Advanced Geospatial Data Analysis with R – Applications in Geosciences

Dr. Marj Tonini, Haokun Liu

09/12/2024

Contents

Preface

Earth surface environmental processes exhibit distinctive characteristics, encompassing both spatial and temporal dimensions, along with various attributes and predictive variables. Coupled with uncertainty and complexity issues, all this contributes to make this field of research highly challenging. Furthermore, in the era of Data Science, the wealth of available data and the rapid development of analytical models have emerged as distinctive aspects in the realm of **Advanced Geospatial Data Analysis (AGDA)**. The domain of AGDA encompasses data exploration, manipulation, and modelling, from the acquisition phase up to the visualization and interpretation of the results. Mapping where events are located and how they relate to each other provides a better understanding of the process being studied. Finally, as an increasing volume of geo-environmental data becomes more and more accessible, the demand for spatial data scientists is growing rapidly, both in public research institutions as well as in private companies.

The term "**Data Science**" was defined for the first time by Peter Naur in 1974 as "*the science of dealing with data, once they have been established, while the relation of the data to what they represent is delegated to other fields and sciences*" (**?**). In essence, this definition frames Data Science as the technical and methodological handling of data, while their initial creation or gathering, as well as the interpretation of its meaning in the broader context, is not the primary focus of this discipline. In 1977 the International Association for Statistical Computing (IASC) further refined the concept of Data Science, emphasizing its role in the process of transforming raw data into valuable information and actionable knowledge. The IASC articulated that Data Science involves not only the collection and processing of data, but also the application of statistical methods and computational techniques to extract meaningful insights, which can then be used to support decision-making and drive innovation across various domains.

In disciplines like environmental and earth Sciences, physical geography, humanities and social sciences, the use of Data Science procedures is emerging only recently, proving to be extremely efficient to deal with the complexity of the investigated process and the heterogeneity of the underlying data sources (**?**). This leads to a cultural shift, moving scientists away from individual working within their own research domain. Indeed, disciplinary boundaries are more and more permeable, pushing scientists to be more open to collaborate among them and with decision makers on the investigation and understanding of real-world problems. Modern earth and environmental scientists need to interact with other disciplines, apparently far from their domain. This openness is increasingly important as society struggles to respond to the implication of anthropogenic pressures on different issues, such as natural hazards and climate change, or the harmful impacts of human activities on biodiversity, water and air quality, human health.

Book overview

This eBook primarily focuses on introducing analytical techniques from the field of Data Science to analyze and understand environmental processes occurring on Earth's surface. Mastering the methodologies in this multidisciplinary and rapidly evolving field—encompassing statistics, mathematics, geomatics, and computer science—can be a challenging endeavor. To address this, the eBook maintains a rigorous foundation in mathematical and statistical principles while emphasizing practical applications within the domain of Geosciences.

Emphasis is placed on the application of advanced geospatial tools to analyze and model both natural and anthropogenic hazards, as well as land-use processes. Theoretical concepts are reinforced through case studies employing spatial clustering techniques and both supervised and unsupervised machine learning algorithms. Examples include stochastic point process models for spatio-temporal cluster analysis and machine learning-based approaches for geodemographic segmentation, susceptibility assessment, and risk mapping.

Computing lab

Computations are conducted using the R free software environment, which is specifically designed for data analysis and manipulation.

After a brief introduction to the main functionalities in R (see Chapter **??**) and an overview of the basic operations with geospatial data (see Chapter **??**), the rest of the chapters will focus on the following geocomputational techniques:

- *Exploratory Data Analysis and Visualization*: examining geographical variations in statistical data distributions, including Geographically Weighted Summary Statistics (Chapter **??**).
- *Cluster Detection and Mapping*: using global clustering methods, namely Ripley's K-function and Kernel Density Estimator (Chapter **??**).
- *Random Forest*: as example of supervised machine learning algorithm. We present both the global (Chapter **??**) and the local version, and we use Random Forest to introduce interpretability & explainability in machine learning (Chapter **??**) .
- *Self-Organizing Maps*: as example of unsupervised machine learning method for data clustering/segmentation and visualization (Chapter **??**).
- *Feature detection*: using DBSCAN for identifying and visualizing clustering features (Chapter **??**).

To complete the applied computing labs presented in each chapter, readers can download the datasets from here: dataset

Target audience

The target audience includes master's and PhD students in Earth and Environmental Sciences, Biology and Spatial Ecology, Physical Geography, and related disciplines. Our goal is to empower students by guiding them through both theoretical knowledge and hands-on practical applications, helping them develop strong problem-solving skills.

This eBook aims to equip the audience with:

- A solid understanding of key practical concepts and applied aspects in AGDA.
- Advanced tools to effectively navigate and analyze spatial datasets in Geosciences.

The eBook is designed for intermediate to advanced R users with previous experience in geospatial data analyses and a keen interest in geocomputing. If you have only a basic knowledge of these areas, we strongly

encourage you to explore the essential references provided in each chapter. These resources offer valuable documentation and additional materials to help deepen your understanding.

The main prerequisites are the following:

- Knowledge of basic statistics: methods of descriptive statistics (measures of central tendency and dispersion); how to assess relationships between variables; concepts of correlation and regression.
- Basic knowledge in geomatics (GIS): basic operations with raster and vector datasets.
- R programming basics and RStudio.

Authors information

Marj Tonini is a spatial data scientist with a profound expertise in geospatial modeling for risk assessment, particularly in relation to wildfires and landslides. She earned her Ph.D. in 2002 from the Sant'Anna School of Advanced Studies in Pisa, Italy, where she developed a thesis on agro-environmental modeling that laid the foundation for her career. In 2004, she joined the University of Lausanne (UNIL) as a postdoctoral researcher in geospatial data analysis, and by 2008, she was appointed as Senior Research Manager at the Institute of Earth Surface Dynamics, where she continues to serve in her current role. Sine 2022 she has been servig a four-year term as the director of the Swiss Geocomputing Centre of the Faculty of Geosciences and environment (UNIL)

Marj's research focuses on the development of innovative methodologies that facilitate the efficient extraction of knowledge from complex environmental datasets. Her work aims to estabilishing a robust methodological framework to understand the spatio-temporal dynamics of environmental processes and to evaluate the influence of various predictor variables. Her current research focuses on the analysis of land use and land cover changes, alongside the development of predictive scenarios and the assessment of susceptibility and risks associated with natural hazards. Through this work she seeks to advance the field of geospatial science by translating data-driven insights into practical solutions for managing and mitigating environmental risks. You can find the full list of her publications on ResearchGate and Google Scholar.

Marj has written the vast majority of this book. Her contributions include the development of the conceptual framework, writing the theoretical background chapters, developing case studies, designing methodologies, conducting data collection and analysis, and software/tool development. The following section "*My Journey of Learning*" refers to her personal experience.

Haokun Liu is a Ph.D. student at the Group of Cities and Dynamics of Networks, University of Lausanne. In addition to his doctoral studies, he serves as a student assistant at the Swiss Geocomputing Center. Benefiting from rigorous and comprehensive training in both China and Switzerland, Haokun has developed a diverse research portfolio. His primary research interests and experience encompass a wide range of interdisciplinary fields, including urban analytics, health geography, and spatial data science. His work focuses on leveraging advanced computational techniques to address complex urban and environmental challenges, bridging the gap between data-driven insights and real-world applications.

Haokun's contributions were instrumental in establishing and managing the GitHub repository used to host and maintain this eBook. He also played a key role in data visualization and mapping, and provided valuable revisions to the applied computing labs.

My Journey of Learning: Integrating Geospatial Data Analysis with Data Science

In the early 2000s, as I (Marj) embarked on my PhD journey, a new frontier was beginning to unfold in Geosciences. Geographical information systems (GIS) were just starting to make waves, offering a fresh perspective on how researchers could investigate and understand the environment. It was through my thesis on the effects of spreading olive vegetation water on agricultural land that I first delved into the world of

GIS. This early exposure was more than just a technical skill; it was a revelation that would drive me to explore the depths of spatial analysis for years to come.

After completing my PhD, my journey led me to a PostDoc position where I had the privile of working with Professor Mikhail Kanevski, a true pioneer in the fields of environmental data mining, geostatistics, and the emerging field of applying machine learning to spatial environmental data. In this dynamic research team, I found myself diving headfirst into the world of spatial data science, driven by the desire to uncover valuable insights from the complex data that described our environment. The work was challenging, yet it was also exhilarating, as every new technique I learned brought me closer to understanding the intricate tapestry of environmental phenomena.

As the years passed, the tools and resources available to researchers like me grew exponentially. The development and widespread adoption of free programming languages like and R and Python revolutionized the way we approached statistical analysis and machine learning. What was once the domain of a select few became accessible to many, thanks to the wealth of pre-developed scripts and resources these languages offered. For me, and for countless others, this accessibility was a game-changer. It meant that advanced techniques were no longer out of reach, and it fostered a culture of greater reproducibility in research. By utilizing standardized, widely adopted open-source programming languages and cloud-based platforms for hosting and managing Git repositories, such as GitHub, researchers can now more easily share their code and methodologies, enabling others to replicate and validate their findings. This collaborative spirit has enhanced the reliability and credibility of scientific research and fostered a community where knowledge is shared and expanded more efficiently.

Reflecting on this journey, I realize that each step – from my early days with GIS to my current role as an independent lead researcher in environmental spatial data analysis in the era of Data Science – has been a chapter in a larger story of discovery and innovation. It's a story of how spatial data and advanced analytics have become indispensable tools in our quest to understand and protect the environment, and it's a story that continues to unfold with each new challenge and breakthrough.

Acknowledgements

The case studies presented in each chapter came from different projects carried out in collaboration with several colleagues, including master and PhD students. All the produced scientific papers are duly cited in the bibliography.

I would like to express my gratitude to Professor Mário Gonzalez Pereira and Dr. Joana Parente, for their extensive and fruitful collaboration in investigating the spatio-temporal distribution of wildfires in Portugal. One of our studies, which explores the evolution of forest fires from spatio-temporal point events to smoothed density maps, forms an integral part of Chapters 3 and 4.

I also thank Professor Stuart Lane and Dr. Natan Micheletti for introducing me to the fascinating world of rock glacier research. Notably, the 3D point cloud dataset analyzed in Chapter 8 was acquired and processed by Natan during his PhD studies.

My thanks also go to Julien Riese, who produced the input dataset and collaborated with me in developing the code that assesses landslide susceptibility in Canton Vaud, the main focus of Chapter 5. The same for Axelle Bersier for her meticulous work in acquiring and pre-processing the Swiss national population census dataset, which is used for the exercise on unsupervised learning in Chapter 7. Both Julien and Axelle exemplify the high caliber of master's students I have had the pleasure of supervising.

For transparency, I acknowledge the use of ChatGPT in assisting with the reformulation of certain sentences in this book and DALL-E to generate the image icon.

1 Introduction to R

1.1 R Language

R is a complete programming language and software environment for statistical computing and graphical representation. As part of the GNU Project – free software, mass collaboration project – (https://www.gnu.org/software/software.en.html), the source code is freely available.

For more details on R see https://www.r-project.org/.

1.1.1 R Packages

Functionalities in R can be expanded by importing packages. A package is a collection of R functions, data and compiled code. The location where the packages are stored is called the "library". If there is a particular functionality that you require, you can download the package from the appropriate site and it will be stored in your library.

In all operation systems the function `install.packages()` can be used to download and install a package automatically. Otherwise, a package already installed in R can be loaded in a session by using the command `library(package_name)`. When you open an R Markdown document (*.Rmd*) the program propose you automatically to install the libraries listed there.

1.1.2 Some tips

- R is case sensitive!
- Previously used command can be recalled in the console by using the up arrow on the keyboard.
- The working directory by default is "`C:/user/.../Documents`".
  - It can be found using the command `getwd()`
  - It can be changed using the command line `setwd("C:/Your/own/path")`
- In R Markdown the working directory when evaluating R code chunks is the directory of the input document by default.
  - To access to a specific file in a sub-folder use "`./subfolder/file.ext`"
  - To access to a specific file in a up-folder use "`../upfolder/file.ext`"

1.1.3 R Commands (online resources)

Many table resuming the main R commands can be found online.
Here some useful links:

- A short list of the most useful R commands
- Table of Useful R commands
- Basic Commands to Get Started with R

1.2 R Markdown

This is an R Markdown document :-)

Markdown is a simple formatting syntax for authoring HTML, PDF, and MS Word documents. It is a simple and easy to use plain text language allowing to combine R code, results from data analysis (including plots and tables), and comments into a single nicely formatted and reproducible document like a report, publication, thesis chapter or web pages.

Code lines are organized into code blocks, seeking to solve specified tasks, and referred to as "code chunk". For more details on using R Markdown see http://rmarkdown.rstudio.com.

All what you have to do during the computing labs is to read each explanatory paragraph before running each individual R code chunk, one by one, and to interpret the results. Finally, to create a personal document (usually a PDF) from R Markdown, you need to *Knit* the document. Knitting a document simply means taking all the text and code and creating a nicely formatted document.

1.3 Data type in computational analysis

1.3.1 Variables

Variables are used to store values in a computer program. Values can be numbers (real and complex), words (string), matrices, and even tables.

The fundamental or atomic data in R Programming can be:

- **integer**: number without decimals
- **numeric**: number with decimals (*float* or *double* depending on the precision)
- **character**: string, label
- **factors**: a label with a limited number of categories
- **logical**: true/false

Variables	Example
integer	100
numeric	0.05
character	"hello"
logical	TRUE
factor	"Green"

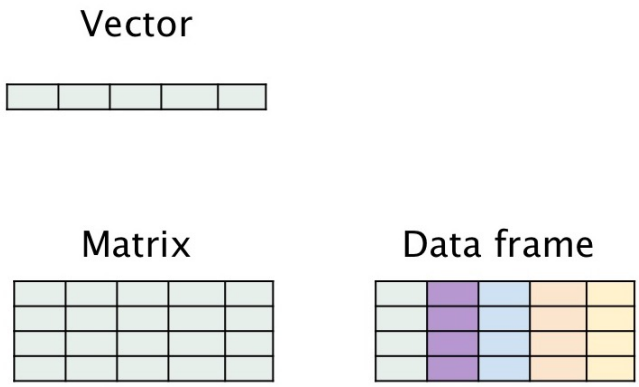


Figure 1: Data Types in R

1.3.2 Data structure in R

R's base data structures can be organised by their dimensionality (1d, 2d, or nd) and whether they are homogeneous (all contents must be of the same type) or heterogeneous (the contents can be of different types). This gives rise to the four data structures most often used in data analysis:

A **Vector** is a one-dimensional structure winch can contain object of one type only: numerical (integer and double), character, and logical.

Dimensions	Homogeneous	Heterogeneous
1d	Atomic vector	List
2d	Matrix	Data frame

Figure 2: Data structures in R

```
# Investigate vector's types:

v1 <- c(0.5, 0.7); v1; typeof(v1)
```

```
## [1] 0.5 0.7
```

```
## [1] "double"
```

```
v2 <-c(1:10); v2; typeof(v2)
```

```
##  [1]  1  2  3  4  5  6  7  8  9 10
```

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

```
v3 <- c(TRUE, FALSE); v3; typeof(v3)
```

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

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

```
v4 <- c("Swiss", "Itay", "France", "Germany"); v4; typeof(v4)
```

```
## [1] "Swiss"   "Itay"    "France"  "Germany"
```

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

```
#Create a sequence from 0 to 5 with a step of 0.5:

v5 <- seq(1, 5, by=0.5); v5; typeof(v5)
```

```
## [1] 1.0 1.5 2.0 2.5 3.0 3.5 4.0 4.5 5.0
```

```
## [1] "double"
```

```
length(v5)
```

```
## [1] 9
```

```r
summary(v5)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##       1       2       3       3       4       5
```

```r
#Extract the third element of the vector
v5[3]
```

```
## [1] 2
```

```r
#Exclude the third element from the vector and save as new vector
v5[-3]
```

```
## [1] 1.0 1.5 2.5 3.0 3.5 4.0 4.5 5.0
```

```r
w5<-v5[-3]; w5
```

```
## [1] 1.0 1.5 2.5 3.0 3.5 4.0 4.5 5.0
```

A **Matrix** is a two-dimensional structure winch can contain object of one type only. The function `matrix()` can be used to construct matrices with specific dimensions.

```r
# Matrix of elements equal to "zero" and dimension 2x5
m1<-matrix(0,2,5); m1  #(two rows by five columns)
```

```
##      [,1] [,2] [,3] [,4] [,5]
## [1,]    0    0    0    0    0
## [2,]    0    0    0    0    0
```

```r
# Matrix of integer elements (1 to 12, 3x4)
m2<-matrix(1:12, 3,4); m2
```

```
##      [,1] [,2] [,3] [,4]
## [1,]    1    4    7   10
## [2,]    2    5    8   11
## [3,]    3    6    9   12
```

```r
# Extract the second row
m2[2, ]
```

```
## [1]  2  5  8 11
```

```r
# Extract the third column
m2[,3]
```

```
## [1] 7 8 9
```

```r
# Extract the the second element of the third column
m2[2,3]
```

```
## [1] 8
```

A **data frame** allows to collect heterogeneous data. All elements must have the same length.

A **list** is a more flexible structure since it can contain variables of different types and lengths. Nevertheless, the preferred structure for statistical analyses and computation is the data frame.

It is a good practice to explore the data frame before performing further computation on the data. This can be simply accomplished by using the commands `str()` to explore the structure of the data and `summary()` to display the summary statistics and quickly summarize the data. For numerical vectors the command `hist()` can be used to plot the basic histogram of the given values.

```r
# Create the vectors with the variables
cities <- c("Berlin", "New York", "Paris", "Tokyo")
area <- c(892, 1214, 105, 2188)
population <- c(3.4, 8.1, 2.1, 12.9)
continent <- c("Europe", "Norh America", "Europe", "Asia")
```

```r
# Concatenate the vectors into a new data frame
df1 <- data.frame(cities, area, population, continent)
df1
```

```
##     cities area population    continent
## 1   Berlin  892        3.4       Europe
## 2 New York 1214        8.1 Norh America
## 3    Paris  105        2.1       Europe
## 4    Tokyo 2188       12.9         Asia
```

```r
#Add a column (e.g., language spoken) using the command "cbind"
df2 <- cbind (df1, "Language" = c ("German", "English", "Freanch", "Japanese"))
df2
```

```
##     cities area population    continent Language
## 1   Berlin  892        3.4       Europe   German
## 2 New York 1214        8.1 Norh America  English
## 3    Paris  105        2.1       Europe  Freanch
## 4    Tokyo 2188       12.9         Asia Japanese
```

```r
#Explore the data frame
str(df2) # see the structure
```

```
## 'data.frame':    4 obs. of  5 variables:
##  $ cities    : chr  "Berlin" "New York" "Paris" "Tokyo"
##  $ area      : num  892 1214 105 2188
##  $ population: num  3.4 8.1 2.1 12.9
##  $ continent : chr  "Europe" "Norh America" "Europe" "Asia"
##  $ Language  : chr  "German" "English" "Freanch" "Japanese"
```

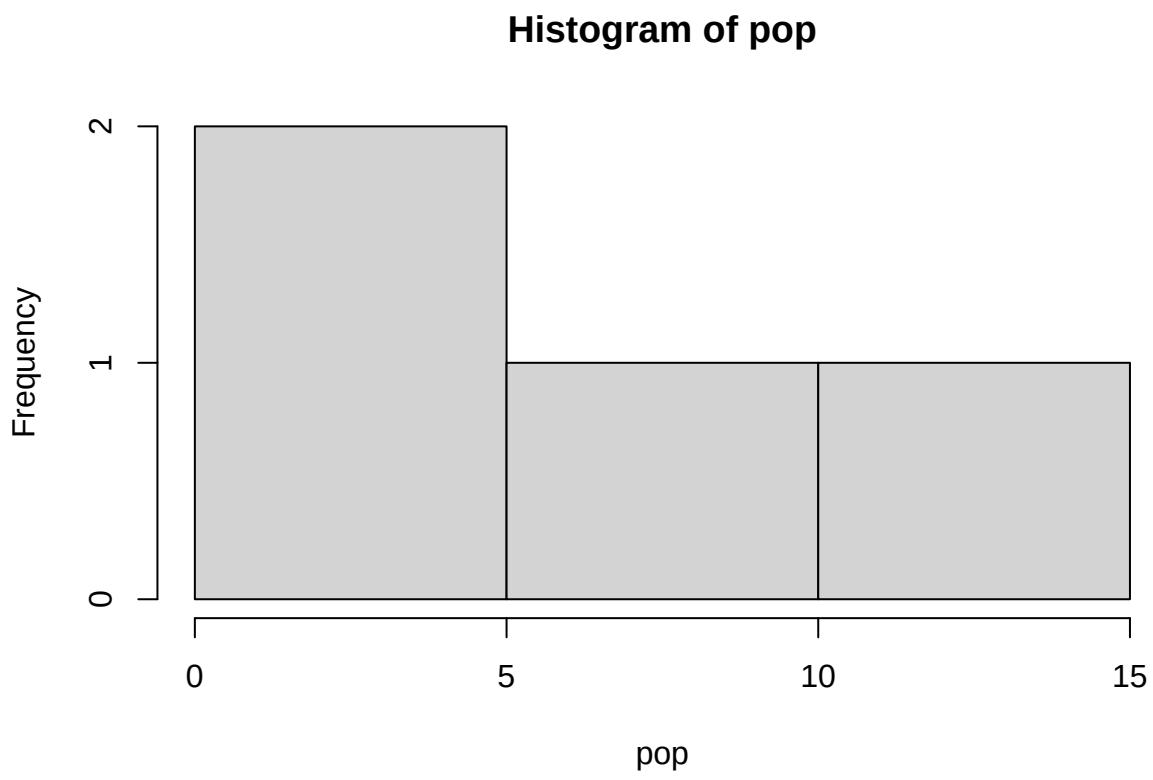
```r
summary(df2) # compute basic statistics
```

```
##     cities                area          population      continent
##  Length:4           Min.   : 105.0   Min.   : 2.100   Length:4
##  Class :character   1st Qu.: 695.2   1st Qu.: 3.075   Class :character
##  Mode  :character   Median :1053.0   Median : 5.750   Mode  :character
##                     Mean   :1099.8   Mean   : 6.625
##                     3rd Qu.:1457.5   3rd Qu.: 9.300
##                     Max.   :2188.0   Max.   :12.900
##    Language
##  Length:4
##  Class :character
##  Mode  :character
##
##
##
```

```r
# Use the symbol "$" to address a particular column
pop<-(df2$population)
pop
```

```
## [1]  3.4  8.1  2.1 12.9
```

```r
hist(pop) # plot the histogram
```



2 Basic operations with geodata in R

2.1 Introduction

Geodata, or geospatial data, refers to features associated with a specific location on the Earth's surface. Geodata can be in various forms and is often used in Geographic Information Systems (GIS) for mapping and analysis. The two key components characterizing geodata are the **spatial component** and **attributes**. The spatial component specifies the location and shape of the features, with different levels of detail, while attributes describe their characteristics.

The two basic formats used to represent the spatial component of geodata are:

1. **Vector**: this format uses points, lines, and polygons to identify each individual features.
2. **Raster**: this format uses a regular grid of pixels to represent the global geographic context.

In addition, **attribute tables** are used to store the characteristics associated with the geospatial features.

The GIS software are specifically designed to help user to edit, manage, analyze, and map geodata. To make working with geodata easier, several packages have been developed in R. These packages allow users to handle geodata directly in R computing environment, without needing separate GIS software.

In this chapter, we introduce the basic functions allowing to work with geodata in the pre-processing and post-processing phase of a pipeline in geomodeling.

2.2 Plotting vector dataset

The spatial component of geodata uses geometric primitives like point, line, and polygon to represent the single features. Each feature in a geodataset is associated with various attributes providing detailed quantitative and qualitative information. A single geodataset includes features of the same type, represented by using the same class of primitive.

The three basic **geometric vector primitives** have the following characteristics:

- *Points*: defined by a single pair of coordinates (x, y) representing a specific location. Used to represent small objects like weather stations, city center, or to identify single features in a geo-hazard inventory (e.g., earthquake's epicenter, landslides location, wildfires, etc.).
- *Lines*: defined by pairs of coordinates connected to each other, representing linear features such as roads or rivers network, pathways, railway, etc.
- *Polygons*: defined by a series of connected coordinates that enclose an area, representing features such as lakes, administrative units, vegetation patches, burned area, landslides footprint.

2.2.1 Package *sf* for vector dataset

The package **sf** (**??**) has been designed to work with vector data as "simple features" in R. Each feature is represented by one row in the data frame, with attributes stored as columns and spatial information stored in a special geometry column.

```
# Load the library sf
library(sf)
```

As a toy example, we will work with the geodataset of administrative boundaries in the Canton of Vaud (Switzerland). This dataset is available in shapefile format, one of the most widely used file formats for vector geospatial data. The field "Munics" represents the municipalities.

A **shapefile** includes multiple files allowing to store different kind of objects:

- **shp*: the features' geometries (i.e., the geometric vector primitive used to describe the features).
- **dbf*: the attributes, describing the characteristics of the features (i.e., tabular information).
- **shx*: shape index format, an index file for the geometry data.
- **prj*: the coordinate reference systems, defining how the geometries are projected on the Earth's surface.

To read a shapefile, you only need to specify the file name with ".shp" extension. However, it is important to have all related files in the same directory to ensures that the shapefile is read correctly and all necessary information is available for the analysis.

Shapefiles can be imported and converted as sf-objects using the command `st_read()`. By setting the argument `quiet = FALSE` suppresses the output from the console when importing geodata.

```
# Load dataset
vaud <- st_read("data/RGIS/Vaud.shp", quiet = FALSE)
```

```
## Reading layer `Vaud' from data source `C:\AGDA_Rbook\data\RGIS\Vaud.shp' using driver `ESRI Shapefil
## Simple feature collection with 315 features and 4 fields
## Geometry type: POLYGON
## Dimension:     XY
## Bounding box:  xmin: 2494306 ymin: 1115149 xmax: 2585462 ymax: 1203493
## Projected CRS: CH1903+ / LV95
```

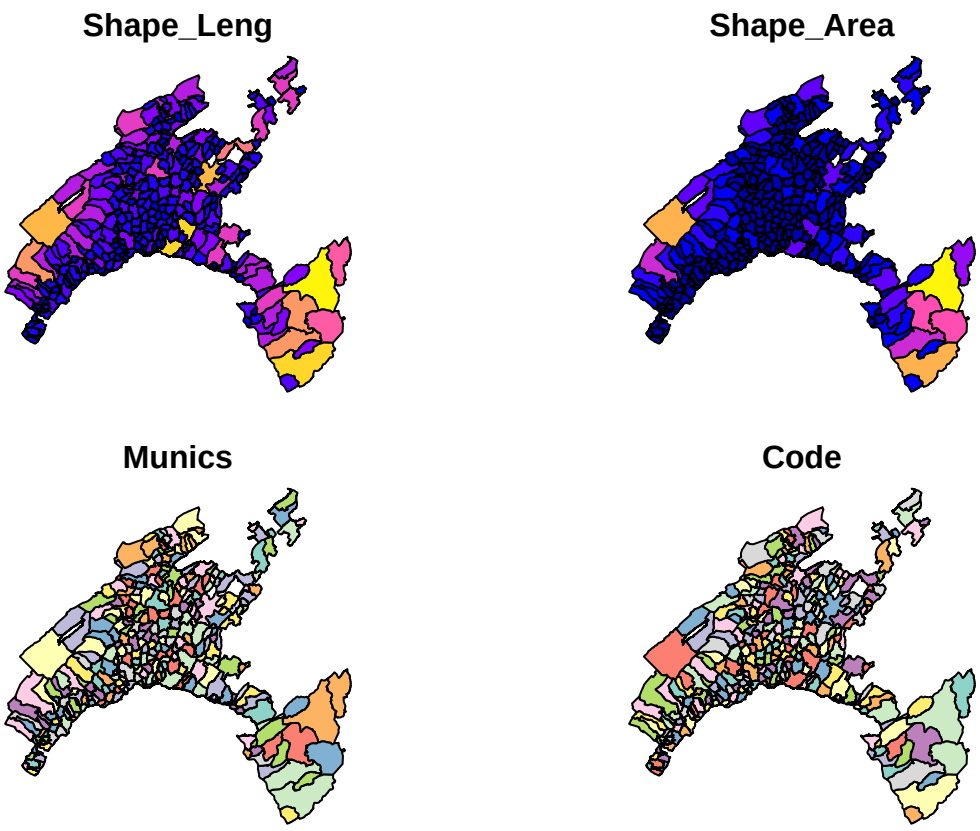
```
# Inspect the attribute table
str(vaud)
```

```
## Classes 'sf' and 'data.frame':   315 obs. of  5 variables:
##  $ Shape_Leng: num  30547 21405 27147 31111 28268 ...
##  $ Shape_Area: num  24178457 23354685 23190781 22858943 22501915 ...
##  $ Munics    : chr  "Payerne" "Rossinière" "Vallorbe" "Puidoux" ...
##  $ Code      : chr  "5822" "5842" "5764" "5607" ...
##  $ geometry  :sfc_POLYGON of length 315; first list element: List of 1
##   ..$ : num [1:783, 1:2] 2559434 2559258 2559286 2559263 2559337 ...
##   ..- attr(*, "class")= chr [1:3] "XY" "POLYGON" "sfg"
##  - attr(*, "sf_column")= chr "geometry"
##  - attr(*, "agr")= Factor w/ 3 levels "constant","aggregate",..: NA NA NA NA
##   ..- attr(*, "names")= chr [1:4] "Shape_Leng" "Shape_Area" "Munics" "Code"
```

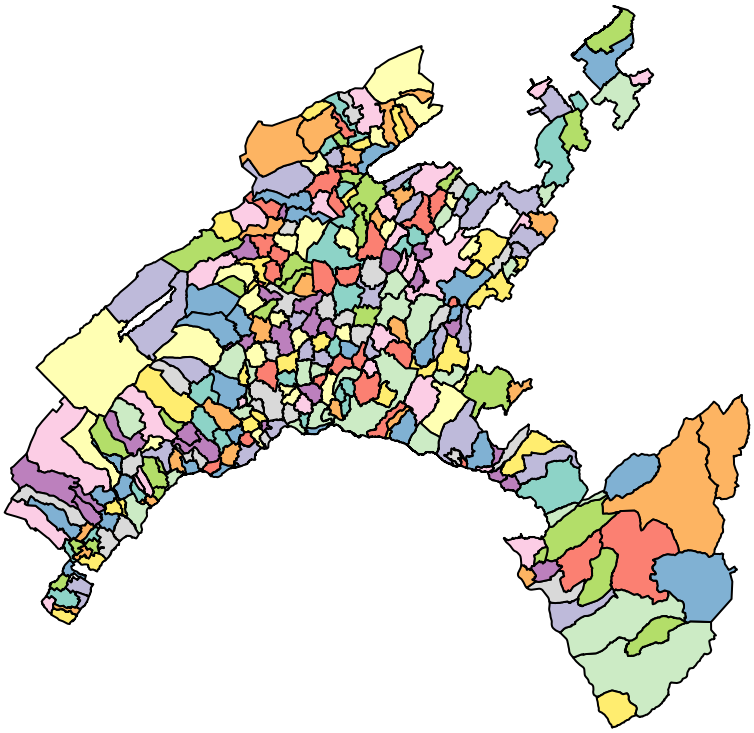
2.2.2 Plot vector features

Basic maps can be created with the command `plot()`. By default this creates a multi-panel plot: one plot for each variable included in the geodata (corresponding to each column). Otherwise, you can specify the name of variable that you wish to display.

```
# Basic plot (diplay all the variables)
plot(vaud)
```

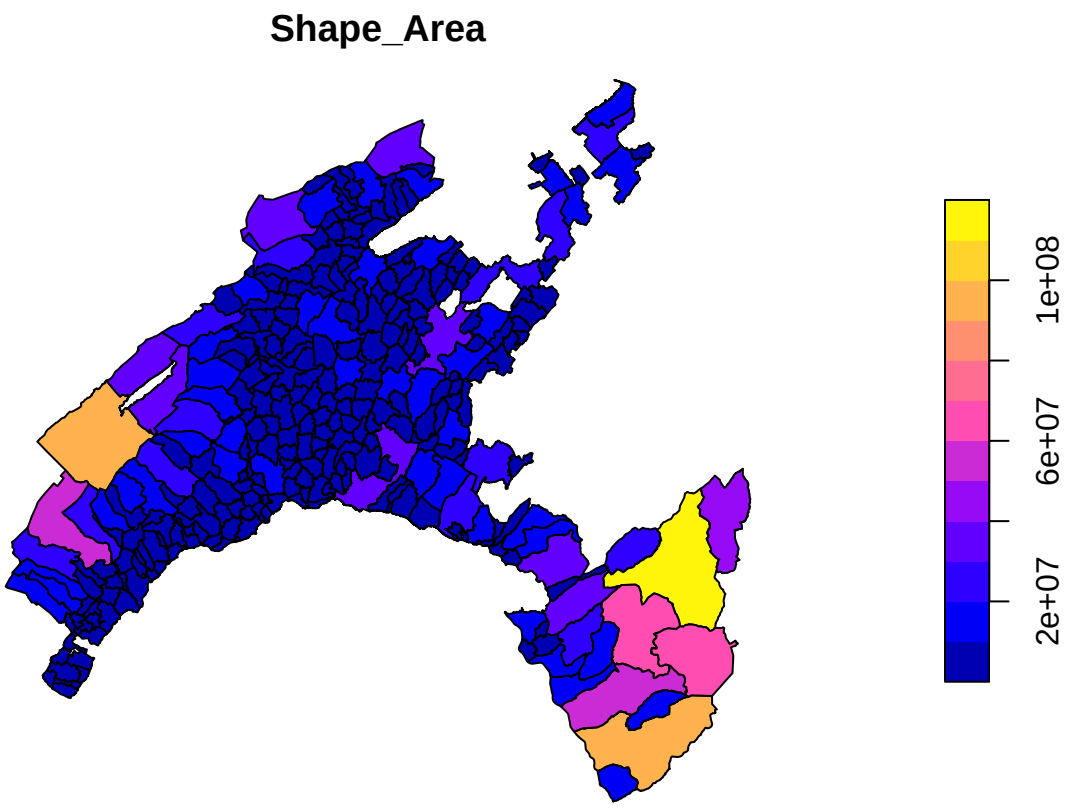



```
# Display each municipality using single color (attributed randomly)
plot(vaud["Munics"], main="Municipal boundaries")
```



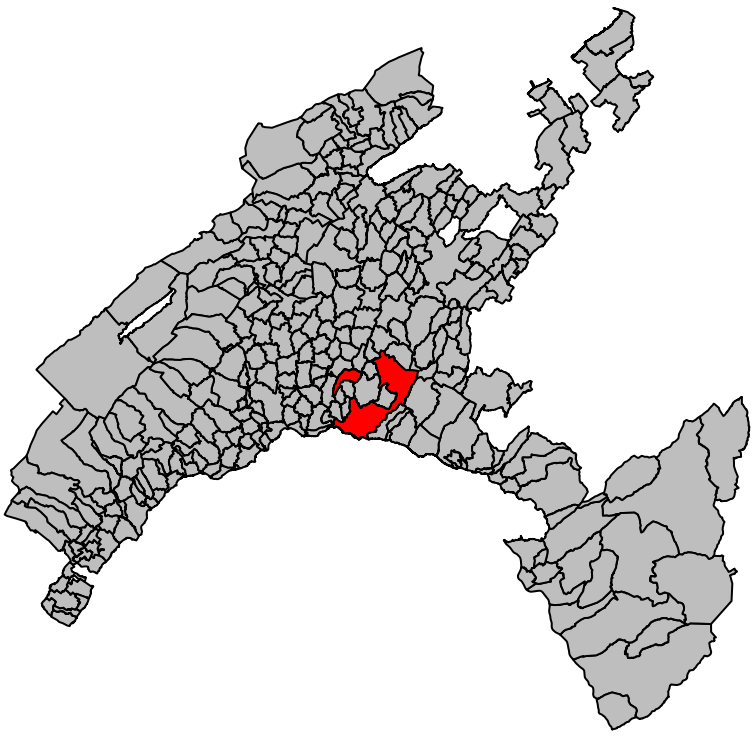
A legend with a continuous color scale is produced by default if the object to be plotted is numeric.

```
# Plot based on the value "area"
plot(vaud["Shape_Area"], reset = FALSE)
```



Different operations can be performed to customize the map. For instance, you can highlight the municipality of Lausanne in red to show its geographic correspondence, keeping all the other municipalities grey.

```
# Extract the boundary of Lausanne
Lausanne = st_union(vaud[vaud$Munics == "Lausanne", ])

# Plot the Lausanne municipal boundaries over the map of Canton of Vaud
plot(vaud["Munics"],col="grey", reset = FALSE, main="Canton Vaud, Lausanne")
plot(Lausanne, add = TRUE, col = "red")
```



2.3 Plotting raster dataset

Raster data are different from vector data in that they are referenced to a regular grid of regular (usually square) cells, called pixels. The spatial characteristics of a raster dataset are defined by its spatial resolution (the height and width of each cell) and its origin (typically the upper left corner of the raster grid, which is associated with a location in a coordinate reference system).

Raster data is highly effective for modeling and visualizing continuous spatial phenomena such as elevation, temperature, and precipitation. Each cell in the grid captures a value representing the attribute at that specific location, allowing for smooth and detailed gradients across the study area. This format is also effective in representing categorical variables such as land cover, where each cell is associated with a class value.

Common **raster formats** used for spatial analyses include:

1. *GeoTIFF (*.tif, *.tiff):*
   - A widely used format that includes geographic metadata such as coordinates and projection information, making it easy to integrate with GIS applications.
2. *ESRI Grid:*
   - A proprietary format developed by ESRI for use with its software, such as ArcGIS. It supports both integer and floating-point grids.
3. *Erdas Imagine (*.img):*
   - A format developed by ERDAS for its Imagine software, often used for remote sensing data and satellite imagery. It supports large files and multiple bands.

4. *NetCDF (*.nc):*
    - Stands for Network Common Data Form, used for array-oriented scientific data, including GIS data. It supports multidimensional data arrays, making it suitable for complex environmental and atmospheric data.
5. *Hierarchical Data Format (HDF):*
    - Similar to NetCDF, HDF is used for managing and storing large amounts of data, especially in scientific computing. It supports various data types and is used for satellite imagery and climate data.
6. *ASCII Grid (*.asc):*
    - A simple, text-based raster format where each cell value is represented by a number in a grid layout. It's easy to read and edit with a text editor.

These formats vary in terms of compression, metadata support, and suitability for different types of raster data, from simple images to complex scientific datasets.

2.3.1 Terra package and raster dataset

The package **terra** provides a variety of specialized classes and functions for importing, processing, analyzing, and visualizing raster datasets (**?**). It is intended to replace the package "raster", which has similar data objects and the function syntax as terra package. However, the terra package contains several major improvements, including faster processing speed for large raster.

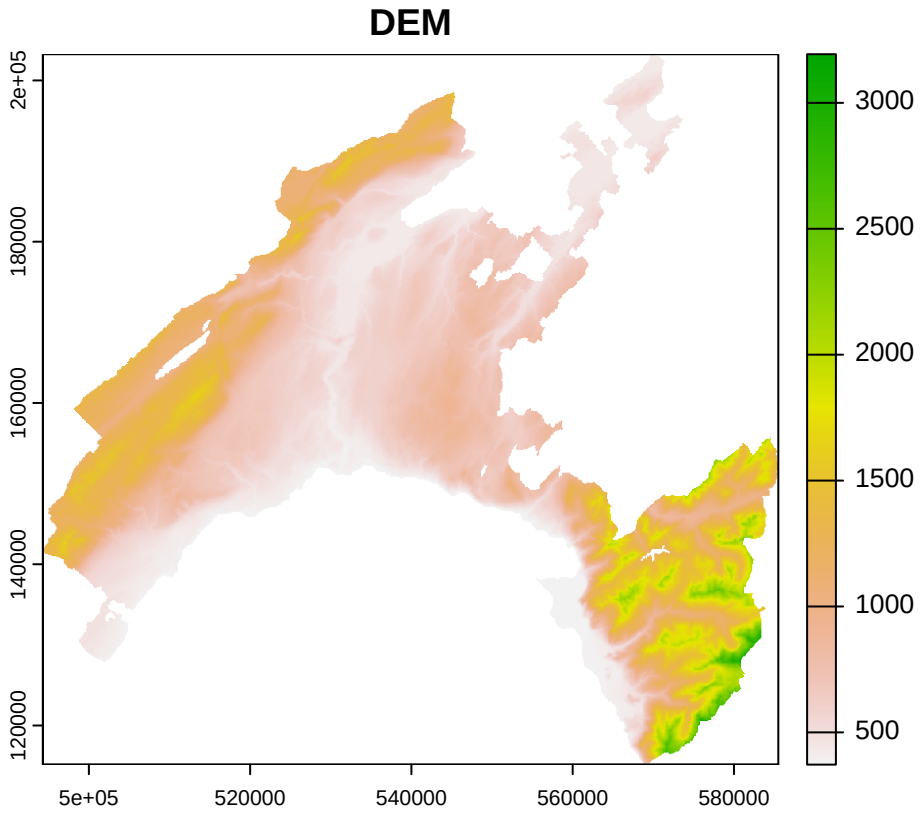
2.3.2 Plot raster features

Raster objects can be imported using the function `rast()` and exported using `writeRaster()`, specifying the format argument. As a toy example, we will work with the raster *.tif representing the digital elevation model (DEM) of Canton Vaud. Similar to the sf package for plotting vector data, terra also provides `plot()` methods for its own classes.

```
# Load the library terra
library(terra)

# Load the raster dataset
DEM_VD <- rast("data/RGIS/DEM100.tif")

# Inspect the raster
DEM_VD
```

```
## class       : SpatRaster
## dimensions  : 881, 912, 1  (nrow, ncol, nlyr)
## resolution  : 100, 100  (x, y)
## extent      : 494300, 585500, 115150, 203250  (xmin, xmax, ymin, ymax)
## coord. ref. : CH1903 / LV03 (EPSG:21781)
## source      : DEM100.tif
## name        : DEM100
## min value   :  372.0
## max value   : 3193.2
```

```r
# Plot raster dataset
plot(DEM_VD, main="DEM")
```



2.4 Geodata manipulation

In this section, we explore some basic manipulations with vector and raster geodata. This will be useful in the following chapters of this book.

2.4.1 Manipulate tabular datasets

In a vector dataset, the characteristics associated with geospatial features are stored in the attribute table. Each feature in the vector dataset is linked to a row in the table, with its characteristics organized into columns. Columns, also known as fields, store the various attributes associated with the features.

Thematic attributes stored in separate tabular dataset (such as census data, environmental monitoring stations, public health, or traffic data) can be imported and added to the attribute table of a given vector datased if the two dataset are referred to the same spatial features.

While tabular dataset can be delivered in different format (such as *.dbf, *.xlsx, *.txt), the most widely used format is ***.csv** (comma-separated values). As for other tabular format, *.csv is structured into rows and columns, where each column is separated by a comma. The first row often contains the column headers (field names), which describes the attribute in each column. The main advantages of this format compared with other formats are:

- *Simplicity*: easy to create, read, and edit.

- *Lightweight*: files are typically small and easy to transfer.
- *Compatibility*: supported by most data processing tools and software.

In most cases, data need to be reworked before beeing visualized and analyzed in R. Common tasks include: selecting subsets of rows or columns from the attribute table, rename a field, calculate new variables from the raw data values, compute summary statistics, combine data from different sources. To this end, the package **dplyr** (**?**) provides consistent set of functions that help you to solve the most common data manipulation challenges. We explore some of them in the following code chunks.

As a toy example, we will work with the the Swioss population census dataset 2020.

```
# Load ths library dply
library(dplyr)

# Load tabular dataset
Swisscensus_2020 <- read.csv("data/RGIS/census2020.csv")

# Inspect the element
str(Swisscensus_2020)
```

```
## 'data.frame':    2145 obs. of  35 variables:
##  $ ID_0                 : int  1 2 3 4 5 6 7 8 9 10 ...
##  $ ID                   : int  1 2 3 4 5 6 7 8 9 10 ...
##  $ p_infrastructure     : num  0.131 0.302 0.178 0.133 0.199 ...
##  $ p_forested           : num  0.307 0.28 0.287 0.291 0.332 ...
##  $ p_agriculture        : num  0.502 0.401 0.526 0.554 0.464 ...
##  $ p_improductible      : num  0.05815 0.01133 0.01077 0.025 0.00766 ...
##  $ surface_polygone     : num  791 1059 743 1360 653 ...
##  $ natural_growth_1000  : num  -1.001 2.115 4.65 2.119 0.528 ...
##  $ density              : num  255 1160 755 279 581 ...
##  $ dependency_ratio     : num  60.1 63.4 68.6 67.4 62.9 ...
##  $ migration_intern     : num  11.014 -6.1 -0.537 9.534 -2.905 ...
##  $ migration            : num  3.5 5.61 1.07 1.32 5.02 ...
##  $ p_employment         : num  0.105 0.415 0.109 0.14 0.364 ...
##  $ primary_sector       : num  0.19388 0.06495 0 2.12 0.00187 ...
##  $ secondary_sector     : num  0.148 1.724 0 2.8 0.045 ...
##  $ tertiary_sector      : num  1.9898 7.6414 0.3155 16.1 0.0569 ...
##  $ p_social_assistance  : num  1.31 3.19 1.13 1.25 1.27 ...
##  $ p_new_buildings      : num  1.514 0.732 0.179 0.8 0.529 ...
##  $ p_new_housings       : num  5.048 4.552 0.179 1.333 1.059 ...
##  $ p_cinema             : num  0 0.0195 0 0 0 ...
##  $ p_museum             : num  0.000993 0 0 0 0.000264 ...
##  $ p_culture_institution: num  0.01142 0.00667 0.00588 0.00816 0.00606 ...
##  $ size_households      : num  2.28 2.19 2.37 2.39 2.38 ...
##  $ p_new_entreprise     : num  0.00497 0.00407 0.00267 0.00421 0.00369 ...
##  $ p_weddings           : num  2.5 4.8 4.65 2.65 1.85 ...
##  $ p_foreigners         : num  14.4 29 17.3 15.8 17.4 ...
##  $ p_individual_houses  : num  69.2 54.7 71.6 69.1 74.7 ...
##  $ Population           : num  2014 12289 5610 3801 3795 ...
##  $ p_pop_19             : num  19.8 22.3 21.9 22.1 19.3 ...
##  $ p_pop_65             : num  18.5 18.6 14.1 21.5 20.4 ...
##  $ lat                  : num  0.571 0.562 0.566 0.582 0.562 ...
##  $ long                 : num  0.735 0.738 0.756 0.719 0.748 ...
##  $ zab_2022             : num  69.6 285.9 92.4 126.6 102.8 ...
```

```
##  $ net_income_h         : num  57603 36521 45237 45791 48875 ...
##  $ p_transport          : num  5.06 7.69 5.11 4.55 4.89 ...
```

```r
# Create a subset including only Land Use information
Swisscensus2020_LU = subset(Swisscensus_2020, select = c(2:6))

str(Swisscensus2020_LU)
```

```
## 'data.frame':    2145 obs. of  5 variables:
##  $ ID              : int  1 2 3 4 5 6 7 8 9 10 ...
##  $ p_infrastructure: num  0.131 0.302 0.178 0.133 0.199 ...
##  $ p_forested      : num  0.307 0.28 0.287 0.291 0.332 ...
##  $ p_agriculture   : num  0.502 0.401 0.526 0.554 0.464 ...
##  $ p_improductible : num  0.05815 0.01133 0.01077 0.025 0.00766 ...
```

```r
# Rename a column for better understanding
CH2020_LU <- rename(Swisscensus2020_LU, Urban=p_infrastructure)
head(CH2020_LU)
```

```
##   ID      Urban p_forested p_agriculture p_improductible
## 1  1 0.13147914  0.3072061     0.5018963     0.058154235
## 2  2 0.30217186  0.2804533     0.4013220     0.011331445
## 3  3 0.17765814  0.2866756     0.5262450     0.010767160
## 4  4 0.13308823  0.2911765     0.5536765     0.025000000
## 5  5 0.19908116  0.3323124     0.4640123     0.007656968
## 6  6 0.08712121  0.2058081     0.6830808     0.011363636
```

2.4.2 Pipes: chaining of multiple operations

Pipes allow for the chaining of multiple operations in a unique sequence, which makes the code easier to understand and reduces the need for nested function calls.

The use of pipes in R, primarily facilitated by the **magrittr** package and now natively supported in base R (R version 4.1.0 and above), is a powerful way to write clear and readable code. The function pipe is represented by the symbol **%>%**. When a pipe is placed on the right side of an object or function, the output from the function is passed as the first argument to the next function after the pipe.

Below is a simple example of using the pipe operator with the function `select()`, used to select the the fields related to the land use in the Swiss census dataset corresponding to a population density less than 100.

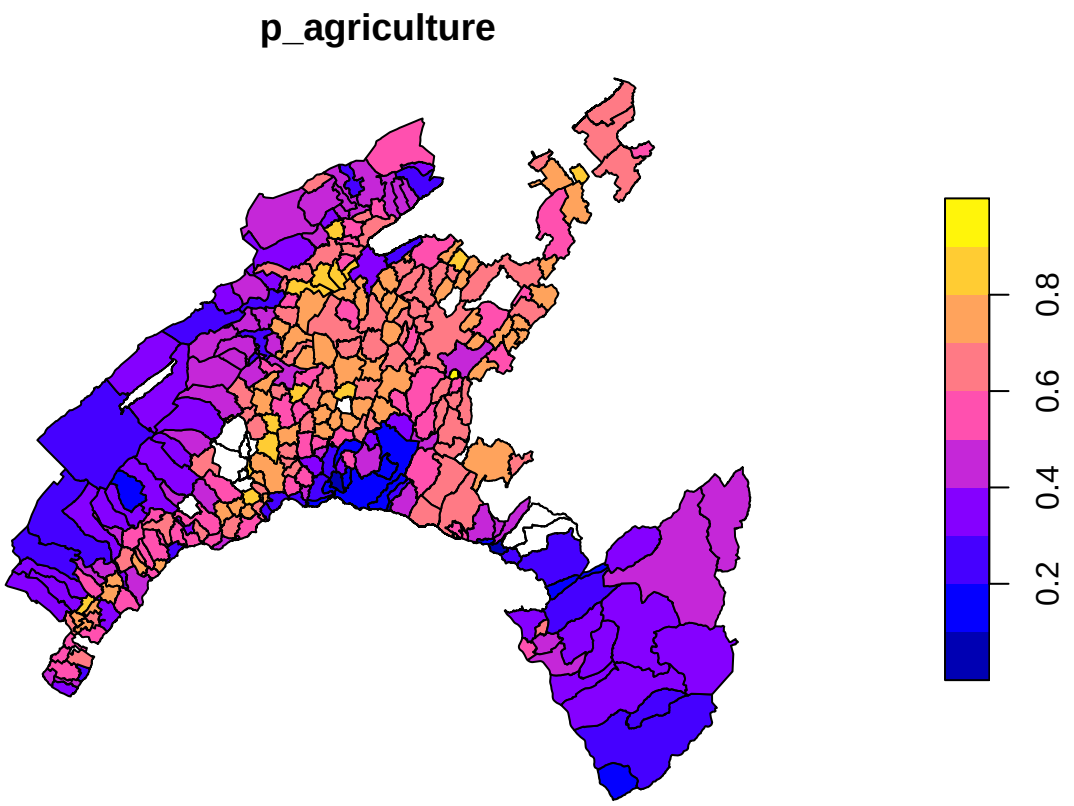
```r
LU_dens100 <-
  Swisscensus_2020 %>%
  filter(density < 100) %>%
  select(2:6)

str(LU_dens100)
```

```
## 'data.frame':    660 obs. of  5 variables:
##  $ ID              : int  23 81 114 182 211 226 309 322 325 326 ...
##  $ p_infrastructure: num  0.0514 0.0635 0.0483 0.0785 0.0727 ...
##  $ p_forested      : num  0.455 0.453 0.614 0.366 0.203 ...
##  $ p_agriculture   : num  0.479 0.487 0.32 0.521 0.683 ...
##  $ p_improductible : num  0.01712 0.00219 0.01521 0.03217 0.04156 ...
```

2.4.3 Join table

A tabular dataset can be joined to the attribute table of a vector dataset by specifying the name of the columns in the two tables used for merging. To this end, we can use the function `merge()` included in the package sp .

```
Vaud_census_2020 <- merge(x=vaud, y=Swisscensus_2020,
                          by.x="Code",
                          by.y="ID",
                          all.x=TRUE)

# Plot based on a joined attribute
plot(Vaud_census_2020["p_agriculture"])
```



2.4.4 Mapping with ggplot2

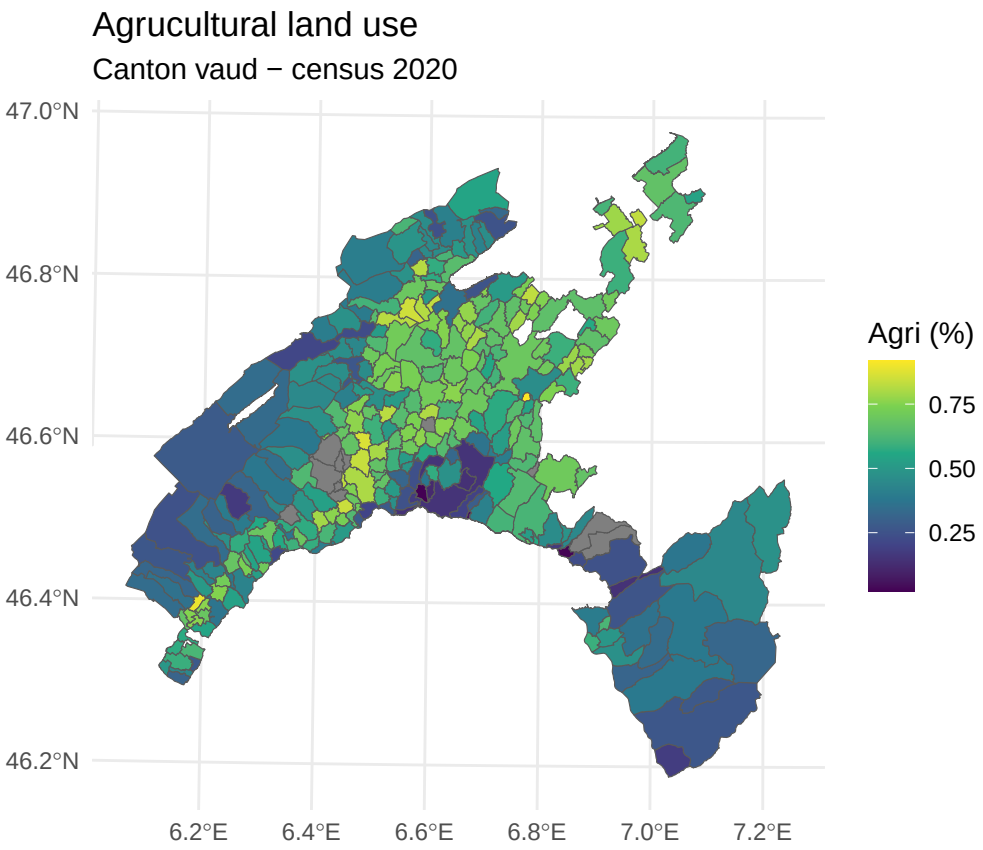
Mapping in R can be efficiently achieved using the powerful visualization package **ggplot2**, especially when combined with additional packages like sf. These tools togheter enable users to create detailed and customized maps for spatial data analysis and visualization. In particular sf facilitates the handling of complex spatial data structures, making it possible to create intricate and informative maps.

In the following example we crate an aesthetic map of Canton Vaud based on the percentage of agricultural land use by municipality. Aesthetic mappings describe how the attributes of the geodata are mapped to the visual properties (aesthetics) of the plot. The command aesthetics – `aes()` – control the appearance of the plot elements, such as points, lines, bars, and so on.

```r
# load the library gglpt2
library (ggplot2)

# Install viridis color scale as a package
install.packages("viridis", repos="http://cran.us.r-project.org")
```

```
## package 'viridis' successfully unpacked and MD5 sums checked
##
## The downloaded binary packages are in
##  C:\Users\mtonini1\AppData\Local\Temp\RtmpiyDQ83\downloaded_packages
```

```r
library(viridis)

# Use ggplot2 for mapping
ggplot(Vaud_census_2020) +
  geom_sf(aes(fill = p_agriculture)) +
  scale_fill_viridis_c(option = "viridis", name = "Agri (%)") +
  theme_minimal() +
  labs(title = "Agrucultural land use",
       subtitle = "Canton vaud - census 2020")
```



2.4.5 Cropping ratser

Many projects in geosciences require integrating raster data covering an exension larger than the study area. In these scenarios, raster cropping and masking are essential for standardizing the spatial extent of the input

data. These operations help to minimize memory usage and computational resources needed for subsequent analysis, and often represnet a crucial pre-processing step before generating detailed and visually appealing maps incorporating raster data.

The first check you need to perform when importing data from different sources is to verify whether the **coordinate reference systems (CRS)** of the input geodatasets are consistent. To this end, simply type the names of the geodatasets and check the "coord. ref." attribute. For a detailed description of the coordinate reference system, use the `crs()` function. If necessary, re-project one of the spatial layers using the `project()` function, specifying the CRS of the other dataset to ensure their extents perfectly overlap.

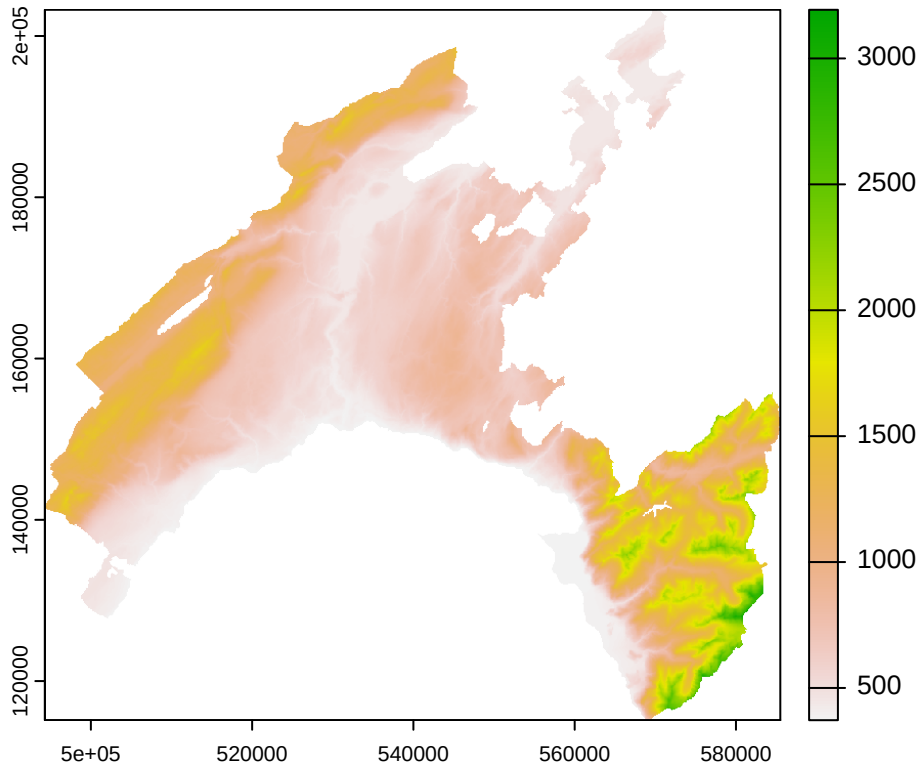
```
# Verify CRS
DEM_VD
```

```
## class       : SpatRaster
## dimensions  : 881, 912, 1  (nrow, ncol, nlyr)
## resolution  : 100, 100  (x, y)
## extent      : 494300, 585500, 115150, 203250  (xmin, xmax, ymin, ymax)
## coord. ref. : CH1903 / LV03 (EPSG:21781)
## source      : DEM100.tif
## name        : DEM100
## min value   :  372.0
## max value   : 3193.2
```

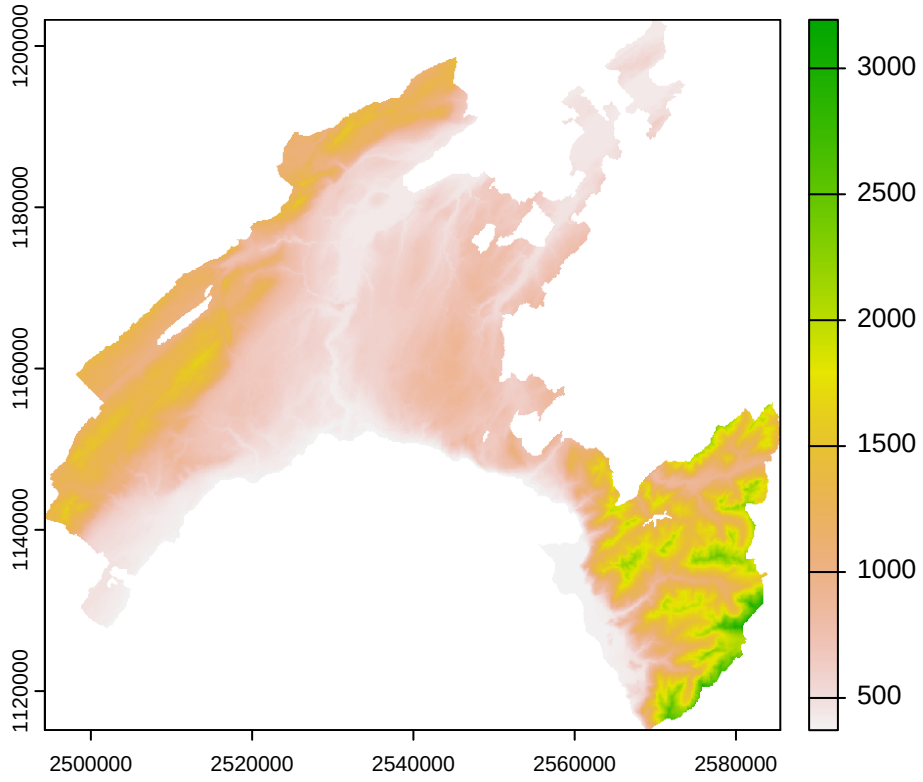
```
vaud
```

```
## Simple feature collection with 315 features and 4 fields
## Geometry type: POLYGON
## Dimension:     XY
## Bounding box:  xmin: 2494306 ymin: 1115149 xmax: 2585462 ymax: 1203493
## Projected CRS: CH1903+ / LV95
## First 10 features:
##    Shape_Leng Shape_Area         Munics Code                       geometry
## 1    30546.68   24178457        Payerne 5822 POLYGON ((2559434 1183900, ...
## 2    21404.88   23354685     Rossinière 5842 POLYGON ((2568027 1143632, ...
## 3    27146.69   23190781       Vallorbe 5764 POLYGON ((2519472 1176171, ...
## 4    31110.58   22858943        Puidoux 5607 POLYGON ((2551941 1152936, ...
## 5    28268.14   22501915        Baulmes 5745 POLYGON ((2524665 1180813, ...
## 6    26122.21   22002176     Corbeyrier 5404 POLYGON ((2565278 1132847, ...
## 7    33182.64   20915807 Vully-les-Lacs 5464 POLYGON ((2565363 1195281, ...
## 8    29224.24   20772714        Bassins 5703 POLYGON ((2508540 1146203, ...
## 9    20254.18   19791177  Mont-la-Ville 5491 POLYGON ((2519784 1169416, ...
## 10   30004.81   19487486       Avenches 5451 POLYGON ((2570746 1194939, ...
```

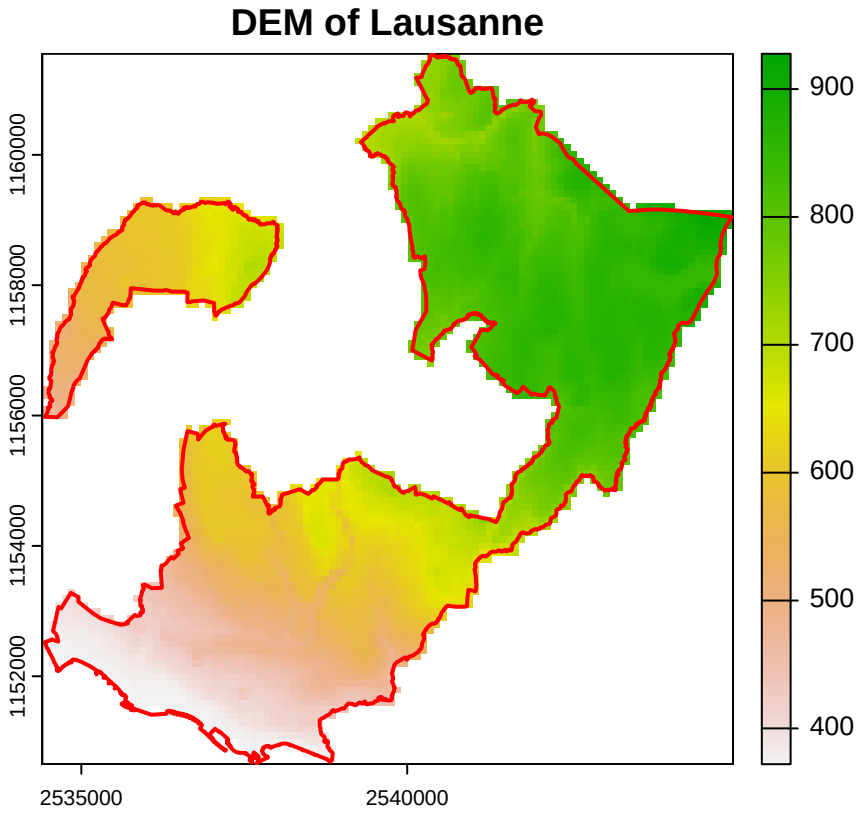
```
# Project the DEM data using the CRS of the shapefile
demVD_prj = project(DEM_VD, crs(vaud))

# Verify again
plot(DEM_VD)
```




```
plot(demVD_prj)
```




```
# Extract the municipality of Lausanne
Lausanne <- filter(vaud, Munics == "Lausanne")

# Crop the DEM to the extent of Lausanne (bounding box)
DEM_Lausanne_cropped <- crop(demVD_prj, Lausanne)

# Mask the cropped DEM using the Lausanne polygon to get the exact shape
DEM_Lausanne <- mask(DEM_Lausanne_cropped, Lausanne)

# Plot the DEM
plot(DEM_Lausanne, main = "DEM of Lausanne")

# Add the Lausanne polygon outline
plot(st_geometry(Lausanne), add = TRUE, border = "red", lwd = 2)
```



2.4.6 Display the categorical variables of a raster

Categorical variables, also known as qualitative variables, are variables that represent distinct groups or categories. These variables are typically non-numeric and can be divided into a finite number of categories or levels. Categorical variables are often used to represent characteristics or attributes that do not have a natural ordering.

Land use classes of a raster dataset are a common example of categorical variables. Each class represents a distinct type of land use labeled with names or, mostly, numeric codes.

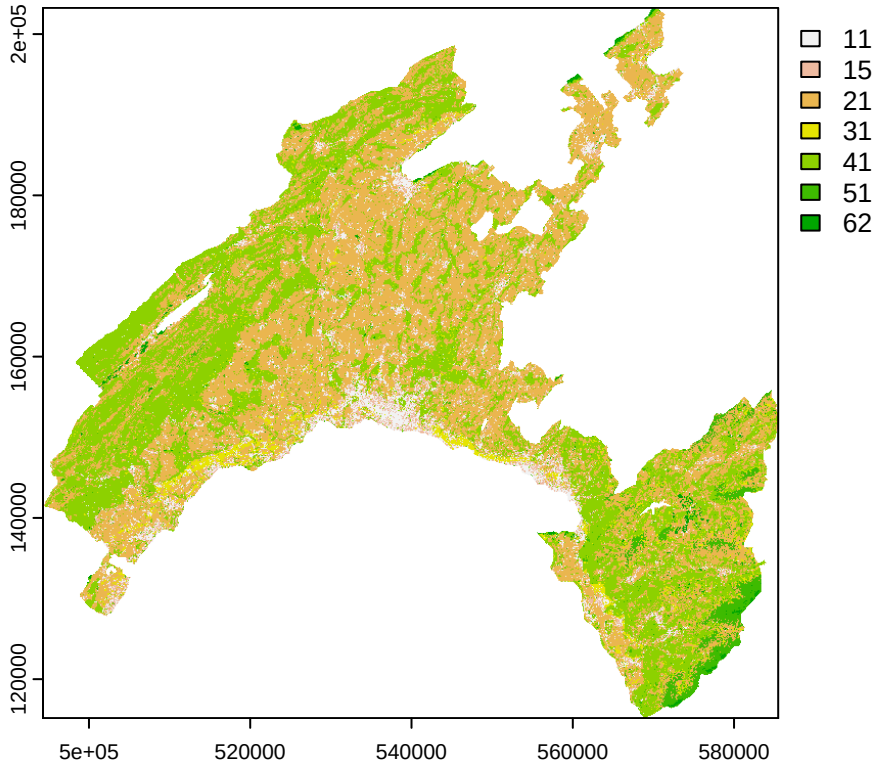
To understand the characteristics of the categorical variables, you can plot the "Land Cover" raster dataset by using its original classes. To visualize the data you need to perform few data manipulations before.

```
# Load the raster data
landCover <- rast("data/RGIS/landCover.tif")

# Inspect the raster
landCover
```

```
## class       : SpatRaster
## dimensions  : 3524, 3647, 1  (nrow, ncol, nlyr)
## resolution  : 25, 25  (x, y)
## extent      : 494300, 585475, 115150, 203250  (xmin, xmax, ymin, ymax)
## coord. ref. : CH1903 / LV03 (EPSG:21781)
## source      : landCover.tif
## name        : landCover
```

```
## min value   :        11
## max value   :        62
```

```
# Plot the raster (using codes)
plot (landCover)
```




```
# Convert the raster as a factor to represent categorical data
landCover <- as.factor(rast("data/RGIS/landCover.tif"))

# Inspect the current levels (classes) of the raster
current_levels <- levels(landCover)[[1]]
print(current_levels)
```

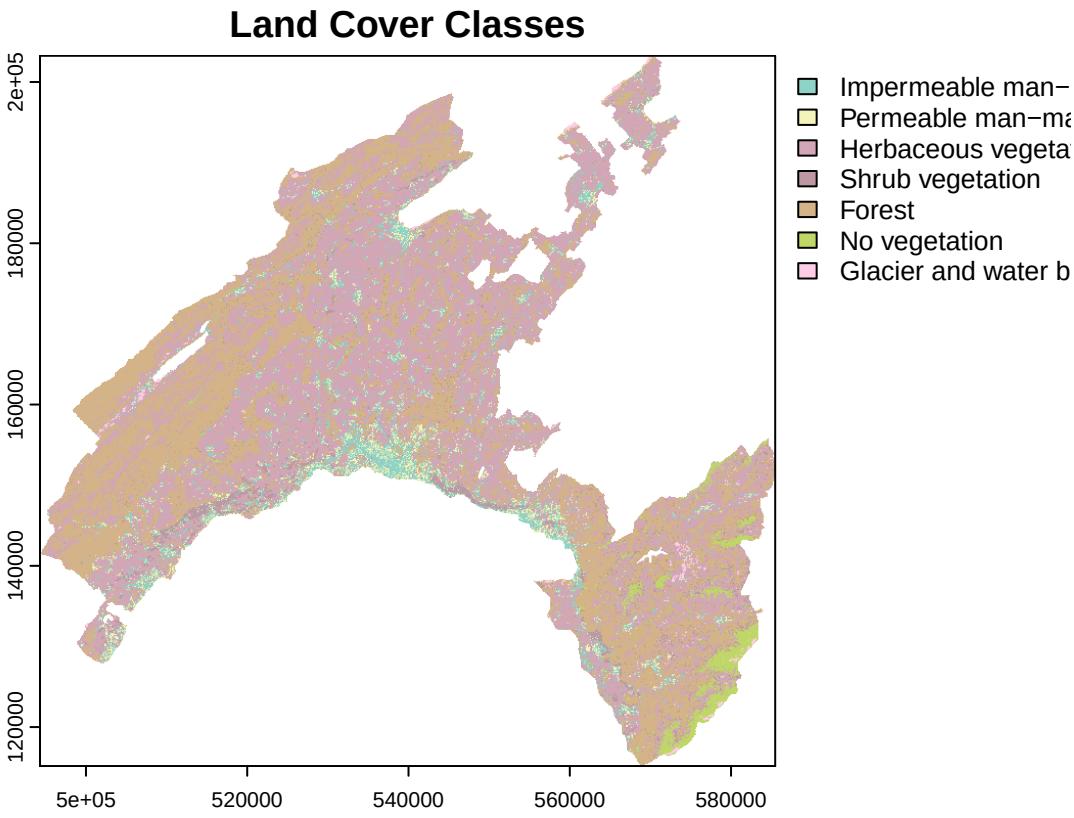
```
##   ID landCover
## 1 11        11
## 2 15        15
## 3 21        21
## 4 31        31
## 5 41        41
## 6 51        51
## 7 62        62
```

```
# Define a vector of new class names (descriptions)
new_class_names <- c(
```

```
 "11" = "Impermeable man-made",
 "15" = "Permeable man-made",
 "21" = "Herbaceous vegetation",
 "31" = "Shrub vegetation",
 "41" = "Forest",
 "51" = "No vegetation",
 "62" = "Glacier and water body")

# Ensure that new class names match the existing levels
current_levels$landCover <- new_class_names[as.character(current_levels$ID)]

# Apply the new class names to the levels of the raster
levels(landCover) <- current_levels

# Verify the new levels
print(levels(landCover))
```

```
## [[1]]
##   ID              landCover
## 1 11   Impermeable man-made
## 2 15     Permeable man-made
## 3 21  Herbaceous vegetation
## 4 31       Shrub vegetation
## 5 41                 Forest
## 6 51          No vegetation
## 7 62 Glacier and water body
```

```
# Create a color palette with enough colors for all classes

library(RColorBrewer)

num_classes <- length(new_class_names)
color_palette <- colorRampPalette(brewer.pal(8, "Set3"))(num_classes)

# Plot the land cover data with the custom color palette
plot(landCover, col = color_palette, main = "Land Cover Classes")
```



2.5 Further Reading

This chapter aims to provide an overview of the main functionalities integrated into the R packages **sf** and **terra** for vector and raster data manipulation. For more in-depth knowledge and understanding, you can refer to the following documentation:

- `sf` Simple Features for R
- `terra` R package for spatial data analysis
- `dplyr` Solves most common data manipulation challenges

3 Geographically Weighted Summary Statistics for Geosciences

Geographically Weighted Summary Statistics (GWSS) represent an advanced analytical approach in geosciences, allowing researchers to explore spatial variations in data across a geographical landscape. Unlike traditional summary statistics that provide a single, overall summary measure – like mean, median, or standard deviation – for the entire dataset (i.e., globally), GWSS techniques calculate these measures locally, reflecting the unique characteristics and variations at different spatial locations. This method is particularly valuable in geosciences, where spatial heterogeneity often plays a critical role.

3.1 GWSS for natural hazards assessment

In the assessment of natural hazards – like forest fires, earthquakes, flooding, or landslides – identifying hotspots is a first step in prioritizing areas for risk management and mitigation strategies. This process helps in understanding which regions are most vulnerable, allowing for the allocation of resources, implementation of protective measures, and development of emergency response plans to reduce the potential impact on communities and infrastructure.

Although spatio-temporal inventories of natural hazards are made available, extracting meaningful insights about their distribution patterns remains challenging when relying solely on the examination of mapped event locations. To address this, Geographically Weighted Summary Statistics (GWSS) can be utilized. GWSS assumes that the spatial distribution of the single occurrences within a certain area exhibits geographic trends, and calculates local statistics to reveal these patterns more clearly. This approach provides valuable insights for effective management and prevention strategies.

As case study, in the present computing lab we compute the GW local means, the GW local standard deviation and the GW localized skewness of wildfires – made available as mapped burned areas dataset – in continental Portugal, registered in the period 1990-2013. This application is inspired by the work of (**?**)

3.1.1 The overall methodology

Summary statistics include a number of measures that can be used to summarize a set of observations, the most important of which are measures of central tendency (arithmetic mean, median, mode) and measures of dispersion around the mean (variance, standard deviation). In addition, measures of skewness and kurtosis are descriptors of the shape of the probability distribution function, the former indicating the asymmetry and the latter the peakedness/tailedness of the curve.

For geoenvironmental processes, these global statistical descriptors may vary from one region to another, as their values may be affected by local environmental and socio-economic factors. In this case, an appropriately localized calibration can provide a better description of the observed values. One way to achieve this goal is to weight the above statistical measures for a given quantitative variable based on their geographical location.

For the comoputation we introduce here the method proposed by (**?**) and implemented in the function `GWSS` presented in the R package GWmodel (**??**).

The evaluation of geographically weighted summary statistics is obtained by computing a summary for a small area around each geolocalized punctual observation, by using the Kernel Density Estimation technique (KDE) (**?**). KDE is estimated at each point, taking into account the influence of the points falling within an area, with increasing weight towards the center, corresponding to the point location. A surface summary statistic is thus obtained.

3.1.2 Forest fires dataset

Forest fires inventories indicating the location, the starting date and other related variables, such as the cause of ignition and the size of the area burned, are broadly available with a different degree of accuracy in different countries.

In the present study, we consider the Portuguese National Mapping Burnt Areas dataset , freely available from the website of the Institute for the Conservation of Nature and Forests. This is a long spatio-temporal dataset resulting from the processing of satellite images acquired once a year at the end of the summer season. Row data consists of records of observed fire scars. The burned areas were estimated by using image classification techniques, then compared with ground data to resolve the discrepancies. Polygons have been converted into a point vector dataset, where each point represent the centroid of the burned areas, while the size of the burned areas and the starting date of the fires events are given as attributes.

In this work, for consistency reasons, we consider only fires occurred between 1990 and 2013, and with a burned area above 5 hectares.

3.2 Computing lab: GWSS

3.2.1 Load the libraries

First you have to load the following libraries:

- *splancs*: for display and analysis of spatial point pattern data
- *GWmodel*: techniques from a particular branch of spatial statistics, termed geographically-weighted (GW) models
- *sf*: support for simple features, a standardized way to encode spatial vector data
- *ggplot2*: a system for 'declaratively' creating graphics
- *sp*: classes and methods for spatial data

```
library(splancs)
library(GWmodel)
library(sf)
library(ggplot2)
library(sp)

(.packages())
```

3.2.2 Import the forest fire dataset

In this section you will load the geodata representing the forest fires inventory for events occurred in the continental Portuguese area in the period 1990-2013. You will also load the boundaries of the study area. You will start by exploring the datasets using mainly visual tools (plotting and histogram).

```
# Import Portugal boundary
Portugal <- st_read("data/GWSS/Area_Portugal.shp")

# Import the Portuguese forest fires dataset for the entire Portuguese area.
FFPorto<-st_read("data/GWSS/FF_Portugal.shp")
```

You can explore the dataset by using different tools for **exploratory data analyses**. You will start by visualizing the databases (i.e., the attribute table). Than you can plot the histogram of events distribution based on the variable *"Area_ha"* (i.e., the size in hectares of the burned area). Since this is a power low distribution, for a better understanding it is recommended to transform the data using a logarithmic scale. Computiong *log10* you can easily evaluate the frequency distribution of the burned areas.

```
# Show the attribute table (first 10 rows)
FFPorto
```

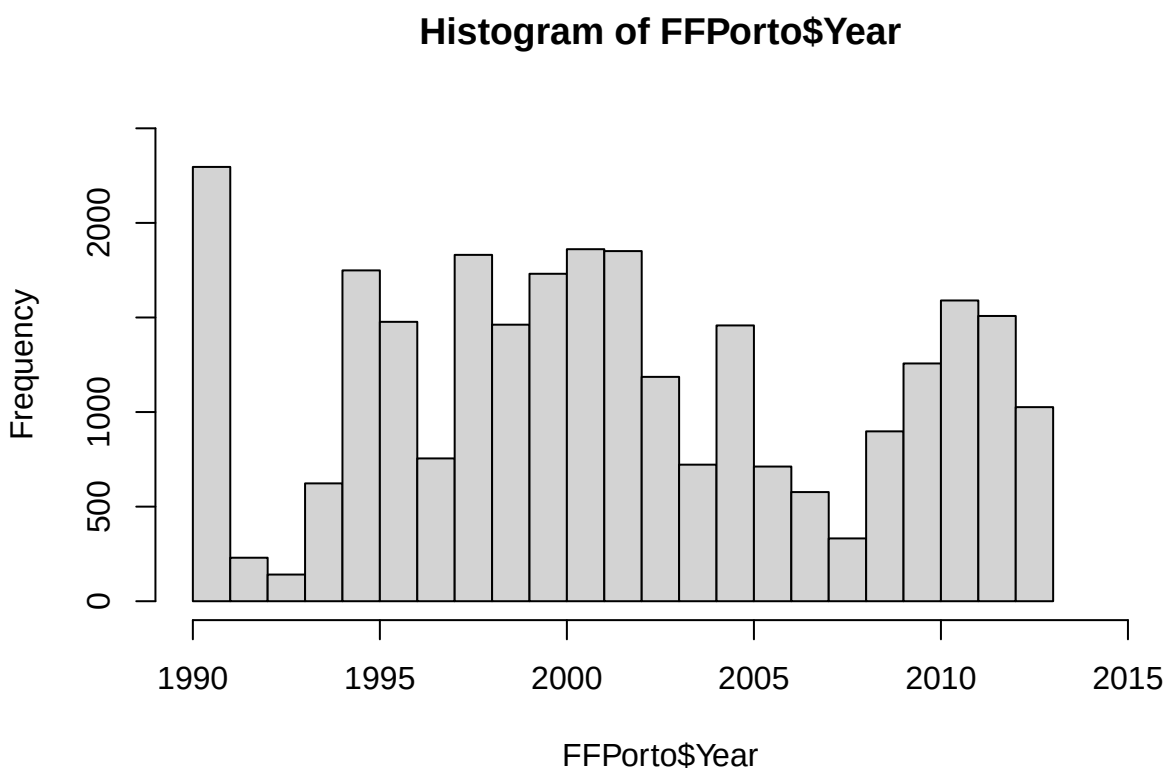
```
## Simple feature collection with 27273 features and 4 fields
## Geometry type: POINT
## Dimension:     XY
## Bounding box:  xmin: 82126.92 ymin: 7083.529 xmax: 361492.2 ymax: 573192.5
## Projected CRS: Lisboa_Hayford_Gauss_IGeoE
## First 10 features:
```

```
##    Year Area_ha        X        Y                  geometry
## 1  1990   5.000 275224.0 525501.4   POINT (275224 525501.4)
## 2  1990   5.000 235278.0 452412.9   POINT (235278 452412.9)
## 3  1990   5.000 200628.8 491568.2 POINT (200628.8 491568.2)
## 4  1990   5.000 120687.8 228816.9 POINT (120687.8 228816.9)
## 5  1990   5.031 207183.9 498247.7 POINT (207183.9 498247.7)
## 6  1990   5.031 162730.0 488388.8   POINT (162730 488388.8)
## 7  1990   5.031 250545.2 532563.7 POINT (250545.2 532563.7)
## 8  1990   5.031 268013.3 356110.4 POINT (268013.3 356110.4)
## 9  1990   5.033 222095.7 543059.9 POINT (222095.7 543059.9)
## 10 1990   5.044 207265.7 377657.8 POINT (207265.7 377657.8)
```

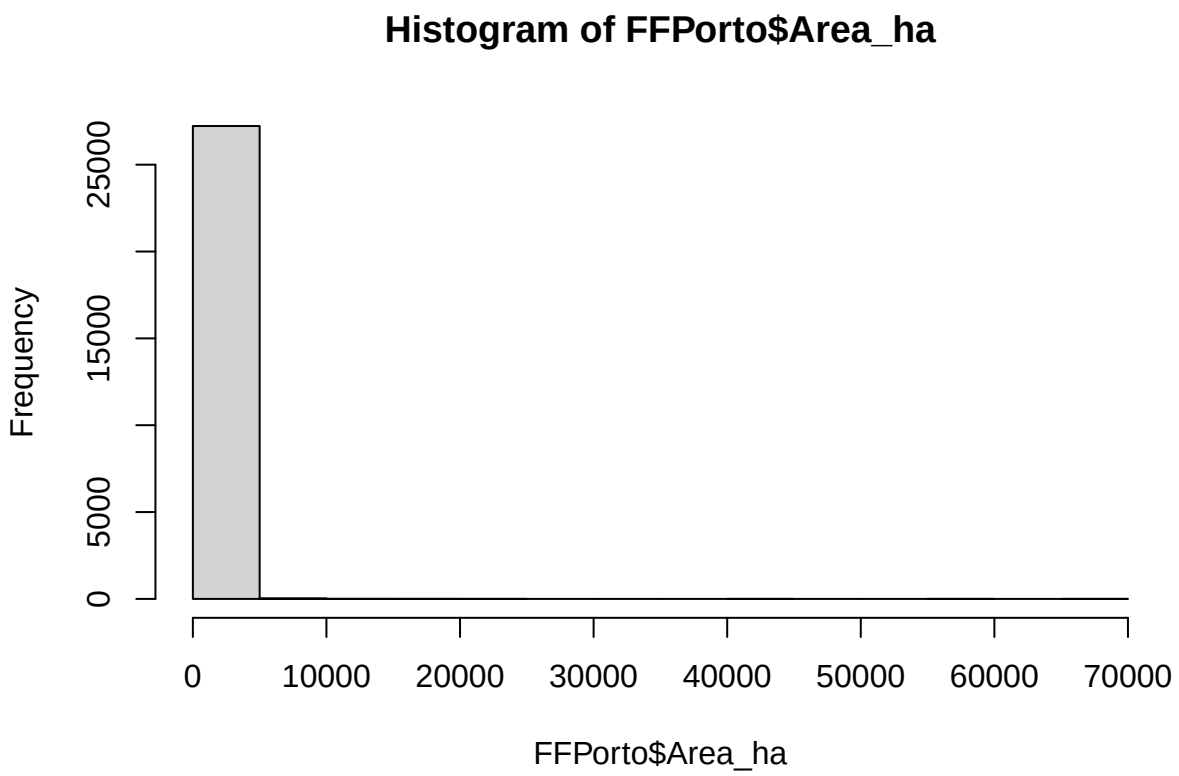
```r
# Open the attribute table in a new tab
View(FFPorto)

# Summary statistics of all of the attributes associated with this dataset
summary(FFPorto$Area_ha)
```

```
##     Min.  1st Qu.   Median     Mean  3rd Qu.     Max.
##     5.00     9.78    20.41   107.14    56.07 66070.63
```

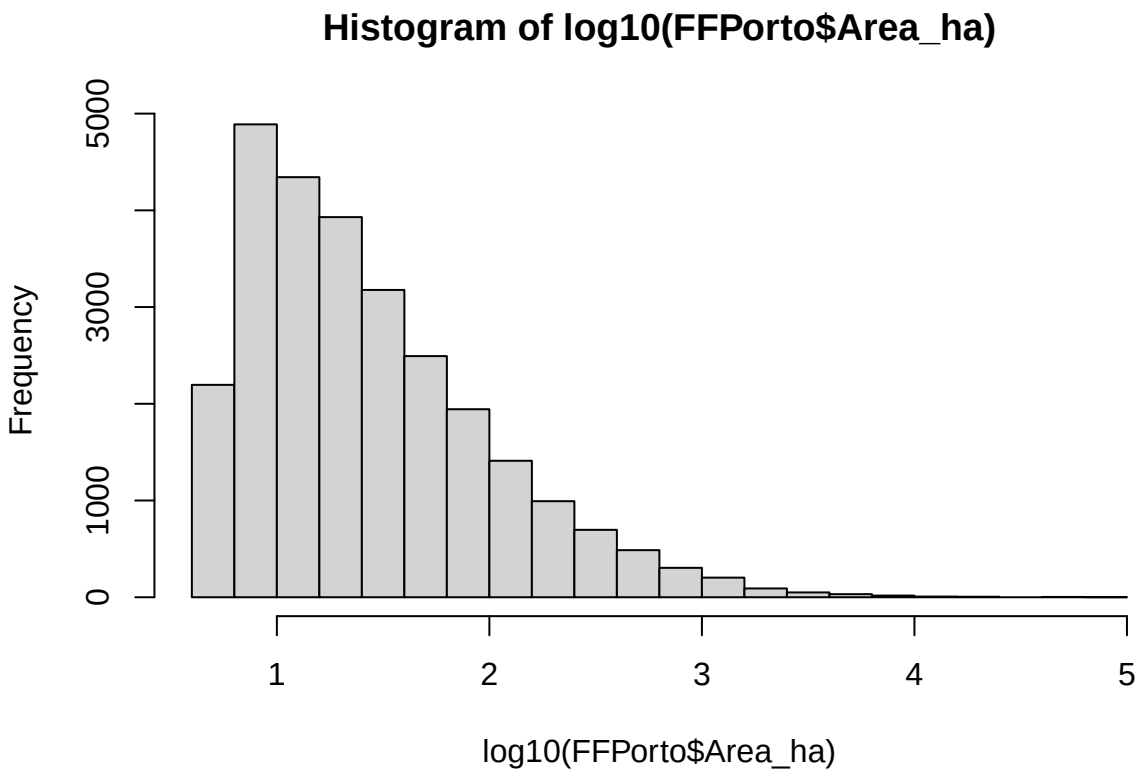
```r
# Histogram of the forest fires distribution by year
hist(FFPorto$Year, breaks = unique(FFPorto$Year), xlim=c(1990,2015), ylim=c(0,2500))
```



```r
# Histogram of the forest fires distribution by burned area


hist(FFPorto$Area_ha) #power low disrtibution
```




```r
hist(log10(FFPorto$Area_ha)) #log transformation
```

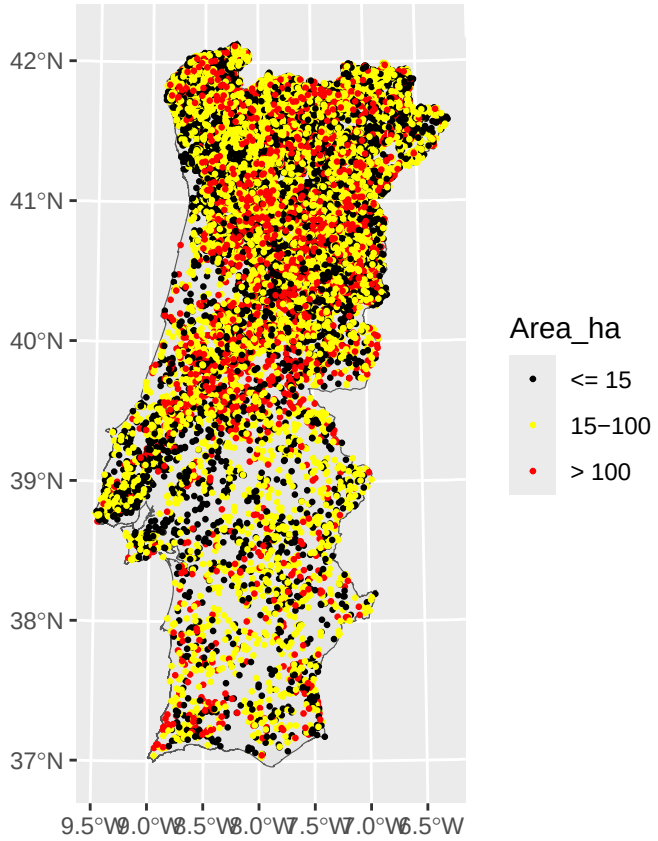


3.2.3 Forest fires spatial distribution

For a better understanding of the phenomenon, you can group the events according to the size of the burned areas. Base on a mix of empirical experience and the frequency distribution of forest fires presented in our dataset, by area, the following three classes can be defined:

- *Small fires:* less than 15 ha
- *Medium fires:* between 15 ha and 100 ha
- *Large fires:* bigger than 100 ha

Plotting the forest fire events using different colors, based on the size of the burned areas, can simplify the understanding of their pattern distribution, knowing that fires of different size have normally different drivers.



3.2.4 Compute the GWSS

From the exploratory data analysis performed above, it seems that a simple plotting of the forest fires events based on their spatial distribution, even if classified based on their size, can not really help to understand their spatial behaviors. This is because we face to a huge number of events and the variable that we are using to characterize them – the size of the burned area, in our case – is very heterogeneous. To this aim, we can compute basic and robust GWSS and plot the data accordingly.

The GWSS includes *geographically weighted means*, *standard deviations* and the *skweness*. As you can see from the R Documentation – command: `help(gwss)` – same data manipulations are necessary to transform the forest fires dataset into a compatible data frame format.

```
# Transform inputs data into a spatial points data frame
FFdf<-data.frame(X=FFPorto$X, Y=FFPorto$Y, Area=FFPorto$Area_ha)
FFspdf<-SpatialPointsDataFrame(FFdf[,1:2], FFdf)
str(FFspdf)
```

GWSS parameters:

- we summarize the data based on the size of the burned area (`vars`)
- we use here an adaptive kernel where the bandwidth (`bw`) corresponds to the number (100 in this case) of nearest neighbors (i.e. `adaptive` distance)
- we keep the default values for the other parameters

```
# Run gwss: this operation can take several minutes...be patient!

# While waiting, you can look at gwss R Documentation:
help(gwss)

FFgwss <- gwss(FFspdf,vars=("Area"),adaptive=TRUE, bw=100)
```

3.2.5 Look at the results

The resulting object (`FFgwss`) has a number of components. The most important one is the spatial data frame containing the results of local summary statistics for each data point location, stored in `FFgwss$SDF` (that is a spatial data frame).

```
# Inspect the output
FFgwss
```

```
##    ********************************************************************
##    *                        Package   GWmodel                          *
##    ********************************************************************
##
##    ***********************Calibration information************************
##
##    Local summary statistics calculated for variables:
##     Area
##    Number of summary points: 27273
##    Kernel function: bisquare
##    Summary points: the same locations as observations are used.
##    Adaptive bandwidth: 100 (number of nearest neighbours)
##    Distance metric: Euclidean distance metric is used.
##
##    ************************Local Summary Statistics:***********************
##    Summary information for Local means:
## Area_LM
##       Min.    1st Qu.     Median    3rd Qu.       Max.
##   10.97782   42.54450   70.89374  120.67898 2300.80424
##    Summary information for local standard deviation :
## Area_LSD
##         Min.      1st Qu.       Median      3rd Qu.         Max.
##     6.856477    74.310044   149.714657   292.508111 11280.136286
##    Summary information for local variance :
## Area_LVar
##         Min.      1st Qu.       Median      3rd Qu.         Max.
## 4.701127e+01 5.521983e+03 2.241448e+04 8.556100e+04 1.272415e+08
##    Summary information for Local skewness:
## Area_LSKe
##       Min.    1st Qu.     Median    3rd Qu.       Max.
##   0.829069   3.178626   4.373189   6.076042 192.906106
##    Summary information for localized coefficient of variation:
## Area_LCV
##       Min.    1st Qu.     Median    3rd Qu.       Max.
##  0.6245756  1.5756289  2.0240543  2.6604986 16.0254231
##
##    ********************************************************************
```

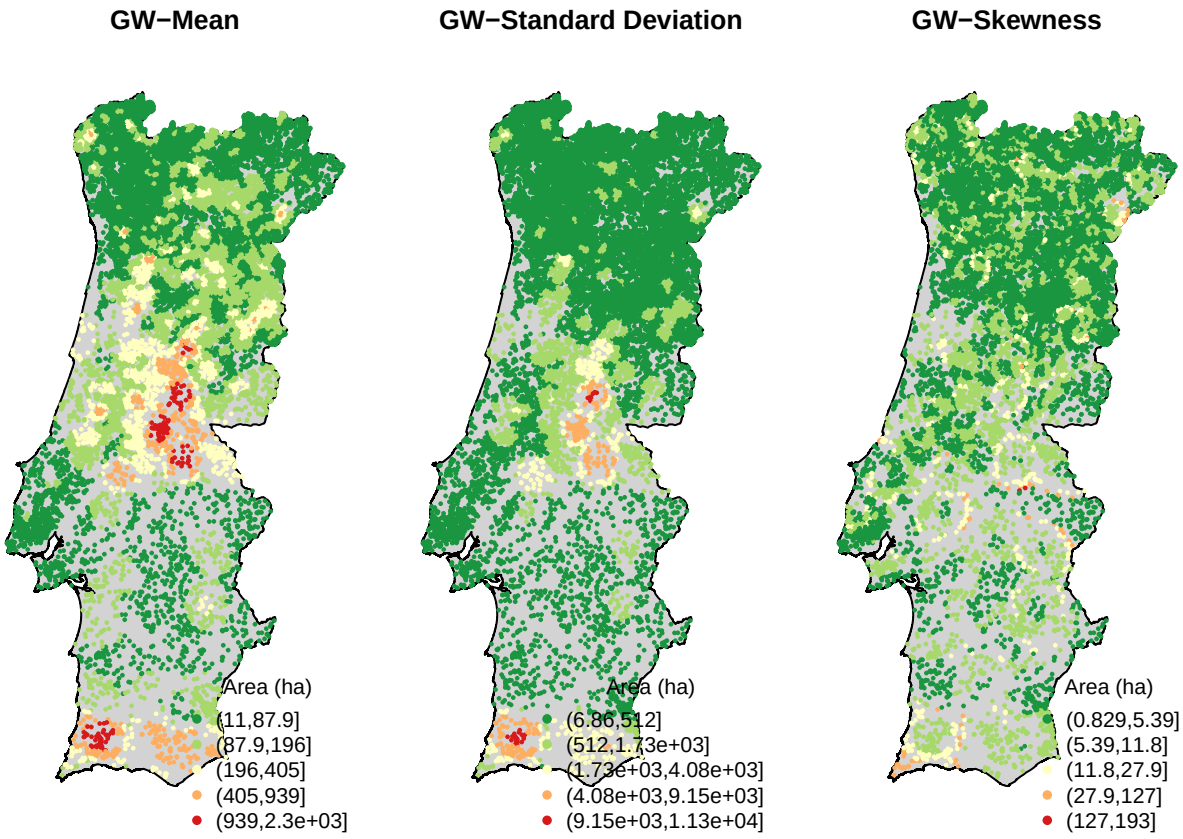
```
# Display the first 6 rows
head(FFgwss$SDF)
```

```
##            coordinates  Area_LM  Area_LSD  Area_LVar Area_LSKe  Area_LCV
## 1   (275224, 525501.4) 56.46496 127.16616 16171.2316  4.612383 2.2521253
## 2   (235278, 452412.9) 89.17704 196.51574 38618.4344  4.384154 2.2036585
## 3 (200628.8, 491568.2) 25.64396  25.02941   626.4712  2.851515 0.9760351
## 4 (120687.8, 228816.9) 16.37948  17.85761   318.8941  3.435847 1.0902427
## 5 (207183.9, 498247.7) 46.54590  84.45980  7133.4579  4.465928 1.8145489
## 6   (162730, 488388.8) 28.54313  35.14293  1235.0258  2.400911 1.2312221
```

3.2.6 GWSS maps

To produce a map of the local geographically weighted summary statistic of your choice, firstly we need to enter a small R function definition. This is just a short R program to draw a map: you can think of it as a command that tells R how to draw a map (from (**?**)). The advantage of defining a function is that the entire map can now be drawn using a single command for each variable, rather than having to repeat those steps each time. To define the intervals for the classification, we use Jenks natural breaks classification method (`style="fisher"`).

```
library(RColorBrewer) #a useful tool for designing map color palettes.
library(classInt) #to define class intervals

# The function definition to draw the map:

quick.map <- function(spdf,var,legend.title,main.title) {
  x <- spdf@data[,var]
  int <- classIntervals(x, n=5, style="fisher")
  cut.vals<-int$brks
  x.cut <- cut(x,cut.vals)
  cut.levels <- levels(x.cut)
  cut.band <- match(x.cut,cut.levels)
  colors <- rev(brewer.pal(length(cut.levels), "RdYlGn"))
  par(mar=c(1,1,1,1))
  plot(Portugal$geometry,col='lightgrey')
  title(main.title)
  plot(spdf,add=TRUE,col=colors[cut.band],pch=16, cex=0.5)
  legend("bottomright",cut.levels,col=colors,pch=16,bty="n",title=legend.title)
  }

# Call the function to display the maps of the Local Mean (LM), Local Standard Deviation(LS), and Local

par(mfrow=c(1,3))
quick.map(FFgwss$SDF, "Area_LM", "Area (ha)", "GW-Mean")
quick.map(FFgwss$SDF, "Area_LSD", "Area (ha)", "GW-Standard Deviation")
quick.map(FFgwss$SDF, "Area_LSKe", "Area (ha)", "GW-Skewness")
```



3.3 Conclusions and further analyses

This practical computer lab allowed you to familiarize with GWSS, by the proposed application about geographically weighted summary statistics. This approach allowed us to explore how the average burned area vary locally through Continental Portugal in the period 1990-2013.

The global Geographically Weighted (GW) mean informs us about the local average value of the burned area, based of the neighboring events occurred in a given period. Similarly, the GW standard deviation allows to inspect the extent to which the size of the burned area spread around this mean. Finally, the GW skewness measures the symmetry of the distribution: a positively skewed distribution means that there is a high number of observations with mean value lower that the median; and the contrary for a negatively skewed distribution.

To ensure that everything is perfectly clear, we propose you to answer the following questions. You can find the correct responses in the reference paper (**?**).

1) Describe the pattern distribution of the GW-Mean for burned area in Portugal during the investigated periods.

2) Does the GW-Standard Deviation follows the same pattern? How can you interpret these two pattern distributions in terms of burned area and their characterization?

3) The GW-Skewness has positive values everywhere: what does it means? What do these values suggest to be the distribution of the burned areas, in terms of their size, around the local mean?

4) Which can be other applications of GWSS in geosciences data? In other words, can you imagine other geo-environmental dataset that can be analysed using GWSS?

5) Finally, you can play with the code and try to run it using a different numbers of nearest neighbors (`bw=x`) and compare the results.

4 Spatio-Temporal Cluster Analysis of GeoEnvironmental Processes

Spatio-temporal cluster analysis (ST clustering) is a powerful tool used to identify patterns and relationships in data that vary across both space and time. In the context of geoenvironmental processes, this type of analysis can helps in understanding how environmental phenomena — such as wildfires, landslides, or flood events — are distributed geographically and how they evolve over time. By detecting clusters, researchers can uncover hotspots, assess trends, and identify potential triggers or influencing factors, leading to more informed decision-making and targeted interventions in environmental management.

4.1 ST clustering for fire management

The configuration of forest fires across space and time presents a complex pattern which significantly affects the forest environment and adjacent human developments. Statistical techniques designed for spatio-temporal random point processes can be utilized to identify a structure, recognize hot-spots and vulnerable areas, and address policy makers to prevention and forecasting measures.

In this practical computing lab we consider the same case study as in the "Geographically Weighted Summary Statistics" lab. The main objective is to reveal if space and time act independently or whether neighboring events are also closer in time, interacting to generate ST clusters. The attribute that we will consider to achieve this goal is the starting date of fires events. To account for the different geographical distribution of fires in Portugal, events occurred in the Norther and Southern areas will be modeled separately.

For more details about the input dataset, please refer to the GWSS lab documentation.

4.2 Clustering methods

To detect spatio-temporal clusters of forest fires, we will use the following statistical methods:

(1) The **Ripley's K- function** to test the space-time interaction and the spatial attraction/independency between fires of different size.

(2) The **Kernel Density Estimator** allowing elaborating smoothed density surfaces representing hotspots.

We provide below a short description for both these methods. More details can be found in (**?**), the paper which inspired the present lab.

4.2.1 Ripley's K-function

The Ripley's K-function allows inferring about the spatial randomness of mapped punctual events (**?**). It is largely applied in geosciences to analyse the pattern distribution of a spatial point process. The original spatial univariate K-function $K(s)$ is defined as the ratio between the expected number E of point events falling at a certain distance r from an arbitrary event and the intensity λ of the spatial point process, this last corresponding to the average number of points per unit area.

Under complete spatial randomness, which assumes the independence among the events, $K(s)$ is equal to the area of the circle around the target event at any distance's value. It follows that events are spatially

clustered within the range of distances at which $K(s)$ assumes vales higher than this area, while they are spatially dispersed for lower values. The temporal K-function $K(t)$ is defined in the same way as for the spatial case, with the time-based intensity and the time length replacing the spatial distance.

4.2.1.1 Spatio-temporal interaction The space-time K-function $K(s,t)$ can be considered as a bivariate function where space and time represent the two variables of the equation. It is defined as the number of further events occurring within a distance r and time t from an arbitrary event.

If there is no space-time interaction, $K(s,t)$ is equal to the product of the purely spatial and purely temporal K-function. Inversely, if space and time interact generating clusters, the difference $D(s,t)$ between these two values is positive, where:

$$D(s,t) = K(s,t) - K(s) * K(t)$$

Thus, we can use the perspective 3D-plot of the function $D(s,t)$ to obtain a first diagnostic of space-time clustering: positive values indicate an interaction between these two variables at a well-detectable spatio-temporal scale.

4.2.2 Kernel density estimator

The Kernel Density Estimator (KDE) is a non-parametric descriptor tool widely applied in geosciences to elaborate smoothed density surfaces from spatial variables. A kernel function K allows weighing up the contribution of each event, based on the relative distance of neighborings to the target. The parameter h, called bandwidth, controls the smoothness of the estimated kernel density. Finally, the kernel density function $f_h(x)$ is estimated by summing all the kernel functions K computed at each point location x and dividing the result by the total number of events (n):

$$f_h(x) = \frac{1}{nh} \sum_{i=j}^{n} K(\frac{x - x_i}{h})$$

The time extension of the kernel density estimator allows to compute the three-dimensional kernel density estimator which includes the spatio-temporal dimensions(**?**).

In the present case study we apply a quadratic weighting kernel function, which is an approximation to the Gaussian kernel. Regarding the bandwidth's value, we propose to consider the results of the spatio-temporal K-function as an indicator. Indeed, the distance values showing a maximum cluster behavior over the displayed perspective $D(s,t)$ plot can be attributed to the h-value, minimizing the problem of under- or over-smoothing due to an arbitrary choice of the bandwidth.

4.3 Computing lab: ST clustering

4.3.1 Load the libraries

Fist you have to load the following libraries:

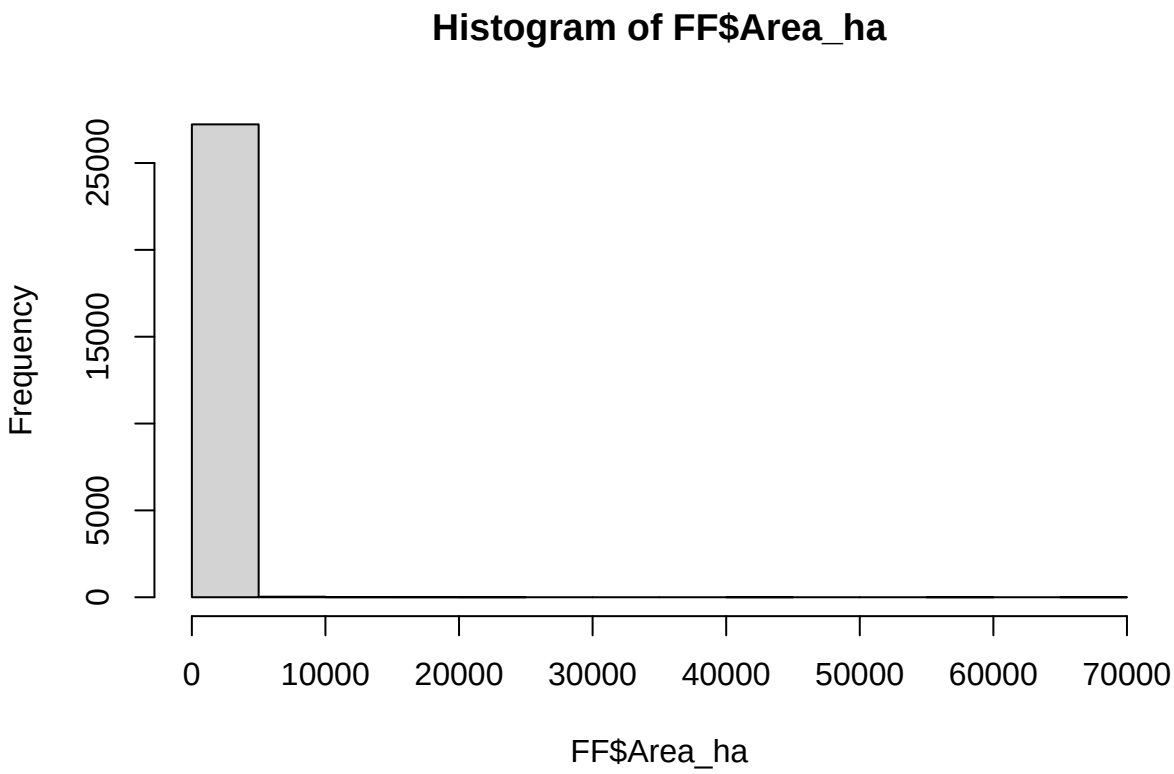
- *splancs*: for display and analysis of spatial point pattern data
- *sf*: support for simple features, a standardized way to encode spatial vector data
- *ggplot2*: s system for 'declaratively' creating graphics
- *sp*: classes and methods for spatial data
- *spatstat*: comprehensive open-source toolbox for analyzing Spatial Point Patterns

```
library(splancs)
library(spatstat)
library(sf)
library(ggplot2)
library(sp)

(.packages())
```

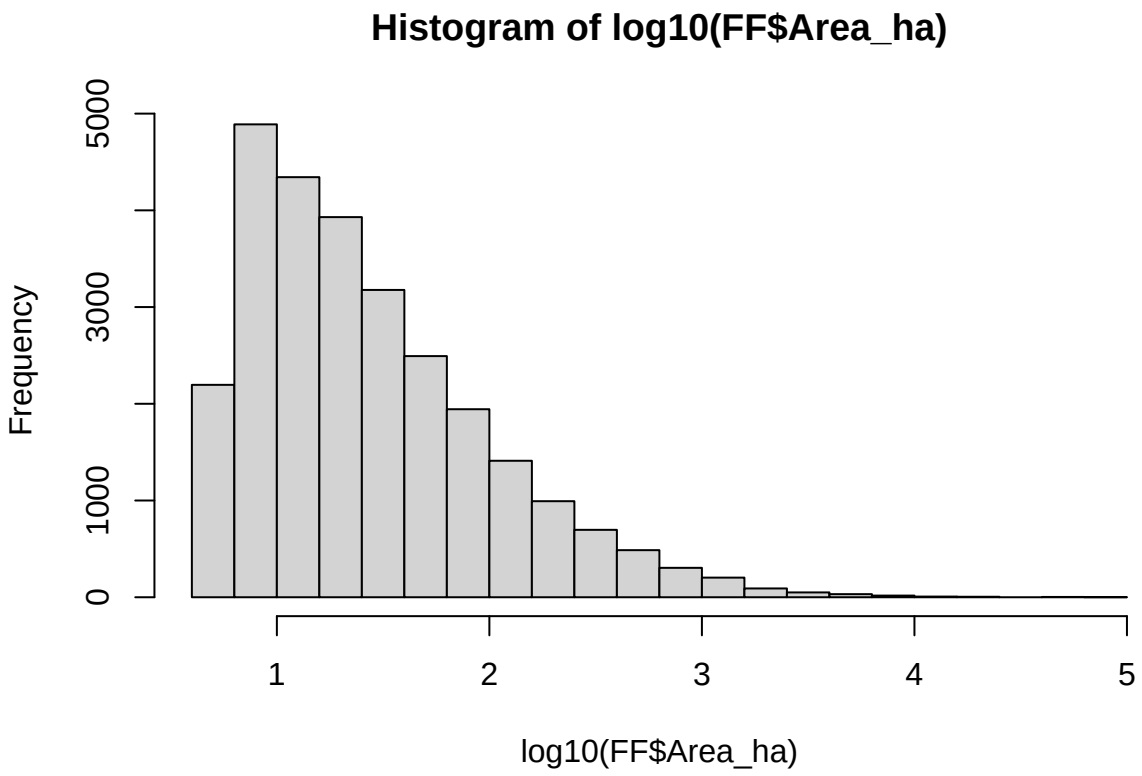
4.3.2 Import the forest fire dataset

In this section you will load the geodata representing the dataset of the forest fires occurred in the continental Portuguese area in the period 1990-2013. You will also load the boundaries of the study area. You will start by exploring the datasets using mainly visual tools (plotting and histogram).

```
# Import Portugal boundary
Portugal <- st_read("data/KDE/Portugal.shp") # entire area

PortN<- st_read("data/KDE/Porto_North.shp") # northern area
PortS<- st_read("data/KDE/Porto_South.shp") # southern

# Import the forest fires dataset

FF<-st_read("data/KDE/ForestFires.shp") # entire area
FFN<-st_read("data/KDE/FF_North.shp") # Northern area
FFS<-st_read("data/KDE/FF_South.shp") # Southern area

# Import the shapefile of the Tagus river
river<-st_read("data/KDE/Rio_Tajo.shp")

summary(FF$Area_ha) # summary statistics
hist(FF$Area_ha)
```




```
hist(log10(FF$Area_ha)) # see the lab GWSS for more details
```



For a better understanding of the phenomenon, events can be grouped according to the size of the burnt area and to their incidence in the northern and in the southern part of continental Portugal.

4.3.3 Forest fires subsets based on the size of the burned area

Remember that fires of different size can have been induced by different drivers. Thus, in the following, we will investigate all the global cluster behavior of forest fires in Portugal considering the three subsets separately.

As you have seen in the lab GWSS, the following three classes can be defined based of the size of the burned area:

- *Small fires:* less than 15 ha
- *Medium fires:* between 15 ha and 100 ha
- *Large fires:* bigger than 100 ha

```
SF=(subset(FF, Area_ha <=15)) #create a sub-set including only small fires.

summary(SF)
```

```
##       Year         Area_ha             X               Y
##  Min.   :1990   Min.   : 5.000   Min.   : 82761   Min.   :  7772
##  1st Qu.:1997   1st Qu.: 6.624   1st Qu.:182973   1st Qu.:403849
##  Median :2001   Median : 8.569   Median :213159   Median :472892
```

```
##  Mean   :2002   Mean   : 9.041   Mean   :219789   Mean   :440692
##  3rd Qu.:2008   3rd Qu.:11.219   3rd Qu.:259054   3rd Qu.:518384
##  Max.   :2013   Max.   :15.000   Max.   :361492   Max.   :573193
##            geometry
##  POINT        :10902
##  epsg:NA      :    0
##  +proj=tmer...:    0
##
##
##
```

```
# This is to save the plot
pSF <- ggplot ()+
  geom_sf(data=Portugal)+
  geom_sf(data=SF, size=0.5, col="yellow")+
 ggtitle("Small fires") +
 coord_sf()
```

```
MF=(subset(FF, Area_ha >15 & Area_ha <=100)) #create a sub-set including only medium fires.

summary(MF)
```

```
##       Year         Area_ha             X                Y
##  Min.   :1990   Min.   :15.00   Min.   : 82127   Min.   :  7084
##  1st Qu.:1997   1st Qu.:21.18   1st Qu.:186031   1st Qu.:403864
##  Median :2001   Median :30.89   Median :219459   Median :468503
##  Mean   :2002   Mean   :38.55   Mean   :223915   Mean   :439442
##  3rd Qu.:2007   3rd Qu.:51.06   3rd Qu.:264349   3rd Qu.:518248
##  Max.   :2013   Max.   :99.98   Max.   :359535   Max.   :572488
##            geometry
##  POINT        :12070
##  epsg:NA      :    0
##  +proj=tmer...:    0
##
##
##
```

```
# This is to save the plot
pMF <- ggplot ()+
  geom_sf(data=Portugal)+
  geom_sf(data=MF, size=0.5, col="orange")+
 ggtitle("Midium fires") +
 coord_sf()
```

```
LF=(subset(FF, Area_ha >100)) #create a sub-set including only large fires.

summary (LF)
```

```
##       Year         Area_ha              X                Y
##  Min.   :1990   Min.   :  100.0   Min.   : 83472   Min.   :  8840
##  1st Qu.:1996   1st Qu.:  139.6   1st Qu.:193831   1st Qu.:383164
##  Median :2002   Median :  220.7   Median :228709   Median :445272
```

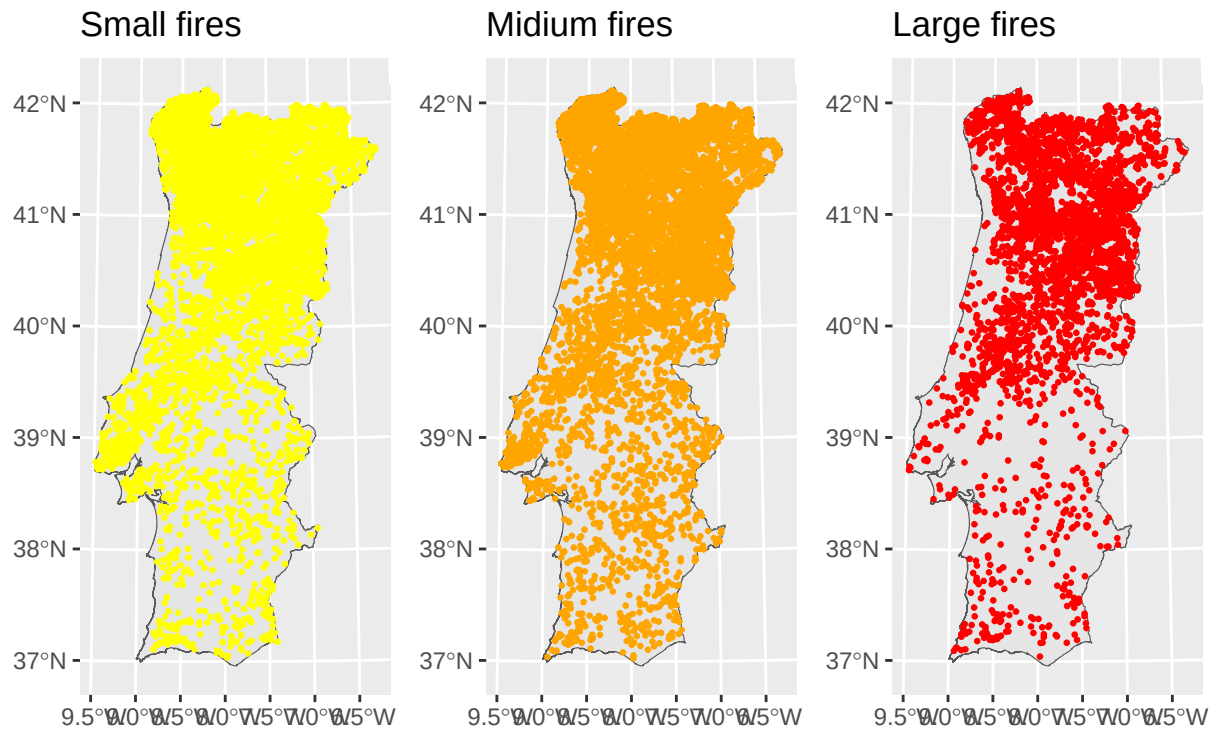
```
##  Mean   :2001   Mean   :  548.3   Mean   :229278   Mean   :423441
##  3rd Qu.:2006   3rd Qu.:  447.9   3rd Qu.:266146   3rd Qu.:500321
##  Max.   :2013   Max.   :66070.6   Max.   :357933   Max.   :571984
##           geometry
##  POINT        :4301
##  epsg:NA      :   0
##  +proj=tmer...:   0
##
##
##
```

```
# This is to save the plot
pLF <- ggplot ()+
  geom_sf(data=Portugal)+
  geom_sf(data=LF, size=0.5, col="red")+
 ggtitle("Large fires") +
 coord_sf()
```

```
# Arrange the three plots side by side

install.packages('patchwork', repos = "http://cran.us.r-project.org")
```

```
## package 'patchwork' successfully unpacked and MD5 sums checked
##
## The downloaded binary packages are in
##  C:\Users\mtonini1\AppData\Local\Temp\Rtmp8SA060\downloaded_packages
```

```
library(patchwork) # Allow to  combine separate ggplots into the same graphic

pSF+pMF+pLF
```

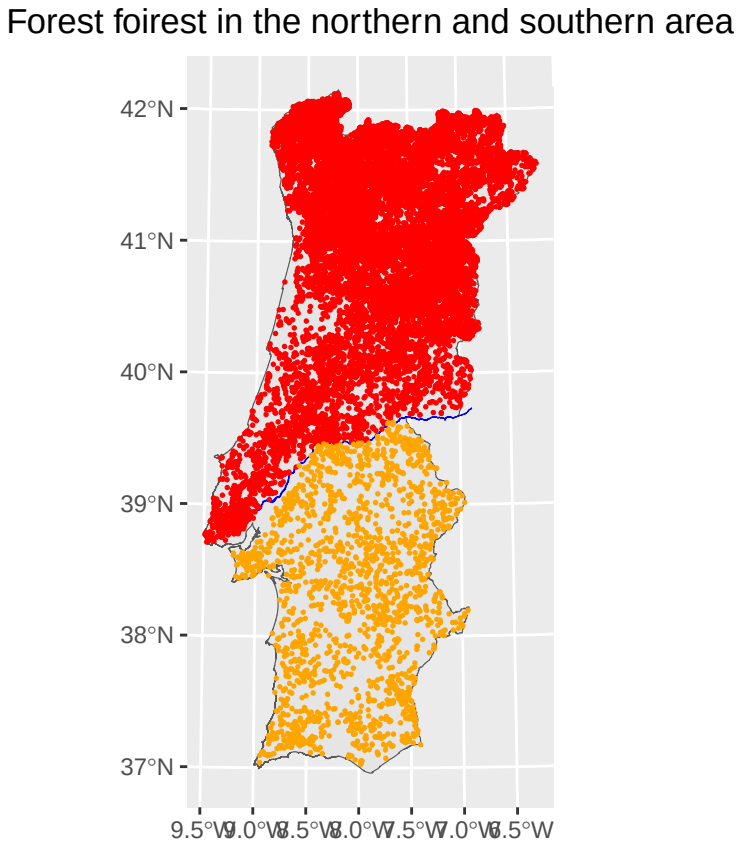


4.3.4 FF-subsets based on their geographical distribution

In continental Portugal, the northern half of the country (above the Tagus River) is characterized by the predominance of forest and semi-natural areas, and by the development of the main cities with their sub-urban ares intermingled with wild land, which makes the northern area highly prone to forest fires.

On the other hand, the southern half of the country is dominated by agricultural areas with mixed and broad-leaved forest concentrated near the south-west coast, which makes this area less affected by forest fires. For this reason we will consider these two areas separately.

```
# Plot the map with all the spatia features
ggplot ()+
  geom_sf(data=Portugal)+
  geom_sf(data=river, col="blue", size=2) +
  geom_sf(data=FFN, size=0.3, col="red") +
  geom_sf(data=FFS, size=0.3, col="orange") +
  ggtitle("Forest foirest in the northern and southern area") +
  theme(plot.title=element_text(hjust=0.5)) +
 coord_sf()
```



4.3.5 Extract time and PTS object

The function **stkhat,** included in the library **spancs** (**?**), allows to compute the space-time Ripley's K-function. As you can see from the R Documentation (command: `help(stkhat)`), same data manipulations are necessary to transform the input data in a compatible data frame format. Namely the user needs to specify:

- the input forest fires dataset, with the coordinates to geolocalize each event (`pts`)
- a vector of times, defined by the starting date of ignition (`times`)
- a polygon of class matrix enclosing the input dataset (`poly`)
- a vector for the spatial distance (`s`) and a vector for the temporal length (`tm`)

```
# Extract "pts" (divided by 1000 to compute in Km)
FFN_pts <- as.points(FFN$X/1000, FFN$Y/1000)
FFN_times<-FFN$Year # extract "times"

# Extract the coordinates (in Km):
PTN_xy<-st_coordinates(PortN$geometry/1000)

# Define the matrix with the set of bounding points ("poly") enclosing the input dataset:
FFN_poly<-PTN_xy[, -c(3,4)]
```

4.3.6 Compute the ST K-function

We compute the space-time K-function for forest fires in the northern area. Since the computation can take a long time (about 20 mints), we propose you to load directly the output R object provided (*STK_North_10y*). The general code is also provided, but preceded by the hashtag, so it is not treated as a command: you have to remove # if you wish to evaluate the code.

- We consider here a subset of forest fires events occurred in the period 2001-2010.
- The parameters `s` and `tm` are defined here as follows: `s`, each kilometer distance up to ten kilometers; `tm`, each year up to five years.

NB: If you wish to run the code, remove # to make it work

```
# Run stkhat function for Northen fires (subset of firs 11 years):

#STK_North_10y <- stkhat(pts=FFN_pts, times=FFN_times, poly=FFN_poly, c(2001,2010), seq(0,10,1), seq(0,

#str(STK_North_10y)
```

```
# Open stkhat documentation
help(stkhat)
```

```
library(readr)

STK_North_10y <- readRDS("data/KDE/STK_North_10y.RData")

str(STK_North_10y)
```

```
## List of 5
##  $ s  : num [1:11] 0 1 2 3 4 5 6 7 8 9 ...
##  $ t  : num [1:6] 0 1 2 3 4 5
##  $ ks : num [1:11] 1.61e-04 9.03 2.98e+01 6.18e+01 1.05e+02 ...
##  $ kt : num [1:6] 0.814 2.29 3.596 4.827 6.085 ...
##  $ kst: num [1:11, 1:6] 1.45e-03 1.06e+01 3.75e+01 7.47e+01 1.22e+02 ...
```

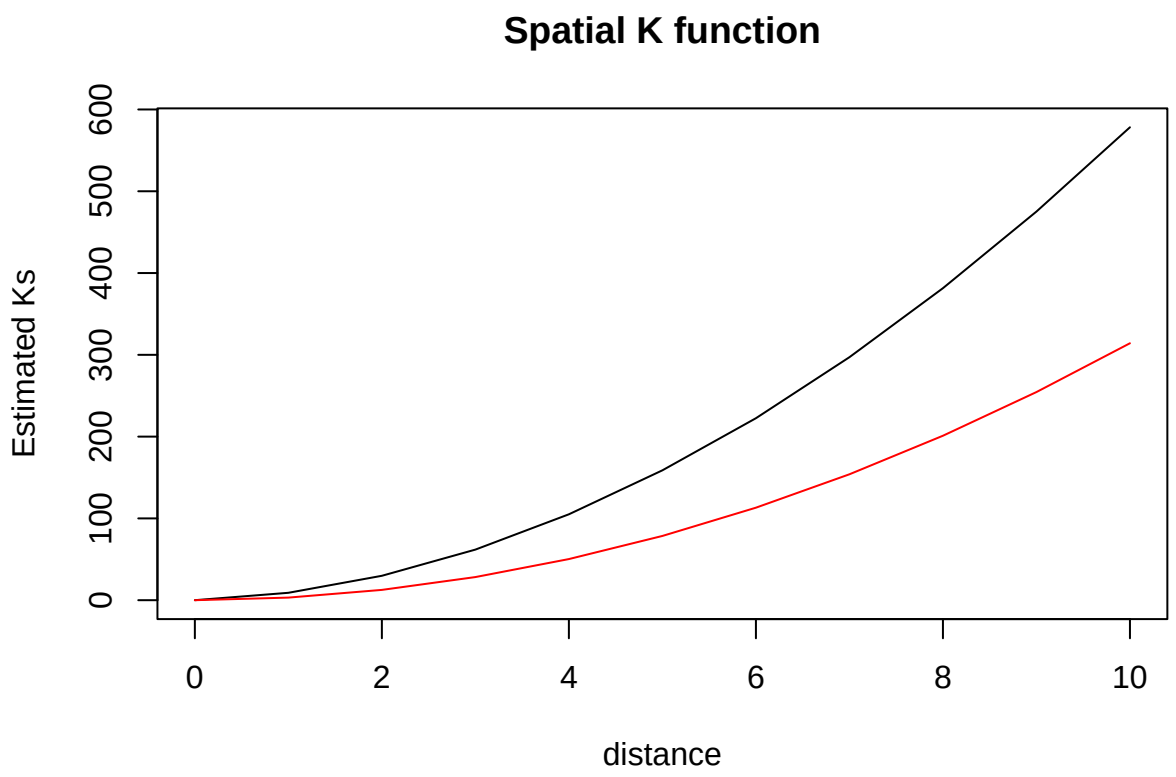
4.3.7 Assess the ST clustering behavior

In the following section you will explore and plot the values of the three components produced as outputs of the function `stkhat`: the spatial K-function (`ks`), the temporal K-function (`kt`); the space-time K-function (`kst`). Then, you will plot the perspective 3D-plot of $D(s,t)$ to evaluate the space-time clustering behavior of forest fires in the present case study. The multifaceted shape of this function, along with the spatial and the temporal dimension, can help to identify peaks of clustering. The corresponding values can be attributed to the bandwidth of the kernel density estimator allowing to elaborate smoothed density maps in the last step of this lab.

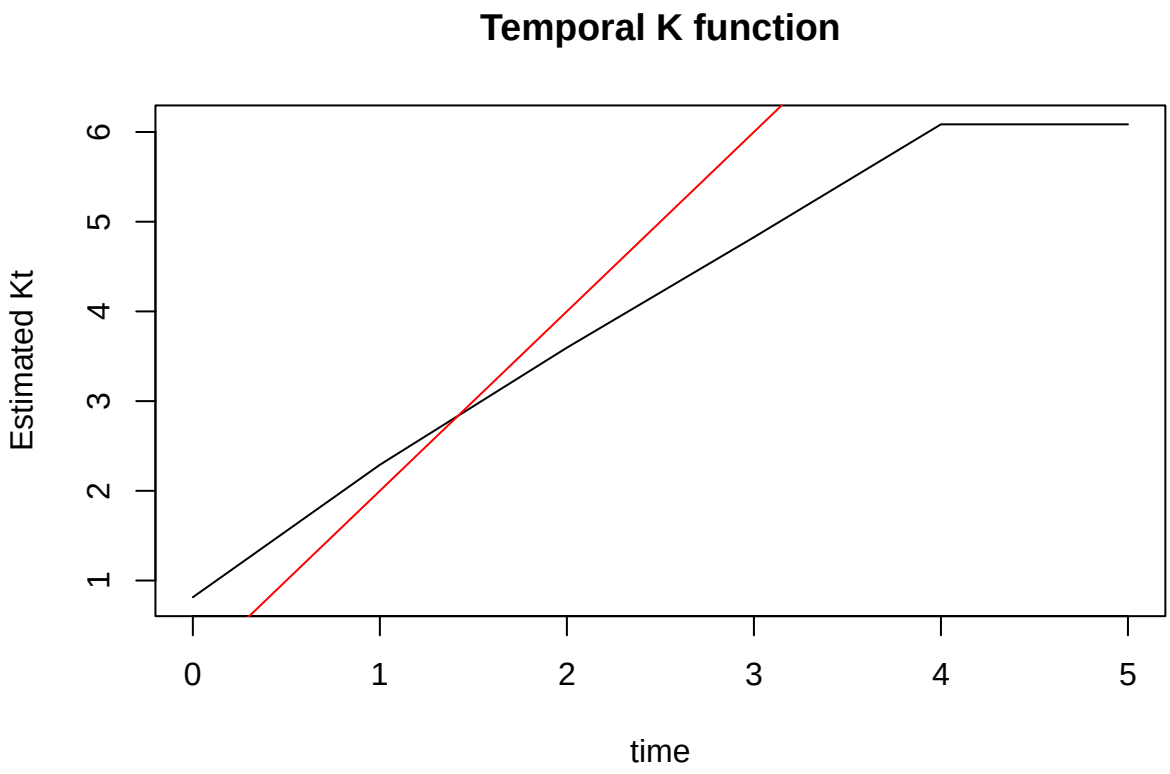
4.3.7.1 Plot the `stkhat`'s outputs

- **Plot of the purely spatial and the purely temporal K function.**

```r
# Plot of the purely spatial K function
plot(STK_North_10y$s, STK_North_10y$ks, type="l", xlab="distance", ylab="Estimated Ks", main="Spatial K
lines(STK_North_10y$s, pi*STK_North_10y$s^2, type="l", col="red")
```

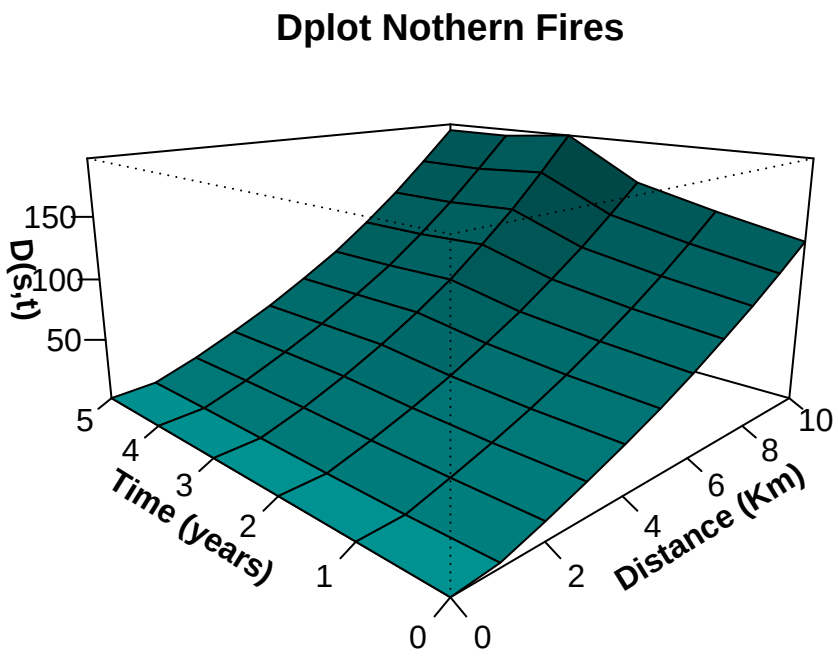



```r
# Plot of the purely temporal K-function
plot(STK_North_10y$t, STK_North_10y$kt, type="l", xlab="time", ylab="Estimated Kt", main="Temporal K fu
lines(STK_North_10y$t, 2*STK_North_10y$t, type="l", col="red")
```



- **Plot the space-time D-plot**

```
# Define the function: D(s,t)=K(s,t)-[K(s)*K(t)]

Dplot <- function (stkhat, Dzero = FALSE, main=TRUE)  {

  oprod <- outer(stkhat$ks, stkhat$kt)
    st.D <- stkhat$kst - oprod
        persp(stkhat$s, stkhat$t, st.D, xlab = "Distance (Km)", ylab = "Time (years)",
            zlab = "D(s,t)", expand = 0.5, ticktype = "detailed",
            theta = -45, shade = 0.75, cex = 0.7, ltheta=120, col="cyan1", font.lab=2)
           }

Dplot(STK_North_10y)
title("Dplot Nothern Fires")
```



4.3.8 Compute the ST Kernel Density Estimator

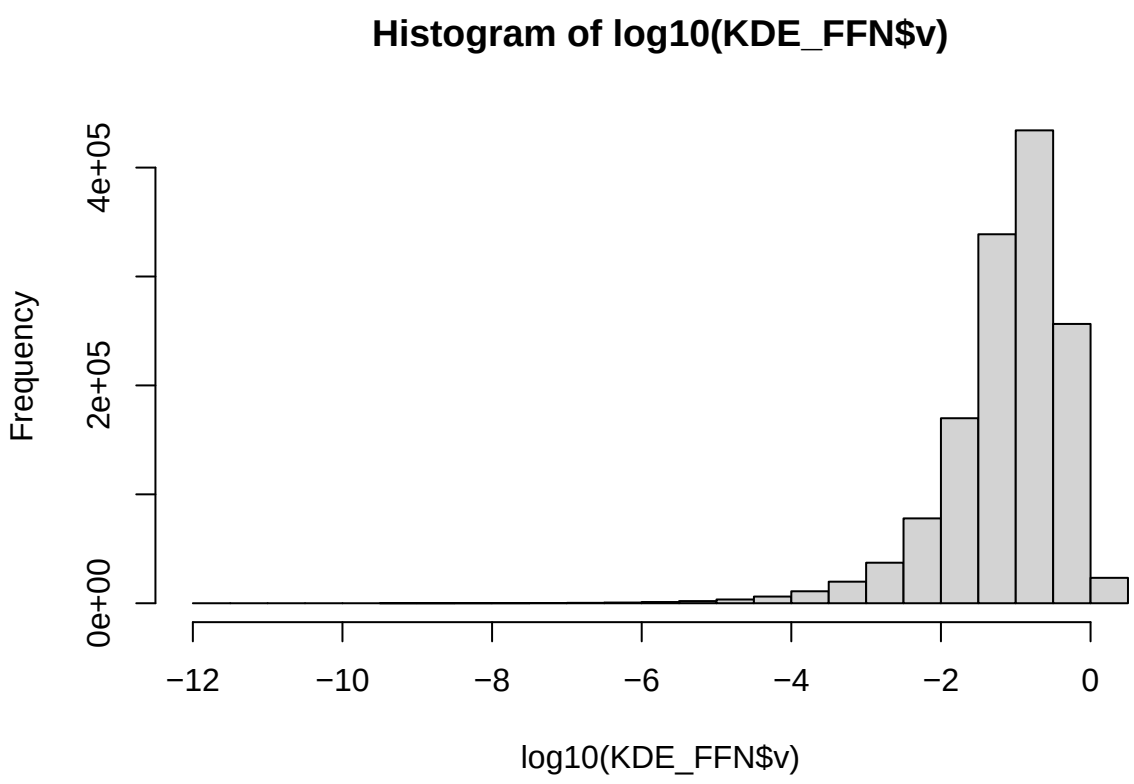
The multifaceted shape the D-Plot identify peaks of clustering at a time interval of 3 year. In space, events are clustered at every distance, so in this case we use the maximum distance-value (10 km). These two values are attributed to the bandwidth of the space-time KDE allowing to elaborate smoothed density maps. To this end, we use the function `kernel3d`, included in the library **spancs** (**?**) too.

```
# Open the help to analyse the parameter of this function:
help(kernel3d)
```

```
# Compute the spatio-temporal KDE
KDE_FFN<-kernel3d(FFN_pts, FFN_times, seq(80, 362, 1), seq(180, 580, 1), seq(1990,2013,1), 10, 3)

#Inspect the results
summary(KDE_FFN$v)
```

```
##      Min.   1st Qu.    Median      Mean   3rd Qu.      Max.
## 0.0000000 0.0000000 0.0002195 0.0995232 0.1093261 2.1627084
```

```
hist(log10(KDE_FFN$v))
```



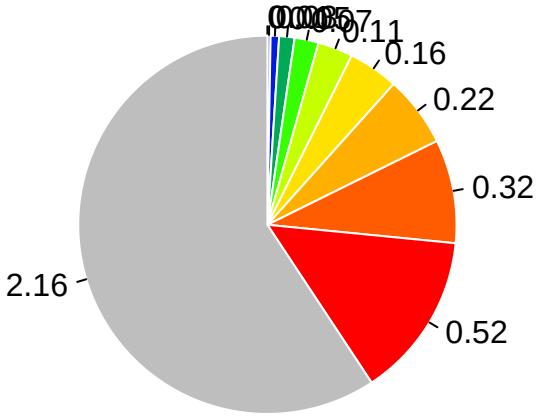

```
min(KDE_FFN$v[KDE_FFN$v>0]) #check the lower non-zero value
```

```
## [1] 2.893333e-12
```

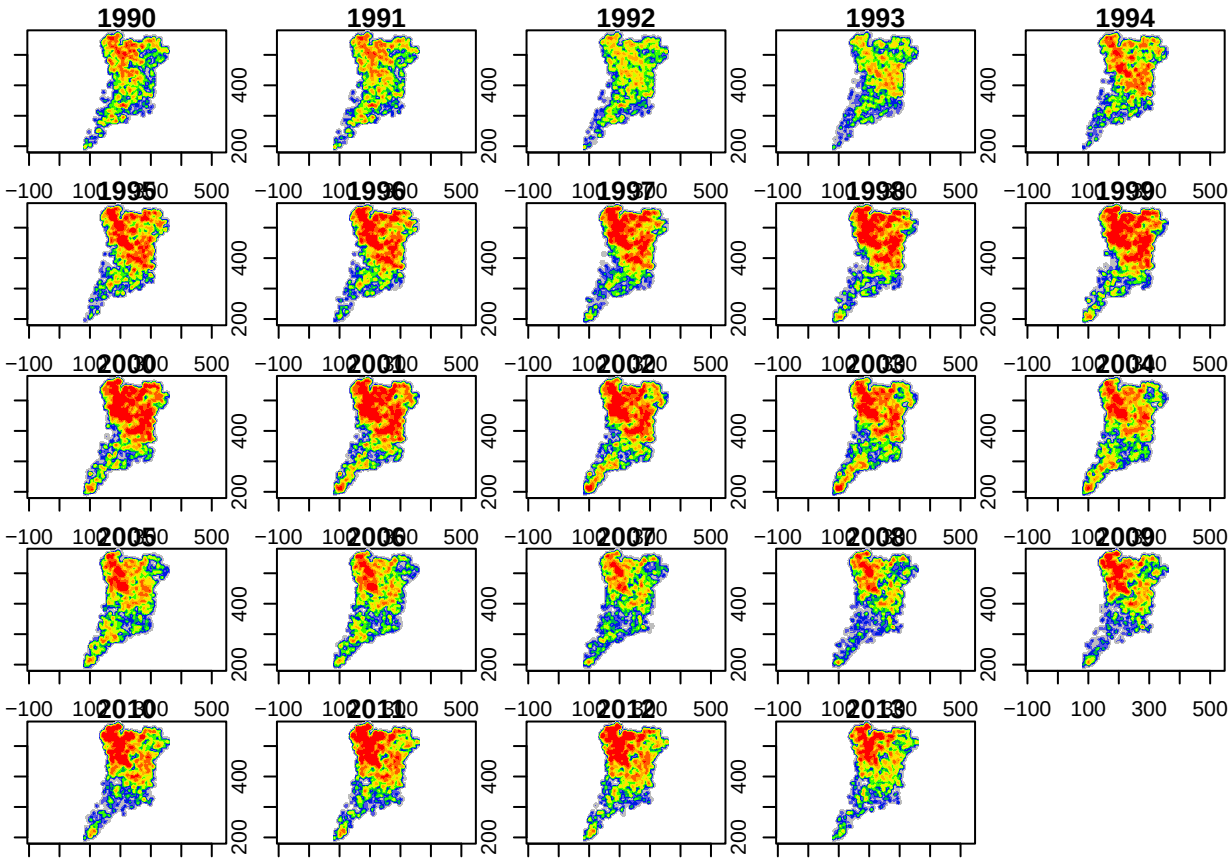
```
# Create quantile clssification
Q<-quantile(KDE_FFN$v, seq(0.5,1,0.05))
Q
```

```
##          50%          55%          60%          65%          70%          75%
## 0.0002194741 0.0097073381 0.0263666531 0.0478856382 0.0748979448 0.1093260662
##          80%          85%          90%          95%         100%
## 0.1560709189 0.2210177893 0.3232627322 0.5162620818 2.1627084390
```

```
# Create a blue/red palette
pal<-colorRampPalette(c("grey","blue","green", "yellow","orange", "red" ))
colsR<-pal(length(Q)-1)

# Display classes
pie(Q, clockwise=TRUE, labels=round(Q, digits=2), border="white", col=colsR)
```




```
# Plot KDE maps for selected years
oldpar<-par(mfrow=c(5,5), mar=c(1,1,1,1))
for (i in 1:24){
 (image(seq(80, 362, 1), seq(180, 580, 1), KDE_FFN$v[,,i], asp=1, xlab="", ylab="", main=1989+i, breaks=
}
```



4.4 Conclusions and further analyses

This practical computing lab allowed you to asses the global cluster behavior of hazardous events, achieved by using the Ripley's k-function. In addition, we learned how smoothed density maps can be elaborated from punctual events, namely using the kernel density estimator. Both the spatial and the temporal dimension have been considered in this case. As practical case study, you could explore the density distribution of forest fires events through Continental Portugal in the period 1990-2013. In the northern half of the country, hot spots are present almost on each investigated years, with a higher concentration in the northern areas.

To ensure that everything is perfectly clear, we propose you to do this lab again, by using this time another subset chosen among the available ones including forest fires in the southern area, the small, medium, or the large forest fires dataset. For whatever dataset you are going to use, try to answer to the following questions. The reading of the paper (**?**), which inspired this lab, can help you in in this task.

1. At which spatial and temporal distance you can observe a peak of clustering?
2. By visually inspecting the KDE maps, describe the spatio-temporal density distribution of forest fires events through the study area.

5 Predictive Mapping of Natural Hazards Using Random Forest

Random Forest (RF) is a robust and widely-used machine learning algorithm particularly suited for predictive mapping in the context of natural hazards and susceptibility assessments. It operates by constructing multiple decision trees during training, and then aggregating their predictions to improve accuracy and

generalizability (**?**). For the sake of clarity, we define *susceptibility* of an area the potential to experience a particular hazard in the future, based only on the intrinsic local properties of the territory, assessed in terms of relative spatial likelihood. Machine Learning (ML) based approaches lend themselves particularly well to this purpose. Essentially ML includes algorithms capable of learning from and making predictions on data by modelling the hidden/non-linear relationships between a set of input variables – also known as predictors – and output observations.

In natural hazard and susceptibility mapping, RF can analyze complex relationships between environmental variables such as topography, soil type, vegetation cover, and climate data to predict the likelihood of hazardous events – like landslides, floods, or earthquakes – across a landscape. Its ability to handle large datasets, manage variables interactions, and provide importance rankings for predictor variables, makes it an invaluable tool for generating reliable susceptibility maps, which are crucial for disaster risk management and land-use planning.

5.1 RF for landslides susceptibility mapping

In this application, we explore the capabilities RF to elaborate landslides susceptibility mapping in Canton Vaud, Switzerland. Landslides are one of the major hazard occurring around the world. In Switzerland, landslides cause damages to infrastructures and sometimes threaten human lives, especially shallow landslides. Such slope movements are mainly triggered by intense rainfalls and generally very rapid and hardly predictable.

The research framework that inspired this computing lab refers to a pioneering study in susceptibility mapping for wildfire events by (**?**) and further developed for the assessment of variable importance by (**?**). Analyses have been adapted to the case study of landslides. The overall methodology is described in the following graphic.

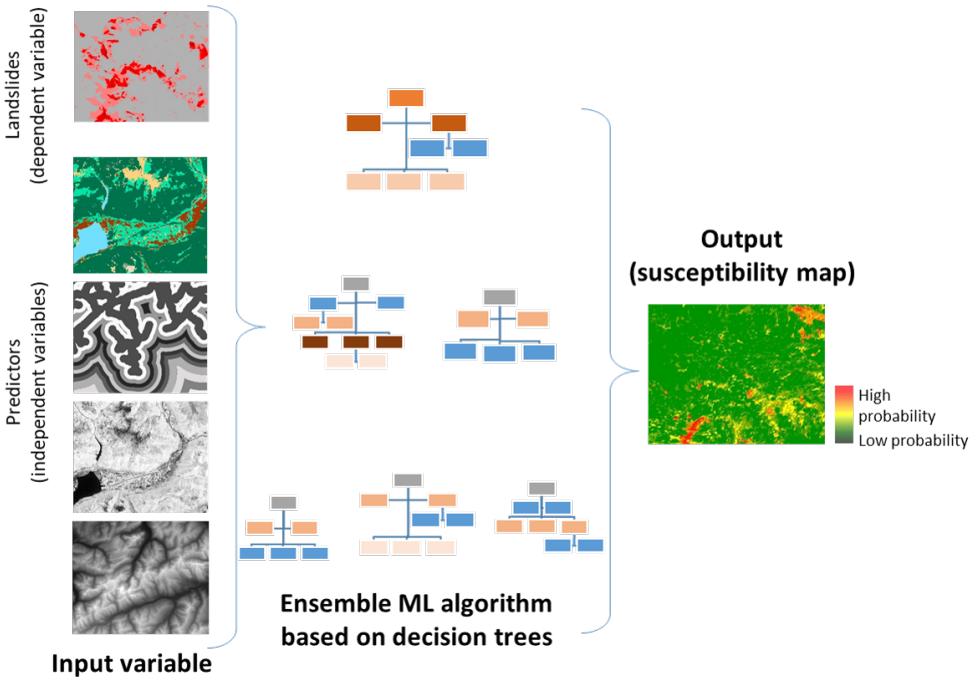


Figure 3: Basic elements of the generic methodology

5.2 Computing lab: Random Forest

5.2.1 Load the libraries

To perform the analysis, you have first to install the following libraries:

- *terra*: methods for spatial data analysis with vector (points, lines, polygons) and raster (grid) data
- *readr*: to provide a fast and friendly way to read tabulat data (like *.csv)
- *randomForest*: classification and regression, based on a forest of trees using random inputs
- *dplyr*: focused on tools for working with data frames
- *pROC*: allowing to compute, analyze ROC curves, and
  - *plotROC*: to display ROC curve
- *ggplot2*: a system for declaratively creating graphics

```
library(terra)
library(readr)
library(randomForest)
library(dplyr)
library(pROC)
library(plotROC)
library(ggplot2)

(.packages())
```

5.2.2 Load the input datasets

In the following you will import the landslides punctual dataset including presences and absences (*LS_pa*) and the predictor variables (in raster format). This will enable to perform the exploratory data analyses step and the understanding of the input data structure.

5.2.2.1 Landslides dataset The landslide inventory has been provided by the environmental office of the Canton Vaud. Only shallow landslides are used for susceptibility modelling. One pixel per landslide-area (namely the one located at the highest elevation) has been extracted. Since the landslide scarp is located in the upper part of the polygon, it makes sense to consider the highest pixel to characterize each single event.

Our model includes the implementation of the landslide **pseudo-absences**, which are the areas where the hazardous events did not took place (i.e. landslide location is known and the mapped footprint areas are available, but the non-landslide areas have to be defined). Indeed, to assure a good generalization of the model and to avoid the overestimation of the absences, pseudo-absences need to be generated in all the cases where they are not explicitly expressed. In this case study, an equal number of point as for presences has been randomly generated in the study area, except within landslides polygons, lakes and glaciers (that is what is called "*validity domain*", where events could potentially occur).

```
# Import the boundary of Canton Vaud
Vaud <- vect("data/RF/Vaud_CH.shp")
Lake <- vect("data/RF/Lakes_VD.shp")


# Import the landslides dataset (dependent variable)
LS_pa <- read.csv("data/RF/LS_pa.csv")
```

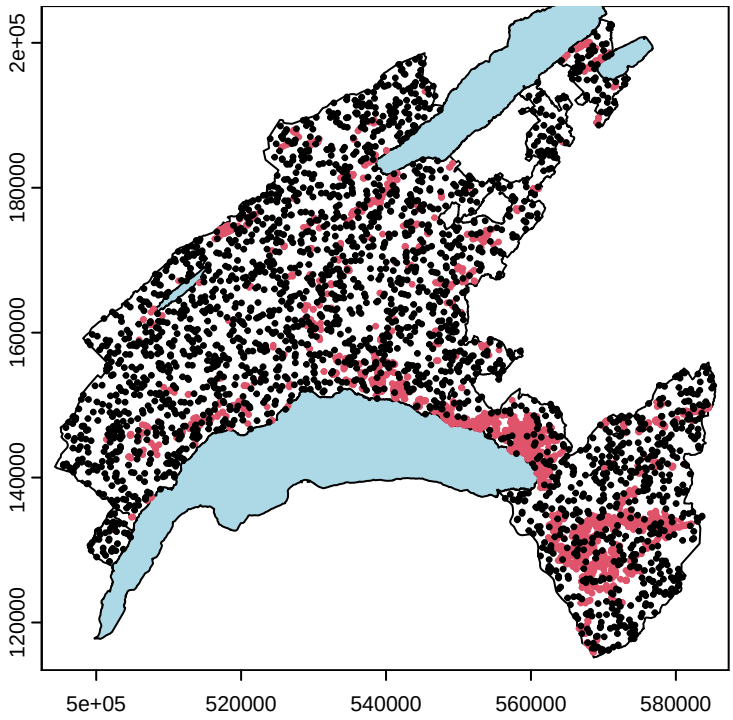
```r
# Convert the numeric values (0/1) as factor
##(i.e. categorical value)
LS_pa$LS<-as.factor(LS_pa$LS)

LS_vect<-vect(LS_pa, geom=c("X", "Y"),crs=crs(Vaud))

# Display the structure (str) and result summaries (summary)
str(LS_vect)
```

```
## S4 class 'SpatVector' [package "terra"]
```

```r
summary(LS_vect)
```

```
##  LS
##  0:2594
##  1:2594
```

```r
# Plot the events
plot(Vaud)
plot(Lake, col="lightblue", add=TRUE)
plot(LS_vect, col=LS_pa$LS, pch=20, cex=0.5, add=TRUE)
```



5.2.2.2 Predictor variables Selecting predictive variables is a key stage in landslide susceptibility modelling when using a data-driven approach. There is no consensus about the number of variables and which variables should be used. In the present exercise we will use the following:

- **DEM (digital elevation model)**: provided by the Swiss Federal Office of Topography. The elevation is not a direct conditioning factor for landslide; however, it can reflect differences in vegetation characteristics and soil.
- **Slope**: is one of the most explicating factor in landslide susceptibility modelling.

$$Slope = arctan(\sqrt{(dz/dx)^2 + (dz/dy)^2}) * (\pi/2)$$

- **Curvature**: curvature is widely used in landslide susceptibility modelling. It allows assessing the water flow acceleration and sediment transport process (*profile curvature*) and the water flow propensity to converge and diverge (*plan curvature*). They have been derived from DEM and directly provided here.
- **TWI (topographical water index)**: topography plays a key role in the spatial distribution of soil hydrological conditions. Defining α as the upslope contributing area describing the propensity of a cell to receive water, and β as the slope angle, TWI (compute by the formula below), reflects the propensity of a cell to evacuate water:

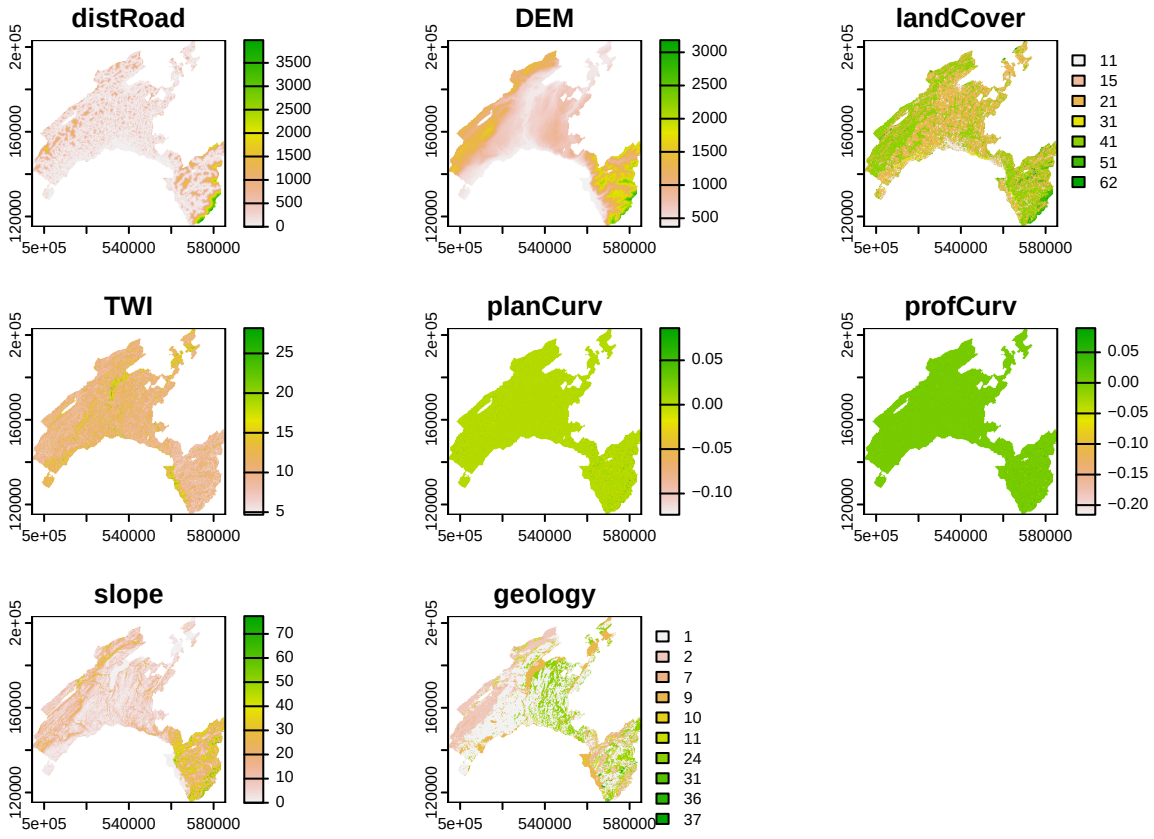
$$TWI = ln(\alpha/tan(\beta))$$

- **Distance to roads**: roads build in mountainous areas often cut the slope, weakening the cohesion of the soil. Moreover, roads surfaces are highly impermeable. This raster has been elaborated by computing the euclidean distance from the swissTLMRegio map where roads are represented by lines.
- **Land Cover**: developed by the Swiss administration and based on aerial photographs and control points. It includes 27 categories distributed in the following 6 domains: human modified surfaces, herbaceous vegetation, shrubs vegetation, tree vegetation, surfaces without vegetation, water surfaces (glaciers included).
- **Geology**: the use of the lithology increase the performance of the susceptibility landslide models. We use here the map elaborated by the Canton Vaud, defining the geotypes and reclassified in 10 classes in order to differentiate sedimentary rocks.

Than the predictor variables have to be aggregated into a single object, storing multiple rasters. We use here the generic function `c` to combine the single raster into a multiple-raster object.

```
## Import raster (independent variables) 25 meter resolution

landCover<-as.factor(rast("data/RF/landCover.tif"))
geology<-as.factor(rast("data/RF/geology.tif"))

planCurv<-rast("data/RF/plan_curvature.tif")/100
profCurv<-rast("data/RF/profil_curvature.tif")/100
# this because the input values was originally multiplied by 100

TWI <- rast("data/RF/TWI.tif")
Slope <- rast("data/RF/Slope.tif")
dem <- rast("data/RF/DEM.tif")
dist <- rast("data/RF/dist_roads.tif")
```

```r
# Combine raster

features<-c(dist, dem, landCover, TWI, planCurv, profCurv, Slope, geology)

# Renames the input features
names(features)<-c("distRoad", "DEM", "landCover", "TWI", "planCurv", "profCurv", "slope", "geology")

# Mask to DEM extension
features <- terra::mask(features, dem)

plot(features)
```

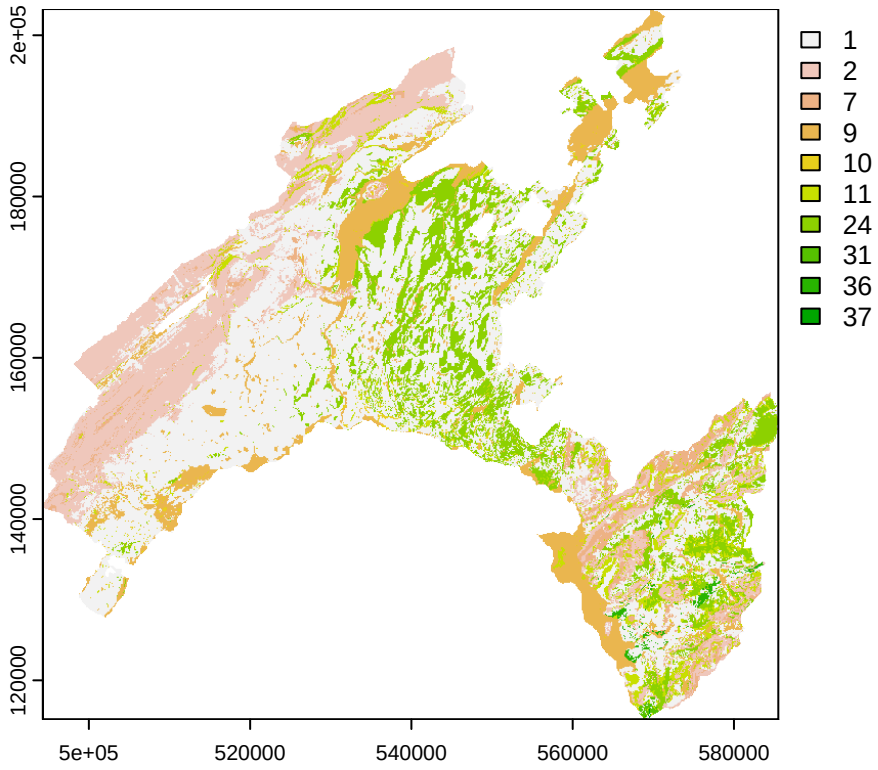


5.2.3 The use of categorical variables in Machine Learning

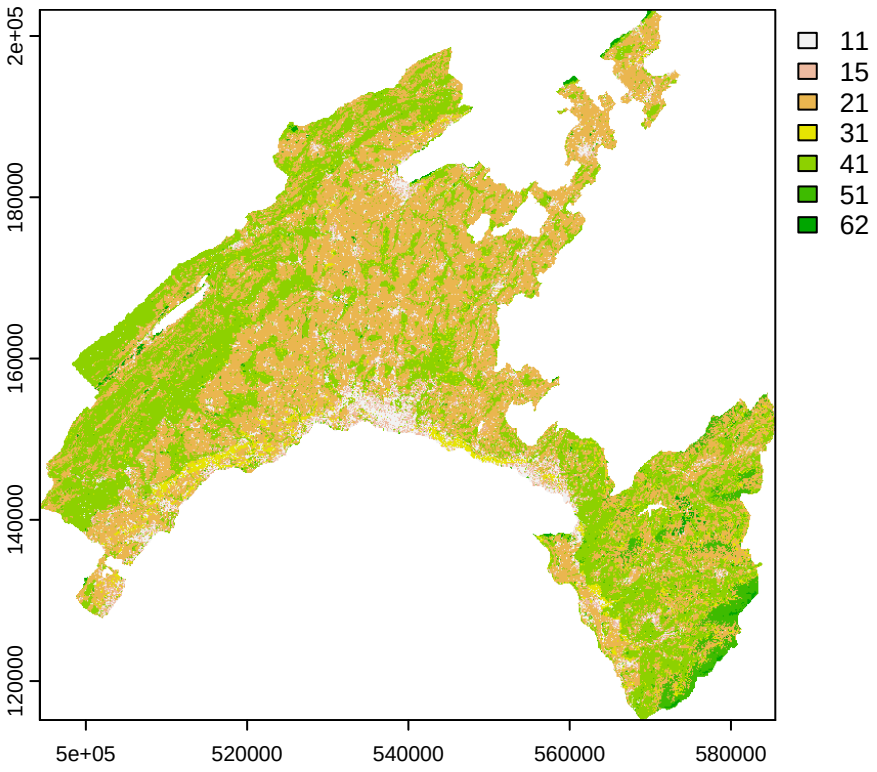
The majority of ML algorithms (e.g., support vector machines, artificial neural network, deep learning) makes predictions on the base of the proximity between the values of the predictors, computed in terms of euclidean distance. This means that these algorithms can not handle directly categorical values (i.e., qualitative descriptors). Thus, in most of the cases, categorical variables need to be transformed into a numerical format. One of the advantage of using Random Forest (as implemented in R) is that it can handle directly categorical variables, since the algorithm operate by constructing a multitude of decision trees at training time and the best split is chosen just by counting the proportion of each class observation.

To understand the characteristics of the categorical variables, you can plot the tow raster **Land Cover** and **Geology** by using their original classes and look at the attribute table to analyse the corresponding definitions.

```
plot(geology)
```




```
plot(landCover)
```


2e+05
180000
160000
140000
120000
5e+05
520000
540000
560000
580000
11
15
21
31
41
51
62

ID	Geological classes
1	Glacial deposits
2	Carbonates
7	Marly-limestone
9	Alluvial deposits
10	Artificial materials
11	Slope deposits
24	Detrital rocks
31	Metamorphic rocks
36	Evaporitic rocks
37	Magmatic rocks

ID	Land Cover classes
11	Impermeable man-made
15	Permeable man-made
21	Herbaceous vegetation
31	Shrub vegetation
41	Forest
51	Bare soil
62	Glacier and water body

Figure 4: Categorical variables

5.2.4 Extract the values

In this step, you will extract the values of the predictors at each location in the landslides (presences and absences) dataset. The final output represents the input dataset with dependent (LS = landslides) and independent (raster features) variables.

```
# Extract values from the raster dataset (features)
LS_input <-extract(features, LS_vect, method="simple",  xy=TRUE)

LS_input$LS <- as.factor(LS_vect$LS) # add LS
str(LS_input) # explore the dataset

# Remove extra column (ID)
LS_input <- LS_input[,2:ncol(LS_input)]
LS_input<-na.omit(LS_input)

# Explore the newly created input dataset
head(LS_input)
str(LS_input)
```

5.2.5 Split the input dataset into training and testing

A well-established procedure in ML is to split the input dataset into training, validation, and testing.

- The **training dataset** is needed to calibrate the parameters of the model, which will be used to get predictions on new data.
- The purpose of the **validation dataset** is to optimize the hyperparameter of the model in the training phase. NB: in RF this subset is represented by the *Out-Of-Bag* (**OOB**)!
- The **testing dataset** is used in thein the prediction phase: results are predicted over these "new" observations (unused before) to provide an unbiased evaluation of the final model and to assess its performance.

```
# Shuffle the rows
set.seed(123) # to ensure reproducibility
LS_input_sh<-LS_input [sample(nrow(LS_input), nrow(LS_input)), ]

# Split the input dataset into training (80%) and testing (20%)
n <- nrow (LS_input_sh)
set.seed(123)
n_train <- round(0.80 * n)
train_indices <- sample(1:n, n_train)

# Create indices
LS_train <- LS_input_sh[train_indices, ]
LS_test <- LS_input_sh[-train_indices, ]

# Count the number of elements in the two subset: training and testing
count(LS_train)
count(LS_test)
```

5.2.6 Run Random Forest

In RF a subset of the training dataset is generated by bootstrapping (i.e. random sampling with replacement). For each subset a decision tree is grown and, at each split, the algorithm randomly selects a number of variables (`mtry`) and it computes the Gini index to identify the best one. The process stops when each node contains less than a fixed number of data points. The main hyperparameters that needs to be defined in RF are `mtry` and the total number of trees (`ntrees`). In this lab we fix these values to 3 and 500, respectively.

The prediction error on the training dataset is finally assessed by evaluating predictions on those observations that were not used in the subset, defined as "out-of-bag" (OOB). This values is used the optimize the values of the hyperparameters, by a trial and error process (that is, trying to minimize the OOB estimate of error rate).

For the computation we introduce here the method proposed by **?** and implemented in the R package `randomForest` (**?**).

```
# Set the seed of R's random number generator,
## this is useful for creating simulations that can be reproduced.
set.seed(123)

# Run RF model
RF_LS<-randomForest(y=LS_train$LS, x=LS_train[1:8],data=LS_train, ntree=500, mtry=3,importance=TRUE)
```

5.2.7 RF main outputs

Printing the results of RF allows you to gain insight into the outputs of the implemented model, namely the following: a summary of the model hyperparameters, the OOB estimate of error rate, the confusion matrix (in this case a 2x2 matrix used for evaluating the performance of the classification model: 1==presence *vs* 0==absence).

The plot of the error rate is useful to estimate the decreasing values on the OOB and on the predictions (1==presence *vs* 0==absence) over increasing number of trees.

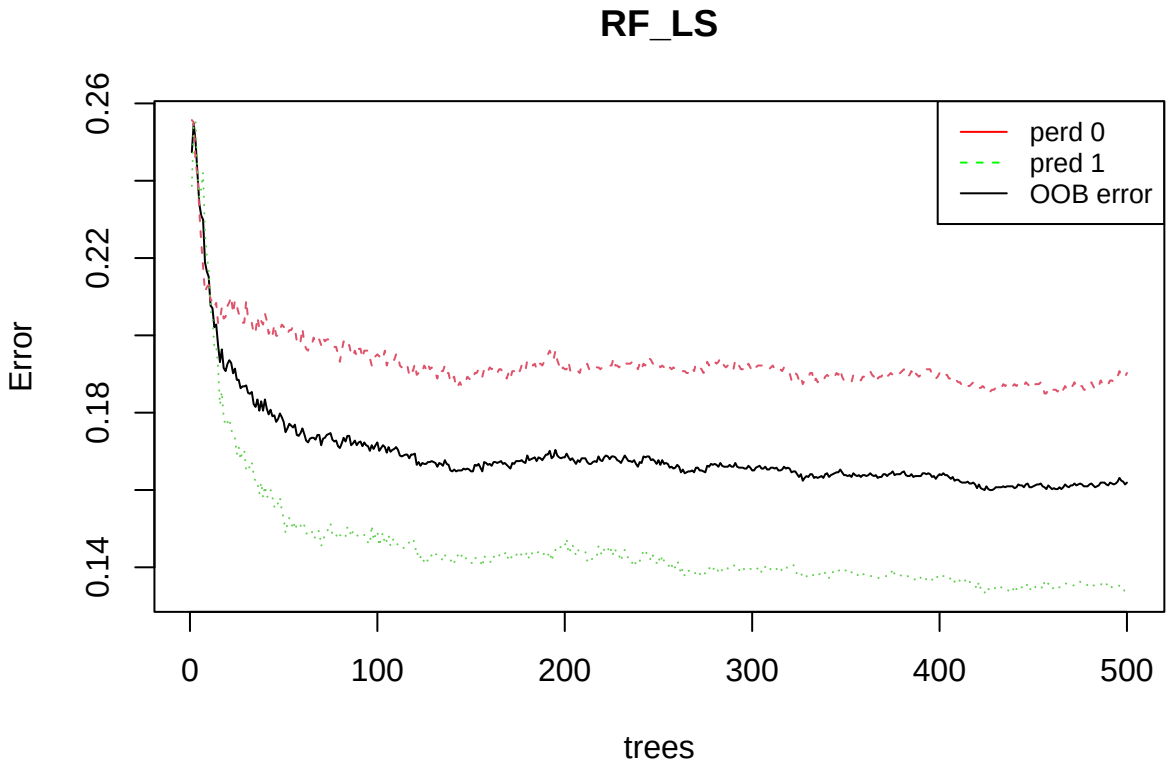
```
# Print the model setting
print(RF_LS)
```

```
##
## Call:
##  randomForest(x = LS_train[1:8], y = LS_train$LS, ntree = 500,      mtry = 3, importance = TRUE, data
##                Type of random forest: classification
##                      Number of trees: 500
## No. of variables tried at each split: 3
##
##         OOB estimate of  error rate: 16.19%
## Confusion matrix:
##      0    1 class.error
## 0 1681  395   0.1902697
## 1  275 1787   0.1333657
```

```
# Show the predicted probability values
RF.predict <- predict(RF_LS,type="prob")
head(RF.predict) # 0 = absence ; 1 = presence
```

```
##               0         1
## 1447 0.20108696 0.7989130
## 671  0.01463415 0.9853659
## 1121 0.22099448 0.7790055
## 4443 0.79207921 0.2079208
## 4032 0.21500000 0.7850000
## 2075 0.16666667 0.8333333
```

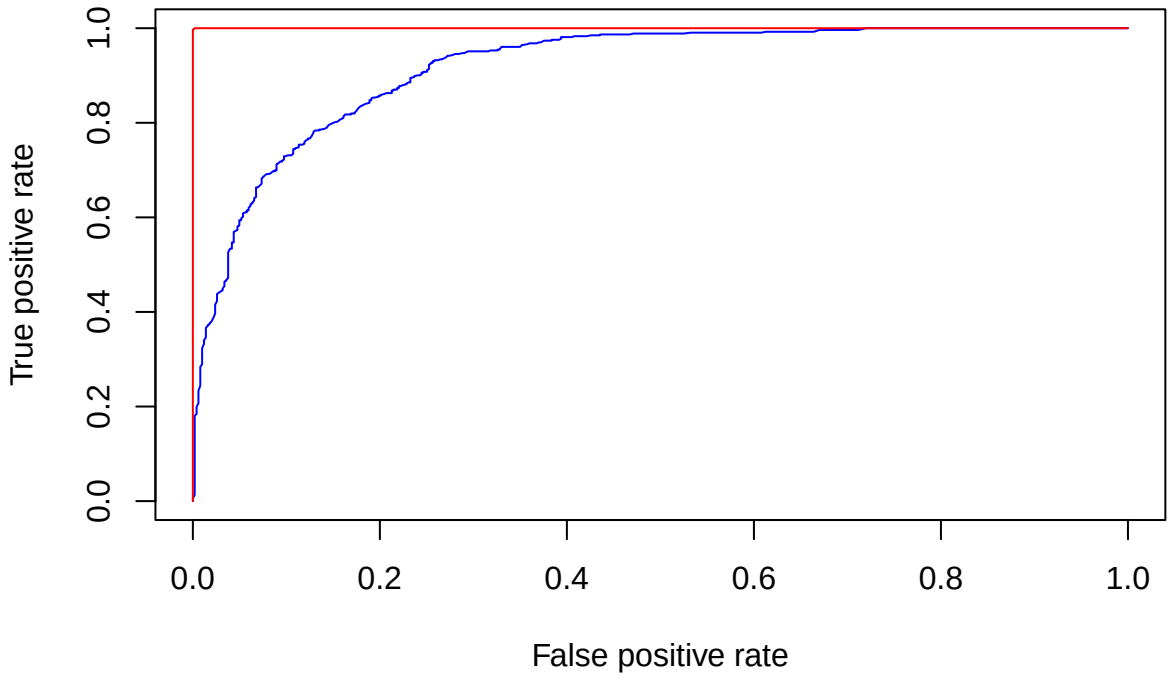
```r
# Plot the OOB error rate
plot(RF_LS)
legend(x="topright", legend=c("perd 0", "pred 1", "OOB error"),
 col=c("red", "green", "black"), lty=1:2, cex=0.8)
```



5.2.8 Model evaluation

The prediction capability of the implemented RF model can be evaluated by predicting the results over previously unseen data, that is the testing dataset. The *Area Under the "Receiver Operating Characteristic (ROC)" Curve* (**AUC**) represents the evaluation score used here as indicator of the goodness of the model in classifying areas more susceptible to landslides. The ROC curve is a graphical technique based on the plot of the percentage of correct classification (the true positives rate) against the false positives rate (occurring when an outcome is incorrectly predicted as belonging to the class "1" when it actually belongs to the class "0"), evaluates for many thresholds. The AUC value lies between 0.5, denoting a bad classifier, and 1, denoting an excellent classifier, which, on the other hand, can indicate overfitting.

```r
# Make predictions on the testing dataset
RFpred_test <- predict(object = RF_LS, newdata = LS_test, type="prob")

# Make predictions on the validation dataset (taining using the Out-of-bag)
RFpred_oob <- predict(object = RF_LS, newdata = LS_train, type="prob", OOB=TRUE)

roc_test <- roc(LS_test$LS, RFpred_test[,2])
roc_oob <- roc(LS_train$LS, RFpred_oob[,2])

plot.new()
plot(1-roc_test$specificities, roc_test$sensitivities, type = 'l', col = 'blue',  xlab = "False positive
lines(1-roc_oob$specificities, roc_oob$sensitivities, type = 'l', col = 'red')
```

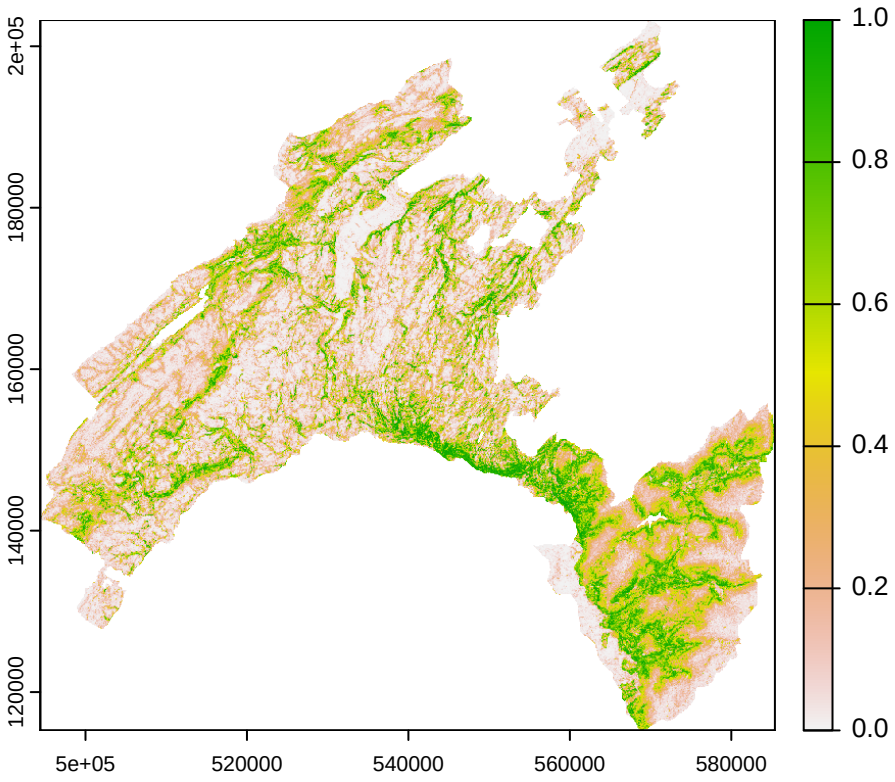



```r
# Print AUC values
roc_test
roc_oob
```

5.3 Susceptibility mapping

You have now all the elements necessary to elaborate the final landslide susceptibility map. This can be achieved by making predictions (of presence only) based on the values of the predictor variables, which are stored into the multiple-raster named *features*, created above.

```r
# Convert the input multiple raster to data frame
features_df<-as.data.frame(features, xy=TRUE, na.rm=TRUE)

# Predict results of RF (probability of fires presence: index = 2) to the feature space
## this operation can take several minuts to run!
scp_map<-predict(object = RF_LS, newdata = features_df, type="prob", index=2)

scp_df=as.data.frame(scp_map) # convert to data frame
# get coordinates
scp_df$X=features_df$x
scp_df$Y=features_df$y

# Convert to raster the probability to get a landslide
## 3=X, 4=Y, 2=probability of presence (1)
scp_rast=rast(scp_df[,c(3,4,2)],type="xyz")
summary(scp_rast)
plot(scp_rast)
```




```r
# Save all outputs
## this operation can take several minuts to run!
save.image(file="LSM_RF.RData")

# Export susceptibility map as raster
writeRaster(scp_rast,"Susceptibility_LSmap.tif",overwrite=T)
```

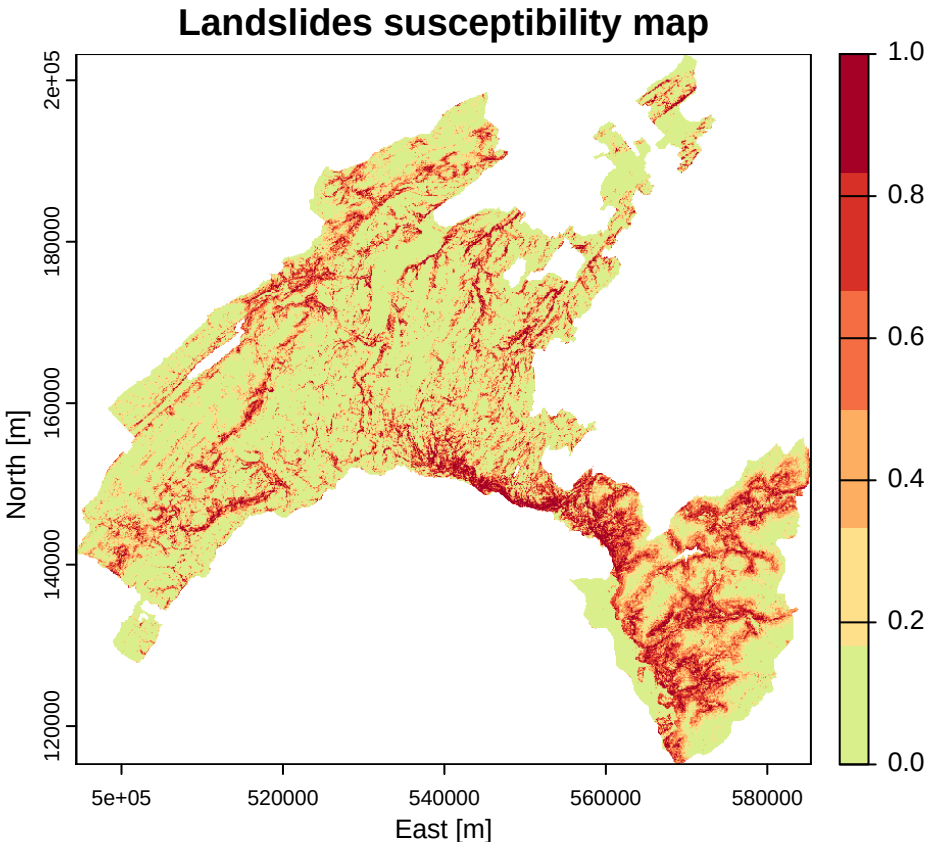
5.3.1 Class intervals for decision maker

What can you say by looking at this map? Actually a risk heat map like this provide a data visualization tool for communicating the level for a specific risk to occur. These maps helps authorities to identify and prioritize the risks associated with a given hazard.

Normally an authority (i.e., a decision maker) prioritize its efforts based on the available resources it has. So, it can be more useful to detect the areas with the highest probability of burning based on certain intervals (i.e., breaks). The authority can thus concentrate its resources for preventive actions on a given threshold (such as 5%, 10%, or 20%) of the area with the highest probability of burning, instead of concentrates on the areas with a "stochastic" output probability value of 0.8 (for example).

5.3.1.1 Equal intervals Susceptibility maps are based on equal intervals, five classes (each 20%) in this case.

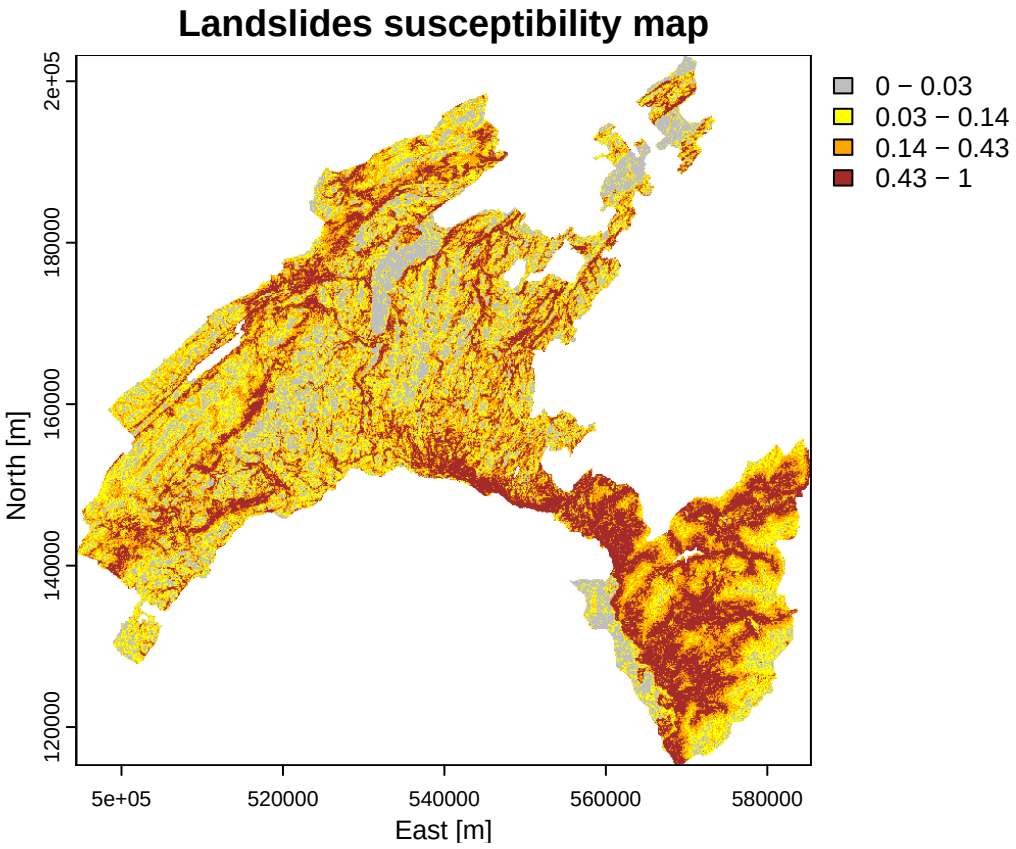
```
library("RColorBrewer")
plot(scp_rast, xlab = "East [m]", ylab = "North [m]", main = "Landslides susceptibility map", col = rev
```



5.3.1.2 Quartile Breaks are chosen based on the summary statics: these values corresponds to the quartiles of the p-value distribution (values divided into four equal partitions).

```
brk<-c(0, 0.03, 0.14, 0.43, 1)

plot(scp_rast, xlab = "East [m]", ylab = "North [m]",
```

```
     main = "Landslides susceptibility map",
     col = rev(c("brown", "orange", "yellow", "grey")), breaks=brk)
```

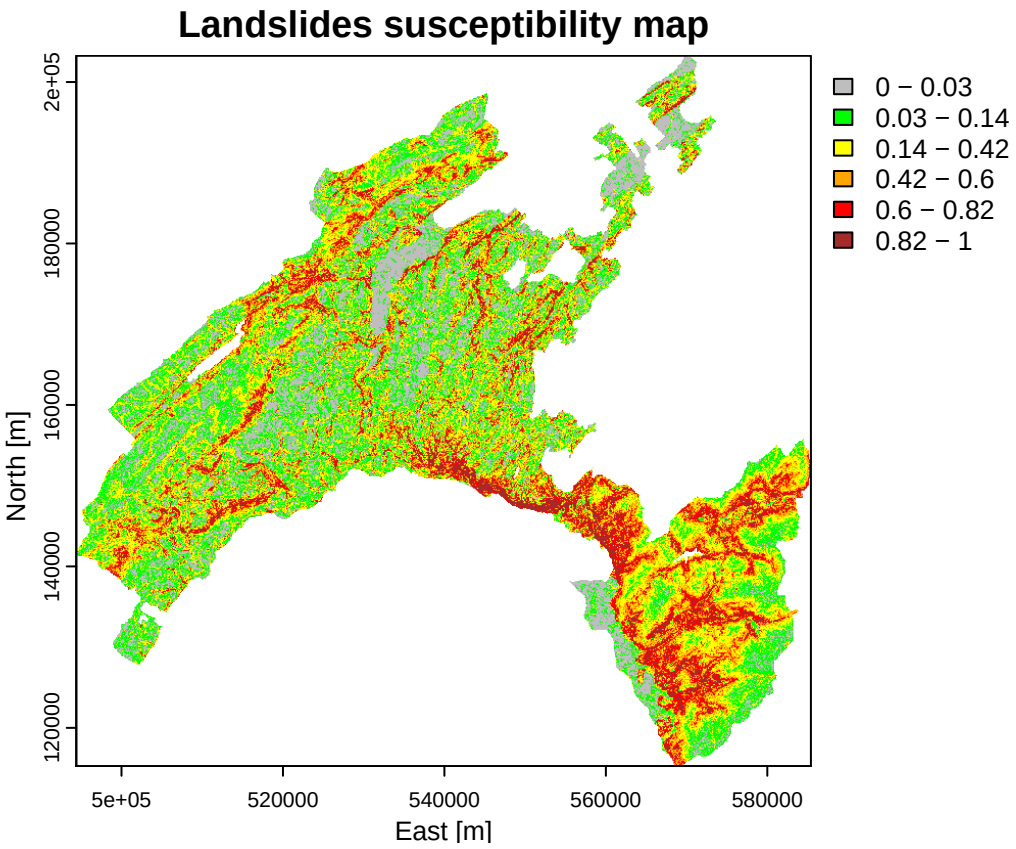


5.3.1.3 Percentiles Breaks are chosen based on well-established percentile classes. For exaple, in this case the 20th percentile correspond to the p-value below wich the 25% of the observation lie, while the 95th can be interpreted as the p-value above which the complementary 5% of the observation lie. The legend show the p-values correspondig to the percentile classes indicated below.

```
# Output predicted values are transformed to a vector
pred.vect <- as.vector(scp_map[, 2])

# The function "quantile" is used to fix classes
qtl.pred <- quantile(pred.vect, probs=c(0.25,0.5,0.75,0.85,0.95), na.rm = TRUE)
qtl.pred
```

```
##   25%   50%   75%   85%   95%
## 0.046 0.166 0.448 0.614 0.832
```

```
# and then extract the corresponding values
qtl.int<- c(0,0.03,0.14,0.42,0.6,0.82,1)
plot(scp_rast, xlab = "East [m]", ylab = "North [m]",
     main = "Landslides susceptibility map",
     col = rev(c("brown", "red", "orange","yellow", "green", "grey")), breaks=qtl.int)
```



5.4 Conclusions and further analyses

This exercise allowed you to familiarize with Random Forest, by the proposed application about landslides susceptibility mapping and variables importance assessment.

To ensure that everything is perfectly clear, we propose you to **answer the following questions**.

1) Why is it important to implement the pseudo-absences, other that the presences (i.e., the observations) in a data-driven modelization?

2) What is the difference between a numerical and a categorical variable? Give some examples of both types. Why RF can handle directly native categorical variables? Is it the same for other algorithms (like logistic regression or neural network)?

3) Which is the values of the OOB estimate error rate of your model? Which parameters you can change to try to reduce it? Be brave and do it (i.e., change the values for *ntree* and *mtry,* than analyse which values for the AUC you obtain and which model perform better.

6 Interpretability & Explainability with Random Forest

In the world of machine learning, interpretability and explainability are two therms commonly used to describe the extent to which an algorithm's behavior can be understood. The distinction between them lies in their focus and depth. We can say that interpretability focuses on understanding the inner workings of the models, while explainability focuses on explaining the decisions made. Define if an algorithm is interpretable or explainable depend on different factors as described below.

Model complexity – When dealing with intricate models like Random Forest (with tens of variables and thousands of trees), up to deep neural networks, interpretability becomes challenging due to their complexity and the interplay among their components. In such scenarios, explainability proves to be a more practical approach, as it focuses on clarifying decisions rather than delving into the complexities of the algorithm.

Communication – In terms of audience and purpose, interpretability primarily concerns machine learning specialists , whereas explainability targets end users seeking to grasp model decisions. Consequently, explainability necessitates a more straightforward and intuitive communication of information.

While Random Forest (RF) is a powerful algorithm and often yield high accuracy, interpretability can be challenging due to its complex structure and the high number of tress. However, the following techniques can enhance the interpretability and explainability of RF models.

- **Interpretability** – A *surrogate model*, such as a single decision tree, can approximate the predictions of a more complex model like a RF, which is composed of thousands of decision trees. The surrogate model is more interpretable and helps in understanding the general rules that RF tipically follows.
- **Explainability** – Examining *feature importance scores*, which measure the contribution of each variable to the model's predictions, allows us to identify the most influential variables in the model's decisions. In addition, *partial dependence plots* enable us to visualize how changes in a variable influence the model's predictions, making this tool useful for interpreting the global effects of predictors across the entire dataset.

6.1 Aim of the present lab

In this computing lab you will work with the outputs of RF resulting from the previous lab on landslides susceptibility map.

- Firstly, we will explore the relative importance of the predictor variables (feature importance scores) , and their relative probability of prediction success (partial dependence plots). These are the core of explainability in RF.
- In the second part, we will apply a local version of RF (named "*Geographical Random Forest*") to analyse the spatial heterogeneity of the local variable importance. This will help to deep our understanding of the influence of the predictor variable explored locally,

6.2 Computing lab: Understanding Random Forest Models

6.2.1 Re-load libraries and workspace

If you have quit the workspace where you have run the RF model for landslide susceptibility mapping, you need to load it again in this new project. Loading the workspace refers to the action of restoring the saved state of R environment. When you save your workspace in R, it typically includes all the objects (such as variables, functions, data frames, etc.) that are currently present in your R session. Loading the workspace means to restore this saved state, bringing back all the previously saved objects into your current R session.

```
##  [1] "RColorBrewer" "tidyr"        "randomForest" "classInt"     "plotROC"
##  [6] "ggplot2"      "pROC"         "dplyr"        "readr"        "foreign"
## [11] "terra"        "stats"        "graphics"     "grDevices"    "utils"
## [16] "datasets"     "methods"      "base"
```

6.2.2 Surrogate model

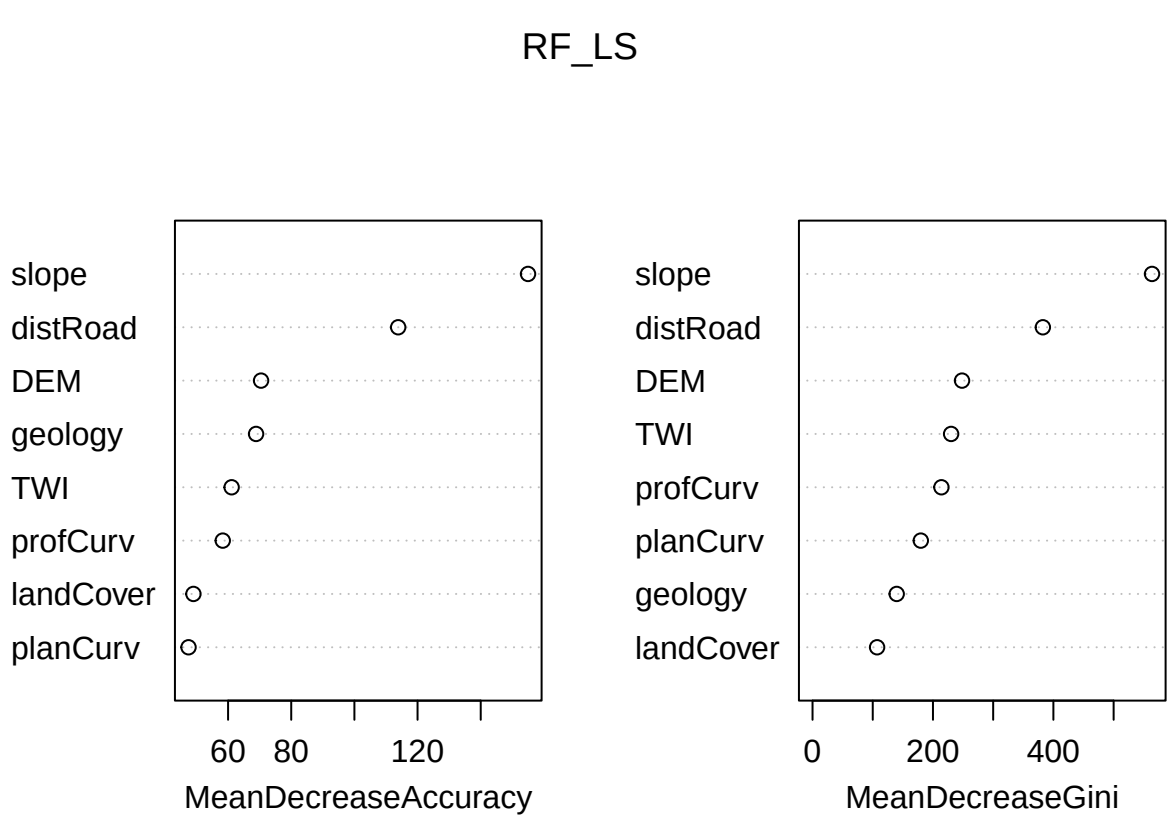
Although machine learning algorithms are often considered as a black box, with RF is possible to plot a sample tree (selected randomly) to analyse its structure and investigate how decisions have been made.

```
library("party")

x <- ctree(LS~., data=LS_train)
plot(x, type="simple")
```

6.2.3 Features importance score

RF provides two scores allowing to assess the importance of each variables in the model: the Mean Decrease in Accuracy (MDA), and the mean decrease in Gini index. The MDA indicates how much the tree nodes that use a given variable enable reducing the mean-square errors evaluated on the Out-Of-Bag and across all the trees. The Gini index measures the probability of incorrectly classifying a randomly chosen element in a dataset if that element were classified based on the distribution of classes in a particular node. The relative importance of the predictor variables can be ranked based on the increasing values of both these scores: the higher the value, the more important the variable.

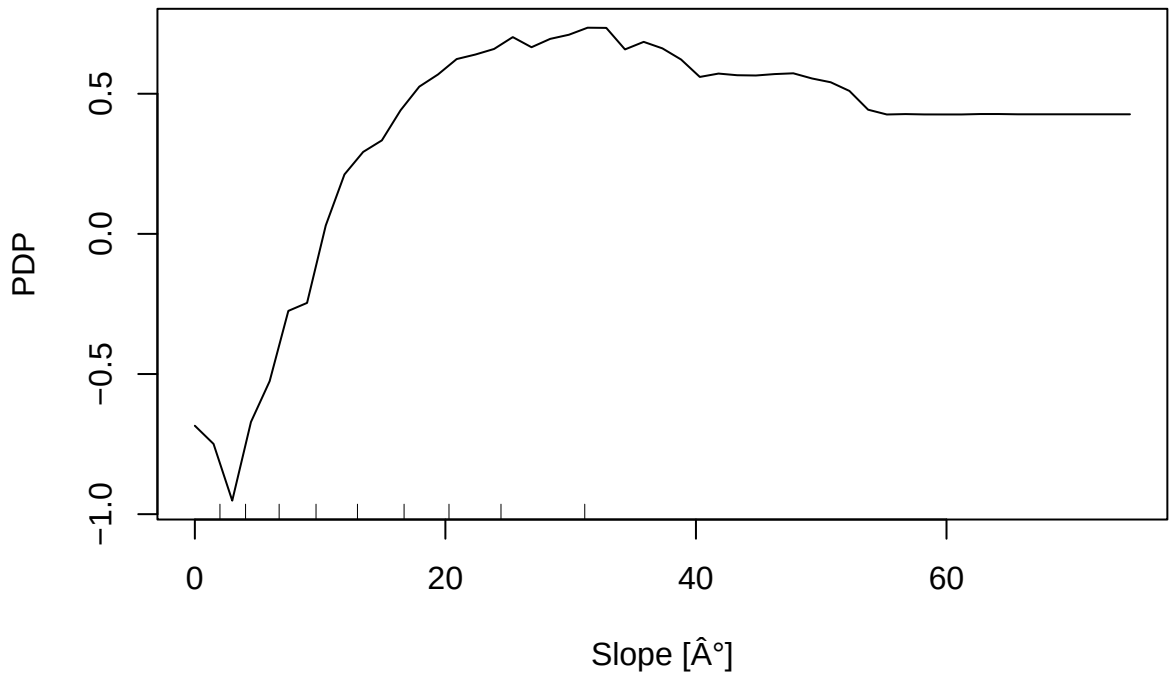
```
# Display the plot with the relative importance of each variable
importance(RF_LS)
varImpPlot(RF_LS)
```



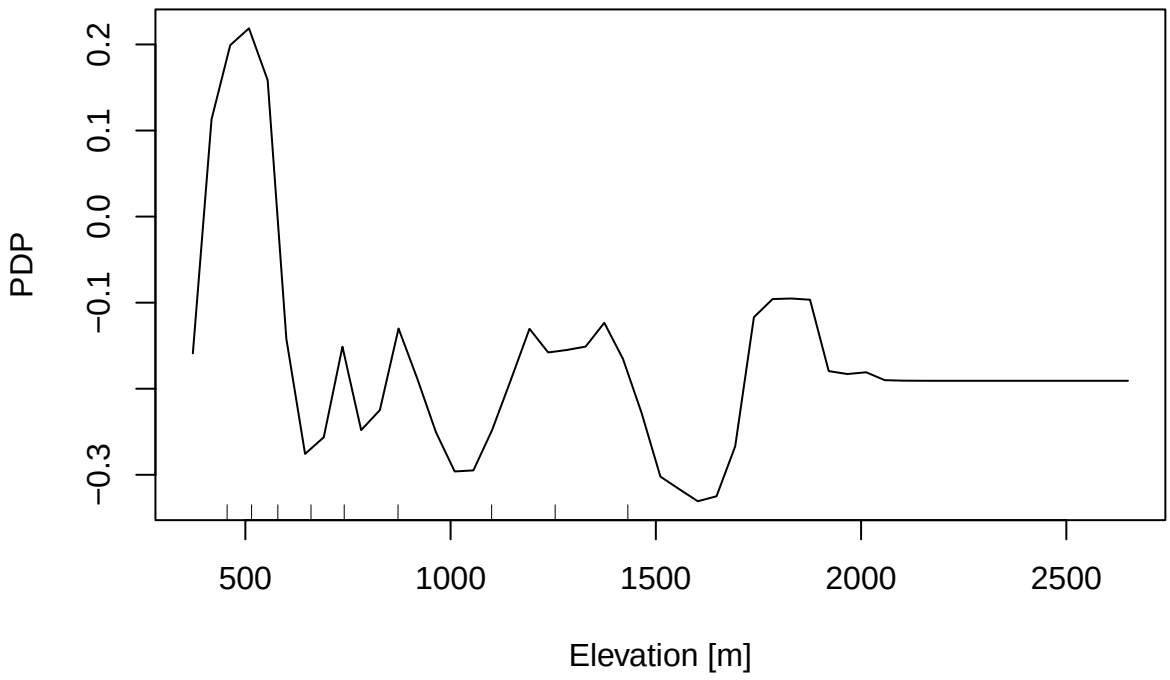
6.2.4 Partial dependence plot

The Partial Dependence Plot (PDP) allows us to estimate, for each single variable, the relative probability of prediction success over different ranges of values. PDP provides a graphical depiction of the marginal effect of each variable on the class probability over different ranges of continuous or discrete values. Positive values are associated with the probability of occurrence of the phenomena (i.e., landslides presence), while negative vales indicate its absence.

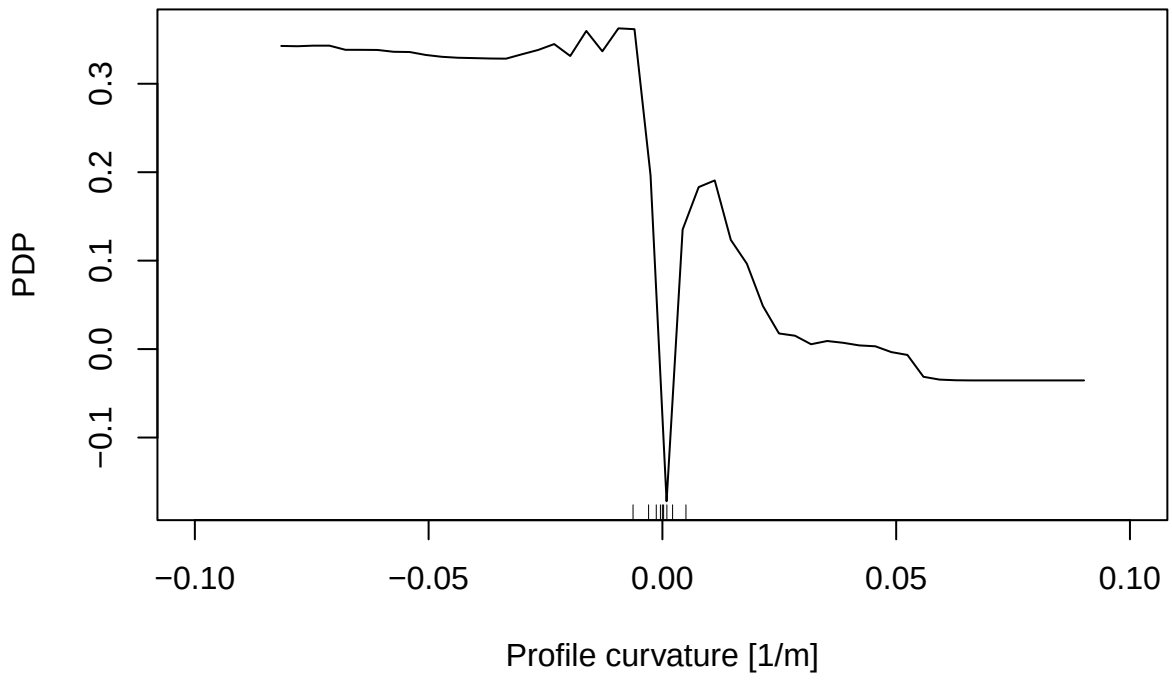
```
#Compute PDP for all the predictor variables

# Slope
partialPlot(RF_LS, LS_train, x.var = slope, rug = TRUE,
            which.class = RF_LS$classes[2],xlab= "Slope [Â°]",
            main = "", ylab = "PDP")
```



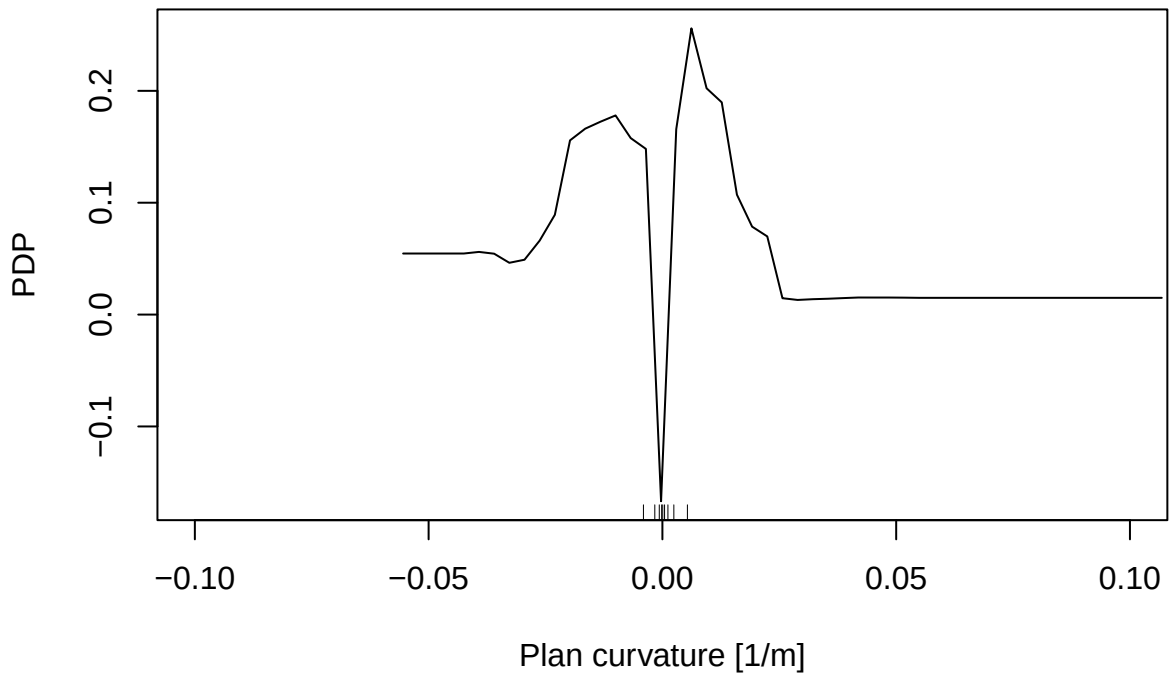

```
# Elevation
partialPlot(RF_LS, LS_train ,x.var = DEM, rug = TRUE,
            which.class = RF_LS$classes[2],xlab= "Elevation [m]",
            main = "",ylab = "PDP")
```



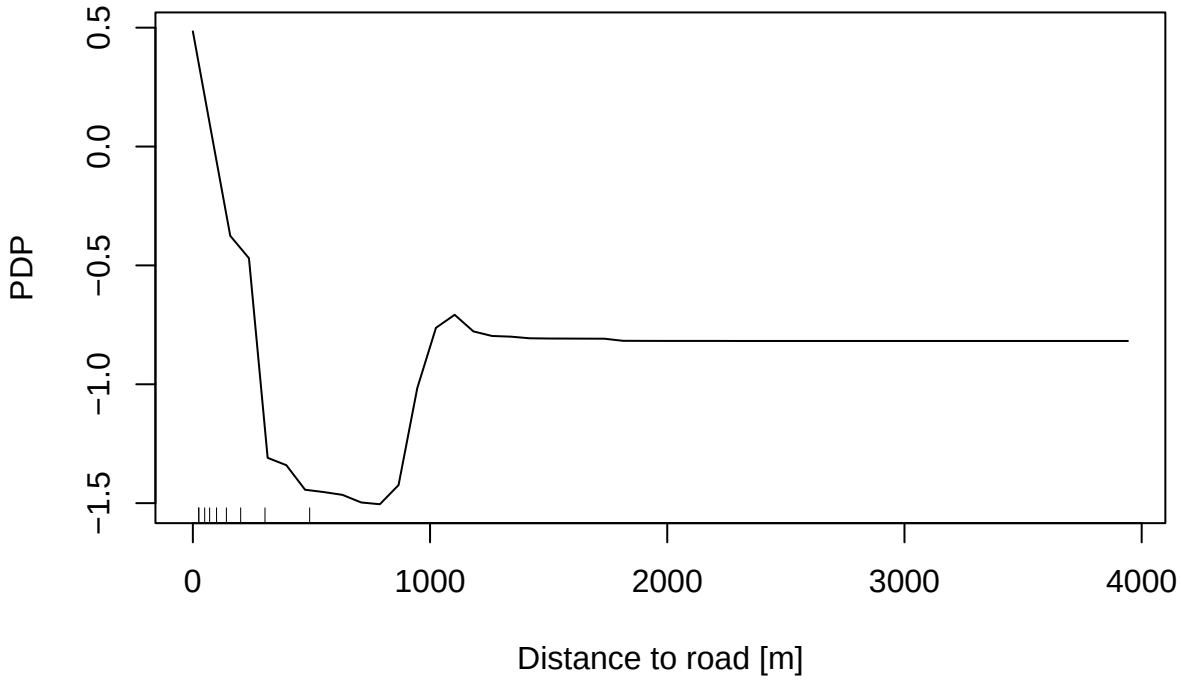

```r
# Profile curvature
partialPlot(RF_LS, LS_train, x.var = profCurv, rug = TRUE,
            which.class = RF_LS$classes[2],xlab= "Profile curvature [1/m]",
            main = "", ylab = "PDP", xlim = c(-0.1,0.1))
```



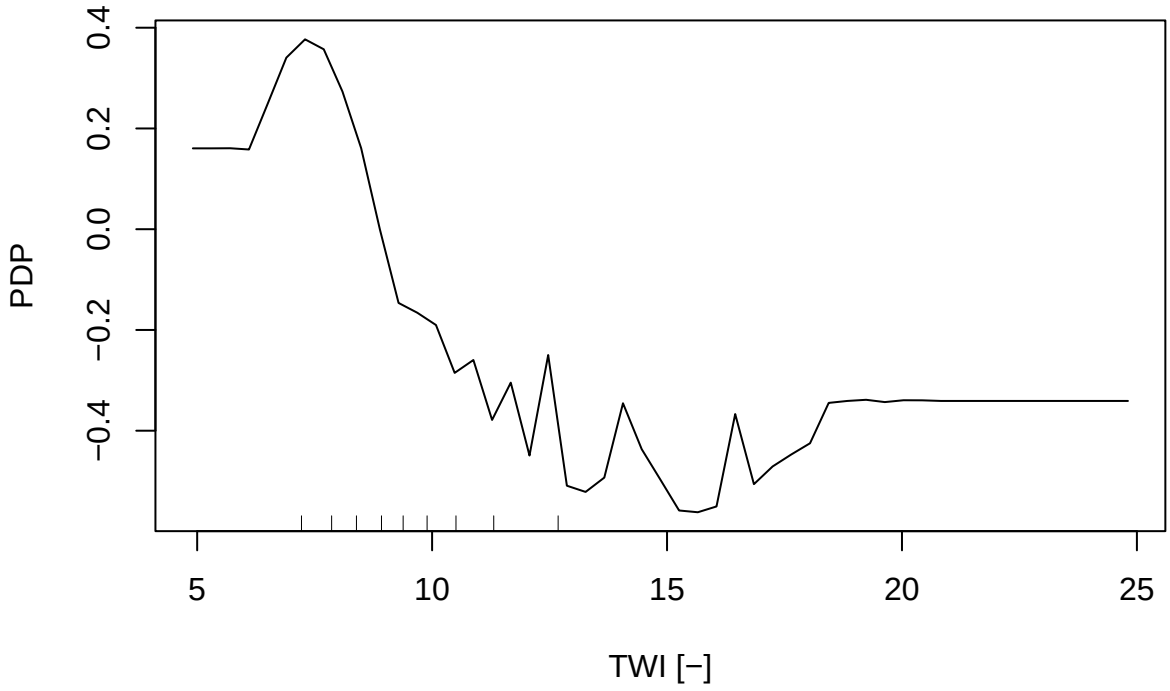

```
# Plan Curvature
partialPlot(RF_LS, LS_train, x.var = planCurv, rug = TRUE,
            which.class = RF_LS$classes[2],xlab= "Plan curvature [1/m]",
            main = "", ylab = "PDP", xlim = c(-0.1,0.1))
```



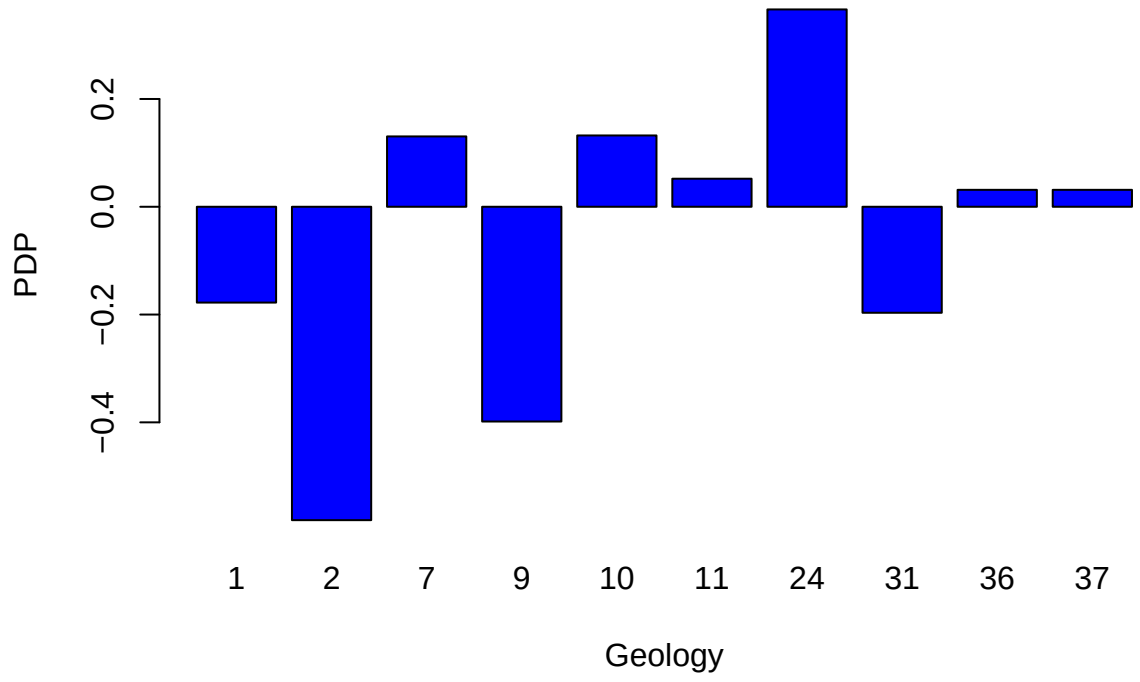

```
# Distance to road
partialPlot(RF_LS, LS_train, x.var = distRoad, rug = TRUE,
            which.class = RF_LS$classes[2],xlab= "Distance to road [m]",
            main = "", ylab = "PDP")
```



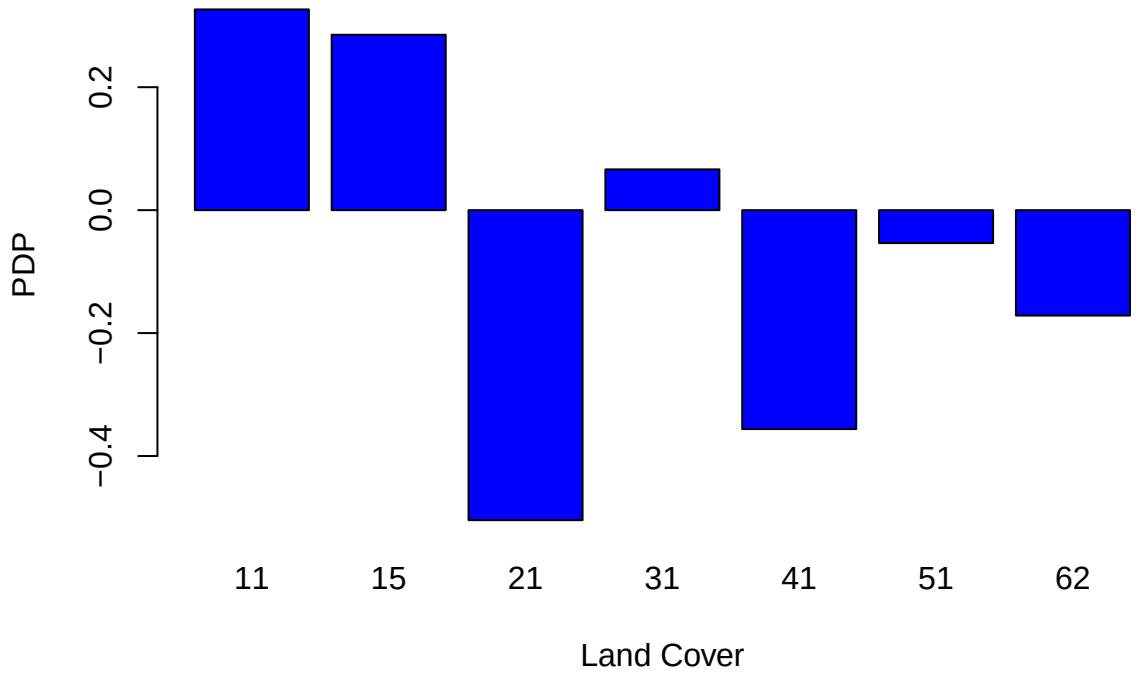

```
# Topographic wetness index
partialPlot(RF_LS, LS_train, x.var = TWI, rug = TRUE,
            which.class = RF_LS$classes[2],xlab= "TWI [-]",
            main = "", ylab = "PDP")
```




```
# Geology
partialPlot(RF_LS, LS_train, x.var = geology, rug = TRUE,
            which.class = RF_LS$classes[2],xlab= "Geology",
            main = "", ylab = "PDP")
```




```
# Land cover
partialPlot(RF_LS, LS_train, x.var = landCover, rug = TRUE,
            which.class = RF_LS$classes[2],xlab= "Land Cover",
            main = "", ylab = "PDP")
```



6.3 Local feature importance

Classical machine learning algorithms like Random Forest lack spatial calibration, hindering capturing the spatial heterogeneity in the relationship between a dependent and a set of independent variables. To account for spatial heterogeneity (i.e. non-stationarity) on the spatial patterns distribution of hazardous events modeled as function of geographical features we can use local models. Specifically, in the present work we explore the **local feature importance** of geographical independent predictor variables on the spatial distribution of landslides in canton Vaud (Switzerland).

We introduce **Geographical Random Forest** (GRF), a spatial analysis method which uses a local version of RF algorithm (**?**) . This is achieved by fitting a sub-model for each observation in space, taking into account the neighboring observations. The GRF can model the non-stationarity coupled with a non-linear model (RF), which, compared to liner models, tends not to overfit due to its bootstrapping nature. In addition RF is suited for datasets with numerous predictor variables.

Essentially, GRF was designed to be a bridge between machine learning and geographical models, combining inferential and explanatory power.

6.4 Computing lab: Geographical Random Forest

For the computation we introduce here the method proposed by **?** and implemented in the R package `SpatialML` (**?**)

The function `grf` fitting a local version of the RF algorithm, has been implemented fro regression problem, so we need to transform our binary response variable (presence==1 / absence==0) to a numeric value which can assume a range of values from zero to one.

```
## 'data.frame':    5173 obs. of  11 variables:
##  $ distRoad : num  176.8 70.7 35.4 35.4 90.1 ...
##  $ DEM      : num  442 450 453 453 462 ...
##  $ landCover: Factor w/ 7 levels "11","15","21",..: 3 1 1 1 3 1 3 3 3 3 ...
##  $ TWI      : num  8.28 8.11 8.3 8.3 8.58 ...
##  $ planCurv : num  0.000814 -0.005182 -0.001086 -0.001086 -0.000253 ...
##  $ profCurv : num  0.01217 0.0041 -0.00189 -0.00189 -0.00121 ...
##  $ slope    : num  17.6 5.38 8.8 8.8 3.36 ...
##  $ geology  : Factor w/ 10 levels "1","2","7","9",..: 7 7 7 7 7 7 7 7 7 7 ...
##  $ x        : num  567850 568050 567950 567950 567850 ...
##  $ y        : num  2e+05 2e+05 2e+05 2e+05 2e+05 ...
##  $ LS       : Factor w/ 2 levels "0","1": 2 2 2 2 2 2 2 2 2 2 ...
##  - attr(*, "na.action")= 'omit' Named int [1:15] 2598 2771 2939 2985 3237 3409 3456 3615 4058 4216 .
##   ..- attr(*, "names")= chr [1:15] "2598" "2771" "2939" "2985" ...

## 'data.frame':    4138 obs. of  12 variables:
##  $ distRoad : num  25 35.4 50 0 50 ...
##  $ DEM      : num  566 549 1151 1058 579 ...
##  $ landCover: Factor w/ 7 levels "11","15","21",..: 4 2 3 5 2 3 5 2 5 5 ...
##  $ TWI      : num  8.09 8.42 9.45 16.44 8.83 ...
##  $ planCurv : num  0.017138 0.001319 0.000871 -0.009321 -0.001597 ...
##  $ profCurv : num  -1.12e-02 3.92e-05 -3.13e-03 1.52e-02 4.64e-03 ...
##  $ slope    : num  5.49 15.33 13.87 5.05 10.35 ...
##  $ geology  : Factor w/ 10 levels "1","2","7","9",..: 1 1 1 2 1 1 1 1 5 1 ...
##  $ x        : num  558650 536550 560850 515450 535550 ...
##  $ y        : num  145400 154500 148400 168900 156800 ...
##  $ LS       : Factor w/ 2 levels "0","1": 2 2 2 1 1 2 1 2 1 2 ...
##  $ LSregr   : num  1 1 1 0 0 1 0 1 0 1 ...
##  - attr(*, "na.action")= 'omit' Named int [1:15] 2598 2771 2939 2985 3237 3409 3456 3615 4058 4216 .
##   ..- attr(*, "names")= chr [1:15] "2598" "2771" "2939" "2985" ...

## 'data.frame':    1035 obs. of  12 variables:
##  $ distRoad : num  79.1 111.8 35.4 35.4 25 ...
##  $ DEM      : num  484 1060 847 556 1327 ...
##  $ landCover: Factor w/ 7 levels "11","15","21",..: 3 5 2 3 3 3 2 5 1 3 ...
##  $ TWI      : num  10.9 11.2 10.1 10.5 10.2 ...
##  $ planCurv : num  -0.002085 -0.000343 -0.000788 0.000117 -0.001574 ...
##  $ profCurv : num  -0.001125 0.000457 0.001293 -0.000683 -0.002214 ...
##  $ slope    : num  14.39 1.97 15.91 4.74 20.62 ...
##  $ geology  : Factor w/ 10 levels "1","2","7","9",..: 7 1 7 1 1 1 1 2 7 1 ...
##  $ x        : num  529950 518550 552950 543550 572150 ...
##  $ y        : num  160700 165100 148200 174100 136000 ...
##  $ LS       : Factor w/ 2 levels "0","1": 2 1 2 1 2 1 2 1 2 1 ...
##  $ LSregr   : num  1 0 1 0 1 0 1 0 1 0 ...
##  - attr(*, "na.action")= 'omit' Named int [1:15] 2598 2771 2939 2985 3237 3409 3456 3615 4058 4216 .
##   ..- attr(*, "names")= chr [1:15] "2598" "2771" "2939" "2985" ...
```

```
Coords<-LS_train[,9:10] # define coordinates

# Run GRF

set.seed(123) # initialize

gwRF_LS<-grf(LSregr~distRoad+DEM+landCover+TWI+planCurv+profCurv+slope+geology,  LS_train, bw=40, mtry=
```

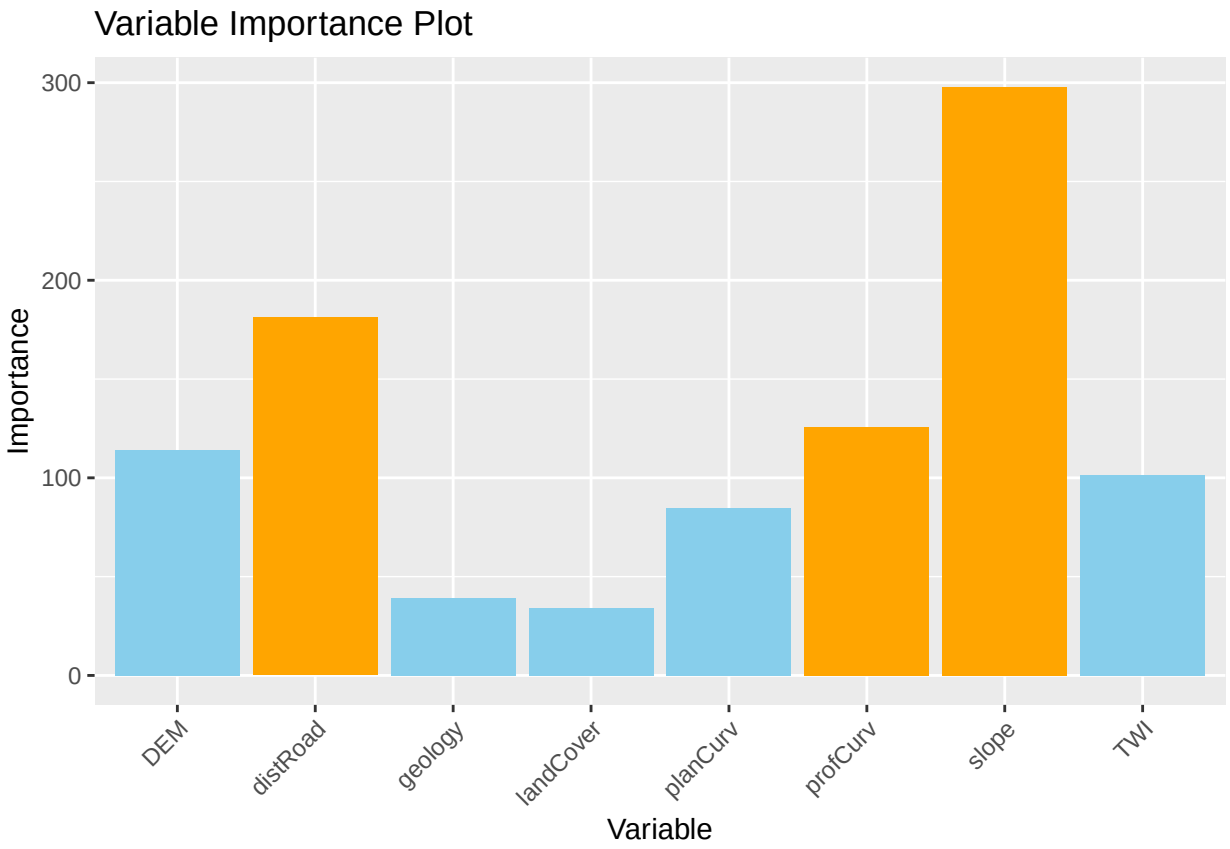
```
## Ranger result
##
## Call:
##  ranger(LSregr ~ distRoad + DEM + landCover + TWI + planCurv +      profCurv + slope + geology, data
##
## Type:                             Regression
## Number of trees:                  500
## Sample size:                      4138
## Number of independent variables:  8
## Mtry:                             3
## Target node size:                 5
## Variable importance mode:         impurity
## Splitrule:                        variance
## OOB prediction error (MSE):       0.1179855
## R squared (OOB):                  0.5281665
##  distRoad       DEM landCover       TWI  planCurv  profCurv     slope   geology
## 189.03796 118.23180  42.58344 112.62676  86.64094 103.52100 275.56255  45.92905
##     Min.  1st Qu.   Median     Mean  3rd Qu.     Max.
## -1.00000 -0.03989  0.00000 -0.02352  0.01996  1.00000
##       Min.    1st Qu.     Median       Mean    3rd Qu.       Max.
## -0.5771719 -0.0038917  0.0000000 -0.0004654  0.0030000  0.4914078
##           Min      Max      Mean       StD
## distRoad    0 5.057082 0.6840634 0.8651427
## DEM         0 5.097790 0.8308835 0.8286699
## landCover   0 3.263085 0.2288518 0.3818137
## TWI         0 4.108891 0.6034478 0.6340272
## planCurv    0 3.295054 0.4977787 0.4819980
## profCurv    0 4.177866 0.4866789 0.5209168
## slope       0 4.749768 0.8323274 0.8537186
## geology     0 4.923056 0.2181370 0.3916300
```

```
saveRDS(gwRF_LS, "gwRF_LS.rds")
```

6.4.1 Global variable importance plot

Based on the results of the GRF, we can plot of the variable importance ranking for illustrative purposes. Values came from "`Global ML Model Summary`" –> "`Importance`"

```
# Create a data frame with variable names and importance values
variable_importance <- data.frame (
  Variable = c("distRoad", "DEM", "landCover", "TWI", "planCurv", "profCurv", "slope", "geology"),
  Importance = c(181.18490, 114.32444,  34.23643, 101.51863,  84.81667, 125.93651, 297.74411,  39.22721
)

# Assign different colors to the top three important variables
variable_importance$Color <- ifelse(variable_importance$Importance >= sort(variable_importance$Importan

# Create a bar plot for variable importance with different colors for the top three variables
ggplot(data = variable_importance, aes(x = Variable, y = Importance, fill = Color)) +
  geom_bar(stat = "identity") +
  scale_fill_identity() +
  labs(title = "Variable Importance Plot", x = "Variable", y = "Importance") +
  theme(axis.text.x = element_text(angle = 45, hjust = 1))  # Rotate x-axis labels for better readabili
```



6.4.2 Local feature importance mapping

We can display the local feature importance scores for the two variables that are globally most important, the slope and the distance to road, with the output values mapped over the geographic space.

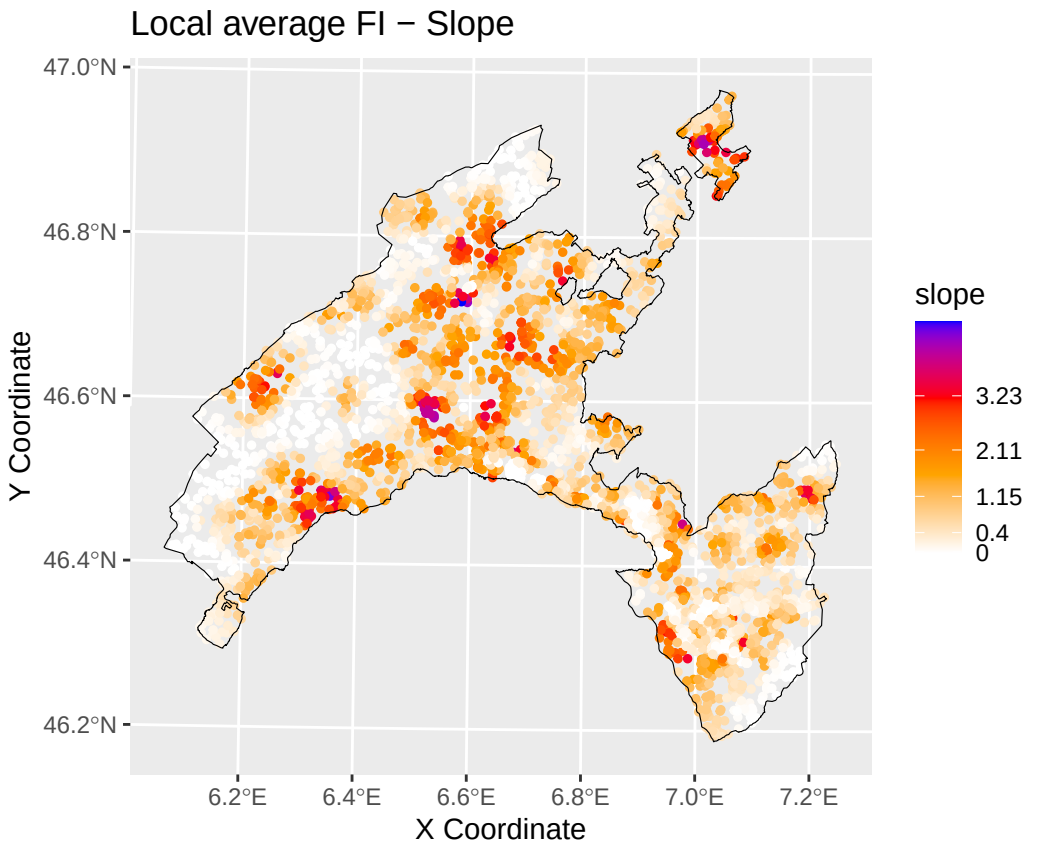
```
# Create a data frame with the values of the local variables importance and the coordinates for each lo
gwRF_LS_var<-gwRF_LS$Local.Variable.Importance
gwRF_LS_var_XY<-cbind(gwRF_LS_var,LS_train$x,LS_train$y ) # add coordinates
colnames(gwRF_LS_var_XY)[9]<- "X" #rename column X-coordinate
colnames(gwRF_LS_var_XY)[10]<- "Y" #rename column Y-coordinate
str(gwRF_LS_var_XY)
```

```
## 'data.frame':    4138 obs. of  10 variables:
##  $ distRoad : num  0.0562 0.271 0.383 0.0165 0.2723 ...
##  $ DEM      : num  0.0427 0.8705 0.9203 0.5591 2.3373 ...
##  $ landCover: num  0.22036 0.04229 0.08014 0.00167 0.43775 ...
##  $ TWI      : num  0.1152 0.6513 0.4731 0.0766 1.3823 ...
##  $ planCurv : num  0.00755 0.59866 1.06332 0.03534 0.66228 ...
##  $ profCurv : num  0.0725 0.2454 0.8325 0.2023 0.8722 ...
##  $ slope    : num  0.00984 1.67939 1.23443 0.02137 1.13352 ...
##  $ geology  : num  0.0107 0.2399 0.3959 0.3438 1.8822 ...
##  $ X        : num  558650 536550 560850 515450 535550 ...
##  $ Y        : num  145400 154500 148400 168900 156800 ...
```

```r
library(sf) #for spatial data operations
# Convert vector to sf (simple feature)
Vaud<-vect("data/RF/Vaud_CH.shp")
Vaud_sf<-st_as_sf(Vaud)

# Output predicted values are transformed to a vector
pred.vect <- as.vector(gwRF_LS_var_XY$slope)

library(classInt) #for classification

brk<-(classIntervals(pred.vect, n=5, style = "fisher"))
brkInt<-round(brk$brks, digits=2)
print(brkInt)
```

```
## [1] 0.00 0.41 0.99 1.61 2.47 4.75
```

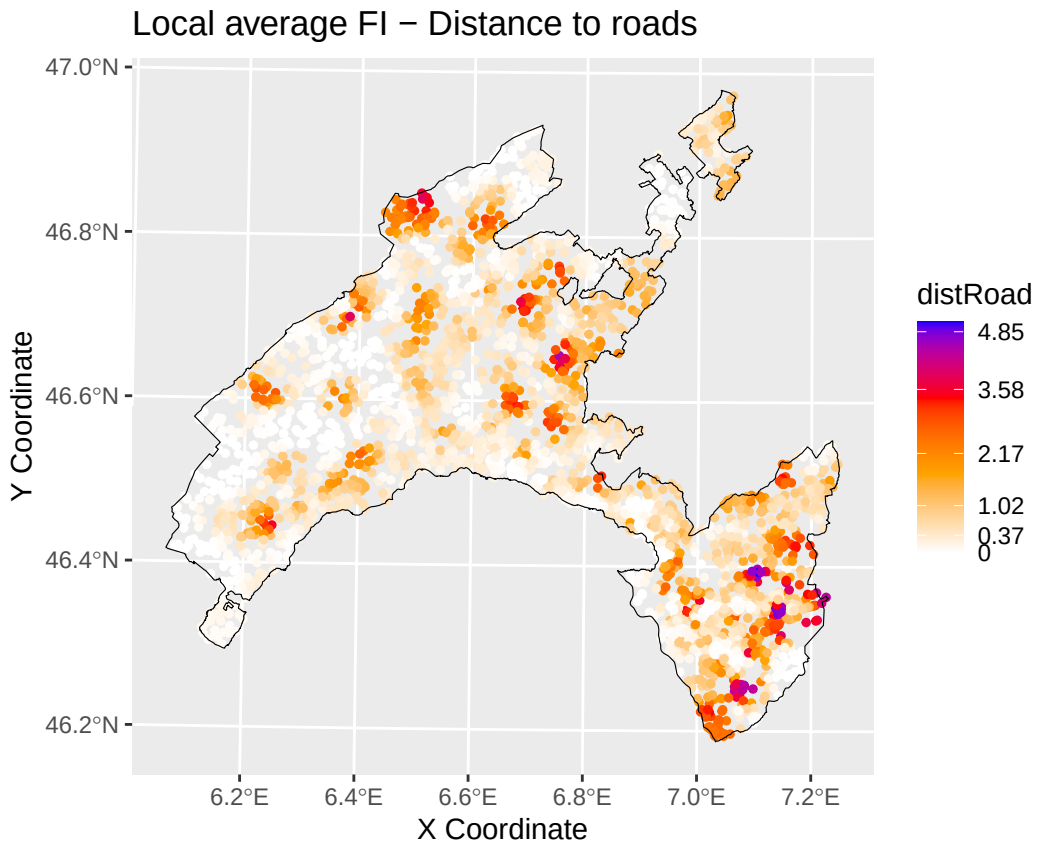
```r
#natural breaks (fisher)

ggplot() +
    geom_point(data = gwRF_LS_var_XY, aes(x = X, y = Y, colour = slope), size = 1) +
    scale_color_gradientn(colors = c("white", "orange","red", "blue"),
                          breaks = c(0.00, 0.40, 1.15, 2.11, 3.23, 4.91),
                          labels=c(0.00, 0.40, 1.15, 2.11, 3.23, 4.91)) +
        labs( x = "X Coordinate", y = "Y Coordinate")+
    ggtitle("Local average FI - Slope")+
    geom_sf(data = Vaud_sf, fill = "transparent", color = "black", size=2) #overlap borders
```




```
# Output predicted values are transformed to a vector
pred.vect <- as.vector(gwRF_LS$Local.Variable.Importance$distRoad)

brk<-(classIntervals(pred.vect, n=5, style = "fisher"))
brkInt<-round(brk$brks, digits=2)
print(brkInt) # print breaks
```

```
## [1] 0.00 0.39 1.06 2.08 3.48 5.06
```

```
#natural breaks (fisher)

ggplot() +
    geom_point(data = gwRF_LS_var_XY, aes(x = X, y = Y, colour = distRoad), size = 1) +
    scale_color_gradientn(colors = c("white", "orange","red", "blue"),
                          breaks = c(0.00, 0.37, 1.02, 2.17, 3.58, 4.85),
                          labels=c(0.00, 0.37, 1.02, 2.17, 3.58, 4.85)) +
    labs( x = "X Coordinate", y = "Y Coordinate")+
    ggtitle("Local average FI - Distance to roads")+
    geom_sf(data = Vaud_sf, fill = "transparent", color = "black", size=2) #overlap borders
```



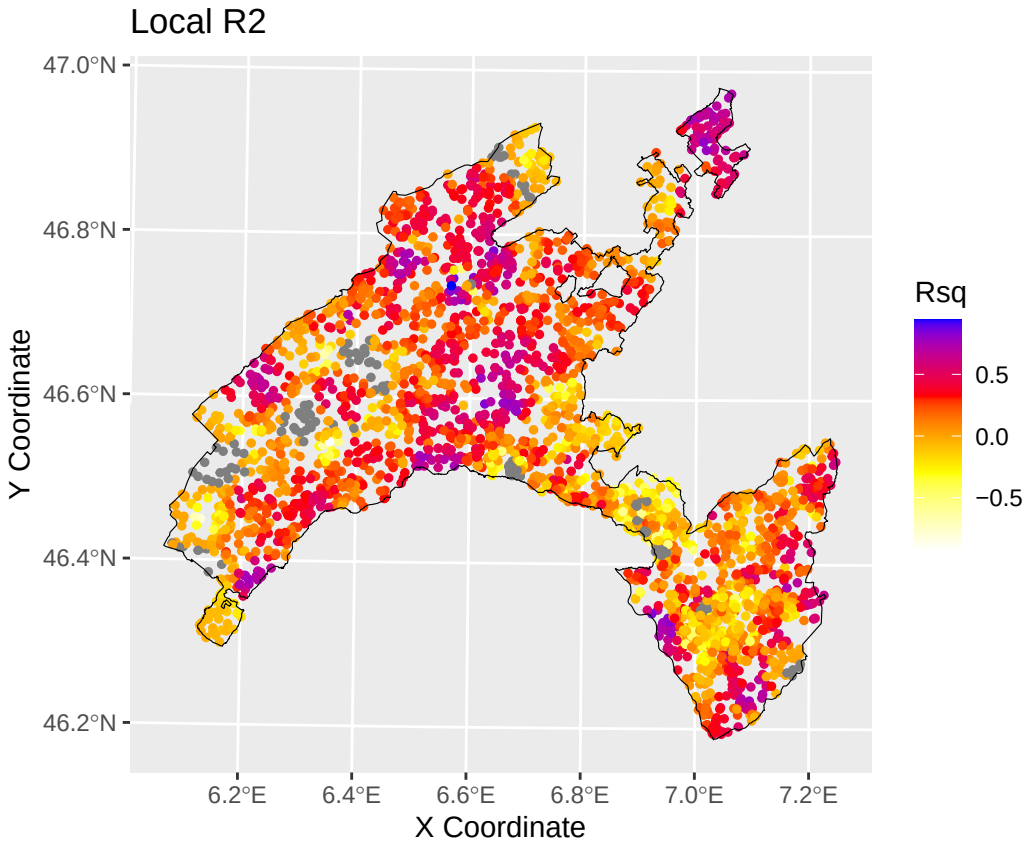
6.4.3 Local R squared

The Local R-squared value represents the strength of the correlations of the local model on the predictor variables and ranges from 0 to 1.

```
Rsq<-gwRF_LS$LGofFit$LM_Rsq100
Rsq_XY<-as.data.frame(cbind(Rsq,LS_train$x,LS_train$y)) # add coordinates
colnames(Rsq_XY)[2]<- "X"
colnames(Rsq_XY)[3]<- "Y"
str(Rsq_XY)
```

```
## 'data.frame':    4138 obs. of  3 variables:
##  $ Rsq: num  -0.258 0.537 -0.144 -0.255 0.322 ...
##  $ X  : num  558650 536550 560850 515450 535550 ...
##  $ Y  : num  145400 154500 148400 168900 156800 ...
```

```
ggplot () +
  geom_point(data = Rsq_XY, aes(x = X, y = Y, colour = Rsq), size = 1)+
  scale_color_gradientn(colors = c("white", "yellow","red", "blue"))+
  labs(title = "Rsq", x = "X Coordinate", y = "Y Coordinate")+
  ggtitle("Local R2")+
  geom_sf(data = Vaud_sf, fill = "transparent", color = "black", size=2) #overlap borders
```



6.5 Conclusions and further analyses

In the present exercise GRF has been used as a purely exploratory tool to estimate the spatial variation of the relationship between landslides in Canton Vaud (Switzerland) and the predictor variables. It allowed to elaborate maps based on the local average importance of the most highly correlated features and to visualize the local fitting performance (R2 local value) into a map.

To ensure that everything is perfectly clear, we propose you to answer the following questions:

1. Among the following algorithms evaluate them in therms of their interpretability and explainability: Support Vector Machines , linear regression, Deep Learning Models, Decision Trees, K-Nearest Neighbors, Neural Networks, Random Forests, logistic regression.
2. Which are the three most important variables of your model (based on the MDA)?
3. What is the slope value (or range of values) that gives the highest probability of landslides occurrence? And for the geology, which are the most important classes?
4. Evaluate the spatial variation of the relationship between landslides and slope / distance to roads in your study area by visually inspecting the local average importance of these features.
5. You can replicate this code (some chiuncks of it) to evaluate the local average importance of the third most important variable, as well as to map the local mean squared error.

7 Defining Territorial Typologies Using Unsupervised Learning: A SOM Approach

Urban expansion across Europe has accelerated in recent decades, significantly blurring the distinctions between urban and rural landscapes. The growing complexity and constant evolution of territorial dynamics require timely urban-rural typologies that systematically classify areas along the continuum from distinctly urban to distinctly rural. Traditional threshold-based classification methods rely heavily on expert input and are labor-intensive, hindering timely updates. This study addresses the issue by developing an unsupervised learning approach based on **Self-Organizing Map (SOM)** (**?**) to group Swiss municipalities, using a diverse set of demographic, socioeconomic, and built environment variables.

7.1 Geo-demographic context in Switzerland

Switzerland has the highest life expectancy in the world. It counts about 8,5 million inhabitants (official census data 2020), twice as much as at the beginning of the 20th century, mainly because of the high level of immigration. The number of foreigners that currently reside in the country corresponds to about one quarter of the total population. Most of the population (85%) lives in cities. Population aging increased over the course of the 20th century, with one in five person of retirement age today.

7.1.1 Swiss census data

The input census data used in this study come from the Swiss Federal Statistical Office and have been downloaded from the Statistical Atlas of Switzerland (**?**). Following an in-depth analysis of all data available in the Atlas, eighteen variables from the 2020 census were selected. These variables provide a comprehensive overview of the physical environment (including settlement, forest, agricultural, unproductive, and traffic areas), demographic characteristics (such as natural growth, population density, international and internal migration, proportion of foreigners, youth, and senior population), and the socio-economic profile (covering employment, new buildings, new houses, individual housing, new enterprises, and net income) of the resident population. This information has been aggregated to the municipality level for the purpose of the present investigation.

7.2 Self Organizing Map

We use SOM, an unsupervised competitive learning neural network allowing representing a high-dimensional feature space (defined by the multivariate input dataset), as two-dimensional discretized pattern (the SOM grid of neurons). In SOM the proximity of the units in the output grid reflects the similarity of the corresponding input observations, which guarantees the preservation of the topological structure of the input data.

Compared to other approaches, SOM is very efficient for data visualization. Indeed it provides additional outputs such as the heatmaps, representing the distribution of each input feature across the SOM grid, extremely useful to visually explore the relationship between the input variables.

7.3 Computing lab: SOM

7.3.1 Load the libraries

To perform the analyses and visualize the results you have to load the following libraries:

- *kohonen*: Supervised and Unsupervised Self-Organizing Maps (SOM)

- *aweSOM*: offers a set of tools to explore and analyze datasets with SOM
- *ggplot2*: create Elegant Data Visualizations Using the Grammar of Graphics
- *colorpatch*: rendering methods (ggplot extensions)

```
library(kohonen)
library(aweSOM)
library(ggplot2)
library(colorpatch)
(.packages())
```

7.3.2 Import the Swiss census dataset

Fist, you have to import the Swiss census dataset for the year 2020, referred to the municipality administrative units. As you can see from the description of the selected variables **??**, some of them can be discarded from the analysis. So, in the following step, we extract a subset of the most meaningful variables for the purpose of the present study.

```
knitr::include_graphics(c("images/Variables1.jpg", "images/Variables2.jpg"))
```

7.3.2.1 Inspect the data Histograms can be used to show the frequency distribution of the variables and help detect patterns, such as skewness, outliers, and the range of values. Similarly, box plots summarize data using quartiles, helping to identify central tendency, spread, and outliers. Together, these visual tools allow to asses if variables have comparable ranges and help determine if data transformation is needed.

```
# Plot variables' frequency distribution
for(i in 3:ncol(subset2020)) {
    hist((subset2020[ , i]),  main=colnames(subset2020[i]))
}
```

```
# Plot variables' frequency distribution
for(i in 3:ncol(subset2020)) {
    boxplot((subset2020[ , i]),  main=colnames(subset2020[i]))
}
```

7.3.2.2 Data transformation To make the variables range comparable, we propose to operate the max-min normalization scaling input census data in the range $[0 - 1]$. Computationally it subtracts the minimum value from the original one, and then it divides the result by the range (i.e., difference between the maximum and minimum values). This process ensures that all variables in the dataset are in the same range, and can be treated and evaluated together.

However, the min-max transformation is sensitive to outliers, as extreme values can distort the interpretation by compressing the range. To mitigate this, values beyond the upper and lower 5th percentiles will be capped at these thresholds before operating the max-min normalization.

```
# Define the general function to detect outlier
replace_outliers <- function(df, lower_quantile_value, upper_quantile_value) {
  replaced_df <- df
  for (col in names(df)) {
    lower_quantile <- quantile(df[[col]], probs = lower_quantile_value)
    upper_quantile <- quantile(df[[col]], probs = upper_quantile_value)
```

Type	Description	Name	Pre-processing indications
Physical space	Built up area	p_infrastructure	The surface in hectare between 2013-2018 divided by the surface of the municipality.
	Agricultural area	p_agriculture	
	Forested area	p_forested	
	Non-productive area	p_improductible	
	Transport area	p_transport	
demographic	Density of the population	density	The number of residents in a municipality divided by the surface of this municipality in 2020
	The number of people in a householsd	Size_household	The number of people in a household over the number of housings in 2020
	Ratio of workers	Dependency_ratio	The number people aged between 0 and19 and over 64 divided by the population between 20 and 64
	Ratio of the population under 19	P_pop_19	The number people aged between 0 and 19 divided by the population.
	Ratio of the population over 64	P_pop_65	The number people aged over 64 divided by the population.
	Number of weddings	p_wedding	The number of weddings in the year 2020 divided by the population
	Internal migration	migration_intern	Difference between arrivals from another region of Switzerland and departures to another region of Switzerland, per 1000 inhabitants
	Migration	migration	Difference between arrivals and departures of resident in the municipality, per 1000 inhabitants
	number of acquisitions of Swiss nationality	p_nationality_aq	Number of acquisitions of Swiss nationality registered during a calendar year per 100 holders of a residence and settlement permits (B + C) at the beginning of the year. Including persons whose municipality of residence is unknown

Figure 5: Variables description

	foreigners	p_foreigners	Percentage of foreign residents of the municipality over the population in 2020
	Natural growth	natural_growth	Natural increase (births-deaths) per 1000 inhabitants in 2020
	Individual houses	p_individual_houses	Percentage of individual houses in 2020
Socio-economics	Employment	p_employment	Employment in active enterprises in 2020 over the population
	workers in the primary sector	p_primary_sector	The ratio between the primary sector and the total workers in 2020
	workers in the secondary sector	p_secondary_sector	The ratio between the secondary sector and the total workers in 2020
	workers in the tertiary sector	p_tertiary_setor	The ratio between the tertiary sector and the total workers in 2020
	New enterprise	p_new_entreprise	New enterprise divided by the population
	Social assistance	p_social_assistance	Rate of social assistance in the municipality in 2020
	Average income	net_income_h	The mean of the net income of the inhabitant in 2019
Consumer attractiveness	New housings	p_new_housings	New housing per 1000 inhabitants in 2020
	New buildings	p_new_buildings	New buildings with housing per 1000 inhabitants in 2020
	Cinema seats	p_cinema	The number of cinema seats divided by the population
	Museum	p_museum	The number of museums divided by the population
	Culture institutions	p_culture_institution	The number of culture institutions divided by the population
Governance	Building area	zab_year	Built-up areas divided by the surface in 2012 and 2022
Spatial data	Latitude	lat	The center of the municipality in the latitude axe
	Longitude	long	The center of the municipality in the longitude axe
Weight	The number of people living in the municipality	Population	The number of residents living in the municipality
	The surface of the municipality	Surface_polygone	The surface of the municipality in hectare

Figure 6: Variables description

```
    outlier_indices_lower <- df[[col]] < lower_quantile
    outlier_indices_upper <- df[[col]] > upper_quantile
    replaced_df[[col]][outlier_indices_lower] <- lower_quantile
    replaced_df[[col]][outlier_indices_upper] <- upper_quantile
  }
  return(replaced_df)
}

# Define the percentile level
lower_quantile_value <- 0.05
upper_quantile_value <- 0.95

# Replace outliers in all variables with upper and lower quantile values

df2020_out_repl <- replace_outliers((subset2020[ , -c(1,2)]), lower_quantile_value, upper_quantile_valu
summary (df2020_out_repl)
```

```
##  p_infrastructure    p_forested      p_agriculture    p_improductible
##  Min.   :0.01906   Min.   :0.07797   Min.   :0.1030   Min.   :0.000000
##  1st Qu.:0.06232   1st Qu.:0.20751   1st Qu.:0.3190   1st Qu.:0.003188
##  Median :0.10815   Median :0.31211   Median :0.4694   Median :0.010854
##  Mean   :0.15093   Mean   :0.32368   Mean   :0.4492   Mean   :0.056692
##  3rd Qu.:0.20430   3rd Qu.:0.42620   3rd Qu.:0.5970   3rd Qu.:0.047372
##  Max.   :0.48058   Max.   :0.61755   Max.   :0.7312   Max.   :0.402194
##   p_transport      natural_growth       density        migration_intern
##  Min.   : 0.7136   Min.   :-9.0172   Min.   :  16.42   Min.   :-24.648
##  1st Qu.: 1.9811   1st Qu.:-2.1720   1st Qu.:  82.34   1st Qu.: -7.159
##  Median : 3.2459   Median : 0.8865   Median : 189.46   Median :  2.575
##  Mean   : 4.1497   Mean   : 0.7120   Mean   : 374.19   Mean   :  3.904
##  3rd Qu.: 5.4475   3rd Qu.: 3.9575   3rd Qu.: 480.88   3rd Qu.: 13.534
##  Max.   :11.9374   Max.   : 9.2336   Max.   :1662.06   Max.   : 37.746
##    migration       p_foreigners    p_individual_houses    p_pop_19
##  Min.   :-5.896   Min.   : 4.419   Min.   :32.94       Min.   :14.83
##  1st Qu.: 0.000   1st Qu.: 9.736   1st Qu.:51.46       1st Qu.:18.61
##  Median : 3.389   Median :15.432   Median :61.54       Median :20.53
##  Mean   : 4.071   Mean   :16.915   Mean   :59.55       Mean   :20.44
##  3rd Qu.: 7.250   3rd Qu.:23.487   3rd Qu.:68.93       3rd Qu.:22.38
##  Max.   :17.350   Max.   :35.327   Max.   :78.69       Max.   :25.47
##     p_pop_65      p_employment     p_new_buildings  p_new_housings
##  Min.   :13.22   Min.   :0.07255   Min.   :0.0000   Min.   : 0.0000
##  1st Qu.:16.60   1st Qu.:0.14204   1st Qu.:0.3201   1st Qu.: 0.5315
##  Median :19.16   Median :0.22970   Median :1.2799   Median : 3.0057
##  Mean   :19.51   Mean   :0.28020   Mean   :1.7691   Mean   : 5.0709
##  3rd Qu.:21.95   3rd Qu.:0.36167   3rd Qu.:2.6316   3rd Qu.: 7.2601
##  Max.   :27.60   Max.   :0.75642   Max.   :6.2107   Max.   :21.1283
##  p_new_entreprise     net_income
##  Min.   :0.000000   Min.   :25908
##  1st Qu.:0.000000   1st Qu.:30789
##  Median :0.002931   Median :34960
##  Mean   :0.002806   Mean   :36848
##  3rd Qu.:0.004514   3rd Qu.:40433
##  Max.   :0.007919   Max.   :59032
```

```r
# Define the general function for min-max normalization
minMax <- function(x) {
  (x - min(x)) / (max(x) - min(x))
}

# Apply max-min to the data
dfnorm2020_out_repl<- as.data.frame(lapply((df2020_out_repl), minMax))
summary(dfnorm2020_out_repl)
```

```
##  p_infrastructure    p_forested     p_agriculture    p_improductible
##  Min.   :0.00000   Min.   :0.0000   Min.   :0.0000   Min.   :0.000000
##  1st Qu.:0.09373   1st Qu.:0.2401   1st Qu.:0.3439   1st Qu.:0.007927
##  Median :0.19304   Median :0.4339   Median :0.5833   Median :0.026987
##  Mean   :0.28573   Mean   :0.4554   Mean   :0.5510   Mean   :0.140958
##  3rd Qu.:0.40137   3rd Qu.:0.6454   3rd Qu.:0.7863   3rd Qu.:0.117785
##  Max.   :1.00000   Max.   :1.0000   Max.   :1.0000   Max.   :1.000000
##   p_transport     natural_growth      density        migration_intern
##  Min.   :0.0000   Min.   :0.0000   Min.   :0.00000   Min.   :0.0000
##  1st Qu.:0.1129   1st Qu.:0.3751   1st Qu.:0.04005   1st Qu.:0.2803
##  Median :0.2256   Median :0.5426   Median :0.10515   Median :0.4363
##  Mean   :0.3061   Mean   :0.5331   Mean   :0.21740   Mean   :0.4576
##  3rd Qu.:0.4218   3rd Qu.:0.7109   3rd Qu.:0.28224   3rd Qu.:0.6119
##  Max.   :1.0000   Max.   :1.0000   Max.   :1.00000   Max.   :1.0000
##    migration       p_foreigners    p_individual_houses    p_pop_19
##  Min.   :0.0000   Min.   :0.0000   Min.   :0.0000      Min.   :0.0000
##  1st Qu.:0.2536   1st Qu.:0.1720   1st Qu.:0.4048      1st Qu.:0.3560
##  Median :0.3994   Median :0.3563   Median :0.6251      Median :0.5357
##  Mean   :0.4288   Mean   :0.4043   Mean   :0.5816      Mean   :0.5273
##  3rd Qu.:0.5655   3rd Qu.:0.6169   3rd Qu.:0.7866      3rd Qu.:0.7097
##  Max.   :1.0000   Max.   :1.0000   Max.   :1.0000      Max.   :1.0000
##     p_pop_65       p_employment    p_new_buildings   p_new_housings
##  Min.   :0.0000   Min.   :0.0000   Min.   :0.00000   Min.   :0.00000
##  1st Qu.:0.2344   1st Qu.:0.1016   1st Qu.:0.05153   1st Qu.:0.02516
##  Median :0.4130   Median :0.2298   Median :0.20607   Median :0.14226
##  Mean   :0.4373   Mean   :0.3036   Mean   :0.28485   Mean   :0.24001
##  3rd Qu.:0.6069   3rd Qu.:0.4228   3rd Qu.:0.42372   3rd Qu.:0.34362
##  Max.   :1.0000   Max.   :1.0000   Max.   :1.00000   Max.   :1.00000
##  p_new_entreprise   net_income
##  Min.   :0.0000   Min.   :0.0000
##  1st Qu.:0.0000   1st Qu.:0.1474
##  Median :0.3702   Median :0.2733
##  Mean   :0.3543   Mean   :0.3303
##  3rd Qu.:0.5701   3rd Qu.:0.4385
##  Max.   :1.0000   Max.   :1.0000
```

7.3.3 Run SOM

The main idea with SOM is to map the input high-dimensional feature space in a lower-dimensional output space organized on a grid made up of regular units (i.e. the neurons). At the end of the process, each observation from the input space ($X_{k,i}$) will be associated (i.e., mapped) to a unit in the SOM grid. Depending on the size of the grid, one unit can include several input observations.

The first step consists in defining the size and geometry of the SOM grid. In this case we define a rectangular

grid of 18 by 13 units, allowing to allocate, on average, about 15 input-items per units, and whose geometry reproduces the shape of the study area. (NB: several parameters and configurations of the SOM grid can be implemented and compared, seeking to minimize the *Quantization Error, QE*). Finally, you can run the SOM model.

For the computation we introduce here the method proposed by (**?**) and implemented in the R package `kohonen` (**?**).

####Create the SOM-grid

Before running SOM, you have to transform the data frame to the format matrix and create the grid of output units.

```
mx2020<-as.matrix(dfnorm2020_out_repl) # max-min
```

```
# Gird size 18x13 units
som_grid <- somgrid(xdim = 20, ydim=15)
```

The general R function `set.seed` is used for creating simulations of random objects that can be reproduced. The `rlen` indicates the number of times the complete data set will be presented to the network, while for the other parameters we will keep the default values.

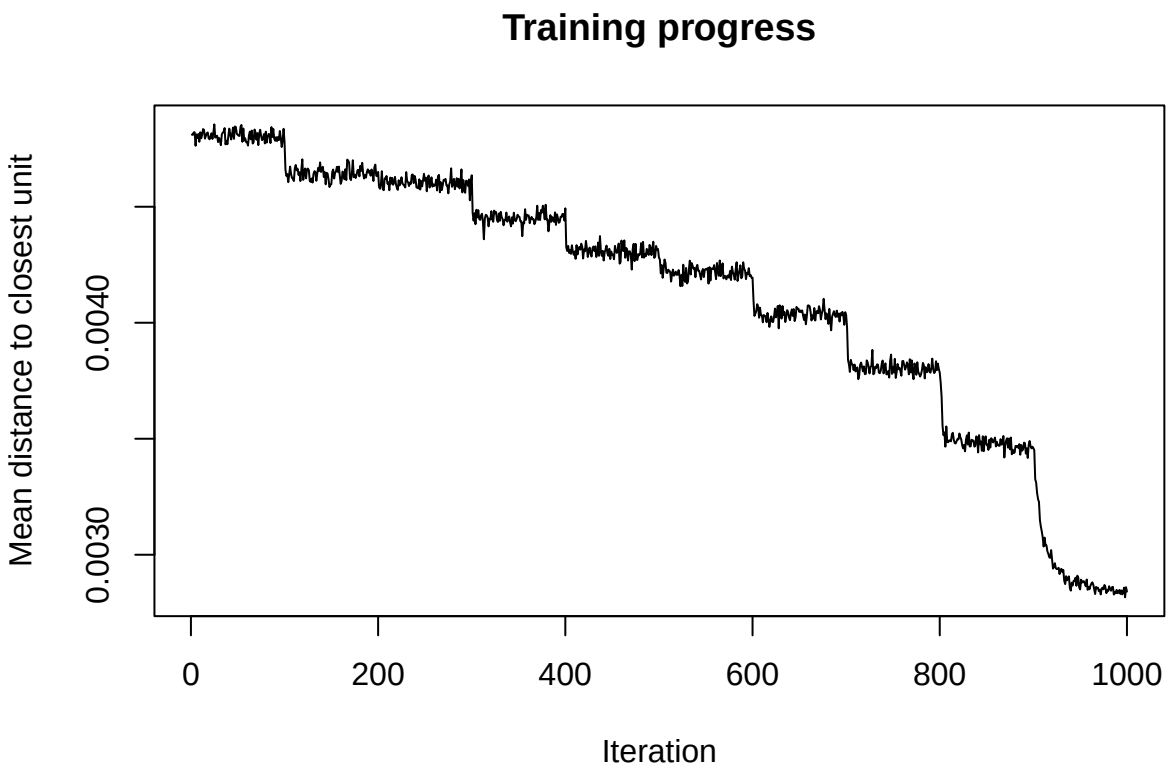
```
# Use max-min data transformation
set.seed(123)

# Run SOM
SOM2020<- som(mx2020,
              grid=som_grid,
              rlen=1000)
print(SOM2020)
```

```
## SOM of size 20x15 with a rectangular topology.
## Training data included.
```

7.3.3.1 Model evaluation Finally, you can optimize the size of the grid by inspecting several quality measure and changing the parameters accordingly. In particular we will explore the following:

- **Changes**: shows the mean distance to the closest codebook vector during the training.
- **Quantization error**: average squared distance between the data points and the map's codebook to which they are mapped. Lower is better.
- **Percentage of explained variance**: similar to other clustering methods, the share of total variance that is explained by the clustering (equal to 1 minus the ratio of quantization error to total variance). Higher is better.

```
# Evaluate rlen
plot(SOM2020, type="changes")
```




```
# Evaluate the results
QEM<-somQuality(SOM2020, dfnorm2020_out_repl)

## Quality measures:
QEM$err.quant # Quantization error
```

```
## [1] 0.311374
```

```
QEM$err.varratio # % explained variance
```

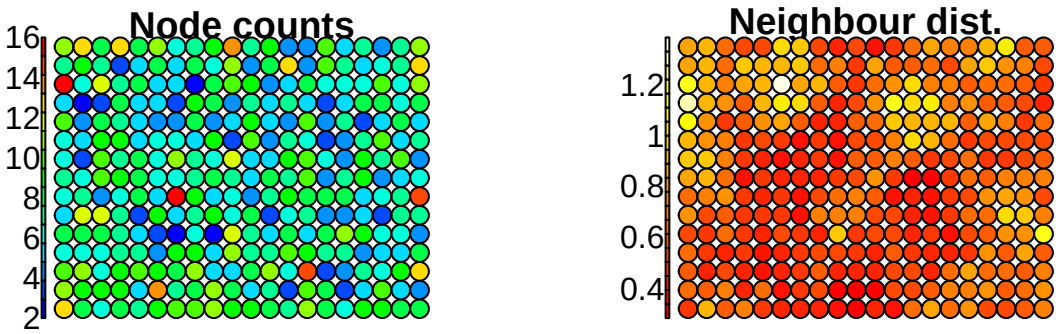
```
## [1] 76.53
```

7.3.3.2 SOM's main outputs The main graphical outputs of SOM are the node counts, the neighborhood distances, and the heatmaps.

- **Node counts map:** informs about the number of input vectors falling inside each output unit.
- **Neighbourhood distance plot:** shows the distance between each unit and its neighborhoods.
- **Heatmaps:** dispaly the distribution of each input variable, associated to each input vectors, across the SOM grid.

```
# Create a color palette
coolBlueHotRed <- function(n, alpha = 1) {rainbow(n, end=4/6, alpha=alpha)[n:1]}
```

```
#display two plots side-by-side
par(mfrow = c(1, 2))

# Plot node counts map
plot(SOM2020, type="count", main="Node counts", palette.name=coolBlueHotRed)

# Plot SOM neighbourhood distance
plot(SOM2020, type="dist.neighbours", main = "Neighbour dist.")
```



Side-by-side heatmaps provide a visual comparison, enabling researchers to identify relationships within the high-dimensional input space **??**. This approach helps assess whether the input variables show similar patterns or complementary trends.

```
# Plot heatmaps for selected variables
for (i in 1:18)
  {
plot(SOM2020, type = "property", property = getCodes(SOM2020)[,i],
     main=colnames(getCodes(SOM2020))[i], palette.name=coolBlueHotRed)
  }
```

```
knitr::include_graphics("images/Heatmaps.jpg")
```

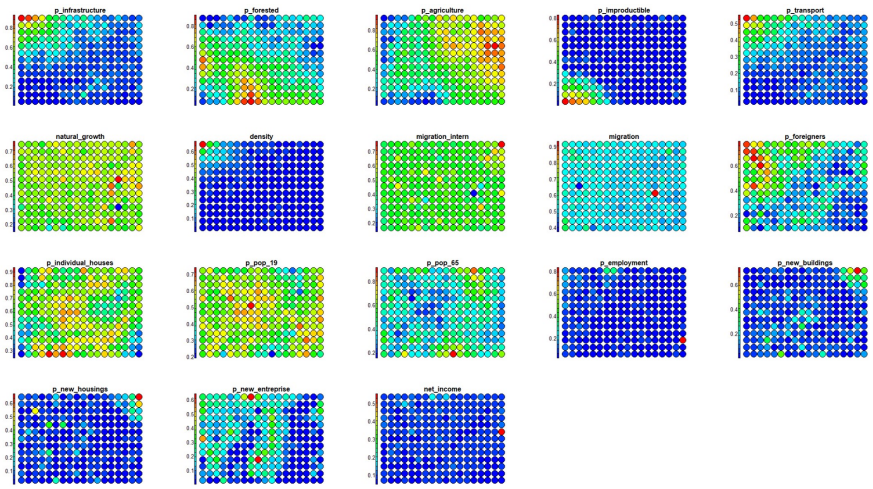


Figure 7: Heatmaps

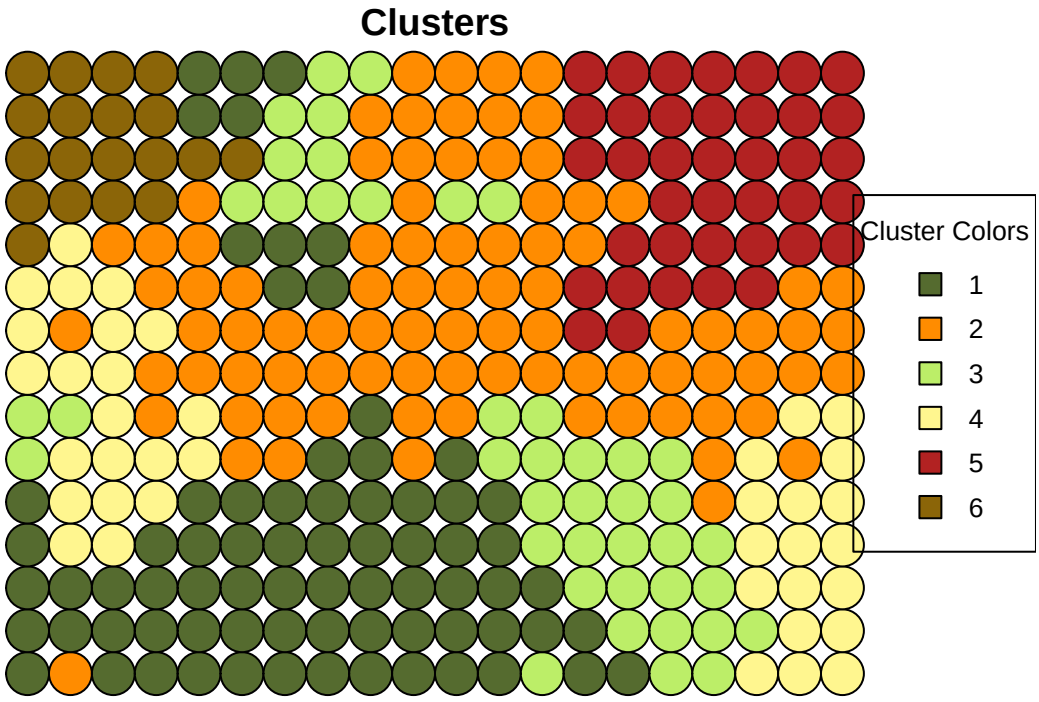
7.4 Hierarchical clustering

The SOM units were ultimately grouped into a smaller set of primary clusters, representing the territorial typologies in Switzerland based on the 2020 census data. Hierarchical clustering was applied to generate these final partitions: SOM units are iteratively merged based on Euclidean distance until the desired number of main clusters (six in our case) is achieved.

```
#Run hierarchical clustering
CV2020 <- getCodes(SOM2020) # Extract codebook vectors

cls2020 <- cutree(hclust(dist(CV2020)), 6)
```

We can display the SOM-grid using different colors for each one of the six final main clusters.

```
# Color palette definition.
# NB: If you change the number of hierarchical clustering, remember to set the number of colors accordi

map_palette <- c("darkolivegreen","darkorange","darkolivegreen2","khaki1","firebrick","darkgoldenrod4")

plot(SOM2020, type="mapping", pchs="", bgcol = map_palette[cls2020], main = "Clusters")
legend("right", legend = unique(cls2020), fill = map_palette, title = "Cluster Colors", cex=0.8)
```




```
#Assign the cluster number (based on hierarchical clustering) to each unit

clasgn <- cls2020[SOM2020$unit.classif]
dfnorm2020_out_repl$hc<-clasgn
clsnorm<-cbind(dfnorm2020_out_repl, subset2020$BFS_nummer)

# Export the table with clusters
write.table(clsnorm, file="hcM_6cls.csv", sep=",")
```

7.4.1 Map clusters onto the geographical space

The six primary clusters can be projected onto the geographical space to elaborate a map that displays the spatial distribution of territorial typologies at the municipal level across the urban-rural continuum in Switzerland. To this end, you can run the code below.

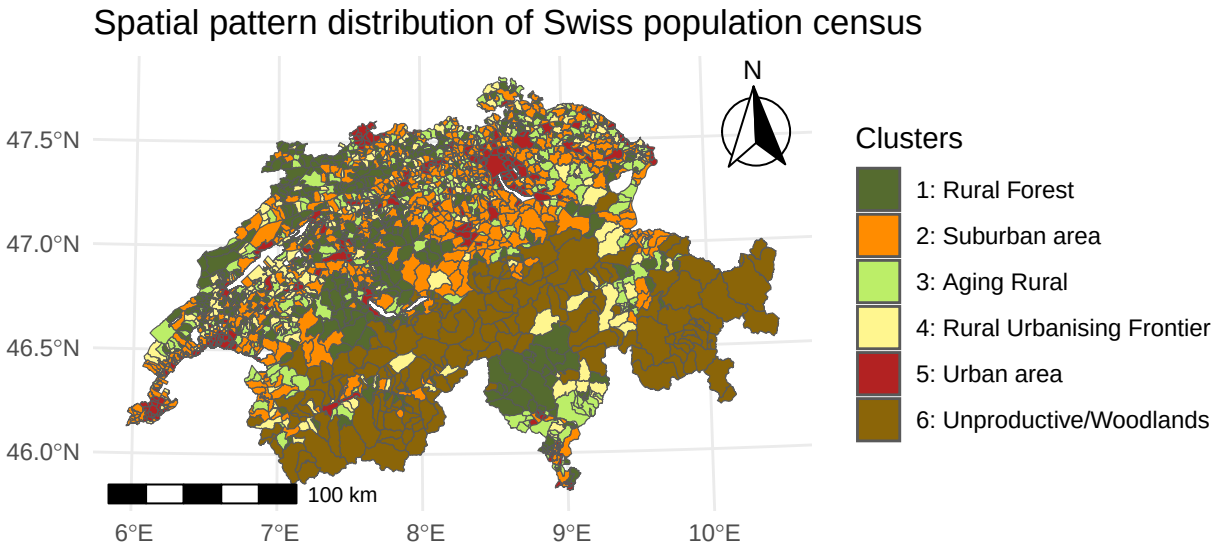
Alternatively , you can import the final table with clusters into a GIS and join it to the values to the administrative limits at municipality level in Switzerland. You need only two columns: the code identifying each municipality ("BFS_nummer") and the cluster number (hc).

```
#Load the following packages needed for visualization
library(st)
library(sf)
library(ggspatial)
library(dplyr)
```

```r
# Load the shapefile data
CH_outline <- st_read("data/SOM/Municiplities.shp")

# Quick check the shapefile data
ggplot() +
  geom_sf(data = CH_outline, size = 1.5, color = "black", fill = "cyan1") +
  ggtitle("Swiss municiplities") +
  coord_sf()
```

```r
# Rename the column
colnames(clsnorm)[20] <- "BFS_NUMMER"

# Merge the dataframe and shapefile together by the column with the same name
z = merge(CH_outline, clsnorm, by = "BFS_NUMMER")

# Rename the column with clustering results
z$cluster <- as.numeric(z$hc)
```

```r
# Mapping the clusters using ggplot (as in a GIS)

CH_cluster <- ggplot(data = z) + # Original data
    geom_sf(aes(fill = as.factor(cluster))) + # Mapping the clusters
    annotation_scale(location = "bl", width_hint = 0.3,
                     pad_x = unit(0.2, "cm"), pad_y = unit(0.1, "cm")) +
# Mapping scales, which are calculated by project coordinates
    annotation_north_arrow(location = "tr", which_north = "true",
          pad_x = unit(0.0, "cm"), pad_y = unit(0.1, "cm"),
          style = north_arrow_fancy_orienteering) + # Mapping north arrow
    scale_fill_manual(values = c("darkolivegreen","darkorange","darkolivegreen2","khaki1","firebrick","
                  name="Clusters",
         labels = c("1: Rural Forest",
                    "2: Suburban area",
                    "3: Aging Rural",
                    "4: Rural Urbanising Frontier",
                    "5: Urban area",
                    "6: Unproductive/Woodlands"))+

    theme_minimal() +
    labs(title = "Spatial pattern distribution of Swiss population census")

# Save the map as jpg
ggsave("CH_typologies.jpg")

CH_cluster
```



7.5 Clusters characterization

To interpret the final main clusters in therms of their geo-demographic characteristics, you can evaluate the distribution of each variable within the clusters by using **box plots** (also known as *whisker plot*).

Box-plot is a standardized way to display a dataset based on the five-number summary statistics: the minimum, the maximum, the sample median, and the first and third quartiles (i.e., the median of the lower half (25%) and the median of the upper half (75%) of the dataset).

```
# Creates an empty list object that will be filled by the loop
cls20M <- list()

# Split the single clusters
for(i in 1:6) {
  cls20M[[i]]<-subset(clsnorm, clsnorm$hc==i)
}

clsvar20M <- lapply(cls20M, "[", c(1:18))

# Box-plots for the single clusters
for (i in 1:6) {boxplot ((clsvar20M[[i]]), main=paste("Cluster", i), mar=c(8,3,3,1), cex.axis=0.5,  las=
```

To better investigate the values assumed by each class of variables withing the different clusters, you can group them by category.

```r
# Box plot by categories: "Physical space"

clsvar20M <- lapply(cls20M, "[", c(1:5))

par(mfrow=c(2,4), mar=c(7,3,3,1), cex.axis=0.7)

for (i in 1:6) {boxplot ((clsvar20M[[i]]), main=paste("Cluster", i),  las=2)}
```

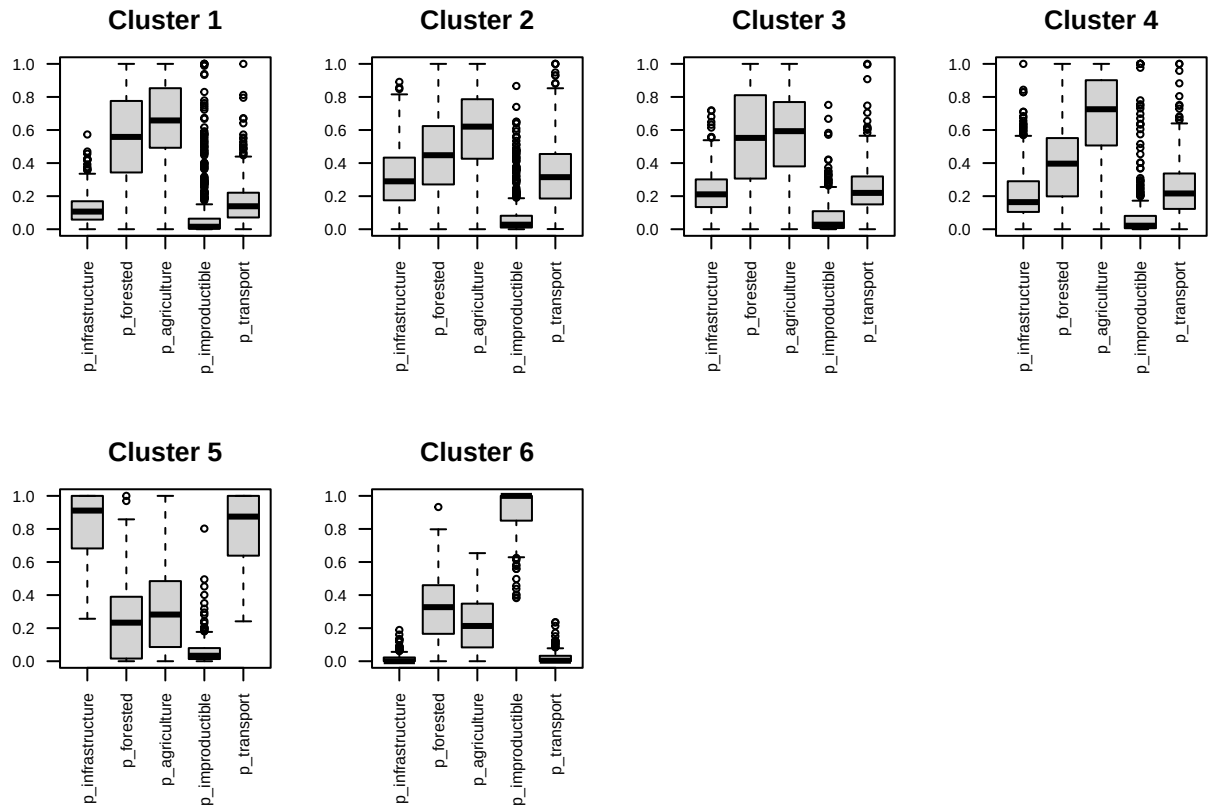


Figure 8: Physical space

```r
# Box plot by categories: "Demographics"

clsvar20M <- lapply(cls20M, "[", c(6:13))

par(mfrow=c(2,4), mar=c(7,3,3,1), cex.axis=0.7)

for (i in 1:6) {boxplot ((clsvar20M[[i]]), main=paste("Cluster", i),  las=2)}
```

```r
# Box plot by categories: "Socio-economics"

clsvar20M <- lapply(cls20M, "[", c(14:18))

par(mfrow=c(2,4), mar=c(7,3,3,1), cex.axis=0.7)
```

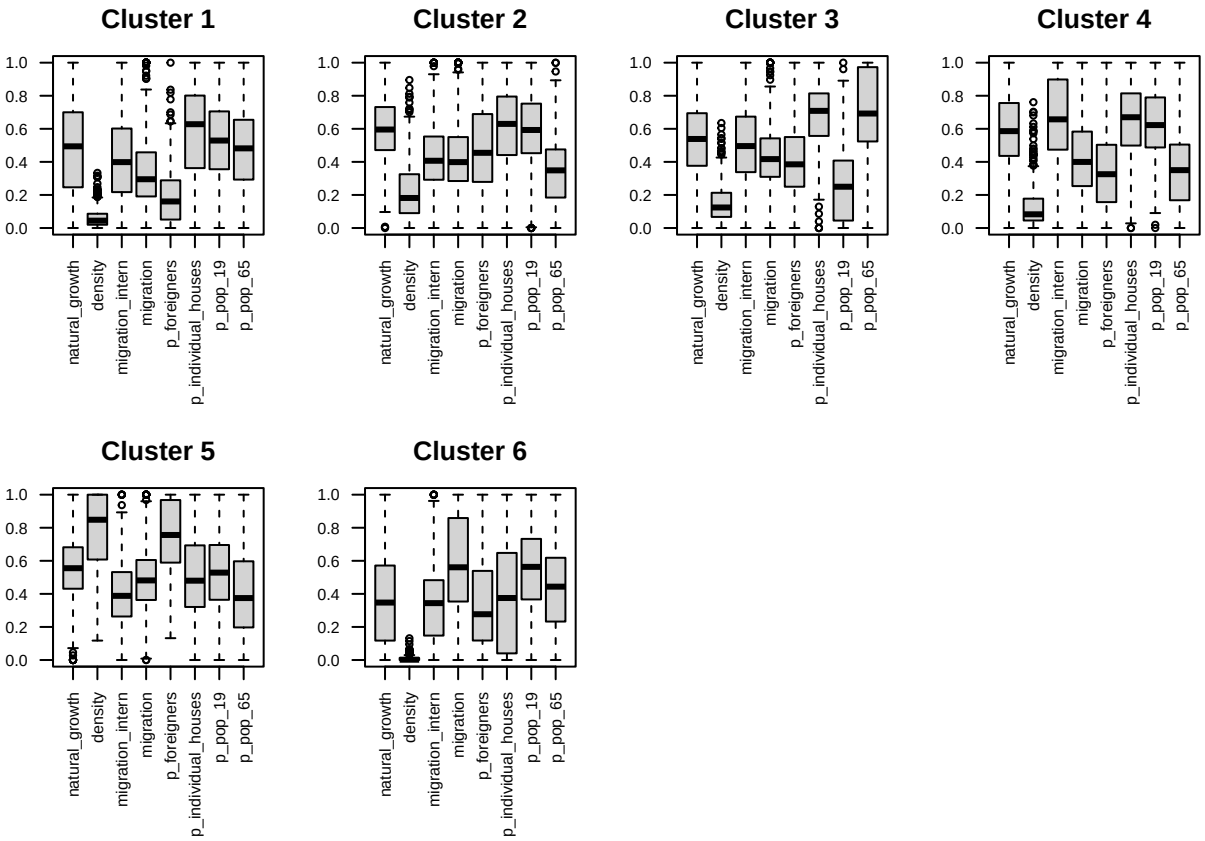


Figure 9: Demographics

```
for (i in 1:6) {boxplot ((clsvar20M[[i]]), main=paste("Cluster", i),  las=2)}
```

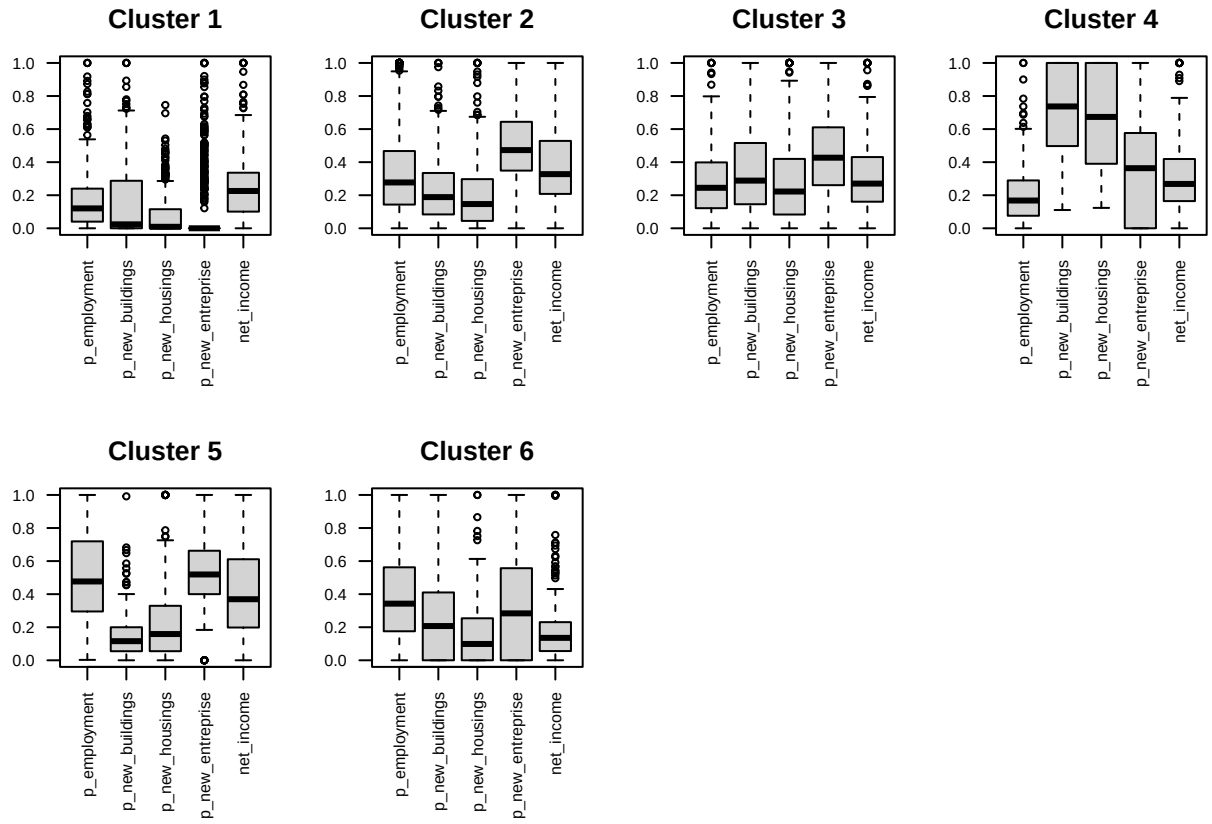


Figure 10: Socio-economics

7.6 Conclusions

Results of the present study reveal the main patterns of the population in Switzerland based on the surveyed land-use, socio-economic, and demographic indicators. To characterize the final main clusters, the distribution of each variable within them have been assessed by using **box plots**. We could thus identify main clusters including the most developed and active cities with higher income, the peri-urban areas mostly devoted to the agricultural activity, or the areas with higher levels of migration.

SOM heatmaps allow to display the pattern distribution of each input variable over the SOM-grid and how values change in space. Visualized side by side, heatmaps show a picture of the different areas and their characteristics. In this way it is possible to explore the level of complementarity that links one or more variables among them.

In conclusion, in the present study we proposed a performant data-driven approach based on unsupervised learning allowing to extract useful information from a huge volume of multivariate population census data. This approach led to represent and interpret the main patterns characterizing the dynamic of population in Switzerland in the recent period.

7.7 Further analyses

To ensure that everything is perfectly clear, we propose you to answer the following questions and to discuss your answers with the other participants to the course or directly with the teacher.

1. Change the size of the gridmap and check if you get better results for SOM. **N.B.** Better results are achieved when the quantisation error decreases, the explained variance increases, and there are no empty observations revealed by the Node Counts map.
2. Focusing on Cluster 6: which variable characterize it the most? Based on these variables, which class of land use can be associated to this cluster? And in the case of Cluster 4?
3. Describe the distribution of the clusters in the geographical space. In more details, describe to which kind of land use the different clusters can be referred and specify why.
4. From the visual inspection of the heatmaps, describe the correlation you can observe between the following variables: a) "p_transport" and "p_infrastructure"; b) "p_agriculture" and "p_improductible".

8 DBSCAN for 3D features detection in geosciences

A terrestrial laser scanner (TLS) is a sophisticated surveying tool designed for capturing highly accurate and dense 3D point cloud data of physical environments. Sequential acquisition are used to detect and quantify surface changes. Neverthless, three main challenges arise from TLS data collection:

1. The large number of points acquired is computationally intense and datasets have to be filtered depending on the aim of the investigation.
2. Data collection methods suffer from perspective effects, which can lead to either zones of occlusion (shadow effect) or spatially-variable point densities.
3. 3D point clouds are normally interpolated to digital elevation models either as regular grids or triangulated irregular networks.

Thus, for change detection proposes (i.e. the determination of topographic change, including erosion and deposition), it is appropriate to develop methods based upon the direct analysis of the point clouds using semi-automatic approaches allowing to detect and extract individual features.

In this practical computing lab we introduce a semi-automated method developed for isolating and identifying features of topographic change (i.e., apparent changes caused by surface displacements and indicating erosion or deposition) directly from point cloud data using a Density-Based Spatial Clustering of Applications with Noise (DBSCAN). This methodology was developed in **?** for a very active rock glacier front located in the Swiss Alps: the Tsarmine rock glacier.

8.1 The overall methodology

Stepwise analysis:

1. Point clouds were generated using a TLS on a number of dates.
2. Point clouds were co-registered using stable zones within the surveyed area.
3. Using a threshold value, only the points that can have experienced topographical changes were selected.
4. The final dataset was treated with DBSCAN, aiming at grouping cluster points into single features and filtering out noise points, found in low-density regions.

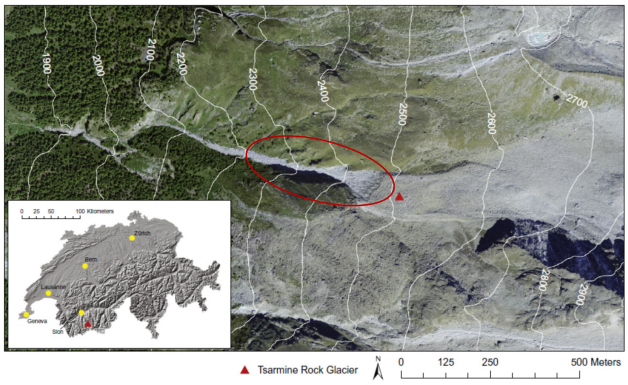


Figure 11: The Tsarmine rock glacier, Hérens Valley, in the Western Swiss Alps. Source: Micheletti et al, 2016

The present lab deals with steps 3 and 4: 3D features detection using DBSCAN. The detected features are finally labeled as clusters and visualized in a 3D map.

Material loss or gain can eventually be analysed in a GIS environment according to the elevation assignment of the change and the volume of change computed for each cluster by a triangulation process (not performed in this lab). The entire work is inspired by the study of **?**.

8.1.1 DBSCAN: 3-D density based clustering algorithm

DBSCAN allows identification of spatial clusters of arbitrary shape on the base of the local density of points. Points that are close together are grouped into the same cluster, while isolated points are labelled as noise. Two parameters are required to perform this classification: the minimum number of points necessary to form a cluster ($MinPts$), and the neighborhood size epsilon (eps). The algorithm explores each point in the dataset, counting the number of the neighboring points falling within a circle (for the 2D model) or a sphere (for the 3D model) of radius equal to eps (**??**a). If this number is equal to or greater than $MinPts$, points are labelled as belonging to the same cluster (**??**b). If this number is lower than $MinPts$, points are classified as noise. The central point of each cluster is called "core-point". Since some points can be density-reachable by more than one core-point, they can belong to more than one cluster. In this case clusters are blended together to form a unique feature of arbitrary shape (**??**c).

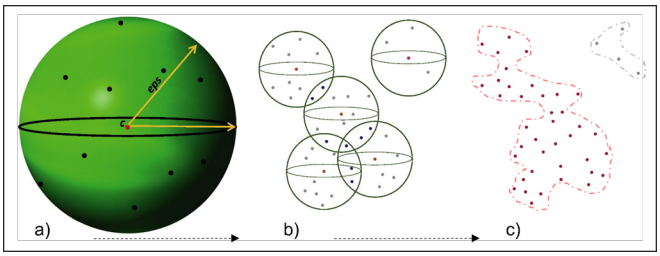


Figure 12: Parameters in DBSCAN to form a cluster

8.1.2 Field campaign

An ultra-long range LiDAR RIEGL VZ-6000 scanner was employed to acquire sequential 3D datasets of the rock glacier front. The TLS scans were performed on different dates over two consecutive summers: a first survey was carried out on the 23th of September 2014, and the last one on the 22th of September 2015.

DBSCAN requires as input a 3-dimensional dataset, which in our case consists on the points clouds plus the displacement distance. For the two co-registered datasets (2014 and 2015), we set the first as the target and the more recent one as the reference. For each point in the target dataset the corresponding nearest point in the reference dataset was identified and the distance between them evaluated using the software Cloud Compare. Co-registration errors, estimated as noise and not real material loss/gain signals, has to be removed from the analysis. Here, we noted from field observations that the size of displaced boulders is typically > 0.30 m and we use this as a change criteria to remove the noise.

8.2 Computing lab: DBSCAN

8.2.1 Load the libraries

Fist you have to load the following libraries:

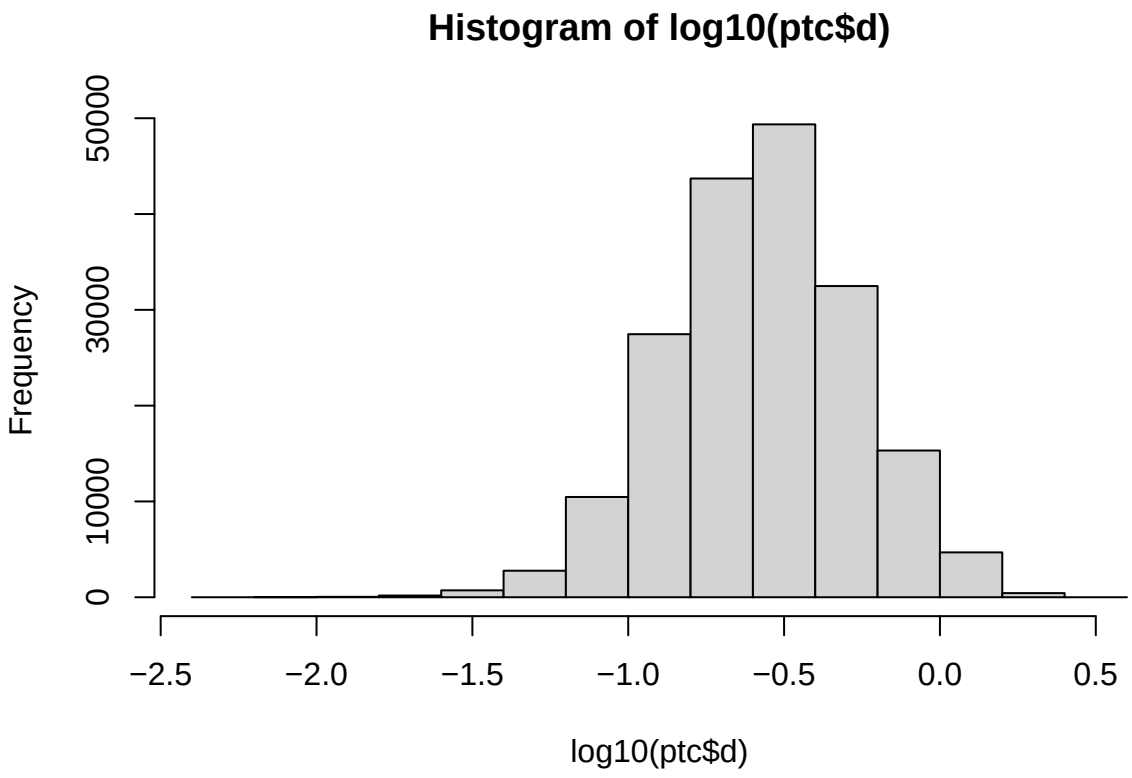
- *dbscan*: a fast implementation of several density-based algorithms of the DBSCAN family.
- *rgl*: provides medium to high level functions for 3D interactive graphics
- *plot3Drgl*: plot 3D graphs in rgl window.
- *classInt*: selected methods to choose class intervals for mapping puposes.
- *RColorBrewer*: provides color schemes for mapping.

```
library("dbscan")
library("rgl")
library("plot3Drgl")
library("classInt")
library("RColorBrewer")

(.packages())
```

8.2.2 Import and visualize the point clouds dataset

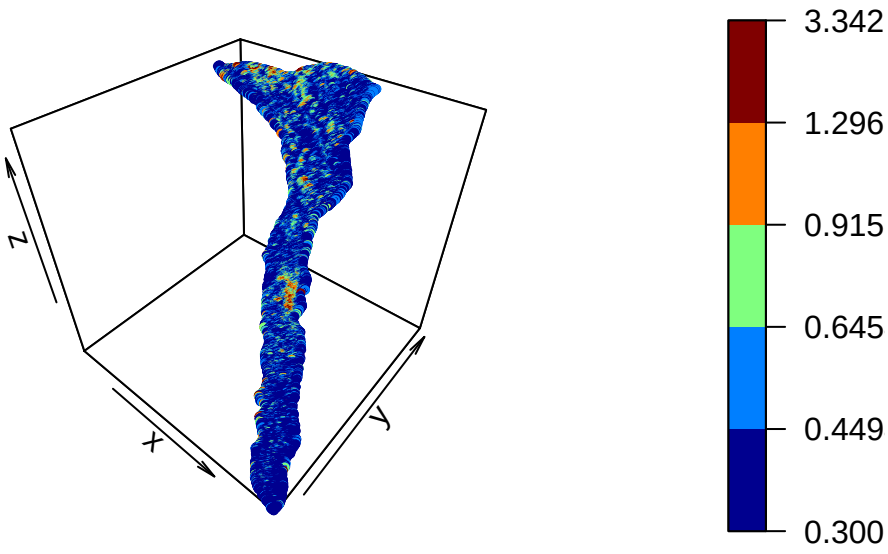
We provide the dataset corresponding to one-year displacement distance, masked over the the rock glacier front. The noise-points are removed using a threshold of 30 cm. Finally we will plot the 3D-points cloud filtered dataset using the displacement distance as attribute to display the map.

```
# Import point cloud dataset (ptc)
# 1 year displacement TLS campaigns; data masked over the active front only.
ptc <- read.table("data/DBSCAN/TsarmineRG_230914_frontonly_XYZ_dist_ref_220915.txt", header=FALSE, sep=

# Add names to columns: X,Y, Z coordinates and the displacement distance "d".
colnames(ptc)<-c("Y","X","Z","d") # rename columns

# Inspect the dataset:
str(ptc)
summary(ptc$d)
hist(log10(ptc$d))
```




```
# Create a subset: removing noise-points (d>30cm).
ptc30 <- subset(ptc, d>=0.3)

# Inspect the subset:
str(ptc30)
summary(ptc30$d)
```

```
# Plot-3D: display 3D plot (X,Y,Z) with class intervals based on the distance d.

# Create a class interval (int) based on natural breaks:
int <- classIntervals(ptc30$d, n=5, style="fisher")
  cut.vals<-int$brks

# Display the 3D-plot:
scatter3D(ptc30$X, ptc30$Y, ptc30$Z, colvar =ptc30$d, breaks=cut.vals, cex=0.5)
```



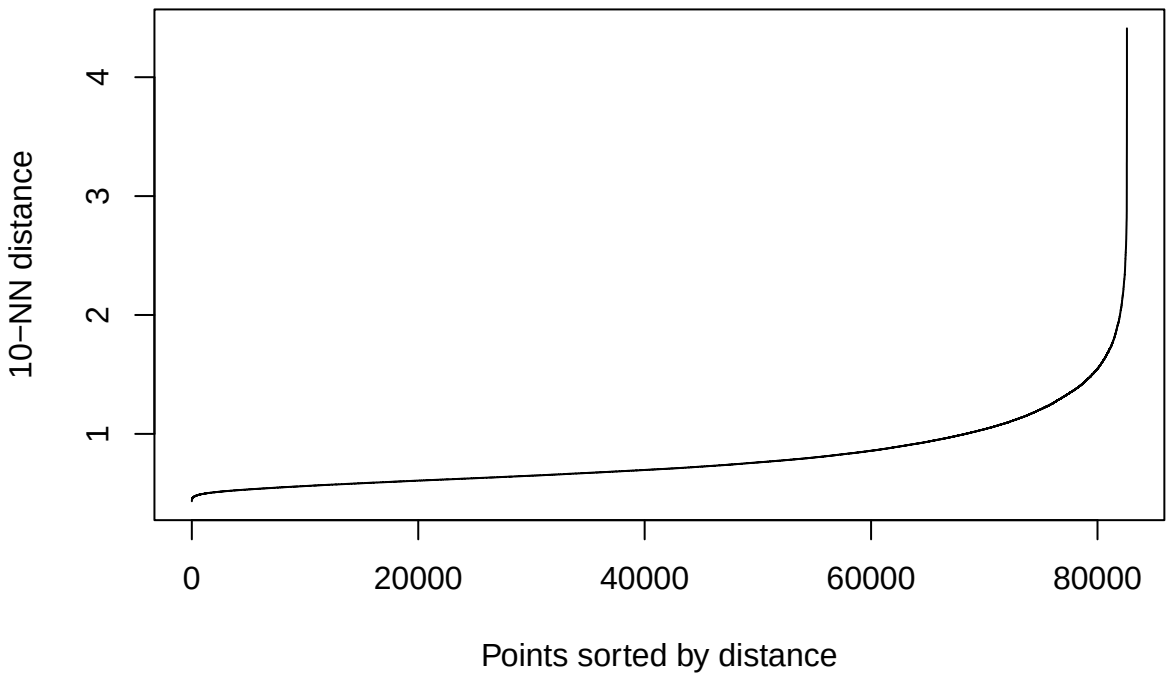
8.2.3 Compute DBSCAN

The two parameters *eps* and $MinPts$ greatly affect the output cluster detection leading to the identification of a large number of small clusters (for small values) or a small number of large features of arbitrary shape (for large values). Once fixed the $MinPts$ parameter, a suitable value for the *eps* neighborhood size can be estimated using k-Nearest Neighbors (k-NN) distance graph, imposing k equal to $MinPts$ and plotting the distance to the nearest neighbors. The optimal *eps*-value should coincide with the stronger curvature of the graph.

In the following, the minimum number of points allowing to form a cluster is fixed first (to 10), and then the plot of the k-NN distance is used to find a suitable value for the *eps* neighborhood size.

For the computation we introduce here the R package `dbscan` (**?**)

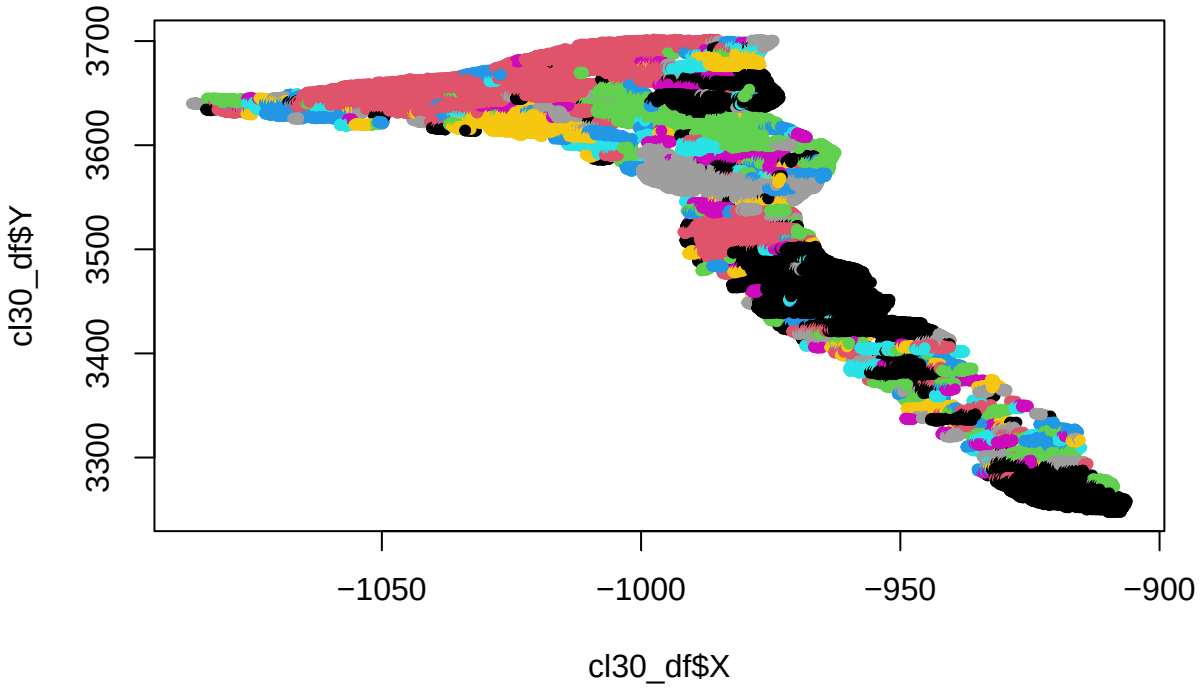
```
# Plot the k-NN distance graph (with k=10)
kNNdistplot(ptc30[,-4], k = 10)
```



Once the best *MinPts* and *eps* parameter have been selected, the DBSCAN function can be run.

```
# Eps=1m (~3xsigma) ; MinPts=10
cl30<-dbscan(ptc30, 1, minPts=10)

# Inspect the results:
str(cl30)
summary(cl30$cluster)
sum(cl30$cluster==0)
```

```
# Plot only the detected clusters.

# Join the data:
cl30<-cbind(ptc30, cluster=cl30$cluster)

# Save as data frame and remove the noise (label==0)
cl30_df<-as.data.frame(subset(cl30, cl30$cluster>=1))

# Simple plot:
plot(cl30_df$X, cl30_df$Y, col=cl30_df$cluster, pch=20)
```




```
# Animated 3D-plot:
#plot3d(cl30_df, col=cl30_df$cluster, pch=20)
```

Finally you can export the result as a *.txt file and import it in a GIS system for further analyses, such as to determine the volumes of loss or gain of material.

```
write.table(cl30_df, file="cls30_1_10.txt", sep="\t")
```

8.3 Conclusions and further analyses

The proposed method allowed to detection of features of changes on a rock glacier front located in the Swiss Alps. Single cluster features of erosion and deposition/front movements were extracted directly from point clouds, detected by DBSCAN without the necessity of interpolate the 3-D original data.

To ensure that everything is perfectly clear, we propose you to do this lab again, by changing the values of *MinPts* and *eps* and discuss the compare the resulting extracted features.