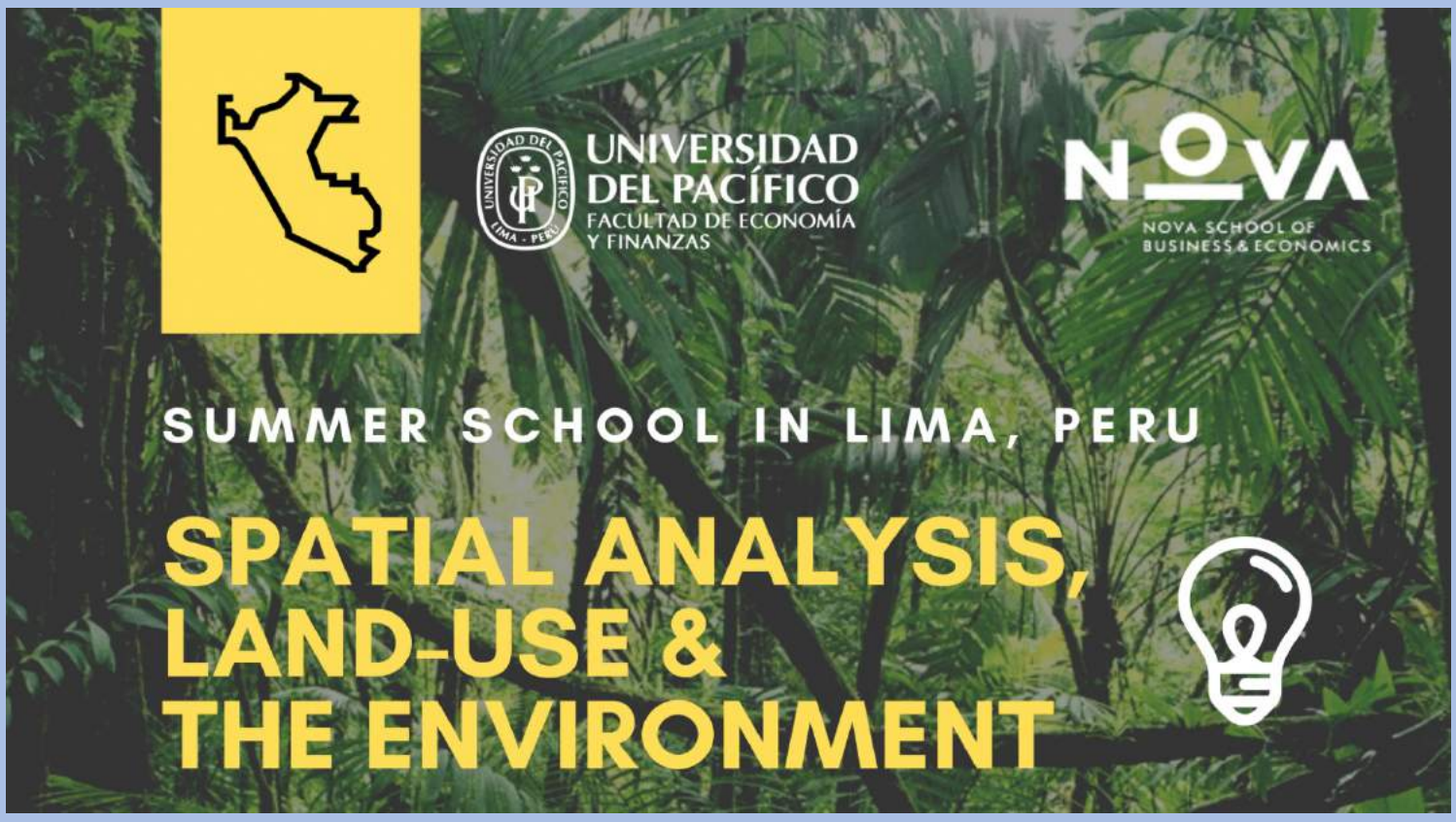


Applications of Spatial Analysis with Spatial Econometrics: *Spatial Urban-Environmental Modelling In R*

Tuesday, January 28th, 2020

Jacob L. Macdonald
Jacob.Macdonald@liverpool.ac.uk

Version Date: 27_01_2020.1

COURSE OUTLINE

Preamble

R is a powerful programming language that is one of the most commonly used for the study and analysis of data, statistics, or econometrics. This language (software) is open-source and available to be downloaded without charge. For this reason, the use of ***R*** has become extremely popular in both academia and in industry.

The open-source nature of the software means that anyone is able to contribute to writing new packages and functions to analyse all types of problems. As new packages and functions were added ***R*** became easier to use to solve complex problems, and so grew in popularity. With close to 10,000 packages, ***R*** has a huge repository of functions and algorithms to work on issues related to data cleaning and management, data visualization, statistical inferences, spatial and temporal modelling, machine learning, etc.

It is important to understand the difference between the software of ***R*** and of ***R***Studio. ***R*** is the underlying software which runs all of the computations and code. ***R***Studio however is the user interface that we will be working in directly. By using ***R***Studio we are able to write and save multiple large *scripts* of code that run a selection of cleaning, statistics or modelling functions which can then be grouped according to research project. This lets us write up and comment on the code with notes associated to a given project and then save our workings for a later session. The alternative is line-by-line coding in ***R***, which is unsuitable for large projects with multi-part code and scripts.

The workshop is split into parts: the first focused on the basics of working with ***R*** objects and our data specifically, the second focuses on understanding spatial variables and pollution interpolation, and the third is the estimation of advanced spatial models which control for underlying spatial dependence – including the running and interpreting of diagnostics. We will be working with a database on housing locations and prices which we will combine with other spatial data to develop and build sophisticated spatial models for the real estate market of Lisbon, Portugal.

The goal is not, however, to teach the intricate details about how to code and work efficiently in ***R*** - for many of you, this will be the first time working in a coding language. Learning the basics of ***R***, or data science programming for other languages (*e.g.* Python, Matlab, Stata), could fill an entire course on its own. For this reason, the finalized code and functions are provided to you and we will work through it together section by section. As we go, part of the emphasis will of course be on what the relevant codes and algorithms for doing GIS in ***R*** are (technical aspects) while further emphasis will be put on how to interpret the results and diagnostics we obtain for robust spatial modelling (econometric aspects).

The tools, methods and code/ functions that we use in this workshop are applicable to any data (which is in a similar format to ours). Using what we learn in this workshop, you will be able to update this code provided to run similar spatial modelling to new data.

Below you will find information on how to download and install both ***R*** and ***RStudio*** onto your personal computer. Both software are easy and free to install/ set up. Firstly, what is the difference between ***R*** and ***RStudio***? ***R*** is the software and programming language which runs all the statistical and data analysis and computations. ***RStudio*** however is an *Integrated Development Environment (IDE)* which provides us with a user interface along with tools and features to make the most out of the computations done in R and allows the researcher to organize and better conduct analysis.

A common example used to highlight the differences would be like considering ***R*** to be engine of a car while ***RStudio*** is the dashboard containing all the information and tools for the driver.

Downloading R

For Mac Users

- From any internet browser, go to https://www.r-project.org/
- Click on the "download *R*" link under the "Getting Started" section.
- This will take you to a list of potential CRAN Mirrors - a list of servers which host the files necessary to download in order for *R* to run. Here you can choose the mirror which is closest to you (one of the mirrors from Portugal).
- On the next page we can select "Download *R* for (Mac) OS X" where you will be taken to a new page which has the most recent release of the *R* to download.
- As of the time of writing, the most recent release for ***R*** is version 3.6.
- Download the most recent *.pkg* file your download folder.
- Click the .pkg file to open and follow the installation instructions provided.

For Windows Users

- From any internet browser, go to https://www.r-project.org/
- Click on the "download *R*" link under the "Getting Started" section.
- This will take you to a list of potential CRAN Mirrors - a list of servers which host the files necessary to download in order for *R* to run. Here you can choose the mirror which is closest to you (one of the mirrors from Portugal).
- On the next page we can select "Download *R* for Windows" where you will be taken to a new page which has the most recent release of the *R* to download.
- As of the time of writing, the most recent release for ***R*** is version 3.6 – please ensure this is the version installed on your machine for this lab.
- Click on "**install R for the first time**" to begin the download process - save the .exe file in your download folder.
- Open the .exe file to open and follow the installation instructions provided.

Downloading RStudio

For Mac Users

- From any internet browser, go to https://www.rstudio.com/
- Choose the "Download ***R****Studio*" option.
- Select the free "***R****Studio* Desktop" version, which will bring you to a list of the most up to date version of ***R****Studio* for different operating systems.
- Download the recommended file for the Mac OS X operating system. As of the time of writing, the most recent version is ***R****Studio* 1.1.463.
- Download the .dmg file into your download folders and click to open and follow the installation instructions provided.

For Windows Users

- From any internet browser, go to https://www.rstudio.com/
- Choose the "Download ***R****Studio*" option.
- Select the free "***R****Studio* Desktop" version, which will bring you to a list of the most up to date version of ***R****Studio* for different operating systems.
- Download the recommended file for the Windows operating system. As of the time of writing, the most recent version is ***R****Studio* 1.1.463.
- Download the .exe file into your download folders and click to open and follow the installation instructions provided.

And finally, some general notes and tips to keep handy as we work through ***R***.

- Make sure to save your work often as you are editing your code, adding notes, new functions, etc. Check your battery to make sure it will not run out during the session (if on a personal laptop).
- Keep your coding scripts tidy, and use **#** at the beginning of a line to start a comment (**#** before anything in ***R*** means that what follows will not be run). Be liberal with using comments to ask yourself questions to answer later, remind yourself of how to calculate a variable, give title sections and explain what each set of functions is accomplishing – write your code in an easy to read format so that anyone without any background could pick it up and understand your process and estimation.
- If you are unsure how a function or a package works, or what objects you need to insert into the function, you can type **`??<FUNCTION/PACKAGE NAME>,`** and you will get the help and reference file for the package.
- If you are stuck with how to code something, create a variable, run a model, etc. use Google, Stackoverflow, R Forums – many people have had the same questions before!

Section 1 - Working and Database Management in R(Studio)

Lesson 00: Setting The Working Directory and Reading Packages

We first need to define the working directory and packages we will be using for this course. The working directory is where we will store all of the files and outputs from this ***R*** session.

> Below we use the **`paste0()`** function to define a character vector specifying the file path-name where our working directory is, which we name **`wd`**. We define (**`<-`**) this here so that we do not have to keep writing out the long path-name each time we want to read or write a file.

If we have downloaded the course folder to the Desktop, we can use the following codes to define the working directory. Since the naming convention of pathnames can differ between different operating systems (OS) (Windows, Mac, Linux), the different general conventions are specified below.

```
# Setting the path on a Windows OS
> wd <- paste0("C:/Users/", Sys.info()[[7]], "/Desktop/Spatial Environmental
  Modelling")


# Setting the path on a Mac OS
> wd <- paste0("/Users/", Sys.info()[[7]], "/Desktop/Spatial Environmental
  Modelling")


# Setting the path on a Linux OS
> wd <- paste0("/home/", Sys.info()[[7]], "/Desktop/Spatial Environmental
  Modelling")


setwd(wd)
```

> The **`Sys.info()`** function prints out all the system information for your specific computer. The 7th element of this block of information (which we retrieve using the **`[[7]]`** index) provides us with the username for your specific computer.

Concatenated (pasted) together, we get **`"C:/Users/<USERNAME>/Desktop/Spatial Modelling"`** which is the path-name of the folder for where all our files are stored.

Using this path string, **`wd`**, we set the directory using the **`setwd()`** function.

Next we have to install all the packages needed.

> A comprehensive list of all ***R*** packages, with brief description of its use, can be found at the following link. Depending on the type of analysis you wish to do, there is likely an ***R*** package available to help:
>
> https://cran.r-project.org/web/packages/available_packages_by_name.html
>
> With so many moving components of **R** (**R**, **RStudio**, Packages), it is crucial to make sure things are compatible and up to date. It is not uncommon to get errors in installing different versions of packages and different dependencies.
>
> For this lab we will be using version 3.6 of **R** (opening **RStudio** for the first time will show you the version number at the header).
>
> If ever you get package errors, the first step is to check the documentation of the package online, check the version of **R** and **RStudio**, and make sure that everything is compatible.

```
# install.packages(c("sp", "spdep", "rgeos", "rgdal", "ggplot2", "geosphere",
#   "gstat", "raster", "car", "lmtest", "tseries", "sandwich", "Matrix",
#   "stargazer"))


> library(sp)
> library(spdep)
> library(rgeos)
> library(rgdal)
> library(ggplot2)
> library(geosphere)
> library(gstat)
> library(raster)
> library(car)
> library(lmtest)
> library(tseries)
> library(sandwich)
> library(Matrix)
> library(stargazer)
```

Lesson 01: Read in Housing Dataset

Let's now read in the csv (a variation of excel) file of our data. This database is a random sample of 400 observations representing dwellings across Lisbon, Portugal, with the relevant price, structural and local environmental attributes, and latitude and longitude (GPS) coordinates.

Here, our database (data frame, table, etc.) has one row for every observation with a number of columns each representing the different variables already available for the data. All observations have been cleaned so the database is complete and without any missing values – sometimes a significant amount of the work has to be dedicated to cleaning the database before any analysis can be done.

The respective code to generate the range of GIS environmental variables for this lab are included, although they have been pre-generated for modelling purposes.

To read in the database we use the **`read.csv()`** function. The file is located in the path-name previously specified by **wd**, in the *Data* folder, and is titled *housing_sample.csv*.

i.e. we are reading in the file from:

```
"C:/Users/<USERNAME>/Desktop/Spatial Modelling/Data/housing_sample.csv"
```

It is always important to make sure that the text in the data is read correctly - especially when using data from Latin languages, since accents or other special characters may be read differently on different machines. In this data, we have already cleaned the variables and removed any accents. If accents do exist in the data and are not being read correctly, then try different **`fileEncoding={latin1 or UTF-8}`** parameters to get the correct reading. As a general rule of thumb, **`UTF-8`** tends to read encoding well on Windows PC while **`latin1`** tends to read encodings well on Mac OS X.

The database has the following variables available:

`latitude|longitude`.....GPS coordinates
`price`.....Price in Euros
`area`.....Size of the dwelling in square meters
`new`.....Dummy variable (0, 1) for whether the dwelling is new
`pool`.....Dummy variable for whether the dwelling has a pool
`parking`.....Dummy variable for whether the dwelling has its own parking
`fireplace`.....Dummy variable for whether the dwelling has a fireplace
`dwindows`.....Dummy variable for whether the dwelling has double windows
`cabletv`.....Dummy variable for whether the dwelling has cable television
`garden`.....Dummy variable for whether the dwelling has a garden
`aircond`.....Dummy variable for whether the dwelling has air conditioning
`view`.....Dummy variable for whether the dwelling has a view of the Tagus
`elevator`.....Dummy variable for whether the dwelling has an elevator
`balcony`.....Dummy variable for whether the dwelling has a balcony
`pantry`.....Dummy variable for whether the dwelling has a pantry
`closet`.....Dummy variable for whether the dwelling has a built in closet

baixa.....Dummy variable for whether the dwelling is in the main CBD

freguesia.....Categorical variable denoting the *freguesia* (neighbourhood)

Pbldg.res.....Percent residential buildings in the census tract

Pbldg.non.....Percent non-residential buildings in the census tract

Ppop.65.....Percent of population over 65 in the census tract

Ppop.secondary.....Percent of population with secondary education in the census tract

Ppop.higher.....Percent of population with higher education

Ppop.empl.....Percent employed population

Ppop.unemp.....Percent unemployed population

pop.ha.....Population density

bldg.ha.....Building density

dist.baixa.....Distance to the main CBD

near.park.....Distance to the nearest park (open space)

avgD.park.....Average distance to all parks (open space)

near.shopping.....Distance to nearest shopping centre

avgD.shopping.....Average distance to all shopping centres

park.100.....Number of parks in 100 meters

park.500.....Number of parks in 500 meters

shopping.100.....Number of shopping centres in 100 meters

shopping.500.....Number of shopping centres in 500 meters

dist.tagus.....Distance to the Tagus riverfront

```
> housing.sample <- read.csv(paste0(wd, "/Data/housing_sample.csv"),
  fileEncoding="latin1", stringsAsFactors=T)
```

We can get a summary of the database variables using the **summary()** function and learn the dimensions of it using **dim()**. We need to make sure that our numeric variables are all indeed the *numeric class*, and our categorical variables are the *factor class*.

```
> names(housing.sample)
# ... [1]

> dim(housing.sample)
# [1] 400  39

> str(housing.sample)

# ...
```

[1] "**#** …" in a code script indicates large outputs better viewed in RStudio.

```
> summary(housing.sample)
# ...
```

Exercises:

Ex 01.01 – Using the **`subset()`** operation on the dataset how could we obtain the summary statistics (complete and for price only) for new dwellings and non-new dwellings? What about for those dwellings which have a view and those which do not?

Ex 01.02 – Again by **`subset()`** and the **`dim()`** operations, how many dwellings in our sample are new? How many have 5 parks or less nearby?

Lesson 02: Generating Basic Database Variables

We can use the variables that we have in our database to generate other measures to use later in our econometric specification and for modelling. Calling on the variables as a *vector* (when we include the **`$`** after the database we can specifically focus on one of the variables), we can operate on them as we would any numeric vector to create additional variables of interest.

Let's take the ratio of two variables (divide) to get the **`PRICE PER SQUARE METER`**.

```
> housing.sample$price.m2 <- housing.sample$price/housing.sample$area
> summary(housing.sample$price.m2)
# Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
# 1125    2055    2368    2468    2838    5538
```

Multiply two variables together. Multiplying two dummy variables together tells us which dwellings have both of attributes like **`NEW DWELLINGS WHICH ALSO HAVE A GARDEN`**. Or multiplying the area by itself to get **`AREA SQUARED`**.

```
> housing.sample$new.garden <- housing.sample$new*housing.sample$garden
> housing.sample$area.2 <- housing.sample$area*housing.sample$area
> summary(housing.sample$new.garden)
# Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
# 0.00    0.00    0.00    0.01    0.00    1.00
> summary(housing.sample$area.2)
# Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
# 400    3813    7225   11594   12100   91809
```

Taking the (natural) log of a variable to get **LOG OF PRICE** or **AREA**.

```
> housing.sample$ln.area <- log(housing.sample$area)
> housing.sample$ln.price <- log(housing.sample$price)
> summary(housing.sample$ln.area)
# Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
# 2.996   4.123   4.443   4.452   4.700   5.714
> summary(housing.sample$ln.price)
# Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
# 10.82   11.85   12.15   12.23   12.52   13.82
```

Many statistical models are built off the assumption of normally distributed data, and so often practitioners tend to take (natural) log transformations of all continuous variables – price, area, and especially distances. This may improve the model later on, and there is further some additional benefit to modelling log variables in terms of interpreting regression outputs.

Note that in R the **log()** function by default returns the natural log, base *e* (ln).

```
> housing.sample$ln.near.park <- log(housing.sample$near.park)
> housing.sample$ln.avgD.park <- log(housing.sample$avgD.park)
> housing.sample$ln.near.shopping <- log(housing.sample$near.shopping)
> housing.sample$ln.avgD.shopping <- log(housing.sample$avgD.shopping)
> housing.sample$ln.dist.baixa <- log(housing.sample$dist.baixa)
> housing.sample$ln.dist.tagus <- log(housing.sample$dist.tagus)
```

Exercises:

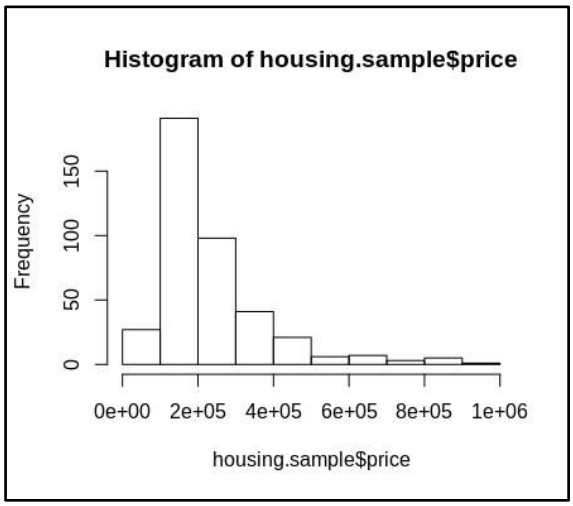
Ex 02.01 – Using the **baixa** variable (a dummy 0/1 indicator of dwellings in the main CBD), how could we interpret an interaction between **baixa** and **near.park**?

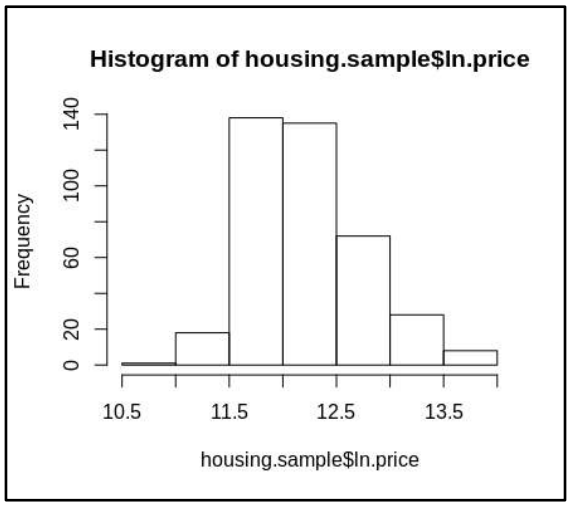
Lesson 03: Summary Statistics, Correlations, Scatter Plots

There are a number of different summary statistics and ways to look at the data - means, distributions, standard deviations, etc. We can conduct summary statistics on the entire database, or call certain variables specifically.

```
> summary(housing.sample)
# ...
> summary(housing.sample$price)
# Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
# 50000  140000  190000  237764  275000 1000000
> sd(housing.sample$price)
# [1] 152551.6
```

```
> hist(housing.sample$price)
# ...
> hist(housing.sample$ln.price)
# ...
```





We can also run correlations on our numeric variables to study the relationship between them one at a time. We can use the **`cor()`** function and specify directly which variables we are interested in getting the correlation for.

Another option is to use a small piece of code, **`unlist(lapply(housing.sample, is.numeric))`**, which gives us an index for which columns in our database are numeric. If a variable is not a numeric vector, then we cannot do any correlations, and so we drop it. Using this indexing function, we can apply the correlation function to our entire data set and get the correlation matrix between all numeric variables.

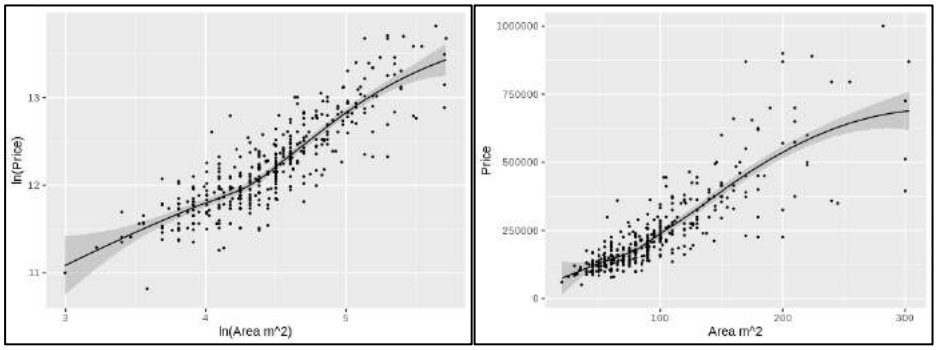
> The **`?lapply`** function is among the most powerful functions available in R. This allows us to apply a function (in this case the **`?is.numeric`** function) across a list of objects (vectors, columns, rows, databases, etc.). This is widely used when you want to split up a process and increase efficiency.
>
> https://www.r-bloggers.com/using-apply-sapply-lapply-in-r/

```
> cor(housing.sample[, unlist(lapply(housing.sample, is.numeric))])
# ...
> cor(housing.sample$price, housing.sample$area)
# [1] 0.8436786
> cor(housing.sample$ln.price, housing.sample$ln.area)
# [1] 0.860904
```

One of the most popular package in ***R*** is **`ggplot2`**. This is a data visualization packages with many, many functions and different parameters to make a range of plots, graphs, maps or charts. We will use a simple ggplot to look at the scatter plot/ correlation between (log) price and (log) area.

> Although we are using a quick plot, the ggplot website is a great resource to find different code parts and learn how to create really nice visual charts of whatever data you have. Chapter 3 of ***R For Data Science*** further has some nice walk-through tutorials on using **`ggplot2`** and creating figures.

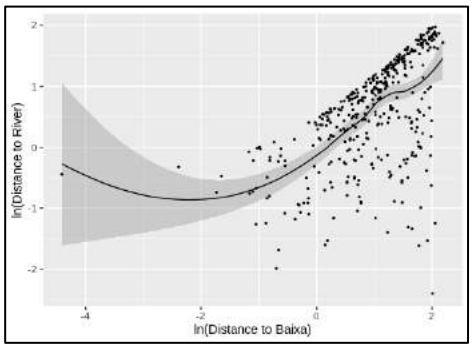
```
> ggplot(data = housing.sample, mapping = aes(x = area, y = price)) +
  geom_smooth(col = "black", size = 0.5) +
  geom_point(size = 0.5) +
  labs(x = "Area m^2", y = "Price")
> ggplot(data = housing.sample, mapping = aes(x = ln.area, y = ln.price)) +
  geom_smooth(col = "black", size = 0.5) +
  geom_point(size = 0.5) +
  labs(x = "ln(Area m^2)", y = "ln(Price)")
```



As we would expect, we see a positive correlation between price and the size of a dwelling. Taking the log transformation minimizes the impact from outlier observations and shows a clearer relationship between the variables.

We do not want, however, there to be any correlation between different groups of variables included in a model and used to explain price – which would give rise to *multicollinearity*. We can look at pairwise comparisons of similar variables to select the most appropriate to include in the model. For example, the distance to the riverfront and the distance to the primary business district – which is located along the river – are likely highly correlated and should therefore not be included in the model at the same time.

```
> ggplot(data = housing.sample, mapping = aes(x = ln.dist.baixa, y =
  ln.dist.tagus)) + geom_smooth(col = "black", size = 0.5) +
  geom_point(size = 0.5) + labs(x = "ln(Distance to Baixa)", y = "ln(Distance to
  River)")
> cor(housing.sample$ln.dist.baixa, housing.sample$ln.dist.tagus)
# [1] 0.5531924
```



Exercises:

Ex 03.01 – Based on the summary statistics, what can we say about the data we have on price? Is there any benefits or drawbacks of using one measure of price over the other (natural log versus direct level) for continued analysis?

Ex 03.02 – Here are a few references which provide some good examples of how to create a range of different plots with a variety of colours, styles, symbols, etc. Play around with the parameters and see how you can alter the above plots to make them more visually appealing. Try making the point symbols differ according to the new dwellings variable.

R for Data Science Chapter 3.....https://r4ds.had.co.nz/data-visualisation.html

ggplot website.....https://ggplot2.tidyverse.org/index.html

Guide for Colours in ***R***.....http://www.stat.columbia.edu/~tzheng/files/Rcolor.pdf

Guide for Symbols in ***R***.....http://www.sthda.com/english/wiki/r-plot-pch-symbols-the-different-point-shapes-available-in-r

Section 2 – Spatial Interpolation of Pollution Levels

Lesson 04: Loading and Generating Spatial Data (Points, Polygons, Rasters)

Since we have the latitude and longitude coordinates in the database, we can geo-reference the data and create a spatial database giving us the point locations across the city.

One important point of working with spatial data is to make sure that it is correctly projected. There are many different coordinate reference systems (CRS) - some can be based on relative global coordinates such as GPS (latitude and longitude which we have), while others may be projected and based on distances from equators.

The primary CRS we will rely on is the WGS84 standard. This 3D coordinate system considers the earth to be a sphere and locations on the sphere are denoted by latitude and longitude (GPS) coordinates.

While the WGS84 is a global spherical projection (equivalent values no matter where on the earth we are located), other CRS may represent a projected system where the earth is *flattened* and looked at as a standard 2D map. In these flattened projections, it does matter where in the world we are and how we flatten the map. Different cities, countries and regions may all have different CRS's which are denoted by the EPSG code – a good reference for finding local EPSG codes is:

https://www.spatialreference.org/

For Peru – the code for the projected EPSG is 24891:

```
"+init=epsg:24891 +proj=tmerc +lat_0=-6 +lon_0=-80.5 +k=0.99983008 +x_0=222000
+y_0=1426834.743 +ellps=intl +towgs84=-288,175,-376,0,0,0,0 +units=m +no_defs"
```

If we are creating a spatial version of the dataset by ourselves inside of R (instead of loading a spatial file directly), then we need to make sure the projection we set matches the type of coordinates we have. Since our observations are in GPS latitude and longitude, we specify the respective CRS when we set the spatial projection.

If we are reading in a spatial file – the projection is embedded directly, but before working with the file, we have to make sure that the pre-defined projection aligns with the rest of the data – it is crucial that when we create spatial variables or do any spatial analysis that the CRS's are correct.

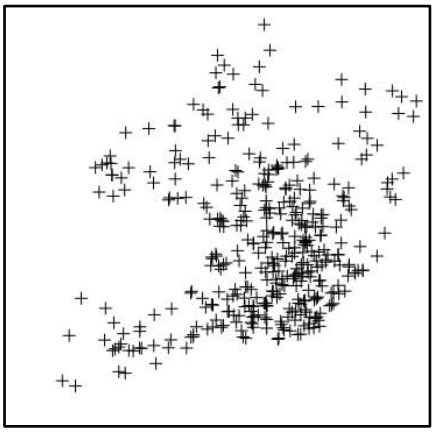
```
> housing.sample <- SpatialPointsDataFrame(cbind(housing.sample$longitude,
  housing.sample$latitude), housing.sample)

> proj4string(housing.sample) <- CRS("+proj=longlat +ellps=WGS84 +datum=WGS84
  +no_defs")
```

Take note that now *housing.sample* is no longer a simple database, but is a spatially referenced database. To call on only the data part of the spatial database (the simple database from before), we can use *housing.sample@data*.

```
> housing.sample

# class       : SpatialPointsDataFrame
# features    : 400
# extent      : -9.21725, -9.105303, 38.69277, 38.78732  (xmin, xmax, ymin, ymax)
# coord. ref. : +proj=longlat +ellps=WGS84 +datum=WGS84 +no_defs +towgs84=0,0,0
# variables   : 50
# ...

> summary(housing.sample)

# Object of class SpatialPointsDataFrame
# Coordinates:
#                 min       max
# coords.x1 -9.217251 -9.105303
# coords.x2 38.692772 38.787320
# Is projected: FALSE
# proj4string :
# [+proj=longlat +ellps=WGS84 +datum=WGS84 +no_defs +towgs84=0,0,0]
# Number of points: 400
# ...

> summary(housing.sample@data)

# ...

> summary(housing.sample@data$price)

# Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
# 50000  140000  190000  237764  275000 1000000
```

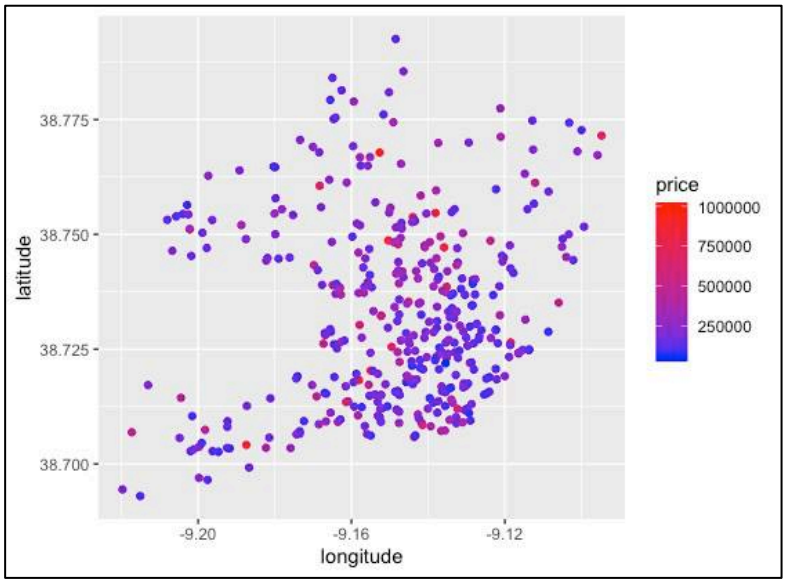
We can do a quick plot of the data to see if the mapping 'appears' to be correct.

```
> plot(housing.sample)
```



If we want to look at a simple plot of price distributions we can use ggplot to do a (non-spatial) scatter plot using latitude and longitude as the X-Y coordinates and shading the intensity by the price level. Using the **`@data`** suffix on the *housing.sample*, allows us to use it like we would any other data frame.

```
> ggplot() + geom_point(data=housing.sample@data, aes(x=longitude, y=latitude,
  col = price)) + scale_colour_gradient(low = "blue", high = "red")
```



We will now introduce some measures of neighbourhood pollution using a spatial interpolation on yearly average pollution levels from monitoring stations around the city. The data is stored in a standard .csv file which includes the monitoring station latitude and longitude, and the average PM10 pollution values from all observations monitored in 2007. This gives us a proxy for the level of pollution as it varies across the city in the year of our housing sales.

The first step is to read in the pollution data .csv file and translate this into a spatial points data frame. We first generate a variable which is the (natural) log value of pollution concentration – partially for a statistical reason. The interpolations we are using – the *Inverse Distance Weight* family – works best when the data is normally distributed, and taking the log value before will tend to improve the interpolation estimates. This is of course case specific and based on the data we are using.

```
> pollution <- read.csv(paste0(wd, "/Data/pollution_2007.csv"),
  stringsAsFactors=T)
> pollution$lnPM10 <- log(pollution$PM10.2007)
> pollution <- SpatialPointsDataFrame(cbind(pollution$LONGITUDE,
  pollution$LATITUDE), pollution)
> proj4string(pollution) <- CRS("+proj=longlat +ellps=WGS84 +datum=WGS84
  +no_defs")
```

We have to build the canvas onto which we are going to interpolate the data. We must first build an empty raster file, and then give each pixel in the raster a value based on the interpolation model later on. To build the raster, we have to get the boundary extents of our data and then cut this area up into our raster pixels. The **`bbox`** function below will give the boundary box which covers the data of interest (the pollution stations), and then we use the **`extent`** function to get the coordinate values of this box.

> Since we want to use normal distance measures to build this raster, both these functions are applied to spatially transformed data. Notice that the CRS of our original pollution points are in the **`long-lat`** projection (based on GPS coordinates) and standard

Euclidean distances don't work here. The new projection we are using is based in meters and so we can calculate distances as we would normally.

Using the four coordinate points of this box – which are saved as components of the *int.unit* object: *int.unit[1], int.unit[2], int.unit[3], int.unit[4]* – we can create a raster using the raster function, and specify that we want to have pixel sizes of 100 meters. Because our CRS is in meters, we specify in the function that the number of columns should be equal to the absolute value of the x distance divided into blocks of 100, and conversely for the number of rows: **`ncol=abs(int.unit[1]-int.unit[2])/100`**.

Now that we have the empty raster grid, we have to give it a coordinate projection. Because it was built under the Euclidean-distance friendly projection **`CRS("+init=epsg:3763")`**, we have to first specify this as the CRS of the raster file. But then – since the rest of our data is in the long-lat projection, and we want the raster to align with everything else, we do a reprojection back to the projection of interest.

```
> int.unit <- extent(bbox(spTransform(pollution, CRS("+init=epsg:3763"))))
> int.unit <- raster(ncol=abs(int.unit[1]-int.unit[2])/100, nrow=abs(int.unit[3]-
  int.unit[4])/100, xmn=int.unit[1], xmx=int.unit[2], ymn=int.unit[3],
  ymx=int.unit[4])
> proj4string(int.unit) <- CRS("+init=epsg:3763")
> int.unit <- projectRaster(int.unit, crs="+proj=longlat +ellps=WGS84
  +datum=WGS84 +no_defs")
```

The next step is to define the aggregation area. This will be the polygon areas to which we will take the average pollution from across the entire raster. Here we use the *freguesia* units as the neighbourhood definition – after we get our raster picture of pollution, we aggregate up to find the average value of pollution for each *freguesia.* This gives us a measure of neighbourhood pollution we can merge with our housing data.

```
> agg.unit <- spTransform(readOGR(dsn = paste0(wd, "/Data"), layer =
  "freguesias"), CRS("+proj=longlat +ellps=WGS84 +datum=WGS84 +no_defs"))
> agg.unit <- agg.unit[,c("FREGUESIA")]
```

Exercises:

Ex 04.01 – Plotting the direct price variable as we did in **`ggplot()`** may not always highlight the full pattern if outliers are present and driving the visualizing – try plotting different variables, and in particular the log of price, to see if other variables change over space.

Lesson 05: Inverse Distance and Inverse Squared Distance

We now want to predict pollution levels across space based on a limited sample of available observations from monitoring stations across the city. We will focus on a broad family of interpolation methods all based on weighting estimated pollution levels at unknown areas of

the raster (map) conditional on the *inverse distance* from given known pollution levels (monitoring stations). There are two parameters that we need to specify.

First we have to specify the number of nearest neighbours, or from which monitoring stations pollution observations will be taken. The default is to use all stations, however in some contexts there may be some rational to indicate that only observations in direct proximity are useful for prediction your variable of interest.

Secondly, we have to specify any weight that we may be giving to the inverse distance. Most commonly, this is left to have a weight of 1 or 2 – which give us the Inverse Distance and the Inverse Squared Distance models respectively. If we think that the rate at which the influence decays fast over space, then the squared distance may more accurately capture this effect.

There is a general mathematical formula which can be used to represent all the specific cases of this family of models. Here, $\mathbb{P}(x)$ represents the (log) pollution level to be predicted/ estimated at location x. N is the number of neighbours (monitoring stations) that we consider in the prediction (the default is to use all observations which we adopt here).

If the distance between the prediction location and a monitoring station is zero $d(x, x_i) = 0$, then the location is exactly the monitoring station and we take $\mathbb{P}(x) = \mathbb{P}(x_i)$. If the prediction location does not have a monitoring station, then the prediction pollution is given as the weighted average of observed pollution levels $\mathbb{P}(x_i)$ from all nearby stations N, weighted by and inverse distance $1/d(x, x_i)^\rho$ and power parameter of ρ.

$$\mathbb{P}(x) = \begin{cases} \dfrac{\sum_{i=1}^{N}[1/d(x,x_i)^\rho] \cdot \mathbb{P}(x_i)}{\sum_{i=1}^{N}[1/d(x,x_i)^\rho]} & \text{if } \; d(x,x_i) \neq 0 \\ \mathbb{P}(x_i) & \text{if } \; d(x,x_i) = 0 \end{cases}$$

The spatial interpolation can be done in one line of code if everything is neatly organized and in the correct format. The first function to define is **`gstat()`**. This is how we define general spatial interpolation models and is very flexible.

```
gstat(formula=lnPM10 ~ 1, locations=pollution, set=list(idp=1))
```

The formula specifies the variable we want to interpolate, and whether we want to condition this interpolation on any other covariates. For example, if we had continuous elevation data for every raster grid our interpolation model could be based not only on the inverse distance but further by the geography of the city. This is outside the scope, and we specify the base model of **`lnPM10 ~ 1`** to indicate that our interpolation is only based on distances between observations.

We specify the locations (spatial points) of our data, and then include the additional parameter of **`idp`** (inverse distance power) to indicate any power function we want to set.

```
> sp.interpolation <- mask(crop(interpolate(int.unit, gstat(formula=lnPM10 ~ 1,
  locations=pollution, set=list(idp=1))), agg.unit), agg.unit)
> plot(sp.interpolation)
> plot(exp(sp.interpolation))


> RMSE <- list()
> for (k in 1:nrow(pollution)){
    init.val <- c(3, 2)
    RMSE[[k]] <- sqrt(as.numeric((predict(gstat(formula=as.formula("lnPM10 ~ 1"),
      locations=pollution[-k,], set=list(idp=1)), newdata=pollution[k,],
      debug.level=0)$var1.pred -
    pollution[k,][,c("lnPM10")]@data)^2))
  }
> mean(unlist(RMSE))
# [1] 0.1451815
```

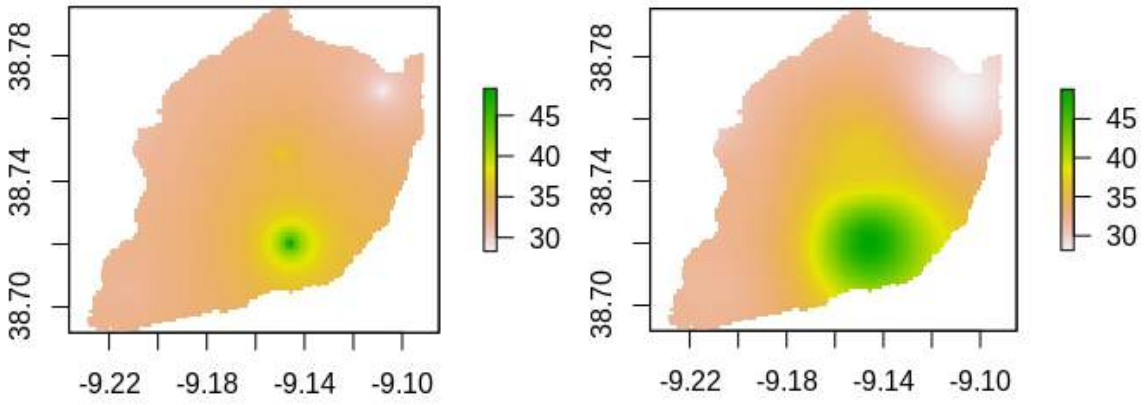
By changing the parameter values, we are able to specify and predict pollution values based on the different statistical models of interest – and then compare the different methods together.

```
> sp.interpolation <- mask(crop(interpolate(int.unit, gstat(formula=lnPM10 ~ 1,
locations=pollution, set=list(idp=2))), agg.unit), agg.unit)
> plot(sp.interpolation)
> plot(exp(sp.interpolation))


> RMSE <- list()
> for (k in 1:nrow(pollution)){
    init.val <- c(3, 2)
    RMSE[[k]] <- sqrt(as.numeric((predict(gstat(formula=as.formula("lnPM10 ~ 1"),
      locations=pollution[-k,], set=list(idp=2)), newdata=pollution[k,],
      debug.level=0)$var1.pred -
    pollution[k,][,c("lnPM10")]@data)^2))
  }
> mean(unlist(RMSE))
# [1] 0.131618
```

The second function is the **`interpolate()`** where we provide the raster onto which we want to interpolate, *int.unit*, and the **`gstat()`** model we want to use. Finally, the **`mask()`** and **`crop()`** functions are ways to remove the excess pixels around the area of interest. We are only interested in looking at pollution in Lisbon to aggregate on, so we remove any excess.

We can **`plot()`** raster's directly to see the distribution of the interpolated pollution over space. We can further apply a transformation, like the exponential **`exp()`**, to the raster to convert back from (natural) log pollution levels to the direct PM10 concentration.

Inverse Distance (exp. Transformed) *Inverse Squared Distance (exp. Transformed)*



A small line of code has been included after each interpolation to calculate the root mean squared error (RMSE) of prediction – a measure of how well our model results are able to match the true observed value of the data. If predicted values are very close to observed values at a given location, then we have low deviation between these values and the RMSE statistic, which weights this deviation, would be lower and indicate a better fit.

$$RMSE = \sqrt{[avg((x - \bar{x})^2)]}$$

The general idea is to iteratively (in a for loop) remove one monitoring station at a time. We first remove the monitoring station and then estimate the output using the model we are interested in evaluating. This gives us a predicted value from the model and a true observed value from a monitoring station which we can compared together using the RMSE statistic.

Doing this systematically, we get an individual statistic for every time we leave one of the monitoring stations out, and taking the average of these values provides a measure of fit for the interpolation model that we can compare against other models. Here, the model is a better fit in using the inverse squared distance over the unweighted inverse distance.

Lesson 06: Neighbourhood Level Aggregation

We select the best interpolation model based on the RMSE. In our case, this is the inverse squared distance using all monitoring stations in the sample. Setting this as the selected interpolation *sp.interpolation*, we can aggregate to get the average value of the raster within the chosen *freguesia* neighbourhood boundaries.

```
> sp.interpolation <- mask(crop(interpolate(int.unit, gstat(formula=lnPM10 ~ 1,
  locations=pollution, set=list(idp=2))), agg.unit), agg.unit)

> sp.aggregation <- do.call(rbind, lapply(extract(sp.interpolation, agg.unit),
  FUN=mean))

> agg.unit <- data.frame(agg.unit, lnPM10 = sp.aggregation)


> housing.sample <- merge(housing.sample, agg.unit, by.x="freguesia",
  by.y="FREGUESIA", all.x=T, sort=F)
```

Here we are applying the **`extract`** function between the raster and the aggregation units, and then apply the **`mean`** across a list to get the average value of raster pixels within each neighbourhood unit. Finally, we bind together this output into a list using the rbind code and combine it with our neighbourhood data frame. The *agg.unit* is therefore now a data frame with the *freguesia* ID and aggregate average value of log pollution interpolated from an Inverse Squared Distance Weight model – this can easily be merged with our original data.

> When merging on a spatial database (housing.sample) it is crucial to include **`sort=F`** - if we do not, then we risk mixing up the locations of the data.

Exercises:

Ex 06.01 – Changing the gstat parameters can allow us to include a power function on the Inverse Distance Weight or to take the interpolation using either all data from all monitoring stations or a select sample of the nearest. Using these parameter variations, try modelling and comparing a few different interpolation models.

Section 3 - Spatial Modelling, Diagnostics and Techniques

Lesson 07: Fit Baseline OLS Model and Run Diagnostics

After cleaning and generating the variables above, we have a database with the following types of variables that we can use to model.

- Price and price per square meter . . . . . P
- Size and structural characteristics X
- Neighbourhood (census) characteristics . . . . . N
- (Environmental) Locational attributes (distance to parks, pollution) L
- *Freguesia* identifiers . . . . . F

We will start with a baseline model with none of the spatial variables and then build up to improve the model with different specifications as we add in more detailed variables. The baseline model being estimated is the OLS specification:

$$ln(P) = \beta 0 + \beta 1 * X + \varepsilon$$

Some notes on choosing the best model: We have generated a large number of variables that we can use to specify our model. Fitting a model is as much an art as it is a science. Keep in only the most relevant variables which can easily be interpreted. Make sure that variables included are not capturing the same effect/ correlated. Often, the best model is the simplest most parsimonious one. The **`lm()`** function runs the OLS specification, and we will save all the models in an empty list together.

```
> OLS.baseline <- list()

> OLS.baseline[[1]] <- lm(ln.price ~ area + area.2 + new + pool + parking +
  fireplace + dwindows + garden + aircond + view, na.action=na.omit, data =
  housing.sample@data)
> summary(OLS.baseline[[1]])

# ...
```

We can save the OLS model to print it out later when we want. If we use the **`summary()`** function on the regression model, we get information on: coefficients and standard errors; p-values; F statistic; and R-squared. More detailed diagnostics can be obtained from various other packages in R.

The first model of **`OLS.baseline[[1]]`** is the full, comprehensive structural model which includes all structural variables. Given our small sample size however, we cannot include too many variables and so we can remove variables with very high p-values which are not well explained in the model, and we can fit a more parsimonious model, **`OLS.baseline[[2]]`**.

```
> OLS.baseline[[2]] <- lm(ln.price ~ area + area.2 + new + parking + fireplace +
  aircond + view, na.action=na.omit, data=housing.sample@data)
> summary(OLS.baseline[[2]])

# ...
```

Model **OLS.baseline[[2]]** is a subset, more parsimonious version of the full **OLS.baseline[[1]]**. Since one model is nested within the other, we can run an ANOVA test to determine whether we gain any additional information from using the more complex model (**OLS.baseline[[1]]**) rather than the simpler model (**OLS.baseline[[2]]**). We run this with the **anova()** function below, testing whether any significant information is gained by using the more complex model.

The null hypothesis of the ANOVA test is that the coefficients for all variables in **OLS.baseline[[1]]** that are not in **OLS.baseline[[2]]** are zero, with the alternative being that those coefficients are not zero. Our results below do not reject the null hypothesis (with a p-value of 0.6359), and thus we can continue our analysis with the more parsimonious model **OLS.baseline[[2]]** without any loss of information.

```
> anova(OLS.list[[1]], OLS.list[[2]])
# ...
```

There are a wide range of regression diagnostics that we can examine - some based on statistics tests, others based on more visual cues.

Variance Inflation Factor (VIF): indication of whether any of the control variables are correlated with each other (multicollinearity). Rule of thumb being that we want the value of this to be less than 10 for the variables of interest.

```
> lapply(OLS.list, vif)
# ...
```

Akaike Information Criterion (AIC): this is a measure of the quality of fit of the model. We cannot however interpret this in isolation - we must always compare the AIC of one model against the AIC of the other model to determine which fits better the data. Rule of thumb is that we select the model with the lowest AIC value.

```
> lapply(OLS.list, AIC)
# ...
```

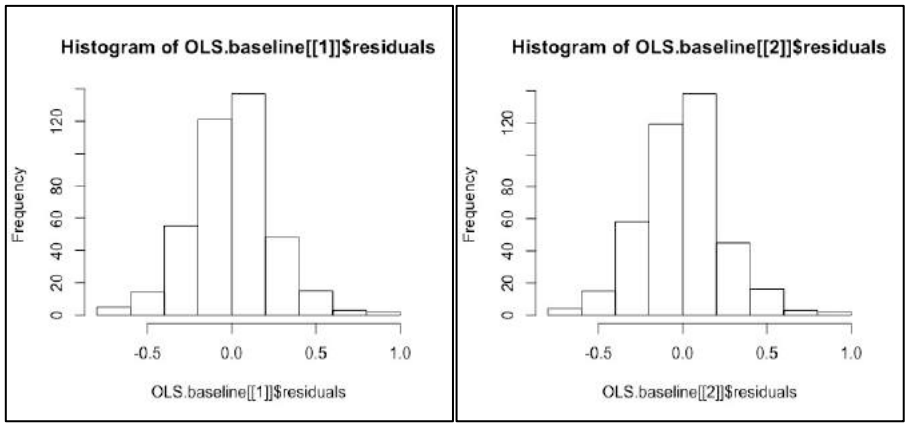
Sum of Squared Errors (SSE): a measure of how well the model fits the data, or how far off the predicted values are from their actual values. A lower value of SSE indicates a better fitting model. We use the **anova()** function on the model to obtain the residual sum of square and then extract it using the correct indexing.

```
> lapply(OLS.list, function(x) tail(anova(x)[,2], 1))
# ...
```

Normality of Residuals: one of the key assumptions of the OLS model is that errors have residuals with a conditional mean of zero. The Jarque-Bera test statistic tests the normality of a data series by comparing the mean and the skewness of the distribution. The null hypothesis here is that the residuals have a normal distribution with mean zero, and so we do not want to reject the null.

```
> lapply(OLS.list, function(x) jarque.bera.test(x$residuals))
# ...
> hist(OLS.list[[1]]$residuals)
# ...
> hist(OLS.list[[2]]$residuals)
# ...
```

This test is sensitive to small samples however, and should be interpreted with caution. One way to double check this assumption is to simply plot a histogram of the residuals to examine their distribution - they should look normally distributed around a mean of zero.



Residual Auto-correlation: In our OLS specification, we require that there is no correlation among the terms of our residuals. The Durbin-Watson and Breusch-Godfrey test statistics test for this type of correlation in the residuals to inform on whether the error term may exhibit correlation.

The Durbin-Watson tests for 1st order correlation between direct observations. The null hypothesis states that there is no serial correlation among the residuals, while the alternative indicates significant correlation.

The Breusch-Godfrey test is a more general version testing broader (larger than 1st order) correlation between all residuals. This statistic is more appropriate for spatial data since it tests for the residual correlation between, not just adjacent (1st degree) observations, but observations which may have unaccounted influences due to proximity which would be missed under the Durbin-Watson statistic.

```
> lapply(OLS.list, dwtest)
# ...
> lapply(OLS.list, bgtest)
# ...
```

Heteroskedasticity: It is further important to test whether or not the residuals of our model significantly depend on the independent covariates. If the variance of the residual terms varies significantly with values of the **x** variables, then we are said to have heteroskedasticity in the model.

We can test formally for heteroskedasticity using the Breusch-Pagan test statistic. The null hypothesis is that there is no heteroskedasticity and the variance of the errors does not depend on the covariates. If not addressed, heteroskedasticity may potentially impact the significance and interpretation of our covariates.

We almost always find some indication of heteroskedasticity in our models, and so it is common to simply always provide the correction to deal with this. If we correct our model using *heteroskedastic consistent standard errors* (or *White's standard errors*) we address this problem directly.

The estimation of the model using *heteroskedastic consistent standard errors* can be done using **`coeftest(OLS.list[[2]], function(x) vcovHC(x, type="HC0"))`**.

```
> lapply(OLS.list, bptest)
# ...
> lapply(OLS.list, function(y){ coeftest(y, function(x) vcovHC(x, type="HC0")) })
# ...
```

A Note on Printing Out Results:

You may notice that we are simply printing out the results directly into the ***R*** terminal and reading/ interpreting them there. This is fine for looking at one, maybe two, models, but when we want to compare many different models and diagnostics this is not feasible to constantly be printing out the results in the terminal.

Because there are many different types of objects in ***R***, and especially since the standard OLS and the spatial models have different classes and information associated to them, it is not always easy to print out all the information that you need consistently.

What many researchers do is to manually build and export output tables, which is obviously outside the scope of this workshop. There would be not much benefit to spending too much time on learning the exporting of data while giving up on learning more detailed econometric techniques. For this reason, all model outputs have already been cleaned and packaged together in an excel file so that we can make easy use of referencing the different models and diagnostics.

The function most commonly used to print out the models however is the **`stargazer()`** package. This is a really well developed package for exporting nice tables on all sorts of models, with many manual options for changing how things are organized and what data is presented.

There are many good references for how to print out single or multiple models in **`stargazer()`** and how to include additional diagnostics.

https://www.rdocumentation.org/packages/stargazer/versions/5.2.2/topics/stargazer

https://cran.r-project.org/web/packages/stargazer/vignettes/stargazer.pdf

For the sake of reference, below is a short example on how you could export a very simple OLS output with the model results. Using the above references, it is possible to expand on this output to include additional diagnostics and parameters. One of the most important alterations to the simple printout that we can make is to specify that the standard errors to be printed should be the robust standard errors which are estimated from the **`coeftest()`** function above.

```
> stargazer(OLS.list, type="html", title="OLS Baseline Model (Robust S.E.)",
  style="aer", star.cutoffs=c(0.10, 0.05, 0.01),
  se = lapply(OLS.list, function(y){ coeftest(y, function(x) vcovHC(x,
  type="HC0"))[,2] }), digits = 5, out=paste0(wd, "/OLS_Baseline.htm"))
```

Exercises:

Ex 07.01 – Try other ways of modelling the non-linearity of area. Instead of using squared values, do results/ interpretations change when we use logs? What is the impact of being simultaneously in a new house with a garden? Check that this variable is not correlated too strongly with any others.

Ex 07.02 – How could we add in the results of the diagnostics from the Breusch-Pagan test statistics to the **`stargazer()`** output using the **`add.lines`** parameter?

Lesson 08: Enhancing the Baseline Model with GIS Variables

We can indirectly include relative space into our model through our GIS variables created. These variables allow us to consider the impact of a dwelling's location relative to important points/ areas of the city. We update the baseline specification to now include the range of neighbourhood and locational attributes of the dwelling:

$$ln(P) = \beta 0 + \beta 1*X + \beta 2*N + \beta 3*L + \varepsilon$$

Notice that we created a large number of structural and locational variables. We have to test out different combinations of variables to find the best, most concise, model. We will need to take into account that some variables capture similar effects and think about how best to interpret and obtain the effects of interest. Note also that the number of variables we can include depends directly on the number of observations we have in our database. In our simple example, we will thus be limited to including only the most relevant characteristics.

```
> OLS.gis <- list()
> OLS.gis[[1]] <- lm(ln.price ~ area + area.2 + new + parking + fireplace +
  aircond + view + Ppop.unemp, na.action=na.omit, data=housing.sample@data)
> summary(OLS.gis[[1]])
```

```
# ...

> OLS.gis[[2]] <- lm(ln.price ~ area + area.2 + new + parking + fireplace +
  aircond + view + Ppop.unemp + log(dist.baixa), na.action=na.omit,
  data=housing.sample@data)
> summary(OLS.gis[[2]])
# ...
> OLS.gis[[3]] <- lm(ln.price ~ area + area.2 + new + parking + fireplace +
  aircond + view + Ppop.unemp + log(dist.tagus), na.action=na.omit,
  data=housing.sample@data)
> summary(OLS.gis[[3]])
# ...
```

We can try different combinations of our locational variables to test different models. We need to limit the number of variables which may be capturing the same effect (collinear) - for example, it may not be possible (given our small sample) to capture simultaneously the effect from being located in *Baixa* and proximity to the riverfront.

My modelling strategy here is to systematically introduce variables in groups – starting with structural characteristics, next including the best measure of neighbourhood socio-economic status (percentage of population unemployed), and then location to the core downtown/ riverfront. At every stage, since we have many variables capturing the same effect, different models should be run and tested, moving forward and evolving the model only when you find the best model to improve upon.

> If the baseline structural variables are not modelled correctly and yielding non-consistent outcomes for example, it would make no sense to build off this model by trying to include environmental effects.

```
> OLS.gis[[4]] <- lm(ln.price ~ area + area.2 + new + parking + fireplace +
  aircond + view + Ppop.unemp + log(dist.baixa) + lnPM10 + ln.avgD.park ,
  na.action=na.omit, data=housing.sample@data)
> summary(OLS.gis[[5]])
# ...
> OLS.gis[[5]] <- lm(ln.price ~ area + area.2 + new + parking + fireplace +
  aircond + view + Ppop.unemp + log(dist.tagus) + lnPM10 + ln.avgD.park ,
  na.action=na.omit, data=housing.sample@data)
> summary(OLS.gis[[6]])
# ...
```

Using the **`stargazer`** package and format that we already specified, we can easily collect all these models and compare them in one comprehensive table. Here, the different measures of distance to *Baixa* and the Tagus riverfront are all important variables to include, but highly correlated to each other. So, we can simultaneously model the three different specifications, print them out in one comprehensive table, and then compare all the diagnostics and results across specifications and combinations of variables. This makes the model selection process robust, comprehensive and systematic.

```
> stargazer(OLS.gis, type="html", title=" OLS with GIS Variables (Robust S.E.)",
  style="aer", star.cutoffs=c(0.10, 0.05, 0.01),
  se = lapply(OLS.gis, function(y){ coeftest(y, function(x) vcovHC(x,
  type="HC0"))[,2] }), digits = 5, out=paste0(wd, "/OLS_GIS.htm"))
```

Exercises:

Ex 08.01 – Discussion: Compare the SSE, variable significance, AIC, R^2, and other measures of model fit – not only in terms of the statistics but from the point of view of what questions you want to answer with your model. Which model would you choose?

Lesson 09: Controlling for Location Price Differentials

We update the baseline specification now to include fixed effects for identifying in which area of the city a dwelling is located. This is one way in which we can use the OLS estimation procedure to model the price difference between different areas of the city, accounting for the unobservable (un-measurable) *freguesia* level local attributes.

$$ln(P) = \beta 0 + \beta 1 * X + \beta 2 * N + \beta 3 * L + \beta 4 * F + \varepsilon$$

Our **`freguesia`** variable is a categorical (non-numeric character vector) variable, and we can use this to manually generate an indicator {0, 1} variable to look at the price impact of being located in a particular area of the city.

The level of how fine a resolution we can estimate depends on the number of observations that we have. If we had a complete coverage of housing values over time and space for example, we would potentially be able to estimate the impact of living on a specific section of a street or road. With fewer samples, we are limited to estimating the average price impact of living in broad areas of the city – which we define into four groups using collections of freguesias.

```
>  housing.sample@data$BairroZone <- NA
>  housing.sample@data$BairroZone <-
    ifelse(as.character(housing.sample@data$freguesia)=="Sao Paulo" |
    as.character(housing.sample@data$freguesia)=="Santa Catarina" |
    as.character(housing.sample@data$freguesia)=="Merces" |
    as.character(housing.sample@data$freguesia)=="Sao Mamede" |
    as.character(housing.sample@data$freguesia)=="Coracao De Jesus" |
    as.character(housing.sample@data$freguesia)=="Sao Jose" |
    as.character(housing.sample@data$freguesia)=="Pena" |
    as.character(housing.sample@data$freguesia)=="Anjos" |
    as.character(housing.sample@data$freguesia)=="Graca" |
    as.character(housing.sample@data$freguesia)=="Sao Vicente De Fora" |
    as.character(housing.sample@data$freguesia)=="Santa Engracia" |
    as.character(housing.sample@data$freguesia)=="Castelo" |
    as.character(housing.sample@data$freguesia)=="Encarnacao" |
    as.character(housing.sample@data$freguesia)=="Madalena" |
    as.character(housing.sample@data$freguesia)=="Martires" |
    as.character(housing.sample@data$freguesia)=="Sao Cristovao e Sao Lourenco" |
    as.character(housing.sample@data$freguesia)=="Sao Miguel" |
    as.character(housing.sample@data$freguesia)=="Sao Nicolau" |
    as.character(housing.sample@data$freguesia)=="Sacramento" |
    as.character(housing.sample@data$freguesia)=="Santa Justa" |
    as.character(housing.sample@data$freguesia)=="Santiago" |
    as.character(housing.sample@data$freguesia)=="Santo Estavao" |
    as.character(housing.sample@data$freguesia)=="Se" |
    as.character(housing.sample@data$freguesia)=="Socorro", "1oBairro",
    housing.sample@data$BairroZone)
>  housing.sample@data$BairroZone <-
    ifelse(as.character(housing.sample@data$freguesia)=="Santa Isabel" |
    as.character(housing.sample@data$freguesia)=="Lapa" |
    as.character(housing.sample@data$freguesia)=="Santos-o-Velho" |
    as.character(housing.sample@data$freguesia)=="Prazeres" |
```

```
    as.character(housing.sample@data$freguesia)=="Santo Condestavel" |
    as.character(housing.sample@data$freguesia)=="Alcantara" |
    as.character(housing.sample@data$freguesia)=="Ajuda" |
    as.character(housing.sample@data$freguesia)=="Sao Francisco Xavier" |
    as.character(housing.sample@data$freguesia)=="Santa Maria De Belem",
    "2oBairro", housing.sample@data$BairroZone)
>  housing.sample@data$BairroZone <-
    ifelse(as.character(housing.sample@data$freguesia)=="Benfica" |
    as.character(housing.sample@data$freguesia)=="Campolide" |
    as.character(housing.sample@data$freguesia)=="Sao Sebastiao Da Pedreira" |
    as.character(housing.sample@data$freguesia)=="Nossa Senhora De Fatima" |
    as.character(housing.sample@data$freguesia)=="Alvalade" |
    as.character(housing.sample@data$freguesia)=="Sao Joao De Brito" |
    as.character(housing.sample@data$freguesia)=="Campo Grande" |
    as.character(housing.sample@data$freguesia)=="Sao Domingos De Benfica" |
    as.character(housing.sample@data$freguesia)=="Carnide" |
    as.character(housing.sample@data$freguesia)=="Lumiar" |
    as.character(housing.sample@data$freguesia)=="Charneca" |
    as.character(housing.sample@data$freguesia)=="Ameixoeira", "3oBairro",
    housing.sample@data$BairroZone)
>  housing.sample@data$BairroZone <-
    ifelse(as.character(housing.sample@data$freguesia)=="Santa Maria Dos Olivais"
    | as.character(housing.sample@data$freguesia)=="Marvila" |
    as.character(housing.sample@data$freguesia)=="Beato" |
    as.character(housing.sample@data$freguesia)=="Sao Joao" |
    as.character(housing.sample@data$freguesia)=="Penha De Franca" |
    as.character(housing.sample@data$freguesia)=="Sao Jorge De Arroios" |
    as.character(housing.sample@data$freguesia)=="Sao Joao De Deus" |
    as.character(housing.sample@data$freguesia)=="Alto Do Pina", "4oBairro",
    housing.sample@data$BairroZone)
>  housing.sample@data$BairroZone <- as.factor(housing.sample@data$BairroZone)
>  housing.sample@data$BairroZone <- relevel(housing.sample@data$BairroZone,
    ref="1oBairro")
```

Using a conditional **`ifelse()`** statement, we can manually define four different areas of the city *1oBairro*, *2oBairro*, *3oBairro* and *4oBairro* – which are each a collection of mutually exclusive *freguesia.* This definition comes from the local authority and is broadly used in their administrative planning, but many different types of spatial fixed effect units can be used – large or small administrative boundaries; census tracts; transportation areas; postal codes; etc.

In order to interpret this variable, we need to determine first what will be our reference location in the city. To properly include fixed effects in the regression, we must leave one category out (our reference) and the estimates we obtain for each other area represents the deviation in housing prices from that area relative to the reference area. The reference category can be anything, although we want to choose one which has a relatively high number of observations and one from which it intuitively makes sense to compare.

The main downtown Pombaline-style architecture and grid-like pattern of dwellings, for which Lisbon is famous, is located in the *Baixa* – or the collection of *freguesias* given by *1oBairro*. We can use this as our reference area and use the **`relevel()`** function to re-specify this so that in the model all results will be interpreted as the price difference relative to this core area of the city.

We can again systematically build upon our previous model testing different ways to include combinations of parameters with the new spatial fixed effects. If there are spatial influences over space which impact housing prices – yet which we cannot easily measure with given data – then including these location variables should improve the model specification.

```
> OLS.spfe <- list()
> OLS.spfe[[1]] <- lm(ln.price ~ area + area.2 + new + parking + fireplace +
  aircond + view + Ppop.unemp + log(dist.baixa) + BairroZone, na.action=na.omit,
  data=housing.sample@data)
> OLS.spfe[[2]] <- lm(ln.price ~ area + area.2 + new + parking + fireplace +
  aircond + view + Ppop.unemp + log(dist.tagus) + BairroZone, na.action=na.omit,
  data=housing.sample@data)


> OLS.spfe[[3]] <- lm(ln.price ~ area + area.2 + new + parking + fireplace +
  aircond + view + Ppop.unemp + log(dist.baixa) + ln.avgD.park + BairroZone,
  na.action=na.omit, data=housing.sample@data)
> OLS.spfe[[4]] <- lm(ln.price ~ area + area.2 + new + parking + fireplace +
  aircond + view + Ppop.unemp + log(dist.tagus) + ln.avgD.park + BairroZone,
  na.action=na.omit, data=housing.sample@data)


> OLS.spfe[[5]] <- lm(ln.price ~ area + area.2 + new + parking + fireplace +
  aircond + view + Ppop.unemp + log(dist.baixa) + lnPM10 + ln.avgD.park +
  BairroZone, na.action=na.omit, data=housing.sample@data)
> OLS.spfe[[6]] <- lm(ln.price ~ area + area.2 + new + parking + fireplace +
  aircond + view + Ppop.unemp + log(dist.tagus) + lnPM10 + ln.avgD.park +
  BairroZone, na.action=na.omit, data=housing.sample@data)
```

Knowing we can easily export and compare many models directly, we can generate a collection of potential models. Still building up systematically, we can look at the impact of including the location fixed effects at all stages of the modelling. If the parameters of interest are the environmental locational variables, then it is easy to look at how the inclusion of a location fixed effect impacts the model and outputs when these environmental variables are included and are not included.

As we control for more of the unobserved or unmeasured location effects – by including these fixed effects – then we should get more robust estimates for the true effect of the other parameters. So, if the model is specified correctly by the inclusion of locational spatial heterogeneity, then the parameter estimates on the other variables from this model are the most preferred estimates.

Moving forward, we will consider that **`OLS.spfe[[6]]`** is our most preferred and complete model specification. From here on out, we will conduct the spatial diagnostics and models to this OLS specification. Once we have a preferred OLS specification of interest, then we must run the diagnostics to see if spatial dependence exists, and if so then we must correct this by estimating the appropriate spatial model.

Exercises:

Ex 09.01 – How could the inclusion of both location fixed effects and locational amenities (say the count of amenities) cause problems in the model?

Ex 09.02 – How could interaction effects be used to look at the impact of, for example, having a new dwelling in the downtown area versus elsewhere in the city?

Lesson 10: Spatial Dependence - Generating and Visualizing Spatial Weights

Up until now we have only been running (non-spatial) OLS estimation techniques. These OLS models have explicitly included spatial (GIS) variables and spatial fixed effects, but has yet to control for any spatial dependence or spatial heterogeneity in the data generating process. If spatial dependence exists in our data and is not corrected for, we will have a bias in our estimated results which may affect out interpretation and predictions from the model.

The first step in spatial modelling is to specify what our *spatial weight matrix* is. This matrix, which is defined by us, is the **NxN** square matrix which defines the "neighbour relationship" between all variables. Depending on how we decide to define this relationship, we will have a matrix which connects each dwelling with all other dwellings that we consider to be its neighbor.

Common ways of defining the weight matrix include:

- {0, 1} identifier to specify whether a property is within some given distance of another property.
- Weighted by inverse distance (closer properties have higher weights) or inverse square distance for properties within a given distance.
- Using a given number of nearest neighbors (can be specified to be within a certain distance or not).

We can specify and test all sorts of underlying neighbourhood definitions to make sure that our estimated results are robust and not sensitive to which matrix we choose. For our example, we will consider neighbours to be properties that are within 500 meters of each other, and also the 20 nearest properties.

We first must use the **dnearneigh()** function to identify all properties within 0.5 km (500 meters) of each other, making sure again that we specify that **longlat = TRUE** to let the function know we have GPS coordinates. Then we can use the **nbdists()** function to determine the distances between our defined neighbours.

```
> dist0.5 <- dnearneigh(coordinates(housing.sample), d1=0, d2=0.5, longlat=TRUE)
> neigh.dist <- nbdists(dist0.5, coordinates(housing.sample), longlat=TRUE)
```

Different metric systems can also be used – the default is calculated in meters however a simple conversion can be used by converting using 1 meter = 0.00062 miles.

We can now define our weight matrix. The weight matrix object in ***R*** is called a **listw** and what we have obtained above is called a neighbourhood (**nb**) object. We transform our **nb** into a **listw** using the **nb2listw()** function. Since we have some connections that may have zero neighbours (*i.e.* not within 500 meters of another dwelling in our sample), then we need to specify the option that **zero.policy = TRUE**.

Simply inserting the **nb** object which identifies all objects within 500 meters of each other, **dist0.5**, into the **nb2listw()** function (**SW1**) returns a spatial weight of {0, 1}; and inserting the **knn2nb()** function where **k = 20** is the number of neighbours (**SW3**), gives us the nearest neighbour matrix. If we want to weight the neighbours by distance, we specify in the **glist** option that we are weighing each neighbourhood link by **1/x** (**SW2**). We further need to specify that **style="W"** to indicate that we want our spatial weight matrix to be row-standardized.

```
> SW1 <- nb2listw(dist0.5, style="W", zero.policy=TRUE)
> SW2 <- nb2listw(dist0.5, glist = lapply(neigh.dist, function(x) (1/x)),
  style="W", zero.policy=TRUE)
> SW3 <- nb2listw(knn2nb(knearneigh(coordinates(housing.sample), k=20)),
  zero.policy=TRUE)

> summary(SW1, zero.policy=TRUE)
# ...
> summary(SW2, zero.policy=TRUE)
# ...
> summary(SW3, zero.policy=TRUE)
# ...
```

```
> plot(census.freguesias)
> plot(SW1, coordinates(housing.sample), add=T, col="grey")
# ...
> plot(census.freguesias)
> plot(SW3, coordinates(housing.sample), add=T, col="grey")
# ...
```

Exercises:

Ex 10.01 – Why are we not plotting **SW2**? What is the difference between **SW1** and **SW2**?

Ex 10.02 – How could we define a spatial weight matrix that uses every other dwelling as a potential neighbour? Try using the inverse distance to each other property in the sample as another type of spatial weight.

Lesson 11: Spatial Diagnostics and Model Selection

For each of our chosen spatial weight matrices we must now run the diagnostics to test whether (conditional on our definition of neighbours) there is any unaccounted for spatial correlation in the data.

There are two sets of spatial diagnostics that we must run, the first being the Moran I and the second being the Lagrange Multiplier tests for spatial dependence.

The Moran I test is a test of general spatial dependence. This tells us whether or not we have significant clustering of our data points (and values) across space. There are two ways to test the Moran I statistics in **R** - we could first look at applying the statistic directly to the values of a variable (log price) over space and specifying the spatial weight matrix. This would tell us whether or not we see any significant clustering of housing prices, the dependent variable, over space (i.e., whether we observe higher priced dwellings located closer to other higher priced dwellings, and vice versa). To test the spatial clustering of the dependent variable directly, we use the **`moran.test()`** function.

The second way in which we can test for general spatial dependence using the Moran I is to apply the statistic to the residuals of the preferred OLS specification. This would tell us whether or not our OLS model is appropriately capturing all the spatial dependence in the data. If the OLS model was sufficient, then we wouldn't expect there to be any significantly correlation across space coming from the leftover residuals of the model.

The **`moran.test()`** function from above is only appropriate to use for checking spatial dependence in a variable. To correctly check for spatial dependence in the residuals, we need to take into account that we have estimated a linear model and we use the **`lm.morantest()`** function applied to the model of interest.

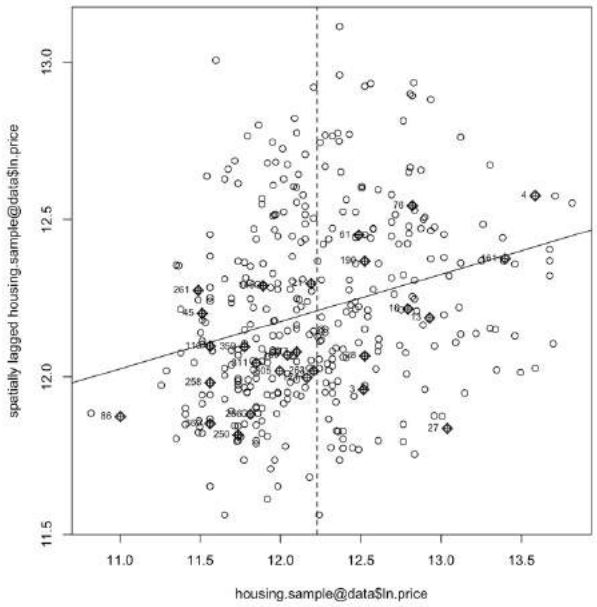
We are specifying a few different parameters. Firstly, we specify neighbourhood relationship which we define according to our different spatial weight matrices. The **`alternative="greater"`** test hypothesis means that we are testing whether we have significant positive spatial clustering - high prices clustered closer to other high price dwellings. Lastly, we are also specifying that **`zero.policy=TRUE`**. We need to specify this in cases where dwellings potentially have zero neighbours in order to avoid getting an error.

```
> global.model <- OLS.spfe[[6]]

> moran.test(housing.sample@data$ln.price, listw=SW1, alternative="greater",
  zero.policy=TRUE)
# ...
> lm.morantest(global.model, listw=SW1, alternative="greater", zero.policy=TRUE)
# ...
```

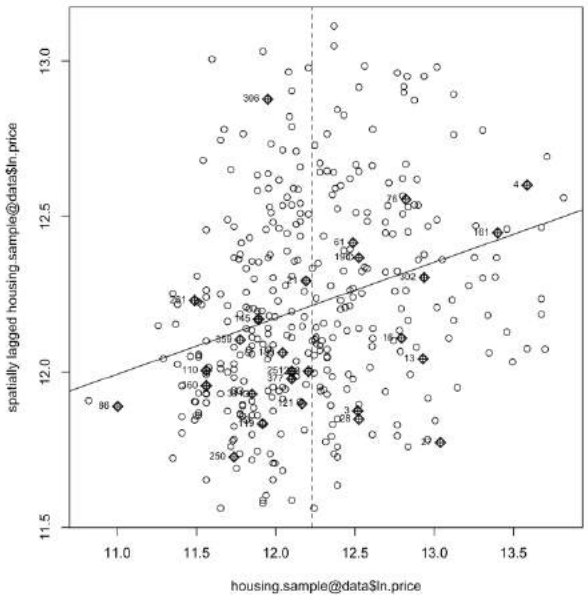
```
> moran.test(housing.sample@data$ln.price, listw=SW2, alternative="greater",
  zero.policy=TRUE)
# ...
> lm.morantest(global.model, listw=SW2, alternative="greater", zero.policy=TRUE)
# ...


> moran.test(housing.sample@data$ln.price, listw=SW3, alternative="greater",
  zero.policy=TRUE)
# ...
> lm.morantest(global.model, listw=SW3, alternative="greater", zero.policy=TRUE)
# ...
```

```
> moran.plot(housing.sample@data$ln.price, SW2, zero.policy=FALSE)
# ...
> moran.plot(global.model$residuals, SW1, zero.policy=TRUE)
# ...
```

Moran I: SW2 (Dep.)



Moran I: SW1 (Error)



We are always concerned about spatial dependence which may be influencing our model via the dependent (price) variable, or spatial dependence which may be influencing our model via the error (unobservable) component. While the above Moran I tests give an indication on whether significant dependence exists, we do not have much information on specifically where this dependence exists - from the dependent variable or from the unobserved error.

We run LM test statistics for spatial dependence which takes into account the testing of both types of spatial dependence simultaneously. These sets of tests provides five different test statistics:

- LM Error - indicates whether significant spatial dependence exists through the error term.
- LM Lag - indicates whether significant spatial dependence exists through the lag of the dependent variable.
- Robust LM Error - indicates whether significant spatial dependence exists through the error term while simultaneously controlling for spatial dependence from the spatial lag of the dependent variable.
- Robust LM Lag- indicates whether significant spatial dependence exists through the lag of the dependent variable while simultaneously controlling for spatial dependence from the error term.
- Joint Error and Lag - indicates whether significant spatial dependence exists jointly from both the error and lag component.

The results of these test statistics will determine which (if any) spatial model is appropriate to estimate. If the test statistics on the lag component (and robust version) are significant and outweigh the error test statistics, then we should be estimating a spatial lag (autoregressive) model (SAR). If the test statistic on the error component (and robust version) are significant and outweigh the lag test statistics, then we should be estimating a spatial error model (SEM).

If both robust versions of the test are significant, and the joint test is also significant, then a joint spatial autoregressive mode with autoregressive errors (SARAR) is appropriate.

```
> lm.LMtests(global.model, listw=SW1, test="all", zero.policy=TRUE)
# ...
> lm.LMtests(global.model, listw=SW2, test="all", zero.policy=TRUE)
# ...
> lm.LMtests(global.model, listw=SW3, test="all", zero.policy=TRUE)
# ...
```

Exercises:

Ex 11.01 – Is it necessary to specify the **`zero.policy=TRUE`** option for the **`SW3`** weight matrix? Why or why not?

Ex 11.02 – Knowing that the Moran I plot is simply a scatter plot of the dependent variable and it's spatial lag, how can we manually plot this in ggplot using the code **`lag.listw(SW3, housing.sample@data$ln.price, zero.policy=TRUE, NAOK=TRUE)`** to manually generate a value of the spatial lag directly in the database.

Ex 11.03 – Compare the spatial diagnostics with and without the location fixed effects **`BairroZone`** included in the global model. Does the inclusion of these fixed effects influence the spatial dependence in the model?

Ex 11.04 – Could we use **`lapply`** to speed up the running of the Moran I and LM Spatial Test diagnostics? – As opposed to coding each model (lag) separately.

Lesson 12: Models of Spatial Dependence (SEM, SAR, SARAR)

Now that we have tested (and confirmed) that spatial dependence exists in our data, we can use the diagnostics above to decide which of the spatial models are the most appropriate to estimate. We will focus on three advanced spatial models which differ in terms of through which channel (dependent variable or residuals) the spatial dependence is observed.

The first will be the Spatial Error Model (SEM). This model accounts for spatial dependence which is observed through the error term. This model has the assumption that there are remaining underlying spatially varying influences which are missing from the model but are important to include. Some examples which could arise in the housing price specification that we are interested in includes omitted spatial variables such as controlling for areas of the city which are more prone to crime, with many derelict buildings, or with significant cultural or historic difference.

It is always advised to control for as many locational aspects as possible, or by including appropriate spatial fixed effects to capture these underlying influences. It may not always be possible to include a large number of control variables if the sample size of the data is small, and so it is important to always check and correct for these unobserved spatial influences which would be picked up by the error term. The SEM specification has the following format:

$$ln(\boldsymbol{P}) = \beta 0 + \beta 1 * \boldsymbol{X} + \beta 2 * \boldsymbol{N} + \beta 3 * \boldsymbol{L} + \beta 4 * \boldsymbol{F} + \lambda \boldsymbol{W}\varepsilon + u$$

Where the standard error component is now broken down into two parts - the first which is capturing the spatial network connections by pre-multiplying by our spatial weight matrix $\boldsymbol{W}$ and with a respective estimated *spatial spillover parameter* of λ. The second component of the error term is a normally distributed part, u.

A Note on The Spatial Weight Matrix:

When we multiply any variable vector by the spatial weight matrix, for example $\boldsymbol{W}x$, what we receive is the average value of x from all my neighbours - depending on how these neighbours are defined. We row-standardize our weight matrix, so each element in the weight matrix represents the percentage of my entire network that each neighbour takes up. So if I have 4 neighbours for example and our definition of neighbours is a {0, 1} indicator, then each neighbour receives a weight of 25% and the value of $\boldsymbol{W}x$ will be 25% of the value of x for each neighbour. If we instead use a distance based weight, then these proportions will be weighted by the proximity of dwellings.

The **`errorsarlm()`** function estimates a maximum likelihood estimation of spatial autoregressive error models. Importantly, we need to specify the spatial weight matrix that we are considering and also use the **`method`** parameter to specify how the function should handle the inversion of the weight matrix to estimate the parameters. We are saying to use a Monte Carlo (MC) approach to estimate the eigenvalues to use for the inversion. For small databases, there may be no difference between using an estimation to run the inversion, but in larger sets the difference between trying to calculate the eigenvalues directly or using an approximation could be a difference of days in the estimation.

We can also test and control for heteroskedasticity in spatial models to make sure the variance in the residuals is not driving any estimated impacts. We do this with the spatial version of the Breusch-Pagan test statistics, and if necessary implement robust standard errors using the **`coeftest()`** function as specified.

```
> SEM.SW3 <- errorsarlm(global.model, data=housing.sample, listw=SW3,
  method="MC", zero.policy=TRUE, na.action=na.omit)
> bptest.sarlm(SEM.SW3)
# ...
> coeftest(lm(SEM.SW3$tary ~ SEM.SW3$tarX - 1), vcov=vcovHC(lm(SEM.SW3$tary ~
  SEM.SW3$tarX - 1), type="HC0"), df=Inf)
# ...
> summary(SEM.SW3)
# ...
```

For each spatial model, specific or additional diagnostics can be generated to compare the spatial model to its OLS equivalent. If the model is correctly controlling for the spatial influences, then we should see an improvement in the model fit relative to the OLS versions – in terms of the SSE, AIC, Log-Likelihood among others.

```
> SEM.SW3$SSE
# ...
> SEM.SW3$s2
# ...
> SEM.SW3$LL
# ...
> SEM.SW3$logLik_lm.model
# ...
> AIC(SEM.SW3)
# ...
> SEM.SW3$AIC_lm.model
# ...
```

The second model we will look at is the Spatial Lag Model, also known as the Spatial Autoregressive Model (SAR). This model captures direct spatial dependence in the dependent variable, in our case the log of housing prices. If our dependent variable exhibits spatial correlation, then we must correct it by modelling directly the spatial lag. This type of mechanism might occur if there is a direct relationship between the price setting behaviour across space - if for example price setters (realtors) use the price of neighbouring dwellings in determining what price to set for their dwelling. The SAR specification has the following format:

$$ln(\boldsymbol{P}) = \beta 0 + \beta 1 * \boldsymbol{X} + \beta 2 * \boldsymbol{N} + \beta 3 * \boldsymbol{L} + \beta 4 * \boldsymbol{F} + \rho \boldsymbol{W} ln(\boldsymbol{P}) + \varepsilon$$

Here we are directly including a lag of the dependent variable, $\boldsymbol{Wln(P)}$. The parameter ρ captures the strength of the direct spatial spillover between property values.

The **lagsarlm()** function is used to obtain the maximum likelihood estimation of the spatial autoregressive lag model, where again we are estimating this for the **SW1** matrix and specifying that we use a Monte Carlo process to invert our weight matrix and obtain the eigenvalues.

One difference between the Spatial Error and the Spatial Lag model is in the interpretation of the estimates. Notice that in the SAR model we have a lag of the dependent variable which we do not have in the SEM model. The parameter estimates that we get from these sorts of models represent the impact on housing prices that a change in a structural or locational characteristic would have. Because this change in a structural characteristic would influence price, and price is further located directly in the model, there is a sort of feedback effect that we need to account for.

We can use the trace of the weight matrix to decompose the estimated effects from these models into a direct effect which comes from the variable of interest, and an indirect effect which comes from this feedback effect. Whenever the direct spatial lag is included in the model, it is necessary to decompose the effect into direct and indirect effects in order to have the correct interpretation. We decompose these effects using the **impacts()** function and specifying how to obtain the trace of our matrix.

```
> SAR.SW3 <- lagsarlm(global.model, data=housing.sample, listw=SW3, method="MC",
  zero.policy=TRUE, na.action=na.omit)
> bptest.sarlm(SAR.SW3)
# ...
> coeftest(lm(SAR.SW3$tary ~ SAR.SW3$tarX - 1), vcov=vcovHC(lm(SAR.SW3$tary ~
  SAR.SW3$tarX - 1), type="HC0"), df=Inf)
# ...
> summary(SAR.SW3)
# ...
> impacts(SAR.SW3, tr=trW(forceSymmetric(as(SW3, "CsparseMatrix")), m=50, p=100,
  type="MC"), zstats=T)
# ...
```

```
> SAR.SW3$SSE
# ...
> SAR.SW3$s2
# ...
> SAR.SW3$LL
# ...
> SAR.SW3$logLik_lm.model
# ...
> AIC(SAR.SW3)
# ...
> SAR.SW3$AIC_lm.model
# ...
```

Lastly, we can specify the joint Spatial Autoregressive Model with Autoregressive Errors (SARAR) model. This specification includes both a direct spatial spillover effect coming from

the dependent variable and an error component capturing the indirect unobservable characteristics. The SARAR specification is written as follows:

$$ln(\boldsymbol{P}) = \beta 0 + \beta 1 * \boldsymbol{X} + \beta 2 * \boldsymbol{N} + \beta 3 * \boldsymbol{L} + \beta 4 * \boldsymbol{F} + \rho \boldsymbol{W} ln(\boldsymbol{P}) + \lambda \boldsymbol{W} \varepsilon + u$$

If there is reason to believe that both types of spatial autocorrelation exists (dependent and error), and if the joint LM diagnostics are significant, then there is rationale behind using the SARAR model.

Since there is a direct lag of the dependent variable included in this specification, we must also decompose the total effects here into the direct and indirect effects, as completed in the SAR specification.

```
> SARAR.SW3 <- sacsarlm(global.model, data=housing.sample, listw=SW3,
  method="MC", zero.policy=TRUE, na.action=na.omit)
> bptest.sarlm(SARAR.SW3)
# ...
> coeftest(lm(SARAR.SW3$tary ~ SARAR.SW3$tarX - 1), vcov=vcovHC(lm(SARAR.SW3$tary
  ~ SARAR.SW3$tarX - 1), type="HC0"), df=Inf)
# ...
> summary(SARAR.SW3)
# ...
> impacts(SARAR.SW3, tr=trW(forceSymmetric(as(SW3, "CsparseMatrix")), m=50,
  p=100, type="MC"), zstats=T)
# ...
```

```
> SARAR.SW3$SSE
# ...
> SARAR.SW3$s2
# ...
> SARAR.SW3$LL
# ...
> SARAR.SW3$logLik_lm.model
# ...
> AIC(SARAR.SW3)
# ...
> SARAR.SW3$AIC_lm.model
# ...
```

Exercises:

Ex 12.01 – While we only conduct the spatial model for one of our weight matrices of interest (SW1), we should in fact check that our estimated values do not vary significantly or change signs from one model to another. The choice of spatial weight definition should not significantly change the results. Compare the estimates using the SW2 or SW3 spatial weights.

References, Resources for *R*

R Website…….....…..https://www.r-project.org/
*Downloading **R**, tutorials, package descriptions, etc.*

R Studio Website.....https://www.rstudio.com/
*Download and instructions for **R** Studio.*

R Packages…….....https://cran.r-project.org/web/packages/available_packages_by_name.html
*Comprehensive list of all available packages in **R** and a short description of what they do.*

R Cheatsheets……...……....https://www.rstudio.com/resources/cheatsheets/
*Cheat sheets with relevant functions and usages for all sorts of applications in **R**: basic functions, data cleaning, working with variables/ characters, data visualization, mapping, etc.*

"***R*** For Data Science" by Wickham & Grolemund…….....https://r4ds.had.co.nz/index.html
*Great (free) online text detailing step by step instructions for cleaning, organizing and working with different types of data in **R**.*

"Spatial Regression Analysis in ***R***" by Anselin
……...http://csiss.org/GISPopSci/workshops/2011/PSU/readings/W15_Anselin2007.pdf

"Introduction to GIS with R" by Jesse Adler
.....https://www.jessesadler.com/post/gis-with-r-intro/
*Detailed blog outlining how to work with and use spatial data in R, focusing on the **`sp`** and **`sf`** packages.*

"Intro to GIS and Spatial Analysis" by Manuel Gimond
…….....https://mgimond.github.io/Spatial/index.html
*Online textbook explaining the theory and background of GIS and spatial analysis. Technical appendix outlines how to conduct many of the spatial techniques using **R**.*

"Econometrics Beat" by Dave Giles.....https://davegiles.blogspot.com/
Online blog with a focus on econometrics, diagnostics, and modelling. Some accompanying R code is provided to help replicate these functions and interpret their results.

Stackoverflow Forums…….....…….....https://stackoverflow.com/
*Forums for discussion of **R** related coding issues, packages, guides, etc. Great to read past forum posts to see how problems were solved, or for posting your own issue if it is not yet been discussed.*

ggPlot Guide for Data Visualization…….....https://ggplot2.tidyverse.org/index.html
Guide for creating and working with many different ways to visualize (graph, plot, map) data.

R Color Visualization Guide.....http://www.stat.columbia.edu/~tzheng/files/Rcolor.pdf
List of all color possibilities to use in plotting and their name/ code to use.