PROYECTO FINAL GRUPO Nº4. Georeferencia de Ciber Activos

JUAN FANEITE, CÉSAR BRICEÑO, ANDRYUS MIJARES, MELISA JURADO, FRANZ PRADO

2024-04-24

PASO 1. Cargar las librerías necesarias.

Se comienza con la carga de una serie de paquetes que son útiles para el análisis de datos, visualización y manipulación geoespacial. A continuación, se explica la utilizad de algunas de las bibliotecas usadas:

library(tidyverse): Carga un conjunto de paquetes diseñados para ciencia de datos que facilitan la importación, manipulación, exploración y visualización de datos.

library(cluster): Carga el paquete cluster, que proporciona funciones para el análisis de clustering.

library(ggplot2): Carga un sistema para crear gráficos declarativos basado en la gramática de gráficos.

library(maps): Carga el paquete maps, que proporciona herramientas para visualizar mapas.

library(leaflet): Carga leaflet, que permite la creación de mapas interactivos que pueden ser usados en documentos web.

library(readr): Carga readr, que proporciona una forma rápida y amigable de leer datos rectangulares (como CSV y archivos delimitados por caracteres).

library(webshot): Carga webshot, que permite tomar capturas de pantalla de páginas web, incluyendo Shiny y documentos R Markdown.

```
library(tidyverse)
```

```
## -- Attaching core tidyverse packages ------------------------ tidyverse 2.0.0 --
## v dplyr     1.1.4     v readr     2.1.5
## v forcats   1.0.0     v stringr   1.5.1
## v ggplot2   3.5.0     v tibble    3.2.1
## v lubridate 1.9.3     v tidyr     1.3.1
## v purrr     1.0.2
```

```
## -- Conflicts ---------------------------------------- tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
## i Use the conflicted package (<http://conflicted.r-lib.org/>) to force all conflicts to become errors
```

```
library(cluster)
library(factoextra)
```

```
## Welcome! Want to learn more? See two factoextra-related books at https://goo.gl/ve3WBa
```

```
library(ggplot2)
library(maps)
```

```
##
## Attaching package: 'maps'
##
## The following object is masked from 'package:cluster':
##
##     votes.repub
##
## The following object is masked from 'package:purrr':
##
##     map
```

```
library(sf)
```

```
## Linking to GEOS 3.11.2, GDAL 3.8.2, PROJ 9.3.1; sf_use_s2() is TRUE
```

```
library(readr)
library(readxl)
library(stats)
library(orloca)
```

```
## Loading required package: png
## Loading required package: ucminf
```

```
library(leaflet)
library(leaflet.extras)
library(dplyr)
library(webshot)
```

PASO 2. Leer el conjunto de datos

Cargar el archivo CSV:

read_delim: Esta función se utiliza para leer un archivo delimitado (como un CSV) en R.

"~/UNPA_3/MODULO_III/TF4/PROYECTO_FINAL/INVENTARIO_EJERCICIO6.csv" Es la ruta al archivo CSV que será cargado.

delim = ";": Indica que el delimitador utilizado en el archivo CSV es el punto y coma (;).

escape_double = FALSE: Significa que no se deben escapar las comillas dobles en el archivo.

col_types = cols(...): Define los tipos de datos para cada columna. Aquí, se especifica que todas las columnas listadas deben tratarse como números (col_number()).

trim_ws = TRUE: Indica que se deben eliminar los espacios en blanco al principio y al final de cada campo.

```
inventario <- read_delim("~/UNPA_3/MODULO_III/TF4/PROYECTO_FINAL/INVENTARIO_EJERCICIO6.csv",
    delim = ";", escape_double = FALSE, col_types = cols(Confidencialidad = col_number(),
        Integridad = col_number(), Disponibilidad = col_number(),
        Media = col_number(), Latitud = col_number(),
        Longitud = col_number()), trim_ws = TRUE)


puntos <- data.frame(inventario)
```

PASO 3. Realizar clustering con k-means.

kmeans(): Es la función que ejecuta el algoritmo k-means.

puntos[, c('Latitud', 'Longitud')]: Selecciona las columnas 'Latitud' y 'Longitud' del dataframe puntos. Estas columnas contienen las coordenadas que se utilizarán para el análisis de clustering.

centers = 4: Define el número de clusters (k) que el algoritmo debe encontrar. En este caso, se han especificado 4 centros.

El algoritmo k-means trabaja de la siguiente manera:

Inicialización: Selecciona k puntos aleatorios como centroides iniciales.

Asignación: Asigna cada punto al centroide más cercano basándose en la distancia euclidiana.

Actualización: Recalcula los centroides como el promedio de todos los puntos asignados a cada cluster.

Iteración: Repite los pasos 2 y 3 hasta que los centroides no cambien o se cumpla un criterio de parada, como un número máximo de iteraciones o un cambio mínimo en la posición de los centroides.

```r
set.seed(123) # Para reproducibilidad
kmeans_result <- kmeans(puntos[, c('Latitud', 'Longitud')], centers = 4)
```

PASO 4. Añadir la clasificación de clusters al inventario.

inventario: Es un data frame en R, que es una estructura de datos que almacena información en formato de tabla. Piensa en ello como una hoja de cálculo con filas y columnas.

: $Estesímboloseutilizaparaaccederaunacolumnaespecíficadentrodeundataframe.Porlotanto, inventario$ **se refiere a la columna llamada 'cluster' dentro del data frame 'inventario'.**

```r
inventario$cluster <- kmeans_result$cluster
```

PASO 5. Convertir a un objeto sf para poder graficar en un mapa

st_as_sf: Esta función es parte del paquete sf y se utiliza para convertir un data frame en un simple feature object (objeto de característica simple), que es una estructura de datos estándar para almacenar datos geoespaciales.

coords = c('Longitud', 'Latitud'): Define las columnas del data frame que contienen las coordenadas geográficas. En este caso, 'Longitud' y 'Latitud' son los nombres de las columnas en inventario que contienen las coordenadas longitudinales y latitudinales, respectivamente.

crs = 4326: Establece el Sistema de Referencia de Coordenadas (CRS) para el objeto de datos espaciales. El número 4326 corresponde al CRS conocido como WGS 84, que es el sistema de referencia estándar utilizado por el sistema de posicionamiento global (GPS) y es comúnmente utilizado para mapas y datos espaciales.

El resultado, inventario_sf, es un objeto de datos espaciales que puede ser utilizado para análisis geoespaciales y visualización en mapas con el paquete sf y otros paquetes compatibles como ggplot2 y leaflet.

```
inventario_sf <- st_as_sf(inventario, coords = c('Longitud', 'Latitud'), crs = 4326)
```

PASO 6. Obtener el mapa de Argentina

maps::map: La función map del paquete maps se utiliza para obtener datos de mapas. El uso de maps:: indica que se está llamando a la función map específicamente del paquete maps.

'world': Es el primer argumento de la función map y especifica que queremos obtener un mapa del mundo.

'argentina': Es el segundo argumento y especifica que queremos obtener los datos del mapa de Argentina.

plot = FALSE: Este argumento indica que no queremos trazar el mapa inmediatamente. En lugar de eso, queremos almacenar los datos del mapa en una variable.

fill = TRUE: Este argumento indica que queremos un mapa "relleno", lo que significa que las áreas dentro de las fronteras de Argentina estarán rellenas en lugar de solo tener las líneas de contorno.

El resultado, argentina_map, es un objeto de datos espaciales que representa el mapa de Argentina y puede ser utilizado para análisis geoespaciales o visualización con otros paquetes compatibles con sf, como ggplot2 o leaflet.

```
argentina_map <- st_as_sf(maps::map('world', 'argentina', plot = FALSE, fill = TRUE))
```

PASO 7. Graficar un mapa con los clusters (MApa sencillo)

ggplot(): Inicializa un gráfico ggplot vacío.

geom_sf(data = argentina_map, fill = 'lightgrey', color = 'white'): Añade una capa al gráfico que representa el mapa de Argentina. Los polígonos del mapa se rellenan de gris claro ('lightgrey') y los bordes son blancos ('white').

geom_sf(data = inventario_sf, aes(color = factor(cluster)), size = 2, alpha = 0.7): Añade otra capa al gráfico que representa los datos espaciales de inventario_sf. Los datos se agrupan por la variable cluster, que se convierte en un factor para asignar colores distintos a cada cluster. El tamaño de los puntos es 2 y la transparencia (alpha) es 0.7 para que se vean ligeramente transparentes.

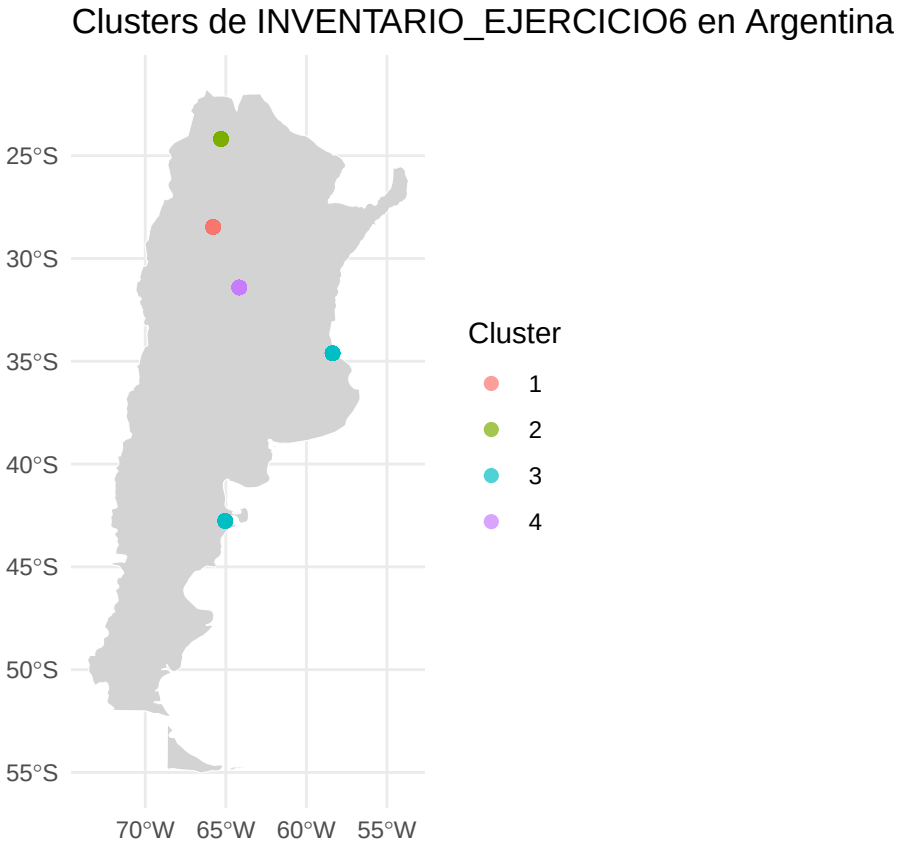
labs(color = 'Cluster'): Define la etiqueta de la leyenda para la estética de color, que en este caso es 'Cluster'.

theme_minimal(): Aplica un tema minimalista al gráfico, que tiene un fondo limpio y menos elementos decorativos.

ggtitle('Clusters de INVENTARIO_EJERCICIO6 en Argentina'): Añade un título al gráfico.

Este código resultará en un mapa de Argentina con puntos o áreas marcadas que representan diferentes clusters de datos del inventario del ejercicio 6, con una leyenda que indica a qué cluster pertenece cada color. Es una forma efectiva de visualizar la distribución geográfica de los datos.

```
ggplot() +
  geom_sf(data = argentina_map, fill = 'lightgrey', color = 'white') +
  geom_sf(data = inventario_sf, aes(color = factor(cluster)), size = 2, alpha = 0.7) +
  labs(color = 'Cluster') +
  theme_minimal() +
  ggtitle('Clusters de INVENTARIO_EJERCICIO6 en Argentina')
```


Clusters de INVENTARIO_EJERCICIO6 en Argentina
25°S
30°S
35°S
40°S
45°S
50°S
55°S
70°W
65°W
60°W
55°W
Cluster
1
2
3
4

PASO 8. Crear un mapa interactivo con leaflet

leaflet(inventario): Inicializa un mapa leaflet utilizando los datos de inventario.

addTiles(): Añade al mapa una capa de teselas (tiles) por defecto, que son las imágenes que componen el fondo del mapa.

addCircleMarkers(...): Añade marcadores circulares al mapa. Los argumentos especifican que:

lng = ~Longitud y lat = ~Latitud: Las coordenadas de los marcadores se toman de las columnas Longitud y Latitud del dataframe inventario.

popup = ~as.character(cluster): Al hacer clic en un marcador, se mostrará un popup con la información de la columna cluster, convertida a texto.

color = ~factor(cluster): El color de los marcadores se determina por la columna cluster, que se convierte en un factor para asignar un color distinto a cada valor único de cluster.

clusterOptions = markerClusterOptions(): Agrupa los marcadores cercanos en clusters para mejorar la legibilidad del mapa cuando hay muchos marcadores.

addProviderTiles(providers$CartoDB.Positron): Añade una capa de teselas del proveedor CartoDB.Positron, que ofrece un diseño claro y minimalista, ideal para superponer otros elementos visuales como los marcadores.

Al ejecutar este código, se creará un objeto mapa que contendrá un mapa interactivo con marcadores circulares representando los datos de inventario, agrupados por la variable cluster, y con un fondo de mapa proporcionado por CartoDB Positron.

```
mapa <- leaflet(inventario) %>%
  addTiles() %>%
  addCircleMarkers(lng = ~Longitud, lat = ~Latitud, popup = ~as.character(cluster), color = ~factor(clus
  addProviderTiles(providers$CartoDB.Positron)
```

PASO 9. Creacion de un Mapa Georeferenciado

activos <-: Este operador asigna el nuevo data.frame que se va a crear al objeto activos.

data.frame(...): Esta función crea un data.frame, que es una estructura de datos para almacenar datos tabulares en R.

lat = puntos$Latitud: Crea una columna llamada lat en el nuevo data.frame y le asigna los valores de la columna Latitud del data.frame puntos.

long = puntos$Longitud: Crea una columna llamada long y le asigna los valores de la columna Longitud de puntos.

dispo = puntos$Disponibilidad: Crea una columna dispo con los valores de Disponibilidad de puntos.

planta = puntos$Planta: Crea una columna planta con los valores de Planta de puntos.

ubicacion = puntos$Ubicacion.Geografica: Crea una columna ubicacion con los valores de Ubicacion.Geografica de puntos.

stringsAsFactors = FALSE: Este argumento indica que las cadenas de texto (strings) no deben convertirse automáticamente en factores, que es el comportamiento predeterminado en versiones anteriores de R. Los factores son útiles para categorías con un número limitado de valores, pero pueden ser inconvenientes si se desea trabajar con texto como cadenas de caracteres.

En resumen, en éste código, se toman ciertas columnas del data.frame puntos y se combinan en un nuevo data.frame llamado activos, manteniendo las cadenas de texto como tales y no como factores. Esto es útil para reorganizar los datos y trabajar solo con las columnas de interés.

```
activos <- data.frame(
  lat = puntos$Latitud,
  long = puntos$Longitud,
  dispo = puntos$Disponibilidad,
  planta = puntos$Planta,
  ubicacion = puntos$Ubicacion.Geografica,
    stringsAsFactors = FALSE
)
```

PASO 10. Al ejecutar este código, se generará un mapa interactivo con marcadores en las ubicaciones especificadas, mostrando la disponibilidad en un popup y la planta junto con la ubicación geográfica en una etiqueta al pasar el cursor sobre los marcadores. La conversión a UTF-8 asegura que todos los caracteres especiales se muestren correctamente en el mapa.

iconv(): Esta función convierte la codificación de caracteres de una cadena.

activos$planta, activos$ubicacion, activos$dispo: Son columnas del data.frame activos.

to = "UTF-8": Especifica que la conversión es hacia la codificación UTF-8, que es una codificación estándar que soporta todos los caracteres de todos los idiomas, lo que asegura que los textos se muestren correctamente.

leaflet(data = activos[1:299,]): Inicia un mapa leaflet con las primeras 299 filas del data.frame activos.

addTiles(): Añade una capa base de imágenes al mapa (por defecto, OpenStreetMap).

addMarkers(...): Añade marcadores al mapa con las siguientes especificaciones:lng = ~long y lat = ~lat: Indican que las columnas long y lat del data.frame activos contienen las coordenadas longitud y latitud para los marcadores.

popup = ~as.character(dispo): Define un popup que se mostrará al hacer clic en un marcador, mostrando el valor de la columna dispo convertido a texto.

label = ~paste(as.character(planta), as.character(ubicacion), sep = " - "): Crea una etiqueta para cada marcador que combina los valores de las columnas planta y ubicacion, separados por un guion (" - ").

```
activos$planta <- iconv(activos$planta, to = "UTF-8")

activos$ubicacion <- iconv(activos$ubicacion, to = "UTF-8")
activos$dispo <- iconv(activos$dispo, to = "UTF-8")


leaflet(data = activos[1:299, ]) %>%
  addTiles() %>%
  addMarkers(
    lng = ~long,
    lat = ~lat,
    popup = ~as.character(dispo),
    label = ~paste(as.character(planta), as.character(ubicacion), sep = " - ")
  )
```