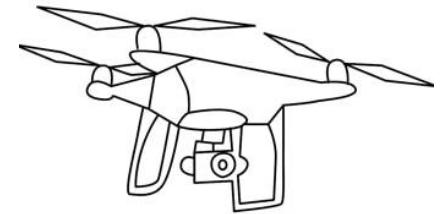




Modelamiento y Programación GIS



Cesar Ivan Alvarez

cesarivanalvarezmendoza@gmail.com

A grayscale photograph of a person sitting at a desk, viewed from the side. They are looking at a large computer monitor that displays a complex GIS map with various colored regions and lines. The person's hands are on a keyboard in front of them. The background is slightly blurred, showing office equipment.

Unidad 1 – Introducción a la Programación GIS

Qué es un GIS?



- Un sistema de información geográfica (GIS) es una estructura para recopilar, gestionar y analizar datos. Con esta capacidad única, GIS revela información más profunda sobre los datos, como los patrones, las relaciones y las situaciones, lo que ayuda a los usuarios a tomar decisiones más inteligentes.

(Esri, 2025)

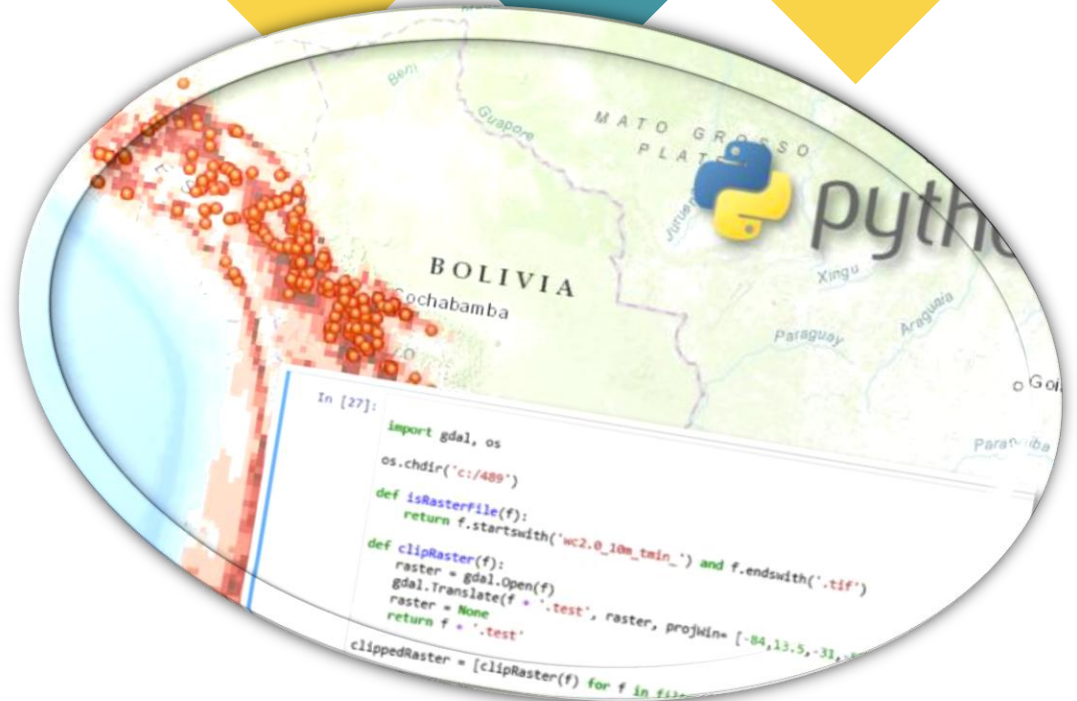
Qué es Programación?



- Es el proceso de escribir instrucciones precisas y detalladas en un lenguaje de programación, para que una computadora pueda realizar una tarea específica. Estas instrucciones se llaman código, y se organizan de manera lógica para que el ordenador pueda comprenderlas y ejecutarlas.

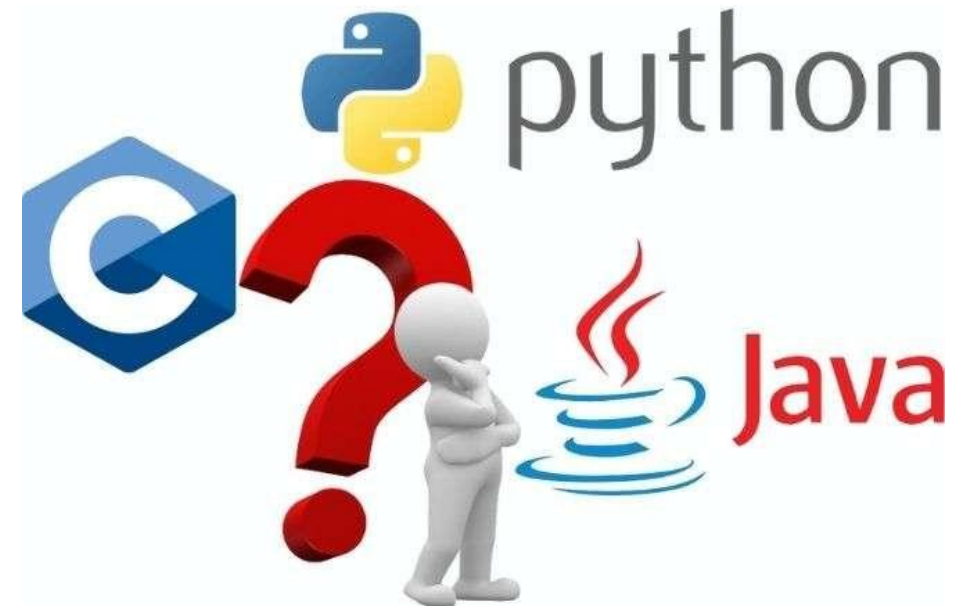
Y la programación en GIS?

- Automatizar procesos.
- Análisis espacial avanzado.
- Integración con otras tecnologías.
- Personalizar herramientas.
- Procesamiento de Big Data en la nube.

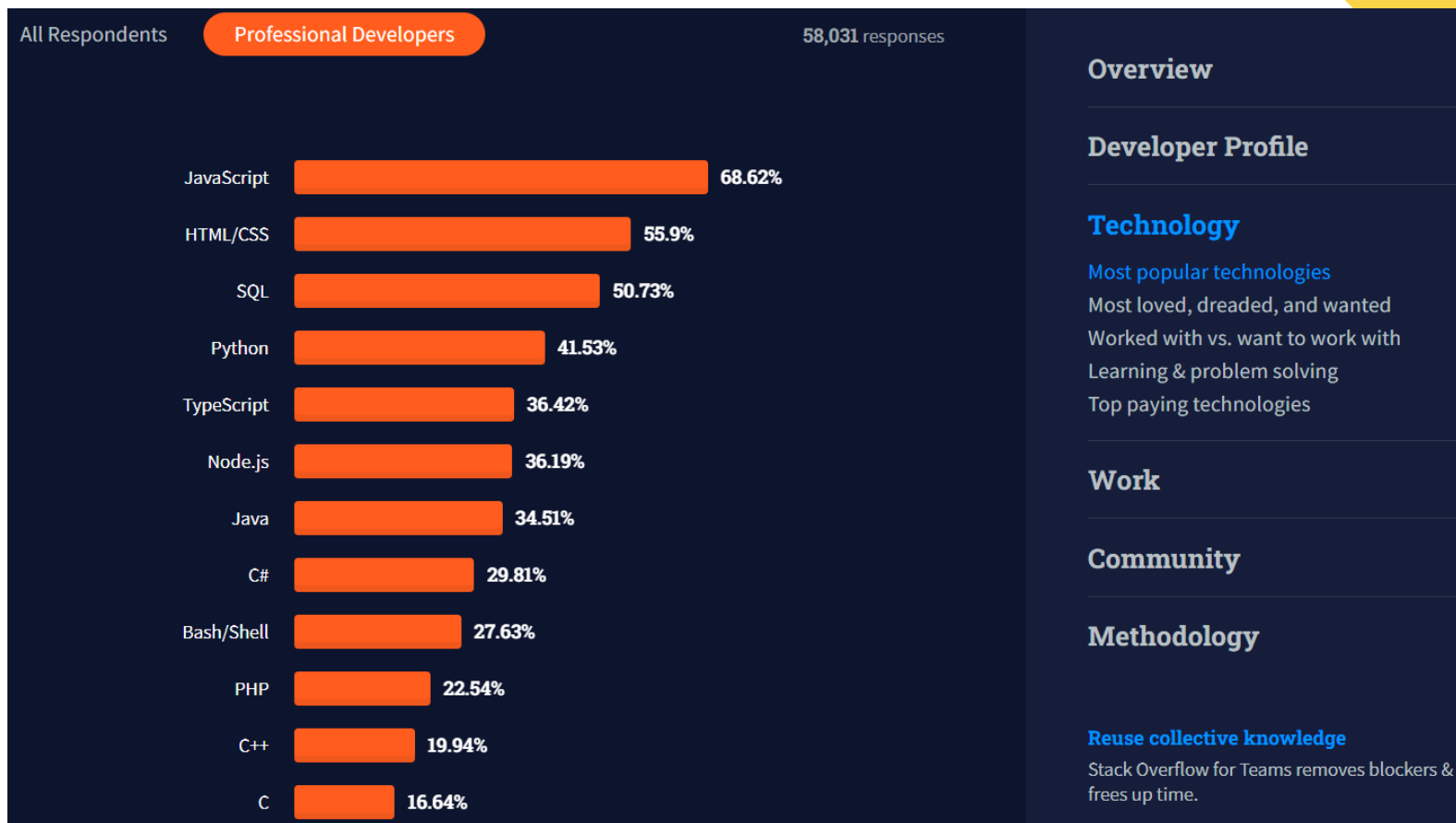


Qué es lenguaje de programación?

- Un lenguaje de programación es una herramienta que permite desarrollar software o programas para computadora. Los lenguajes de programación son empleados para diseñar e implementar programas encargados de definir y administrar el comportamiento de los dispositivos físicos y lógicos de una computadora.
- Los lenguajes de programación de alto nivel permiten dar órdenes al ordenador con un lenguaje parecido al nuestro (Visual Basic, Pascal, Logo, C++, JavaScript, etc.) y siempre o casi siempre en inglés.



Los lenguajes de programación en GIS

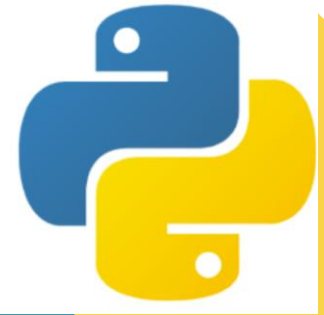


R - Rstudio



- R es un lenguaje de programación y entorno de desarrollo diseñado específicamente para el análisis estadístico y la manipulación de datos. Es un software libre y de código abierto que proporciona una amplia gama de herramientas y bibliotecas para el procesamiento de datos y la generación de gráficos.
- RStudio es un entorno de desarrollo integrado (IDE) para el lenguaje de programación R, dedicado a la computación estadística y gráficos. Incluye una consola, editor de sintaxis que apoya la ejecución de código, así como herramientas para el trazado, la depuración y la gestión del espacio de trabajo.

Python – Jupyter Notebook



- Python es un lenguaje de programación informático que se utiliza a menudo para crear sitios web y software, automatizar tareas y realizar análisis de datos. Python es un lenguaje de propósito general, lo que significa que se puede utilizar para crear una variedad de programas diferentes y no está especializado en ningún problema específico.
- Jupyter Notebook es un IDE que se lo ejecuta en el navegador y tiene la particularidad de combinar código con texto, imágenes, ecuaciones, enlaces, entre otros; permite la incorporación de Markdowns y también el uso del lenguaje LaTeX para la escritura de ecuaciones matemáticas.

Qué es un Algoritmo?

Entrada – Cuáles son los datos que necesito para lograr el objetivo?

Proceso – Con qué métodos o herramientas lo logro?

Salida – Qué busco o cuál es el objetivo?

Un algoritmo es una secuencia de PASOS a seguir para resolver un problema.



Qué es un Diagrama de Flujo?



Un diagrama de flujo es una representación gráfica del Algoritmo

Principios de Programación – Variables y constantes

- Una variable es un elemento de datos con nombre cuyo valor puede cambiar durante el curso de la ejecución de un programa. Un nombre de variable debe seguir el convenio de denominación de un identificador (carácter alfabético o número y el signo de subrayado).

	
<pre>a <- 1 b <- 2 c <- a+b print(c) nam = "Cesar" cat("Hi my name is", nam) decision = TRUE //Boolean variable in R TRUE or FALSE</pre>	<pre>a = 1 b = 2 a+b print(c) nam = 'Cesar' print('Hi my name is ' + nombre) decision = True #Boolean variable in Python True or False</pre>

Principios de Programación – Variables y constantes

- Existen otro tipo de estructuras de variables de entrada como matrices, dataframe, listas, diccionarios, entre otros.



```
dataset_r <- data.frame(  
  ID = 1:5,  
  Name = c("Alice", "Bob", "Charlie", "David", "Eve"),  
  Age = c(25, 30, 22, 35, 28),  
  Score = c(85.5, 90.0, 78.5, 88.0, 92.5))  
dataset_r$Name  
dataset_r[,3]  
  
dictionary_r <- list(  
  name = "Alice",  
  age = 25,  
  scores = c(85, 90, 78))  
print(dictionary_r$name)
```



```
import pandas as pd  
dataset_py = pd.DataFrame({  
  'ID': [1, 2, 3, 4, 5],  
  'Name': ['Alice', 'Bob', 'Charlie', 'David', 'Eve'],  
  'Age': [25, 30, 22, 35, 28],  
  'Score': [85.5, 90.0, 78.5, 88.0, 92.5]})  
dataset_py["Name"]  
dataset_py.iloc[:, 2]  
dataset_py.loc[:, 'Age':'Score']  
dictionary_py = {  
  "name": "Alice",  
  "age": 25,  
  "scores": [85, 90, 78]}  
print(dictionary_py["name"])
```

Principios de Programación - Condicionantes

Los condicionales son estructuras que permiten elegir entre la ejecución de una acción u otra. Se relaciona como el “si” condicional que usamos dentro de una frase. Se utiliza en inglés las palabras if (si) y else (caso contrario).



```
a <- 200
b <- 125

if (b > a) {
  c <- 2*a + b
  cat("b is greater than a and c is",c)
} else if (a == b) {
  print("a and b are equal")
} else {
  c <- a + 2*b
  cat("a is greater than b and c is",c)
}
```



```
a = 100
b = 125

if b > a:
    c = 2 * a + b
    print(f"b is greater than a and c is {c}")
elif a == b:
    print("a and b are equal")
else:
    c = a + 2 * b
    print(f"a is greater than b and c is {c}")
```

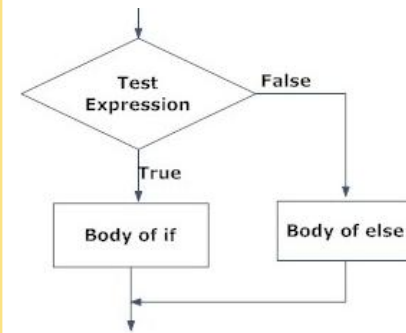


Fig: Operation of if...else statement

Principios de Programación - Bucles

Los bucles son secuencias de instrucciones cíclicas que nos permiten repetir una acción mientras se está cumpliendo una determinada condición y el proceso no se para hasta que esta condición no se cumple.



```
for(numero in 1:10) {  
  print(numero)  
}
```

```
v1 <- c(3,4,5)  
v2 <- c(3,4,6)  
v3 <- c(0,0,0)  
n = length(v1)
```

```
for (i in 1:n) {  
  v3[i] <- v1[i] + v2[i]  
}  
V3
```



```
for numero in range(1, 11):  
  print(numero)
```

```
v1 = [3, 4, 5]  
v2 = [3, 4, 6]  
v3 = [0, 0, 0]  
n = len(v1)
```

```
for i in range(n):  
  v3[i] = v1[i] + v2[i]  
  
print(v3)
```

Principios de Programación - Bucles

La declaración `break` se utiliza para salir de un bucle inmediatamente cuando se cumple una condición.

La declaración `next` se utiliza para omitir la iteración actual y continuar con la siguiente iteración del bucle, similar a la declaración `continue` en Python



```
for (i in 1:10) {  
  if (i == 5) {  
    break  
  }  
  cat("The i-value is:", i, "\n")  
}
```

```
for (i in 1:5) {  
  if (i == 3) {  
    next  
  }  
  cat("The i-value is:", i, "\n")  
}
```



```
for i in range(1, 11):  
    if i == 5:  
        break  
    print("The i-value is:", i)
```

```
for i in range(1, 6):  
    if i == 3:  
        continue  
    print("The i-value is:", i)
```

Principios de Programación - Funciones

Una función es un bloque de código que realiza alguna operación. Las funciones son útiles para encapsular las operaciones comunes en un solo bloque reutilizable, idealmente con un nombre que describa claramente lo que hace la función.



```
mean_two_numbers <-  
function(num_1, num_2) {  
  mean <- (num_1 + num_2) / 2  
  return (mean)}  
  
mean_two_numbers(1,3)
```



```
def mean_two_numbers(num_1,  
num_2):  
    mean = (num_1 + num_2) / 2  
    return mean  
  
mean_two_numbers(1, 3)
```

Práctica 1 – Carga de Shapefile

Check if the sf package is installed, and install it only if it's missing

```
if (!requireNamespace("sf", quietly = TRUE)) {
```

```
  install.packages("sf")}
```

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

```
shapefile_path <- "C:/Users/alvarece/MOROCCO_BOUNDARIES.shp" # Define the path to your shapefile
```

```
shapefile_data <- st_read(shapefile_path) # Read the shapefile
```

```
plot(shapefile_data$geometry) # Plot the shapefile
```

```
crs_info <- st_crs(shapefile_data) # Get CRS (Coordinate Reference System)
```

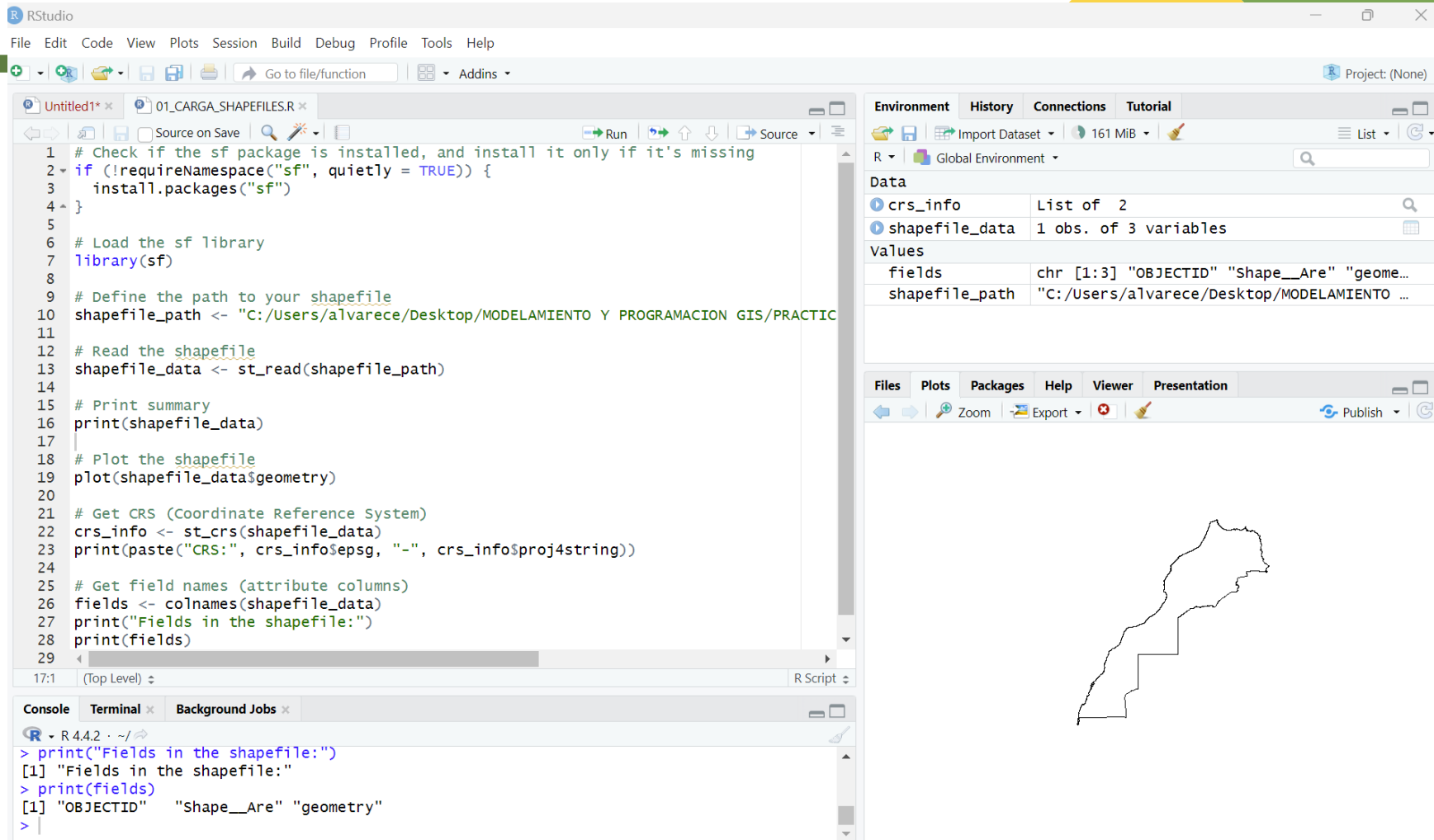
```
print(paste("CRS:", crs_info$epsg, "-", crs_info$proj4string))
```

```
fields <- colnames(shapefile_data) # Get field names (attribute columns)
```

```
print(fields)
```



Práctica 1 – Carga de Shapefile



RStudio interface showing the process of loading a shapefile.

Script (01_CARGA_SHAPEFILES.R):

```
1 # Check if the sf package is installed, and install it only if it's missing
2 if (!requireNamespace("sf", quietly = TRUE)) {
3   install.packages("sf")
4 }
5
6 # Load the sf library
7 library(sf)
8
9 # Define the path to your shapefile
10 shapefile_path <- "C:/Users/alvarece/Desktop/MODELAMIENTO Y PROGRAMACION GIS/PRACTIC
11
12 # Read the shapefile
13 shapefile_data <- st_read(shapefile_path)
14
15 # Print summary
16 print(shapefile_data)
17
18 # Plot the shapefile
19 plot(shapefile_data$geometry)
20
21 # Get CRS (Coordinate Reference System)
22 crs_info <- st_crs(shapefile_data)
23 print(paste("CRS:", crs_info$epsg, "-", crs_info$proj4string))
24
25 # Get field names (attribute columns)
26 fields <- colnames(shapefile_data)
27 print("Fields in the shapefile:")
28 print(fields)
29
```

Environment:

Object	Class	Size
crs_info	List of 2	
shapefile_data	1 obs. of 3 variables	

Values:

Field	Value
fields	chr [1:3] "OBJECTID" "Shape__Are" "geome..."
shapefile_path	"C:/Users/alvarece/Desktop/MODELAMIENTO ..."

Console:

```
> print("Fields in the shapefile:")
[1] "Fields in the shapefile:"
> print(fields)
[1] "OBJECTID" "Shape__Are" "geometry"
>
```

Plots: A map of Argentina is displayed.



Práctica 1 – Carga de Shapefile

```
import arcpy # Import the arcpy library

# Define the path to the shapefile
shapefile_path = r"C:/Users/alvarece/Desktop/MOROCCO_BOUNDARIES.shp"

# Load the shapefile into a feature class (feature layer)
shapefile_fc = arcpy.management.MakeFeatureLayer(shapefile_path, "shapefile_layer")

desc = arcpy.Describe(shapefile_fc)

print(f"Feature Type: {desc.shapeType}")

print(f"Spatial Reference: {desc.spatialReference.name}") #Information about CRS from the shapefile

# List the fields in the shapefile
fields = [field.name for field in arcpy.ListFields(shapefile_fc)]

print("Fields in the shapefile:", fields)
```

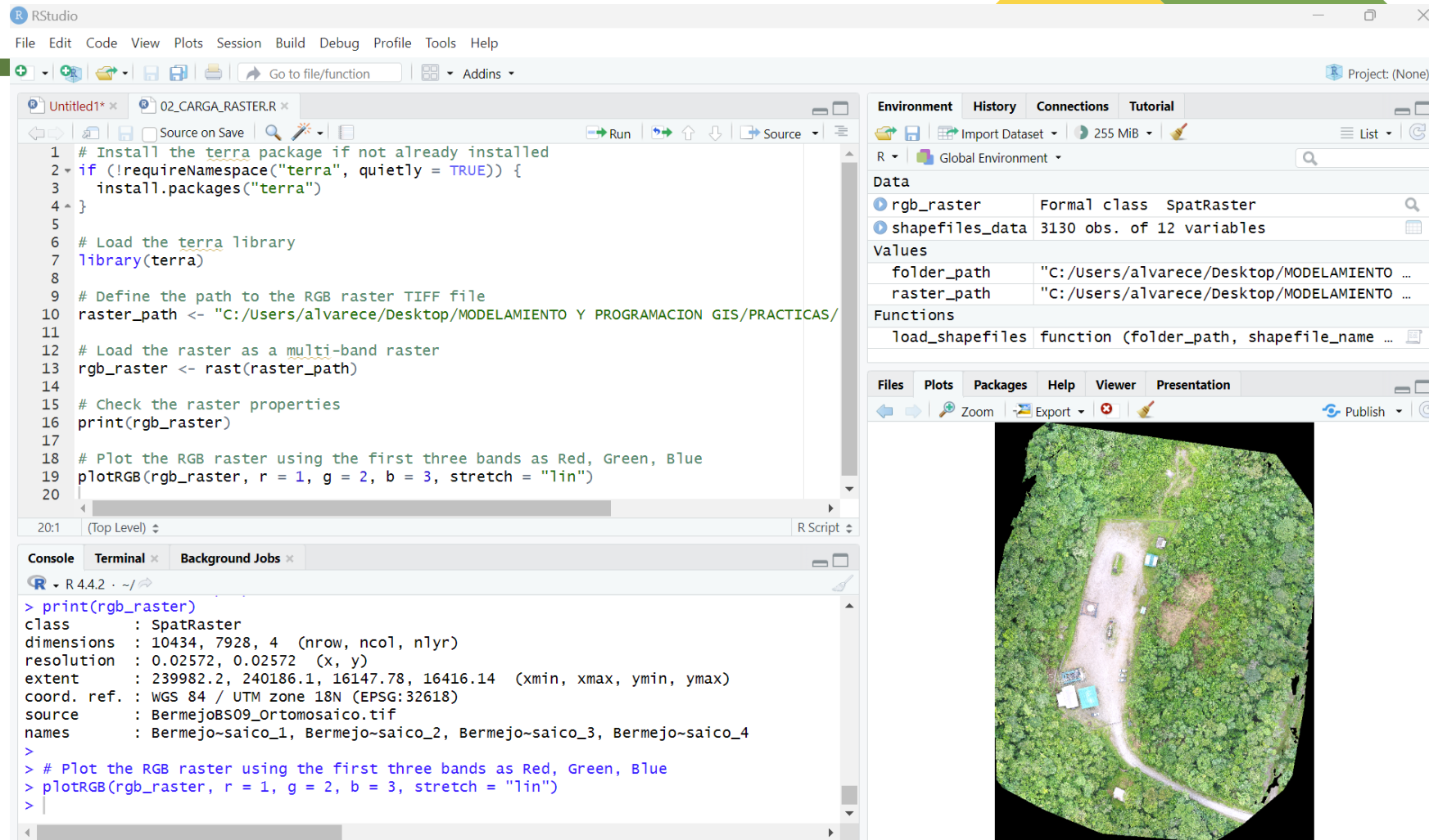


Práctica 2 – Carga de Raster

```
if (!requireNamespace("terra", quietly = TRUE)) {  
  install.packages("terra")  
}  
library(terra)  
raster_path <- "C:/DATOS/BermejoBS09_Ortomosaico.tif"  
  
rgb_raster <- rast(raster_path) # Load the raster as a multi-band raster  
  
print(rgb_raster) # Check the raster properties  
  
# Plot the RGB raster using the first three bands as Red, Green, Blue  
plotRGB(rgb_raster, r = 1, g = 2, b = 3, stretch = "lin")
```



Práctica 2 – Carga de Raster



The screenshot displays the RStudio interface with the following components:

- Source Editor:** Contains R code for installing the `terra` package, loading it, and loading a multi-band raster from a TIFF file.
- Environment Panel:** Shows the objects created in the Global Environment.
- Console:** Displays the output of the `print(rgb_raster)` command, showing the raster's properties.
- Plots Panel:** Shows a plot of the loaded RGB raster.

```
1 # Install the terra package if not already installed
2 if (!requireNamespace("terra", quietly = TRUE)) {
3   install.packages("terra")
4 }
5
6 # Load the terra library
7 library(terra)
8
9 # Define the path to the RGB raster TIFF file
10 raster_path <- "C:/Users/alvarece/Desktop/MODELAMIENTO Y PROGRAMACION GIS/PRACTICAS/
11
12 # Load the raster as a multi-band raster
13 rgb_raster <- rast(raster_path)
14
15 # Check the raster properties
16 print(rgb_raster)
17
18 # Plot the RGB raster using the first three bands as Red, Green, Blue
19 plotRGB(rgb_raster, r = 1, g = 2, b = 3, stretch = "lin")
20
```

Environment Panel:

Object	Class	Details
rgb_raster	Formal class	SpatRaster
shapefiles_data		3130 obs. of 12 variables

Console Output:

```
> print(rgb_raster)
class           : SpatRaster
dimensions      : 10434, 7928, 4  (nrow, ncol, nlyr)
resolution      : 0.02572, 0.02572  (x, y)
extent          : 239982.2, 240186.1, 16147.78, 16416.14  (xmin, xmax, ymin, ymax)
coord. ref.     : WGS 84 / UTM zone 18N (EPSG:32618)
source          : BermejoBS09-Ortomosaico.tif
names           : Bermejo-saico_1, Bermejo-saico_2, Bermejo-saico_3, Bermejo-saico_4
>
> # Plot the RGB raster using the first three bands as Red, Green, Blue
> plotRGB(rgb_raster, r = 1, g = 2, b = 3, stretch = "lin")
>
```

Plots Panel: Shows a plot of the loaded RGB raster, displaying a landscape with green vegetation and a dirt road.



Práctica 2 – Carga de Raster

```
import arcpy

raster_path = r"C:/DATOS/BermejoBS09_Ortomosaico.tif"

# Set the current ArcGIS Pro project and map
aprx = arcpy.mp.ArcGISProject("CURRENT") # Use the currently open project
map_obj = aprx.activeMap # Get the active map

# Directly add the raster to the map without creating an intermediate layer
map_obj.addDataFromPath(raster_path)

aprx.save() # Save the changes to the project

print("Raster loaded and added to the map successfully.")
```



Práctica 3 – Carga de una carpeta con varios Shapefiles

```
library(sf)
```

```
library(ggplot2)
```

```
folder_path <- "C:/Users/alvarece/Desktop/MODELAMIENTO Y PROGRAMACION GIS/PRACTICAS/DATOS/IGM_1G"
```

```
# List all shapefiles in the folder (and subfolders)
```

```
shapefile_paths <- list.files( path = folder_path, pattern = "\\\\.shp$", recursive = TRUE, full.names = TRUE)
```

```
if (length(shapefile_paths) == 0) { stop("No shapefiles found in the specified directory.")} # Check if any shapefiles were found
```

```
# Load all shapefiles and extract only geometry
```

```
shapefiles_list <- lapply(shapefile_paths, function(path) { tryCatch({ shp <- st_read(path, quiet = TRUE) st_geometry(shp) # Extract only the geometry  
, error = function(e) { message(paste("Error reading file:", path, "-", e$message)) return(NULL) })})
```

```
shapefiles_list <- shapefiles_list[!sapply(shapefiles_list, is.null)] # Remove NULL elements from the list (failed shapefile loads)
```



Práctica 3 – Carga de una carpeta con varios Shapefiles

```
if (length(shapefiles_list) > 0) { combined_geometries <- do.call(c, shapefiles_list) # Combine only geometries into a single spatial dataframe
```

```
combined_shapefiles <- st_as_sf(combined_geometries) # Create a simple sf object with only geometry
```

```
# Plot all shapefiles on a single map using ggplot2
```

```
ggplot() +
```

```
  geom_sf(data = combined_shapefiles, fill = "blue", color = "black", alpha = 0.5) +
```

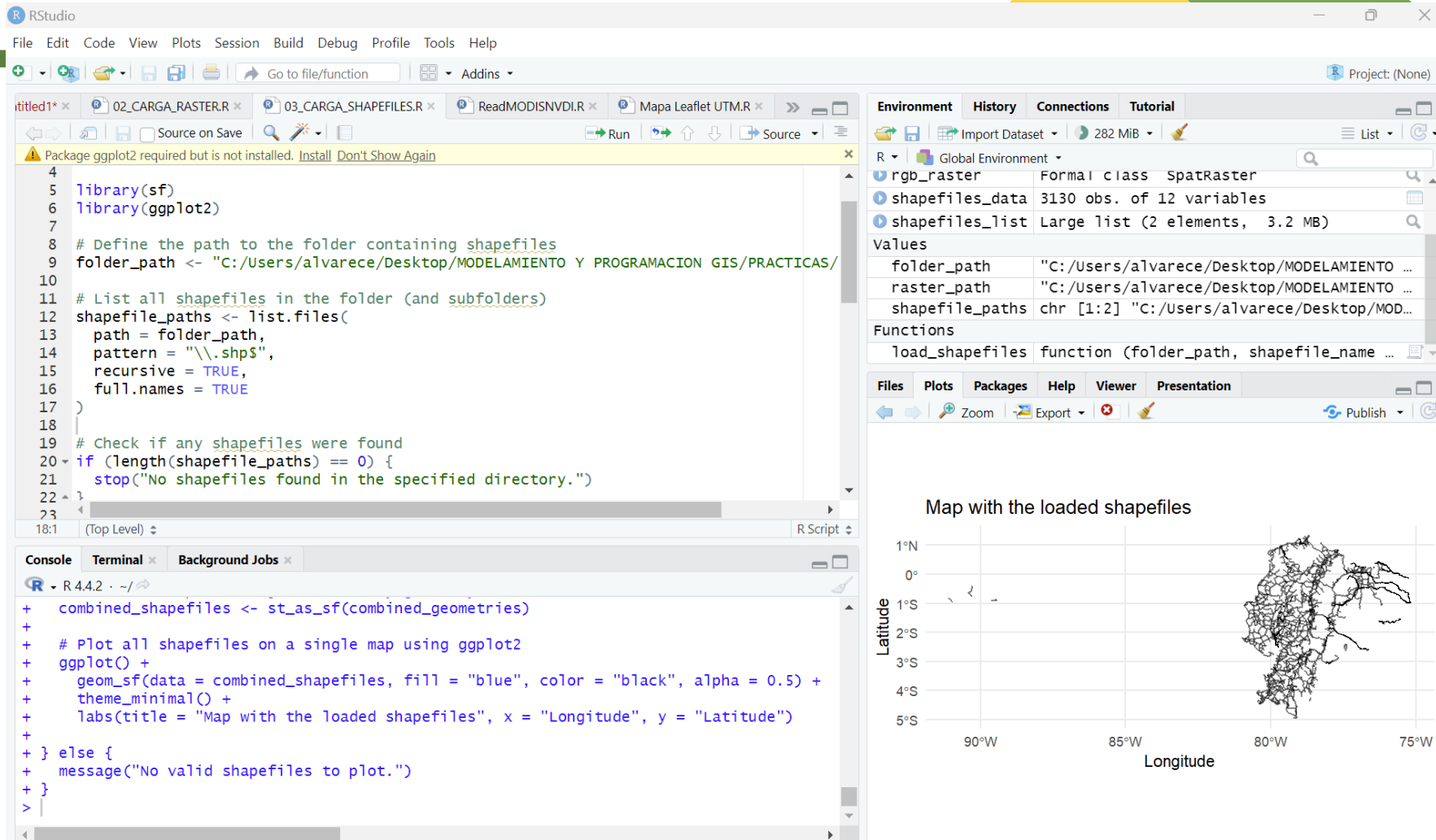
```
  theme_minimal() +
```

```
  labs(title = "Map with the loaded shapefiles", x = "Longitude", y = "Latitude")
```

```
} else { message("No valid shapefiles to plot.")}
```



Práctica 3 – Carga de una carpeta con varios Shapefiles



The screenshot displays the RStudio interface with the following components:

- Source Editor:** Contains R code for loading shapefiles from a folder.
- Environment:** Shows the loaded objects: `rgb_raster` (SpatRaster), `shapefiles_data` (3130 obs. of 12 variables), and `shapefiles_list` (Large list (2 elements, 3.2 MB)).
- Console:** Shows the execution of the code, including the creation of `combined_shapefiles` and the plotting of the map.
- Plots:** Displays a map titled "Map with the loaded shapefiles" showing the outline of South America with internal boundaries.

```
4 library(sf)
5 library(ggplot2)
6
7 # Define the path to the folder containing shapefiles
8 folder_path <- "C:/Users/alvarece/Desktop/MODELAMIENTO Y PROGRAMACION GIS/PRACTICAS/"
9
10 # List all shapefiles in the folder (and subfolders)
11 shapefile_paths <- list.files(
12   path = folder_path,
13   pattern = "\\\\.shp$",
14   recursive = TRUE,
15   full.names = TRUE
16 )
17
18 # Check if any shapefiles were found
19 if (length(shapefile_paths) == 0) {
20   stop("No shapefiles found in the specified directory.")
21 }
22
23
24 combined_shapefiles <- st_as_sf(combined_geometries)
25
26 # Plot all shapefiles on a single map using ggplot2
27 ggplot() +
28   geom_sf(data = combined_shapefiles, fill = "blue", color = "black", alpha = 0.5) +
29   theme_minimal() +
30   labs(title = "Map with the loaded shapefiles", x = "Longitude", y = "Latitude")
31
32 } else {
33   message("No valid shapefiles to plot.")
34 }
35
36 >
```

Environment:

Object	Class	Size
<code>rgb_raster</code>	Formal class SpatRaster	
<code>shapefiles_data</code>	3130 obs. of 12 variables	
<code>shapefiles_list</code>	Large list (2 elements, 3.2 MB)	

Values:

Variable	Value
<code>folder_path</code>	"C:/Users/alvarece/Desktop/MODELAMIENTO Y PROGRAMACION GIS/PRACTICAS/"
<code>raster_path</code>	"C:/Users/alvarece/Desktop/MODELAMIENTO Y PROGRAMACION GIS/PRACTICAS/"
<code>shapefile_paths</code>	chr [1:2] "C:/Users/alvarece/Desktop/MODELAMIENTO Y PROGRAMACION GIS/PRACTICAS/"

Functions:

Function	Definition
<code>load_shapefiles</code>	function (folder_path, shapefile_name ...)

Plots:

Map with the loaded shapefiles

Latitude: 1°N, 0°, 1°S, 2°S, 3°S, 4°S, 5°S

Longitude: 90°W, 85°W, 80°W, 75°W



Práctica 3 – Carga de una carpeta con varios Shapefiles

```
import arcpy, os

folder_path = r"C:/Users/alvarece/Desktop/MODELAMIENTO Y PROGRAMACION GIS/PRACTICAS/DATOS/IGM_1G"

aprx = arcpy.mp.ArcGISProject("CURRENT") # Use the currently open project
map_obj = aprx.activeMap # Get the active map

# Iterate through all files in the directory and subdirectories
for root, dirs, files in os.walk(folder_path):
    for file in files:
        # Check if the file is a shapefile
        if file.lower().endswith(".shp"):
            shapefile_path = os.path.join(root, file)
            print(f>Loading shapefile: {shapefile_path}")
            map_obj.addDataFromPath(shapefile_path) # Add the shapefile directly to the map
aprx.save() # Save changes to the ArcGIS Pro project
```



Práctica 4 – CSV a shapefile

```
library(sf)
library(ggplot2)

csv_path <- "C:/Users/alvarece/Desktop/MODELAMIENTO Y PROGRAMACION GIS/PRACTICAS/DATOS/POINTS.csv" # Define the path to the CSV file
points_data <- read.csv(csv_path) # Load the CSV file into a data frame
head(points_data) # Inspect the data

crs_epsg <- 32717 # Define the coordinate reference system (EPSG 32717 - WGS 84 / UTM zone 17S)
points_sf <- st_as_sf(points_data, coords = c("X", "Y"), crs = crs_epsg) # Convert the data frame to an sf object (point geometry)
output_shapefile <- "C:/Users/alvarece/Desktop/MODELAMIENTO Y PROGRAMACION GIS//points_data.shp" # Define the path to save the shapefile
st_write(points_sf, output_shapefile, delete_layer = TRUE) # Save the sf object as a shapefile
print(paste("Shapefile created at:", output_shapefile))

# Plot the points using ggplot2

ggplot() + geom_sf(data = points_sf, color = "red", size = 5) + theme_minimal() + labs(title = "Points from CSV as Shapefile", x = "Easting", y = "Northing")
```



Práctica 4 – CSV a shapefile Punto a Polígono

```
library(sf)
library(ggplot2)

csv_path <- "C:/Users/alvarece/Desktop/MODELAMIENTO Y PROGRAMACION GIS/PRACTICAS/DATOS/POINTS.csv" # Define the path to the CSV file
points_data <- read.csv(csv_path) # Load the CSV file into a data frame
head(points_data) # Inspect the data

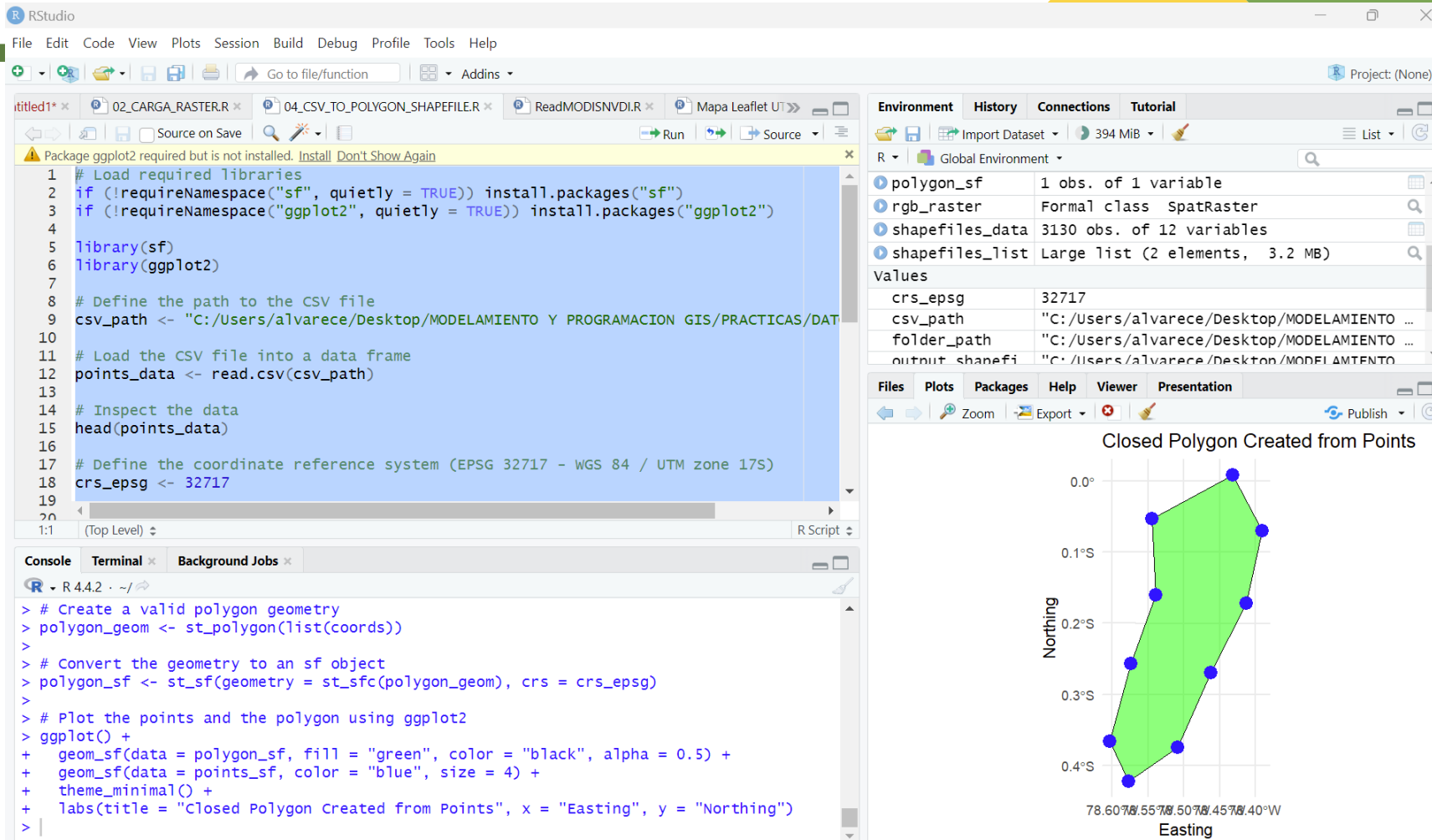
crs_epsg <- 32717 # Define the coordinate reference system (EPSG 32717 - WGS 84 / UTM zone 17S)
points_sf <- st_as_sf(points_data, coords = c("X", "Y"), crs = crs_epsg) # Convert the data frame to an sf object (point geometry)
coords <- st_coordinates(points_sf) # Extract coordinates and ensure the polygon is closed
coords <- rbind(coords, coords[1, ]) # Add the first point to the end to close the polygon
polygon_geom <- st_polygon(list(coords)) # Create a valid polygon geometry
polygon_sf <- st_sf(geometry = st_sfc(polygon_geom), crs = crs_epsg) # Convert the geometry to an sf object

# Plot the points and the polygon using ggplot2

ggplot() + geom_sf(data = polygon_sf, fill = "green", color = "black", alpha = 0.5) + geom_sf(data = points_sf, color = "blue", size = 4) + theme_minimal() +
  labs(title = "Closed Polygon Created from Points", x = "Easting", y = "Northing")
```



Práctica 4 – CSV a shapefile Punto a Polígono



Práctica 4 – CSV a shapefile

```
import arcpy, os

csv_path = r"C:/Users/alvarece/Desktop/MODELAMIENTO Y PROGRAMACION GIS/PRACTICAS/DATOS/POINTS.csv"

output_folder = r"C:/Users/alvarece/Desktop/MODELAMIENTO Y PROGRAMACION GIS/PRACTICAS/DATOS" # Define the output shapefile path

output_shapefile = os.path.join(output_folder, "points_data.shp")

# Define the coordinate fields and spatial reference (e.g., WGS 84 with EPSG 4326)

x_field = "X" # Replace with the actual column name for longitude

y_field = "Y" # Replace with the actual column name for latitude

spatial_ref = arcpy.SpatialReference(32717) # WGS 84

# Convert the CSV to a shapefile

arcpy.management.XYTableToPoint(csv_path, output_shapefile, x_field, y_field, coordinate_system=spatial_ref)

print(f"Shapefile created at: {output_shapefile}")

# Load the new shapefile into the ArcGIS Pro map

aprx = arcpy.mp.ArcGISProject("CURRENT") # Use the currently open project

map_obj = aprx.activeMap # Get the active map

aprx.save() # Save changes to the ArcGIS Pro project
```



Taller 1

Con un Script de R:

Generar un polígono mostrando los puntos cargados desde un CSV, similar a la práctica 4



Práctica 5 – Reproyectar un Shapefile

```
library(sf, ggplot2)
```

```
setwd("C:/Users/alvarece/Desktop/MODELAMIENTO Y PROGRAMACION GIS/PRACTICAS/DATOS/ZonificacionAguacateMAGAP")
```

```
shapefile <- st_read("Zonificación_agroecologica_aguacate_litoral_2013.shp")
```

```
shapefile_reprojected <- st_transform(shapefile, crs = 4326) # Reproject the shapefile
```

```
# Create the plot
```

```
ggplot(data = shapefile_reprojected) + geom_sf() + theme_minimal() +
```

```
theme(
```

```
  panel.grid.major = element_line(color = "gray", size = 0.5), # Major grid lines
```

```
  panel.grid.minor = element_line(color = "lightgray", size = 0.5), # Minor grid lines
```

```
  panel.background = element_rect(fill = "white") # Background color
```

```
) + labs(x = "Longitude", y = "Latitude") + coord_sf() # Coordinate system for the map
```

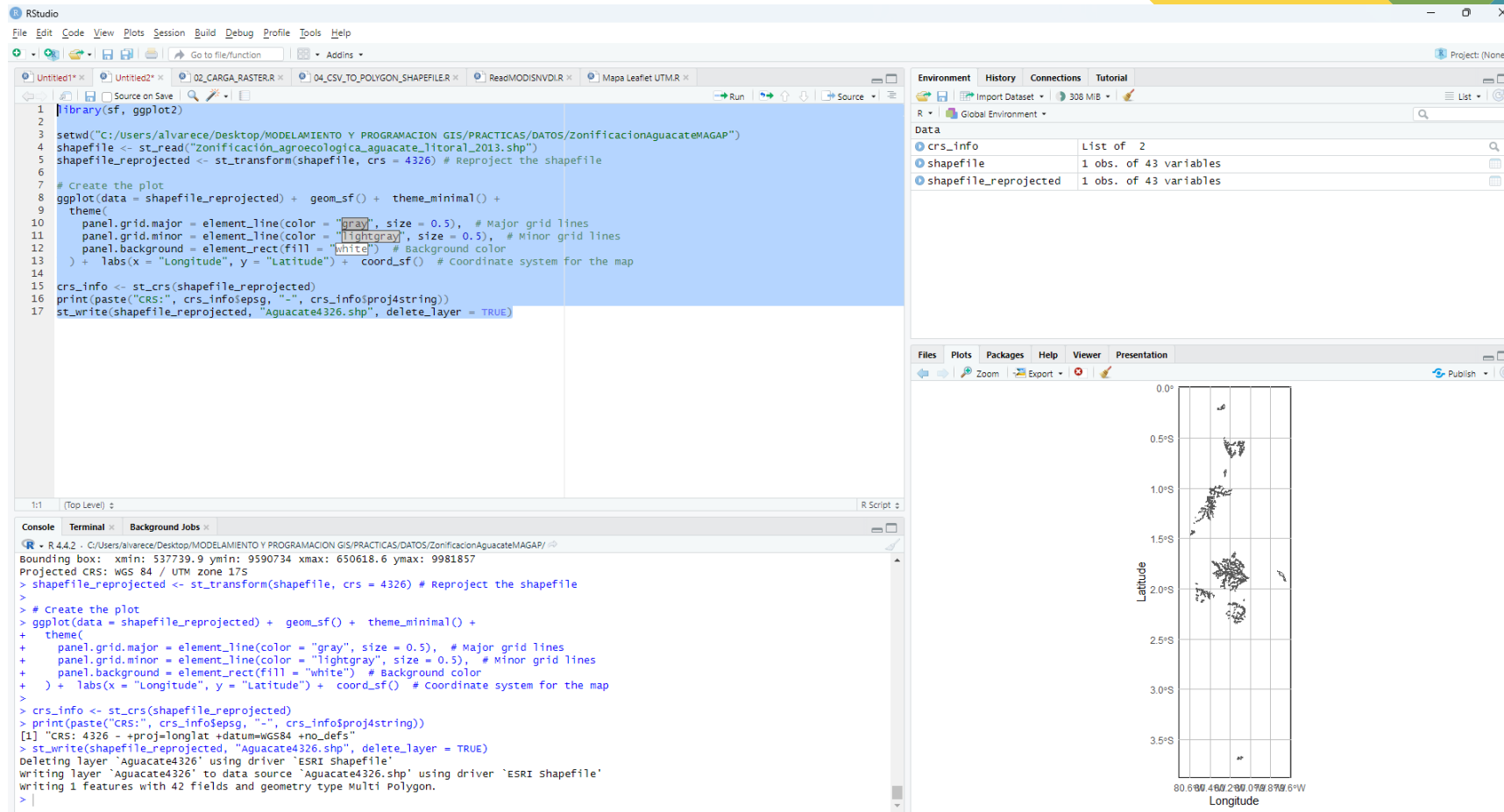
```
crs_info <- st_crs(shapefile_reprojected)
```

```
print(paste("CRS:", crs_info$epsg, "-", crs_info$proj4string))
```

```
st_write(shapefile_reprojected, "Aguacate4326.shp", delete_layer = TRUE)
```



Práctica 5 – Reproyectar un Shapefile



Práctica 5 – Reproyectar un Shapefile

```
import arcpy

arcpy.env.workspace = r"C:/Users/alvarece/Desktop/MODELAMIENTO Y PROGRAMACION GIS/PRACTICAS/DATOS/ZonificacionAguacateMAGAP"

input_shapefile = "Zonificación_agroecologica_aguacate_litoral_2013.shp"

output_shapefile = "Aguacate4326.shp"

# Define the spatial reference for EPSG:4326 (WGS 84)

output_spatial_ref = arcpy.SpatialReference(4326)

# Reproject the shapefile from EPSG:32717 to EPSG:4326 using arcpy

arcpy.management.Project(input_shapefile, output_shapefile, output_spatial_ref)

# Print the CRS info

spatial_ref = arcpy.Describe(output_shapefile).spatialReference

print(f"CRS: {spatial_ref.name} - EPSG: {spatial_ref.factoryCode}")
```



Práctica 6 – Buffer

```
library(sf, dplyr, ggplot2)

setwd("C:/Users/alvarece/Desktop/MODELAMIENTO Y PROGRAMACION GIS/PRACTICAS/DATOS")

data <- read.csv("POINTS.csv")

data$X <- as.numeric(data$X)

data$Y <- as.numeric(data$Y)

points <- st_as_sf(data, coords = c("X", "Y"), crs = 32717)

points_A <- points %>% filter(Type %in% c("A", "C")) # Filter by the 'Type' attribute where Type is "A"

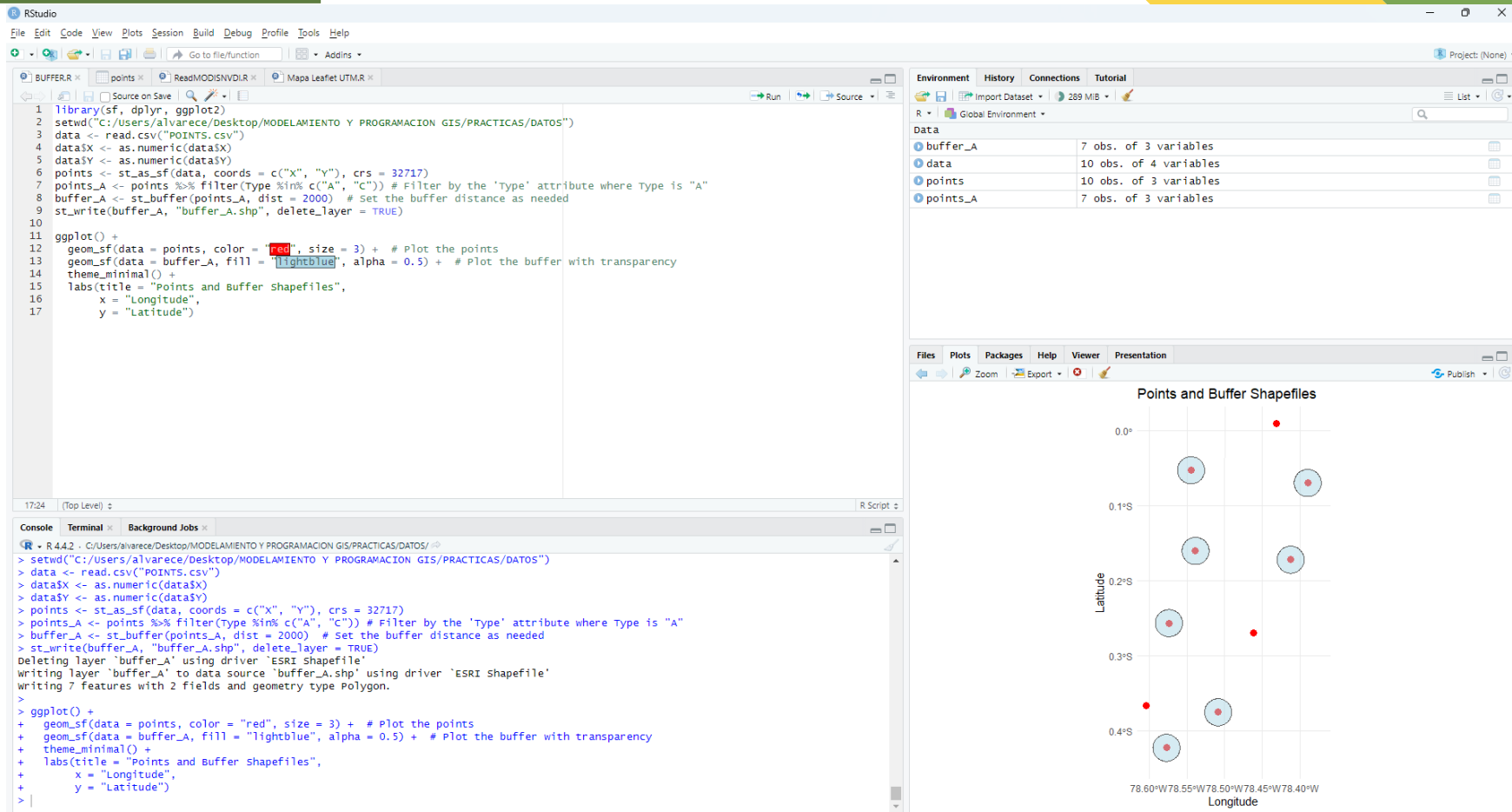
buffer_A <- st_buffer(points_A, dist = 2000) # Set the buffer distance as needed

st_write(buffer_A, "buffer_A.shp", delete_layer = TRUE)

ggplot() + geom_sf(data = points, color = "red", size = 3) + # Plot the points
  geom_sf(data = buffer_A, fill = "lightblue", alpha = 0.5) + # Plot the buffer with transparency
  theme_minimal() + labs(title = "Points and Buffer Shapefiles", x = "Longitude", y = "Latitude")
```



Práctica 6 – Buffer



Práctica 6 – Buffer

```
import arcpy

arcpy.env.workspace = r"C:/Users/alvarece/Desktop/MODELAMIENTO Y PROGRAMACION GIS/PRACTICAS/DATOS"

# Input CSV and output shapefiles

csv_file = "POINTS.csv" # CSV file with the points

output_shapefile = "points.shp"

buffer_shapefile = "buffer_A_C.shp" # Output buffer shapefile

# Define the fields in the CSV that contain the X (Longitude) and Y (Latitude) coordinates

arcpy.management.XYTableToPoint(csv_file, output_shapefile, "X", "Y", "", 32717)

expression = "Type IN ('A', 'C')" # Create a query to select Type = "A" or "C"

arcpy.management.MakeFeatureLayer(output_shapefile, "points_lyr") # Make a feature layer from the points shapefile

arcpy.management.SelectLayerByAttribute("points_lyr", "NEW_SELECTION", expression)

arcpy.analysis.Buffer("points_lyr", buffer_shapefile, "500 METERS") # Create a buffer around the selected points (500 meters)
```



Taller 2

Con un Script de R:

Obteniendo las coordenadas de un software como Google Earth Pro, realizar:

1. Generar la capa de puntos de las capitales de provincias del Ecuador, agregando dentro de sus atributos sus nombres y una clasificación aleatoria en el tema de seguridad (Alta, media o baja).
2. Generar un buffer de 5 km para las ciudades bajas, 10 km para las ciudades medias y 15 km para las ciudades altas.
3. Mostrar en un mapa de salida donde se colocarán de fondo un shapefile de provincias del Ecuador, los buffer resultantes y los puntos de las ciudades.



Práctica 7 – Clip

```
library(sf, dplyr, ggplot2)

setwd("C:/Users/alvarece/Desktop/MODELAMIENTO Y PROGRAMACION GIS/PRACTICAS/DATOS")

shapefiles <- list.files(path = "EURODATA", pattern = "\\\\.shp$", full.names = TRUE)

clip_polygon <- st_read("GERMANY/Germany_boundary.shp")

clip_crs <- st_crs(clip_polygon)

# Create a new folder for the output shapefiles (if it doesn't already exist)

output_folder <- "Germany_clipped"

if (!dir.exists(output_folder)) {

  dir.create(output_folder, recursive = TRUE)

}
```

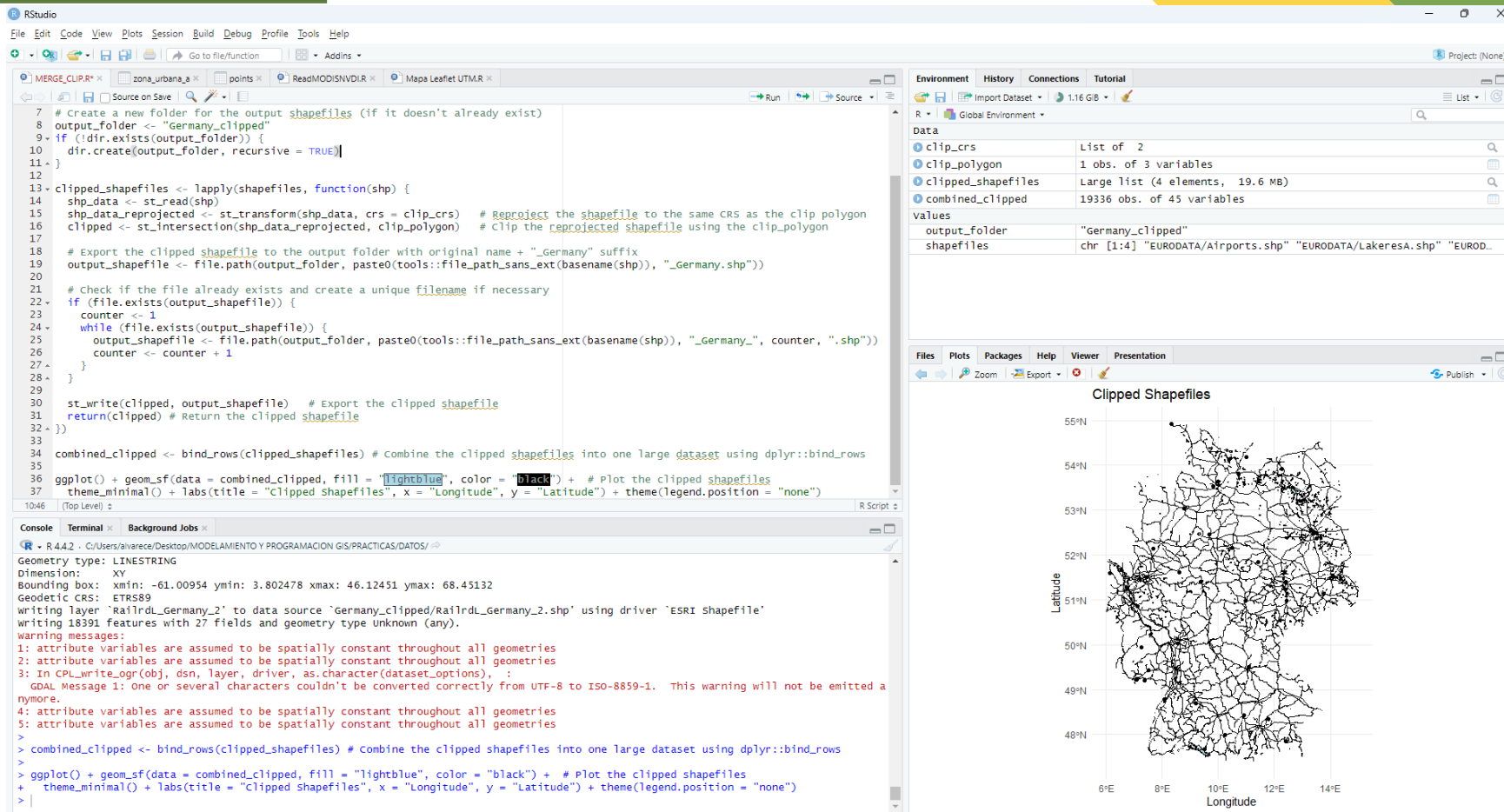


Práctica 7 – Clip

```
clipped_shapefiles <- lapply(shapefiles, function(shp) {  
  shp_data <- st_read(shp)  
  shp_data_reprojected <- st_transform(shp_data, crs = clip_crs) # Reproject the shapefile to the same CRS as the clip polygon  
  clipped <- st_intersection(shp_data_reprojected, clip_polygon) # Clip the reprojected shapefile using the clip_polygon  
  output_shapefile <- file.path(output_folder, paste0(tools::file_path_sans_ext(basename(shp)), "_Germany.shp")) # Export the clipped shapefile with original name + "_Germany" suffix  
  if (file.exists(output_shapefile)) {# Check if the file already exists and create a unique filename if necessary  
    counter <- 1  
    while (file.exists(output_shapefile)) {  
      output_shapefile <- file.path(output_folder, paste0(tools::file_path_sans_ext(basename(shp)), "_Germany_", counter, ".shp"))  
      counter <- counter + 1 }}  
  st_write(clipped, output_shapefile) # Export the clipped shapefile  
  return(clipped) # Return the clipped shapefile})  
combined_clipped <- bind_rows(clipped_shapefiles) # Combine the clipped shapefiles into one large dataset using dplyr::bind_rows  
  
ggplot() + geom_sf(data = combined_clipped, fill = "lightblue", color = "black") + # Plot the clipped shapefiles  
theme_minimal() + labs(title = "Clipped Shapefiles", x = "Longitude", y = "Latitude") + theme(legend.position = "none")
```



Práctica 7 – Clip



The screenshot displays the RStudio interface with the following components:

- Source Editor:** Contains R code for clipping shapefiles. The code includes comments and functions to create an output folder, clip a polygon, and export the result.
- Environment:** Shows the following objects:

Object	Description
clip_crs	List of 2
clip_polygon	1 obs. of 3 variables
clipped_shapefiles	Large list (4 elements, 19.6 MB)
combined_clipped	19336 obs. of 45 variables
- Files:** Shows the project files, including 'output_folder' and 'shapefiles'.
- Plots:** Displays a map titled 'Clipped Shapefiles' showing the rail network in Germany. The map includes latitude and longitude axes.
- Console:** Shows the output of the R code, including warnings about spatially constant variables and the successful execution of the clipping process.

Práctica 7 – Clip

```
import arcpy, os

arcpy.env.workspace = r"C:/Users/alvarece/Desktop/MODELAMIENTO Y PROGRAMACION GIS/PRACTICAS/DATOS"

input_folder = os.path.join(arcpy.env.workspace, "EURODATA")

clip_polygon = os.path.join(arcpy.env.workspace, "GERMANY", "Germany_boundary.shp")

clip_crs = arcpy.Describe(clip_polygon).spatialReference # Get the CRS of the clip polygon

# Set the output folder to save the clipped shapefiles

output_folder = os.path.join(arcpy.env.workspace, "Germany_clipped")

if not os.path.exists(output_folder):

    os.makedirs(output_folder)

shapefiles = [] # List all shapefiles in the folder (including subfolders)

for root, dirs, files in os.walk(input_folder):

    for file in files:

        if file.endswith(".shp"):

            shapefiles.append(os.path.join(root, file))
```



Práctica 7 – Clip

for shp in shapefiles:

try:

```
shp_crs = arcpy.Describe(shp).spatialReference # Get the CRS of the current shapefile
```

```
if shp_crs.name != clip_crs.name: # Reproject the shapefile if necessary
```

```
    reprojected_shp = os.path.join(output_folder, f"reprojected_{os.path.basename(shp)}")
```

```
    arcpy.management.Project(shp, reprojected_shp, clip_crs)
```

```
    shp = reprojected_shp # Update shapefile path to the reprojected version
```

```
clipped_shp = os.path.join(output_folder, f"clipped_{os.path.basename(shp)}") # Clip the reprojected shapefile with the clipping polygon
```

```
arcpy.analysis.Clip(shp, clip_polygon, clipped_shp)
```

```
print(f"Clipped shapefile saved as: {clipped_shp}")
```

except Exception as e:

```
    print(f"Error processing {shp}: {e}")
```



Práctica 8 – Merge

```
library(sf, dplyr, ggplot2)

setwd("C:/Users/alvarece/Desktop/MODELAMIENTO Y PROGRAMACION GIS/PRACTICAS/DATOS/IGM_50K")

shapefiles <- list.files(pattern = "curva_nivel_l.*\\.shp$", recursive = TRUE, full.names = TRUE) # List all the shapefiles called "curva_nivel_l" in the directory

curva_nivel_l_list <- lapply(shapefiles, st_read) # Read each shapefile

merged_curva_nivel_l <- do.call(rbind, curva_nivel_l_list) # Merge all shapefiles

zona_urbana_a <- st_read("zona_urbana_a/zona_urbana_a.shp")

quito_polygon <- zona_urbana_a %>% filter(nam == "QUITO")

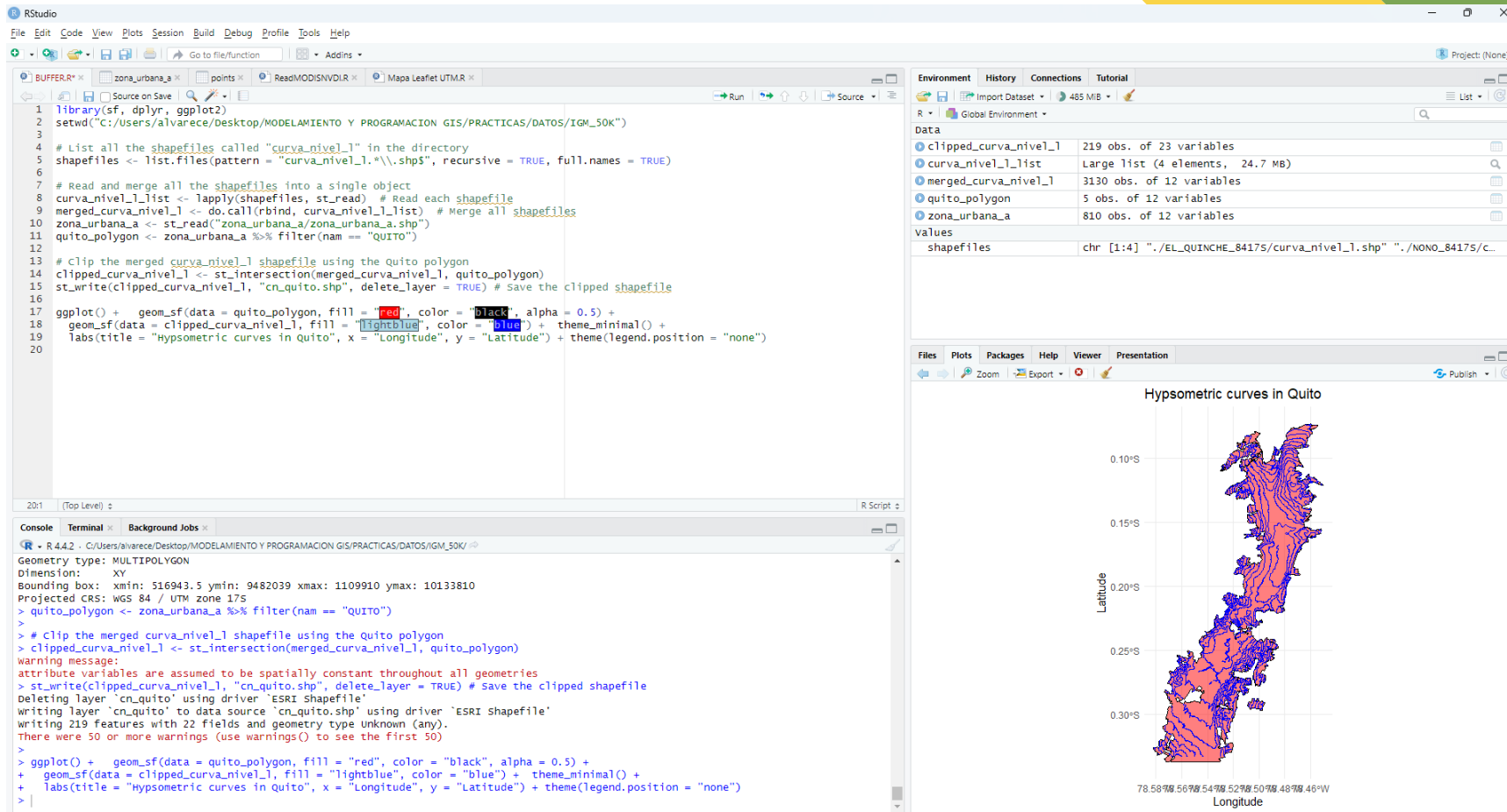
clipped_curva_nivel_l <- st_intersection(merged_curva_nivel_l, quito_polygon) # Clip the merged curva_nivel_l shapefile using the Quito polygon

st_write(clipped_curva_nivel_l, "cn_quito.shp", delete_layer = TRUE) # Save the clipped shapefile

ggplot() + geom_sf(data = quito_polygon, fill = "red", color = "black", alpha = 0.5) +
  geom_sf(data = clipped_curva_nivel_l, fill = "lightblue", color = "blue") + theme_minimal() +
  labs(title = "Hypsometric curves in Quito", x = "Longitude", y = "Latitude") + theme(legend.position = "none")
```



Práctica 8 – Merge



Práctica 8 – Merge

```
import arcpy, os

arcpy.env.workspace = r"C:/Users/alvarece/Desktop/MODELAMIENTO Y PROGRAMACION GIS/PRACTICAS/DATOS/IGM_50K"

shapefiles = [] # List all shapefiles called "curva_nivel_l" in the directory and subdirectories

for root, dirs, files in os.walk(arcpy.env.workspace):

    for file in files:

        if file.endswith(".shp") and "curva_nivel_l" in file:

            shapefiles.append(os.path.join(root, file))

merge1 = arcpy.management.Merge(shapefiles, x) # Merge all the shapefiles into one (use the "Merge" tool)

zona_urbana_a = "zona_urbana_a/zona_urbana_a.shp" # Read the zona_urbana_a shapefile and select the "QUITO" polygon

query = "nam = 'QUITO'"

quito1 = arcpy.management.SelectLayerByAttribute(zona_urbana_a, "NEW_SELECTION", query)

clipped_shapefile = "cn_quito.shp" # Clip the merged shapefile using the "Quito" polygon

arcpy.analysis.Clip(merge1, quito1, clipped_shapefile)
```



Práctica 9 – Generar un mapa

```
library(sf); library(dplyr); library(ggplot2); library(ggspatial)

setwd("C:/Users/alvarece/Desktop/MODELAMIENTO Y PROGRAMACION GIS/PRACTICAS/DATOS")
```

Define the function to read, transform, and assign color to shapefiles based on user input

```
rtc <- function(shapefile_path) {

  shapefile <- st_read(shapefile_path) # Read the shapefile

  transformed_shapefile <- st_transform(shapefile, crs = 4326) # Transform to EPSG:4326 (WGS 84)

  return(transformed_shapefile)}
```

```
roads <- rtc("Germany_clipped/RailrdL_Germany.shp")

boundaries <- rtc("GERMANY/Germany_boundary.shp")

rivers <- rtc("Germany_clipped/MajorRivers_Germany.shp")

airports <- rtc("Germany_clipped/Airports_Germany.shp")
```



Práctica 9 – Generar un mapa

ggplot() +

geom_sf(data = boundaries, aes(fill = "Boundaries"), size = 1, color = "gray", **alpha = 0.5**) +

geom_sf(data = roads, aes(fill = "Roads"), size = 1, color = "red") +

geom_sf(data = rivers, aes(fill = "Rivers"), size = 1, color = "blue") +

geom_sf(data = airports, aes(fill = "Airports"), **shape = 23, size = 3**, color = "yellow") +

scale_fill_manual(**# Manually define the fill colors and create a legend**

name = "Legend", **# Legend title**

values = c("Boundaries" = "gray", "Roads" = "red", "Rivers" = "blue", "Airports" = "yellow"), **# Map colors to labels**

guide = guide_legend(override.aes = list(shape = 22, size = 5)) **# Customize legend appearance**

) + theme_minimal() + theme(

panel.grid.major = element_line(color = "gray", size = 0.5), **# Major grid lines**

panel.grid.minor = element_line(color = "lightgray", size = 0.5), **# Minor grid lines**

panel.background = element_rect(fill = "white") **# Background color**) +

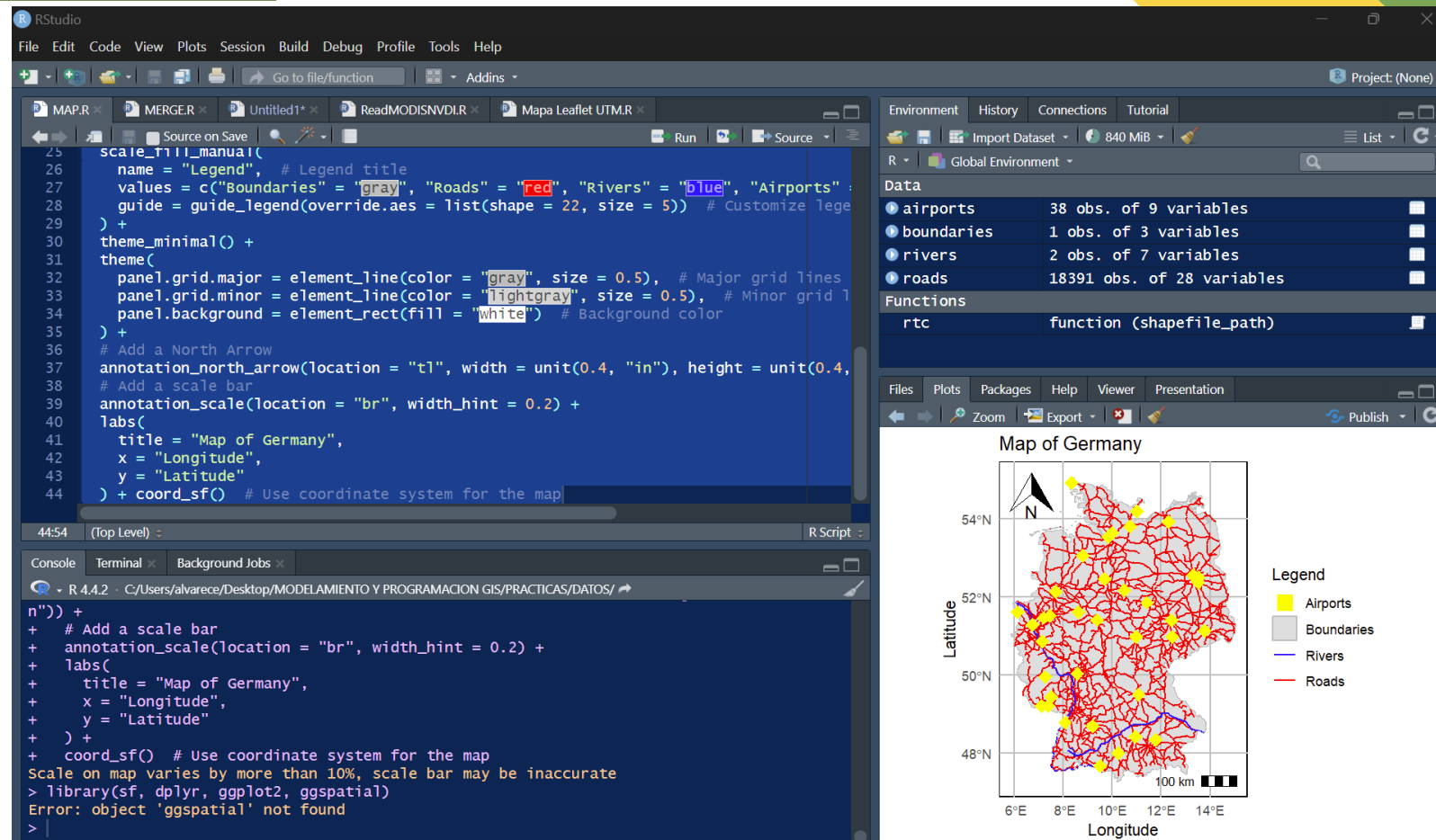
annotation_north_arrow(location = "tl", width = unit(0.4, "in"), height = unit(0.4, "in")) + **# Add a North Arrow**

annotation_scale(location = "br", width_hint = 0.2) + **# Add a scale bar**

labs(title = "Map of Germany", x = "Longitude", y = "Latitude") + coord_sf() **# Use coordinate system for the map**



Práctica 9 – Generar un mapa



Taller 3

Con un Script de R:

Generar un mapa con los elementos básicos similares a los mostrados en clase. El mapa es un mapa base que contenga los ríos, vías, poblados y el polígono urbano de una de las capitales del Ecuador en Escala 1:50000, con los colores correspondientes. Para eso usted deberá buscar las cartas por separado y netamente el polígono que al menos ocupe dos cartas, deberá ser el seleccionado. Con esto se realizará una práctica completa de la primera sección del módulo.



Gracias por su atención

César Iván Alvarez

cesarivanalvarezmendoza@gmail.com

<https://www.linkedin.com/in/cesar-ivan-alvarez-0847253a/>

