `0.1152881`.

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`, `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.

7.4.2 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.[2] For our case, the VPC at the MSOA level indicates that 38% of the variation in percentage of unemployed resident population across OAs can be explained by differences across MSOAs.

Task What is the VPC at the LSOA level?

[2]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 same group is 0.1.