Programación SIG: Python, R y Julia

Alexys H. Rodríguez-Avellaneda Ph.D.

2026-02-19

Tabla de Contenidos

Listado de Figuras

Listado de Tablas

Bienvenida

Bienvenido a la documentación del curso **Programación en SIG** de la Maestría en Geomática de la Universidad Nacional de Colombia.

Este recurso ha sido diseñado como una guía teórico-práctica para el análisis y procesamiento de datos geográficos utilizando los tres lenguajes más potentes en la ciencia de datos actual: **Python, R y Julia**.

Objetivos del curso

El objetivo principal es que el estudiante desarrolle habilidades para automatizar procesos geoespaciales y construir flujos de trabajo reproducibles.

- **Python**: Lenguaje principal de enfoque.
- **R**: Comparativa para análisis estadístico espacial.
- **Julia**: Alternativa de alto rendimiento para computación científica.

Cómo usar este libro

A lo largo de los capítulos, encontrarás ejemplos de código organizados en **pestañas (tabs)**. Aunque el énfasis de las explicaciones está en Python, puedes alternar entre lenguajes para ver cómo se implementa la misma lógica en los otros entornos:

> **Nota**
>
> Utiliza las pestañas superiores en los bloques de código para cambiar entre **Python**, **R** y **Julia**.

Requisitos previos

Para seguir este curso, se proveen guías de instalación de: - Docker Desktop (para el entorno unificado). - Editor de código VSCode. - QGIS - ArcGIS Pro - Conocimientos básicos de Sistemas de Información Geográfica (SIG) y manejo de datos vectoriales/raster.

Licencia y citación del material

Licencia de uso

Este material de curso se distribuye bajo una licencia **Creative Commons Atribución-NoComercial 4.0 Internacional (CC BY-NC 4.0)**.

> ℹ Lo que esto significa para el estudiante:
>
> Bajo los principios de **Open Science**, usted es libre de:
>
> 1. **Compartir**: Copiar y redistribuir el material en cualquier medio o formato.
> 2. **Adaptar**: Remezclar, transformar y construir sobre el material para fines académicos e investigativos.
>
> **Bajo las siguientes condiciones:**
>
> - **Atribución**: Debe dar crédito de manera adecuada, brindar un enlace a la licencia e indicar si se han realizado cambios.
> - **No comercial**: No puede hacer uso del material con fines comerciales o de lucro.

Cómo citar este material

Si utiliza este código, las guías o los datos en sus propios proyectos o investigaciones, por favor incluya las siguientes referencias:

Referencia en formato APA (7ma edición)

> Rodríguez-Avellaneda, A. H. (2026). *Programación en SIG: R, Python y Julia* [Material de curso]. Maestría en Geomática, Universidad Nacional de Colombia. Recuperado de https://github.com/alexyshr/ProgramacionSIG2026

Referencia en formato BibTeX (Para LaTeX/Zotero)

```
@misc{rodriguez2026sig,
  author = {Rodriguez-Avellaneda, Alexys H.},
  title = {Programación en SIG: R, Python y Julia},
  year = {2026},
  howpublished = {Material de curso, Maestría en Geomática},
  publisher = {Universidad Nacional de Colombia},
  url =
   ↪  {[https://github.com/alexyshr/ProgramacionSIG2026](https://github.com/alexyshr/ProgramacionSIG2026)},
  note = {Licencia CC BY-NC 4.0}
}
```

Contacto y colaboración

Este documento es una obra en construcción permanente en el marco de la ciencia abierta. Si encuentra errores, omisiones o desea proponer mejoras técnicas en los flujos de **R**, **Python** o **Julia**, puede colaborar de las siguientes formas:

1. **Abrir un Issue**: Para reportar errores en el código o en la redacción.
2. **Enviar un Pull Request**: Si desea proponer una corrección directa o una nueva funcionalidad.

Reconocimiento a herramientas de IA

Gran parte de la orquestación técnica de este entorno (especialmente la resolución de conflictos de librerías en **Docker** y la creación de funciones puente como `j_eval`) ha sido refinada con el apoyo de herramientas de Inteligencia Artificial, bajo la supervisión y validación del autor principal.

Prefacio

Este documento constituye el material de apoyo principal para el curso de **Programación en SIG** de la Maestría en Geomática. En un mundo donde la cantidad de datos geoespaciales crece de forma exponencial, la capacidad de procesar información mediante interfaces gráficas de usuario (GUI) tradicionales se vuelve insuficiente. La programación no es solo una herramienta adicional; es el lenguaje fundamental de la Geomática moderna.

Filosofía del curso: El enfoque "Políglota"

A diferencia de los cursos tradicionales centrados exclusivamente en un software o un solo lenguaje, este curso adopta una visión inspirada en el trabajo de Edzer Pebesma y Roger Bivand en *Spatial Data Science* (Pebesma & Bivand, 2023, 2025).

Nuestro enfoque es **Python-céntrico**, reconociendo su dominio en la industria y la inteligencia artificial, pero integramos **R** y **Julia** como aliados estratégicos:

- **Python**: Nuestra base para la automatización, desarrollo de scripts y flujos de trabajo en la nube.
- **R**: Para cuando el análisis requiere un rigor estadístico espacial profundo y una visualización cartográfica de alta calidad.
- **Julia**: Para computación científica de alto rendimiento donde la velocidad de ejecución es crítica.

¿Por qué tres lenguajes?

El objetivo no es que el estudiante sea un experto programador en los tres, sino que comprenda que los **motores geoespaciales** subyacentes (como GDAL, GEOS y PROJ) son los mismos. Al aprender a interactuar con ellos desde diferentes sintaxis, el estudiante desarrolla una flexibilidad mental que le permitirá adaptarse a cualquier tecnología futura.

Datos de Colombia como eje central

La teoría es universal, pero la práctica es local. Todos los ejemplos y ejercicios de este curso están diseñados utilizando datos reales del contexto colombiano:

- Capas vectoriales del **IGAC** y límites administrativos del **DANE**.
- Modelos Digitales de Elevación y productos satelitales sobre la geografía nacional.
- Sistemas de Referencia de Coordenadas basados en **MAGNA-SIRGAS**.

Estructura de la documentación

Para facilitar el aprendizaje comparado, esta guía utiliza un sistema de pestañas. En la mayoría de los capítulos, verás bloques de código organizados así:

0.0.1 Python (principal)

Explicación de la implementación en Python usando librerías como `geopandas`, `rasterio` o `shapely`.

0.0.2 R (alternativa)

Equivalente funcional utilizando `sf`, `terra` o `stars`.

0.0.3 Julia (alto rendimiento)

Implementación moderna usando `ArchGDAL.jl`, `GeomStats.jl` o `Rasters.jl`.

Reproducibilidad

Siguiendo los estándares de la ciencia abierta, todo el entorno de este curso está preconfigurado en contenedores **Docker**. Esto garantiza que el código que funciona en el computador del profesor funcionará exactamente igual en el del estudiante, eliminando el "infierno de las dependencias" de instalación.

Parte I

Instalación y Entornos

Capítulo 1

Introducción a Entornos de Desarrollo SIG

1.1 Estrategia de Trabajo: Arquitectura Híbrida

Para el desarrollo del curso, utilizaremos tres enfoques que permiten equilibrar la potencia del software de escritorio (comercial y libre) con el rigor de la ciencia de datos reproducible. Esta estructura garantiza que el estudiante adquiera competencias tanto en la automatización de software SIG tradicional como en el análisis de datos geoespaciales moderno.

1.1.1 Comparativa de Entornos

Criterio	**Opción A: Contenedor (Docker)**	**Opción B: Nativo (OSGeo4W)**	**Opción C: Nativo (ArcGIS Pro)**
Tecnología	Docker (Inc., 2025) Compose (JupyterLab).	OSGeo4W - QGIS (Rosas-Chavoya et al., 2022) LTR.	ArcGIS Pro (Law & Collins, 2019) (Conda Environment).
Uso Principal	Análisis masivo y Ciencia de Datos.	Automatización SIG con PyQGIS (Lawhead, 2017).	Geoprocesamiento y ArcPy (Toms, 2015).
Ventaja	Entorno idéntico y reproducible.	Acceso a drivers locales y QGIS.	Herramientas comerciales avanzadas.
Lenguajes	**Python (Python, 2021), R (R Core Team, 2025) y Julia (Bezanson et al., 2017)** (Preconfigurados).	**Python (PyQGIS)**.	**Python (ArcPy)**.

1.1.2 Secuencia Didáctica y Uso de Entornos

El curso seguirá una progresión lógica desde la abstracción de datos y bases de datos hacia la automatización en software especializado:

Etapa	Entorno Sugerido	Contenido Temático	Razón
Fase 1: Fundamentos y DB (Semanas 1-7)	**Opción A: Docker**	Principios de programación, estructuras básicas, Ciencia de Datos políglota y **PostGIS**.	Entorno controlado para aprender lógica de programación y gestión de bases de datos sin conflictos de instalación.
Fase 2: Extensibilidad Libre (Semanas 8-11)	**Opción B: OSGeo4W**	Desarrollo de scripts, complementos y automatización dentro de QGIS.	Uso de Python (PyQGIS) para extender las capacidades del software libre más relevante.
Fase 3: Extensibilidad Comercial (Semanas 12-15)	**Opción C: ArcGIS**	Scripts de geoprocesamiento, Toolbox y flujos de trabajo en ArcGIS Pro.	Uso de Python (ArcPy) para automatizar procesos en el entorno comercial líder.

1.1.3 Propósito según el Escenario

Escenario	**Usa la Opción A (Docker/PostGIS)**	**Usa la Opción B (QGIS/Python)**	**Usa la Opción C (ArcGIS/Python)**
Prácticas de clase	Para aprender Python y consultas SQL espaciales.	Para automatizar tareas repetitivas en QGIS.	Para crear herramientas dentro de ArcGIS Pro.
Automatización	Para flujos de análisis de datos masivos.	Para interactuar con la API de QGIS (PyQGIS).	Para interactuar con la API de ArcGIS (ArcPy).
Ecosistema	Independiente del Sistema Operativo.	Ecosistema de Código Abierto (OSGeo).	Ecosistema Comercial (ESRI).

1.2 Decisiones de Diseño del Entorno

La **Opción A** constituye el núcleo de la Ciencia de Datos Espaciales del curso. Al integrar **PostGIS** en este entorno, se garantiza que los estudiantes practiquen la persistencia de datos y el análisis SQL desde la primera fase.

Las **Opciones B y C** tienen como propósito específico el dominio de Python dentro de los entornos de software SIG de mayor relevancia, permitiendo al estudiante tener acceso a todas las funcionalidades, motores de renderizado y herramientas de geoprocesamiento nativas de QGIS y ArcGIS respectivamente.

Capítulo 2

Guía de Instalación. Opción A: Docker (Python, R, Julia)

2.1 Introducción

Para garantizar un entorno de análisis SIG reproducible y políglota, utilizaremos la orquestación de contenedores con Docker (Inc., 2025). Este enfoque permite ejecutar, de forma aislada y coordinada, tanto el motor de procesamiento como el servidor de datos.

Esta guía establece un entorno de **reproducibilidad científica**. Utilizaremos una arquitectura híbrida: **VSCode** como editor local (Microsoft Corporation, 2026) y **Docker** como laboratorio de ejecución (Inc., 2025). Esta aproximación garantiza que todos los estudiantes operen bajo las mismas versiones de motores geoespaciales (GDAL, GEOS, PROJ), eliminando el problema de "en mi computador no funciona".

Este documento detalla el procedimiento actualizado para la instalación de las herramientas Docker del curso utilizando un método **offline**. Este método es ideal para entornos con conexión limitada y garantiza la homogeneidad del software.

2.2 Preparación de herramientas en equipo local

Para preparar su equipo de trabajo, instale las siguientes herramientas básicas:

1. **VSCode IDE**: En Apéndice F encontrará el proceso de instalación y las principales extensiones (Sección F.6) necesarias a instalar en VSCode.
2. **Docker Desktop**: En Apéndice D encontrará la guía de instalación y el proceso de configuración previo (Sección D.2) para garantizar un funcionamiento óptimo.

   Importante: Antes de la instalación, siga el procedimiento descrito en Sección D.2.

2.3 Limpieza del entorno (opcional pero recomendado)

Ver Sección D.3.

2.4 Descarga y preparación de archivos

Ver Sección D.4.

2.5 Instalación de las imágenes (maquinas virtuales)

Ubique los archivos en una carpeta local que cumpla con los siguientes requisitos:

- **No** estar dentro de carpetas sincronizadas (OneDrive, Dropbox, etc.).
- Que la ruta (**path**) no contenga espacios, tildes ni caracteres especiales.
- Que sea preferiblemente una **ruta corta**.
- Evite carpetas temporales o de sistema como **Descargas** o **Temp**.

Dentro de la carpeta deben existir los archivos descargados en el paso anterior:

- `sig_unal_completo.tar` (El archivo de la imagen).
- `docker-compose.yml` (El archivo de orquestación contenido en la descarga).

Este procedimiento instalará dos imágenes (una con la base de datos espacial y otra con Python, R y Julia) y configurará una red interna para comunicar su máquina anfitriona (Windows) con las dos máquinas virtuales Linux.

Desde una terminal situada en dicha carpeta, ejecute:

```
docker load -i sig_unal_completo.tar
```

> **Nota**: El terminal quedará en modo de espera sin enviar mensajes hasta que termine. El proceso suele tardar unos 20 minutos, pero puede extenderse a más de una hora según su hardware. **No interrumpa el proceso**.

- **Paciencia**: El comando de carga es pesado. Si el proceso falla, el sistema enviará un mensaje de error; de lo contrario, simplemente espere a que la terminal le devuelva el control. No olvide limpiar el entorno para intentar nuevamente en caso de fallos en el proceso.

2.6 Puesta en marcha

Una vez terminada la carga (`load`), inicie los servicios desde la misma carpeta con:

```
docker compose up -d
```

Si la ejecución es exitosa, verá un mensaje confirmando el inicio de:

- Red: `final_default`
- Contenedor: `contenedor_postgis_unal` (basado en la imagen `postgis_unal:final`)
- Contenedor: `contenedor_sig_unal` (basado en la imagen `image_sig_unal:final`)

2.7 Iniciar contenedores

Ver Sección D.7

2.8 Cargar contenedores en VSCode

Ver Sección B.5.

2.9 Apagar los contenedores

Ver Sección D.7.

2.10 Contenido de los contenedores

Ver Sección B.3.1.

Capítulo 3

Guía de Instalación. Opción B: QGIS y PyQGIS

La **Opción B** realmente se basa en dos alternativas para la instalación de QGIS (Rosas-Chavoya et al., 2022): **OSGeo4W** y **QGIS + GEE usando Pixi.**

3.1 Opción B.1 - OSGeo4W

OSGeo4W es un entorno para Windows que agrupa y facilita la gestión de software geoespacial de código abierto como QGIS (Rosas-Chavoya et al., 2022), GDAL/OGR (Mitchell, 2015), GRASS (Neteler et al., 2012) y SAGA (Passy & Théry, 2018).

En este escenario, **Python** (Python Software Foundation, 2025) es el motor fundamental, permitiendo la automatización directa de tareas sobre el software de escritorio (Lawhead, 2017). Para mantener la estabilidad del sistema, en este entorno nativo no intentaremos vincular R (R Core Team, 2025) o Julia (Bezanson et al., 2017); para dicho propósito políglota utilizaremos exclusivamente la **Opción A (Docker)** (Inc., 2025).

> **Nota**: OSGeo4W permite instalar diversas aplicaciones de escritorio, como se muestra en la Figura 3.1. Mientras que la Tabla 3.1 se centra en las librerías binarias requeridas para QGIS, el instalador también permite integrar las demás herramientas mostradas en la figura. El último item de la figura es el instalador `Setup` que permitirá agregar mas componentes en cualquier momento.

3.1.1 Instrucciones de Instalación

1. Descargue el **Network Installer** de QGIS.
2. Ejecute y seleccione **Advanced Install**.
3. Mantenga las rutas por defecto (`C:\OSGeo4W`).
4. En el paso de **Select Packages**, busque los elementos de la tabla y marque únicamente la casilla **Bin** (Binarios).

3.1.2 Lista de paquetes a seleccionar

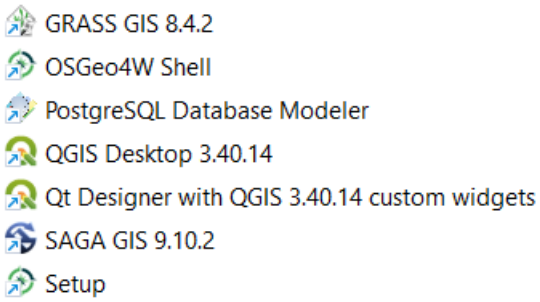


Figura 3.1: Aplicaciones Desktop OSGeo4W.

Tabla 3.1: Listado de herramientas y librerías OSGeo4W para el curso

Grupo	Paquete	Módulo	Propósito / Uso en Clase
Entorno Core	`qgis-ltr` / `qgis-ltr-full`	Todo	Software base y metapaquete de dependencias.
Bases de Datos	`pgmodeler`	5	Diseño visual de modelos para PostGIS.
Bases de Datos	`libspatialite`	5	Motor de base de datos espacial liviano para verificación en shell.
Bases de Datos	`python3-psycopg2`	5	Adaptador de Python para PostgreSQL/PostGIS.
Geoprocesamiento	`grass`	6	Motor para análisis topológico e hidrología.
Geoprocesamiento	`saga`	7	Algoritmos de terreno y geomorfometría.
Geoprocesamiento	`gdal`	6	Lectura/escritura de formatos raster y vector.
Ciencia de Datos	`python3-numpy`	2	Manejo de estructuras Arrays y Matrices.
Ciencia de Datos	`python3-pandas`	3	Análisis exploratorio de datos (EDA).
Ciencia de Datos	`python3-geopandas`	3	Extensión espacial para GeoDataFrames.
Visualización	`python3-matplotlib`	3	Gráficos básicos y mapas estáticos.
Visualización	`python3-seaborn`	3	Gráficos estadísticos avanzados.
Desarrollo	`python3-pip`	1	Gestor para instalar librerías adicionales.

Grupo	Paquete	Módulo	Propósito / Uso en Clase
Desarrollo	`python3-jupyterlab`	Todo	Entorno interactivo para prototipado rápido (Wijayaningrum et al., 2022).
Motores SIG	`proj` / `geos`	6	Cálculos de proyecciones y geometría.
Machine Learning	`python3-scikit-learn`	6	Modelado predictivo espacial.

! Aceptación de Dependencias Adicionales

Al avanzar, el instalador mostrará la ventana **"Unmet Dependencies"**. Es obligatorio aceptar todos los paquetes sugeridos para contar con los controladores de formatos `.ecw` y `.sid` requeridos por la cartografía oficial nacional. Para que el comando `spatialite` funcione en la shell, asegúrese de marcar el paquete `libspatialite` en la sección **Libs**.

3.1.3 Verificación de la Instalación

Abra la **OSGeo4W Shell** y ejecute los comandos para verificar la correcta integración de Python y los motores SIG:

```
python3 --version
gdalinfo --version
spatialite --version
```

3.1.3.1 Corrección de Errores al Lanzar QGIS

Si al iniciar QGIS aparece un error crítico indicando `ModuleNotFoundError: No module named 'gdal'`, se debe a una sintaxis de importación obsoleta en ciertos complementos (Rosas-Chavoya et al., 2022).

Error identificado: `File ".../agknow_utils.py", line 29, in <module> import gdal, osr`

Solución: Debe modificar el archivo del plugin para usar el espacio de nombres de OSGeo:

1. Localice el archivo en: `%AppData%\QGIS\QGIS3\profiles\default\python\plugins\agknow_qgis\agknow_utils.py`.
2. Reemplace la línea 29:
   - **Incorrecto:** `import gdal, osr`
   - **Correcto:** `from osgeo import gdal, osr`

i Nota sobre R y Julia

Para el uso intensivo de R y Julia, se recomienda la **Opción A (Docker)** (Inc., 2025) ya que viene con las librerías espaciales pre-configuradas, evitando errores de vinculación de DLLs comunes en instalaciones nativas de Windows.

3.2 Opción B.2: QGIS + GEE usando Pixi

Antes de comenzar, es indispensable contar con una cuenta de **Google Cloud** habilitada para **Google Earth Engine (GEE)**. Puede dar de alta su cuenta en earthengine.google.com.

> ❗ Requisito de Google Cloud
>
> Asegúrese de haber creado un proyecto en la consola de Google Cloud y de tener activadas las **APIs for Earth Engine** para dicho proyecto; de lo contrario, el proceso de autenticación fallará.

Video gúia de creación y registro en GEE

3.2.1 Instalación de Pixi

Pixi es un gestor de paquetes extremadamente rápido basado en el ecosistema de Conda, pero mucho más ligero. Permite crear entornos aislados sin romper las librerías de su sistema operativo.

3.2.1.1 En Linux / macOS (bash/zsh)

Ejecute el siguiente comando en su terminal:

```
curl -fsSL https://pixi.sh/install.sh | sh
```

Nota: Cierre y vuelva a abrir su terminal para que `pixi` se añada a su `PATH`. Confirme la instalación con:

```
pixi --version
```

3.2.1.2 En Windows (PowerShell)

Abra **PowerShell** (no requiere permisos de Administrador) y ejecute:

```
powershell -ExecutionPolicy Bypass -c "irm -useb https://pixi.sh/install.ps1 | iex"
```

Reinicie PowerShell y confirme la versión:

```
pixi --version
```

3.2.2 Creación del Proyecto

Navegue hasta la carpeta (**diferente a todas las utilizadas hasta el momento**) donde desea guardar sus trabajos y cree un nuevo proyecto llamado `geocd`:

```
pixi init geocd
cd geocd
```

3.2.3 Configuración del Entorno Geográfico

Instalaremos un stack potente que incluye **QGIS** y las herramientas necesarias para trabajar con **Google Earth Engine** y datos ráster/vectoriales en Python.

Desde la carpeta `geocd`, ejecute:

```
pixi add qgis geemap geopandas xee rioxarray
```

> ¿Qué estamos instalando?
>
> - **QGIS**: El software SIG profesional de uso libre mas popular.
> - **geemap**: Librería para visualización interactiva de GEE.
> - **geopandas**: Gestión de datos vectoriales.
> - **xee & rioxarray**: Motores para leer cubos de datos y archivos ráster.

3.2.4 Autenticación de Earth Engine

Finalmente, debe vincular su entorno local con su cuenta de Google Cloud. Este comando abrirá una ventana en su navegador para autorizar el acceso:

```
pixi run earthengine authenticate
```

Siga las instrucciones en pantalla, seleccione su proyecto de Google Cloud y copie el código de verificación si el sistema se lo solicita. ¡Ya está listo para procesar datos satelitales a escala global!

3.2.5 Trabajar GEE dentro de QGIS

Una vez que el entorno está configurado y autenticado, el siguiente paso es integrar **Google Earth Engine** directamente en la interfaz de QGIS. Para esto, utilizaremos el complemento desarrollado por el equipo de **OpenGeos**: El Plugin: **GEE Data Catalog**. Este complemento permite navegar por los miles de datasets de Earth Engine (Sentinel, Landsat, MODIS, etc.) y cargarlos en el lienzo de QGIS con un solo clic, utilizando el motor de procesamiento en la nube.

3.2.6 Recursos Adicionales

- **Repositorio oficial:** opengeos/qgis-gee-data-catalogs-plugin
- Video Guía **Earth Engine Data Catalogs Plugin for QGIS**: aquí

- Video Guía **Time Series Satellite Images in Seconds**: aquí

3.3 Otras referencias importantes

- PyQGIS cookbook en markdown y Jupyter notebook formats (Wu, 2023a)
- QGIS Notebook Plugin: Integrate Jupyter Notebooks into QGIS (Wu, 2023b)

Capítulo 4

Guía de Instalación. Opción C: ArcGIS Pro y ArcPy

4.1

Falta

Parte II

Fundamentos

Capítulo 5

Variables y tipos de datos

5.1 Funciones j_eval y j_plot en R

```
# #| include: false
source("./docs/j_eval_j_plot.r")
```

5.2 Introducción

En el análisis espacial, un dato nunca es solo un número o una letra; es la representación digital de un fenómeno físico. La latitud de una estación GPS, el nombre de un municipio en una tabla de atributos o la firma espectral de una imagen satelital deben almacenarse en la memoria del computador de una forma específica para poder procesarse.

Una **variable** actúa como un contenedor etiquetado en la memoria del sistema, permitiéndonos guardar, recuperar y manipular esa información. Sin embargo, la forma en que **Python**, **R** y **Julia** gestionan estos contenedores varía significativamente.

Entender la diferencia entre un entero (`int`) y un decimal de doble precisión (`float64`) no es solo una curiosidad informática: en geomática, usar el tipo de dato incorrecto puede significar la diferencia entre localizar un punto con precisión milimétrica o desviarlo por cientos de metros debido a errores de redondeo. Este capítulo sienta las bases para manipular la "materia prima" de cualquier Sistema de Información Geográfica (SIG).

5.3 Objetivos de aprendizaje

Al finalizar este capítulo, el estudiante estará en capacidad de:

1. **Definir y asignar variables** correctamente en Python, R y Julia, comprendiendo las diferencias de sintaxis y asignación de memoria.
2. **Diferenciar tipos de datos fundamentales** (enteros, flotantes, cadenas y lógicos) y seleccionar el más adecuado para optimizar el almacenamiento de datos masivos (Big Data Geoespacial).

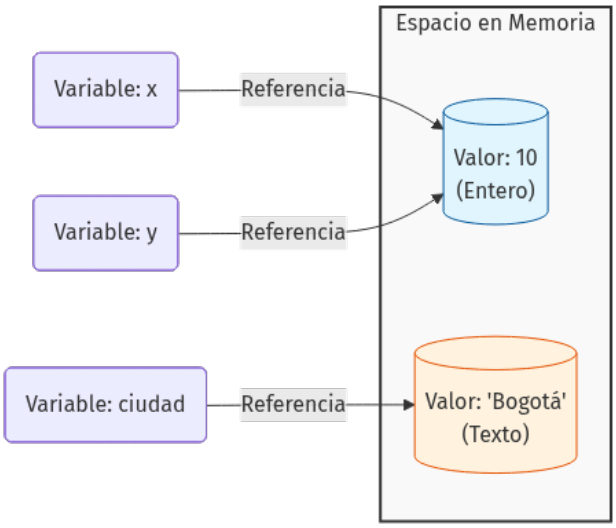


Figura 5.1: Abstracción de una Variable: En lenguajes modernos como Python o R, funciona más como una etiqueta que apunta a un objeto en memoria.

3. **Manipular cadenas de texto** para la limpieza y estandarización automática de tablas de atributos (corrección de mayúsculas, eliminación de espacios, interpolación).
4. **Aplicar formatos de precisión** para controlar la salida de coordenadas y datos numéricos en reportes técnicos.
5. **Comprender la mutabilidad**, distinguiendo cuándo una operación modifica los datos originales (Python/Julia) y cuándo genera una copia nueva (R).

5.4 Variables

Tabla 5.1: Comparación de sintaxis básica entre Python, R y Julia

Lenguaje	Operador de asignación	Comando para imprimir	Descripción y Alternativas
Python	`=`	`print()`	El operador `=` es el estándar único para asignación. Para imprimir, `print()` añade automáticamente un salto de línea. En entornos interactivos, se puede usar `display()` para una representación visual más rica.
R	`<-`	`print()`	Aunque `=` funciona, se prefiere `<-`. Además de `print()`, existe `cat()`, útil para concatenar texto sin mostrar índices de vector.
Julia	`=`	`println()`	Se usa `=` para asignación. `println()` imprime con salto de línea, mientras que `print()` lo hace sin él. Al igual que en Python, existe `display()`.

5.4.1 Python

```
# #| eval: false
#Variable numérica
numero_dptos = 32

#Imprimir
print(numero_dptos)
```

```
32
```

```
#Ver contenido (en modo interactivo)
numero_dptos
```

```
32
```

5.4.2 R

```
# #| eval: false
# Variable numérica (en R se prefiere el operador <-)
numero_dptos <- 32

# Imprimir
print(numero_dptos)
```

```
[1] 32
```

```
# Ver contenido
numero_dptos
```

```
[1] 32
```

5.4.3 Julia

```
# #| eval: false
j_eval('
# Variable numérica
numero_dptos = 32

# Imprimir
println(numero_dptos)

# Ver contenido
numero_dptos

atributos_elemento
')
```

```
Starting Julia ...
```

5.4.4 Asignación de variables

Tabla 5.2: Estructuras de datos y comportamiento de asignación

Lenguaje	Colección Base	Operador	Descripción y Flexibilidad
Python	Lista `[]`	`=`	Permite mezclar tipos en una lista. Es dinámico y **fuerte**: no permite operaciones inválidas entre tipos (ej. sumarle un texto a un número).
R	Vector `c()`	`<-`	El vector atómico (`c`) exige que todos los elementos sean del mismo tipo. Si se mezclan, R los convierte automáticamente (coerción).
Julia	Arreglo `[]`	`=`	Muy similar a la lista de Python pero optimizado para rendimiento. Es dinámico pero permite declarar tipos para ganar velocidad.

5.4.5 Conceptos clave de programación

Para entender cómo estos lenguajes manejan la información de la Tabla 5.2, es fundamental diferenciar los sistemas de tipado:

1. **Tipado Dinámico:** El tipo de la variable se define en tiempo de ejecución. No es necesario declarar que una variable es un entero o un texto antes de usarla; el lenguaje lo infiere. (Python, R y Julia son dinámicos).
2. **Tipado Estático:** El tipo de la variable debe definirse al momento de escribir el código (ej. C++ o Java). Una vez definida como "entero", no puede guardar texto. Esto previene errores antes de ejecutar el programa.
3. **Tipado Fuerte:** El lenguaje no permite operaciones entre tipos incompatibles sin una conversión explícita. Por ejemplo, en **Python**, `5 + "10"` arrojará un error.
4. **Tipado Débil:** El lenguaje intenta realizar conversiones automáticas (coerción) para que la operación funcione. En **R**, si intentas unir un número y un texto en un vector `c(1, "Bogotá")`, R convertirá el `1` en texto `"1"` silenciosamente.

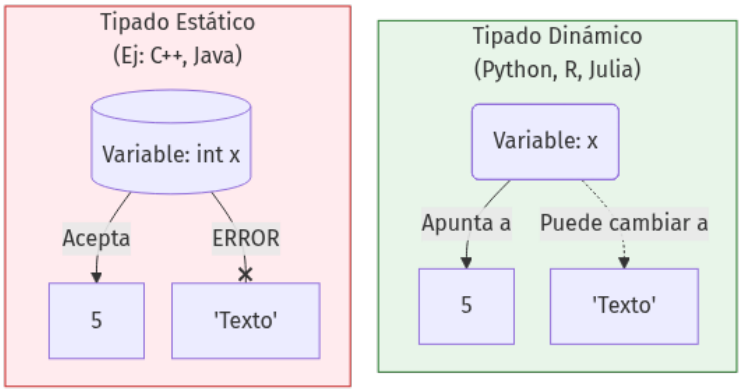


Figura 5.2: Comparativa de Tipado: La flexibilidad tiene un costo en seguridad, pero gana en velocidad de desarrollo.

5.4.5.1 Python

```python
# #| eval: false
# Iniciar con un número (Latitud de Monserrate)
datos_ubicacion = 4.6052

# Cambiar a texto (Nombre del lugar)
datos_ubicacion = "Cerro de Monserrate"

# Cambiar a una lista de coordenadas [Lat, Lon]
datos_ubicacion = [4.6052, -74.0554]

# Imprimir resultado final
print(datos_ubicacion)
```

```
[4.6052, -74.0554]
```

5.4.5.2 R

```r
# #| eval: false
# Iniciar con un número (Latitud de Monserrate)
datos_ubicacion <- 4.6052

# Cambiar a texto (Nombre del lugar)
datos_ubicacion <- "Cerro de Monserrate"

# Cambiar a un vector de coordenadas (c es para combinar/concatenar)
datos_ubicacion <- c(4.6052, -74.0554)

# Imprimir resultado final
print(datos_ubicacion)
```

```
[1]   4.6052 -74.0554
```

5.4.5.3 Julia

```julia
# #| eval: false
j_eval('
# Iniciar con un número (Latitud de Monserrate)
datos_ubicacion = 4.6052

# Cambiar a texto (Nombre del lugar)
datos_ubicacion = "Cerro de Monserrate"

# Cambiar a un arreglo (Array) de coordenadas
datos_ubicacion = [4.6052, -74.0554]

# Imprimir resultado final
println(datos_ubicacion)
')
```

5.5 Nombres para las variables

¿Por qué es clave usar nombres adecuados?

En programación, los nombres que eliges para tus variables no son solo etiquetas; son las instrucciones que permiten que otras personas (y tú mismo en el futuro) entiendan la lógica del código sin errores. Usar nombres estandarizados es vital por las siguientes razones:

1. **Evitar errores técnicos:** Los lenguajes de programación son muy estrictos. Un espacio, una tilde o

una `ñ` pueden hacer que un programa que funciona bien en tu computadora falle al abrirlo en otra.

2. **Claridad en la lectura:** Es mucho más fácil entender un proceso si la variable se llama `distancia_metros` en lugar de simplemente `d`.
3. **Portabilidad:** El uso de estándares internacionales asegura que tus datos e investigaciones puedan integrarse fácilmente entre diferentes plataformas como **Python**, **R** o **Julia**.

> **Resumen de Reglas de Estilo (snake_case)**
>
> Para que tus programas sean profesionales y compatibles, sigue estas reglas básicas:
>
> - **Todo en minúsculas:** Evita mezclar mayúsculas para mantener la uniformidad.
> - **Usa el guion bajo (`_`):** Dado que los espacios están prohibidos, el guion bajo une las palabras (ej. `precio_gasolina`).
> - **Solo letras básicas y números:** Usa solo caracteres de la `a` a la `z`. Evita tildes, eñes o símbolos especiales (`!`, `#`, `$`).
> - **Prohibido el guion medio (`-`):** El computador lo interpreta como una operación de resta.

```
#| code-summary: "Ejemplos comparativos (Python, R, Julia)"

# FORMA CORRECTA: Clara y sin errores
estacion_climatologica = "Dorado"
temperatura_celsius = 20.5
conteo_puntos_gps = 15

# FORMA INCORRECTA: Genera errores o confusión
estacion climatologica = "Dorado"  # ERROR: Los espacios no están permitidos
temperatura-celsius = 20.5         # ERROR: El guion medio intenta restar
t = 20.5                           # MAL: Es una variable muy vaga
año_inicio = 2024                  # EVITAR: La 'ñ' causa fallos de lectura
```

5.5.1 Python

```
# #| eval: false
# Nombres de variables recomendados
# Variables geoespaciales claras
latitud_bogota = 4.7110
longitud_bogota = -74.0721
altura_msnm = 2640.0
municipio_nombre = "Chinchiná"
conteo_viviendas_afectadas = 150
sistema_referencia = "MAGNA-SIRGAS / Origen Nacional"


# Nombres de variables NO recomendados
# Evitar nombres genéricos o confusos
x = 4.7110  # ¿Es latitud o un índice?
d = "Chinchiná"  # Muy vago (¿Distrito, Departamento, Dato?)
val = 2640  # Ambiguo: ¿Valor, Valencia, Variable?
coords = [4.71, -74.07]  # Lista sin etiquetas claras
```

5.5.2 R

```
# #| eval: false
# Nombres de variables recomendados
# Estilo snake_case (común en R para análisis de datos)
latitud_medellin <- 6.2442
longitud_medellin <- -75.5812
cota_terreno <- 1495
```

```r
nombre_departamento <- "Antioquia"
poblacion_censo_2018 <- 6407000
proyeccion_cartografica <- "EPSG:9377"


# Nombres de variables NO recomendados
# Evitar abreviaturas extremas
l <- 6.2442 # ¿Latitud, Longitud, Límite, Localidad?
nom <- "Antioquia" # "nombre" es mejor
temp <- 1495 # ¿Temperatura o un archivo temporal?
p <- 6407000 # ¿Población, Perímetro, Pendiente, P-value?
```

5.5.3 Julia

```r
# #| eval: false
j_eval('
# Nombres de variables recomendados
# Aprovechando el soporte Unicode de Julia
latitud_cali = 3.4516
longitud_cali = -76.5320
elevación_cauca = 995.0
nombre_vereda = "La Elvira"
densidad_poblacional = 120.5
referencia_espacial = "MAGNA-SIRGAS"
')
```

```r
j_eval('
# Nombres de variables NO recomendados
# Nombres que no dicen nada del contexto geográfico
var1 = 3.4516
info = "La Elvira"
tmp = 995.0 # Típico error: ¿Temperatura o Temporal?
lista = [3.45, -76.53] # Difícil de leer en modelos de optimización
')
```

5.6 Tipos de datos

5.6.1 Python

```r
# #| eval: false

# Representa el número de elementos en un dataset geoespacial
num_elementos = 500 # Igual a num_elementos = int(500)

# Validación de tipo (Type Checking)
# Lanza un error si no es un entero, ideal para debugging en scripts de Geomatización
if not isinstance(num_elementos, int):
    raise TypeError(f"Se esperaba un entero, pero se recibió {type(num_elementos)}")

# Representa la altitud de un punto (Nevado del Ruiz)
import numpy as np
altitud_32 = np.float32(2640.5) # np.float32() para simple
altitud_64 = 2640.5   # float() es el defecto (doble)

# Representa la latitud de un punto (Nevado del Ruiz)
latitud = 4.8920

# Representa la longitud de un punto
longitud = -75.3188

# Representa el sistema de referencia oficial de Colombia
sistema_coordenadas = "MAGNA-SIRGAS / Origen Nacional"

# Representa si el dataset está georreferenciado o no
esta_georeferenciado = True

# Una lista que representa latitud y longitud
```

```python
coordenadas = [4.8920, -75.3188]

# Un diccionario con los atributos del elemento geográfico
atributos_elemento = {
    "nombre": "Nevado del Ruiz",
    "altura_msnm": 5321,
    "tipo": "Estratovolcán",
    "ubicacion": [4.8920, -75.3188],
}

print(atributos_elemento)
```

```
{'nombre': 'Nevado del Ruiz', 'altura_msnm': 5321, 'tipo': 'Estratovolcán', 'ubicacion': [4.892, -75.3188]}
```

```python
# Tipo de dato
type(atributos_elemento)
```

```
<class 'dict'>
```

```python
# Acceder a la primera coordenada (índice 0) dentro de la clave 'ubicacion'
latitud_ruiz = atributos_elemento["ubicacion"][0]
latitud_ruiz
```

```
4.892
```

5.6.2 R

```r
# #| eval: false

# Representa el número de elementos (L indica tipo integer)
num_elementos <- 500L  # Diferente a num_elementos = 500 (numeric)

# Validación del tipo de dato
if (!is.integer(num_elementos)) {
  stop(paste("Error: Se esperaba un tipo 'integer', pero se recibió uno de tipo '",
             class(num_elementos), "'.", sep = ""))
}

# Representa la altitud de un punto (Nevado del Ruiz)
altitud_32 <- as.single(5321.0) # as.single() para precisión simple (32 bits)
altitud_64 <- 5321.0            # double es el defecto en R (64 bits)

# Representa la latitud de un punto (Nevado del Ruiz)
latitud <- 4.8920

# Representa la longitud de un punto
longitud <- -75.3188

# Representa el sistema de referencia oficial (tipo character)
sistema_coordenadas <- "MAGNA-SIRGAS / Origen Nacional"

# Representa si el dataset está georreferenciado (logical)
esta_georeferenciado <- TRUE

# Un vector que representa latitud y longitud
coordenadas <- c(4.8920, -75.3188)

# Una lista que representa los atributos del elemento geográfico
atributos_elemento <- list(
    "nombre" = "Nevado del Ruiz",
    "altura_msnm" = 5321,
    "tipo" = "Estratovolcán",
    "ubicacion" = c(4.8920, -75.3188)
)

print(atributos_elemento)
```

```
$nombre
[1] "Nevado del Ruiz"

$altura_msnm
[1] 5321

$tipo
[1] "Estratovolcán"

$ubicacion
[1]   4.8920 -75.3188
```

```r
# Tipo de dato
class(atributos_elemento)
```

```
[1] "list"
```

```r
# Acceder a la latitud (índice 1 en R) dentro del elemento 'ubicacion'
latitud_ruiz <- atributos_elemento$ubicacion[1]
latitud_ruiz
```

```
[1] 4.892
```

5.6.3 Julia

```
# #| eval: false
j_eval('
# Representa el número de elementos en el dataset
num_elementos = 500  # Igual a num_elementos = Int64(500)
typeof(num_elementos)

# Validación del tipo de dato
if !(num_elementos isa Int64)
    error("Error: Se esperaba un \'Int64\', pero se recibió un \'$(typeof(num_elementos))\'.")
end

# Representa la altitud de un punto (Nevado del Ruiz)
altitud_32 = Float32(5321.0) # Float32() para precisión simple (32 bits)
altitud_64 = 5321.0          # Float64 es el defecto en Julia (64 bits)

# Representa la latitud del punto (Nevado del Ruiz)
latitud = 4.8920

# Representa la longitud del punto
longitud = -75.3188

# Representa el sistema de referencia espacial
sistema_coordenadas = "MAGNA-SIRGAS / Origen Nacional"

# Representa el estado de georreferenciación (tipo Bool)
esta_georeferenciado = true

# Un arreglo que representa latitud y longitud
coordenadas = [4.8920, -75.3188]

# Un diccionario (Dict) para almacenar los atributos
atributos_elemento = Dict(
    "nombre" => "Nevado del Ruiz",
    "altura_msnm" => 5321,
    "tipo" => "Estratovolcán",
    "ubicacion" => [4.8920, -75.3188],
)
```

```
println(atributos_elemento)

# Tipo de dato
typeof(atributos_elemento)

# Acceder a la latitud (índice 1 en Julia) dentro de la clave "ubicacion"
latitud_ruiz = atributos_elemento["ubicacion"][1]
latitud_ruiz
')
```

Tabla 5.3: Comparación detallada de sintaxis y tipos de datos

Característica / Variable	Python	R	Julia
Sufijo `L` (num_elementos)	No se usa. `500` es entero.	Se usa `500L` para `integer`. Sin `L` es `numeric`.	No se usa. `500` es `Int64`.
Asignación en Diccionario	`v = {clave: valor, ...}` (Usa `:`).	`v <- list(clave = valor, ...)`. (Usa `=`)	`v = Dict(clave => valor, ...)` (Usa `=>`).
`num_elementos`	`int` (Automático)	`integer` (Requiere `L`)	`Int64` (Automático)
`latitud` (doble precisión por defecto)	`float`	`numeric`	`Float64`
`altitud_32` (simple precisión)	`np.float32()`	`as.single()`	`Float32`
`sistema_coordenadas`	`str`	`character`	`String`
`esta_georeferenciado`	`bool` (`True` / `False`)	`logical` (`TRUE` / `FALSE`)	`Bool` (`true` / `false`)
`coordenadas`	`list [ , ]`: Colección mutable de objetos.	`vector c( , )`: Colección indexada de elementos del mismo tipo.	`Array [ , ]` (Vector): Arreglo indexado y optimizado para cómputo.
`atributos_elemento`	`dict { : }`: Estructura nativa de pares clave-valor.	`list list( = )`: Lista con nombres (etiquetas) para cada elemento.	`Dict Dict( => )`: Tipo de dato optimizado para mapeos clave-valor.

5.6.4 ¿Qué es un diccionario?

Un **Diccionario** es una estructura de datos que organiza la información mediante pares de **Clave-Valor** (*Key-Value*). A diferencia de las secuencias indexadas (como las listas o vectores) donde los elementos se recuperan por su posición numérica, en un diccionario se accede a la información a través de etiquetas únicas llamadas claves.

- **Identificación:** Cada valor almacenado debe tener una clave asociada que funciona como su identificador único.
- **Heterogeneidad:** Permite agrupar diversos tipos de datos (cadenas, números, arreglos) bajo un mismo objeto.
- **Acceso Directo:** Su implementación técnica permite que la recuperación de un dato sea extremadamente eficiente, sin importar el volumen de información.
- **Estándares:** Es el concepto fundamental detrás de formatos como JSON, facilitando la organización

jerárquica de los datos.

5.7 Caracteres de escape

5.7.1 Python

```
# #| eval: false

# Salto de línea para separar información de capas geográficas
print("Capa: Departamentos_Colombia\nEstado: Procesada exitosamente.")
```

```
Capa: Departamentos_Colombia
Estado: Procesada exitosamente.
```

```
# Salto de línea y tabulación para jerarquizar metadatos
print("Metadatos del proyecto:\n\tAutor: IGAC\n\tEscala: 1:100.000")
```

```
Metadatos del proyecto:
    Autor: IGAC
    Escala: 1:100.000
```

```
# Uso de comillas simples dentro de dobles para nombres propios
print("Nombre del archivo: 'Mapa_Relieve_Colombia.shp'")
```

```
Nombre del archivo: 'Mapa_Relieve_Colombia.shp'
```

5.7.2 R

```
# #| eval: false

# Salto de línea para separar información de capas geográficas
print("Capa: Departamentos_Colombia\nEstado: Procesada exitosamente.")
```

```
[1] "Capa: Departamentos_Colombia\nEstado: Procesada exitosamente."
```

```
# Salto de línea y tabulación para jerarquizar metadatos (cat interpreta mejor los caracteres)
cat("Metadatos del proyecto:\n\tAutor: IGAC\n\tEscala: 1:100.000")
```

```
Metadatos del proyecto:
    Autor: IGAC
    Escala: 1:100.000
```

```
# Uso de comillas simples dentro de dobles para nombres propios
print("Nombre del archivo: 'Mapa_Relieve_Colombia.shp'")
```

```
[1] "Nombre del archivo: 'Mapa_Relieve_Colombia.shp'"
```

5.7.3 Julia

```
# #| eval: false
j_eval('
# Salto de línea para separar información de capas geográficas
println("Capa: Departamentos_Colombia\nEstado: Procesada exitosamente.")

# Salto de línea y tabulación para jerarquizar metadatos
println("Metadatos del proyecto:\n\tAutor: IGAC\n\tEscala: 1:100.000")

# Uso de comillas simples dentro de dobles para nombres propios
println("Nombre del archivo: \'Mapa_Relieve_Colombia.shp\'")
')
```

5.7.4 Caracteres especiales de escape

Tabla 5.4: Caracteres de escape comunes en lenguajes de programación

Carácter	Nombre	Descripción Técnica
`\n`	Salto de línea (Newline)	Mueve el cursor al inicio de la siguiente línea para organizar texto.
`\t`	Tabulación (Tab)	Inserta un espacio horizontal para crear jerarquías o sangrías.
`\\`	Barra invertida	Permite imprimir el carácter `\` (crucial para rutas de archivos en sistemas locales).
`\"`	Comilla doble	Permite insertar comillas dobles sin cerrar prematuramente la cadena de texto.
`\'`	Comilla simple	Permite insertar comillas simples; en Julia/R dentro de `j_eval` suele requerir escape extra.

5.8 Comentarios

5.8.1 Python

```
# #| eval: false

# Comentario de bloque: Información de procesamiento
print("Capa: Departamentos_Colombia\nEstado: Procesada exitosamente.")
```

```
Capa: Departamentos_Colombia
Estado: Procesada exitosamente.
```

```
# Comentario en línea: RMSE en metros
rmse = 0.05  # Tolerancia para precisión submétrica

# El intérprete lo procesa (""") como un string,
# pero al no asignarse a una variable, actúa como comentario.
"""
Este es un comentario multilínea en Python.
Se utilizan triples comillas (aunque técnicamente son cadenas
de texto no asignadas, se usan para documentar bloques extensos).
"""
```

'\nEste es un comentario multilínea en Python.\nSe utilizan triples comillas (aunque técnicamente son ca

```
print("Nombre del archivo: 'Mapa_Relieve_Colombia.shp'")
```

```
Nombre del archivo: 'Mapa_Relieve_Colombia.shp'
```

5.8.2 R

```
# #| eval: false

# Comentario de bloque: Información de procesamiento
print("Capa: Departamentos_Colombia\nEstado: Procesada exitosamente.")
```

```
[1] "Capa: Departamentos_Colombia\nEstado: Procesada exitosamente."
```

```
# Comentario en línea: RMSE en metros
rmse <- 0.05  # Tolerancia para precisión submétrica

# R no tiene un operador nativo para comentarios multilínea.
# Se deben usar múltiples numerales consecutivos para
# documentar bloques de código extensos.
cat("Metadatos del proyecto:\n\tAutor: IGAC\n\tEscala: 1:100.000")
```

```
Metadatos del proyecto:
    Autor: IGAC
    Escala: 1:100.000
```

5.8.3 Julia

```
# #| eval: false
j_eval('
# Comentario de bloque: Información de procesamiento
println("Capa: Departamentos_Colombia\nEstado: Procesada exitosamente.")

# Comentario en línea: RMSE en metros
rmse = 0.05  # Tolerancia para precisión submétrica

#=
Este es un comentario multilínea nativo en Julia.
Permite comentar grandes bloques de código o explicaciones
extensas sin necesidad de colocar numerales en cada línea.
=#

# En Julia las tres comillas sirven para
# definir archivos de texto en memoria (no son comentarios)
csv_data = """
municipio,altitud
Bogota,2625
Medellin,1495
Cali,1018
Quibdo,43
"""
csv_data

println("Nombre del archivo: \'Mapa_Relieve_Colombia.shp\'")
')
```

5.8.4 Documentación y caracteres especiales

Tabla 5.5: Sintaxis de documentación, escape y cadenas especiales

Carácter	Nombre	Función Técnica
`#`	Numeral	Inicia comentarios de una sola línea en los tres lenguajes.
`\n`	Newline	Inserta un salto de línea dentro de una cadena de texto.
`\t`	Tab	Inserta una tabulación horizontal (sangría).
`"""`	Triple comilla	**Python:** Documentación de bloques (*docstrings*). **Julia:** Cadenas multilínea (útil para definir archivos o bloques de texto en memoria). **R:** No se utiliza (causa error de sintaxis).
`#= =#`	Block comment	Delimitador nativo de **Julia** para comentarios multilínea.

Nota técnica: En el desarrollo de herramientas automatizadas, los comentarios multilínea son el estándar para definir el **encabezado del script**, detallando licencias, el sistema de referencia de coordenadas (CRS) y las dependencias. Es fundamental diferenciar entre un comentario real y una cadena de texto: mientras que `#` y `#= =#` son ignorados por el compilador, las triples comillas `"""` en Python y Julia generan objetos en la memoria. En Python, estas actúan como comentarios "de facto" solo si no se asignan a una variable (siendo la base de los *docstrings*), mientras que en Julia su propósito principal es la creación de bloques de texto estructurado o archivos en memoria.

5.9 Trabajando con variables y tipos de datos

5.9.1 Python

```python
# #| eval: false

import math

# Inicialización de variables para asegurar la ejecución
num_elementos = 500

# Incrementar el número de elementos existentes
num_elementos += 20
num_elementos
```

520

```python
# Convertir la latitud de grados decimales a radianes
latitud_ruiz = 4.8920
latitud_radianes = math.radians(latitud_ruiz)

# Latitud, Longitud del Nevado del Ruíz
coordenadas = [4.8920, -75.3188]

# Agregar datos de Bogotá al FINAL de la lista
coordenadas.append(4.6097)   # Agrega Latitud de Bogotá al final
coordenadas.append(-74.0817)  # Agrega Longitud de Bogotá al final

# --- Uso de INSERT ---
# Supongamos que queremos insertar el nombre del país al principio de la lista
# El método .insert(índice, valor) desplaza los demás elementos
coordenadas.insert(0, "Colombia") # Inserta en la posición inicial (índice 0)

# Ver el resultado final
```

```
coordenadas
```

```
['Colombia', 4.892, -75.3188, 4.6097, -74.0817]
```

```
# Acceso y formateo de cadenas usando el diccionario de atributos
nombre_volcan = atributos_elemento["nombre"]
altura_volcan = atributos_elemento["altura_msnm"]
mensaje = f"{nombre_volcan} tiene una altura de {altura_volcan} metros."

# Mostrar el resultado final
mensaje
```

```
'Nevado del Ruiz tiene una altura de 5321 metros.'
```

5.9.2 R

```
# #| eval: false

# Inicialización de variables necesarias
num_elementos <- 500
# Incrementar el número de elementos
num_elementos <- num_elementos + 20
num_elementos
```

```
[1] 520
```

```
# Convertir latitud a radianes
latitud_ruiz <- 4.8920
latitud_radianes <- latitud_ruiz * (pi / 180)

# Latitud, Longitud del Nevado del Ruíz
coordenadas <- c(4.8920, -75.3188)

# --- Uso de c() para combinar (Equivalente a append) ---
# Agregar elementos a un vector (creando una copia combinada)
# Adicionamos coordenadas de Bogotá al final
coordenadas <- c(coordenadas, 4.6097, -74.0817)

# --- ADVERTENCIA DE COERCIÓN ---
# Si intentamos insertar un texto en este vector numérico,
# R convertirá TODO el vector a tipo 'character'.
# coordenadas <- c("Colombia", coordenadas) # ¡Cuidado con esto!

# --- Uso de append() con after (Equivalente a insert en Python) ---
# Insertar "Colombia" al inicio (after = 0)
# ¡ADVERTENCIA!: Al insertar un texto, R convertirá todos los números
# del vector a tipo 'character' (coerción).
coordenadas <- append(coordenadas, "Colombia", after = 0)

# Acceso a lista y formateo de cadenas
nombre_volcan <- atributos_elemento$nombre
altura_volcan <- atributos_elemento$altura_msnm
mensaje <- paste(nombre_volcan, "tiene una altura de", altura_volcan, "metros.", sep = " ")
mensaje
```

```
[1] "Nevado del Ruiz tiene una altura de 5321 metros."
```

```
# Si no quieres NINGÚN espacio, puedes usar paste0() o sep = ""
mensaje <- paste(nombre_volcan, " tiene una altura de ", altura_volcan, " metros.", sep = "")
mensaje
```

```
[1] "Nevado del Ruiz tiene una altura de 5321 metros."
```

```r
mensaje <- paste0(nombre_volcan, " tiene una altura de ", altura_volcan, " metros.")
mensaje
```

```
[1] "Nevado del Ruiz tiene una altura de 5321 metros."
```

```r
# El formato define exactamente qué caracteres van entre las variables
# %s para texto (string), %d para enteros, %f para decimales
# La máscara de texto define el espacio natural entre las variables
mensaje <- sprintf("%s tiene una altura de %d metros.", nombre_volcan, altura_volcan)
mensaje
```

```
[1] "Nevado del Ruiz tiene una altura de 5321 metros."
```

```r
# Interpolación con más de una variable entera, texto o decimal
# Definición de variables
pais <- "Colombia"
nombre_volcan <- "Nevado del Ruiz"
altura_volcan <- 5321L          # Entero (Integer)
num_sensores <- 12L             # Entero (Integer)
latitud <- 4.8920               # Decimal (Double)
longitud <- -75.3188            # Decimal (Double)

# Interpolación: 2 strings (%s), 2 enteros (%d) y 2 decimales con precisión (%f)
# Usamos %.4f para asegurar 4 decimales en las coordenadas
mensaje <- sprintf("En %s, el %s (Altitud: %d m) tiene %d sensores. Ubicación: %.4f, %.4f",
                   pais,
                   nombre_volcan,
                   altura_volcan,
                   num_sensores,
                   latitud,
                   longitud)
mensaje
```

```
[1] "En Colombia, el Nevado del Ruiz (Altitud: 5321 m) tiene 12 sensores. Ubicación: 4.8920, -
75.3188"
```

5.9.3 Julia

```julia
# #| eval: false
j_eval('
# Inicialización de variables necesarias
num_elementos = 500
# Incrementar el número de elementos
num_elementos += 20
num_elementos

# Convertir latitud a radianes usando la función nativa deg2rad
latitud_ruiz = 4.8920
latitud_radianes = deg2rad(latitud_ruiz)

# Latitud, Longitud del Nevado del Ruiz
coordenadas = [4.8920, -75.3188]

# --- Uso de push!() (Equivalente a append de Python) ---
# Agregar elementos al final del vector original
# Agregar coordenadas de Bogotá al final
push!(coordenadas, 4.6097, -74.0817)

# --- ADVERTENCIA DE TIPO (Type Safety) ---
# A diferencia de R, Julia NO convertirá los números a texto automáticamente.
# Si el vector es de tipo Float64, esta línea lanzará un ERROR:
# insert!(coordenadas, 1, "Colombia")

# Para que funcione como en Python (mezclando tipos), el vector
# debe ser de tipo \'Any\' o estar predefinido para aceptar strings.
coordenadas_mix = Any[4.8920, -75.3188, 4.6097, -74.0817]

# --- Uso de insert!() (Equivalente a insert en Python) ---
```

```
# Insertar "Colombia" al inicio (Índice 1 en Julia)
insert!(coordenadas_mix, 1, "Colombia")

# Ver el resultado
coordenadas_mix

# Acceso e interpolación de texto
nombre_volcan = atributos_elemento["nombre"]
altura_volcan = atributos_elemento["altura_msnm"]
mensaje = "$nombre_volcan tiene una altura de $altura_volcan metros."

# Mostrar el resultado final
mensaje
')
```

Tabla 5.6: Comparación de operaciones comunes con variables

Operación	Python	R	Julia
Incremento	`+=`	`x <- x + n`	`+=`
Grados a Radianes	`math.radians()`	`x * (pi/180)`	`deg2rad()`
Agregar al final	`.append()` (Mutable)	`c(x, n)` (Copia/Coerción)	`push!()` (Mutable)
Insertar en posición	`.insert(índice, v)`	`append(x, v, after=n)`	`insert!(x, índice, v)`
Primer índice	`0`	`1`	`1`
Mezclar tipos (Heterogéneo)	Nativo (Listas)	No (Coerción a texto)	Requiere `Any[]`
Interpolación de texto	`f"{var}"`	`sprintf()` / `paste()`	`"$var"`
Marcadores (Formato)	Ver Sección 5.9.4	`%s`, `%d`, `%f`	Ver Sección 5.9.4
Precisión (4 dec)	`f"{v:.4f}"`	`sprintf("%.4f", v)`	`@sprintf("%.4f", v)`

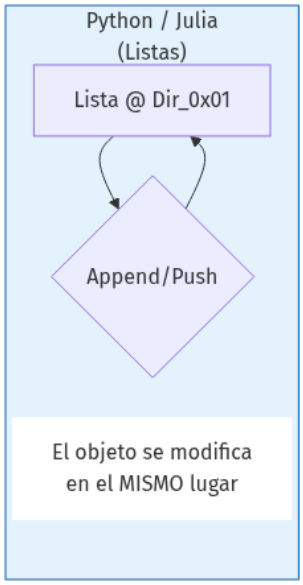
Donde los marcadores representan: * **s**: String (Texto). * **d**: Integer (Entero). * **f**: Float (Decimal). * **.4f**: Precisión a 4 decimales.

Nota técnica: En el desarrollo de algoritmos geográficos, la mutabilidad es un concepto clave. Mientras que Python y Julia permiten modificar una lista de coordenadas directamente (`append` / `push!`), en R los vectores son atómicos e inmutables; cada vez que "agregamos" un dato, R crea una copia nueva del vector en memoria. Esta diferencia es explicada en Figura 5.3a y Figura 5.3b.

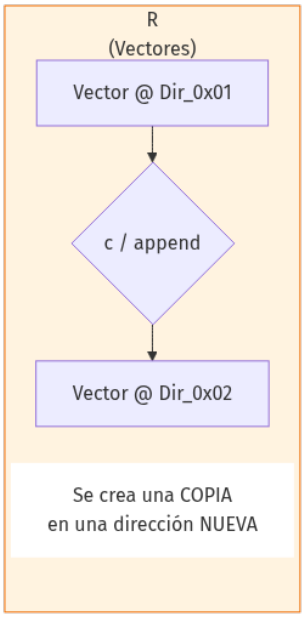
5.9.4 Nota sobre especificadores de formato

Aunque Python y Julia tienen formas nativas más simples (`f""` y `""`), el uso de **s, d, f** es un estándar universal inspirado en el lenguaje C para controlar la apariencia de los datos:

- **En R:** Es el estándar para controlar la salida prolija.
  - *Ejemplo:* `sprintf("%s tiene %d m", "Ruiz", 5321)`
- **En Python:** Estilo "antiguo" con el operador `%`. Útil para entender código heredado.
  - *Ejemplo:* `"%s tiene %d m" % ("Ruiz", 5321)`
- **En Julia:** Requiere la macro `@sprintf` de la librería `Printf`.



(a) Python/Julia: Listas Mutables. El objeto cambia sin cambiar de dirección de memoria.



(b) R: Vectores Inmutables. Al modificar, R crea una copia nueva silenciosamente.

Figura 5.3: Comparativa de manejo de memoria y mutabilidad.

  - *Ejemplo:* `using Printf; @sprintf("%s tiene %d m", "Ruiz", 5321)`

Ejemplos controlando los decimales:

- **En R:** Se usa el símbolo `%` como prefijo obligatorio.
  - *Ejemplo:* `sprintf("%.4f", 4.892)`
- **En Python:** Dentro de las f-strings, se usa el prefijo `:` para indicar el formato.
  - *Ejemplo:* `f"{4.892:.4f}"`
- **En Julia:** Se usa la macro `@sprintf` (librería `Printf`) con el prefijo `%`.
  - *Ejemplo:* `using Printf; @sprintf("%.4f", 4.892)`

En todos los lenguajes, la instrucción `.4f` le indica al intérprete que debe mostrar el número decimal (float) con exactamente **4 decimales**, rellenando con ceros o redondeando si es necesario.

5.10 Operaciones básicas con texto

5.10.1 Python

```
# #| eval: false

# Inicialización de la variable de texto
nombre_ciudad = "cartagena de indias"

# Convertir a minúsculas
ciudad_minusculas = nombre_ciudad.lower()
print("Minúsculas:", ciudad_minusculas)
```

```
Minúsculas: cartagena de indias
```

```python
# Convertir a mayúsculas (útil para estandarizar tablas de atributos)
ciudad_mayusculas = nombre_ciudad.upper()
print("Mayúsculas:", ciudad_mayusculas)
```

```
Mayúsculas: CARTAGENA DE INDIAS
```

```python
# Convertir a formato título (primera letra de cada palabra en mayúscula)
ciudad_titulo = nombre_ciudad.title()
print("Formato título:", ciudad_titulo)
```

```
Formato título: Cartagena De Indias
```

```python
# Reemplazar texto específico dentro de una cadena
ciudad_original = "San Andrés"
nueva_ciudad = ciudad_original.replace("San", "Isla de San")
print("Original:", ciudad_original)
```

```
Original: San Andrés
```

```python
print("Modificada:", nueva_ciudad)
```

```
Modificada: Isla de San Andrés
```

```python
# Limpiar espacios en blanco al inicio y al final (Trim)
datos_ubicacion = "   Bogotá D.C.   "
ubicacion_limpia = datos_ubicacion.strip()
print("Texto limpio:", f"'{ubicacion_limpia}'")
```

```
Texto limpio: 'Bogotá D.C.'
```

5.10.2 R

```r
# #| eval: false

# Cargar paquete nativo recomendado para formato título
library(tools)

# Inicialización de la variable de texto
nombre_ciudad <- "cartagena de indias"

# Convertir a minúsculas
ciudad_minusculas <- tolower(nombre_ciudad)
cat("Minúsculas:", ciudad_minusculas, "\n")
```

```
Minúsculas: cartagena de indias
```

```r
# Convertir a mayúsculas (útil para estandarizar tablas de atributos)
ciudad_mayusculas <- toupper(nombre_ciudad)
cat("Mayúsculas:", ciudad_mayusculas, "\n")
```

```
Mayúsculas: CARTAGENA DE INDIAS
```

```r
# Convertir a formato título
ciudad_titulo <- toTitleCase(nombre_ciudad)
cat("Formato título:", ciudad_titulo, "\n")
```

```
Formato título: Cartagena De Indias
```

```r
# Reemplazar texto específico dentro de una cadena
ciudad_original <- "San Andrés"
nueva_ciudad <- gsub("San", "Isla de San", ciudad_original)
cat("Original:", ciudad_original, "\n")
```

```
Original: San Andrés
```

```r
cat("Modificada:", nueva_ciudad, "\n")
```

```
Modificada: Isla de San Andrés
```

```r
# Limpiar espacios en blanco al inicio y al final (Trim)
datos_ubicacion <- "   Bogotá D.C.   "
ubicacion_limpia <- trimws(datos_ubicacion)
cat("Texto limpio: '", ubicacion_limpia, "'\n", sep="")
```

```
Texto limpio: 'Bogotá D.C.'
```

5.10.3 Julia

```julia
# #| eval: false
j_eval('
# Inicialización de la variable de texto
nombre_ciudad = "cartagena de indias"

# Convertir a minúsculas
ciudad_minusculas = lowercase(nombre_ciudad)
println("Minúsculas: ", ciudad_minusculas)

# Convertir a mayúsculas (útil para estandarizar tablas de atributos)
ciudad_mayusculas = uppercase(nombre_ciudad)
println("Mayúsculas: ", ciudad_mayusculas)

# Convertir a formato título (primera letra de cada palabra en mayúscula)
ciudad_titulo = titlecase(nombre_ciudad)
println("Formato título: ", ciudad_titulo)

# Reemplazar texto específico dentro de una cadena (usa el operador Pair =>)
ciudad_original = "San Andrés"
nueva_ciudad = replace(ciudad_original, "San" => "Isla de San")
println("Original: ", ciudad_original)
println("Modificada: ", nueva_ciudad)

# Limpiar espacios en blanco al inicio y al final (Trim)
datos_ubicacion = "   Bogotá D.C.   "
ubicacion_limpia = strip(datos_ubicacion)
println("Texto limpio: \'", ubicacion_limpia, "\'")
')
```

Tabla 5.7: Equivalencias de funciones para manipulación de cadenas de texto

Operación	Python	R	Julia
Minúsculas	`.lower()`	`tolower()`	`lowercase()`
Mayúsculas	`.upper()`	`toupper()`	`uppercase()`
Formato Título	`.title()`	`tools::toTitleCase()`	`titlecase()`
Reemplazar	`.replace("viejo", "nuevo")`	`gsub("viejo", "nuevo", var)`	`replace(var, "viejo" => "nuevo")`
Limpiar Espacios	`.strip()`	`trimws()`	`strip()`

Nota técnica: En el geoprocesamiento, la limpieza de cadenas de texto (*string parsing*) es una fase crítica antes de realizar cruces espaciales o uniones de tablas (joins). Diferencias imperceptibles como espacios en blanco accidentales (`"Bogotá "` vs `"Bogotá"`) o discrepancias de mayúsculas y minúsculas provocarán que el software GIS falle al emparejar los registros. Las funciones de `strip()` y `upper()` son la primera línea de defensa para garantizar la integridad topológica y tabular de los datos.

5.11 Resumen de aprendizajes (cheat sheet)

En esta sección hemos abordado los cimientos de la programación aplicados al análisis de datos. A continuación, se presenta un **Cheat Sheet (Hoja de Referencia)** consolidado con las equivalencias sintácticas y funcionales entre Python, R y Julia.

5.11.1 1. Asignación y Operaciones Básicas

Tabla 5.8: Resumen de sintaxis y manejo de variables

Concepto / Operación	Python	R	Julia
Asignación Básica	`=`	`<-` (Estándar)	`=`
Imprimir en consola	`print(var)`	`print(var)` o `cat(var)`	`println(var)`
Incrementar valor	`x += 1`	`x <- x + 1`	`x += 1`
Índice Inicial (Arrays)	`[0]` (Base 0)	`[1]` (Base 1)	`[1]` (Base 1)
Último elemento	`[-1]`	`[length(x)]`	`[end]`

5.11.2 2. Estructuras, Tipos y Gestión de Memoria

Tabla 5.9: Resumen de estructuras de datos y su comportamiento en memoria (Mutabilidad)

Concepto	Python	R	Julia
Número Entero	`int` (ej. `500`)	`integer` (ej. `500L`)	`Int64` (ej. `500`)
Número Decimal	`float`	`numeric` (Double)	`Float64`

Concepto	Python	R	Julia
Booleano	`bool` (`True`/`False`)	`logical` (`TRUE`/`FALSE`)	`Bool` (`true`/`false`)
Lista/Vector	`list` `[...]`	`vector` `c(...)`	`Array` `[...]`
Agregar al Final	`lista.append(x)`	`v <- c(v, x)`	`push!(lista, x)`
Insertar en Posición	`lista.insert(i, x)`	`append(v, x, after=i)`	`insert!(lista, i, x)`
Comportamiento Memoria	**Mutable** (In-place). Modifica el objeto original.	**Inmutable** (Copy-on-modify). Crea una copia nueva.	**Mutable** (In-place). Modifica el objeto original.
Diccionario	`dict` `{k: v}`	`list` `list(k = v)`	`Dict` `Dict(k => v)`

5.11.3 3. Interpolación y Formato de Salida (Formatters)

Esta tabla resume cómo insertar variables dentro de texto y controlar su precisión (ej. coordenadas).

Tabla 5.10: Resumen de técnicas de interpolación y formateo numérico. Nota: En Julia `@sprintf` requiere `using Printf`.

Operación	Python	R	Julia
Interpolación Básica	`f"Texto {var}"`	`paste("Texto", var)`	`"Texto $var"`
Interpolación con Formato	`f"Valor: {var:.4f}"`	`sprintf("Valor: %.4f", var)`	`@sprintf "Valor: %.4f" var`
Marcador Texto	`{var}`	`%s`	`%s`
Marcador Entero	`{var}` o `{:d}`	`%d`	`%d`
Marcador Decimal	`{var}` o `{:f}`	`%f`	`%f`
Control Decimales	`:.4f` (4 decimales)	`%.4f` (4 decimales)	`%.4f` (4 decimales)

5.11.4 4. Manipulación de Cadenas de Texto (Limpieza Tabular)

Tabla 5.11: Resumen de funciones para limpieza de texto (String Parsing)

Operación	Python	R	Julia
A minúsculas	`var.lower()`	`tolower(var)`	`lowercase(var)`
A MAYÚSCULAS	`var.upper()`	`toupper(var)`	`uppercase(var)`
Formato Título	`var.title()`	`tools::toTitleCase(var)`	`titlecase(var)`
Reemplazar texto	`var.replace("a", "b")`	`gsub("a", "b", var)`	`replace(var, "a" => "b")`
Quitar espacios vacíos	`var.strip()`	`trimws(var)`	`strip(var)`

5.11.5 5. Documentación y caracteres especiales

Tabla 5.12: Resumen de caracteres de escape y sintaxis de documentación

Carácter	Python	R	Julia
`#`	Comentario en línea.	Comentario en línea.	Comentario en línea.
`\n`	Salto de línea.	Salto de línea.	Salto de línea.
`\t`	Tabulación.	Tabulación.	Tabulación.
`"""`	*Docstrings* (Documentación).	No soportado (Error).	Cadenas/Archivos multilínea.
`#= =#`	No soportado.	No soportado.	Comentario de bloque nativo.

Conclusión del Módulo: La correcta elección de los tipos de datos (Listas vs. Diccionarios), la comprensión del índice de origen (Base 0 vs. Base 1) y el control estricto del formato (`%.4f` vs `.4f`) son pilares fundamentales para evitar errores de redondeo en coordenadas. Asimismo, entender la **mutabilidad** es vital: mientras Python y Julia permiten **insertar** o modificar datos "in-place" (eficiente), R tiende a generar copias en memoria, lo cual debe considerarse al procesar grandes volúmenes de información geoespacial.

5.12 Ejercicios

Para poner en práctica los conceptos aprendidos, deberás resolver los siguientes dos ejercicios. Puedes elegir resolverlos en **Python, R o Julia** (o implementar la solución en varios lenguajes si deseas retarte).

5.12.1 Ejercicio 1: Asignación de variables y operaciones básicas

Contexto: Formas parte del equipo SIG de una corporación autónoma regional y necesitas estructurar los metadatos de una nueva estación de monitoreo ambiental antes de ingresarla a la base de datos espacial.

Instrucciones de código:

1. Define las siguientes variables respetando estrictamente el estándar `snake_case`:
   - Nombre de la estación: "Páramo de Santurbán"
   - Latitud: `7.2514`
   - Longitud: `-72.9069`
   - Elevación en metros: `3350`
   - ¿Está activa?: `Verdadero` (Usa el tipo booleano correcto según tu lenguaje).
2. Crea una estructura de colección (Lista en Python, Vector en R, o Array en Julia) que contenga la Latitud y la Longitud, en ese orden.
3. Crea un **Diccionario** (o Lista con nombres en R) llamado `metadatos_estacion` que agrupe todas las variables anteriores bajo claves descriptivas.
4. Usando la indexación adecuada (Base 0 o Base 1, según el lenguaje que elegiste), extrae específicamente la **Longitud** desde la colección de coordenadas que está dentro del diccionario y guárdala en una nueva variable.
5. Se ha realizado una nueva medición topográfica y la elevación real es 12.5 metros más alta. Actualiza la variable de elevación utilizando el operador de incremento.

6. Imprime en consola un mensaje dinámico utilizando interpolación de texto (ej. `f-strings`, `paste()` o `$`) que diga exactamente: *"La estación [Nombre] se encuentra operativa en la longitud [Longitud] a una altura actualizada de [Elevación] msnm."*

5.12.2 Ejercicio 2: Trabajando con strings

Contexto: Has recibido una tabla de Excel con la toponimia de áreas protegidas de Colombia capturada manualmente por diferentes operarios. Los textos están sucios y, si intentas hacer un cruce espacial (*Spatial Join*) con la capa oficial del IGAC, el software arrojará errores de topología tabular.

Instrucciones de código:

1. Inicializa una variable con el siguiente texto crudo y problemático: `registro_crudo = "   sAnTuaRio de fAuna Y flora iGuaQue   "`
2. **Utiliza el método o función nativa de tu lenguaje** (vista en la sección anterior) para eliminar los espacios en blanco accidentales al inicio y al final del texto.
3. **Usa el comando correspondiente para convertir** el texto limpio a formato de Título (*Title Case*), estandarizando así las mayúsculas y minúsculas.
4. **Emplea la herramienta de reemplazo** de texto para cambiar la letra " Y " por " y " (en minúscula), de modo que el conector gramatical sea correcto.
5. Utiliza caracteres de escape (`\n` y `\t`) para imprimir un pequeño reporte estructurado que muestre el antes y el después, similar a esto:

```
Reporte de Limpieza Toponímica:
    Registro Original: "   sAnTuaRio de fAuna Y flora iGuaQue   "
    Registro Limpio: "Santuario De Fauna y Flora Iguaque"
Estado: Listo para cruce espacial.
```

5.12.3 Entregables y Criterios de Evaluación

El objetivo de esta evaluación no es solo que el código funcione, sino que seas capaz de documentar y explicar tus decisiones técnicas.

1. Archivos de Código: Debes desarrollar los algoritmos en al menos uno de los siguientes formatos de archivo:

- Script tradicional (`.py`, `.R`, `.jl`)
- Notebook interactivo (`.ipynb`)
- Documento computacional (`.qmd` con *chunks* de código)

2. Documento Analítico (Quarto): Independientemente del formato de tu código fuente, **debes redactar un documento en Quarto (`.qmd`) y renderizarlo tanto en HTML como en PDF**. En este documento debes incluir tus bloques de código y responder argumentativamente a las siguientes preguntas:

- **Sobre el Ejercicio 1:** ¿Qué índice numérico utilizaste para extraer la longitud de la colección y por qué ese número específicamente? ¿Qué error arrojaría el lenguaje si usas el índice del lenguaje opuesto (ej. usar 0 en R o 1 en Python buscando el primer elemento)?
- **Sobre el Ejercicio 2:** En términos de geoprocesamiento de bases de datos relacionales, ¿por qué es un paso crítico y obligatorio aplicar métodos de limpieza como `.strip()` o `trimws()` antes de realizar

uniones de tablas (*Joins*) basadas en nombres de municipios o regiones?

- **Pregunta General:** Explica brevemente la diferencia fundamental entre el uso de triples comillas `"""` en Python versus su uso en Julia.

3. Repositorio en GitHub: Sube tu carpeta del proyecto (que debe contener tus scripts, el archivo `.qmd` y los renders finales en HTML y PDF) a un repositorio público en tu cuenta personal de **GitHub**.

- **Entrega:** Deberás enviar únicamente el enlace (URL) a tu repositorio de GitHub para la calificación.

Capítulo 6

Estructuras de datos

6.1 Funciones j_eval y j_plot en R

```r
# #| include: false
source("./docs/j_eval_j_plot.r")
```

6.2 Introducción

6.3 Objetivos de aprendizaje

6.4 Tuplas

Las **Tuplas** son estructuras de datos diseñadas para almacenar una colección de elementos. Su característica más importante es la **inmutabilidad**: una vez creada una tupla, sus elementos no pueden ser alterados, añadidos ni eliminados. Esto las hace perfectas para almacenar pares de coordenadas estáticas `(Latitud, Longitud)` que no deben modificarse accidentalmente durante la ejecución de un algoritmo espacial.

6.4.1 Python

```python
# #| eval: false

# Creación de una tupla estática (Latitud, Longitud) de Bogotá
punto_bogota = (4.6097, -74.0817)
print(f"Coordenadas de Bogotá: {punto_bogota}")
```

```
Coordenadas de Bogotá: (4.6097, -74.0817)
```

```python
# Acceso a los elementos de la tupla (Recordar: Índice Base 0)
latitud = punto_bogota[0]
longitud = punto_bogota[1]
print(f"Latitud: {latitud}")
```

```
Latitud: 4.6097
```

```
print(f"Longitud: {longitud}")
```

```
Longitud: -74.0817
```

```
# Desempaquetado de tuplas (Tuple Unpacking) en variables separadas
lat, lon = punto_bogota
print(f"Bogotá está ubicada a {lat}°N, {lon}°W")
```

```
Bogotá está ubicada a 4.6097°N, -74.0817°W
```

```
# Almacenamiento de múltiples puntos geográficos como tuplas
medellin = (6.2442, -75.5812)
cali = (3.4516, -76.5320)
cartagena = (10.3910, -75.4794)

print(f"Medellín: {medellin}")
```

```
Medellín: (6.2442, -75.5812)
```

```
print(f"Cali: {cali}")
```

```
Cali: (3.4516, -76.532)
```

```
print(f"Cartagena: {cartagena}")
```

```
Cartagena: (10.391, -75.4794)
```

6.4.2 R

```
# #| eval: false

# R no tiene "tuplas" nativas; usamos vectores 'c()' para coordenadas
punto_bogota <- c(4.6097, -74.0817)
cat("Coordenadas de Bogotá:", punto_bogota, "\n")
```

```
Coordenadas de Bogotá: 4.6097 -74.0817
```

```
# Acceso a los elementos (Recordar: Índice Base 1)
latitud <- punto_bogota[1]
longitud <- punto_bogota[2]
cat("Latitud:", latitud, "\n")
```

```
Latitud: 4.6097
```

```
cat("Longitud:", longitud, "\n")
```

```
Longitud: -74.0817
```

```r
# Asignación de variables (R nativo no tiene desempaquetado automático)
lat <- punto_bogota[1]
lon <- punto_bogota[2]
cat(sprintf("Bogotá está ubicada a %s°N, %s°W\n", lat, lon))
```

```
Bogotá está ubicada a 4.6097°N, -74.0817°W
```

```r
# Almacenamiento de múltiples puntos geográficos
medellin <- c(6.2442, -75.5812)
cali <- c(3.4516, -76.5320)
cartagena <- c(10.3910, -75.4794)

cat("Medellín:", medellin, "\n")
```

```
Medellín: 6.2442 -75.5812
```

```r
cat("Cali:", cali, "\n")
```

```
Cali: 3.4516 -76.532
```

```r
cat("Cartagena:", cartagena, "\n")
```

```
Cartagena: 10.391 -75.4794
```

6.4.3 Julia

```julia
# #| eval: false
j_eval('
# Creación de una tupla estática (Latitud, Longitud) de Bogotá
punto_bogota = (4.6097, -74.0817)
println("Coordenadas de Bogotá: ", punto_bogota)

# Acceso a los elementos de la tupla (Recordar: Índice Base 1)
latitud = punto_bogota[1]
longitud = punto_bogota[2]
println("Latitud: ", latitud)
println("Longitud: ", longitud)

# Desempaquetado de tuplas (Tuple Unpacking)
lat, lon = punto_bogota
println("Bogotá está ubicada a $lat°N, $lon°W")

# Almacenamiento de múltiples puntos geográficos como tuplas
medellin = (6.2442, -75.5812)
cali = (3.4516, -76.5320)
cartagena = (10.3910, -75.4794)

println("Medellín: ", medellin)
println("Cali: ", cali)
println("Cartagena: ", cartagena)
')
```

```
Starting Julia ...
```

Tabla 6.1: Comparación técnica de tuplas y equivalentes

Característica	Python	R	Julia
Sintaxis de Creación	`t = (x, y)`	`v <- c(x, y)` (Vector)	`t = (x, y)`
Inmutabilidad estricta	Sí (genera error si se intenta alterar).	No aplica. Los vectores hacen una copia en memoria si se alteran.	Sí (genera error si se intenta alterar).
Desempaquetado (*Unpacking*)	`x, y = t`	Requiere extraer por índice manual `x <- v[1]; y <- v[2]`.	`x, y = t`
Uso Principal en SIG	Proteger pares de coordenadas estáticas de alteraciones en iteraciones.	Creación rápida de puntos geométricos.	Tipado estricto y seguro de coordenadas para máximo rendimiento.

Nota técnica: Las tuplas no existen nativamente en R con el comportamiento de "desempaquetado" e "inmutabilidad estricta" que vemos en Python y Julia. En R, utilizamos **vectores** (`c()`) o **listas** (`list()`) para agrupar coordenadas. Es crucial entender que en Python y Julia, si intentas ejecutar `punto_bogota[0] = 5.0`, el programa colapsará inmediatamente para proteger los datos (porque una tupla es inmutable). En R, `punto_bogota[1] <- 5.0` sí funcionará, reescribiendo la coordenada silenciosamente.

6.5 Listas (y Vectores)

A diferencia de las tuplas, las **Listas** (en Python y Julia) o los **Vectores** (en R) son secuencias **mutables** y dinámicas. Son la estructura de datos más utilizada en geomática para almacenar series temporales, perfiles de elevación, atributos de una tabla o una secuencia ordenada de vértices que conforman una ruta, ya que permiten agregar, eliminar o modificar datos sobre la marcha.

6.5.1 Python

```
# #| eval: false

# 1. CREACIÓN DE LISTAS
# Una lista de tuplas de coordenadas representando una ruta de viaje
ruta = [
    (4.6097, -74.0817),  # Bogotá
    (6.2442, -75.5812),  # Medellín
    (3.4516, -76.5320),  # Cali
]
print("Ruta inicial:", ruta)
```

```
Ruta inicial: [(4.6097, -74.0817), (6.2442, -75.5812), (3.4516, -76.532)]
```

```
# Una lista de mediciones de elevación (en metros)
elevaciones = [2600.5, 2650.2, 2710.8, 2800.3, 2950.7, 3100.1]
print("Perfil de elevación:", elevaciones)
```

```
Perfil de elevación: [2600.5, 2650.2, 2710.8, 2800.3, 2950.7, 3100.1]
```

```
# Una lista de cadenas de texto (nombres de ciudades)
ciudades = ["Bogotá", "Medellín", "Cali"]
print("Ciudades a visitar:", ciudades)
```

```
Ciudades a visitar: ['Bogotá', 'Medellín', 'Cali']
```

```
# 2. AGREGAR ELEMENTOS (Mutabilidad)
# Agregamos Cartagena a nuestra ruta y su elevación al perfil
ruta.append((10.3910, -75.4794))
elevaciones.append(2.0)
print("Ruta actualizada:", ruta)
```

```
Ruta actualizada: [(4.6097, -74.0817), (6.2442, -75.5812), (3.4516, -76.532), (10.391, -
75.4794)]
```

```
# 3. ACCEDER A ELEMENTOS (Índices)
# Acceder a la primera ciudad de la ruta (Índice 0)
primera_parada = ruta[0]
print(f"Primera parada: {primera_parada}")
```

```
Primera parada: (4.6097, -74.0817)
```

```
# Acceder a la última ciudad usando indexación negativa
ultima_parada = ruta[-1]
print(f"Última parada: {ultima_parada}")
```

```
Última parada: (10.391, -75.4794)
```

```
# 4. SLICING (Extraer porciones de la lista)
# Obtener las dos primeras paradas (índices 0 y 1, el 2 es exclusivo)
primeras_dos = ruta[:2]
print("Primeras dos paradas:", primeras_dos)
```

```
Primeras dos paradas: [(4.6097, -74.0817), (6.2442, -75.5812)]
```

```
# Obtener una porción central de las elevaciones (índices 2, 3 y 4)
elevaciones_medias = elevaciones[2:5]
print("Elevaciones intermedias:", elevaciones_medias)
```

```
Elevaciones intermedias: [2710.8, 2800.3, 2950.7]
```

```
# 5. OPERACIONES ÚTILES
# Contar el número de vértices (puntos de ruta)
num_vertices = len(ruta)
print(f"Número de vértices: {num_vertices}")
```

```
Número de vértices: 4
```

```python
# Encontrar la elevación máxima
elev_maxima = max(elevaciones)
print(f"Elevación máxima: {elev_maxima} metros")
```

```
Elevación máxima: 3100.1 metros
```

```python
# Calcular la elevación promedio
elev_promedio = sum(elevaciones) / len(elevaciones)
print(f"Elevación promedio: {elev_promedio:.1f} metros")
```

```
Elevación promedio: 2402.1 metros
```

6.5.2 R

```r
# #| eval: false

# 1. CREACIÓN DE LISTAS Y VECTORES
# En R, usamos 'list' para agrupar objetos complejos como vectores de coordenadas
ruta <- list(
    c(4.6097, -74.0817),  # Bogotá
    c(6.2442, -75.5812),  # Medellín
    c(3.4516, -76.5320)   # Cali
)
cat("Ruta inicial:\n"); print(ruta)
```

```
Ruta inicial:

[[1]]
[1]   4.6097 -74.0817

[[2]]
[1]   6.2442 -75.5812

[[3]]
[1]   3.4516 -76.5320
```

```r
# Los datos homogéneos (números o textos solos) van en vectores atómicos 'c()'
elevaciones <- c(2600.5, 2650.2, 2710.8, 2800.3, 2950.7, 3100.1)
cat("Perfil de elevación:", elevaciones, "\n")
```

```
Perfil de elevación: 2600.5 2650.2 2710.8 2800.3 2950.7 3100.1
```

```r
ciudades <- c("Bogotá", "Medellín", "Cali")
cat("Ciudades a visitar:", ciudades, "\n")
```

```
Ciudades a visitar: Bogotá Medellín Cali
```

```r
# 2. AGREGAR ELEMENTOS
# Agregamos Cartagena a la ruta (usando append para listas)
ruta <- append(ruta, list(c(10.3910, -75.4794)))
```

```r
# Agregamos la elevación al vector (creando un nuevo vector combinado)
elevaciones <- c(elevaciones, 2.0)

# 3. ACCEDER A ELEMENTOS (Índice Base 1)
# En listas de R, se usa doble corchete [[ ]] para extraer el elemento real
primera_parada <- ruta[[1]]
cat("Primera parada:", primera_parada, "\n")
```

```
Primera parada: 4.6097 -74.0817
```

```r
# Acceder a la última ciudad usando la longitud total
ultima_parada <- ruta[[length(ruta)]]
cat("Última parada:", ultima_parada, "\n")
```

```
Última parada: 10.391 -75.4794
```

```r
# 4. SLICING (Extraer porciones)
# En R, el índice final SÍ se incluye en el rango
primeras_dos <- ruta[1:2] # Usa corchete simple para rebanar listas

# Para extraer las elevaciones equivalentes (3er, 4to y 5to elemento)
elevaciones_medias <- elevaciones[3:5]
cat("Elevaciones intermedias:", elevaciones_medias, "\n")
```

```
Elevaciones intermedias: 2710.8 2800.3 2950.7
```

```r
# 5. OPERACIONES ÚTILES
num_vertices <- length(ruta)
cat("Número de vértices:", num_vertices, "\n")
```

```
Número de vértices: 4
```

```r
elev_maxima <- max(elevaciones)
cat("Elevación máxima:", elev_maxima, "metros\n")
```

```
Elevación máxima: 3100.1 metros
```

```r
# R tiene una función nativa para el promedio
elev_promedio <- mean(elevaciones)
cat(sprintf("Elevación promedio: %.1f metros\n", elev_promedio))
```

```
Elevación promedio: 2402.1 metros
```

6.5.3 Julia

```julia
# #| eval: false
j_eval('
# 1. CREACIÓN DE ARREGLOS (Listas)
# Un Array de tuplas de coordenadas
ruta = [
    (4.6097, -74.0817),  # Bogotá
    (6.2442, -75.5812),  # Medellín
    (3.4516, -76.5320),  # Cali
]
println("Ruta inicial: ", ruta)

# Array de valores numéricos de coma flotante
```

```
elevaciones = [2600.5, 2650.2, 2710.8, 2800.3, 2950.7, 3100.1]
println("Perfil de elevación: ", elevaciones)

# Array de cadenas de texto
ciudades = ["Bogotá", "Medellín", "Cali"]
println("Ciudades a visitar: ", ciudades)

# 2. AGREGAR ELEMENTOS (Mutabilidad directa con push!)
push!(ruta, (10.3910, -75.4794))
push!(elevaciones, 2.0)
println("Ruta actualizada: ", ruta)

# 3. ACCEDER A ELEMENTOS (Índice Base 1)
# Acceder al primer vértice
primera_parada = ruta[1]
println("Primera parada: ", primera_parada)

# Julia tiene la palabra clave "end" para acceder al último elemento
ultima_parada = ruta[end]
println("Última parada: ", ultima_parada)

# 4. SLICING (Extraer porciones)
# El rango [1:2] incluye ambos límites
primeras_dos = ruta[1:2]
println("Primeras dos paradas: ", primeras_dos)

# Las posiciones equivalentes (3er, 4to y 5to elemento)
elevaciones_medias = elevaciones[3:5]
println("Elevaciones intermedias: ", elevaciones_medias)

# 5. OPERACIONES ÚTILES
num_vertices = length(ruta)
println("Número de vértices: ", num_vertices)

# En Julia se prefiere "maximum" para colecciones completas
elev_maxima = maximum(elevaciones)
println("Elevación máxima: ", elev_maxima, " metros")

# Promedio (sumatoria sobre longitud)
elev_promedio = sum(elevaciones) / length(elevaciones)
println("Elevación promedio: ", round(elev_promedio, digits=1), " metros")
')
```

6.5.4 Resumen de Operaciones en Listas/Vectores

Tabla 6.2: Equivalencias para el manejo de secuencias ordenadas

Operación	Python	R	Julia
Creación	`[x, y, z]`	`c(x, y, z)`	`[x, y, z]`
Agregar	`.append(x)`	`c(vector, x)`	`push!(array, x)`
Último Elemento	`[-1]`	`[length(x)]`	`[end]`
Rango (*Slicing*)	`[2:5]` (Excluye el 5)	`[3:5]` (Incluye el 5)	`[3:5]` (Incluye el 5)
Largo total	`len(x)`	`length(x)`	`length(x)`
Máximo	`max(x)`	`max(x)`	`maximum(x)`
Promedio	`sum(x)/len(x)`	`mean(x)`	`sum(x)/length(x)`

Nota técnica sobre Slicing (Rebanado): Extraer porciones de datos (*slicing*) es una de las mayores fuentes de error topológico y estadístico al migrar código. * En **Python**, la sintaxis `[2:5]` extrae los elementos en los índices `2, 3 y 4` (omitiendo el final). Dado que Python inicia en 0, esto corresponde al 3er, 4to y 5to elemento real de los datos. * En **R y Julia**, la sintaxis `[3:5]` extrae los elementos en los índices `3, 4 y 5` (incluyendo ambos extremos). Como inician en 1, esto corresponde directamente al 3er, 4to y 5to dato.

El dilema en R: ¿Vectores `c()` o Listas `list()`?

Existe un choque de terminología fundamental al pasar de lenguajes como Python o Julia hacia R: **lo que Python y Julia llaman "Lista" (`[1, 2, 3]`), R lo llama "Vector Atómico" (`c(1, 2, 3)`). Y lo que R llama "Lista" (`list()`), en Python suele comportarse más como una lista anidada o, si tiene nombres, como un Diccionario.**

Para evitar errores al manipular datos espaciales en R, es vital entender esta diferencia práctica:

1. **Vectores Atómicos `c()`:** Exigen que **todos los elementos sean del mismo tipo** (todos números o todos texto). Son estructuras planas (1D). Se usan para series de datos simples, como un perfil de elevaciones o una columna de nombres de municipios.
   - *Ejemplo:* `c(2600.5, 2650.2, 2710.8)`
2. **Listas `list()`:** Son contenedores flexibles. Permiten mezclar texto con números, e incluso **pueden contener otros vectores u otras listas dentro de sí mismas** (anidación). Se usan para agrupar objetos complejos, como los pares de coordenadas `(x,y)` independientes que componen una ruta.
   - *Ejemplo:* `list( c(4.6, -74.0), c(6.2, -75.5) )`

Regla de oro en R: Usa `c()` para colecciones de datos homogéneos. Usa `list()` cuando necesites agrupar geometrías complejas, atributos mixtos o emular la estructura de un diccionario clave-valor.

6.6 Conjuntos (*Sets*)

Los **Conjuntos** son estructuras de datos que almacenan colecciones de elementos **sin orden y sin duplicados**. Son la traducción computacional de la teoría matemática de conjuntos. En el análisis espacial, son indispensables para obtener listas de valores únicos (por ejemplo, extraer todos los códigos de municipios de una gran base de datos sin repeticiones) y para realizar operaciones espaciales lógicas como intersecciones, uniones o diferencias de atributos entre dos zonas geográficas.

[Image of set theory Venn diagram operations]

6.6.1 Python

```
# #| eval: false

# 1. CREACIÓN Y LIMPIEZA DE DUPLICADOS
# Crear un conjunto con llaves {}
regiones_visitadas = {"Andina", "Caribe", "Pacífica"}
print("Regiones:", regiones_visitadas)
```

```
Regiones: {'Andina', 'Pacífica', 'Caribe'}
```

```
# Crear un conjunto desde una lista (elimina duplicados automáticamente)
codigos_dptos = ["ANT", "BOG", "VAL", "ANT", "BOG"]
codigos_unicos = set(codigos_dptos)
print("Códigos únicos:", codigos_unicos)
```

```
Códigos únicos: {'ANT', 'VAL', 'BOG'}
```

```
# 2. AGREGAR ELEMENTOS
# Agregar una región nueva
regiones_visitadas.add("Orinoquía")
# Intentar agregar un duplicado (Python lo ignorará sin dar error)
regiones_visitadas.add("Caribe")
print("Regiones actualizadas:", regiones_visitadas)
```

```
Regiones actualizadas: {'Orinoquía', 'Andina', 'Pacífica', 'Caribe'}
```

```
# 3. OPERACIONES DE CONJUNTOS (Venn)
# Especies observadas en dos polígonos diferentes
area_a = {"Roble", "Pino", "Frailejón", "Aliso"}
area_b = {"Pino", "Frailejón", "Palma de Cera", "Yagrumo"}

# Intersección: Especies comunes en ambas áreas
comunes = area_a.intersection(area_b)
print("Especies en ambas áreas:", comunes)
```

```
Especies en ambas áreas: {'Pino', 'Frailejón'}
```

```
# Diferencia: Especies exclusivas del Área A
solo_a = area_a - area_b
print("Especies exclusivas de A:", solo_a)
```

```
Especies exclusivas de A: {'Aliso', 'Roble'}
```

```
# Unión: Todas las especies registradas
todas_especies = area_a.union(area_b)
print("Todas las especies:", todas_especies)
```

```
Todas las especies: {'Frailejón', 'Yagrumo', 'Aliso', 'Pino', 'Roble', 'Palma de Cera'}
```

```
# 4. MEMBRESÍA (Comprobar si existe)
if "Pino" in comunes:
    print("El Pino está en la zona de intersección.")
```

```
El Pino está en la zona de intersección.
```

6.6.2 R

```
# #| eval: false

# 1. CREACIÓN Y LIMPIEZA DE DUPLICADOS
# En R, usamos vectores atómicos normales y les aplicamos funciones lógicas
regiones_visitadas <- c("Andina", "Caribe", "Pacífica")

# Eliminar duplicados usando unique()
codigos_dptos <- c("ANT", "BOG", "VAL", "ANT", "BOG")
codigos_unicos <- unique(codigos_dptos)
cat("Códigos únicos:", codigos_unicos, "\n")
```

```
Códigos únicos: ANT BOG VAL
```

```
# 2. AGREGAR ELEMENTOS
# Concatenamos y aplicamos unique() para mantener la lógica de conjuntos
regiones_visitadas <- unique(c(regiones_visitadas, "Orinoquía", "Caribe"))
```

```
cat("Regiones actualizadas:", regiones_visitadas, "\n")
```

```
Regiones actualizadas: Andina Caribe Pacífica Orinoquía
```

```
# 3. OPERACIONES DE CONJUNTOS (Venn)
area_a <- c("Roble", "Pino", "Frailejón", "Aliso")
area_b <- c("Pino", "Frailejón", "Palma de Cera", "Yagrumo")

# Intersección: Especies comunes
comunes <- intersect(area_a, area_b)
cat("Especies en ambas áreas:", comunes, "\n")
```

```
Especies en ambas áreas: Pino Frailejón
```

```
# Diferencia: Especies exclusivas de A
solo_a <- setdiff(area_a, area_b)
cat("Especies exclusivas de A:", solo_a, "\n")
```

```
Especies exclusivas de A: Roble Aliso
```

```
# Unión: Todas las especies (elimina duplicados automáticamente)
todas_especies <- union(area_a, area_b)
cat("Todas las especies:", todas_especies, "\n")
```

```
Todas las especies: Roble Pino Frailejón Aliso Palma de Cera Yagrumo
```

```
# 4. MEMBRESÍA (Comprobar si existe con %in%)
if ("Pino" %in% comunes) {
  cat("El Pino está en la zona de intersección.\n")
}
```

```
El Pino está en la zona de intersección.
```

6.6.3 Julia

```
# #| eval: false
j_eval('
# 1. CREACIÓN Y LIMPIEZA DE DUPLICADOS
# En Julia existe el tipo nativo Set
regiones_visitadas = Set(["Andina", "Caribe", "Pacífica"])

# Crear un Set a partir de un Array elimina duplicados
codigos_dptos = ["ANT", "BOG", "VAL", "ANT", "BOG"]
codigos_unicos = Set(codigos_dptos)
println("Códigos únicos: ", codigos_unicos)

# 2. AGREGAR ELEMENTOS
# Usamos push! para agregar elementos in-place
push!(regiones_visitadas, "Orinoquía")
push!(regiones_visitadas, "Caribe") # Se ignora el duplicado
println("Regiones actualizadas: ", regiones_visitadas)

# 3. OPERACIONES DE CONJUNTOS (Venn)
area_a = Set(["Roble", "Pino", "Frailejón", "Aliso"])
area_b = Set(["Pino", "Frailejón", "Palma de Cera", "Yagrumo"])

# Intersección
comunes = intersect(area_a, area_b)
println("Especies en ambas áreas: ", comunes)

# Diferencia
```

```
solo_a = setdiff(area_a, area_b)
println("Especies exclusivas de A: ", solo_a)

# Unión
todas_especies = union(area_a, area_b)
println("Todas las especies: ", todas_especies)

# 4. MEMBRESÍA (Operador in)
if in("Pino", comunes) # También se puede escribir: "Pino" in comunes
    println("El Pino está en la zona de intersección.")
end
')
```

6.6.4 Resumen de Operaciones con Conjuntos

Tabla 6.3: Equivalencias para operaciones lógicas de conjuntos

Operación	Python	R	Julia
Creación Básica	`{"A", "B"}` o `set()`	`c("A", "B")` (Vectores)	`Set(["A", "B"])`
Eliminar Duplicados	`set(lista)`	`unique(vector)`	`Set(array)` o `unique(array)`
Agregar Elemento	`set.add(x)`	`unique(c(v, x))`	`push!(set, x)`
Intersección (Común)	`a.intersection(b)` o `a & b`	`intersect(a, b)`	`intersect(a, b)`
Unión (Todos)	`a.union(b)` o `a | b`	`union(a, b)`	`union(a, b)`
Diferencia (Solo en A)	`a - b`	`setdiff(a, b)`	`setdiff(a, b)`
Pertenece a	`x in set`	`x %in% v`	`in(x, set)` o `x in set`

Nota técnicas:

- Tenga cuidado al crear conjuntos vacíos en **Python**. Si escribe `mis_datos = {}`, Python creará un *Diccionario* vacío, no un conjunto. Para crear un conjunto vacío obligatoriamente debe usar `mis_datos = set()`. Por otro lado, si inicializa con datos como `{"Andina", "Caribe"}`, Python sí entenderá automáticamente que es un conjunto.
- A diferencia de Python y Julia, que tienen una estructura de datos `Set` específica (las cuales ignoran intrínsecamente los duplicados), **R no posee un tipo de dato "Conjunto" nativo**. En R, trabajamos con vectores regulares y utilizamos funciones lógicas (`union`, `intersect`, `setdiff`, `unique`) que simulan el comportamiento de los conjuntos matemáticos al evaluarlos.

6.7 Diccionarios

Los **Diccionarios** son estructuras de datos que almacenan información mediante pares de **Clave-Valor** (*Key-Value*). A diferencia de las listas (donde accedes por la posición numérica), en un diccionario accedes a los datos a través de "etiquetas" únicas (las claves).

En el ámbito de los Sistemas de Información Geográfica, los diccionarios son, literalmente, la base técnica de cómo se estructuran los atributos de un vector. De hecho, el popular formato **GeoJSON** no es más que un gran diccionario anidado.

6.7.1 Python

```
# #| eval: false

# 1. CREACIÓN DE DICCIONARIOS
# Diccionario almacenando atributos de una ciudad
info_bogota = {
    "nombre": "Bogotá",
    "poblacion": 7181469,
    "coordenadas": (4.6097, -74.0817),
    "pais": "Colombia",
    "fundacion": 1538,
}
print("Información de Bogotá:", info_bogota)
```

```
Información de Bogotá: {'nombre': 'Bogotá', 'poblacion': 7181469, 'coordenadas': (4.6097, -74.0817), 'pais': 'Colombia', 'fundacion': 1538}
```

```
# 2. ACCESO A VALORES
# Acceder a información específica usando la clave entre corchetes
nombre_ciudad = info_bogota["nombre"]
poblacion_ciudad = info_bogota["poblacion"]
print(f"Ciudad: {nombre_ciudad} | Población: {poblacion_ciudad:,}")
```

```
Ciudad: Bogotá | Población: 7,181,469
```

```
# Acceso seguro con get() para evitar errores si la clave no existe
area = info_bogota.get("area_km2", "No especificada")
print(f"Área: {area}")
```

```
Área: No especificada
```

```
# 3. AGREGAR Y ACTUALIZAR VALORES
# Agregar nueva información al diccionario
info_bogota["area_km2"] = 1775
info_bogota["poblacion"] = 7900000  # Actualizar un valor existente
print("Info actualizada:", info_bogota)
```

```
Info actualizada: {'nombre': 'Bogotá', 'poblacion': 7900000, 'coordenadas': (4.6097, -74.0817), 'pais': 'Colombia', 'fundacion': 1538, 'area_km2': 1775}
```

```
# 4. TRABAJANDO CON COLECCIONES ANIDADAS (Estilo GeoJSON)
capitales_deptos = {
    "Antioquia": {
        "capital": "Medellín",
        "coordenadas": (6.2442, -75.5812),
        "poblacion": 2533424
    },
    "Valle del Cauca": {
        "capital": "Cali",
        "coordenadas": (3.4516, -76.5320),
        "poblacion": 2227642
    }
}

# Acceder a información anidada
```

```
info_valle = capitales_deptos["Valle del Cauca"]
print(f"Capital del Valle: {info_valle['capital']}")
```

```
Capital del Valle: Cali
```

```
# 5. EXPLORACIÓN DE DICCIONARIOS
print("Claves disponibles:", list(info_bogota.keys()))
```

```
Claves disponibles: ['nombre', 'poblacion', 'coordenadas', 'pais', 'fundacion', 'area_km2']
```

```
# Comprobar si una clave existe
if "coordenadas" in info_bogota:
    print("La información de coordenadas está disponible.")
```

```
La información de coordenadas está disponible.
```

```
# 6. EJEMPLO PRÁCTICO: Metadatos de un Waypoint GPS
punto_gps = {
    "id": "PC001",
    "nombre": "Inicio Sendero Cocora",
    "latitud": 4.6395,
    "longitud": -75.4851,
    "elevacion": 2400,
    "tipo_cobertura": "Bosque de Palma",
    "facilidades": ["parqueadero", "guía", "restaurante"]
}

print(f"Punto: {punto_gps['nombre']} a {punto_gps['elevacion']} msnm")
```

```
Punto: Inicio Sendero Cocora a 2400 msnm
```

```
print(f"Servicios: {', '.join(punto_gps['facilidades'])}")
```

```
Servicios: parqueadero, guía, restaurante
```

6.7.2 R

```
# #| eval: false

# 1. CREACIÓN DE DICCIONARIOS
# En R, usamos listas nombradas (named lists) para emular diccionarios
info_bogota <- list(
    nombre = "Bogotá",
    poblacion = 7181469,
    coordenadas = c(4.6097, -74.0817),
    pais = "Colombia",
    fundacion = 1538
)
cat("Información de Bogotá:\n"); print(info_bogota)
```

```
Información de Bogotá:

$nombre
[1] "Bogotá"
```

```
$poblacion
[1] 7181469

$coordenadas
[1]   4.6097 -74.0817

$pais
[1] "Colombia"

$fundacion
[1] 1538
```

```r
# 2. ACCESO A VALORES
# Accedemos mediante el símbolo del dólar '$' o doble corchete '[[]]'
nombre_ciudad <- info_bogota$nombre
poblacion_ciudad <- info_bogota$poblacion
cat(sprintf("Ciudad: %s | Población: %s\n", nombre_ciudad, format(poblacion_ciudad, big.mark=",")))
```

```
Ciudad: Bogotá | Población: 7,181,469
```

```r
# En R, acceder a una clave inexistente devuelve NULL de forma nativa
area <- if(is.null(info_bogota$area_km2)) "No especificada" else info_bogota$area_km2
cat(sprintf("Área: %s\n", area))
```

```
Área: No especificada
```

```r
# 3. AGREGAR Y ACTUALIZAR VALORES
info_bogota$area_km2 <- 1775
info_bogota$poblacion <- 7900000
cat("Info actualizada:\n"); print(info_bogota)
```

```
Info actualizada:

$nombre
[1] "Bogotá"

$poblacion
[1] 7900000

$coordenadas
[1]   4.6097 -74.0817

$pais
[1] "Colombia"

$fundacion
[1] 1538
```

```
$area_km2
[1] 1775
```

```r
# 4. TRABAJANDO CON COLECCIONES ANIDADAS
capitales_deptos <- list(
    Antioquia = list(
        capital = "Medellín",
        coordenadas = c(6.2442, -75.5812),
        poblacion = 2533424
    ),
    Valle_del_Cauca = list(
        capital = "Cali",
        coordenadas = c(3.4516, -76.5320),
        poblacion = 2227642
    )
)

info_valle <- capitales_deptos$Valle_del_Cauca
cat(sprintf("Capital del Valle: %s\n", info_valle$capital))
```

```
Capital del Valle: Cali
```

```r
# 5. EXPLORACIÓN DE DICCIONARIOS
cat("Claves disponibles:", names(info_bogota), "\n")
```

```
Claves disponibles: nombre poblacion coordenadas pais fundacion area_km2
```

```r
if ("coordenadas" %in% names(info_bogota)) {
    cat("La información de coordenadas está disponible.\n")
}
```

```
La información de coordenadas está disponible.
```

```r
# 6. EJEMPLO PRÁCTICO: Metadatos de un Waypoint GPS
punto_gps <- list(
    id = "PC001",
    nombre = "Inicio Sendero Cocora",
    latitud = 4.6395,
    longitud = -75.4851,
    elevacion = 2400,
    tipo_cobertura = "Bosque de Palma",
    facilidades = c("parqueadero", "guía", "restaurante")
)

cat(sprintf("Punto: %s a %s msnm\n", punto_gps$nombre, punto_gps$elevacion))
```

```
Punto: Inicio Sendero Cocora a 2400 msnm
```

```r
cat(sprintf("Servicios: %s\n", paste(punto_gps$facilidades, collapse=", ")))
```

```
Servicios: parqueadero, guía, restaurante
```

6.7.3 Julia

```r
# #| eval: false
j_eval('
# 1. CREACIÓN DE DICCIONARIOS
# Julia usa Dict() con el operador =>
info_bogota = Dict(
```

```
    "nombre" => "Bogotá",
    "poblacion" => 7181469,
    "coordenadas" => (4.6097, -74.0817),
    "pais" => "Colombia",
    "fundacion" => 1538
)
println("Información de Bogotá: ", info_bogota)

# 2. ACCESO A VALORES
nombre_ciudad = info_bogota["nombre"]
poblacion_ciudad = info_bogota["poblacion"]
println("Ciudad: $nombre_ciudad | Población: $poblacion_ciudad")

# Acceso seguro con get()
area = get(info_bogota, "area_km2", "No especificada")
println("Área: $area")

# 3. AGREGAR Y ACTUALIZAR VALORES
info_bogota["area_km2"] = 1775
info_bogota["poblacion"] = 7900000
println("Info actualizada: ", info_bogota)

# 4. TRABAJANDO CON COLECCIONES ANIDADAS
capitales_deptos = Dict(
    "Antioquia" => Dict(
        "capital" => "Medellín",
        "coordenadas" => (6.2442, -75.5812),
        "poblacion" => 2533424
    ),
    "Valle del Cauca" => Dict(
        "capital" => "Cali",
        "coordenadas" => (3.4516, -76.5320),
        "poblacion" => 2227642
    )
)

info_valle = capitales_deptos["Valle del Cauca"]
println("Capital del Valle: ", info_valle["capital"])

# 5. EXPLORACIÓN DE DICCIONARIOS
println("Claves disponibles: ", collect(keys(info_bogota)))

if haskey(info_bogota, "coordenadas")
    println("La información de coordenadas está disponible.")
end

# 6. EJEMPLO PRÁCTICO: Metadatos de un Waypoint GPS
punto_gps = Dict(
    "id" => "PC001",
    "nombre" => "Inicio Sendero Cocora",
    "latitud" => 4.6395,
    "longitud" => -75.4851,
    "elevacion" => 2400,
    "tipo_cobertura" => "Bosque de Palma",
    "facilidades" => ["parqueadero", "guía", "restaurante"]
)

println("Punto: ", punto_gps["nombre"], " a ", punto_gps["elevacion"], " msnm")
println("Servicios: ", join(punto_gps["facilidades"], ", "))
')
```

6.7.4 Resumen de Diccionarios y Listas Nombradas

Tabla 6.4: Manejo de estructuras Clave-Valor

Operación	Python	R	Julia
Creación	`{"k": v}`	`list(k = v)`	`Dict("k" => v)`
Acceso a valor	`dict["k"]`	`lista$k` o `lista[["k"]]`	`dict["k"]`
Acceso seguro (Fallback)	`dict.get("k", "Null")`	`if(is.null(lista$k))`	`get(dict, "k", "Null")`
Actualizar/Añadir	`dict["k"] = nuevo`	`lista$k <- nuevo`	`dict["k"] = nuevo`
Verificar si existe	`"k" in dict`	`"k" %in% names(lista)`	`haskey(dict, "k")`

Operación	Python	R	Julia
Obtener todas las claves	`dict.keys()`	`names(lista)`	`keys(dict)`

Nota técnica importante sobre errores de acceso: Uno de los comportamientos que más confunde a los analistas de datos ocurre cuando intentan acceder a una clave o atributo que **no existe** en la estructura.

- En **Python y Julia**, pedir una clave inexistente (como `info_bogota["clima"]`) hará que el programa **colapse inmediatamente** (`KeyError`). Por eso, es mejor práctica usar la función segura `.get()` cuando no estamos seguros de si el atributo existe.
- En **R**, la aproximación es mucho más laxa. Si pides `info_bogota$clima`, R no arrojará ningún error, simplemente devolverá `NULL` (vacío). Esto puede ser peligroso porque el programa sigue ejecutándose silenciosamente, pero los cálculos posteriores podrían fallar o contaminarse con datos nulos.

6.8 Guía de Selección de Estructuras de Datos

Saber elegir la estructura de datos correcta es la diferencia entre un script que procesa millones de coordenadas en segundos y uno que colapsa tu computadora. A continuación, te presentamos una guía práctica para tomar esta decisión en tus proyectos de análisis espacial.

6.8.1 Árbol de decisión rápido

Hazte las siguientes preguntas sobre los datos que necesitas almacenar:

1. **¿Los datos tienen un orden secuencial que importa?**
   - *Ejemplo:* Los vértices de un polígono o los puntos de una ruta GPS. El orden define la geometría.
   - **SÍ:** Pasa a la pregunta 2.
   - **NO:** Pasa a la pregunta 3.
2. **¿Necesitarás agregar, borrar o modificar datos después de crearlos?**
   - **SÍ (Mutables):** Usa una **Lista** (o Vector `c()` en R). *Ideal para acumular resultados de un cálculo.*
   - **NO (Inmutables):** Usa una **Tupla** (en Python/Julia). *Ideal para fijar pares de coordenadas exactas (Lat, Lon) que nadie deba alterar por error.*
3. **¿Cómo vas a buscar la información?**
   - **Por una etiqueta o nombre (Clave-Valor):** Usa un **Diccionario** (o `list()` nombrada en R). *Ideal para almacenar los atributos de un municipio (Nombre, Área, Población) o crear metadatos tipo GeoJSON.*
   - **Solo me importan los valores únicos y sin repeticiones:** Usa un **Conjunto** (`Set` o funciones lógicas en R). *Ideal para limpiar bases de datos (ej. obtener los códigos únicos de departamentos desde una tabla nacional de 10,000 registros).*

6.8.2 Casos de Uso Comunes en Geomática

Tabla 6.5: Escenarios prácticos para la selección de estructuras de datos

Escenario	Estructura Recomendada	¿Por qué?
Coordenada estática `(X, Y)`	**Tupla**	Protege el dato. Si un algoritmo intenta mover la X sin mover la Y, el programa arroja error y salva la topología.
Perfil de elevación de una ruta	**Lista / Vector**	Mantiene el orden estricto de los datos. Permite usar funciones matemáticas rápidas (`max`, `mean`).
Tabla de atributos de un lote	**Diccionario**	Permite acceder a los datos intuitivamente por nombre (ej. `lote["propietario"]`) en lugar de recordar en qué columna numérica estaban.
Intersectar áreas de influencia	**Conjunto (Set)**	Permite operaciones de Teoría de Conjuntos nativas y ultrarrápidas (`union`, `intersect`) para saber, por ejemplo, qué fincas están en zona de riesgo.

Consejo de Rendimiento

Si estás procesando *Big Data* espacial (ej. nubes de puntos LiDAR con millones de registros), evita usar **Listas** si vas a realizar búsquedas frecuentes ("¿Existe este punto en la lista?"). Buscar en una lista obliga al computador a revisar elemento por elemento. Por el contrario, buscar en un **Conjunto** o en las claves de un **Diccionario** es casi instantáneo debido a cómo se estructuran internamente en la memoria (Tablas Hash).

6.9 Resumen de Aprendizajes (Cheat Sheet)

En este capítulo hemos explorado cómo organizar, almacenar y manipular grupos de datos usando las cuatro estructuras fundamentales de la programación. A continuación, se presenta tu **Hoja de Referencia (Cheat Sheet)** para traducir estos conceptos entre Python, R y Julia.

6.9.1 1. Creación y Propiedades Fundamentales

Tabla 6.6: Definición de estructuras de datos y mutabilidad

Estructura	Python	R	Julia	Mutabilidad
Tupla (Pares fijos)	`t = (1, 2)`	*Usa vectores* `c(1, 2)`	`t = (1, 2)`	**Inmutable** (Py/Jl)
Lista / Vector (Secuencias)	`l = [1, 2, 3]`	`v <- c(1, 2, 3)`	`a = [1, 2, 3]`	**Mutable**
Conjunto (Valores únicos)	`s = {"A", "B"}`	*Usa vectores* `c("A")`	`s = Set(["A", "B"])`	**Mutable**
Diccionario (Clave-Valor)	`d = {"x": 1}`	`d <- list(x = 1)`	`d = Dict("x" => 1)`	**Mutable**

6.9.2 2. Acceso, Índices y Extracción (*Slicing*)

Tabla 6.7: Reglas de indexación y acceso a elementos

Operación	Python (Base 0)	R (Base 1)	Julia (Base 1)
Primer Elemento	`lista[0]`	`vector[1]` o `lista[[1]]`	`array[1]`
Último Elemento	`lista[-1]`	`vector[length(vector)]`	`array[end]`
Rango (*Slicing*)	`lista[2:5]` *(Excluye el 5)*	`vector[3:5]` *(Incluye el 5)*	`array[3:5]` *(Incluye el 5)*
Acceso a Diccionario	`dict["clave"]`	`lista$clave`	`dict["clave"]`
Acceso Seguro (Fallback)	`dict.get("clave", 0)`	`if(is.null(l$c)) 0 else l$c`	`get(dict, "clave", 0)`

6.9.3 3. Operaciones Principales y Modificaciones

Tabla 6.8: Funciones nativas para manipulación de colecciones

Objetivo / Acción	Python	R	Julia
Tamaño / Longitud	`len(datos)`	`length(datos)`	`length(datos)`
Agregar al final	`lista.append(x)`	`v <- c(v, x)`	`push!(array, x)`
Eliminar Duplicados	`set(lista)`	`unique(vector)`	`unique(array)` o `Set(array)`
Valor Máximo	`max(lista)`	`max(vector)`	`maximum(array)`
Intersección (Comunes)	`set1 & set2`	`intersect(v1, v2)`	`intersect(s1, s2)`
Comprobar si existe	`x in lista`	`x %in% vector`	`x in array`

! Conclusiones Críticas del Módulo

1. **La trampa del Índice de Origen:** Nunca olvides que Python empieza a contar desde `0`, mientras que R y Julia empiezan desde `1`. Migrar un código de análisis de series de tiempo de R a Python sin ajustar los índices resultará en datos desplazados o errores de ejecución (*IndexError*).
2. **La ilusión de las Listas en R:** En Python y Julia, los corchetes `[]` crean listas o arreglos dinámicos. En R, una verdadera colección homogénea se crea con la función concatenar `c()`. El comando `list()` en R equivale en comportamiento a un Diccionario o a una estructura anidada.
3. **Tuplas para Topología:** Acostúmbrate a guardar las coordenadas geográficas siempre como Tuplas en Python y Julia. Su inmutabilidad evitará que un bucle mal programado altere silenciosamente la posición espacial de tus datos originales.

6.10 Ejercicios

Para poner en práctica los conceptos aprendidos sobre estructuras de datos, deberás resolver los siguientes dos ejercicios. Puedes elegir resolverlos en **Python, R o Julia** (o implementar la solución en varios lenguajes si deseas retarte).

6.10.1 Ejercicio 1: Rutas de monitoreo (Tuplas y Listas)

Contexto: Trabajas en un proyecto de conservación en la Sierra Nevada de Santa Marta. Los guardabosques realizan recorridos diarios y te envían las coordenadas de los puntos de avistamiento de fauna. Necesitas estructurar esta ruta temporal de forma segura, garantizando que las coordenadas individuales no se alteren, pero permitiendo que la ruta crezca a medida que reportan nuevos puntos.

Instrucciones de código:

1. Define tres variables (`punto_1`, `punto_2`, `punto_3`) que almacenen las coordenadas (Latitud, Longitud) como **Tuplas** estáticas (o su equivalente más seguro en tu lenguaje).
   - Punto 1: `11.1198, -74.0321`
   - Punto 2: `11.1250, -74.0280`
   - Punto 3: `11.1302, -74.0215`
2. Crea una colección secuencial y mutable (**Lista** en Python/Julia o **Vector/Lista** en R) llamada `ruta_avistamiento` que contenga los tres puntos iniciales en orden.
3. Los guardabosques acaban de reportar un cuarto punto (`11.1355, -74.0150`). Utiliza el método nativo de tu lenguaje para **agregar** esta nueva coordenada al final de tu ruta.
4. Utilizando la indexación dinámica (indexación negativa o comandos como `length/end`), extrae el **último punto** de la ruta y guárdalo en una variable llamada `ultimo_reporte`.
5. Crea una nueva lista/vector paralela llamada `elevaciones` que contenga las alturas de los cuatro puntos: `[850.5, 920.0, 1050.2, 1180.8]`.
6. Calcula e imprime la elevación máxima alcanzada en el recorrido y el número total de puntos visitados (vértices), utilizando las funciones nativas (ej. `max()`, `len()`, `length()`).

6.10.2 Ejercicio 2: Metadatos y Ecosistemas (Diccionarios y Conjuntos)

Contexto: Estás diseñando el esquema de base de datos para los Parques Nacionales Naturales (PNN). Necesitas estructurar los atributos de un parque usando un modelo de clave-valor (similar a GeoJSON) y depurar una lista de observaciones de campo que tiene datos duplicados.

Instrucciones de código:

1. Crea un **Diccionario** (o Lista nombrada en R) llamado `pnn_tayrona` que contenga exactamente las siguientes claves y valores:
   - `"nombre"`: "PNN Tayrona"
   - `"area_hectareas"`: 15000
   - `"abierto_turismo"`: Verdadero (tipo Booleano)
   - `"ecosistemas"`: Una lista/vector con los textos `"Manglar"`, `"Bosque Seco"`, y `"Coral"`.
2. Emplea la función o método de acceso seguro (ej. `.get()` en Python o condicionales en R/Julia) para intentar extraer la clave `"fecha_creacion"`. Como no existe, debe devolver el texto `"Dato no disponible"` sin que el programa colapse. Imprime el resultado.
3. Actualiza el diccionario añadiendo una nueva clave llamada `"departamento"` con el valor `"Magdalena"`.
4. Recibes un reporte de campo crudo con los ecosistemas observados hoy, el cual contiene duplicados: `observaciones_crudas = ["Manglar", "Coral", "Bosque Seco", "Manglar", "Coral", "Matorral"]` Convierte esta colección en un **Conjunto** (*Set* o usando `unique()`) para eliminar los duplicados y guárdalo en la variable `observaciones_unicas`.
5. Utiliza una **operación lógica de conjuntos** (intersección) entre la lista oficial de ecosistemas del parque (guardada dentro de tu diccionario en el paso 1) y tus `observaciones_unicas`. Imprime el resultado para saber qué ecosistemas oficiales fueron efectivamente avistados hoy.

6.10.3 Entregables y Criterios de Evaluación

El objetivo de esta evaluación no es solo que el código funcione, sino que seas capaz de documentar y explicar tus decisiones técnicas.

1. Archivos de Código: Debes desarrollar los algoritmos en al menos uno de los siguientes formatos de archivo: * Script tradicional (`.py`, `.R`, `.jl`) * Notebook interactivo (`.ipynb`) * Documento computacional (`.qmd` con *chunks* de código)

2. Documento Analítico (Quarto): Independientemente del formato de tu código fuente, **debes redactar un documento en Quarto (`.qmd`) y renderizarlo tanto en HTML como en PDF**. En este documento debes incluir tus bloques de código y responder argumentativamente a las siguientes preguntas:

- **Sobre el Ejercicio 1:** En términos de topología espacial y seguridad del código, ¿cuál es la ventaja de guardar las coordenadas individuales `(X, Y)` como *Tuplas* inmutables, pero guardar la ruta completa como una *Lista* mutable? ¿Qué pasaría si intentas modificar directamente la latitud del `punto_1` usando `punto_1[0] = 12.0000` en Python o Julia?
- **Sobre el Ejercicio 2:** En R, existe un debate constante entre usar `c()` o `list()`. Si estuvieras programando en R y necesitaras almacenar la lista de `"ecosistemas"` dentro de las propiedades del parque, ¿qué comando usarías para agrupar esos textos específicos y por qué?
- **Pregunta General (Slicing):** Explica brevemente la diferencia crítica en el comportamiento de

extracción de rangos (*slicing*) si ejecutas `ruta[1:3]` en **Python** comparado con ejecutar el mismo código en **R o Julia**. ¿Cuántos elementos te devuelve cada uno y por qué?

3. Repositorio en GitHub: Sube tu carpeta del proyecto (que debe contener tus scripts, el archivo `.qmd` y los renders finales en HTML y PDF) a un repositorio público en tu cuenta personal de **GitHub**. * **Entrega:** Deberás enviar únicamente el enlace (URL) a tu repositorio de GitHub para la calificación.

https://gispro.gishub.org/book/python/data-structures.html#creating-and-using-tuples

Capítulo 7

Operaciones con Cadenas de Texto (Strings)

7.1 Funciones j_eval y j_plot en R

7.2 Introducción

En el análisis de datos espaciales, gran parte del tiempo se invierte en la limpieza y manipulación de texto. Ya sea para estandarizar nombres de municipios colombianos, armar rutas de archivos raster, separar coordenadas o construir consultas SQL espaciales, dominar las cadenas de texto es fundamental.

7.3 Objetivos de aprendizaje

7.4 1. Creación y Propiedades Básicas

Una cadena de texto (`string`) se puede definir con comillas simples o dobles. Para textos largos (como descripciones de metadatos geográficos), usamos comillas triples.

7.4.1 Python

```
lugar = "Pico Cristóbal Colón"
departamento = 'Magdalena'
descripcion = """El Pico Cristóbal Colón es la montaña más alta
de Colombia, ubicada en la Sierra Nevada de Santa Marta."""

print(f"Lugar: {lugar}")
```

```
Lugar: Pico Cristóbal Colón
```

```
print(f"Departamento: {departamento}")
```

```
Departamento: Magdalena
```

```
print(f"Descripción: {descripcion}")
```

```
Descripción: El Pico Cristóbal Colón es la montaña más alta
de Colombia, ubicada en la Sierra Nevada de Santa Marta.
```

```
# Concatenación y Repetición
ubicacion_completa = lugar + ", " + departamento
separador = "=" * 40

print(separador)
```

```
========================================
```

```
print(f"Ubicación Completa: {ubicacion_completa}")
```

```
Ubicación Completa: Pico Cristóbal Colón, Magdalena
```

```
print(f"Longitud del nombre: {len(lugar)} caracteres")
```

```
Longitud del nombre: 20 caracteres
```

```
print(separador)
```

```
========================================
```

7.4.2 R

```
lugar <- "Pico Cristóbal Colón"
departamento <- 'Magdalena'
descripcion <- "El Pico Cristóbal Colón es la montaña más alta\nde Colombia, ubicada en la Sierra Nevada
↪  de Santa Marta."

cat("Lugar:", lugar, "\n")
```

```
Lugar: Pico Cristóbal Colón
```

```
cat("Departamento:", departamento, "\n")
```

```
Departamento: Magdalena
```

```
# Concatenación y Repetición
ubicacion_completa <- paste0(lugar, ", ", departamento)
separador <- strrep("=", 40)

cat(separador, "\n")
```

```
========================================
```

```
cat("Ubicación Completa:", ubicacion_completa, "\n")
```

```
Ubicación Completa: Pico Cristóbal Colón, Magdalena
```

```
cat("Longitud del nombre:", nchar(lugar), "caracteres\n")
```

```
Longitud del nombre: 20 caracteres
```

```
cat(separador, "\n")
```

```
========================================
```

7.4.3 Julia

```
j_eval(r"-(
lugar = "Pico Cristóbal Colón"
departamento = "Magdalena" # En Julia siempre comillas dobles
descripcion = """El Pico Cristóbal Colón es la montaña más alta
de Colombia, ubicada en la Sierra Nevada de Santa Marta."""

println("Lugar: $lugar")
println("Departamento: $departamento")

# Concatenación (*) y Repetición (^)
ubicacion_completa = lugar * ", " * departamento
separador = "=" ^ 40

println(separador)
println("Ubicación Completa: $ubicacion_completa")
println("Longitud del nombre: $(length(lugar)) caracteres")
println(separador)
)-")
```

```
Starting Julia ...
```

7.4.4 1.1 Interpolación de Cadenas

La interpolación nos permite insertar variables directamente dentro de un texto sin tener que estar abriendo y cerrando comillas constantemente.

7.4.5 Python

```
pico = "Pico Cristóbal Colón"
altura = 5700

# F-strings (Interpolación directa)
print(f"El {pico} tiene una altitud de {altura} msnm.")
```

```
El Pico Cristóbal Colón tiene una altitud de 5700 msnm.
```

7.4.6 R

```r
pico <- "Pico Cristóbal Colón"
altura <- 5700

# En R usamos sprintf (%s para texto, %d para enteros)
mensaje <- sprintf("El %s tiene una altitud de %d msnm.", pico, altura)
cat(mensaje)
```

```
El Pico Cristóbal Colón tiene una altitud de 5700 msnm.
```

7.4.7 Julia

```julia
j_eval(r"-(
pico = "Pico Cristóbal Colón"
altura = 5700

# Julia usa el símbolo $ para interpolar variables
println("El $pico tiene una altitud de $altura msnm.")
)-")
```

Resumen de Sintaxis: Creación y Propiedades

Operación	Python	R	Julia
Definir String	`"Col"`, `'Col'`	`"Col"`, `'Col'`	`"Col"` (Solo dobles)
Multilínea	`"""Texto"""`	`"Texto\nTexto"`	`"""Texto"""`
Concatenar	`+`	`paste0(a, b)`	`*`
Repetir	`* n`	`strrep(str, n)`	`^ n`
Longitud	`len(str)`	`nchar(str)`	`length(str)`
Interpolación	`f"Texto {var}"`	`sprintf("Texto %s", var)`	`"Texto $var"`

7.5 2. Métodos de Limpieza y Transformación

Los datos de entidades como el DANE o el IGAC a veces vienen con errores de digitación o espacios accidentales. Veamos cómo limpiar un topónimo colombiano.

7.5.1 Python

```python
toponimo = "   sAnTuaRio de fAuna Y flora iGuaQue   "

# 1. Quitar espacios laterales
texto_limpio = toponimo.strip()

# 2. Formato Título
texto_titulo = texto_limpio.title()

# 3. Reemplazo de caracteres (Corregir la 'y')
registro_final = texto_titulo.replace(" Y ", " y ")
```

```
print(f"Original: '{toponimo}'")
```

```
Original: '   sAnTuaRio de fAuna Y flora iGuaQue   '
```

```
print(f"Mayúsculas: '{registro_final.upper()}'")
```

```
Mayúsculas: 'SANTUARIO DE FAUNA Y FLORA IGUAQUE'
```

```
print(f"Limpio:   '{registro_final}'")
```

```
Limpio:   'Santuario De Fauna y Flora Iguaque'
```

7.5.2 R

```
toponimo <- "   sAnTuaRio de fAuna Y flora iGuaQue   "

# 1. Quitar espacios
texto_limpio <- trimws(toponimo)

# 2. Formato Título (Requiere pasar a minúscula primero)
texto_titulo <- tools::toTitleCase(tolower(texto_limpio))

# 3. Reemplazo de caracteres
registro_final <- gsub(" Y ", " y ", texto_titulo)

cat(sprintf("Original: '%s'\n", toponimo))
```

```
Original: '   sAnTuaRio de fAuna Y flora iGuaQue   '
```

```
cat(sprintf("Mayúsculas: '%s'\n", toupper(registro_final)))
```

```
Mayúsculas: 'SANTUARIO DE FAUNA Y FLORA IGUAQUE'
```

```
cat(sprintf("Limpio:   '%s'\n", registro_final))
```

```
Limpio:   'Santuario De Fauna y Flora Iguaque'
```

7.5.3 Julia

```
j_eval(r"-(
toponimo = "   sAnTuaRio de fAuna Y flora iGuaQue   "

# 1. Quitar espacios
texto_limpio = strip(toponimo)

# 2. Formato Título
texto_titulo = titlecase(texto_limpio)

# 3. Reemplazo de caracteres
registro_final = replace(texto_titulo, " Y " => " y ")

println("Original: '$toponimo'")
println("Mayúsculas: '$(uppercase(registro_final))'")
println("Limpio:   '$registro_final'")
```

```
)-")
```

Resumen de Sintaxis: Transformaciones

Operación	Python	R	Julia
Mayúsculas	`.upper()`	`toupper()`	`uppercase()`
Minúsculas	`.lower()`	`tolower()`	`lowercase()`
Formato Título	`.title()`	`tools::toTitleCase()`	`titlecase()`
Quitar Espacios	`.strip()`	`trimws()`	`strip()`
Reemplazar	`.replace(old, new)`	`gsub(old, new, str)`	`replace(str, old=>new)`

7.6 3. Separación y Unión (Split & Join)

Es muy común recibir coordenadas en formato de texto separado por comas, o necesitar armar rutas de archivos uniendo directorios para exportar nuestros Shapefiles.

7.6.1 Python

```python
# Separar coordenadas (Plaza de Bolívar, Bogotá)
coord_str = "4.5981,-74.0758"
lat_str, lon_str = coord_str.split(",")

print(f"Latitud: {float(lat_str)}, Longitud: {float(lon_str)}")
```

```
Latitud: 4.5981, Longitud: -74.0758
```

```python
# Unir elementos de una lista (Ruta de archivo)
carpetas = ["datos", "colombia", "cundinamarca", "bogota.geojson"]
ruta_completa = "/".join(carpetas)

print(f"Ruta generada: {ruta_completa}")
```

```
Ruta generada: datos/colombia/cundinamarca/bogota.geojson
```

7.6.2 R

```r
# Separar coordenadas
# TABLA COMPARATIVA DE EXTRACCIÓN EN R:
# | Método                        | Resultado | Metáfora                       |
# |-------------------------------|-----------|--------------------------------|
# | strsplit(x, ",")[1]           | Lista     | La caja con el sobre adentro   |
# | strsplit(x, ",")[[1]]         | Vector    | El sobre abierto (Contenido)   |
# | unlist(strsplit(x, ","))      | Vector    | El contenido sin el sobre      |

coord_str <- "4.5981,-74.0758"

# Explicación: strsplit es 'vectorizada' (está lista para procesar miles de textos a la vez).
# Por eso, siempre devuelve una "Gran Caja" (Lista).
```

```r
# Para sacar el "Sobre" con nuestras coordenadas, usamos [[1]].
# Si usaras solo [1], tendrías la caja pero no podrías tocar los números.
partes <- strsplit(coord_str, ",")[[1]]

# Alternativa más legible (El 'machete'): unlist()
# Esta función "desempaqueta" automáticamente la lista y nos entrega el vector puro.
partes_alt <- unlist(strsplit(coord_str, ","))

cat("Resultado con [[1]]:", partes[1], "\n")
```

```
Resultado con [[1]]: 4.5981
```

```r
cat("Resultado con unlist:", partes_alt[1], "\n")
```

```
Resultado con unlist: 4.5981
```

```r
cat(sprintf("Latitud: %f, Longitud: %f\n", as.numeric(partes[1]), as.numeric(partes[2])))
```

```
Latitud: 4.598100, Longitud: -74.075800
```

```r
# Unir elementos de una lista
carpetas <- c("datos", "colombia", "cundinamarca", "bogota.geojson")
ruta_completa <- paste(carpetas, collapse = "/")

cat(paste("Ruta generada:", ruta_completa, "\n"))
```

```
Ruta generada: datos/colombia/cundinamarca/bogota.geojson
```

7.6.3 Julia

```julia
j_eval(r"-(
# Separar coordenadas
coord_str = "4.5981,-74.0758"
lat_str, lon_str = split(coord_str, ",")

println("Latitud: $(parse(Float64, lat_str)), Longitud: $(parse(Float64, lon_str))")

# Unir elementos de una lista
carpetas = ["datos", "colombia", "cundinamarca", "bogota.geojson"]
ruta_completa = join(carpetas, "/")

println("Ruta generada: $ruta_completa")
)-")
```

Resumen de Sintaxis: Split y Join

Operación	Python	R	Julia
Separar (Split)	`str.split(",")`	`strsplit(str, ",")[[1]]` o `unlist(strsplit(str, ","))`	`split(str, ",")`
Unir (Join)	`"/".join(array)`	`paste(arr, collapse="/")`	`join(array, "/")`
Texto a Decimal (Float)	`float(cadena)`	`as.numeric(cadena)`	`parse(Float64, cadena)`

7.7 4. Formateo de Precisión (WKT y SQL)

Construir geometrías en formato de texto (Well-Known Text) o consultas a bases de datos espaciales requiere un control preciso de la interpolación y los decimales.

7.7.1 Python

```python
# 1. Definición de variables (Medellín, Antioquia)
lat, lon = 6.244203, -75.581211
ciudad = "Medellín"
poblacion = 2529000
poblacion_min = 1000000

# 2. WKT con precisión decimal (.4f)
wkt_point = f"POINT({lon:.4f} {lat:.4f})"

# 3. Reporte con separador de miles (:,)
reporte = f"Población de {ciudad}: {poblacion:,} hab."

print(f"Geometría: {wkt_point}")
```

```
Geometría: POINT(-75.5812 6.2442)
```

```python
print(reporte)
```

```
Población de Medellín: 2,529,000 hab.
```

```python
# 4. Consulta SQL Dinámica (Multilínea)
sql_query = f"""
SELECT geom, nombre FROM municipios
WHERE depto = 'Antioquia' AND poblacion > {poblacion_min:,}
"""
print("Consulta SQL:")
```

```
Consulta SQL:
```

```python
print(sql_query)
```

```
SELECT geom, nombre FROM municipios
WHERE depto = 'Antioquia' AND poblacion > 1,000,000
```

7.7.2 R

```r
# 1. Definición de variables (Medellín, Antioquia)
lat <- 6.244203
lon <- -75.581211
ciudad <- "Medellín"
poblacion <- 2529000
poblacion_min <- 1000000

# 2. WKT con precisión decimal (%.4f)
wkt_point <- sprintf("POINT(%.4f %.4f)", lon, lat)

# 3. Reporte con separador de miles (big.mark)
poblacion_fmt <- format(poblacion, big.mark=",")
reporte <- sprintf("Población de %s: %s hab.", ciudad, poblacion_fmt)
```

```
cat("Geometría:", wkt_point, "\n")
```

```
Geometría: POINT(-75.5812 6.2442)
```

```
cat(reporte, "\n")
```

```
Población de Medellín: 2,529,000 hab.
```

```
# 4. Consulta SQL Dinámica (sprintf para multilínea)
sql_query <- sprintf("
SELECT geom, nombre FROM municipios
WHERE depto = 'Antioquia' AND poblacion > %s
", format(poblacion_min, big.mark=",", scientific = FALSE))

cat("Consulta SQL:\n")
```

```
Consulta SQL:
```

```
cat(sql_query)
```

```
SELECT geom, nombre FROM municipios
WHERE depto = 'Antioquia' AND poblacion > 1,000,000
```

7.7.3 Julia

```
j_eval(r"-(
using Printf

# 1. Definición de variables (Medellín, Antioquia)
lat, lon = 6.244203, -75.581211
ciudad = "Medellín"
poblacion = 2529000
poblacion_min = 1000000

# 2. WKT con precisión decimal (@sprintf)
wkt_point = @sprintf("POINT(%.4f %.4f)", lon, lat)

# 3. Reporte con separador de miles (Regex)
poblacion_coma = replace(string(poblacion), r"(?<=\d)(?=(\d{3})+(?!\d))" => ",")
println("Geometría: $wkt_point")
println("Población de $ciudad: $poblacion_coma hab.")

# 4. Consulta SQL Dinámica (Triples comillas)
poblacion_min_coma = replace(string(poblacion_min), r"(?<=\d)(?=(\d{3})+(?!\d))" => ",")
sql_query = """
SELECT geom, nombre FROM municipios
WHERE depto = 'Antioquia' AND poblacion > $poblacion_min_coma
"""
println("Consulta SQL:")
println(sql_query)
)-")
```

7.7.4 Explicación técnica: El “Radar” de Miles en Julia

A diferencia de Python, Julia no tiene un flag nativo (como el `:,`) para formatear miles automáticamente en el macro `@sprintf`. Por ello, utilizamos una **Expresión Regular (Regex)** que actúa como un “radar” de posiciones:

- `(?<=\d)` **(Lookbehind):** Le dice al motor: "asegúrate de que haya un número justo antes de esta posición".
- `(?=(\d{3})+(?!\d))` **(Lookahead):** Le dice: "busca grupos de exactamente tres dígitos hacia adelante que lleguen hasta el final del texto (o antes de un punto decimal)".

Lo interesante es que este Regex tiene un ancho de cero; no captura caracteres, sino que localiza el **espacio vacío** exacto donde debe "inyectarse" el separador.

7.7.5 Julia (Explicación Regex)

```
j_eval(r"-(
# Ejemplo con la población estimada de la sabana de Bogotá
poblacion = 10700000

# Usamos replace con el radar de posiciones (Regex)
# El símbolo => indica el reemplazo en el hueco hallado
poblacion_fmt = replace(string(poblacion), r"(?<=\d)(?=(\d{3})+(?!\d))" => ",")

println("Población formateada: $poblacion_fmt")
)-")
```

Resumen de Sintaxis: Formateo de Precisión y Consultas

Operación	Python	R	Julia
Interpolación Básica	`f"Texto {var}"`	`sprintf("Texto %s", var)`	`"Texto $var"`
Control Decimal (ej. 4)	`f"{lon:.4f}"`	`sprintf("%.4f", lon)`	`@sprintf("%.4f", lon)`
Separador de Miles[1]	`f"{var:,}"`	`format(var, big.mark=",", scientific = FALSE)`	`replace(string(v), r"..." => ",")`
Consulta SQL / Multilínea	`f""" ... {var} ... """`	`sprintf("\n ... %s \n", var)`	`""" ... $var ... """`

7.8 Resumen de Aprendizajes (Cheat Sheet)

En este capítulo hemos explorado cómo limpiar, transformar, dividir y dar formato de precisión a los datos de texto (Strings), habilidades esenciales para procesar topónimos, coordenadas crudas y construir consultas espaciales. A continuación, se presenta tu **Hoja de Referencia (Cheat Sheet)** para traducir estos conceptos entre Python, R y Julia.

7.8.1 1. Creación y Propiedades Fundamentales

[1] En Colombia, el estándar suele preferir el punto (.) como separador de miles. Para aplicarlo, simplemente cambie el carácter de reemplazo en cada función: `f"{var:,}"`, `format(var, big.mark=",", scientific = FALSE)`, `replace(string(v), r"..." => ",")`.

Tabla 7.5: Sintaxis básica para creación y medición de cadenas

Operación	Python	R	Julia
Definir String	`"Col"`, `'Col'`	`"Col"`, `'Col'`	`"Col"` (Solo dobles)
Multilínea	`"""Texto"""`	`"Texto\nTexto"`	`"""Texto"""`
Concatenar	`+`	`paste0(a, b)`	`*`
Repetir	`* n`	`strrep(str, n)`	`^ n`
Longitud	`len(str)`	`nchar(str)`	`length(str)`
Interpolación	`f"Texto {var}"`	`sprintf("Texto %s", var)`	`"Texto $var"`

7.8.2 2. Transformaciones y Limpieza

Tabla 7.6: Métodos nativos para estandarización de texto

Operación	Python	R	Julia
Mayúsculas	`.upper()`	`toupper()`	`uppercase()`
Minúsculas	`.lower()`	`tolower()`	`lowercase()`
Formato Título	`.title()`	`tools::toTitleCase()`	`titlecase()`
Quitar Espacios	`.strip()`	`trimws()`	`strip()`
Reemplazar	`.replace(old, new)`	`gsub(old, new, str)`	`replace(str, old=>new)`

7.8.3 3. Split, Join y Formateo de Precisión

Tabla 7.7: Funciones críticas para parseo de coordenadas y WKT

Operación	Python	R	Julia
Separar (Split)	`str.split(",")`	`unlist(strsplit(str, ","))`	`split(str, ",")`
Unir (Join)	`"/".join(array)`	`paste(arr, collapse="/")`	`join(array, "/")`
Texto a Decimal	`float(cadena)`	`as.numeric(cadena)`	`parse(Float64, cadena)`
Control Decimal	`f"{lon:.4f}"`	`sprintf("%.4f", lon)`	`@sprintf("%.4f", lon)`
Separador Miles	`f"{v:,}"`	`format(v, big.mark=",", scientific=F)`	`replace(string(v), r"..." => ",")`

! Conclusiones Críticas del Módulo

1. **La trampa del Split en R:** A diferencia de Python y Julia (que devuelven arreglos simples), la función `strsplit()` en R siempre devuelve una *Lista*. Si intentas aplicar cálculos matemáticos directamente al resultado de un `strsplit` sin desempaquetarlo antes (usando `[[1]]` o `unlist()`), R arrojará un error.
2. **Cuidado con la Notación Científica:** Al formatear grandes números (como poblaciones o áreas en metros cuadrados), R y Julia pueden saltar a notación científica (ej. `1e+06`). En geomática, esto rompe las consultas SQL y las etiquetas de los mapas. Acostúmbrate a forzar la salida decimal (ej. `scientific = FALSE` en R).
3. **El poder de la Interpolación:** Construir geometrías WKT o sentencias SQL concatenando pedazos de texto con el símbolo `+` o `*` es propenso a errores de espacios y comillas. Usa siempre las herramientas de interpolación literal (`f-strings` en Python, `sprintf` en R, y `$` en Julia) para mantener tu código legible y seguro.

7.9 Ejercicios

Para poner en práctica los conceptos aprendidos sobre manipulación de cadenas de texto, deberás resolver los siguientes dos ejercicios. Puedes elegir resolverlos en **Python, R o Julia** (o implementar la solución en varios lenguajes si deseas retarte).

7.9.1 Ejercicio 1: Limpieza de Topónimos (PNN Macarena)

Contexto: Has recibido una base de datos antigua del IGAC con los nombres de las áreas protegidas de Colombia. Los datos fueron ingresados manualmente a lo largo de los años, por lo que tienen serios problemas de formato (espacios extra, mayúsculas inconsistentes y errores de digitación). Necesitas estandarizar el nombre del Parque Nacional Natural Serranía de la Macarena para cruzarlo con el sistema actual.

Instrucciones de código:

1. Define una variable llamada `registro_crudo` que contenga exactamente este texto: `"  pnn seRRania de la mACaRena  "`
2. Utiliza el método nativo de tu lenguaje para **eliminar los espacios en blanco** al principio y al final del texto. Guarda el resultado en `texto_sin_espacios`.
3. Convierte el texto resultante a **Formato Título** (donde la primera letra de cada palabra es mayúscula) y guárdalo en `texto_titulo`.
4. El sistema oficial requiere que las siglas "PNN" estén en mayúscula sostenida. Utiliza la función de **reemplazo** para cambiar la subcadena "Pnn" por "PNN" dentro de tu `texto_titulo`. Guarda esto como `registro_oficial`.
5. Calcula la **longitud** (cantidad de caracteres) de tu `registro_oficial`.
6. Imprime un reporte final utilizando **interpolación de cadenas** que diga exactamente: `"El área protegida limpia es: [AQUI_TU_VARIABLE] y su nombre tiene [AQUI_LONGITUD] caracteres."`

7.9.2 Ejercicio 2: Parseo de Coordenadas y WKT (Volcán Galeras)

Contexto: Un sensor sísmico ubicado en el Volcán Galeras (Nariño) envía las alertas a través de mensajes de texto planos. Tu objetivo es extraer las coordenadas de ese mensaje de texto, convertirlas a formato numérico, y ensamblar un punto espacial estándar (Well-Known Text) para insertarlo en la base de datos PostGIS del Servicio Geológico.

Instrucciones de código:

1. Define la variable `mensaje_sensor` con el siguiente texto: `"ALERTA_SISMICA:1.221,-77.359:PROFUNDIDAD_5KM"`
2. Utiliza la función de **separación (Split)** usando los dos puntos (`:`) como delimitador. Extrae únicamente el bloque central que contiene las coordenadas y guárdalo en la variable `bloque_coords`.
3. Vuelve a aplicar la función de **separación (Split)**, esta vez sobre `bloque_coords`, usando la coma (`,`) como delimitador para separar la latitud de la longitud.
4. Las coordenadas extraídas siguen siendo texto. Conviértelas a **formato numérico decimal (Float)** y guárdalas en las variables `latitud` y `longitud`.
5. La base de datos requiere una precisión estricta. Utilizando las herramientas de formateo de tu lenguaje, construye una cadena de texto en formato WKT (`POINT(lon lat)`) asegurando que tanto la latitud como la longitud tengan **exactamente 4 decimales**. Guarda esto en `punto_wkt`. *(Nota: Recuerda que WKT usa el orden Longitud Latitud, separados por un espacio, no por coma).*
6. El sensor reportó una anomalía térmica que afectó un área de `1250000` metros cuadrados. Formatea este número agregando un **separador de miles** (coma o punto) para que sea legible en el reporte final.
7. Imprime el WKT generado y el área afectada formateada.

7.9.3 Entregables y Criterios de Evaluación

El objetivo de esta evaluación no es solo que el código funcione, sino que seas capaz de documentar y explicar tus decisiones técnicas.

1. Archivos de Código: Debes desarrollar los algoritmos en al menos uno de los siguientes formatos de archivo:

- Script tradicional (`.py`, `.R`, `.jl`)
- Notebook interactivo (`.ipynb`)
- Documento computacional (`.qmd` con *chunks* de código)

2. Documento Analítico (Quarto): Independientemente del formato de tu código fuente, **debes redactar un documento en Quarto (`.qmd`) y renderizarlo tanto en HTML como en PDF**. En este documento debes incluir tus bloques de código y responder argumentativamente a las siguientes preguntas:

- **Sobre el Ejercicio 1:** ¿Por qué es una mala práctica depender únicamente de la función "Reemplazar" (`.replace()` / `gsub()`) para quitar espacios accidentales al inicio o final de un texto en bases de datos geográficas grandes, en lugar de usar funciones dedicadas como `strip()` o `trimws()`?
- **Sobre el Ejercicio 2:** En la creación de la geometría WKT, si omitieras el paso de convertir el texto extraído a número decimal (Float) y construyeras el WKT directamente concatenando los textos originales, ¿qué impacto tendría esto al intentar realizar un cálculo de distancia espacial dentro de PostGIS o QGIS posteriormente?

- **Pregunta General (Split en R):** Si decidiste resolver el Ejercicio 2 usando **R**, explica por qué fue necesario utilizar `unlist()` o los dobles corchetes `[[1]]` después de ejecutar la función `strsplit`. Si usaste Python o Julia, explica cómo se diferencia la salida de la función `split` en tu lenguaje respecto a R.

3. Repositorio en GitHub: Sube tu carpeta del proyecto (que debe contener tus scripts, el archivo `.qmd` y los renders finales en HTML y PDF) a un repositorio público en tu cuenta personal de **GitHub**.

- **Entrega:** Deberás enviar únicamente el enlace (URL) a tu repositorio de GitHub para la calificación.

Capítulo 8

Introducción a Python, R y Julia

Capítulo 9

Introducción a Python, R y Julia

9.1 Funciones j_eval y j_plot en R

9.2 Introducción

Este capítulo presenta una **introducción práctica y comparativa** a los lenguajes **Python, R y Julia**, enfocada en el uso de **comandos básicos** y en la lectura e interpretación de resultados en consola. El objetivo no es profundizar en programación avanzada, sino ofrecer una **puerta de entrada común** para estudiantes que se inician en el uso de herramientas computacionales para geocomputación.

9.2.1 Ejecución de código en cada lenguaje

- **Python** y **R** se ejecutan de forma nativa mediante *code chunks* estándar.
- **Julia**, en este curso, **no se ejecuta directamente**. Todo el código Julia se evalúa desde R utilizando funciones auxiliares:
  - `j_eval()` para ejecutar instrucciones generales
  - `j_plot()` para generar gráficos

Por esta razón, en el documento encontrarás dos tipos de bloques relacionados con Julia:

- Bloques con **código Julia puro**, que sirven como referencia y pueden copiarse y reutilizarse.
- Bloques en **R** que llaman a `j_eval()` o `j_plot()`, que son los que realmente ejecutan el código.

9.2.2 Sobre los bloques de código

- **Todos los bloques de código tienen un `label`**, lo que permite su correcta identificación y reutilización.
- Algunos bloques están pensados **exclusivamente para HTML** y no se ejecutan. Su función es permitir que el estudiante **copie y pegue el código** directamente en su entorno de trabajo.
- Cuando un bloque no se ejecuta, esto se indica explícitamente en sus opciones.

9.2.3 Alcance del capítulo

En este capítulo aprenderás a:

- Acceder a la ayuda y documentación básica en cada lenguaje
- Ejecutar comandos simples y entender su salida
- Reconocer tipos de objetos y estructuras básicas
- Comparar cómo Python, R y Julia resuelven tareas similares

Este contenido servirá como base para los capítulos posteriores, donde se utilizarán estos lenguajes para la creación de mapas, análisis de datos espaciales y visualización geográfica.

9.3 Ayuda y documentación

Tarea	Descripción	Python	R	Julia
Ver ayuda general	Abre la documentación interactiva	`help()`	`help.start()`	`?` en REPL
Ayuda de función	Documentación de un comando	`help(sum)`	`?mean`	`?sum`
Ver ejemplos	Código de ejemplo de una función	**Docstrings**	`example("lapply")`	**Docstrings**
Vignettes / guías	Tutoriales extendidos	`obj.__doc__`	`browseVignettes()`	`?Modulo`
Demo	Ejecutar ejemplos	`libreria.ejemplos()`*	`demo()`	`include("test")`*

Nota sobre los Docstrings: En Python y Julia, la documentación y los ejemplos de uso residen directamente dentro del código fuente en bloques llamados *Docstrings*. Al ejecutar los comandos de ayuda (`help()` o `@doc`), el intérprete extrae estos comentarios y los muestra en la terminal, permitiendo que el usuario vea ejemplos reales de implementación de inmediato sin necesidad de manuales externos.

Nota sobre los Demos (*): A diferencia de R, Python y Julia no tienen un comando universal `demo()`. El asterisco indica que el acceso a ejemplos depende de la librería. En Python, se suele explorar la propiedad `__doc__` o módulos de `datasets`. En Julia, se acostumbra a inspeccionar la carpeta de instalación mediante `pathof(Modulo)` para encontrar archivos de prueba o ejemplos.

9.3.1 Python

```
# help() busca y muestra el "Docstring" de la función
help(sum)

# Ayuda sobre un módulo completo para ver sus funciones disponibles
import math
help(math)

# dir() lista todos los métodos y atributos (la "anatomía" del objeto)
import pandas as pd
dir(pd.DataFrame)

# pydoc renderiza la documentación técnica en la terminal
import pydoc
print(pydoc.render_doc("math"))

# Ejemplo de "Demo" en Python (vía datasets de una librería SIG)
import geopandas as gpd
```

```python
# Listamos los mapas de ejemplo que vienen con la librería
print(gpd.datasets.available)
```

9.3.2 R

```r
# Inicia el servidor de ayuda local en formato HTML
help.start()

# Acceso rápido a la documentación de una función específica
?mean
help("plot")

# Ejecuta automáticamente el código de ejemplo del manual
example("lapply")

# --- Vignettes y Demos ---

# Abre el índice de tutoriales detallados (vignettes)
browseVignettes()

# Listar guías del paquete 'stars' (análisis de cubos de datos)
# Nota: presione "q" al final de la lista para liberar la consola
vignette(package = "stars")

# Visualizar una vignette específica por su nombre
vignette("stars1", package = "stars")

# Importante: requiere dispositivo gráfico (ej. httpgd::hgd())
# presione "q" al final de la lista para salir
demo()

# Demos específicos de librerías de Geomática
library(sf)
demo("nc", package = "sf")
demo("ggplot", package = "sf")
```

9.3.3 Julia

```julia
# Este comando (?) solo funciona dentro del REPL interactivo
# ?sum

# En Quarto usamos la macro @doc para acceder al Docstring
@doc sum
@doc println

# Julia usa docstrings con ejemplos y ayuda integrados, similar a Python
# Para encontrar "Demos", localizamos la carpeta del paquete en el disco
using DataFrames
println("Ubicación del código: ", pathof(DataFrames))
```

9.4 Instalación y carga de paquetes

Tarea	Descripción	Python	R	Julia
Instalar paquete	Agregar librerías externas	`pip3 install ...`	`install.packages()`	`Pkg.add()`
Cargar paquete	Habilitar funciones del paquete	`import ...`	`library()`	`using ...`

9.4.1 Python

```
#| label: instalacion_python
# #| eval: false

# En terminal usamos pip3
# para asegurar la versión de Python 3.x
# pip3 install colorama

# Dentro de python usamos el prefijo "!"
# !pip3 install colorama

# Cargamos solo lo necesario para manejar colores en consola
from colorama import Fore, Style

print("Librería cargada en Python")

# Aparecerán caracteres especiales al inicio del texto
# porque el HTML no entiende la instrucción colorear en rojo
# Fore.RED aplica el color; Style.RESET_ALL evita que el color "manche" las siguientes líneas
print(Fore.RED + "Librería cargada en Python" + Style.RESET_ALL)
```

9.4.2 R

```
# #| eval: false

# Este comando carga el paquete con require() o lo instala
# si no existe
if (!require("crayon", character.only = TRUE)) {
  install.packages("crayon", dependencies = TRUE, repos = "https://cran.rstudio.com/")
}
```

```
Loading required package: crayon
```

```
# Activamos el paquete para la sesión actual de R
library(crayon)

# cat() interpreta los códigos de escape ANSI que generan los colores
cat(red("Librería cargada en R"))
```

```
Librería cargada en R
```

9.4.3 Julia

```
#| label: instalacion_julia_ejecucion
#| kernel: julia-1.10
#| results: asis
# #| eval: false
#j_eval('
using Pkg
# Verificamos si "Crayons" ya está en el proyecto; si no, lo instalamos
if !haskey(Pkg.dependencies(), Base.UUID("a83e43d3-9d41-5979-9952-4e448995a975"))
    Pkg.add("Crayons")
end
using Crayons
println("Librería cargada en Julia")

# Aparecerán caracteres especiales al inicio del texto
# porque el HTML no entiende la instrucción colorear en rojo
println(Crayon(foreground=:red), "Librería cargada en Julia")
#')
```

9.5 Objetos básicos y estructuras de datos

El manejo eficiente de vectores y matrices es el corazón del procesamiento ráster. A continuación, se presenta una comparativa de cómo se declaran y manipulan los objetos básicos.

Tabla 9.3: Comparativa de estructuras de datos

Estructura	Descripción	Python	R	Julia
Vector	Colección lineal ordenada	`v = np.array([1, 2, 3])`	`v <- c(1, 2, 3)`	`v = [1, 2, 3]`
Matriz	Arreglo bidimensional (Ráster)	`m = np.zeros((3,3))` o `np.array([[...]])`	`m <- matrix(0, 3, 3)`	`m = zeros(3, 3)` o `[... ; ...]`
Diccionario	Pares clave-valor (Metadatos)	`{key: val}` ej. `{"id": 1}`	`list(k=v)` ej. `list(id = 1)`	`Dict(k=>v)` ej. `Dict("id"=>1)`
Data Frame	Estructura tabular (Atributos)	`pd.DataFrame()`	`data.frame()`	`DataFrame(...)`
Indexación	Posición del primer elemento	Inicia en **0**	Inicia en **1**	Inicia en **1**

9.5.1 Relación entre estructuras y componentes SIG

Para entender por qué usamos diferentes estructuras de datos, es útil ver cómo se mapean con los componentes de un Sistema de Información Geográfica:

> **Nota**
>
> Como se observa en la Figura 9.1, la **Matriz** es la estructura reina para el análisis ráster, mientras que los **DataFrames** son los que nos permiten realizar consultas sobre las bases de datos de predios o coberturas.

> **El peligro de la indexación espacial**
>
> Esta es la fuente más frecuente de errores. Mientras que en **Python** el primer píxel de una banda satelital es el `[0,0]`, en **R** y **Julia** es el `[1,1]`. Un error de este tipo desplazará todos sus resultados una celda, invalidando análisis de precisión o cambios de cobertura.

Nota sobre estructuras: En Geomática, los vectores suelen representar coordenadas o valores de píxeles, mientras que los diccionarios son fundamentales para manejar atributos (como en formato JSON o GeoJSON). Las tablas (DataFrames) son el estándar para bases de datos alfanuméricas de predios, municipios o estaciones climáticas.

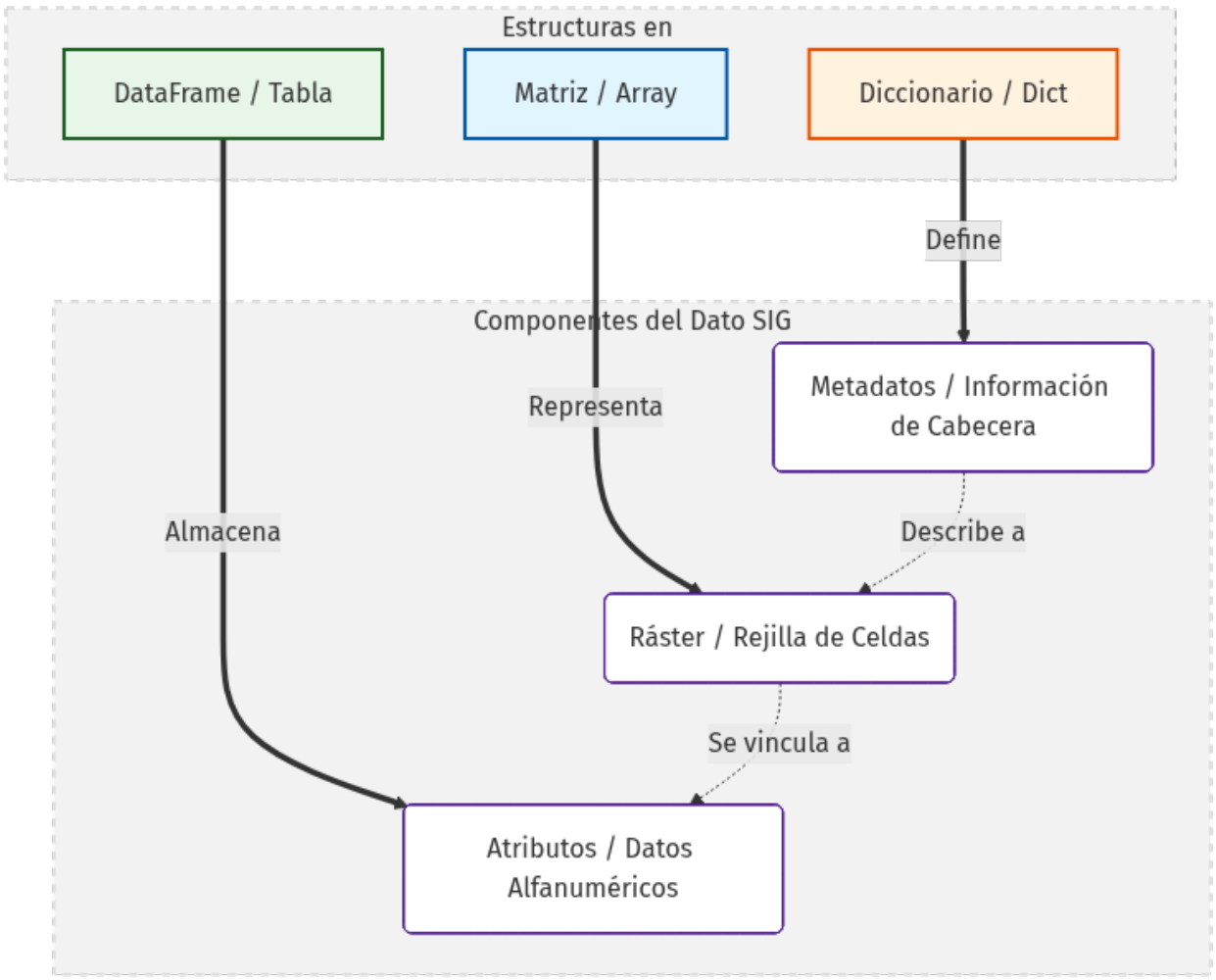

Estructuras en
DataFrame / Tabla
Matriz / Array
Diccionario / Dict
Define
Componentes del Dato SIG
Metadatos / Información de Cabecera
Representa
Almacena
Describe a
Ráster / Rejilla de Celdas
Se vincula a
Atributos / Datos Alfanuméricos

Figura 9.1

9.5.2 Python

```python
# #| eval: false
# 1. Listas: colecciones mutables de elementos
# Ejemplo: Códigos DANE de departamentos (Antioquia, Cundinamarca, Valle)
codigos = [5, 25, 76]
type(codigos)
```

```
<class 'list'>
```

```python
# 2. Diccionarios: Estructuras de clave-valor
# Ideal para representar metadatos de un departamento
metadatos = {
    "departamento": ["Antioquia", "Cundinamarca", "Valle"],
    "area_km2": [63612, 22623, 22140],
    "capital": ["Medellín", "Bogotá", "Cali"]
}
type(metadatos)
```

```
<class 'dict'>
```

```python
# 3. DataFrames: Para manejar tablas de atributos SIG
import pandas as pd
df_colombia = pd.DataFrame(metadatos)
type(df_colombia)
```

```
<class 'pandas.core.frame.DataFrame'>
```

```python
# Visualizamos el resumen técnico de la tabla
print(df_colombia.info())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3 entries, 0 to 2
Data columns (total 3 columns):
 #   Column        Non-Null Count  Dtype 
---  ------        --------------  ----- 
 0   departamento  3 non-null      object
 1   area_km2      3 non-null      int64 
 2   capital       3 non-null      object
dtypes: int64(1), object(2)
memory usage: 204.0+ bytes
None
```

9.5.3 R

```r
# #| eval: false
# 1. Vectores: La unidad básica en R (todos los elementos del mismo tipo)
# Códigos DANE de departamentos
codigos <- c(5, 25, 76)
class(codigos)
```

```
[1] "numeric"
```

```r
# 2. Listas: Pueden contener objetos de diferentes tipos y tamaños
# En R, las listas con nombres funcionan como los diccionarios
metadatos <- list(
  departamento = c("Antioquia", "Cundinamarca", "Valle"),
  area_km2 = c(63612, 22623, 22140),
  capital = c("Medellín", "Bogotá", "Cali")
)
class(metadatos)
```

```
[1] "list"
```

```r
# 3. Data Frame: La estructura tabular nativa por excelencia
df_colombia <- data.frame(metadatos)
class(df_colombia)
```

```
[1] "data.frame"
```

```r
# str() muestra la estructura interna del objeto (equivalente a info() en Python)
str(df_colombia)
```

```
'data.frame':   3 obs. of  3 variables:
 $ departamento: chr  "Antioquia" "Cundinamarca" "Valle"
 $ area_km2    : num  63612 22623 22140
 $ capital     : chr  "Medellín" "Bogotá" "Cali"
```

9.5.4 Julia

```
# #| eval: false
j_eval('
# 1. Vectores: Se definen con corchetes (similar a Python)
codigos = [5, 25, 76]

typeof(codigos)

# 2. Diccionarios: Se usa el operador => para asociar clave y valor
metadatos = Dict(
    "departamento" => ["Antioquia", "Cundinamarca", "Valle"],
    "area_km2" => [63612, 22623, 22140]
)

typeof(metadatos)

# 3. Tablas: En Julia Base se pueden usar vectores de Tuplas Nombradas,
# pero en Geomática siempre usaremos la librería DataFrames
using DataFrames
df_colombia = DataFrame(
    departamento = ["Antioquia", "Cundinamarca", "Valle"],
    area_km2 = [63612, 22623, 22140],
    capital = ["Medellín", "Bogotá", "Cali"]
)

typeof(df_colombia)

# describe() da un resumen estadístico de la tabla
println(describe(df_colombia))
')
```

```
Starting Julia ...
```

9.6 4. Comandos básicos y manipulación de tablas

Tarea	Descripción	Python	R	Julia
Directorio	Ruta de trabajo actual	`os.getcwd()`	`getwd()`	`pwd()`
Tipo / clase	Tipo de estructura	`type()`	`class()`	`typeof()`
Estructura	Resumen técnico	`df.info()`	`str()`	`describe(df)`
Dimensiones	Filas y columnas	`df.shape`	`dim()`	`size()`
Primeras filas	Vista rápida inicial	`df.head(n)`	`head(df, n)`	`first(df, n)`
Frecuencias	Conteo de categorías	`value_counts()`	`table()`	`countmap()`
Unir filas	Concatenar vertical	`pd.concat(..., axis=0)`	`rbind()`	`vcat()`
Omitir NA	Limpiar datos faltantes	`dropna()`	`na.omit()`	`dropmissing()`
Ordenar	Organizar por columna	`sort_values()`	`order()` / `arrange()`	`sort()`

9.6.1 Python

```
# #| eval: false
import os
import numpy as np
import pandas as pd

# 1. Gestión del entorno de trabajo
print(f"Ruta actual: {os.getcwd()}")
```

```
Ruta actual: /home/rstudio/work/01_prog_sig
```

```
# 2. Creación de datos: Departamentos de Colombia (Población en millones, Área en km2)
data = {
    "depto": ["Antioquia", "Cundinamarca", "Valle", "Bolívar", "Atlántico"],
    "pob_2023": [6.8, 3.2, 4.6, 2.2, 2.8],
    "area_km2": [63612, 22623, 22140, 25978, 3388],
    "region": ["Andina", "Andina", "Pacífica", "Caribe", "Caribe"]
}
df = pd.DataFrame(data)

# 3. Inspección de la tabla
print(df.info())          # Resumen de tipos de datos y memoria
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5 entries, 0 to 4
Data columns (total 4 columns):
 #   Column    Non-Null Count  Dtype
---  ------    --------------  -----
 0   depto     5 non-null      object
 1   pob_2023  5 non-null      float64
 2   area_km2  5 non-null      int64
```

```
 3   region    5 non-null      object
dtypes: float64(1), int64(1), object(2)
memory usage: 292.0+ bytes
None
```

```
print(f"Forma: {df.shape}") # (filas, columnas)
```

```
Forma: (5, 4)
```

```
print(df.head(3))          # Ver los primeros 3 registros
```

```
          depto  pob_2023  area_km2    region
0     Antioquia       6.8     63612    Andina
1  Cundinamarca       3.2     22623    Andina
2         Valle       4.6     22140  Pacífica
```

```
# 4. Análisis de frecuencias (¿Cuántos deptos por región?)
print(df["region"].value_counts())
```

```
region
Andina      2
Caribe      2
Pacífica    1
Name: count, dtype: int64
```

```
# 5. Operaciones de unión
# Crear un nuevo registro para un depto faltante
nuevo_depto = pd.DataFrame({"depto":["Chocó"], "pob_2023":[0.5], "area_km2":[46530],
↪  "region":["Pacífica"]})
# Concatenar verticalmente (axis=0 es por filas)
df_extendido = pd.concat([df, nuevo_depto], axis=0, ignore_index=True)

# 6. Ordenamiento
# Ordenar por población de mayor a menor
df_sorted = df_extendido.sort_values(by="pob_2023", ascending=False)
print(df_sorted)
```

```
          depto  pob_2023  area_km2    region
0     Antioquia       6.8     63612    Andina
2         Valle       4.6     22140  Pacífica
1  Cundinamarca       3.2     22623    Andina
4     Atlántico       2.8      3388    Caribe
3       Bolívar       2.2     25978    Caribe
5         Chocó       0.5     46530  Pacífica
```

```
# 7. Localización por condición (¿Dónde la población es > 4M?)
indices = np.where(df_extendido["pob_2023"] > 4.0)
print(f"Índices detectados: {indices}")
```

```
Índices detectados: (array([0, 2]),)
```

9.6.2 R

```r
# #| eval: false
# 1. Gestión del entorno
getwd()
```

```
[1] "/home/rstudio/work/01_prog_sig"
```

```r
# 2. Datos de ejemplo (Departamentos de Colombia)
df <- data.frame(
  depto = c("Antioquia", "Cundinamarca", "Valle", "Bolívar", "Atlántico"),
  pob_2023 = c(6.8, 3.2, 4.6, 2.2, 2.8),
  area_km2 = c(63612, 22623, 22140, 25978, 3388),
  region = c("Andina", "Andina", "Pacífica", "Caribe", "Caribe")
)

# 3. Inspección básica
str(df)          # Estructura del objeto
```

```
'data.frame':   5 obs. of  4 variables:
 $ depto   : chr  "Antioquia" "Cundinamarca" "Valle" "Bolívar" ...
 $ pob_2023: num  6.8 3.2 4.6 2.2 2.8
 $ area_km2: num  63612 22623 22140 25978 3388
 $ region  : chr  "Andina" "Andina" "Pacífica" "Caribe" ...
```

```r
dim(df)          # Dimensiones (filas y columnas)
```

```
[1] 5 4
```

```r
head(df, 3)      # Primeras 3 filas
```

```
         depto pob_2023 area_km2   region
1    Antioquia      6.8    63612   Andina
2 Cundinamarca      3.2    22623   Andina
3       Valle      4.6    22140 Pacífica
```

```r
# 4. Frecuencias (Equivalente a value_counts)
table(df$region)
```

```
  Andina   Caribe Pacífica
       2        2        1
```

```r
# 5. Uniones verticales
# Creamos el registro adicional
nuevo <- data.frame(depto="Chocó", pob_2023=0.5, area_km2=46530, region="Pacífica")
# rbind une por filas (necesita que las columnas se llamen igual)
df_extendido <- rbind(df, nuevo)

# 6. Ordenamiento
# order() devuelve los índices; los usamos para reindexar el dataframe
```

```r
df_sorted <- df_extendido[order(-df_extendido$pob_2023), ]
print(df_sorted)
```

```
         depto pob_2023 area_km2   region
1    Antioquia      6.8    63612   Andina
3        Valle      4.6    22140 Pacífica
2 Cundinamarca      3.2    22623   Andina
5    Atlántico      2.8     3388   Caribe
4      Bolívar      2.2    25978   Caribe
6        Chocó      0.5    46530 Pacífica
```

```r
# 7. Alternativa con dplyr (El estándar de "Tidyverse" para manipular tablas)
library(dplyr)
```

```
Attaching package: 'dplyr'

The following objects are masked from 'package:stats':

    filter, lag

The following objects are masked from 'package:base':

    intersect, setdiff, setequal, union
```

```r
# --- Opción A: Paso a paso con variables intermedias ---
# Útil para depurar y entender qué sucede en cada etapa

# Paso 1: Ordenar la tabla de departamentos por población de forma descendente
df_ordenado <- arrange(df_extendido, desc(pob_2023))
print(df_ordenado)
```

```
         depto pob_2023 area_km2   region
1    Antioquia      6.8    63612   Andina
2        Valle      4.6    22140 Pacífica
3 Cundinamarca      3.2    22623   Andina
4    Atlántico      2.8     3388   Caribe
5      Bolívar      2.2    25978   Caribe
6        Chocó      0.5    46530 Pacífica
```

```r
# Paso 2: Filtrar los resultados para quedarnos solo con los mayores a 4M
df_final <- filter(df_ordenado, pob_2023 > 4.0)
print(df_final)
```

```
      depto pob_2023 area_km2   region
1 Antioquia      6.8    63612   Andina
2     Valle      4.6    22140 Pacífica
```

```r
# --- Opción B: Uso de Operadores Pipe (Flujo continuo) ---
# El pipe permite "pasar" el resultado de una función a la siguiente sin crear variables nuevas.
```

```r
# 1. Pipe de la librería dplyr (el clásico: %>%)
# Se lee como: "Toma df_extendido, ENTONCES ordena, ENTONCES filtra"
df_extendido %>%
  arrange(desc(pob_2023)) %>%
  filter(pob_2023 > 4.0)
```

```
      depto pob_2023 area_km2   region
1 Antioquia      6.8    63612   Andina
2     Valle      4.6    22140 Pacífica
```

```r
# 2. Pipe nativo de R (disponible desde la versión 4.1: |>)
# Es más eficiente en memoria y no depende de cargar librerías extra
df_extendido |>
  arrange(desc(pob_2023)) |>
  filter(pob_2023 > 4.0)
```

```
      depto pob_2023 area_km2   region
1 Antioquia      6.8    63612   Andina
2     Valle      4.6    22140 Pacífica
```

9.6.3 Julia

```julia
# #| eval: false
j_eval('
using DataFrames

# Recreamos la tabla en el entorno de Julia
df = DataFrame(
    depto = ["Antioquia", "Cundinamarca", "Valle", "Bolívar", "Atlántico"],
    pob_2023 = [6.8, 3.2, 4.6, 2.2, 2.8],
    area_km2 = [63612, 22623, 22140, 25978, 3388],
    region = ["Andina", "Andina", "Pacífica", "Caribe", "Caribe"]
)

# Ejecutamos inspección y unión
nuevo = DataFrame(depto=["Chocó"], pob_2023=[0.5], area_km2=[46530], region=["Pacífica"])
df_ext = vcat(df, nuevo)

# Ordenar por población y mostrar
println(sort(df_ext, :pob_2023, rev=true))
')
```

9.6.4 Operaciones numéricas y matemáticas básicas

En el análisis espacial, estas operaciones son la base para calcular distancias, transformar coordenadas o procesar índices de vegetación (NDVI). A continuación, comparamos la sintaxis para las funciones matemáticas más comunes.

Tarea	Descripción	Python	R	Julia
División entera	Cociente sin decimales	`//`	`%/%`	`div()`
Módulo	Residuo de la división	`%`	`%%`	`%`
Raíz cuadrada	Cálculo de $x^{1/2}$	`math.sqrt()`	`sqrt()`	`sqrt()`

Tarea	Descripción	Python	R	Julia
Logaritmo	Logaritmo natural (ln)	`math.log()`	`log()`	`log()`
Constantes	Valores de π y e	`math.pi`, `math.e`	`pi`, `exp(1)`	`pi`, `exp(1)`

9.6.5 Python

```
# #| eval: false
import math

# 1. Aritmética entera (Útil para indexación de matrices)
print(f"División entera (5 // 2): {5 // 2}")
```

```
División entera (5 // 2): 2
```

```
print(f"Residuo/Módulo (5 % 2): {5 % 2}")
```

```
Residuo/Módulo (5 % 2): 1
```

```
# 2. Constantes matemáticas universales
print(f"Número e: {math.e}")
```

```
Número e: 2.718281828459045
```

```
print(f"Número pi: {math.pi}")
```

```
Número pi: 3.141592653589793
```

```
# 3. Funciones matemáticas (Requieren el módulo math)
print(f"Raíz cuadrada de 2: {math.sqrt(2)}")
```

```
Raíz cuadrada de 2: 1.4142135623730951
```

```
print(f"Logaritmo natural de 3: {math.log(3)}")
```

```
Logaritmo natural de 3: 1.0986122886681098
```

```
print(f"Logaritmo base 10 de 3: {math.log(3, 10)}")
```

```
Logaritmo base 10 de 3: 0.47712125471966244
```

```
# 4. Valor absoluto (Distancia sin dirección)
print(f"Absoluto de -3.4: {abs(-3.4)}")
```

```
Absoluto de -3.4: 3.4
```

9.6.6 R

```r
# #| eval: false

# 1. Aritmética entera
cat("División entera (5 %/% 2):", 5 %/% 2, "\n")
```

```
División entera (5 %/% 2): 2
```

```r
cat("Residuo/Módulo (5 %% 2):", 5 %% 2, "\n")
```

```
Residuo/Módulo (5 %% 2): 1
```

```r
# 2. Constantes (pi es nativo, e se obtiene con exp)
cat("Número e (exp(1)):", exp(1), "\n")
```

```
Número e (exp(1)): 2.718282
```

```r
cat("Número pi:", pi, "\n")
```

```
Número pi: 3.141593
```

```r
# 3. Funciones matemáticas (Nativas en Base R)
cat("Raíz cuadrada de 2:", sqrt(2), "\n")
```

```
Raíz cuadrada de 2: 1.414214
```

```r
cat("Logaritmo natural de 3:", log(3), "\n")
```

```
Logaritmo natural de 3: 1.098612
```

```r
cat("Logaritmo base 10 de 3:", log(3, 10), "\n")
```

```
Logaritmo base 10 de 3: 0.4771213
```

```r
# 4. Valor absoluto
cat("Absoluto de -3.4:", abs(-3.4), "\n")
```

```
Absoluto de -3.4: 3.4
```

9.6.7 Julia

```
# #| eval: false
j_eval('
# Ejecución de operaciones matemáticas vía j_eval
println("División entera Julia: ", div(5, 2))
println("Residuo Julia: ", 5 % 2)
println("Raíz de 2: ", sqrt(2))
println("Log natural 3: ", log(3))
println("Log base 10: ", log(10, 3)) # Inversión de argumentos frente a R/Python
println("Absoluto: ", abs(-3.4))
println("Pi: ", pi)
')
```

9.7 5. Lectura y escritura de datos

Tarea	Descripción	Python	R	Julia
Leer CSV	Cargar tabla desde texto/archivo	`pd.read_csv()`	`read.csv()`	`CSV.read()`
Escribir CSV	Guardar tabla en disco	`df.to_csv()`	`write.csv()`	`CSV.write()`

Para este ejercicio, utilizaremos una técnica de **simulación de archivos en memoria**. Esto es extremadamente útil en programación SIG para procesar datos que vienen de servicios web (APIs) antes de guardarlos físicamente.

9.7.1 Python

```
# #| eval: false
import pandas as pd
from io import StringIO

# Simulamos el contenido de un archivo CSV con municipios y altitudes (msnm)
csv_data = """
municipio,altitud
Bogota,2625
Medellin,1495
Cali,1018
Quibdo,43
"""

# StringIO convierte un texto en un "objeto de archivo" que pandas puede leer
df = pd.read_csv(StringIO(csv_data))

# Mostramos el resultado cargado
print(df)
```

```
  municipio  altitud
0    Bogota     2625
1  Medellin     1495
2      Cali     1018
3    Quibdo       43
```

```
# Guardamos en el disco (index=False evita que se guarde la columna de índices)
```

```
df.to_csv("municipios_altitud.csv", index=False)
```

9.7.2 R

```
# #| eval: false
# Contenido simulado
csv_data <- "
municipio,altitud
Bogota,2625
Medellin,1495
Cali,1018
Quibdo,43
"

# textConnection crea un flujo de lectura a partir de la cadena de texto
con <- textConnection(csv_data)
df <- read.csv(con)

# Es buena práctica cerrar la conexión después de usarla
close(con)

# Visualizamos la tabla
print(df)
```

```
  municipio altitud
1    Bogota    2625
2  Medellin    1495
3      Cali    1018
4    Quibdo      43
```

```
# Escribimos en el disco (row.names=FALSE para evitar la columna adicional de números)
write.csv(df, "municipios_altitud.csv", row.names = FALSE)
```

9.7.3 Julia

```
# #| eval: false
j_eval('
using CSV
using DataFrames

csv_data = "
municipio,altitud
Bogota,2625
Medellin,1495
Cali,1018
Quibdo,43
"

# Cargamos el string como DataFrame usando el buffer de memoria
df = CSV.read(IOBuffer(csv_data), DataFrame)

println("Tabla cargada en Julia:")
println(df)

# Escritura física
CSV.write("municipios_altitud.csv", df)
')
```

9.8 6. Indexación y filtrado

En esta sección aprenderemos a extraer subconjuntos de datos. En Geomática, esto es vital para aislar, por ejemplo, municipios que superen una altitud crítica o departamentos que pertenecen a una región específica.

Tarea	Descripción	Python	R	Julia
Seleccionar filas	Elegir registros por posición	`df.iloc[0:2]`	`df[1:2, ]`	`df[1:2, :]`
Filtrar por condición	Subset basado en reglas	`df[df["altitud"] > 1000]`	`df[df$altitud > 1000, ]`	`filter(row -> ...)`

9.8.1 Python

```
# #| eval: false
import pandas as pd

# Datos de municipios colombianos con su altitud (msnm)
df = pd.DataFrame({
    "municipio": ["Bogotá", "Medellín", "Cali", "Quibdó", "Barranquilla"],
    "altitud": [2625, 1495, 1018, 43, 18],
    "departamento": ["Cundinamarca", "Antioquia", "Valle", "Chocó", "Atlántico"]
})

# 1. Seleccionar filas por posición (índices 0 y 1)
# iloc permite acceso puramente posicional
primeros_dos = df.iloc[0:2]
print("Primeros dos municipios:\n", primeros_dos)
```

```
Primeros dos municipios:
   municipio  altitud  departamento
0    Bogotá     2625  Cundinamarca
1  Medellín     1495     Antioquia
```

```
# 2. Filtrar por condición (Municipios de "Tierras Altas" > 1500 msnm)
tierras_altas = df[df["altitud"] > 1500]
print("\nMmunicipios en tierras altas:\n", tierras_altas)
```

```
Municipios en tierras altas:
   municipio  altitud  departamento
0    Bogotá     2625  Cundinamarca
```

```
# 3. Filtrado con múltiples condiciones (Andinos y con altitud > 1000)
# Usamos & para 'y' lógico
andinos_altos = df[(df["altitud"] > 1000) & (df["departamento"] != "Chocó")]
```

9.8.2 R

```
# #| eval: false
df <- data.frame(
  municipio = c("Bogotá", "Medellín", "Cali", "Quibdó", "Barranquilla"),
```

```
  altitud = c(2625, 1495, 1018, 43, 18),
  departamento = c("Cundinamarca", "Antioquia", "Valle", "Chocó", "Atlántico")
)

# 1. Seleccionar filas por posición (En R los índices inician en 1)
primeros_dos <- df[1:2, ]
print(primeros_dos)
```

```
  municipio altitud departamento
1    Bogotá    2625 Cundinamarca
2  Medellín    1495    Antioquia
```

```
# 2. Filtrar por condición lógica
tierras_altas <- df[df$altitud > 1500, ]
print(tierras_altas)
```

```
  municipio altitud departamento
1    Bogotá    2625 Cundinamarca
```

```
# 3. Filtrado usando subset() - más legible en Base R
andinos_altos <- subset(df, altitud > 1000 & departamento != "Chocó")
print(andinos_altos)
```

```
  municipio altitud departamento
1    Bogotá    2625 Cundinamarca
2  Medellín    1495    Antioquia
3      Cali    1018        Valle
```

9.8.3 Julia

```
# #| eval: false
j_eval('
using DataFrames

df = DataFrame(
    municipio = ["Bogotá", "Medellín", "Cali", "Quibdó", "Barranquilla"],
    altitud = [2625, 1495, 1018, 43, 18],
    departamento = ["Cundinamarca", "Antioquia", "Valle", "Chocó", "Atlántico"]
)

# Ejecutamos un filtro y mostramos el resultado
println("Municipios con altitud > 1000 msnm:")
println(filter(row -> row.altitud > 1000, df))
')
```

9.9 7. Estadística descriptiva básica

El análisis estadístico permite entender la distribución de variables geográficas, como la precipitación acumulada o la densidad de población.

Tarea	Descripción	Python	R	Julia
Media	Promedio aritmético	`np.mean()`	`mean()`	`mean()`
Desviación	Medida de dispersión	`np.std()`	`sd()`	`std()`
Resumen	Estadísticos básicos	`df.describe()`	`summary()`	`describe()`

9.9.1 Python

```
# #| eval: false
import numpy as np
import pandas as pd

# Altitudes de una muestra de estaciones climáticas en los Andes colombianos
altitudes = np.array([2625, 1495, 1018, 2150, 1850])

# 1. Estadísticos individuales con Numpy
print(f"Altitud Media: {np.mean(altitudes)} msnm")
```

```
Altitud Media: 1827.6 msnm
```

```
print(f"Desviación Estándar: {np.std(altitudes):.2f}")
```

```
Desviación Estándar: 548.88
```

```
# 2. Resumen completo con Pandas
df_alt = pd.DataFrame(altitudes, columns=["msnm"])
print("\nResumen Descriptivo:")
```

```
Resumen Descriptivo:
```

```
print(df_alt.describe())
```

```
              msnm
count     5.000000
mean   1827.600000
std     613.670351
min    1018.000000
25%    1495.000000
50%    1850.000000
75%    2150.000000
max    2625.000000
```

9.9.2 R

```r
# #| eval: false
# Vector de altitudes msnm
altitudes <- c(2625, 1495, 1018, 2150, 1850)

# 1. Estadísticos descriptivos básicos
cat("Media:", mean(altitudes), "\n")
```

```
Media: 1827.6
```

```r
cat("Desviación Estándar:", sd(altitudes), "\n")
```

```
Desviación Estándar: 613.6704
```

```r
# 2. Resumen completo (Min, 1st Qu, Median, Mean, 3rd Qu, Max)
summary(altitudes)
```

```
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max. 
   1018    1495    1850    1828    2150    2625 
```

9.9.3 Julia

```julia
# #| eval: false
j_eval('
using Statistics
using DataFrames

altitudes = [2625, 1495, 1018, 2150, 1850]

println("Media aritmética en Julia: ", mean(altitudes))
println("Desviación estándar: ", std(altitudes))

# Mostramos el resumen tabular
df = DataFrame(msnm = altitudes)
println(describe(df))
')
```

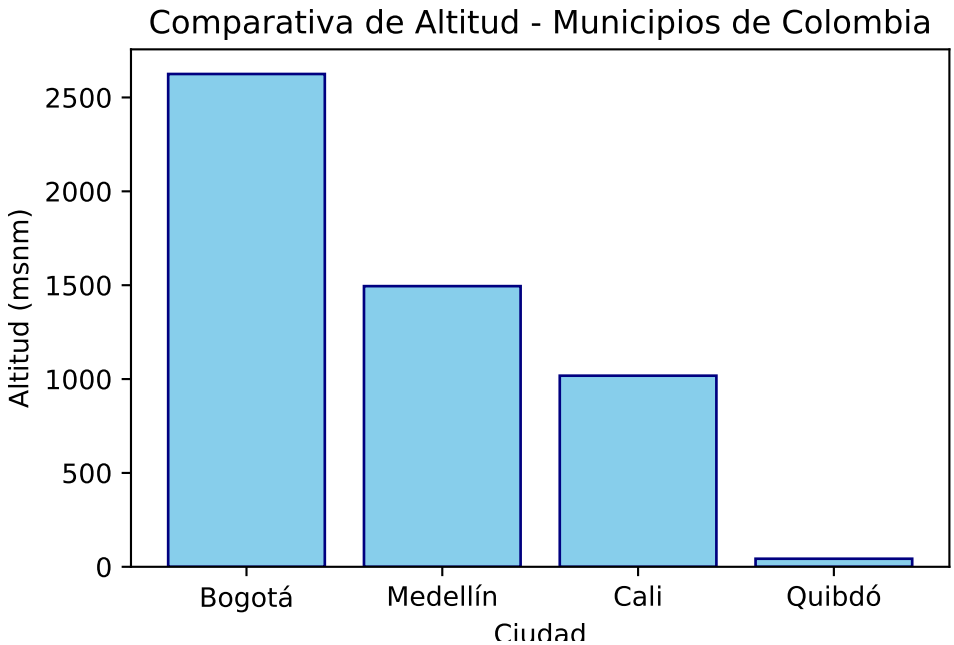
9.10 8. Gráficos básicos

La visualización es el primer paso del análisis exploratorio de datos (EDA). En SIG, usamos barras para comparar atributos entre regiones y los histogramas para entender la distribución de variables como la elevación o la precipitación.

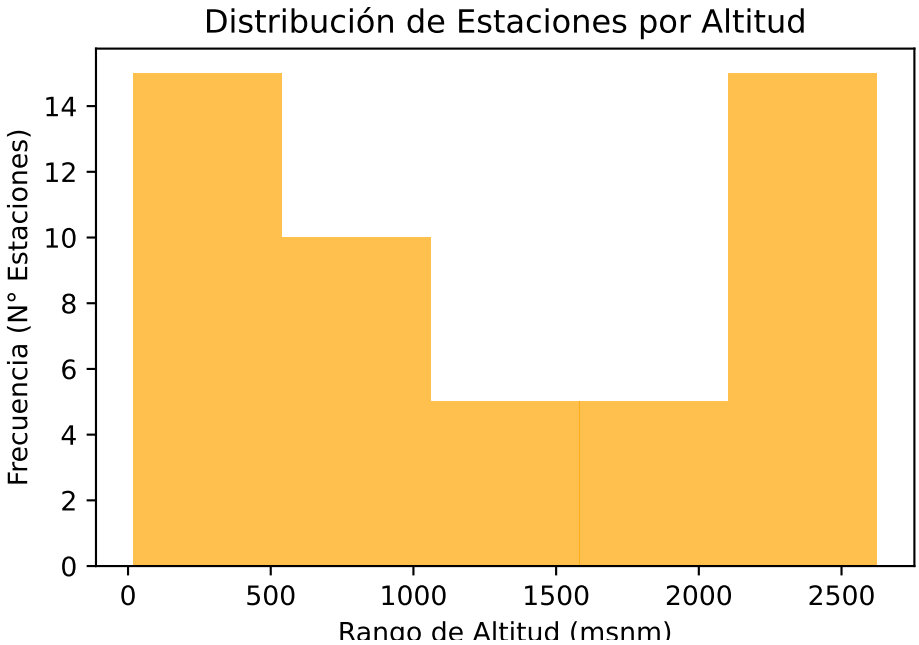
Tarea	Descripción	Python	R	Julia
Barras	Comparar categorías	`plt.bar()`	`barplot()`	`bar()`
Histograma	Distribución de frecuencias	`plt.hist()`	`hist()`	`histogram()`

```
plt.show()
```

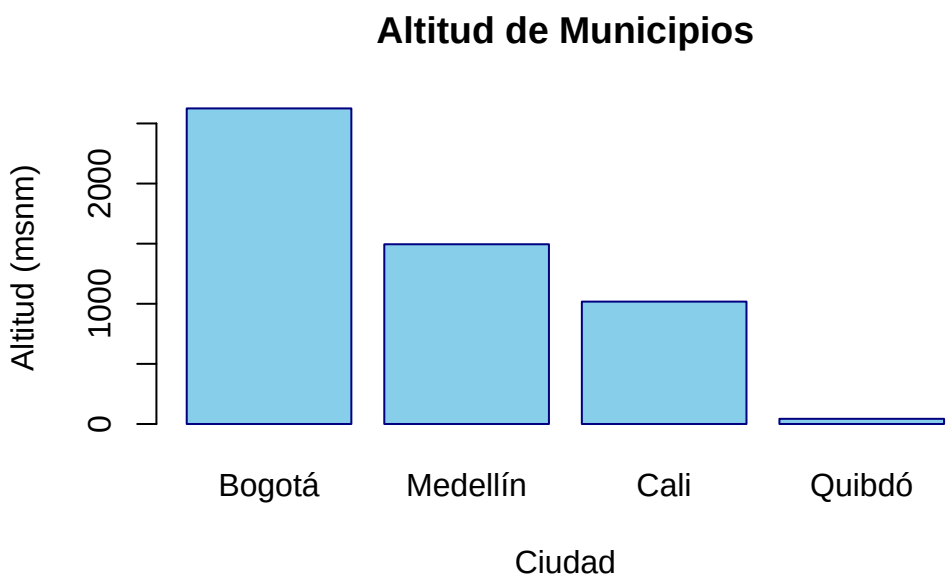

9.10.1 Python



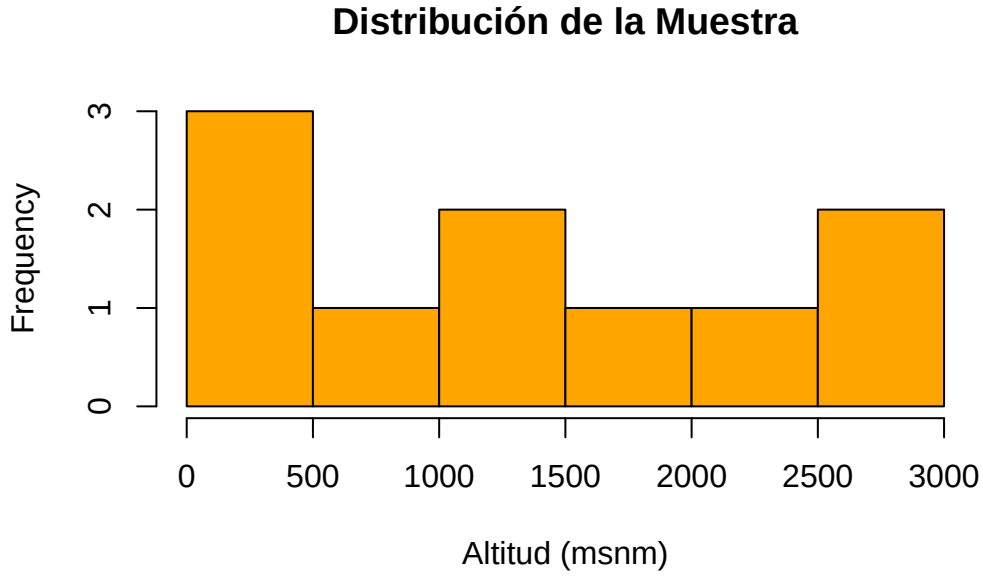

```
# 2. Histograma: Distribución de una muestra de altitudes
# Simulamos 50 estaciones climáticas en diferentes pisos térmicos
import numpy as np
muestra_altitudes = [2625, 1495, 1018, 43, 18, 2527, 959, 467, 2150, 1850] * 5

plt.hist(muestra_altitudes, bins=5, color='orange', alpha=0.7)
plt.xlabel("Rango de Altitud (msnm)")
plt.ylabel("Frecuencia (N° Estaciones)")
plt.title("Distribución de Estaciones por Altitud")
plt.show()
```




```r
# #| eval: false
# 1. Gráfico de Barras
ciudades <- c("Bogotá", "Medellín", "Cali", "Quibdó")
altitudes <- c(2625, 1495, 1018, 43)

# En R base, barplot es simple y potente
barplot(altitudes,
        names.arg = ciudades,
        col = "skyblue",
        border = "navy",
        xlab = "Ciudad",
        ylab = "Altitud (msnm)",
        main = "Altitud de Municipios")
```

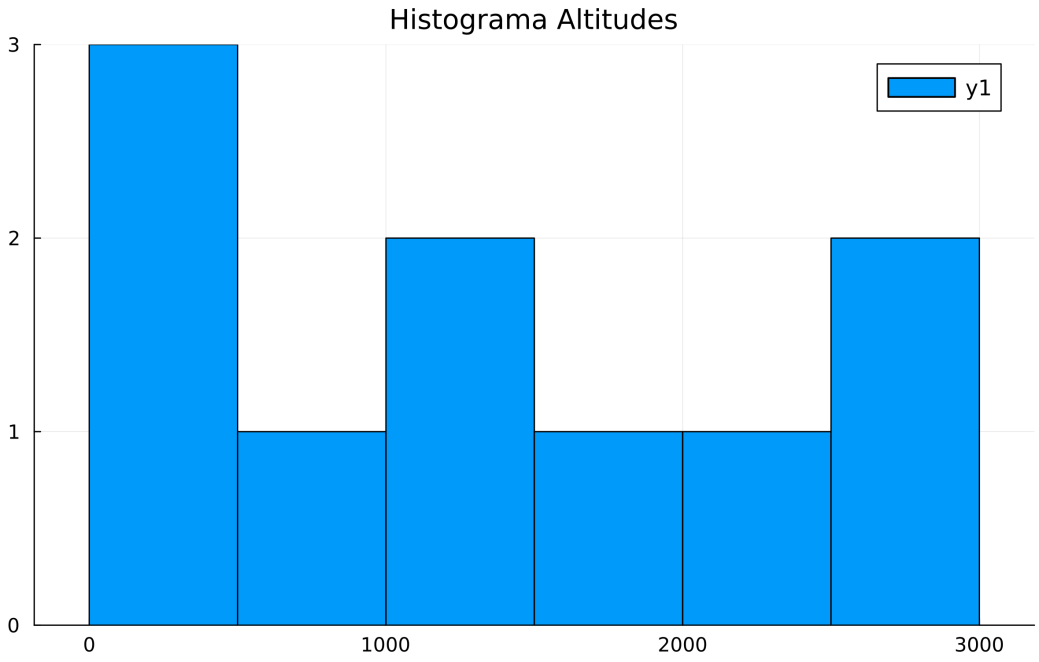



```r
# 2. Histograma
muestra_altitudes <- c(2625, 1495, 1018, 43, 18, 2527, 959, 467, 2150, 1850)
hist(muestra_altitudes,
     breaks = 5,
     col = "orange",
     xlab = "Altitud (msnm)",
     main = "Distribución de la Muestra")
```



9.10.3 Julia

```
# #| eval: false
j_plot('
using Plots

# Datos de Colombia
ciudades = ["Bogotá", "Medellín", "Cali", "Quibdó"]
altitudes = [2625, 1495, 1018, 43]

# En j_plot, el gráfico se genera en el dispositivo configurado
p1 = bar(ciudades, altitudes, title="Altitud Municipios", legend=false)
display(p1)

muestra = [2625, 1495, 1018, 43, 18, 2527, 959, 467, 2150, 1850]
p2 = histogram(muestra, bins=5, title="Histograma Altitudes")
display(p2)
')
```



9.11 9. Transformación de datos

En el análisis de datos espaciales, frecuentemente necesitamos derivar nuevas variables a partir de las existentes, como convertir unidades de medida (metros a pies) o categorizar valores (crear rangos climáticos).

Tarea	Descripción	Python	R	Julia
Crear columna	Nueva variable calculada	`df["nueva"] = ...`	`df$nueva <- ...`	`df.nueva = ...`

9.11.1 Python

```
# #| eval: false
```

```python
import pandas as pd

# Definimos un DataFrame con municipios y su altitud en metros (msnm)
df = pd.DataFrame({
    "municipio": ["Bogotá", "Medellín", "Cali"],
    "altitud_m": [2625, 1495, 1018]
})

# 1. Creación de una columna mediante una operación aritmética simple
# Convertimos metros a pies (aprox. 3.28 pies por metro)
df["altitud_ft"] = df["altitud_m"] * 3.28

# 2. Creación de una columna con lógica condicional (Clasificación climática)
# Usamos una función lambda para evaluar cada fila
df["clima"] = df["altitud_m"].apply(lambda x: "Frío" if x > 2000 else "Templado")

print(df)
```

```
  municipio  altitud_m  altitud_ft     clima
0    Bogotá       2625     8610.00      Frío
1  Medellín       1495     4903.60  Templado
2      Cali       1018     3339.04  Templado
```

9.11.2 R

```r
# #| eval: false
# Creamos el data frame base
df <- data.frame(
  municipio = c("Bogotá", "Medellín", "Cali"),
  altitud_m = c(2625, 1495, 1018)
)

# 1. Creación de columna usando el operador de asignación $
# La operación se aplica de forma vectorizada a toda la columna
df$altitud_ft <- df$altitud_m * 3.28

# 2. Creación de columna con lógica condicional usando ifelse()
df$clima <- ifelse(df$altitud_m > 2000, "Frío", "Templado")

print(df)
```

```
  municipio altitud_m altitud_ft    clima
1    Bogotá      2625    8610.00     Frío
2  Medellín      1495    4903.60 Templado
3      Cali      1018    3339.04 Templado
```

9.11.3 Julia

```julia
# #| eval: false
j_eval('
using DataFrames

# Recreamos el DataFrame en el entorno de Julia
df = DataFrame(
    municipio = ["Bogotá", "Medellín", "Cali"],
    altitud_m = [2625, 1495, 1018]
)

# Aplicamos las transformaciones
df.altitud_ft = df.altitud_m .* 3.28
df.clima = [x > 2000 ? "Frío" : "Templado" for x in df.altitud_m]

# Mostramos el resultado final
println(df)
```

```
')
```

9.12 10. Resumen comparativo (cheat sheet)

Esta tabla sirve como guía de referencia rápida para transitar entre los tres lenguajes durante el desarrollo de proyectos de análisis espacial.

Categoría	Tarea	Python	R	Julia
Ecosistema	Ayuda de función	`help(f)`	`?f`	`?f`
	Instalar paquete	`pip3 install x`	`install.packages("x")`	`Pkg.add("x")`
Estructuras	Vector / Lista	`[1, 2, 3]`	`c(1, 2, 3)`	`[1, 2, 3]`
	Diccionario / Mapa	`{"k": v}`	`list(k = v)`	`Dict(k => v)`
	Data Frame	`pd.DataFrame(d)`	`data.frame(d)`	`DataFrame(d)`
I/O	Leer CSV	`pd.read_csv("f.csv")`	`read.csv("f.csv")`	`CSV.read("f.csv", DF)`
Manipulación	Ver estructura	`df.info()`	`str(df)`	`describe(df)`
	Filtrar filas	`df[df.col > x]`	`df[df$col > x, ]`	`filter(r -> r.col > x, df)`
	Crear columna	`df["n"] = x * 2`	`df$n <- x * 2`	`df.n = x .* 2`
Análisis	Media	`np.mean(x)`	`mean(x)`	`mean(x)`
	Dimensiones	`df.shape`	`dim(df)`	`size(df)`
Visualización	Barras	`plt.bar(x, y)`	`barplot(y, names=x)`	`bar(x, y)`

9.12.1 Observaciones finales

Para evitar errores comunes en el procesamiento de datos geoespaciales, tenga siempre en cuenta estos tres pilares de la programación moderna:

9.12.1.1 1. Índices de arreglos (posicionamiento)

La forma en que los lenguajes cuentan las posiciones es la causa principal de errores en la extracción de coordenadas o píxeles:

- **Python:** Utiliza indexación **base-0** (el primer elemento está en la posición `0`).
- **R:** Utiliza indexación **base-1** (el primer elemento está en la posición `1`).
- **Julia:** Al igual que R, utiliza indexación **base-1**.

9.12.1.2 2. Vectorización y broadcasting (eficiencia)

La capacidad de operar sobre columnas completas sin usar bucles manuales (que son lentos) varía en su sintaxis y naturaleza:

- **Python:** La vectorización **no es nativa** de las listas base. Depende totalmente de librerías como `numpy` o `pandas`. Si intenta multiplicar una lista estándar por 2 (`[1, 2] * 2`), Python duplicará los elementos de la lista en lugar de realizar el cálculo matemático.
- **R:** La vectorización es **nativa y automática**. Casi todas las funciones de R están diseñadas para recibir un vector y devolver otro. Al hacer `vector * 2`, R entiende por defecto que debe multiplicarse cada elemento.
- **Julia:** Utiliza el concepto de **Broadcasting**. Es el más explícito: requiere añadir un **punto** antes del operador (`.*`, `./`, `.^`) o de la función. Este punto le indica al compilador de Julia que "esparza" la operación sobre todos los elementos del vector con una eficiencia comparable al lenguaje C.

9.12.1.3 3. El flujo de datos: el pipe (`.`, `%>%`, `|>`)

El "Pipe" permite escribir código que se lee de izquierda a derecha (como una receta), evitando el anidamiento excesivo de paréntesis.

- **Python (`pandas`):** Utiliza el **encadenamiento de métodos** mediante el punto (`.`). Cada operación devuelve un nuevo objeto sobre el cual se aplica la siguiente: `df.filter(...).sort(...)`.
- **R (`dplyr` / Nativo):** El pipe clásico de *Tidyverse* (`%>%`) o el nativo (`|>`) pasa el objeto automáticamente como **primer argumento** de la siguiente función: `df %>% filter(...)`.
- **Julia (`|>` / `Chain.jl`):** El pipe nativo (`|>`) es un operador de tubería simple. Para flujos de datos complejos y legibles, la comunidad de Julia prefiere la macro `@chain` del paquete `Chain.jl`.

9.12.2 Librerías de referencia (caja de herramientas)

Para que su entorno de trabajo esté completo, asegúrese de tener instaladas y cargadas estas librerías base:

1. **Python:** `pandas` (tablas de atributos), `numpy` (álgebra de mapas), `matplotlib` (salidas gráficas).
2. **R:** `base` y `dplyr` (manipulación), `graphics` (visualización rápida).
3. **Julia:** `DataFrames` (tablas), `CSV` (lectura), `Statistics` y `Plots`.

Nota sobre el rendimiento: Ninguno de los comandos vistos en este capítulo ejecuta procesamiento en paralelo (uso de múltiples núcleos). La vectorización y el pipe son herramientas de **eficiencia lógica y computacional en un solo núcleo**. El procesamiento multihilo se reservará para el análisis de grandes volúmenes de datos en capítulos posteriores.

9.13 Ejercicios

Para poner en práctica los conceptos introductorios abordados en este capítulo, deberás resolver los siguientes dos ejercicios. Puedes elegir resolverlos en **Python, R o Julia** (o implementar la solución en varios lenguajes si deseas retarte).

9.13.1 Ejercicio 1: Hidrología del Río Magdalena (Vectores y Matemáticas)

Contexto: Estás analizando el comportamiento de las estaciones hidrológicas del IDEAM a lo largo del Río Magdalena. Has recibido el reporte del caudal medio mensual (en metros cúbicos por segundo, m^3/s) de cinco

estaciones clave y necesitas extraer estadísticos básicos y transformar las unidades para un informe ambiental.

Instrucciones de código:

1. Define una colección ordenada (Vector/Lista/Array) llamada `estaciones` con los siguientes nombres: `"Honda", "Puerto Berrío", "Barrancabermeja", "Puerto Wilches", "Calamar"`.
2. Define otra colección paralela llamada `caudales_m3s` con los siguientes valores numéricos: `1500, 2100, 2800, 3200, 4500`.
3. Utilizando las funciones estadísticas nativas de tu lenguaje elegido, calcula e imprime:
   - El **caudal máximo** registrado en el río.
   - El **caudal promedio** de las cinco estaciones.
4. Para un estudio local, necesitas el caudal en **litros por segundo (l/s)**. Utilizando el concepto de *vectorización* (o *broadcasting*), multiplica la colección `caudales_m3s` por `1000` y guarda el resultado en una nueva variable llamada `caudales_ls`. (Asegúrate de no usar bucles `for`).
5. Imprime la colección resultante `caudales_ls`.

9.13.2 Ejercicio 2: Calidad del Aire en Bogotá (DataFrames y Filtrado)

Contexto: La Red de Monitoreo de Calidad del Aire de Bogotá ha publicado los datos promedio diarios de Material Particulado 2.5 ($PM_{2.5}$). Debes organizar estos datos en una estructura tabular, filtrar las estaciones que presentan riesgos para la salud (según la OMS) y visualizar los resultados.

Instrucciones de código:

1. Crea un **DataFrame** llamado `df_aire` (utilizando la librería adecuada según tu lenguaje: `pandas`, `data.frame` o `DataFrames.jl`) que contenga dos columnas:
   - `estacion`: `"Carvajal", "Kennedy", "Fontibón", "Suba", "Usaquén"`
   - `pm25`: `55, 42, 38, 15, 12`
2. Imprime un **resumen descriptivo/técnico** de tu DataFrame (usando la función equivalente a `info()`, `str()` o `describe()`).
3. El límite diario recomendado por la OMS para $PM_{2.5}$ es de 15 $\mu g/m^3$. Crea un nuevo DataFrame llamado `df_alerta` que **filtre** y contenga únicamente las estaciones donde el nivel de `pm25` sea **estrictamente mayor a 15**.
4. Crea una **nueva columna** en `df_alerta` llamada `estado` y asígnale a todas sus filas el valor de texto `"Crítico"`.
5. Utiliza la librería gráfica nativa de tu entorno para generar un **gráfico de barras** simple usando el DataFrame original (`df_aire`), donde el eje X sean las estaciones y el eje Y sean los niveles de $PM_{2.5}$.

9.13.3 Entregables y Criterios de Evaluación

El objetivo de esta evaluación no es solo que el código funcione, sino que seas capaz de documentar y explicar las diferencias fundamentales entre lenguajes.

1. Archivos de Código: Debes desarrollar los algoritmos en al menos uno de los siguientes formatos de archivo:

- Script tradicional (`.py`, `.R`, `.jl`)
- Notebook interactivo (`.ipynb`)

- Documento computacional (`.qmd` con *chunks* de código)

2. Documento Analítico (Quarto): Independientemente del formato de tu código fuente, **debes redactar un documento en Quarto (`.qmd`) y renderizarlo tanto en HTML como en PDF**. En este documento debes incluir tus bloques de código y responder argumentativamente a las siguientes preguntas:

- **Sobre el Ejercicio 1 (Vectorización):** Intenta explicar qué sucedería si decides hacer este ejercicio en **Python** y creas `caudales_m3s` como una Lista nativa (`[1500, 2100...]`) y la multiplicas directamente por 1000 (`caudales_m3s * 1000`). ¿Realizaría la operación matemática deseada? ¿Qué librería de Python soluciona este problema y cómo se diferencia este comportamiento del de R o Julia?
- **Pregunta General (Indexación):** Escribe la línea de código exacta que usarías en tu lenguaje elegido para extraer **las tres primeras estaciones** del DataFrame del Ejercicio 2 usando indexación por rangos (*slicing*). Explica brevemente si el límite superior del rango que escribiste se incluye o se excluye en el resultado final, justificando esto según el tipo de indexación (Base-0 vs Base-1) de tu lenguaje.

3. Repositorio en GitHub: Sube tu carpeta del proyecto (que debe contener tus scripts, el archivo `.qmd` y los renders finales en HTML y PDF) a un repositorio público en tu cuenta personal de **GitHub**.

- **Entrega:** Deberás enviar únicamente el enlace (URL) a tu repositorio de GitHub para la calificación.

Capítulo 10

Estructuras de Datos Geoespaciales

10.1 Funciones j_eval y j_plot en R

10.2 El Estándar Simple Features (ISO 19125)

La mayoría de las librerías modernas de programación SIG (Geopandas en Python, `sf` en R y `LibGEOS`/`ArchGDAL` en Julia) implementan el estándar **Simple Features Access** de la OGC. Este estándar define un modelo común para almacenar y acceder a geometrías en 2D.

Las geometrías fundamentales que utilizaremos son:

- **Point (Punto):** Un par de coordenadas (x, y).
- **LineString (Línea):** Una secuencia de puntos conectados.
- **Polygon (Polígono):** Un anillo cerrado que puede contener "huecos" (anillos interiores).

10.3 Creación de Geometrías Básicas

A continuación, definiremos la ubicación de la **Plaza de Bolívar en Bogotá** (aprox. $-74.076, 4.598$) usando los tres lenguajes.

10.3.1 Python (Shapely)

```python
# 1. Crear punto (Longitud, Latitud) para la Plaza de Bolívar
from shapely.geometry import Point, LineString

plaza_bolivar = Point(-74.076, 4.598)

# 2. Crear una línea (un segmento de la Carrera Séptima)
calle_septima = LineString([(-74.076, 4.598), (-74.075, 4.605)])

# 3. Imprimir propiedades
print(f"Tipo de geometría: {plaza_bolivar.geom_type}")
```

```
Tipo de geometría: Point
```

```
print(f"Representación WKT: {plaza_bolivar.wkt}")
```

```
Representación WKT: POINT (-74.076 4.598)
```

10.3.2 R (sf)

```
library(sf)
```

```
Linking to GEOS 3.12.1, GDAL 3.8.4, PROJ 9.4.0; sf_use_s2() is TRUE
```

```
# 1. Crear punto (Longitud, Latitud) para la Plaza de Bolívar
plaza_bolivar <- st_point(c(-74.076, 4.598))

# 2. Crear una línea (un segmento de la Carrera Séptima)
calle_septima <- st_linestring(rbind(c(-74.076, 4.598), c(-74.075, 4.605)))

# 3. Imprimir propiedades
print(plaza_bolivar)
```

```
POINT (-74.076 4.598)
```

10.3.3 Julia (LibGEOS)

```
j_eval(r"-(
using LibGEOS

# 1. Crear punto desde formato WKT (Plaza de Bolívar)
plaza_bolivar = LibGEOS.readgeom("POINT (-74.076 4.598)")

# 2. Imprimir propiedades
println("Tipo de geometría: ", typeof(plaza_bolivar))
)-")
```

```
Starting Julia ...
```

10.4 Atributos y Tablas Espaciales

Un SIG es la unión de **geometría + atributos**. Cada lenguaje tiene una estructura principal para manejar estas tablas:

Concepto	Python	R	Julia
Librería	`geopandas`	`sf`	`GeoTables.jl`
Estructura	`GeoDataFrame`	`sf` (data.frame)	`GeoTable`

10.4.1 Ejemplo: Ciudades Principales de Colombia

10.4.2 Python (GeoPandas)

```python
import geopandas as gpd
import pandas as pd
from shapely.geometry import Point

# 1. Datos alfanuméricos
df = pd.DataFrame({
    'Ciudad': ['Bogotá', 'Medellín', 'Cali'],
    'Pob_Millones': [7.9, 2.5, 2.2]
})

# 2. Geometrías (Lista de puntos: Longitud, Latitud)
geoms = [Point(-74.08, 4.60), Point(-75.56, 6.25), Point(-76.52, 3.42)]

# 3. Unión en GeoDataFrame asignando el sistema de referencia
gdf = gpd.GeoDataFrame(df, geometry=geoms, crs="EPSG:4326")
print(gdf)
```

```
     Ciudad  Pob_Millones             geometry
0    Bogotá           7.9   POINT (-74.08 4.6)
1  Medellín           2.5  POINT (-75.56 6.25)
2      Cali           2.2  POINT (-76.52 3.42)
```

10.4.3 R (sf)

```r
library(sf)

# 1. Datos alfanuméricos y coordenadas en un solo DataFrame
ciudades <- data.frame(
  Ciudad = c("Bogotá", "Medellín", "Cali"),
  Pob_Millones = c(7.9, 2.5, 2.2),
  lon = c(-74.08, -75.56, -76.52),
  lat = c(4.60, 6.25, 3.42)
)

# 2. Convertir a objeto espacial (sf) asignando columnas y CRS
gdf <- st_as_sf(ciudades, coords = c("lon", "lat"), crs = 4326)
print(gdf)
```

```
Simple feature collection with 3 features and 2 fields
Geometry type: POINT
Dimension:     XY
Bounding box:  xmin: -76.52 ymin: 3.42 xmax: -74.08 ymax: 6.25
Geodetic CRS:  WGS 84
    Ciudad Pob_Millones            geometry
1   Bogotá          7.9  POINT (-74.08 4.6)
2 Medellín          2.5 POINT (-75.56 6.25)
3     Cali          2.2 POINT (-76.52 3.42)
```

10.4.4 Julia (GeoTables)

```julia
j_eval(r"-(
import Pkg
Pkg.add(["GeoTables", "Meshes"])
```

```
using GeoTables, Meshes, DataFrames

# 1. Datos alfanuméricos
df = DataFrame(Ciudad = ["Bogotá", "Medellín", "Cali"], Pob_Millones = [7.9, 2.5, 2.2])

# 2. Geometrías (Llamamos a Meshes.Point explícitamente para evitar conflictos en memoria)
puntos = [Meshes.Point(-74.08, 4.60), Meshes.Point(-75.56, 6.25), Meshes.Point(-76.52, 3.42)]

# 3. Unión en GeoTable usando georef (el constructor estándar)
gt = georef(df, puntos)

println(gt)
)-")
```

10.5 Sistemas de Referencia de Coordenadas (CRS)

En Colombia, trabajamos principalmente con:

1. **WGS84 (EPSG:4326):** Grados decimales.
2. **MAGNA-SIRGAS / Origen Nacional (EPSG:9377):** Metros.

> **Importante**
>
> Nunca realice cálculos de área o distancia usando coordenadas en grados (`EPSG:4326`). Siempre proyecte a `EPSG:9377` para obtener resultados en metros.

10.6 Resumen de Aprendizajes (Cheat Sheet)

En este capítulo sentamos las bases del estándar Simple Features. A continuación, se presenta tu **Hoja de Referencia (Cheat Sheet)** para transitar entre las principales librerías espaciales de los tres lenguajes.

10.6.1 Funciones Centrales (Simple Features)

Tabla 10.2: Estructuras fundacionales para análisis espacial

Operación	Python (`shapely`/`geopandas`)	R (`sf`)	Julia (`LibGEOS`/`GeoTables`)
Crear Punto	`Point(x, y)`	`st_point(c(x, y))`	`readgeom("POINT(x y)")`
Crear Línea	`LineString([(x,y), ...])`	`st_linestring(rbind(...))`	`readgeom("LINESTRING(...)")`
Crear Tabla Geo	`GeoDataFrame(df, geometry=g)`	`st_as_sf(df, coords=...)`	`GeoTable(df, geometry=g)`
Asignar CRS	`crs="EPSG:4326"`	`crs = 4326`	Implícito / Vía `Proj.jl`

! Conclusiones Críticas del Módulo

1. **Orden de las Coordenadas:** Históricamente en geografía decimos "Latitud, Longitud" (Y, X). Sin embargo, en el estándar Simple Features y en programación pura, el orden matemático estricto es **siempre (X, Y)**, lo que equivale a **(Longitud, Latitud)**. Intercambiarlos ubicará sus datos en el océano o en otro país.
2. **Tablas Mágicas:** Un `GeoDataFrame` en Python o un `sf` en R no son más que tablas tradicionales (`DataFrames`) a las que se les ha inyectado una columna especial con "superpoderes" geométricos. Si borra esa columna, pierde las capacidades espaciales.

10.7 Ejercicios

Para consolidar tu comprensión de las geometrías básicas y las tablas espaciales, deberás resolver los siguientes ejercicios en **Python, R o Julia**.

10.7.1 Ejercicio 1: Topología del Río Cauca (Líneas y Puntos)

Contexto: Estás analizando estaciones de monitoreo de calidad del agua sobre un tramo del Río Cauca. Tienes las coordenadas de las estaciones y necesitas construir los objetos geométricos individuales para luego incluirlos en tu reporte base.

Instrucciones de código:

1. Utilizando las herramientas de creación de geometrías nativas de tu lenguaje, crea un **Punto** que represente la Estación Juanchito (Coordenadas aproximadas: `Lon: -76.45, Lat: 3.45`). Guárdalo en la variable `pt_juanchito`.
2. Crea un segundo **Punto** para la Estación Mediacanoa (`Lon: -76.38, Lat: 3.88`). Guárdalo en la variable `pt_mediacanoa`.
3. Imagina que el río conecta ambos puntos en línea recta. Crea una geometría tipo **Línea (LineString)** que una la coordenada de Juanchito con la de Mediacanoa. Guárdalo en la variable `tramo_cauca`.
4. Imprime el tipo de geometría (o la representación en texto/WKT) de tu variable `tramo_cauca`.

10.7.2 Ejercicio 2: Catastro de Zonas de Riesgo (Tabla Espacial)

Contexto: El equipo de Gestión del Riesgo te ha entregado un archivo tabular tradicional (CSV) con los barrios que reportan riesgo de inundación y sus coordenadas. Tu misión es convertir esta tabla "plana" en un objeto espacial oficial con un sistema de referencia.

Instrucciones de código:

1. Crea un **DataFrame** clásico con la siguiente información:
   - Columna `Barrio`: "La Virginia", "Puerto Mallarino"
   - Columna `Riesgo`: "Alto", "Medio"
   - Columna `Lon`: -75.88, -76.21
   - Columna `Lat`: 4.90, 3.95

2. Utiliza la función apropiada de tu librería espacial (`GeoPandas`, `sf` o `GeoTables`) para convertir ese DataFrame en una **Tabla Espacial**. Asegúrate de mapear las columnas `Lon` y `Lat` como las geometrías.
3. Al realizar la conversión, asigna explícitamente el sistema de coordenadas geográficas estándar (WGS84), es decir, el código **EPSG:4326**.
4. Imprime en consola tu nueva tabla espacial.

10.7.3 Entregables y Criterios de Evaluación

1. Archivos de Código: Debes desarrollar los algoritmos en al menos uno de los siguientes formatos de archivo: * Script tradicional (`.py`, `.R`, `.jl`) * Notebook interactivo (`.ipynb`) * Documento computacional (`.qmd` con *chunks* de código)

2. Documento Analítico (Quarto): Independientemente del formato de tu código fuente, **debes redactar un documento en Quarto (`.qmd`) y renderizarlo tanto en HTML como en PDF**. En este documento debes incluir tus bloques de código y responder argumentativamente a las siguientes preguntas:

- **Sobre la Topología (Ejercicio 1):** Basado en la lectura y tu experiencia práctica, ¿por qué es crítico respetar el orden matemático `(X, Y)` o `(Longitud, Latitud)` al momento de construir un `Point` o `LineString` en lugar de usar el orden verbal cotidiano "Latitud, Longitud"?
- **Sobre la Proyección:** El texto de la lección hace una advertencia sobre calcular áreas usando el `EPSG:4326` (Grados). Explica con tus palabras por qué medir un área en "grados cuadrados" carece de utilidad práctica en la ingeniería o el planeamiento territorial en Colombia, y por qué el `EPSG:9377` soluciona esto.

3. Repositorio en GitHub: Sube tu carpeta del proyecto a un repositorio público en tu cuenta personal de **GitHub**. * **Entrega:** Deberás enviar únicamente el enlace (URL) a tu repositorio de GitHub para la calificación.

Parte III

Prácticas

Capítulo 11

Evaluación Comparativa de Procesamiento Geoespacial

11.1 Introducción

- **Materia**: Programación SIG: Python, R, Julia
- **Práctica 1**: Sentinel-2 (1GB) - R (`terra`) vs R (`stars`) vs Python (`rasterio`) vs Julia (`ArchGDAL + Raster.jl`)
- **Autores**: Alexys Rodríguez-Avellaneda Ph.D. & herramientas IA

11.2 Funciones j_eval y j_plot en R

11.3 Preparación de los Datos: Sentinel-2A

En este ejercicio procesamos una escena de **Sentinel-2A** en formato `.zip`. En lugar de extraer el archivo (lo cual duplicaría el espacio en disco a casi 2GB), usamos el driver **VSI (Virtual Systems Interface)** de GDAL.

11.3.1 Anatomía de la Imagen

La imagen Sentinel-2 se organiza por bandas. Para este benchmark usaremos la **Banda 4 (Red)**, fundamental para el cálculo de índices de vegetación como el NDVI.

Banda	Resolución	Longitud de Onda	Uso
B02 (Blue)	10m	490 nm	Mapeo de aguas, suelos
B03 (Green)	10m	560 nm	Vigor de vegetación
B04 (Red)	**10m**	**665 nm**	**Absorción de clorofila**
B08 (NIR)	10m	842 nm	Biomasa, salud foliar

11.3.1.1 Descubrir la Ruta de la Imagen Sentinel-2

```
library(starsdata)
library(terra)
```

```
terra 1.8.93

Attaching package: 'terra'

The following object is masked from 'package:grid':

    depth
```

```
library(stars)
```

```
Loading required package: abind

Loading required package: sf

Linking to GEOS 3.12.1, GDAL 3.8.4, PROJ 9.4.0; sf_use_s2() is TRUE
```

```
library(reticulate)

# 1. Localización del ZIP dentro del paquete starsdata
f <- "sentinel/S2A_MSIL1C_20180220T105051_N0206_R051_T32ULE_20180221T134037.zip"
granule <- system.file(file = f, package = "starsdata")
granule
```

```
[1] "/usr/local/lib/R/site-library/starsdata/sentinel/S2A_MSIL1C_20180220T105051_N0206_R051_T32ULE_2018
```

```
base_name <- strsplit(basename(granule), ".zip")[[1]]
base_name
```

```
[1] "S2A_MSIL1C_20180220T105051_N0206_R051_T32ULE_20180221T134037"
```

```
# 2. Construcción de la ruta Virtual de GDAL (/vsizip/)
# Esta ruta permite leer directamente el XML de metadatos dentro del ZIP sin descomprimir.
s2_path <- paste0("SENTINEL2_L1C:/vsizip/", granule, "/", base_name,
 ↪  ".SAFE/MTD_MSIL1C.xml:10m:EPSG_32632")
s2_path
```

```
[1] "SENTINEL2_L1C:/vsizip//usr/local/lib/R/site-library/starsdata/sentinel/S2A_MSIL1C_20180220T105051_N
```

```
# Guardamos la ruta en un archivo compartido para que Python y Julia la lean
writeLines(s2_path, "s2_shared_path.txt")
```

11.4 Metodología de la evaluación comparativa (benchmarking)

Este experimento evalúa el rendimiento de cuatro motores geoespaciales ampliamente utilizados —`terra` (**R**), `stars` (**R**), `rasterio` (**Python**) y `Rasters.jl` (**Julia**)— frente a una operación numéricamente simple pero

computacionalmente exigente sobre datos raster de gran tamaño.

El flujo de trabajo consiste en:

1. Apertura del archivo raster Sentinel-2.
2. Selección de una sola banda (B4).
3. Aplicación de una operación aritmética escalar.
4. Cálculo de la **media global** (*mean*), que fuerza la evaluación completa del raster.

A diferencia de benchmarks centrados en *materialización explícita*, este experimento utiliza la operación `mean()` como **operación común de evaluación**, permitiendo que cada motor ejecute el cálculo conforme a su propio modelo interno de ejecución (*lazy vs eager*).

11.4.1 Dimensión del problema

Una banda Sentinel-2 a 10 m de resolución contiene:

$$10.980 \times 10.980 = 120.560.400 \text{ píxeles}$$

Asumiendo datos en punto flotante de 32 bits (4 bytes), el volumen teórico mínimo es:

$$120.560.400 \times 4 \approx 482{,}24 \text{ MB}$$

Este tamaño excede ampliamente la caché de CPU, por lo que el experimento está dominado por **I/O, acceso a memoria y eficiencia de recorrido**, no por complejidad algorítmica.

11.4.2 Exclusiones deliberadas

- La **generación de gráficos (plotting)** se ejecuta fuera del bloque cronometrado.
- El tiempo de renderizado y escritura en disco no refleja la velocidad de procesamiento numérico.
- En **Julia**, se realiza una ejecución previa (*warm-up*) para excluir el costo de compilación *Just-In-Time (JIT)* del tiempo reportado.
- En **Python** y **R**, el código numérico crítico se ejecuta en librerías ya compiladas (GDAL, NumPy, C/C++), por lo que no existe un costo de compilación comparable. Cualquier efecto de "calentamiento" en estos casos se limita a inicialización de librerías y caché de disco, y no altera de forma significativa los tiempos medidos.

11.4.3 Etapas del proceso evaluado

Etapa	Descripción técnica	Implementación por motor
1. Apertura del dataset	Lectura de metadatos y establecimiento de conexión al raster (sin carga completa a RAM)	**R / terra:** `rast()` **R / stars:** `read_stars(proxy = TRUE)` **Python / rasterio:** `rasterio.open()` **Julia / ArchGDAL + Rasters.jl:** `ArchGDAL.read()`
2. Selección de banda B4	Referencia a la banda espectral sin materializar todos los píxeles	**R / terra:** `r[[1]]` **R / stars:** `s[,,,1]` **Python / rasterio:** `src.read(1)` **Julia / Rasters.jl:** `Raster(ds)[Band(1)]`
3. Operación aritmética escalar	Multiplicación de cada píxel por un factor constante (1.5)	**R / terra:** `b4 * 1.5` **R / stars:** `b4 * 1.5` **Python / NumPy:** `b4 * 1.5` **Julia / Rasters.jl:** `r .* 1.5`
4. Reducción global (mean)	Cálculo de la media global, forzando el recorrido completo del raster	**R / terra:** `global(res_terra, "mean", na.rm = TRUE)[1, 1]` *(streaming)* **R / stars:** `mean(as.vector(res_mem[[1]]), na.rm = TRUE)` *(tras materialización explícita)* **Python / NumPy:** `res.mean()` *(array ya en RAM)* **Julia / Rasters.jl:** `mean(res)` *(streaming lazy)*

11.4.4 Interpretación clave del paso de reducción (`mean`)

El cálculo de la media es fundamental porque:

- Obliga a **recorrer todos los píxeles** del raster.
- Garantiza que la operación aritmética fue realmente ejecutada.
- Permite forzar la evaluación completa del flujo de cálculo sin introducir operaciones adicionales.

No obstante, **cada motor implementa este paso de forma distinta**: algunos realizan la reducción en *streaming* sin materializar el raster completo, mientras que otros requieren una materialización explícita en memoria. Estas diferencias responden a decisiones de diseño propias de cada librería y constituyen una limitación inevitable de la comparación.

11.4.5 Diferencias estructurales entre motores

Cada motor está optimizado para un tipo distinto de análisis. `terra` y `rasterio` están especialmente afinados para cálculos numéricos simples sobre grandes volúmenes de datos, mientras que `stars` y `Rasters.jl` priorizan

flexibilidad y modelos de datos más generales, lo cual puede afectar el rendimiento en operaciones simples como una media global.

Motor	¿Cómo trabaja internamente?	¿Qué implica en este benchmark?
terra (R)	Usa archivos raster "por referencia" y hace los cálculos en C++	Recorre el raster una sola vez de forma muy eficiente
stars (R)	Maneja los datos como cubos multidimensionales con mucha información espacial	Es más flexible, pero la media global es más lenta por el manejo de metadatos
rasterio (Python)	Carga la banda completa en un arreglo NumPy	Los datos quedan contiguos en memoria y se procesan muy rápido
Rasters.jl (Julia)	Evalúa las operaciones paso a paso y por bloques	Es muy general, pero en este caso introduce más sobrecarga

---

11.4.6 Limitaciones inevitables del benchmarking

Este benchmark **no mide qué lenguaje es "más rápido"**, sino cómo funciona **todo el conjunto de herramientas** que usa cada uno (librerías, forma de leer datos y manera de calcular).

En particular:

- **Python (rasterio)** es muy rápido porque lee la banda completa en memoria y usa arreglos NumPy optimizados.
- **terra (R)** está muy bien optimizado para hacer operaciones matemáticas sobre rasters usando código en C++.
- **stars (R)** se enfoca en manejar bien la información espacial y los metadatos, lo que hace más lenta una media global.
- **Julia (Rasters.jl)** está pensado para análisis espaciales más generales y flexibles, no para un único cálculo masivo como en NumPy.

Por eso, estos resultados deben interpretarse así:

> *Miden el rendimiento para una tarea específica (leer un raster y calcular una media),*
> *No un ranking general de lenguajes de programación.*

11.4.7 Interpretación del benchmark

Este benchmark representa un **caso extremo y muy simplificado**:

- Se utiliza **una sola banda raster**.
- Se aplica **una operación matemática trivial** (multiplicación escalar).
- Se calcula **una única media global**.

Por lo tanto, **no evalúa**:

- Análisis con múltiples bandas.
- Operaciones espaciales complejas (vecindarios, máscaras, reproyecciones).
- Flujos de trabajo largos, iterativos o modelos estadísticos.

El objetivo **no es declarar un "lenguaje ganador"**, sino **entender los costos reales** de: - leer los datos, - manejar las abstracciones, - y calcular una estadística global.

11.4.7.1 ¿Qué significa "manejar las abstracciones"?

Las abstracciones son capas de software que facilitan el trabajo del usuario.
Estas capas se encargan de:

- Leer los datos de forma segura.
- Mantener la información espacial (coordenadas, resolución, extensión).
- Coordinar las operaciones sin que el usuario controle cada paso.

Aunque hacen el código más claro y seguro, **introducen un costo adicional**, que se vuelve visible en operaciones simples y masivas, como una reducción global (`mean`).

11.4.7.2 Nivel de abstracción por motor

Motor / librería	Nivel de abstracción	Forma de trabajar (idea intuitiva)
Python / rasterio + NumPy	Baja	"Aquí tienes un arreglo de números en memoria, hagamos cuentas rápido"
R / terra	Media	"Yo manejo el raster y optimizo las operaciones por ti"
R / stars	Alta	"Además de los valores, manejo dimensiones, tiempo, atributos y geometría"
Julia / Rasters.jl	Flexible	"Construyo un flujo de operaciones que se evalúa cuando es necesario"

Idea clave:
> A mayor nivel de abstracción, mayor comodidad y expresividad para el usuario,
> pero también mayor costo computacional en operaciones simples como una media global.

11.5 Análisis de Rendimiento y Paralelismo

11.5.1 Benchmark en Python vs. Julia

11.5.1.1 Python (Rasterio)

`rasterio` es la navaja suiza de Python para rasters. Al combinarse con **NumPy**, utiliza instrucciones **SIMD** que paralelizan el cálculo a nivel de procesador (vectorización), aunque la lectura de GDAL sigue siendo

monohilo.

```
import rasterio
import numpy as np
import matplotlib.pyplot as plt
import time
import gc

# ------------------------------------------------
# 1. Leer ruta compartida
# ------------------------------------------------
with open("s2_shared_path.txt", "r") as f:
    s2_path = f.read().strip()

# ------------------------------------------------
# 2. WARM-UP (compila + cachea)
# ------------------------------------------------
with rasterio.open(s2_path) as src:
    _ = (src.read(1) * 1.5).mean()

gc.collect()
```

480

```
# ------------------------------------------------
# 3. BENCHMARK REAL
# ------------------------------------------------
t0 = time.perf_counter()

with rasterio.open(s2_path) as src:
    b4 = src.read(1)          # lectura banda 4
    res = b4 * 1.5            # operación
    m_py = res.mean()         # FORZADO REAL

t_python = time.perf_counter() - t0

print(f"🐍 Python: {t_python:.3f} seg | mean = {m_py:.6f}")
```

🐍 Python: 3.716 seg | mean = 3766.624630

```
# ------------------------------------------------
# 4. Plot (FUERA DEL BENCHMARK)
# ------------------------------------------------
plt.imshow(res, cmap="terrain")
```

<matplotlib.image.AxesImage object at 0x7f3e448ff3e0>

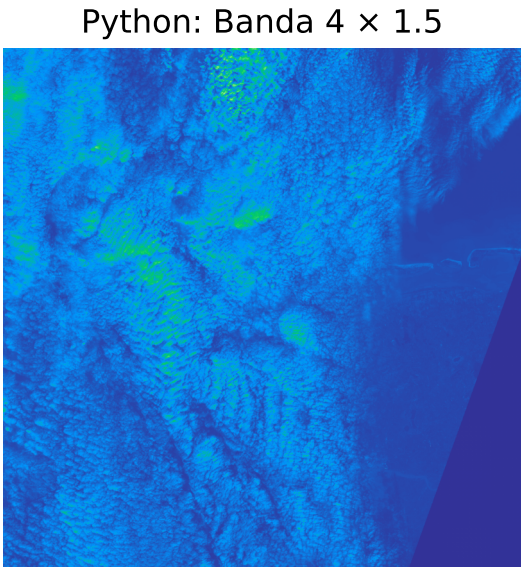
```
plt.title("Python: Banda 4 × 1.5")
```

Text(0.5, 1.0, 'Python: Banda 4 × 1.5')

```
plt.axis("off")
```

(np.float64(-0.5), np.float64(10979.5), np.float64(10979.5), np.float64(-0.5))

```
plt.show()
```




```
# ------------------------------------------------
# 5. Limpieza
# ------------------------------------------------
del b4, res
#gc.collect()
```

```
# Aquí es donde creamos t_python para R de la variable t_python en Python. Lo necesitamos en R para la
↪  tabla final.
# Extraemos el valor desde el objeto 'py'
t_python <- py$t_python
#t_python <- 0

cat("🐍 Tiempo capturado de Python:", round(t_python, 3), "seg.")
```

```
🐍 Tiempo capturado de Python: 3.716 seg.
```

11.5.1.2 ϟ Julia (Rasters.jl)

Julia es el único de los cuatro motores evaluados que puede explotar **paralelismo multihilo** en esta operación específica, sin recurrir a librerías externas adicionales, aprovechando los núcleos asignados al contenedor.

```
# 1. Desde R, llamamos a julia con j_eval (la función al inicio de este archivo o en el Rprofile)
# Ejecutamos dos veces:
#   la primera compila "costo de arranque" (JIT),
#   la segunda mide el tiempo
t_julia <- j_eval('
using Rasters, ArchGDAL, Statistics, Plots

# Evita restricciones artificiales de memoria
Rasters.checkmem!(false)

# ------------------------------------------------
# 1. Leer ruta compartida
# ------------------------------------------------
path = strip(read("s2_shared_path.txt", String))

# ------------------------------------------------
# 2. FUNCIÓN DE BENCHMARK (proxy + mean)
```

```
# ------------------------------------------------
function process_band_mean(path)
    ArchGDAL.read(path) do ds
        r   = Raster(ds)[Band(1)]   # proxy, solo banda 4
        res = r .* 1.5              # operación lazy
        return mean(res)            # FORZADO REAL (streaming)
    end
end

# ------------------------------------------------
# 3. WARM-UP (compilación)
# ------------------------------------------------
process_band_mean(path)
GC.gc()

# ------------------------------------------------
# 4. BENCHMARK REAL
# ------------------------------------------------
t0 = time_ns()
m_julia = process_band_mean(path)
t1 = time_ns()

t_julia = (t1 - t0) / 1e9

println("⏱ Julia: ", round(t_julia, digits=3), " seg | mean = ", round(m_julia, digits=6))


# ------------------------------------------------
# 5. Plot (FUERA DEL BENCHMARK, proxy)
# ------------------------------------------------
ArchGDAL.read(path) do ds
    r   = Raster(ds)[Band(1)]
    res = r .* 1.5
    p = plot(res, colormap = :terrain,
         title = "Julia: Banda 4 × 1.5")
    savefig(p, "julia_plot.png")
end

# Debe ser la última para que j_eval en R capture solo el número
t_julia
')
```

```
Starting Julia ...
```

```
#Si la función devuelve un texto lo convertimos a float
t_julia <- as.numeric(t_julia)

# 2. R muestra la imagen guardada por Julia en el HTML
knitr::include_graphics("julia_plot.png")
```

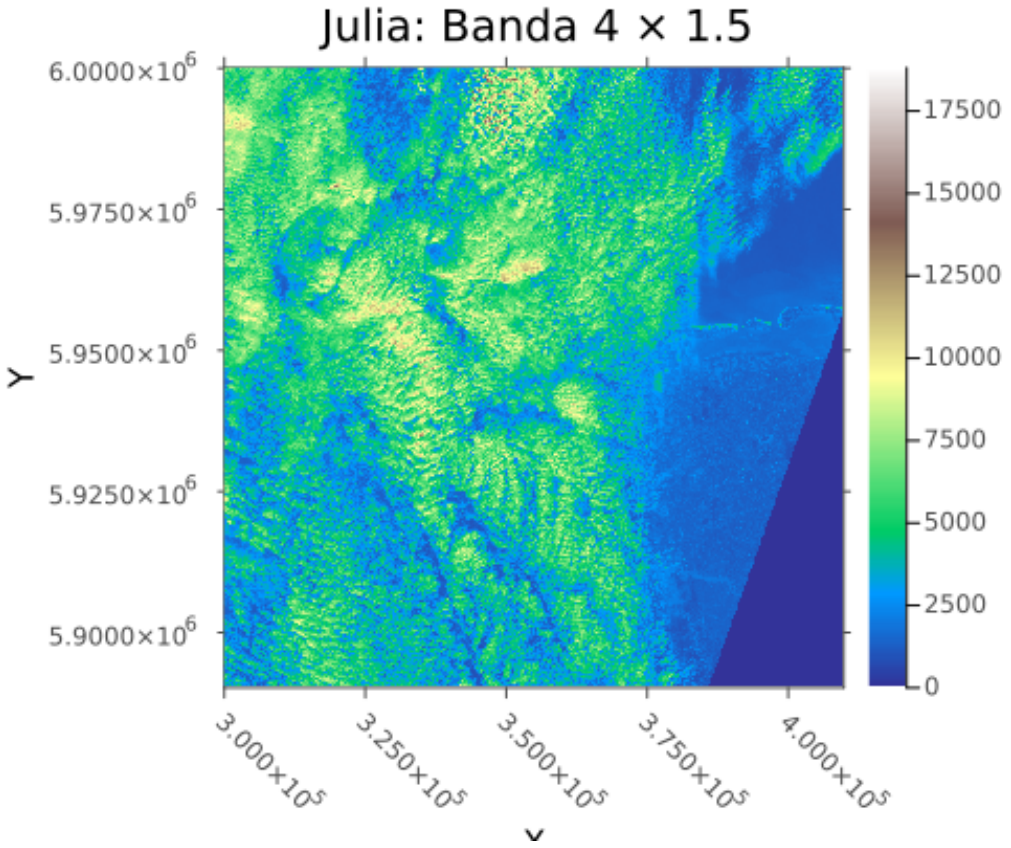


Figura 11.1: Procesamiento de alta resolución en Julia

```
# 3. Impresión desde R
print(paste("ϟ Tiempo capturado en R:", t_julia, " seg."))
```

[1] "ϟ Tiempo capturado en R: 13.030168471 seg."

11.5.2 Benchmark en R: Terra vs Stars

11.5.2.1 🏁 R: terra

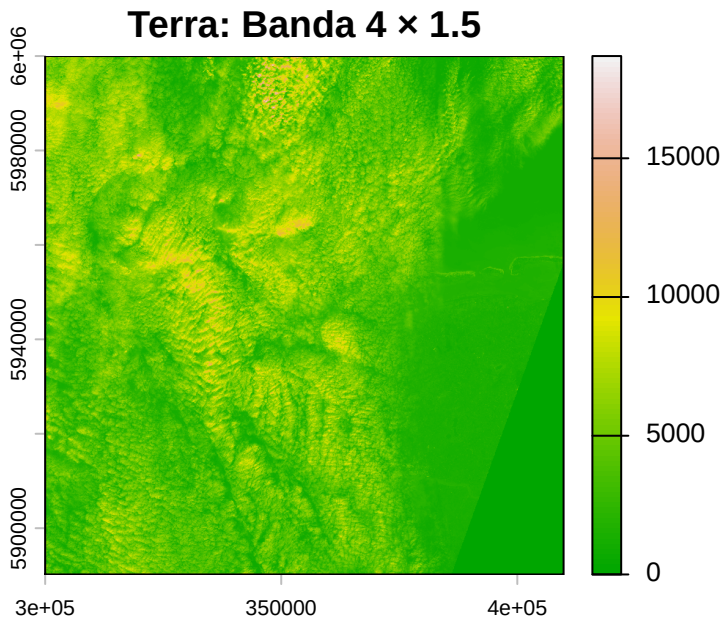
`terra` está desarrollado sobre C++. Su fortaleza es la velocidad de lectura y el manejo de memoria mediante punteros externos. **Paralelismo:** Para esta tarea (operación escalar), `terra` trabaja de forma **secuencial** (monohilo), confiando en la optimización de sus bucles en C++.

```
# library(terra)
# ------------------------------------------------
# 0. Inicio del cronómetro
# ------------------------------------------------
t0 <- Sys.time()

# ------------------------------------------------
# 1. Abrir raster en modo proxy (NO RAM)
# ------------------------------------------------
r <- rast(s2_path)

# ------------------------------------------------
# 2. Seleccionar solo la banda 4 (sigue siendo proxy)
# ------------------------------------------------
```

```
b4 <- r[[1]]

# ------------------------------------------------
# 3. Operación aritmética (lazy)
# ------------------------------------------------
res_terra <- b4 * 1.5

# ------------------------------------------------
# 4. FORZADO REAL (streaming, sin materializar)
# ------------------------------------------------
m_terra <- global(res_terra, "mean", na.rm = TRUE)[1, 1]

# ------------------------------------------------
# 5. Tiempo total
# ------------------------------------------------
t_terra <- as.numeric(Sys.time() - t0)

cat("⏱ Terra:",
    round(t_terra, 3), "seg |",
    "mean =", round(m_terra, 6), "\n")
```

```
⏱ Terra: 4.938 seg | mean = 3766.625
```

```
# ------------------------------------------------
# 6. Plot (FUERA DEL BENCHMARK, proxy)
# ------------------------------------------------
plot(res_terra, col = terrain.colors(100),
     main = "Terra: Banda 4 × 1.5")
```




```
# ------------------------------------------------
# 7. Limpieza
# ------------------------------------------------
rm(r, b4, res_terra)
gc()
```

```
         used  (Mb) gc trigger  (Mb) max used  (Mb)
Ncells 2614511 139.7    4928814 263.3  4928814 263.3
Vcells 5107929  39.0   10602366  80.9 10598174  80.9
```

11.5.2.2 ☆ R: stars

El paquete **stars** está especialmente diseñado para trabajar con **cubos de datos multidimensionales**, como múltiples bandas, series temporales y atributos espaciales complejos. Esta capacidad lo hace muy expresivo y adecuado para análisis espaciales avanzados.

Sin embargo, cuando se utiliza `proxy = FALSE`, los datos se **materializan completamente en la memoria de R**. En rasters de gran tamaño, esto puede introducir un mayor costo computacional asociado a:

- Lectura completa de los datos desde disco.
- Copia de grandes matrices a la memoria de R.
- Gestión de metadatos espaciales y dimensionales.

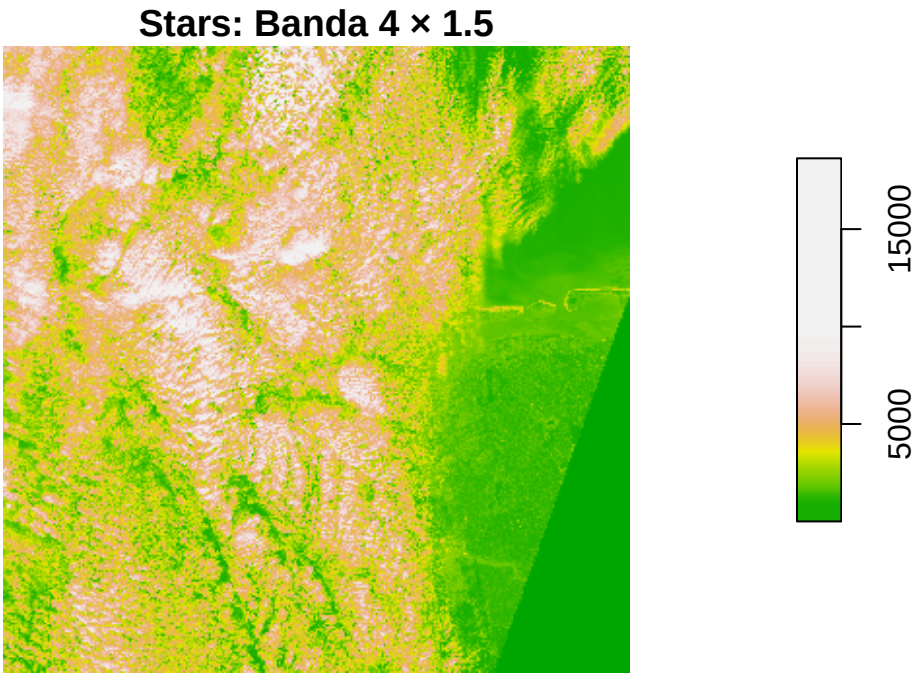
Paralelismo: En operaciones aritméticas simples —como una **multiplicación escalar** seguida de una **media global**— **ni stars ni terra garantizan paralelismo explícito por defecto**. En estos casos, el procesamiento suele realizarse de forma: - **Secuencial**, o
- **Por bloques**, dependiendo de la configuración interna del paquete y del backend utilizado (por ejemplo, GDAL).

```
# library(stars)

# ------------------------------------------------
# 0. Inicio del cronómetro
# ------------------------------------------------
t0 <- Sys.time()

# ------------------------------------------------
# 1. Leer raster como proxy (NO RAM)
# ------------------------------------------------
s <- read_stars(s2_path, proxy = TRUE)

# ------------------------------------------------
# 2. Seleccionar solo la banda 4 (proxy)
# ------------------------------------------------
b4 <- s[,,,1]

# ------------------------------------------------
# 3. Operación aritmética (lazy)
# ------------------------------------------------
res_stars <- b4 * 1.5

# ------------------------------------------------
# 4. FORZADO REAL (materializa la banda resultante)
# ------------------------------------------------
res_mem <- st_as_stars(res_stars)

# ------------------------------------------------
# 5. Media escalar (ya numérica)
# ------------------------------------------------
m_stars <- mean(as.vector(res_mem[[1]]), na.rm = TRUE)

# ------------------------------------------------
# 6. Tiempo total
# ------------------------------------------------
t_stars <- as.numeric(Sys.time() - t0)

cat("☆ Stars:",
    round(t_stars, 3), "seg |",
    "mean =", round(m_stars, 6), "\n")
```

```
☆ Stars: 8.325 seg | mean = 3766.625
```

```
# ------------------------------------------------
# 7. Plot (FUERA DEL BENCHMARK, proxy)
# ------------------------------------------------
plot(res_stars, col = terrain.colors(100),
     main = "Stars: Banda 4 × 1.5")
```

```
downsample set to 33
```




```
# ------------------------------------------------
# 8. Limpieza
# ------------------------------------------------
rm(s, b4, res_stars)
gc()
```

```
            used  (Mb) gc trigger    (Mb)  max used    (Mb)
Ncells    2665211 142.4    4928814   263.3   4928814   263.3
Vcells  124991287 953.7  562399569 4290.8 672242938 5128.9
```

11.6 Resultados finales

A continuación, se presenta la comparativa de rendimiento para procesar la Banda 4 (Red) de 10m desde el archivo comprimido original.

```
               Motor           Lenguaje          Paralelismo Tiempo_Seg
1          R: terra            R (C++)             Monohilo   4.938182
2          R: stars                  R             Monohilo   8.325164
3  Python: rasterio Python (C++/NumPy)   SIMD (Vectorizado)   3.716376
4 Julia: Rasters.jl     Julia (Nativo) Multihilo (12 hilos)  13.030168
```

Tabla 11.5: Duelo de Titanes: Procesamiento de 1GB Sentinel-2

Motor	Lenguaje	Paralelismo	Tiempo (s)	Eficiencia (X)
R: terra	R (C++)	Monohilo	4.938	2.64
R: stars	R	Monohilo	8.325	1.57

Python: rasterio	Python (C++/NumPy)	SIMD (Vectorizado)	3.716	3.51
Julia: Rasters.jl	Julia (Nativo)	Multihilo (12 hilos)	13.030	1.00

Los tiempos no deben compararse fuera del contexto de este patrón de acceso (lectura secuencial + reducción global).

> El benchmark favorece motores optimizados para **recorridos contiguos de memoria** y **reducciones monolíticas**, en particular **NumPy (vía rasterio en Python)** y **el motor C++ interno de `terra` en R**, los cuales pueden ejecutar la operación aritmética y el cálculo estadístico en una única pasada sobre un bloque contiguo de datos en memoria.

En contraste, motores basados en evaluaciones diferidas (*lazy evaluation*) y procesamiento por bloques con mayor carga de metadatos, como **`stars` en R** y **`Rasters.jl` en Julia**, incurren en mayor overhead de abstracción y llamadas intermedias, lo que afecta su desempeño relativo en este escenario específico.

1. **Eficiencia de Memoria:** `terra` es el ganador aquí, ya que su gestión de objetos fuera de la RAM de R le permite manejar archivos gigantes sin colapsar.
2. **Paralelismo Real:** Solo **Julia** aprovecha los hilos de ejecución de la CPU para la operación matemática de forma nativa. Python usa optimización de hardware (SIMD) vía NumPy, mientras que R se mantiene secuencial pero optimizado en sus librerías de C++.
3. **GDAL VSI:** Todos los lenguajes demostraron que el driver `/vsizip/` es la forma más eficiente de interactuar con datos Sentinel-2 sin el costo de descompresión.

11.6.1 Julia: paralelismo y pipelines composables

El potencial de paralelismo multihilo no siempre se traduce en mejores tiempos en benchmarks simples como el presente. Esto se debe a que Julia, a través de `Rasters.jl`, utiliza un modelo basado en **pipelines composables**.

Un **pipeline composable** significa que las operaciones no se ejecutan inmediatamente. En su lugar, Julia construye un *flujo de operaciones* (lectura → selección de banda → operación aritmética → reducción) que se evalúa solo cuando se solicita un resultado final, como la media global.

Este enfoque tiene ventajas claras en análisis complejos y encadenados, pero introduce un costo adicional de planificación y abstracción que se vuelve visible en tareas muy simples y masivas, como una única multiplicación seguida de una reducción global.

En otras palabras, Julia está optimizada para **flujos de trabajo complejos**, no para reducciones monolíticas de una sola pasada al estilo NumPy.

Herramienta / librería	¿Pipeline composable?	Ejemplo típico
Julia (Rasters.jl)	✔ Sí	`mean(r .* 1.5)` → se evalúa al final
R (dplyr + dbplyr)	✔ Sí*	Cadena de transformaciones luego `collect()`

Herramienta / librería	¿Pipeline composable?	Ejemplo típico
Python (xarray + Dask)	✔ Sí	`result = data.mean()` luego `compute()`
Apache Spark	✔ Sí	Plan de ejecución (DAG) antes de correr
Python (rasterio + NumPy)	✖ No	Lee y calcula todo inmediatamente
R (terra)	Parcial	Optimiza en C++ pero no expone pipeline diferido
R (stars)	✖ No	`proxy` limitado, sin DAG composable completo

* **Nota sobre R (dplyr + dbplyr):**
No es una solución espacial por sí misma. El pipeline composable existe, pero requiere un **backend espacial** (por ejemplo: PostGIS, DuckDB + spatial, Spark, BigQuery GIS). Sin ese backend, no aplica directamente a raster/cubos geoespaciales.

Nota sobre `stars`:
Aunque puede trabajar con `proxy = TRUE`, `stars` **no implementa un pipeline composable tipo tidyverse**, ni un DAG diferido completo. Las operaciones tienden a materializar datos relativamente pronto y no se integran con `dplyr/dbplyr` para optimización global del flujo.

11.7 Más allá del benchmarking: optimización y virtualización de datos geoespaciales

Este benchmark evalúa un caso simple y controlado, pero **los proyectos reales con grandes volúmenes de datos geoespaciales** rara vez dependen de un solo archivo raster leído de forma local. Hoy en día existen múltiples estrategias para **optimizar el rendimiento**, muchas de las cuales se basan en **virtualización del acceso a datos** y **formatos eficientes**.

Algunas de las principales alternativas que deben considerarse en proyectos de gran escala son:

11.7.1 Formatos optimizados para alto volumen

- **Cloud Optimized GeoTIFF (COG)**
  Permite leer solo las partes necesarias del raster mediante acceso por bloques y overviews, sin descargar el archivo completo.
- **Zarr / GeoZarr**
  Formato orientado a datos multidimensionales y computación distribuida. Muy eficiente para acceso parcial, paralelismo y almacenamiento en la nube.
- **GeoParquet**
  Formato columnar optimizado para datos vectoriales masivos. Ideal para análisis a gran escala, consultas selectivas y procesamiento distribuido.

11.7.2 Virtualización y acceso remoto

- **GDAL VFS (`/vsicurl/`, `/vsis3/`, `/vsiaz/`)**
  Permite trabajar con datos remotos como si fueran archivos locales, leyendo solo los bloques necesarios.
- **STAC (SpatioTemporal Asset Catalog)**
  Facilita la búsqueda y acceso estructurado a grandes catálogos de datos espaciales distribuidos.

11.7.3 Paralelismo y ejecución distribuida

- **Procesamiento por bloques y multihilo** (GDAL, terra, rasterio)
- **Frameworks distribuidos** como **Dask**, **Spark** o **Ray**, especialmente combinados con Zarr o Parquet.
- **Aceleración en la nube** mediante almacenamiento objeto y cómputo escalable.

11.7.4 Mensaje clave

Este ejercicio muestra los **costos mínimos inevitables** de leer, abstraer y reducir datos raster. Sin embargo, **la verdadera optimización en proyectos reales no suele venir de cambiar de lenguaje**, sino de:

- Elegir **formatos de datos adecuados**.
- Minimizar movimientos innecesarios de datos.
- Aprovechar **acceso parcial, paralelismo y virtualización**.
- Diseñar flujos de trabajo pensados desde el inicio para grandes volúmenes.

En resumen:
> *Cuando los datos crecen, la arquitectura y el formato importan tanto o más que el lenguaje.*

11.8 🏁 Desafío de laboratorio: primer día

Para cerrar esta sesión del **"Duelo de Titanes"**, deberán resolver el siguiente desafío práctico.
Pueden apoyarse en herramientas de IA para investigar, pero recuerden:

> **Buscamos precisión y evidencia**, no "carreta".

Este laboratorio incluye ejecución real de código en **distintos entornos**.

11.8.1 Instrucciones de entrega

1. Creen un nuevo repositorio en su **GitHub personal** llamado `taller1-sig`.
2. Todas las respuestas escritas deben estar en un archivo `respuestas.qmd`.
3. Rendericen `respuestas.qmd` a **HTML** y **PDF**.
4. Suban al repositorio:
   - `respuestas.qmd`
   - `respuestas.html`
   - `respuestas.pdf`
   - Los **notebooks** y **scripts** solicitados (ver abajo).

11.8.2 Parte A — Ejecución en JupyterLab (notebooks)

Ejecuten los **cuatro procesos del benchmark** desde **JupyterLab**, usando el kernel adecuado para cada lenguaje.

11.8.2.1 Notebooks obligatorios

Creen los siguientes notebooks:

- `01_benchmark_terra.ipynb`
- `02_benchmark_stars.ipynb`
- `03_benchmark_rasterio.ipynb`
- `04_benchmark_rasters_julia.ipynb`

Cada notebook debe:

- Leer el raster
- Aplicar la operación matemática ($\times$ 1.5)
- Calcular la **media global**
- Imprimir el **tiempo total de ejecución**

Entrega: - Suban los **cuatro notebooks** al repositorio. - En `respuestas.qmd`, incluyan: - El tiempo reportado por cada motor - Una breve observación (1–2 líneas) por notebook

---

11.8.3 Parte B — Ejecución desde VSCode (terminal integrada)

Ahora repitan el benchmark **fuera de Jupyter**, usando la terminal integrada de **VSCode**.

11.8.3.1 Scripts obligatorios

Creen los siguientes archivos:

- `benchmark_terra.R`
- `benchmark_stars.R`
- `benchmark_rasterio.py`
- `benchmark_rasters.jl`

Cada script debe:

- Leer el raster
- Ejecutar la operación
- Calcular la media global
- Imprimir:
  - El tiempo total
  - El valor de la media

11.8.3.2 Ejecución esperada

Desde la terminal de VSCode:

```
Rscript benchmark_terra.R
Rscript benchmark_stars.R
python3 benchmark_rasterio.py
julia benchmark_rasters.jl
```

Entrega: - Suban los **cuatro scripts** al repositorio. - Reporten los tiempos obtenidos en `respuestas.qmd`.

11.8.4 Parte C — Ejecución desde el Termina de Windows (PowerShell)

Finalmente, ejecuten los procesos **sin usar VSCode ni Jupyter**, directamente desde **Windows Terminal (PowerShell)**, trabajando con Docker.

Pueden usar **una o ambas opciones**.

11.8.4.1 Opción 1 — Entrando al intérprete

Ejemplos:

```
docker exec -it contenedor_sig_unal R
docker exec -it contenedor_sig_unal python3
docker exec -it contenedor_sig_unal julia
```

Y luego ejecutar el script correspondiente dentro del intérprete.

11.8.4.2 Opción 2 — Ejecución directa

Ejemplos:

```
docker exec contenedor_sig_unal Rscript benchmark_terra.R
docker exec contenedor_sig_unal Rscript benchmark_stars.R
docker exec contenedor_sig_unal python3 benchmark_rasterio.py
docker exec contenedor_sig_unal julia benchmark_rasters.jl
```

Entrega: - Indiquen en `respuestas.qmd`: - Qué opción usaron - Los tiempos obtenidos - Si notaron diferencias frente a JupyterLab o VSCode

---

11.8.5 Preguntas de análisis

11.8.5.1 1. Entorno de ejecución

¿Notaron diferencias de tiempo entre:

- JupyterLab
- VSCode (terminal integrada)
- Windows Terminal (PowerShell)

Den **una razón técnica posible** (overhead del kernel, entorno, proceso, etc.).

11.8.5.2 2. Abstracción en la práctica

¿En qué motor creen que el **costo de las abstracciones** es más visible?
Relacionen su respuesta con los tiempos observados.

11.8.5.3 3. Julia y el costo de compilación (Warm-up)

¿El efecto del *warm-up* de Julia se notó más en algún entorno específico?
Expliquen brevemente por qué.

11.8.5.4 4. Elección informada

Después de ejecutar el benchmark en **tres entornos distintos**,
¿cambiarían su elección del "Titán" para una emergencia ambiental real?

Justifiquen en **máximo 5 líneas**.

> **Nota para el éxito**
>
> Este laboratorio no busca que memoricen comandos,
> sino que entiendan que **el rendimiento depende del stack completo**: lenguaje, librerías, entorno y forma de ejecución.

11.9 Limpieza de recursos

Parte IV

Presentaciones

Capítulo 12

Temas por Charlar

Asunto: varios

12.1 Clase 1: Instalación y uso básico del software

- 1. Docker Desktop
- 2. Instalar nuestros contenedores
- 3. VSCode + Extensiones (dentro del nuestros contenerores)
- 4. Git + GitHub
- 5. Acceso a Jupyter Lab
- 6. Creación de archivos Quarto (*.qmd): HTML & PDF
- 7. OSGeo4W (QGIS) - QGIS + GEE (Pixi)
- 8. Chequeo ArcGIS Pro

12.2 Syllabus anterior - Contenido

- 1. Principios de Programación.
- 2. Fundamentos de Python.
- 3. Marcos de Datos.
- 4. Programación Orientada a Objetos.
- 5. Python y PostgreSQL.
- 6. Librerías Geoespaciales.
- 7. ArcGIS.

Necesitamos modificarlo.

12.3 Syllabus anterior - Estructura

- 1. Exposiciones del Profesor.
  - Teoría.
  - Práctica.
- 2. Trabajos en Grupo.
- 3. Exposiciones de los Estudiantes.
- 4. Lecturas.
- 5. Prácticas.
- 6. Trabajo Final.

Necesitamos concretarlo y asignar porcentajes.

12.4 Propuestas de Estudiantes para el Curso

Tienes la palabra.

12.5 Curso: Enfoque Open Science

- Material presentado por profesor
  - Diapositivas
  - Código
  - Teoría
  - Archivos de configuración
- Material de los estudiantes
- HTML, PDF
- GitHub

12.6 Seminario LatinGeo

- Organizado por GeoCorp y UNAL.
- Presentar trabajos finales.
- Presentar temas específicos.
- Abierto al público (en línea).
- Formato Open Science.
- Puede contribuir a la nota final.
- Podemos traer invitados.

LatinGeo: Python, R y Julia

12.7 Temas para Proyecto Final

- 1. Geocomputación en la Nube.
- 2. Virtualización.
- 3. Librerías Geo: Pipelines Composables.
- 4. Creación de Paquetes (R, Python, Julia).
- 5. Creación de Extensiones.
- 6. GitHub Actions.
- 7. DuckDB
- 8. GEE - GEEMAP
- 9. Temas aplicados
- 10. …

Capítulo 13

Benchmark geoespacial: cómo leer los resultados

Una comparación de stacks, no de lenguajes

13.1 ¿Qué estamos comparando?

Este ejercicio **no compara lenguajes de programación**.

Evalúa el rendimiento del **stack completo**:

- GDAL (Geospatial Data Abstraction Library): raster (GDAL) + vector (OGR: OpenGIS Simple Features Reference Implementation)
- Librerías raster
- Modelo de ejecución
- Nivel de abstracción

El lenguaje es solo una parte del sistema.

13.2 ¿Qué hace exactamente el benchmark?

El benchmark ejecuta un **caso extremo y muy simplificado**:

- Usa **una sola banda raster**
- Aplica **una operación matemática simple** ($\times$ 1.5)
- Calcula **una media global**

Diseñado para forzar la lectura completa de los datos.

13.3 ¿Qué NO evalúa?

Este benchmark **no evalúa**:

- Análisis multibanda
- Operaciones espaciales complejas
- Vecindarios, máscaras o reproyecciones
- Flujos iterativos o modelos estadísticos

No representa un flujo SIG real completo.

13.4 ¿Por qué usamos `mean()`?

El cálculo de la media es clave porque:

- Obliga a **recorrer todos los píxeles**
- Garantiza que la operación aritmética fue ejecutada
- Fuerza la evaluación completa del raster

Cada motor implementa este paso de forma distinta.

13.5 Modelos de ejecución comparados

Cada stack sigue una filosofía diferente:

- **Python / rasterio + NumPy**
  Lectura completa a memoria + ejecución inmediata
- **R / terra**
  Álgebra de ráster optimizado en C++ y procesamiento por bloques, pero sin exponer un pipeline diferido composable
- **R / stars**
  Manejo de cubos de datos multidimensionales y metadatos ricos; el modo `proxy` difiere la lectura, pero **no construye un DAG composable completo**
- **Julia / Rasters.jl**
  Flujos *lazy* y **pipelines composables**, evaluados al final como un plan coherente

---

13.6 ¿Qué favorece este benchmark?

Este escenario favorece motores optimizados para:

- Recorridos contiguos de memoria
- Operaciones simples en una sola pasada
- Reducciones globales monolíticas

No todos los motores están diseñados para este patrón.

13.7 Abstracción vs rendimiento

Más abstracción implica:

- Más metadatos
- Más coordinación interna
- Más costo administrativo

Pero también ofrece:

- Código más claro
- Menos errores
- Mayor expresividad analítica

Es un intercambio inevitable.

13.8 Niveles de abstracción por motor

Motor	Nivel de abstracción	Forma de trabajar
NumPy / rasterio	Baja	"Aquí tienes un arreglo, calcula ahora"
terra (R)	Media	"Yo optimizo internamente en C++"
stars (R)	Alta	"Gestiono dimensiones, tiempo y metadatos"
Rasters.jl (Julia)	Flexible	"Defino un pipeline que se evalúa al final"

Más abstracción = más expresividad, pero más costo administrativo.

13.9 Paralelismo y pipelines composables

Solo algunos stacks permiten **componer operaciones** y ejecutarlas al final:

- **Julia (Rasters.jl)**
  Pipelines composables + **paralelismo multihilo nativo**,
  sin librerías externas adicionales
- **Python (xarray + Dask)**
  Pipelines lazy con ejecución distribuida explícita
- **R (dplyr + dbplyr)**
  Lenguaje de pipelines, **no espacial por sí mismo**;
  requiere backends como PostGIS, DuckDB o Spark

Optimización interna **no equivale** a pipeline composable.

13.10 No hay un ganador universal

Este benchmark mide:

> Rendimiento bajo un patrón específico de acceso y reducción global

No mide:

- Calidad general del lenguaje
- Flexibilidad analítica
- Escalabilidad en flujos SIG complejos

Los resultados deben interpretarse con contexto.

13.11 Escalando a datos realmente grandes

En proyectos reales con grandes volúmenes de datos se consideran:

- Cloud Optimized GeoTIFF (COG)
- Zarr
- GeoParquet
- Procesamiento por bloques
- Infraestructura virtualizada - cloud / HPC - High-Performance Computing (AWS - GCP - Azure)

La **arquitectura de datos** suele importar más que el lenguaje.

13.12 Mensaje final del Benchmark

Este ejercicio sirve para:

- Entender costos reales de lectura y abstracción
- Leer benchmarks de forma crítica

- Elegir herramientas según el problema

No existe el “lenguaje más rápido”
existe el **stack adecuado para cada tarea**.

13.13 ¿Qué es lo importante hoy (y qué no)?

En este punto del curso:

✗ **No es importante** entender cada línea de código
✗ No es importante memorizar sintaxis
✗ No es importante “ser rápido programando”

✓ **Sí es importante**:

- Ver el **panorama completo** de la programación SIG actual

- Entender que existen **múltiples stacks y enfoques**
- Reconocer que el rendimiento depende de **arquitectura**, no solo del lenguaje

El código lo aprenderemos paso a paso.

13.14 ¿Qué estamos aprendiendo realmente?

Más allá del benchmark, este laboratorio busca que ustedes aprendan a:

- Usar **GitHub** como bitácora de trabajo
- Documentar con **Quarto**
- Ejecutar análisis en **Jupyter Lab**
- Trabajar en **VSCode**
- Usar la **terminal** (Windows / Linux)
- Ejecutar entornos reproducibles con **Docker**
- Correr el mismo proceso **de muchas formas distintas**

Programar SIG hoy es saber **orquestar herramientas**, no solo escribir código.

13.15 El límite lo ponen ustedes

En clase aprenderemos:

- Los conceptos fundamentales
- Las herramientas base
- Los patrones comunes de trabajo

Pero el verdadero aprendizaje vendrá de:

- Sus **proyectos**
- Su **trabajo individual**
- Lo que decidan explorar más allá del aula

En programación SIG,
el límite no lo pone el lenguaje, lo pone la curiosidad y el problema que quieran resolver.

Apéndice A

Resumen instalación de herramientas

A.1 Introducción

Resumen de herramientas requeridas:

Herramienta	Uso	Detalles
git	-	Apéndice E
GitHub + SSH (local y contenedor)	-	Sección E.2, Sección E.6, Sección E.7
VSCode + Extensiones + Tinytex (local y contenedor)	-	Apéndice F, Sección F.6, Sección F.5
Docker Desktop + Preparación inicial	-	Apéndice D, Sección D.2
QGIS (OSGeo4w & PIXI + GEE)	-	Sección L.1.1, Sección L.1.2
ArcGIS Pro	-	Apéndice M

Apéndice B

Uso de la Infraestructura Instalada

B.1 Configuración inicial del entorno (Pre-flight)

Para automatizar el soporte de gráficos y capturas sin configurar cada archivo individualmente, ejecute estos comandos en la terminal del contenedor `contenedor_sig_unal` inmediatamente después de iniciar los servicios con `docker compose up -d`:

Tabla B.1: Comandos de configuración global del contenedor

Categoría	Comando de Configuración	Propósito
Automatización R	`echo 'Sys.setenv(CHROMOTE_CHROME = "/opt/google/chrome/chrome")' >> /etc/R/Rprofile.site`	Configura la ruta de Chrome para todo el sistema R de forma persistente. (si falla usa `/usr/lib/R/etc/Rprofile.site`. El `R.home("etc")` en nuestro sistema es `/usr/lib/R/etc`)
Seguridad Root	`echo 'options(chromote.args = c("--no-sandbox", "--disable-gpu", "--headless", "--remote-debugging-port=9222"))' >> /etc/R/Rprofile.site`	Habilita el modo *headless* y evita bloqueos de *sandbox* al ejecutar como root. (si falla usa `/usr/lib/R/etc/Rprofile.site`. El `R.home("etc")` en nuestro sistema es `/usr/lib/R/etc`)

/usr/bin/google-chrome /usr/lib/R/etc/Rprofile.site /etc/R/Rprofile.site

Google Chrome se necesita para compilar código `mermaid` dentro de Quarto. Ejecute uno de lo siguientes comandos para comprobar si Google Chrome web browser está instalado correctamente:

```
google-chrome --version
```

```
which google-chrome
```

```
quarto check
```

La imagen cargada del archivo tar ya incluye los programas descritos a continuación en Tabla B.2. Esta tabla se deja como referencia para facilitar una futura compilación. La tabla descrita líneas abajo servirá para complementar el archivo Dockerfile.

Tabla B.2: Comandos de configuración global del contenedor ya incluidos en la imagen cargada a partir del archivo .tar

Categoría	Comando de Configuración	Propósito
Soporte Base	`apt-get update && apt-get install -y fonts-symbola wget`	Fuentes para emojis y herramienta de descarga de paquetes externos.
Navegador	`wget https://dl.google.com/linux/direct/google-chrome-stable_current_amd64.deb && apt install -y ./google-chrome-stable_current_amd64.deb`	Instala Google Chrome (Bypass de Snap) Obligatorio para capturas de mapas en Docker.
Librerías R	`Rscript -e "install.packages('webshot2', repos='https://cloud.r-project.org/')"`	Instala el motor de captura de widgets HTML y mapas de Leaflet.
Librerías Python	`pip3 install selenium`	Habilita la automatización de capturas para visualizaciones dinámicas de Python.
Librerías Julia	`julia -e 'using Pkg; Pkg.add(["FileIO", "ImageIO"])'`	Soporte esencial para procesar y exportar gráficos en el ecosistema Julia.

B.1.1 Ventajas de esta configuración

Al usar el archivo `Rprofile.site`, hemos logrado que:

1. **Limpieza:** Sus archivos `.qmd` solo contendrán código de análisis geográfico, eliminando bloques de configuración de sistema repetitivos.
2. **Persistencia:** Cualquier usuario o script que inicie una sesión de R dentro de este contenedor heredará automáticamente la capacidad de tomar capturas de pantalla.
3. **Compatibilidad:** Quarto detectará `webshot2` y el navegador Chrome de forma nativa al renderizar a PDF.

B.1.2 Notas de implementación

> **i** ¿Por qué Google Chrome y no Chromium?
>
> En distribuciones basadas en Ubuntu 22.04 o superiores, el comando `apt install chromium-browser` instala una versión ligada a **Snap**, la cual no puede ejecutarse dentro de un contenedor Docker por restricciones de seguridad del kernel. La instalación manual del paquete `.deb` de Google Chrome garantiza un binario funcional en `/usr/bin/google-chrome`.

> **!** Importancia de ImageIO y FileIO en Julia
>
> En Julia, estas librerías actúan como los "drivers" de imagen. Sin ellas, aunque el código genere un mapa o gráfico, Quarto no podrá convertirlo a un formato que LaTeX entienda (como PNG), resultando en bloques vacíos en el PDF final.

B.2 Función j_eval y j_plot en R

El motor de ejecución de los archivos Quarto (`.qmd`) dentro del contenedor instalado para el curso es `knitr`, el cual está basado en `R`. Esto significa que el código de `Python` y `Julia` se ejecuta a través de `R` para generar los archivos de salida HTML y PDF.

`Python` se ejecuta mediante el paquete de R `reticulate`, mientras que `Julia` suele hacerlo a través del paquete `JuliaCall`. Debido a que este último presenta problemas de compatibilidad con librerías geoespaciales en `R`, se desarrollaron con apoyo de inteligencia artificial las funciones `j_eval` y `j_plot`, las cuales utilizan el paquete `JuliaConnectoR` en su lugar.

Las funciones `j_eval` y `j_plot` son usadas para ejecutar código **Julia** en **R**. `j_eval` ejecuta comandos que no tengan salidas gráficas, y `j_plot` es necesaria cuándo el código Julia produce salidas gráficas.

```
# #| include: false
source("./docs/j_eval_j_plot.r")
```

B.2.1 Flujo de ejecución de texto con j_eval

Para entender por qué los resultados aparecen con colores y cómo se gestionan los bloques de código, observe el siguiente flujo de ejecución de la función `j_eval`:

fig-flujo-jeval

> **i** Nota
>
> Como se ilustra en la Figura B.1, la función no solo "pasa" el texto, sino que actúa como un supervisor que espera a que los bloques de programación estén completos antes de despertar al motor de Julia. Esto garantiza la estabilidad del sistema.

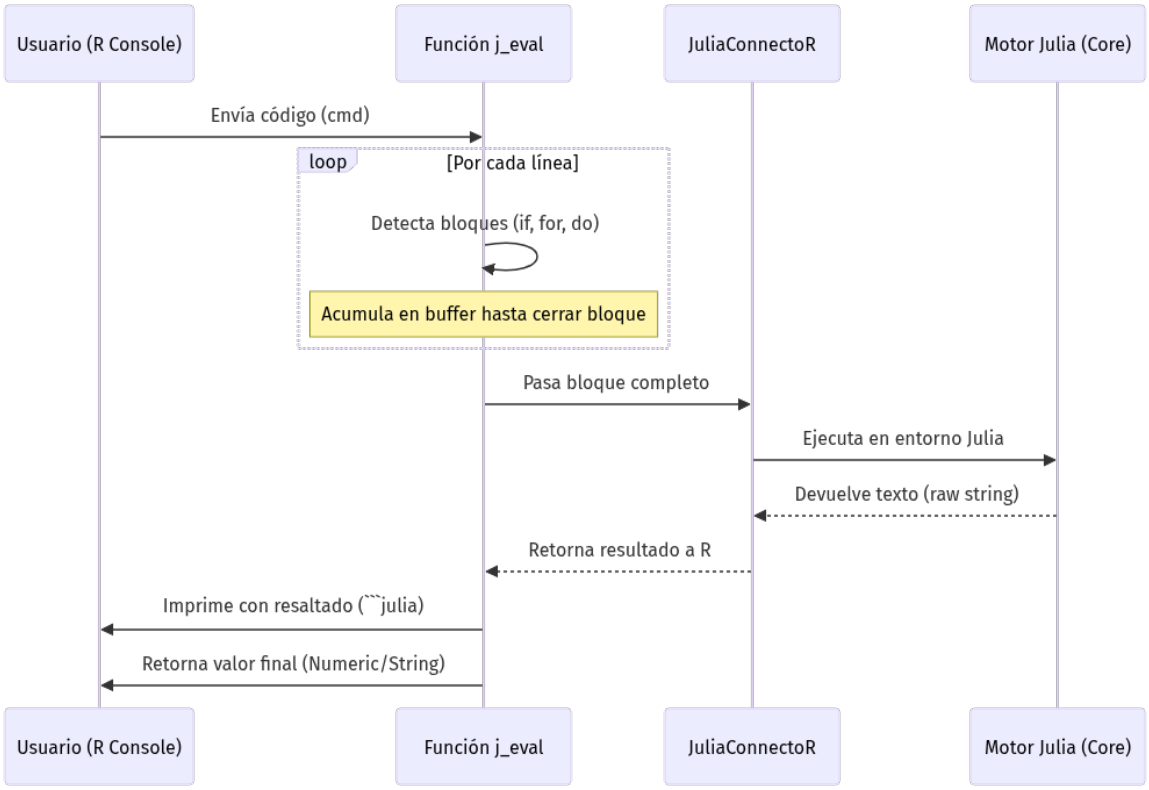


Figura B.1

B.2.2 Flujo de generación gráfica con j_plot

A diferencia de la ejecución de texto, `j_plot` requiere una coordinación adicional para gestionar archivos físicos. Observe el proceso:

> **El rol del disco duro**
>
> Como muestra la Figura B.2, el secreto de `j_plot` es que utiliza un archivo temporal como "puente" visual. Por eso, si usted tiene el archivo `tmp_plot.png` abierto en otro programa, j_plot podría fallar al intentar sobrescribirlo.

B.2.3 Interpretación de errores y consola

Al trabajar con la interoperabilidad entre lenguajes (usando las funciones `j_eval` y `j_plot` definidas arriba, la consola de **VSCode** nos devolverá mensajes que debemos aprender a interpretar.

B.2.3.1 El error de conexión TCP

Si al usar Julia recibe un mensaje de error que menciona `TCP connection`, usualmente significa que el motor de Julia se ha cerrado o ha entrado en conflicto de memoria. * **Solución**: Ejecute `.ensure_julia_ready()` en su consola de R para reiniciar el puente de comunicación.

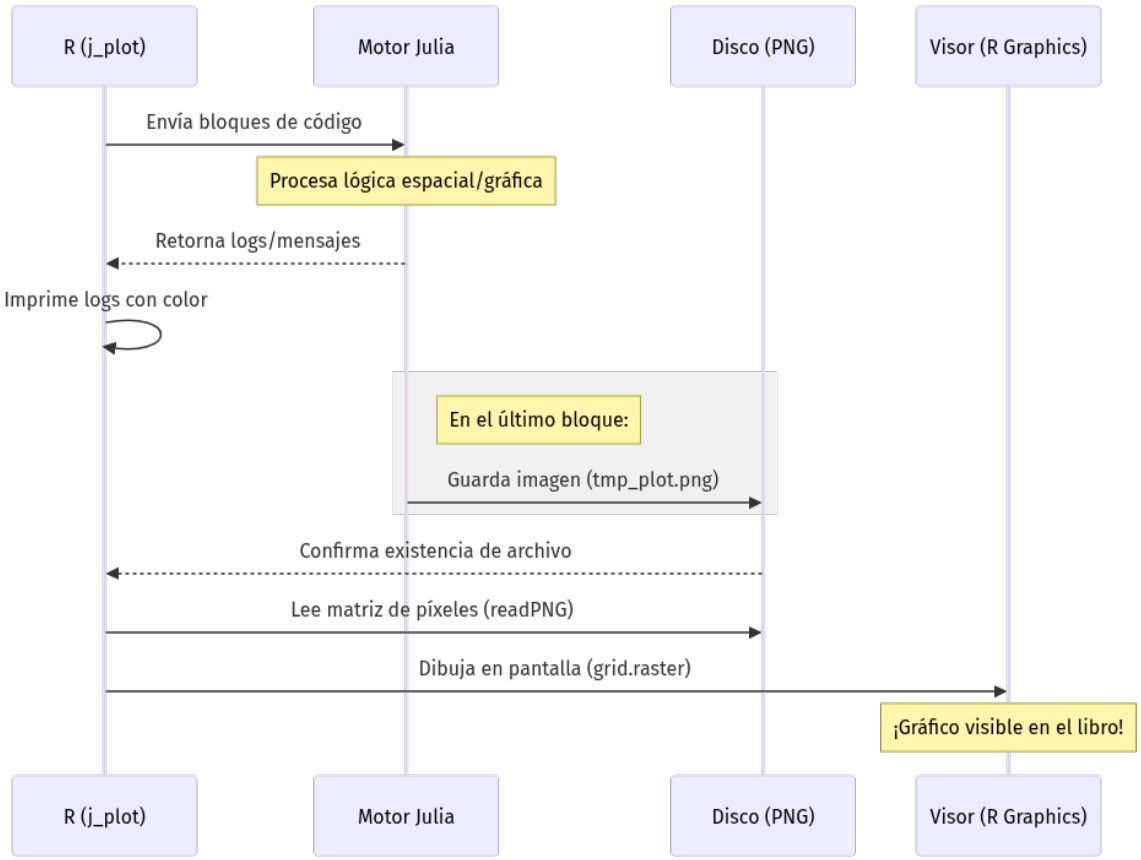

R (j_plot)
Motor Julia
Disco (PNG)
Visor (R Graphics)
Envía bloques de código
Procesa lógica espacial/gráfica
Retorna logs/mensajes
Imprime logs con color
En el último bloque:
Guarda imagen (tmp_plot.png)
Confirma existencia de archivo
Lee matriz de píxeles (readPNG)
Dibuja en pantalla (grid.raster)
¡Gráfico visible en el libro!
R (j_plot)
Motor Julia
Disco (PNG)
Visor (R Graphics)

Figura B.2

B.2.3.2 Interpretación del Stacktrace en Julia

Cuando Julia detecta un error, genera un rastro de llamadas o *Stacktrace*. No intente leer cada línea; busque siempre la primera (que define el tipo de error) y la última línea de código escrita por usted.

```
# Ejemplo de error por índice fuera de rango
ERROR: BoundsError: attempt to access 5-element Vector{Int64} at index [6]
```

B.2.3.3 Resaltado de sintaxis en la salida

Gracias a la configuración de `results: asis` en los bloques de código, la salida de nuestras funciones aparecerá con el prompt `julia>` coloreado, facilitando la distinción entre lo que es un mensaje informativo y un resultado del cálculo.

B.3 Introducción a la infraestructura de datos

Este anexo constituye la guía técnica para la gestión del entorno de desarrollo instalado. Es decir, información que describe y detalla los contenedores instalados (instalación opción A), su estructura y funcionamiento, así mismo como la descripción y detalles adicionales de las dos instalaciones de QGIS (Instalación Opción B)

B.3.1 Contenedores instalados (Instalación Opción A)

Siguiendo los pasos detallados en Capítulo 2 para instalar los contenedores Docker, en resumen, ellos contendrán:

1. Un contenedor `contenedor_sig_unal` correspondiente a la imagen `image_sig_unal:final` el cual contendrá las siguientes herramientas:

- **Python** (Apéndice I), **R** (Apéndice J) y **Julia** (Apéndice K) con los principales paquetes/librerías para geoprocesamiento.
- **Quarto** (Ver Apéndice C).
- **TinyTeX** (Ver Sección F.5).

2. Una contenedor `contenedor_postgis_unal` correspondiente a la imagen `postgis_unal:final` el cual contendrá las siguientes herramientas:

- **PostgreSQL** + **PostGIS** (Ver Apéndice N).

B.3.1.1 Batería de pruebas de integridad del docker

ID	Comando de Verificación (Docker)	Descripción
01	`docker exec contenedor_sig_unal gdalinfo --version`	Verifica que el núcleo de GDAL está activo.
02	`docker exec contenedor_sig_unal R -e "library(sf); st_point(c(0,0))"`	Prueba la librería `sf` y el motor de geometría en R.

ID	Comando de Verificación (Docker)	Descripción
03	`docker exec contenedor_sig_unal python3 -c "import geopandas; print(geopandas.__version__)"`	Verifica el stack espacial de Python.
04	`docker exec contenedor_sig_unal R -e "j_eval('sum([1, 2, 3])')"`	**Test Crítico:** Verifica la función personalizada y el puente R -> Julia.
05	`docker exec contenedor_sig_unal R -e "j_plot('plot(rand(10))')"`	Prueba la generación de gráficos Julia capturados por R.
06	`docker exec contenedor_sig_unal julia -e 'using ArchGDAL; println(ArchGDAL.GDAL.gdalversioninfo("RELEASE_NAME"))'`	**Cirugía Exitosa:** Confirma que Julia accede a GDAL del sistema (v3.12.1).
07	`docker exec contenedor_sig_unal R -e "library(RPostgres); dbConnect(Postgres(), host='db-postgis', dbname='sig_db_unal', user='profe_unal', password='geomatica2025')"`	Prueba la conexión R -> PostGIS (Interna).
08	`docker exec contenedor_sig_unal python3 -c "import psycopg2; conn = psycopg2.connect(host='db-postgis', dbname='sig_db_unal', user='profe_unal', password='geomatica2025'); print('Python PostGIS conectado ✔'); conn.close()"`	Prueba la conexión Python -> PostGIS (Interna).
09	`docker exec contenedor_sig_unal R -e "cat(whitebox::wbt_version())"`	Verifica que los binarios de WhiteboxTools están instalados y accesibles.
10	`docker exec contenedor_sig_unal R -e "library(httpgd); print('Visor OK')"`	Verifica que el motor gráfico para VSCode está listo en el puerto `8787`.

B.3.1.2 Configuración de puertos y conectividad

Para que su computadora (Host) pueda comunicarse con los servicios dentro del contenedor, hemos diseñado un sistema de "puentes" o mapeo de puertos. Esto evita conflictos si ya tiene instalados otros servidores de bases de datos o Jupyter en su PC.

Servicio	Puerto en su PC (Host)	Puerto en Contenedor	Propósito	Acceso / Conexión
Jupyter Lab	`8889`	`8888`	Programación y Notebooks	`http://localhost:8889`
Visor R (httpgd)	`8788`	`8787`	Gráficos de R y VSCode	`http://localhost:8788`
PostGIS (DB)	`5434`	`5432`	Base de Datos Espacial	`localhost:5434`

B.3.1.3 Credenciales de la base de datos

Utilice estos datos para configurar sus conexiones en QGIS, ArcGIS Pro o mediante código (R/Python/Julia):

- **Base de Datos:** `sig_db_unal`
- **Usuario:** `profe_unal`
- **Contraseña:** `geomatica2025`
- **Host:** `localhost`
- **Puerto:** `5434`
- **Host Interno:** `db-postgis` (Use este nombre únicamente para conexiones **dentro** de sus scripts).

B.3.1.4 Lógica de arquitectura (Dockerfile y docker-compose)

La configuración del entorno se ha "blindado" técnicamente para garantizar la estabilidad:

1. **En el Dockerfile**: Se usan las instrucciones `EXPOSE 8888` y `EXPOSE 8787`. Esto le avisa a Docker que el contenedor tiene dos "puertas" abiertas internamente. Al fijar `httpgd.port = 8787`, nos aseguramos de que el visor de gráficos de R no pelee con Jupyter por el mismo canal.
2. **En el Docker-Compose**:
   - `8889:8888`: Permite que este curso conviva con otras instalaciones de Jupyter (que suelen usar el 8888).
   - `8788:8787`: Habilita la conexión independiente de VSCode al visor de gráficos de R.
   - `5434:5432`: Evita el choque con bases de datos locales (Postgres suele usar el 5432 o 5433).

B.3.1.5 Guía de acceso: El concepto de "lados"

Es fundamental entender desde dónde está intentando conectar:

B.3.1.5.1 Desde afuera (Su PC / Host)

Es lo que usted configura en su navegador o en QGIS. Usted ve los puertos mapeados: * **Jupyter/Notebooks**: `http://localhost:8889` (usando el token `geomatica2025`). * **Base de Datos (QGIS/DBeaver)**: Host: `localhost`, Puerto: `5434`.

B.3.1.5.2 Desde adentro (El Contenedor)

Sus scripts de Python, R y Julia no saben que existe un mapeo externo. Dentro de su "casa" Docker, nada ha cambiado: * **PostGIS**: El código debe buscar el puerto estándar `5432` y el host `db-postgis`. * **httpgd**: El servidor de gráficos sigue escuchando en el puerto interno `8787`.

> **Tip de Conexión**
>
> Si intenta conectar QGIS usando el puerto `5432` y falla, recuerde que el "puente" hacia el contenedor se construyó específicamente sobre el puerto `5434`.

B.3.1.6 Resumen de la infraestructura instalada

Componente	Versión / Estado	Detalles Técnicos
R Engine	`4.3.3` (Angel Food Cake)	Puente `JuliaConnectoR` y visor `httpgd` configurados.
Julia Stack	`v1.10.x`	**ArchGDAL 3.12.1** operativo mediante enlaces simbólicos.
Python Stack	`3.12.x`	GeoPandas, PyTorch y drivers `psycopg2` listos.
Base de Datos	**PostGIS** (Noble)	Host interno `db-postgis` con extensión espacial activa.
Visualización	**Dual Mode**	Puertos **8788** (R/Julia Plots) y **8889** (Jupyter Lab).
Persistencia	**Volúmenes Docker**	Mapeo bidireccional en `/home/rstudio/work` confirmado.
Cirugía SSL	✔ Aplicada	Compatibilidad OpenSSL 3.0 (Sistema) vs 3.3 (Julia).

B.4 Localización de archivos y persistencia

En el contenedor, su carpeta del equipo local se encuentra vinculada a la ruta **/home/rstudio/work**. * Cualquier archivo guardado en esa ruta dentro de Jupyter aparecerá en su carpeta de Windows. * Se recomienda organizar su trabajo en las subcarpetas: `notebooks`, `scripts`, y `data`. * La carpeta `imagenes` (sin tílde y provista para las imágenes usadas en los archivos Quarto) debe residir también en esta ruta para un renderizado correcto.

> **!** Persistencia de Datos
>
> Si su contenedor se apaga o se reinicia, los datos de su base de datos PostGIS y los datos almancenados en `/home/rstudio/work` **no se pierden**. El volumen nombrado actúa como un disco duro externo que sobrevive a cualquier caída del sistema.

B.5 Cargar los contenedores dentro de VSCode

Para una experiencia de desarrollo profesional, conecte VSCode directamente al contenedor:

- Inicie Docker Desktop
- En el "PowerShell" de windows ubíquese en la carpeta donde instaló los contenedores (ver Sección D.5)
- Ejecute el comando: `docker compose up -d`
- Abra VSCode
- Instale (local) la extensión `Dev Containers`
- Acceda a la paleta de comandos con `Ctrl + Shift + P`, escriba (o seleccione) **Dev Containers: Attach to Running Container...**, y seleccione el contenedor `contenedor_sig_unal` (imagen `image_sig_unal`).
- Abra la carpeta `/home/rstudio/work` (esa carpeta corresponde a la carpeta local en dónde instaló los contenedores)
- Una vez conectado "dentro" del contenedor, debe habilitar/instalar las extensiones (*'Install in Container'*). Vea el listado completo en Sección F.6.

B.6 Inicialización del visor gráfico (solo una vez por sesión)

Para que los gráficos de R (y los puentes de Python/Julia) se visualicen correctamente en VSCode, debe inicializar el dispositivo gráfico. En su terminal de R, ejecute:

```
# Lanza el servidor de gráficos httpgd
httpgd::hgd()
```

- **Acceso al Visor**: VSCode debería abrir automáticamente una pestaña con el visor. Si esto no sucede o prefiere usar su navegador externo (Chrome/Edge), acceda a la dirección: `http://127.0.0.1:8788`.
- **Nota sobre Puertos**: Aunque el comando en R pueda imprimir una URL interna con el puerto `8787` o un token aleatorio, **ignore esa dirección**. Gracias a nuestro archivo `docker-compose.yml`, el puerto **8788** de su Windows está "cableado" permanentemente al visor, eliminando la necesidad de buscar tokens o puertos dinámicos. (xxx)

B.7 Ingreso a Jupyter Lab

Abra su navegador y acceda a: http://localhost:8889 * **Contraseña/Token:** (Si se solicita) `geomatica2025` * **Persistencia:** Todo archivo guardado en la carpeta `/home/rstudio/work` aparecerá automáticamente en su carpeta de Windows. **No guarde nada fuera de esa ruta**, o se perderá al cerrar el contenedor.

Puede pasar directamente la url completa:

http://127.0.0.1:8889/lab?token=geomatica2025

Si la url anterior no funciona, puede verificar cuál url usa actualmente el contenedor, la cual aparece al final del log después de ejecutar el siguiente comando:

```
docker logs contenedor_sig_unal
```

Alternativa: Puede iniciar/reiniciar el servidor de Jupyter Lab en la url http://127.0.0.1:8889/lab?token=geomatica2025. Use el siguiente comando ejecutado desde el terminal de VSCode:

```
jupyter lab --ip=0.0.0.0 --port=8889 --no-browser --allow-root --ServerApp.token='geomatica2025'
```

B.8 Guía visual de JupyterLab

Al ingresar, se encontrará con el centro de mando de sus kernels, donde podrá elegir entre R, Python o Julia para sus Notebooks:

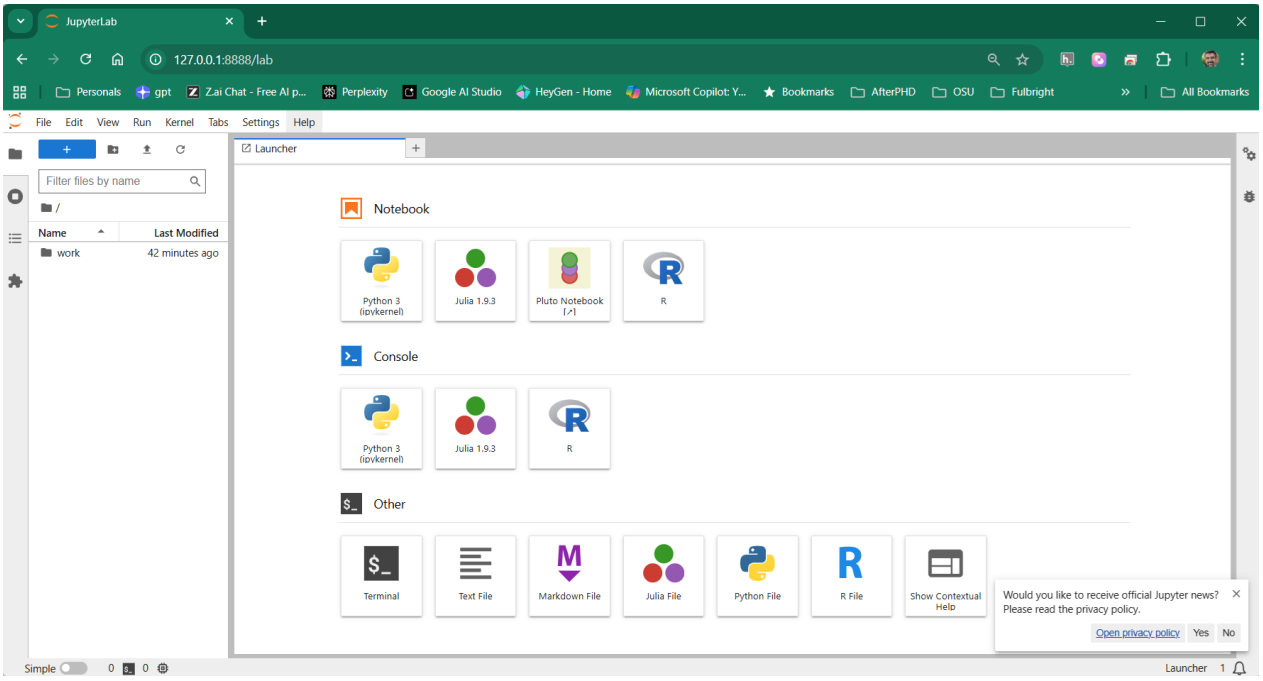


Figura B.3: Interfaz de JupyterLab configurada para el laboratorio.

B.8.1 La Carpeta 'work' y el Espejo de Datos

En el panel izquierdo de la Figura B.3, la carpeta **`/home/rstudio/work/`** es el espejo de su directorio local en Windows. Gracias a la configuración de volúmenes en el archivo `docker-compose.yml`, existe un puente directo: todo cambio realizado en Jupyter se refleja en su disco duro y viceversa, garantizando que su trabajo no se pierda al apagar el contenedor.

B.9 Compilación de la guía completa o documentos individuales

La guía del curso está organizada a partir de un **archivo orquestador** `_quarto.yml`, el cual contiene una referencia explícita a cada uno de los archivos Quarto (`.qmd`) que definen los capítulos, las presentaciones, los talleres y los anexos. Este archivo centraliza la organización y el formato de la guía tanto en **PDF** como en **HTML**.

B.9.1 Compilación del proyecto completo

Si desea compilar la guía completa del curso y generar todos los formatos, simplemente ejecute el comando `quarto render` en la carpeta raíz (donde se encuentra el archivo `_quarto.yml`):

```
# Desde la carpeta raíz donde reside el archivo '_quarto.yml'
quarto render
```

B.9.2 Compilación de archivos individuales

En ocasiones, querrá trabajar en un solo capítulo sin procesar el libro entero. Para compilar un archivo específico (ej. `archivo.qmd`) de manera independiente y evitar que Quarto aplique las reglas de numeración y referencias del proyecto global, siga estos pasos:

1. **Renombrar temporalmente el orquestador:** Cambie el nombre de `_quarto.yml` a `_quarto.yml.back`. Esto hace que Quarto trate al archivo como un documento "solitario".
2. **Ejecutar el renderizado específico:**

- **Generar todos los formatos (HTML y PDF):**

```
quarto render archivo.qmd --to all
```

- **Generar solo formato HTML:**

```
quarto render archivo.qmd --to html
```

- **Generar solo formato PDF:**

```
quarto render archivo.qmd --to pdf
```

B.9.3 Notas de estudio y personalización

¡Personaliza tu aprendizaje!

En Quarto, puedes agregar tus propias **notas de estudio** usando la sintaxis de "callouts". Recuerda que este documento está en proceso de **construcción** y sufrirá cambios permanentes hasta terminar el curso.
Puedes buscar **ayuda en Internet** para personalizar la apariencia de tus notas. La sintaxis básica es:

```
::: {.callout-tip icon="true"}
### ¡Escribe tu nota aquí!
Este es un espacio para tus observaciones personales y recordatorios.
:::
```

B.10 Mapeo de capacidades SIG

Es vital entender que, aunque usemos lenguajes distintos, todos "beben" de las mismas librerías de bajo nivel instaladas en nuestra imagen base de OSGeo:

Operación	R (`sf` / `terra`)	Python (`GeoPandas`)	Julia (`ArchGDAL`)	Motor de Sistema
Lectura de Datos	`st_read()` / `rast()`	`read_file()` / `open()`	`ArchGDAL.read()`	**GDAL**
Buffers / Geometría	`st_buffer()`	`.buffer()`	`LibGEOS.buffer()`	**GEOS**
Reproyección	`st_transform()`	`.to_crs()`	`ArchGDAL.reproject()`	**PROJ**

B.11 Verificación de conectividad multilenguaje

Nota técnica: Dentro de la red de Docker, el host es `db-postgis`.

B.11.1 Python

```
import psycopg2
import geopandas as gpd
import fiona
import matplotlib.pyplot as plt
from shapely.geometry import Point

print("--- Inicio de Verificación de Python SIG ---")
```

```
--- Inicio de Verificación de Python SIG ---
```

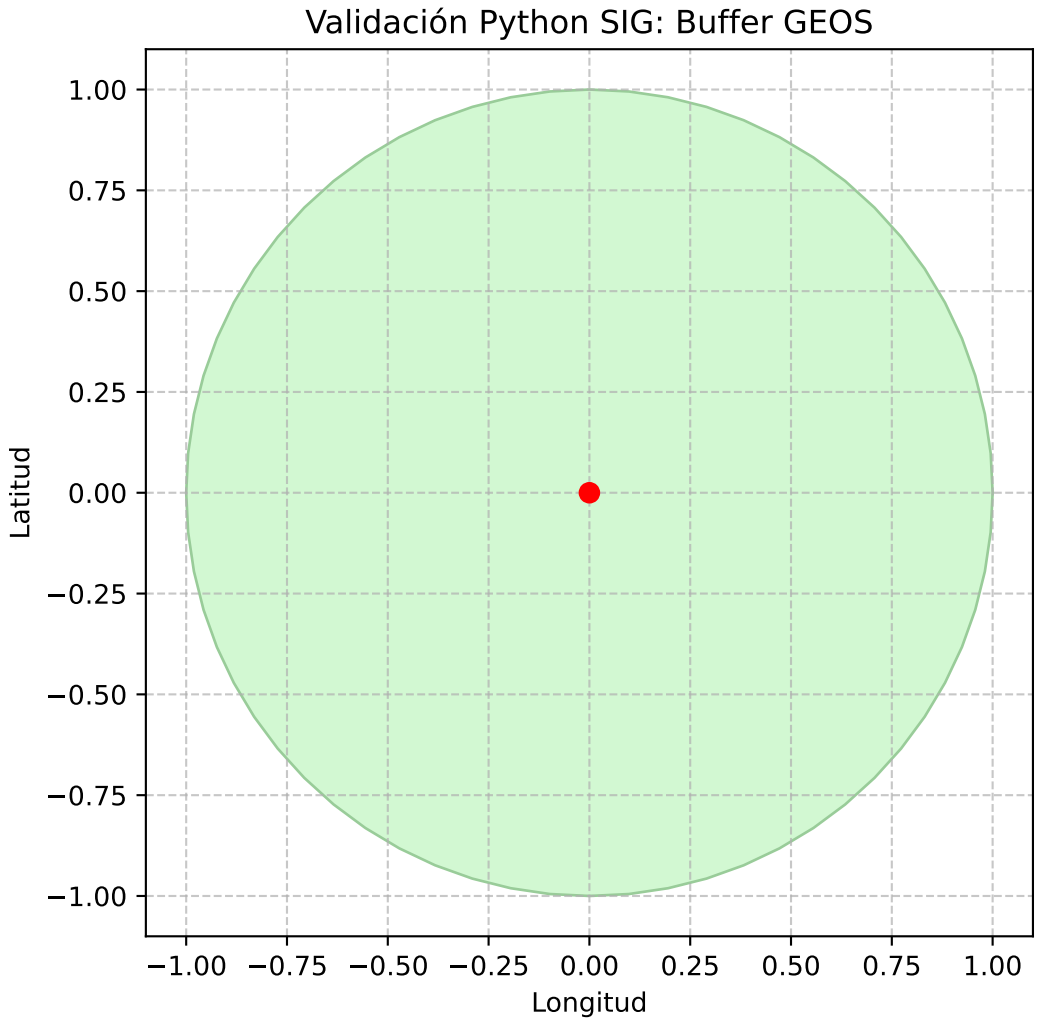
```
# 1. Prueba de conexión a la base de datos PostGIS
try:
    conn = psycopg2.connect(
        host="db-postgis",
        dbname="sig_db_unal",
        user="profe_unal",
        password="geomatica2025"
    )
    print("✅ Conexión a PostGIS: Exitosa")
    conn.close()
except Exception as e:
    print(f"❌ Error de conexión a PostGIS: {e}")
```

```
✅ Conexión a PostGIS: Exitosa
```

```
# 2. Prueba de Fiona y drivers GDAL
try:
    drivers = len(fiona.supported_drivers)
    print(f"✔ Fiona operativo: {drivers} drivers GDAL detectados")
except Exception as e:
    print(f"✖ Error en Fiona/GDAL: {e}")
```

```
✔ Fiona operativo: 17 drivers GDAL detectados
```

```
# 3. Prueba de GeoPandas, Motores GEOS y Visualización
try:
    # Creamos un punto y su buffer (GEOS)
    punto = Point(0, 0)
    buffer_geom = punto.buffer(1.0)

    # Creamos GeoDataFrames para graficar
    gdf_buffer = gpd.GeoDataFrame({'geometry': [buffer_geom]}, crs="EPSG:4326")
    gdf_punto = gpd.GeoDataFrame({'geometry': [punto]}, crs="EPSG:4326")

    print(f"✔ GeoPandas {gpd.__version__}: Operativo")
    print(f"✔ Motores GEOS/Shapely: Verificados")

    # Generación del Plot Espacial con Ejes y Cuadrícula
    fig, ax = plt.subplots(figsize=(6, 6))

    # Graficamos el buffer
    gdf_buffer.plot(ax=ax, color='lightgreen', edgecolor='green', alpha=0.4, label='Buffer')

    # Graficamos el punto original (en rojo)
    gdf_punto.plot(ax=ax, color='red', markersize=50, zorder=5, label='Centro')

    # Configuración de estilo consistente (Ejes y Grilla)
    ax.set_title("Validación Python SIG: Buffer GEOS")
    ax.set_xlabel("Longitud")
    ax.set_ylabel("Latitud")
    ax.grid(True, linestyle='--', alpha=0.7) # Cuadrícula activada
    ax.set_aspect('equal') # Proporción 1:1 para evitar deformación

    plt.show()
    print("✔ Visualización GeoPandas: Mapa generado con éxito")

except Exception as e:
    print(f"✖ Error en el stack espacial de Python: {e}")
```




```
print("--- Verificación Finalizada ---")
```

```
--- Verificación Finalizada ---
```

B.11.2 R

```
library(DBI)
library(RPostgres)
library(sf)
```

```
Linking to GEOS 3.12.1, GDAL 3.8.4, PROJ 9.4.0; sf_use_s2() is TRUE
```

```
library(terra)
```

```
terra 1.8.93

Attaching package: 'terra'

The following object is masked from 'package:grid':

    depth
```

```
cat("--- Inicio de Verificación de R-Spatial ---\n")
```

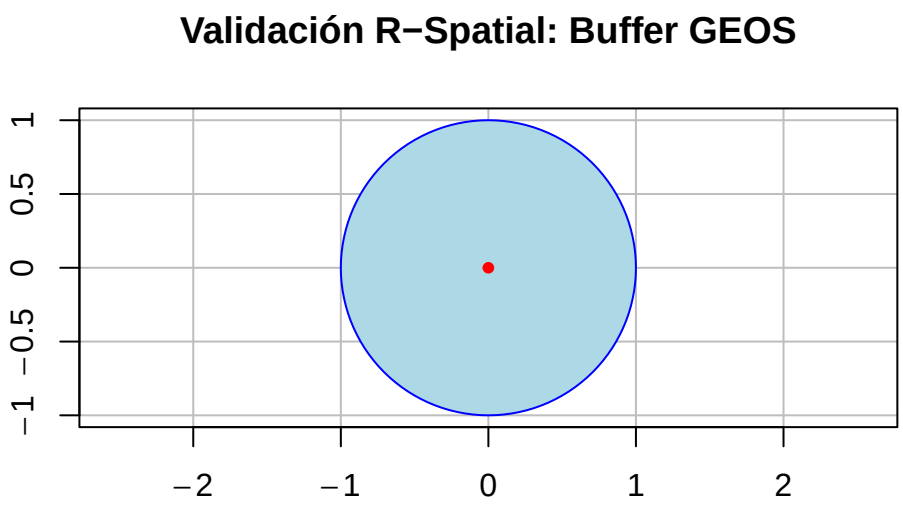
```
--- Inicio de Verificación de R-Spatial ---
```

```
# 1. Prueba de conexión a la base de datos PostGIS
tryCatch({
  con <- dbConnect(
    RPostgres::Postgres(),
    host = "db-postgis",
    dbname = "sig_db_unal",
    user = "profe_unal",
    password = "geomatica2025"
  )
  cat("✅ Conexión a PostGIS: Exitosa\n")
  dbDisconnect(con)
}, error = function(e) {
  cat("❌ Error de conexión a PostGIS:", conditionMessage(e), "\n")
})
```

```
✅ Conexión a PostGIS: Exitosa
```

```
# 2. Prueba de Motores de Sistema y Visualización (sf)
tryCatch({
  conf <- sf_extSoftVersion()
  cat(paste0("✅ sf operativo. Motores detectados:\n",
             "   - GDAL: ", conf["GDAL"], "\n",
             "   - GEOS: ", conf["GEOS"], "\n",
             "   - PROJ: ", conf["PROJ"], "\n"))

  # Creamos el punto y el buffer
  punto <- st_point(c(0, 0))
  buffer_geom <- st_buffer(punto, dist = 1)
  cat("✅ Prueba geométrica (GEOS): Buffer creado correctamente\n")

  # Generación del Plot Espacial
  # Usamos st_geometry para graficar solo la forma
  plot(st_geometry(buffer_geom),
       col = 'lightblue',
       border = 'blue',
       main = "Validación R-Spatial: Buffer GEOS",
       axes = TRUE,
       graticule = TRUE)

  # Añadimos el punto original para referencia
  plot(st_geometry(punto), add = TRUE, col = 'red', pch = 20)

  cat("✅ Visualización sf: Mapa generado con éxito\n")

}, error = function(e) {
  cat("❌ Error en el stack sf/GEOS:", conditionMessage(e), "\n")
})
```

```
✅ sf operativo. Motores detectados:
   - GDAL: 3.8.4
   - GEOS: 3.12.1
   - PROJ: 9.4.0
✅ Prueba geométrica (GEOS): Buffer creado correctamente
```



✔ Visualización sf: Mapa generado con éxito

```
# 3. Prueba de Raster (terra)
tryCatch({
  r <- rast(ncols=10, nrows=10)
  values(r) <- 1:ncell(r)
  cat("✔ Paquete 'terra' operativo: Objetos Raster verificados\n")
}, error = function(e) {
  cat("✖ Error en el stack terra:", conditionMessage(e), "\n")
})
```

✔ Paquete 'terra' operativo: Objetos Raster verificados

```
cat("--- Verificación Finalizada ---\n")
```

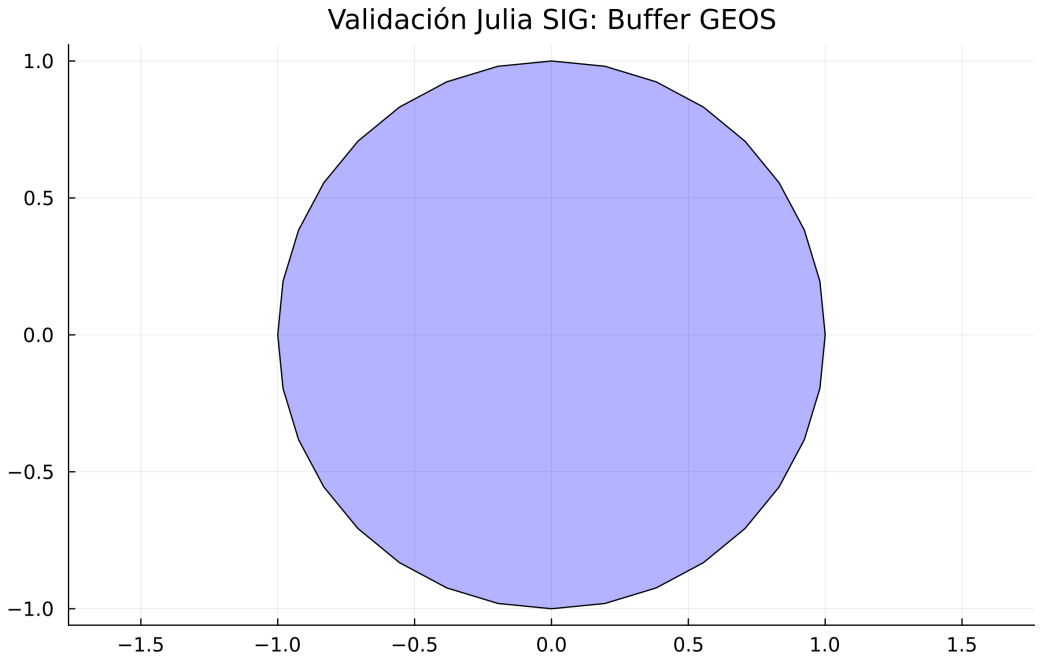
--- Verificación Finalizada ---

B.11.3 Julia

```
# #| eval: false
j_plot('
using LibPQ
using LibGEOS
using ArchGDAL
using Plots

println("--- Inicio de Verificación de Julia SIG ---")

# 1. Prueba de conexión a la base de datos PostGIS
try
    conn = LibPQ.Connection("host=db-postgis dbname=sig_db_unal user=profe_unal password=geomatica2025")
    println("✔ Conexión a PostGIS: Exitosa")
    close(conn)
catch e
    println("✖ Error de conexión a PostGIS: ", e)
end

# 2. Verificación de LibGEOS y Visualización de Geometría
```

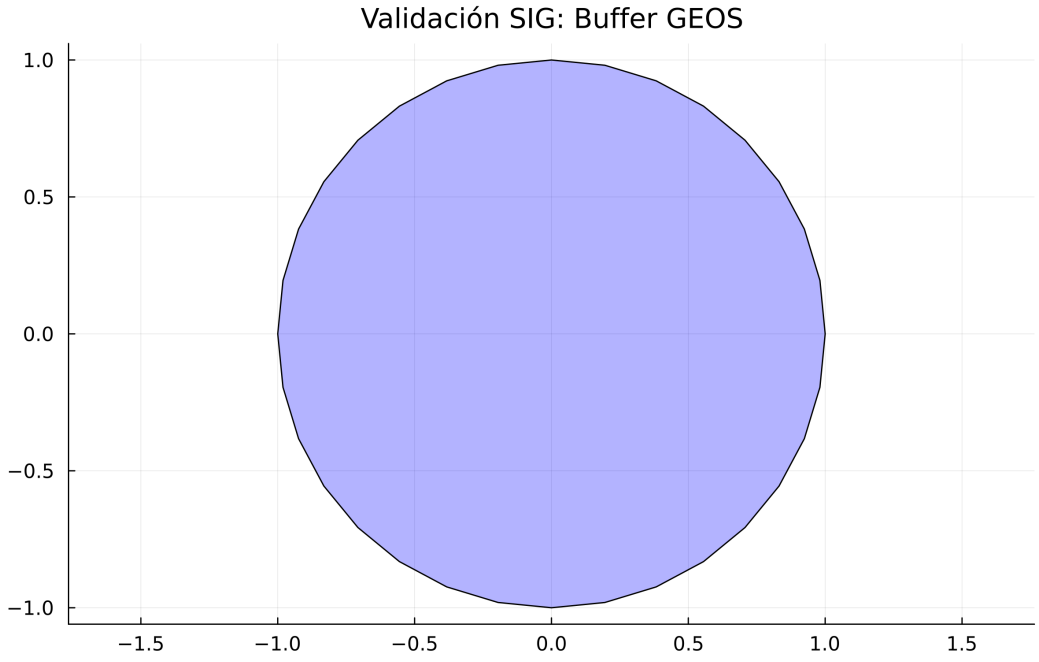
```
try
    # Creamos un punto y le aplicamos un buffer de 1.0 unidades
    # Esto valida la integración de Julia con la librería GEOS del sistema
    punto = LibGEOS.readgeom("POINT (0 0)")
    buffer_geom = LibGEOS.buffer(punto, 1.0)

    println("✅ LibGEOS operativo: Motores geométricos verificados")

    # Graficamos el objeto del buffer
    # fillcolor y alpha ayudan a ver que es un polígono real
    plt = plot(buffer_geom,
               title="Validación Julia SIG: Buffer GEOS",
               fillcolor=:blue,
               fillalpha=0.3,
               aspect_ratio=:equal,
               legend=false)

    # display() es OBLIGATORIO para mostrar gráficos dentro de bloques try/catch
    display(plt)

catch e
    println("❌ Error en LibGEOS o Visualización: ", e)
end

# 3. Verificación de ArchGDAL (Usando llamada de bajo nivel)
try
    # Accedemos directamente al motor de C para evitar errores de exportación
    gdal_ver = ArchGDAL.GDAL.gdalversioninfo("--version")
    println("✅ ArchGDAL operativo (Versión GDAL: $gdal_ver)")
catch e
    println("❌ Error en ArchGDAL: ", e)
end

println("--- Verificación Finalizada ---")
')
```

```
Starting Julia ...
```




```
# Validación de Motores SIG y Conectividad
j_plot('
using LibPQ, LibGEOS, ArchGDAL, Plots

# 1. Prueba de conexión a PostGIS
try
    conn = LibPQ.Connection("host=db-postgis dbname=sig_db_unal user=profe_unal password=geomatica2025")
    println("✅ Conexión a PostGIS: Exitosa")
    close(conn)
```

```
catch e
    println("✗ Error de conexión: ", e)
end

# 2. Prueba de Motores Geométricos (GEOS)
try
    punto = LibGEOS.readgeom("POINT (0 0)")
    buffer_geom = LibGEOS.buffer(punto, 1.0)
    println("✓ Motores geométricos operativos (GEOS)")

    # Visualización del Buffer
    plt = plot(buffer_geom, title="Validación SIG: Buffer GEOS",
               fillcolor=:blue, fillalpha=0.3, aspect_ratio=:equal)
    display(plt)
catch e
    println("✗ Error en motores geométricos o gráficos: ", e)
end
')
```



Apéndice C

Quarto: Orquestación y Configuración

Quarto se basa en un archivo de configuración centralizado donde se definen los parámetros de renderizado, bibliografía y estilos.

C.1 Opciones del encabezado `yml`

Ejemplo 1: Solo HTML

```
---
title: "Título de la Práctica"
author: "Nombre del Estudiante"
date: last-modified
format:
  html:
    toc: true
    toc-depth: 3
    toc-location: left
    number-sections: true
    theme: lux
---
```

Ejemplo 2: HTML y Python

```
---
title: "Informe Técnico: Análisis Geoespacial"
author: "ID Correo UNAL"
date: today
format:
  html:
    toc: true
    number-sections: true
    theme: cosmo
  pdf:
    toc: true
    toc-title: "Contenido"
    number-sections: true
    documentclass: scrreprt
    geometry:
      - top=25mm
      - left=25mm
      - right=25mm
      - bottom=25mm
---
```

C.2 Resumen de comandos Quarto

Renderizar todos los formatos definidos en el encabezado `yml`:

```
quarto render archivo.qmd --to all
```

Renderizar a HTML:

```
quarto render archivo.qmd --to html
```

Renderizar a PDF:

```
quarto render archivo.qmd --to pdf
```

Renderizar a PDF, mostrando todos los detalles en la consola:

```
quarto render archivo.qmd --to pdf --trace --verbose`
```

Chequear estado de instalación de Quarto

```
quarto check
```

C.3 Control de ejecución en Quarto - opciones de `chunks`

Para que sus informes técnicos sean profesionales, no basta con que el código funcione; es necesario controlar qué se muestra y qué se oculta al lector final. En Quarto, esto se logra mediante el uso de "opciones de chunk" utilizando la sintaxis de la tubería de comentarios (`#|`).

Opción de Chunk	¿Se ejecuta?	¿Muestra el Código?	¿Muestra Resultados / Plots?	Uso Ideal
`#| echo: false`	✔ SÍ	✖ NO	✔ SÍ	**Para Resultados Finales.** Muestra el mapa/tabla sin la "receta".
`#| include: false`	✔ SÍ	✖ NO	✖ NO	**Para el Setup.** Corre todo en secreto, sin mensajes ni advertencias.

Opción de Chunk	¿Se ejecuta?	¿Muestra el Código?	¿Muestra Resultados / Plots?	Uso Ideal
`#| code-fold: true`	✔ SÍ	Plegado	✔ SÍ	**Para el Proceso.** El código se oculta tras un botón "Code".
`#| eval: false`	✖ NO	✔ SÍ	✖ NO	**Para Tutoriales.** Solo muestra el texto del código sin procesarlo.
`#| output: false`	✔ SÍ	✔ SÍ	✖ NO	**Para Debugging.** Ves su código pero no los resultados pesados.
`#| warning: false`	✔ SÍ	-	-	Oculta esos textos naranjas de advertencia de las librerías.
`#| message: false`	✔ SÍ	-	-	Oculta mensajes de carga (ej: "Loading terra package…").

C.3.1 Recomendaciones de uso según el contexto

Dependiendo de la parte del script en la que se encuentre, existen configuraciones que garantizan un documento más limpio y fluido:

Tipo de Chunk	Opción recomendada	¿Por qué?
Setup / Librerías	`#| include: false`	Ejecuta todo pero oculta el código y los mensajes de carga de R/Julia/Python.
Limpieza de RAM	`#| include: false`	Borra objetos y vacía el cache en silencio. El lector no necesita ver la "limpieza".
Procesamiento	`#| code-fold: true`	Muestra el mapa pero esconde el código tras un clic. Ideal para transparencia académica.

Tipo de Chunk	Opción recomendada	¿Por qué?
Resultados Finales	`#| echo: false`	Muestra solo la tabla comparativa y conclusiones. Es el "veredicto" limpio y profesional.

Sintaxis Correcta

Recuerde que las opciones deben ir al inicio del chunk, justo después de los corchetes del lenguaje:

```
#| echo: false
#| warning: false
library(terra)
# Su código aquí...
```

C.4 Instalación de Extensiones de Quarto

Las extensiones permiten ampliar las funcionalidades nativas de Quarto, añadiendo nuevos formatos de salida, filtros personalizados o librerías de componentes visuales. Para que el proceso de descarga desde repositorios remotos sea exitoso y seguro, es fundamental haber completado previamente la configuración descrita en la Sección E.7.

C.4.1 El proceso de instalación

Para agregar una extensión a tu proyecto, Quarto utiliza el comando `add` seguido del identificador del repositorio (usualmente `usuario/repositorio`). Este proceso descarga los archivos necesarios dentro de una carpeta local denominada `_extensions/`.

C.4.1.1 Ejemplo práctico: Librería de iconos FontAwesome

Como caso de estudio, instalaremos la extensión oficial para el uso de iconos vectoriales. Ejecuta el siguiente comando en tu terminal:

```
quarto add quarto-ext/fontawesome
```

Confianza del Autor

Al ejecutar el comando, el sistema solicitará confirmar si confías en los autores de la extensión. Escribe `Y` para permitir que Quarto gestione los archivos del filtro en tu directorio de trabajo.

C.4.2 Gestión de archivos del proyecto

Una vez finalizada la instalación, notarás que en la raíz de tu proyecto se ha creado la siguiente estructura:

```
mi-proyecto/
  _extensions/
     quarto-ext/
         fontawesome/
  tu-archivo.qmd
```

Es importante **no modificar manualmente** los archivos dentro de la carpeta `_extensions`, ya que Quarto los requiere para procesar el documento final durante el renderizado.

C.4.3 Aplicación en el documento

Tras la instalación exitosa, puedes utilizar las nuevas funcionalidades mediante *shortcodes*. En el caso de nuestra extensión de ejemplo, ahora es posible insertar iconos de la siguiente manera:

```
Para este análisis utilizaremos:

* {{< fa satellite >}} Datos satelitales.
* {{< fa location-dot "red" >}} Puntos de control terrestre.
```

Actualización de extensiones

Si en el futuro deseas actualizar una extensión instalada a su última versión, simplemente vuelve a ejecutar el mismo comando `quarto add`. El sistema detectará la versión existente y te preguntará si deseas sobrescribirla.

Apéndice D

Docker: Gestión de Contenedores e Imágenes

Docker es un motor de virtualización (Inc., 2025) en el que instalaremos las dos máquinas virtuales Linux que permitirán la ejecución de código Python, R y Julia y el acceso a PostGIS + PostgreSQL.

D.1 Instalación del software

Descargue e instale la versión oficial de **Docker Desktop** desde el siguiente enlace:

- **URL**: https://www.docker.com/products/docker-desktop/
- Descargue la versión más reciente para el sistema operativo de su máquina local y verifique que el instalador coincida con la arquitectura de su máquina.
- **Instalación**: Ejecute el instalador y asegúrese de aceptar la actualización del kernel de WSL 2 si el sistema lo solicita, o previamente usar en el terminal `wsl --update`. Durante la instalación, asegúrese de activar la opción **"Use WSL 2 instead of Hyper-V"**.

D.2 Preparación Docker Desktop

Antes de iniciar la carga o instalación de imágenes/contenedores, es imperativo configurar el entorno local:

- **Configuración de Almacenamiento**: Debido a que la imagen políglota y la base de datos requieren un espacio considerable, asegúrese de que el disco donde Docker guarda sus archivos `.vhdx` tenga al menos **50 GB de espacio libre**. Tenga en cuenta que el tamaño en disco de las dos imágenes Docker, una vez instaladas, es de aproximadamente **45 GB**.
- **Procedimiento para cambiar la ubicación de las imágenes** (archivos con extensión `.vhdx`): Si su disco principal (`C:`) está sin espacio, mueva los archivos Docker a otro disco. En Docker Desktop:
  a. Diríjase a **Settings** (engranaje) -> **Resources** -> **Advanced**.
  b. Localice **Disk image location**.
  c. Haga clic en `Browse` y seleccione una carpeta en un disco con mayor capacidad.

d. Haga clic en **Apply & restart**.

- **Optimización de Memoria Swap**: Para procesar datos raster de gran tamaño sin interrupciones en R o Julia, es fundamental ampliar la memoria Swap (espacio de intercambio). Puede consultar más detalles en la Sección D.3 sobre cómo esto impacta el rendimiento.
- **Procedimiento para aumentar Swap en Windows (Host)**: Ajuste la memoria virtual del sistema operativo para evitar cierres por falta de RAM:
  a. En el buscador de Windows, escriba y seleccione **"Ver la configuración avanzada del sistema"**.
  b. En la pestaña **Opciones avanzadas**, sección **Rendimiento**, haga clic en el botón `Configuración`.
  c. Diríjase a **Opciones avanzadas** -> **Memoria Virtual** y haga clic en `Cambiar`.
  d. Desmarque "Administrar automáticamente", seleccione el disco principal, elija **Tamaño personalizado** y asigne estos valores sugeridos, que dependen del espacio disponible en el disco (verifíquelo): Inicial **16384 MB** / Máximo **32768 MB**.
  e. Haga clic en `Establecer` y luego en **Aceptar** (requerirá reiniciar el equipo).
- **Configuración de Swap en el Contenedor (WSL2)**: Dado que Docker opera sobre el subsistema Linux, debe configurar el archivo de intercambio global de WSL2:
  a. Abra el explorador de archivos y diríjase a su carpeta de usuario (escriba `%USERPROFILE%` en la barra de direcciones).
  b. Cree un archivo nuevo llamado `.wslconfig` (asegúrese de que no tenga extensión `.txt` al final).
  c. Pegue el siguiente contenido para definir la RAM y asegurar **32 GB de swap** para sus procesos espaciales (sugerido, depende de su espacio en disco):

```
[wsl2]
memory=12GB # RAM máxima física asignada a Linux
swap=32GB   # Memoria de intercambio para evitar el cierre de contenedores
```

> **Importante**
>
> Si desea verificar si estos cambios surtieron efecto dentro de su laboratorio, puede ejecutar el comando `free -h` en la terminal de Jupyter o VSCode.

D.3 Limpieza del entorno

D.3.1 Eliminar todas las imagenes/contenedores y liberar todo el espacio usado por Docker.

Advertencia: Los siguientes comandos eliminarán todos los contenedores, imágenes y volúmenes existentes en su sistema. Úselos solo si no tiene contenedores o imágenes que desee preservar.

Ejecute estos comandos en su terminal (PowerShell) para liberar espacio y evitar conflictos:

```
# Detener y eliminar todos los contenedores
docker stop $(docker ps -aq)
docker rm -f $(docker ps -aq)

# Eliminar todas las imágenes
```

```
docker rmi -f $(docker images -aq)

# Eliminar todos los volúmenes (libera MUCHO espacio)
docker volume rm $(docker volume ls -q)

# Eliminar redes de usuario (no borra bridge/host/none)
docker network rm $(docker network ls -q | Select-String -NotMatch "bridge|host|none")

# Eliminar caché de construcción
docker builder prune -a -f

# OPCIÓN MÁS SIMPLE (Todo en un solo comando)
# docker system prune -a --volumes -f
```

Para verificar el espacio liberado, use: `docker system df`. El resultado debería mostrar valores cercanos a **0 bytes**.

D.3.2 Comandos de rescate de espacio

El entorno políglota de este curso es robusto y, por lo tanto, pesado. Tras realizar actualizaciones o varias pruebas de construcción, es posible que el espacio en disco se agote rápidamente.

Si recibe errores de "Disk Full" o desea limpiar su sistema, ejecute los siguientes comandos en su terminal de Windows (PowerShell):

```
# 1. Eliminar contenedores detenidos y redes en desuso
docker system prune -f

# 2. Limpiar caché de construcción (libera mucho espacio tras errores de build)
docker builder prune -f

# 3. (Uso extremo) Eliminar TODAS las imágenes que no estén siendo usadas
# docker image prune -a -f
```

Seguridad de sus Datos

No tema realizar limpiezas periódicas. Gracias a la configuración de volúmenes en nuestro archivo `docker-compose.yml`, todo su código, scripts y datos espaciales están **físicamente en su disco local** (en la carpeta de su ID UNAL).
Al apagar o borrar el contenedor, lo que está dentro de `/home/rstudio/work/` **siempre estará a salvo** en su carpeta de Windows. El contenedor es solo el "motor", sus archivos son el "combustible" que usted posee.

D.4 Descarga y preparación de archivos

Siga estos pasos para obtener la imagen base:

1. **Descarga**: Obtenga el archivo de 15.5 GB desde el siguiente enlace: Google Drive - sig_unal_completo.zip.
2. **Descompresión Parte A**: Extraiga el contenido de `sig_unal_completo.zip`.
3. **Renombrado**: Al archivo resultante llamado `sig_unal_completo.txt`, cámbiele manualmente la extensión a `.zip`.
4. **Descompresión Parte B**: Extraiga este nuevo archivo `.zip` para obtener los dos archivos definitivos:
   - `sig_unal_completo.tar` (El archivo de la imagen).

- `docker-compose.yml` (El **NUEVO** archivo de orquestación contenido en la descarga).

D.5 Cargar imagen/contenedor a partir de imagen/contenedor base guardada en `sig_unal_completo.tar` y `docker-compose.html`

Este procedimiento instalará (mediante carga, es decir sin compilación) dos imágenes (una con la base de datos espacial y otra con Python, R y Julia) y configurará una red interna para comunicar su máquina anfitriona (Windows) con las dos máquinas virtuales Linux.

Desde una terminal situada en la carpeta que contiene los archivos `sig_unal_completo.tar` y `docker-compose.yml`, ejecute:

```
docker load -i sig_unal_completo.tar
```

Este proceso puede tardar varios minutos o inclusive horas dependiendo de su procesador y disco duro. Verifique que al finalizar aparezca el mensaje "Loaded image" para ***`image_sig_unal:final`*** *y* ***`postgis_unal:final`****.*

> Nota: Existen dos versiones **diferentes** del archivo `docker-compose.html`. El primero es el que se describe en Sección D.6 y el segundo, que aplica para este procedimiento, es el que se descarga en Sección D.4.

D.5.1 Archivo `docker-compose.yml` (orquestación)

Este archivo es el **manifiesto técnico** que automatiza la construcción y coordinación de los servicios del curso. Su propósito es definir las reglas de convivencia entre los contenedores, configurando los siguientes pilares:

- **Imágenes**: Versiones exactas de software (GDAL/Ubuntu para análisis y PostGIS para datos).
- **Puertos**: Mapeos específicos para acceder a las herramientas desde el equipo local sin conflictos (8889 para Jupyter, 8788 para el visor de R y 5434 para la base de datos).
- **Volúmenes**: Garantizan la persistencia de datos, sincronizando su carpeta local en tiempo real con el entorno interno del contenedor.

A continuación, se describen los dos servicios integrados en este manifiesto:

- **Servicio de Análisis (`analisis-geo`)**: Identificado en el archivo como `analisis-geo`, este servicio no usa `Dockerfile` (la etiqueta `build: .` está no está habilitada). Es el motor políglota encargado de procesar R, Python y Julia sobre una base robusta de GDAL. Incluye una configuración de `LD_PRELOAD` para la estabilidad de las librerías dinámicas y una integración profunda que permite usar el visor `httpgd` de R como terminal gráfica unificada para todos los lenguajes. Crea el contenedor `contenedor_sig_unal`. La carpeta `/home/rstudio/work` del contenedor, mapea la carpeta del equipo local donde instaló las imágenes y contenedores a partir de `sig_unal_completo.tar` y `docker-compose.yml`.
- **Servicio PostGIS (`db-postgis`)**: Identificado como `db-postgis`, este servicio apunta a una imagen que fue creada a partir de la imagen especializada de Kartoza. Levanta el servidor de base de datos `sig_db_unal`, configurado para recibir conexiones espaciales desde sus scripts o herramientas externas como QGIS o ArcGIS a través del puerto `5434`.

```
services:
  analisis-geo:
    # build: .                          # <--- No utiliza Dockerfile
    image: image_sig_unal:final         # <--- Agrega esto para nombrar la imagen
    container_name: contenedor_sig_unal # <--- Cambia esto para el contenedor
    tty: true
    stdin_open: true
    volumes:
      - .:/home/rstudio/work
    environment:
      - RETICULATE_PYTHON=/usr/bin/python3
      - QUARTO_PYTHON=/usr/bin/python3
      - JULIA_HOME=/opt/julia/bin
      # Mantenemos la "cirugía" de Julia para que no choque con GDAL
      - LD_PRELOAD=/usr/lib/x86_64-linux-gnu/libcurl.so.4:/usr/lib/x86_64-linux-gnu/libstdc++.so.6
      - GKSwstype=100
    ports:
      # Forzamos IPv4 para que VS Code no se pierda en el limbo del [::1]
      # JUPYTER: El 8889 de tu PC va al 8888 del contenedor
      - "127.0.0.1:8889:8888" # <----- Para programar en Notebooks localhost:8889
      # R VISOR (httpgd): El 8788 de tu PC va al 8787 del contenedor
      - "127.0.0.1:8788:8787" # <----- Para ver los gráficos de R (httpgd) localhost:8788

    depends_on:
      - db-postgis

  db-postgis:
    image: postgis_unal:final
    container_name: contenedor_postgis_unal
    environment:
      - POSTGRES_USER=profe_unal
      - POSTGRES_PASS=geomatica2025
      - POSTGRES_DB=sig_db_unal
    ports:
      # Para acceder a la base de datos desde fuera del contenedor se usa 5434
      - "127.0.0.1:5434:5432"
    volumes:
      - postgis_data_unal:/var/lib/postgresql

volumes:
  postgis_data_unal:
```

D.6 Compilar imagen/contenedor a partir de `Dockerfile` y `docker-compose.html`

El procedimiento que garantiza una instalación (carga) exitosa de las imágenes y contenedores fue documentado en Sección D.5. Use este procedimiento **solo si** necesita **compilar** una nueva imagen con contenedores que satisfagan necesidades particulares definidas a partir de un archivo `Dockerfile`. Sin embargo, el `Dockerfile` que se presenta en esta sección fue el que se utilizó para compilar la imagen empaquetada cuyo proceso de carga se define en Sección D.5.

> **Organización del Proyecto**
>
> Para automatizar el despliegue de los dos contenedores del curso (`analisis-geo` y `db-postgis`), se utilizó la orquestación de Docker. Mientras que el entorno de análisis se construyó a medida sobre la imagen base `ghcr.io/osgeo/gdal:ubuntu-full-latest`, el servicio de base de datos utilizó la imagen especializada `kartoza/postgis`.
> Para este proceso, se utilizaron dos archivos clave: **docker-compose.yml** y **Dockerfile**.

Archivo	Rol Crítico
`Dockerfile`	**El "Qué" (La Receta):** Crea el entorno políglota desde cero, instala las librerías de NASA/Copernicus y aplica la **"cirugía" de OpenSSL** para que Julia sea estable en Ubuntu Noble.
`docker-compose.yml`	**El "Cómo" (La Orquesta):** Despliega los servicios, mapea los puertos externos (**8889**, **8788**, **5434**) y asegura que sus mapas y bases de datos no se borren gracias al volumen persistente (`postgis_data_unal`).

1. Cree una carpeta utilizando su **ID de usuario de correo institucional** como nombre (el identificador que aparece antes del `@unal.edu.co`). Por ejemplo, si su correo es `juperez@unal.edu.co`, la carpeta deberá llamarse `juperez`. En adelante llamaremos a esa carpeta `su_carpeta` o la carpeta `ID UNAL`.

2. Dentro de ella, guarde los archivos que se presentan a continuación.

Antes de compilar debes:

1. Arrancar Docker Desktop
2. Cree una carpeta donde desea compilar las imágenes y copie allí los archivos `Dockerfile` y `docker-compose.html`.
3. Usando la terminal (**PowerShell** en Windows) del equipo local, cambiase a la carpeta usando el comando `cd`, ej:

```
cd "ruta_a_carpeta"
```

4. **Configuración de Terminal**: Abra PowerShell y ejecute el siguiente comando para visualizar correctamente caracteres especiales y logs:

```
# Forzar UTF8 en el PowerShell
[Console]::OutputEncoding = [System.Text.Encoding]::UTF8
```

El siguiente paso es **compilar** (solo una vez):

5. **Compilación limpia (sin usar caché) con archivo log:**

Opción 1: (recomendado) - Guarda el log al archivo y también lo muestra en la terminal - No utiliza archivos previamente descargados (`--no-cache`)

```
docker compose build --no-cache 2>&1 | tee build_sig_unal.log
```

Opción 2: - Guarda el log al archivo y no deja ver nada en la terminal - No utiliza archivos previamente descargados (`--no-cache`) - Se le asigna un etiqueta y una versión a la imagen (**mi__sig__env:v1**), lo cual evita referirse a ella por su ID hexadecimal que es imposible de recordar.

```
docker build --no-cache -t mi_sig_env:v1 . > build_details.log 2>&1
```

D.6.1 Archivo docker-compose.yml

```
services:
  # ============================================================
  # SERVICIO PRINCIPAL
  # Entorno de análisis geoespacial y científico
  # Integra R, Python, Julia, GDAL, Quarto y Jupyter
  # ============================================================
  analisis-geo:
    build: .
    image: image_sig_unal                # Nombre de la imagen Docker que se construye localmente
    container_name: contenedor_sig_unal   # Nombre del contenedor (más fácil de usar en docker exec)

    # Permite sesiones interactivas (terminal, REPLs, etc.)
    tty: true
    stdin_open: true

    # Carpeta compartida:
    # Todo lo que esté en el proyecto (host) aparece dentro del contenedor
    volumes:
      - .:/home/rstudio/work

    # ------------------------------------------------------------
    # VARIABLES DE ENTORNO
    # Aseguran que R, Python, Julia y Quarto usen versiones correctas
    # ------------------------------------------------------------
    environment:
      - RETICULATE_PYTHON=/usr/bin/python3   # Python que usará R (reticulate)
      - QUARTO_PYTHON=/usr/bin/python3       # Python que usará Quarto
      - JULIA_HOME=/opt/julia/bin             # Ruta base de Julia

      # "Cirugía" necesaria para evitar conflictos entre Julia y GDAL
      # (muy común en entornos científicos mixtos)
      - LD_PRELOAD=/usr/lib/x86_64-linux-gnu/libcurl.so.4:/usr/lib/x86_64-linux-gnu/libstdc++.so.6

      # Backend gráfico de Julia (evita errores al graficar en contenedores)
      - GKSwstype=100

    # ------------------------------------------------------------
    # PUERTOS
    # izquierda = computador del estudiante (acceso externo con localhost)
    # derecha   = contenedor (acceso interno - EXPOSE)
    # ------------------------------------------------------------
    ports:
      # Jupyter Notebook / JupyterLab
      # Se accede desde el navegador en: http://localhost:8889
      - "127.0.0.1:8889:8888"

      # Visor de gráficos de R (httpgd)
      # Permite ver gráficos interactivos fuera de RStudio
      - "127.0.0.1:8788:8787"

    # Este servicio solo se inicia cuando la base de datos esté lista
    depends_on:
      - db-postgis

  # ============================================================
  # SERVICIO DE BASE DE DATOS
  # PostgreSQL + PostGIS para datos espaciales
  # ============================================================
  db-postgis:
    image: kartoza/postgis:latest
    container_name: contenedor_postgis_unal

    # Credenciales y base de datos inicial
    environment:
      - POSTGRES_USER=profe_unal
      - POSTGRES_PASS=geomatica2025
      - POSTGRES_DB=sig_db_unal

    # Puerto para conectarse desde QGIS, DBeaver, PgAdmin, etc.
    ports:
      - "127.0.0.1:5434:5432"

    # Volumen persistente:
    # Los datos NO se pierden aunque el contenedor se borre
    volumes:
```

```
      - postgis_data_unal:/var/lib/postgresql

# ============================================================
# VOLUMENES DOCKER
# Espacio en disco administrado por Docker
# ============================================================
volumes:
  postgis_data_unal:
```

D.6.2 Archivo `Dockerfile` (Construcción del Entorno)

Este archivo constituye la **receta de construcción** del entorno de análisis. Su función es "congelar" un sistema operativo Ubuntu Noble optimizado, garantizando que todos los estudiantes trabajen exactamente con las mismas versiones de librerías, compiladores y paquetes. Con este archivo `Dockerfile` y con el anterior archivo `docker-compose.yml` (Sección D.6.1) fue que se **compiló** la imagen que después se empaquetó en el archivo `sig_unal_completo.tar` para la carga de contenedores sin conexión a Internet. Note que el archivo `docker-compose.yml` (Sección D.5.1) que acompaña a `sig_unal_completo.tar` es diferente al archivo `docker-compose.yml` detallado en esta sección.

Los pilares técnicos del archivo `Dockerfile` son:

- **Imagen Base Profesional**: Utiliza la distribución oficial de **OSGeo/GDAL**, que provee el stack más estable de librerías geoespaciales (`PROJ`, `GEOS`, `GDAL`) a nivel de sistema.
- **Pila Políglota Integrada**: Automatiza la instalación y configuración de **R**, **Python 3** y **Julia 1.10.4**, resolviendo dependencias cruzadas que suelen ser difíciles de configurar manualmente.
- **Motor de Reportes Científicos**: Instala **Quarto** y **TinyTeX**, permitiendo la generación automática de informes en PDF y HTML con calidad editorial.
- **Puente de Comunicación Maestro**: Configura el archivo `Rprofile.site`, el cual actúa como el "cerebro" que permite a R ejecutar código de Julia y capturar gráficos de Python de forma transparente.

```
# ============================================================
# Imagen base
# ------------------------------------------------------------
# Imagen oficial de OSGeo con GDAL completo, PROJ, GEOS y soporte
# raster/vector profesional. Base estándar en SIG reproducible.
# ============================================================
FROM ghcr.io/osgeo/gdal:ubuntu-full-latest


# ============================================================
# 1. Base, Locales y Pandoc
# ------------------------------------------------------------
# Configuración UTF-8 para evitar problemas con acentos,
# R, Python, Julia, LaTeX y generación de documentos.
# ============================================================
ENV LANG=en_US.UTF-8
ENV LC_ALL=en_US.UTF-8

# Herramientas base del sistema:
# - compiladores y toolchain (C/C++)
# - git / curl / wget para descargas
# - pandoc como motor universal de documentos
RUN apt-get update && apt-get install -y locales git curl wget ca-certificates \
    build-essential cmake libtool automake pkg-config software-properties-common \
    pandoc && \
    locale-gen en_US.UTF-8


# ============================================================
# Librerías criptográficas y de red
# ------------------------------------------------------------
# Necesarias para:
# - conexiones HTTPS
# - Julia
# - GDAL
```

```
# - acceso a APIs externas
# ============================================================
RUN apt-get update && apt-get install -y --no-install-recommends \
    libmbedtls-dev \
    libnng-dev \
    libssl-dev \
    libxml2-dev \
    libcurl4-openssl-dev \
    git \
    cmake


# ============================================================
# 2. R, Python y dependencias de sistema
# ------------------------------------------------------------
# Incluye soporte para:
# - SIG (GDAL / GEOS / PROJ)
# - NetCDF / HDF5
# - NASA / Copernicus
# - WhiteboxTools
# ============================================================
RUN apt-get update && apt-get install -y --no-install-recommends \
    r-base r-base-dev python3-pip python3-dev \
    psmisc lsof net-tools ffmpeg \
    libpng-dev libcairo2-dev libsystemd-dev \
    libfontconfig1-dev libfreetype6-dev \
    libharfbuzz-dev libfribidi-dev \
    libcurl4-openssl-dev libsqlite3-dev libxml2-dev libssl-dev \
    libgeos-dev libproj-dev libgdal-dev libudunits2-dev \
    libgit2-dev libssh2-1-dev libxt-dev libglpk-dev libmount-dev \
    libmagick++-dev libpcre2-dev libnetcdf-dev libhdf5-dev \
    libxt6 libxrender1 libxext6 default-jdk \
    # Elimina la restricción de pip en Debian/Ubuntu modernos
    && rm /usr/lib/python3.12/EXTERNALLY-MANAGED || true && \
    rm -rf /var/lib/apt/lists/*


# ============================================================
# 3. Quarto CLI y TinyTeX
# ------------------------------------------------------------
# Quarto: documentos reproducibles (HTML, PDF, slides)
# TinyTeX: LaTeX liviano para generación de PDF
# ============================================================
RUN curl -LO
 ↪  https://github.com/quarto-dev/quarto-cli/releases/download/v1.4.550/quarto-1.4.550-linux-amd64.deb \
    && dpkg -i quarto-1.4.550-linux-amd64.deb && rm quarto-1.4.550-linux-amd64.deb \
    && quarto install tinytex --no-prompt


# ============================================================
# 4. Python Stack
# ------------------------------------------------------------
# Librerías SIG, ciencia de datos, notebooks y visualización
# ============================================================
RUN pip3 install --upgrade --ignore-installed --break-system-packages \
    # Ya estaban:
    shapely matplotlib numpy geopandas fiona pyyaml nbformat nbclient ipykernel \
    # Nuevos agregados:
    pandas rasterio rasterstats scipy psycopg2 pysal earthaccess cdsapi leafmap \
    geemap segment-geospatial geoai-py lidar pygis whitebox whiteboxgui streamlit \
    ghp-import jupyter-book jupyterlab jupytext mystmd notebook


# ============================================================
# 5. R Stack
# ------------------------------------------------------------
# Instalación desde CRAN optimizado de Posit para Linux
# Incluye SIG, visualización, modelado y ML espacial
# ============================================================
RUN R -e "options(timeout = 1000, Ncpus = parallel::detectCores(), repos = c(CRAN =
 ↪  'https://packagemanager.posit.co/cran/__linux__/noble/latest')); \
    install.packages(c('ggplot2', 'patchwork', 'dplyr', 'remotes', 'languageserver', \
    'rmarkdown', 'units', 's2', 'sf', 'terra', 'stars', 'reticulate', 'IRkernel', \
    'unigd', 'cpp11', 'systemfonts', 'AsioHeaders', 'png', 'grid', 'JuliaCall', 'JuliaConnectoR', \
    # Nuevos CRAN:
    'tidyverse', 'tmap', 'leaflet', 'googleway', 'ggspatial', 'mapview', 'plotly', \
    'rasterVis', 'cartogram', 'geogrid', 'geofacet', 'linemap', 'tanaka', 'rayshader', \
    'lwgeom', 'gstat', 'spdep', 'spatialreg', 'stplanr', 'sfnetworks', 'spatstat', \
    'stpp', 'magrittr', 'giscoR', 'caret', 'tidymodels', 'spatialsample', 'CAST', \
    'mlr3spatial', 'mlr3spatiotempcv', 'ncdf4', 'whitebox'))"


# ============================================================
# Starsdata y repositorios específicos
# ------------------------------------------------------------
# Se compila desde código fuente y se usan repos especiales
```

```
# ============================================================
RUN R -e "options(timeout = 30000, Ncpus = parallel::detectCores()); \
    install.packages('starsdata', repos='https://cran.uni-muenster.de/pebesma/', type='source')" && \
    R -e "options(timeout = 2000, Ncpus = parallel::detectCores()); \
    install.packages(c('mlr3cmprsk', 'survdistr'), repos=c('https://mlr3learners.r-universe.dev',
 ↪  'https://cloud.r-project.org')); \
    install.packages('geocompkg', repos=c('https://geocompr.r-universe.dev',
 ↪  'https://cloud.r-project.org'), dependencies=TRUE); \
    whitebox::install_whitebox(); IRkernel::installspec(user = FALSE)"


# ============================================================
# httpgd estable
# ------------------------------------------------------------
# Dispositivo gráfico moderno para R (gráficos en navegador)
# ============================================================
RUN wget https://cran.r-project.org/src/contrib/Archive/httpgd/httpgd_2.0.3.tar.gz && \
    R CMD INSTALL httpgd_2.0.3.tar.gz && rm httpgd_2.0.3.tar.gz

# ============================================================
# 6. Puente Python ↔ R (Backend de Matplotlib para reticulate)
# ------------------------------------------------------------
# Permite que gráficos de matplotlib generados desde Python
# puedan ser capturados y mostrados correctamente desde R
# usando reticulate (especialmente en notebooks y httpgd).
# ============================================================

# Creamos la estructura esperada por reticulate
RUN mkdir -p /usr/local/lib/python3.12/dist-packages/reticulate/matplotlib && \
    touch /usr/local/lib/python3.12/dist-packages/reticulate/__init__.py

# Definimos una función que R puede interceptar
# para recibir el path de la imagen generada por Python
RUN printf 'def r_graphic_command(path):\n    import os\n    if os.path.exists(path):
 ↪  print(f"r_graphic_command: {path}")\n' >
 ↪  /usr/local/lib/python3.12/dist-packages/reticulate/__init__.py

# Backend custom de matplotlib:
# - Renderiza con Agg
# - Guarda el gráfico como PNG temporal
# - Notifica a R para que lo muestre
RUN printf 'import matplotlib\nfrom matplotlib.backends.backend_agg import FigureCanvasAgg\nfrom
 ↪  matplotlib.backend_bases import FigureManagerBase\n\ndef show(*args, **kwargs):\n    import os,
 ↪  tempfile, reticulate\n    fd, path = tempfile.mkstemp(suffix=".png")\n    os.close(fd)\n
 ↪  matplotlib.pyplot.savefig(path)\n    if hasattr(reticulate, "r_graphic_command"):\n
 ↪  reticulate.r_graphic_command(path)\n\nclass FigureManager(FigureManagerBase):\n    def show(self):\n
 ↪  show()\n\ndef new_figure_manager(num, *args, **kwargs):\n    FigureClass = kwargs.pop("FigureClass",
 ↪  matplotlib.figure.Figure)\n    thisFig = FigureClass(*args, **kwargs)\n    return
 ↪  new_figure_manager_given_figure(num, thisFig)\n\ndef new_figure_manager_given_figure(num, figure):\n
 ↪  canvas = FigureCanvasAgg(figure)\n    manager = FigureManager(canvas, num)\n    return
 ↪  manager\n\nFigureCanvas = FigureCanvasAgg\n' >
 ↪  /usr/local/lib/python3.12/dist-packages/reticulate/matplotlib/backend.py


# ============================================================
# 7. Julia: Instalación y Wrapper de Seguridad
# ------------------------------------------------------------
# Se instala Julia binaria oficial y se fuerza el uso de
# librerías del sistema para evitar conflictos con GDAL/OpenSSL
# ============================================================

# Versión fija de Julia (reproducibilidad total)
ENV JULIA_VERSION=1.10.4

# Descarga e instalación manual de Julia
RUN wget https://julialang-s3.julialang.org/bin/linux/x64/1.10/julia-${JULIA_VERSION}-linux-x86_64.tar.gz
 ↪  \
    && tar -xvzf julia-${JULIA_VERSION}-linux-x86_64.tar.gz && mv julia-1.10.4 /opt/julia

# Wrapper de Julia:
# - Fuerza LD_PRELOAD
# - Evita conflictos de libcurl / libstdc++
# - Garantiza compatibilidad con GDAL
RUN printf '#!/bin/bash\nexport
 ↪  LD_PRELOAD=/usr/lib/x86_64-linux-gnu/libcurl.so.4:/usr/lib/x86_64-linux-gnu/libstdc++.so.6\nexport
 ↪  JULIA_PKG_USE_CLI_GIT=true\n/opt/julia/bin/julia "$@"\n' > /usr/local/bin/julia && chmod +x
 ↪  /usr/local/bin/julia


# ============================================================
# 8. Julia: Configuración de librerías nativas
# ------------------------------------------------------------
# Se fijan explícitamente las rutas del sistema para:
# - GDAL
# - GEOS
```

```
# evitando que Julia use binarios incompatibles
# ============================================================
RUN mkdir -p /root/.julia/environments/v1.10 && \
    printf "[LocalPreferences]\nGDAL_jll = { libgdal_path = \"/usr/lib/libgdal.so\" }\nGEOS_jll = {
     ↪  libgeos_path = \"/usr/lib/x86_64-linux-gnu/libgeos_c.so\" }\n" >
     ↪  /root/.julia/environments/v1.10/LocalPreferences.toml


# ============================================================
# 9. Julia: Instalación de paquetes
# ------------------------------------------------------------
# Stack SIG completo:
# - Rasters, ArchGDAL, GeoStats, Makie
# - Conectores DB, CSV, NetCDF
# - Visualización y notebooks (IJulia)
# ============================================================
RUN julia -e 'using Pkg; Pkg.add(["Preferences", "Suppressor", "RCall", "LibGEOS", "Tables", "DataFrames",
 ↪  "Plots", \
    "Statistics", "ArchGDAL", "LibPQ", "GeoDataFrames", "IJulia", "CSV", "CairoMakie",
 ↪  "AlgebraOfGraphics", \
    "DimensionalData", "FlexiJoins", "GeoFormatTypes", "GeoInterface", "GeoJSON", "GeoMakie",
 ↪  "GeometryOps", \
    "Makie", "MakieCore", "NaturalEarth", "Proj", "Rasters", "StatsBase", "Tyler", "GeoStats", "Graphs", \
    "NCDatasets", "MetaGraphsNext"])'


# ============================================================
# Cirugía de librerías (OpenSSL)
# ------------------------------------------------------------
# Fuerza a Julia a usar OpenSSL del sistema (Ubuntu Noble)
# Evita errores de TLS y descargas de paquetes
# ============================================================
RUN find /root/.julia/artifacts -name "libssl.so*" -exec ln -sf /usr/lib/x86_64-linux-gnu/libssl.so.3 {}
 ↪  \; && \
    find /root/.julia/artifacts -name "libcrypto.so*" -exec ln -sf
     ↪  /usr/lib/x86_64-linux-gnu/libcrypto.so.3 {} \;


# ============================================================
# Precompilación total de Julia
# ------------------------------------------------------------
# Garantiza arranque instantáneo en:
# - VSCode
# - Jupyter
# - IJulia
# ============================================================
RUN julia -e 'using Pkg; Pkg.precompile()'


# ============================================================
# Configuración de paralelismo
# ------------------------------------------------------------
# Julia usará automáticamente todos los núcleos disponibles
# ============================================================
ENV JULIA_NUM_THREADS=auto

# ============================================================
# 10. CONFIGURACIÓN MAESTRA Rprofile.site
# ------------------------------------------------------------
# Este archivo se ejecuta automáticamente cada vez que inicia R.
# Centraliza la integración R ↔ Julia ↔ Python ↔ VSCode.
#
# Build 47.42:
# - Control de DPI, tamaño y fuentes
# - Ejecución segura de Julia desde R
# - Renderizado consistente de gráficos
# - Hook para Matplotlib vía reticulate
# ============================================================

RUN cat << 'EOF' > /usr/lib/R/etc/Rprofile.site
# ============================================================
# --- 1. AJUSTES DE SISTEMA ---
# ------------------------------------------------------------
# Variables globales para que R sepa dónde encontrar:
# - Julia
# - Python usado por Quarto
# ============================================================
Sys.setenv(JULIA_BINDIR = "/opt/julia/bin")
Sys.setenv(QUARTO_PYTHON = "/usr/bin/python3")

# ============================================================
# Código Julia embebido (auto-sanable)
# ------------------------------------------------------------
# Se define como string para:
# - Inyectarse dinámicamente en Julia
```

```
# - Evitar errores si el kernel se reinicia
# - Garantizar reproducibilidad en notebooks
# ============================================================
.unal_julia_code <- '
using Suppressor, Plots, Statistics

# Ejecutor central de código Julia desde R
# - Evalúa múltiples expresiones
# - Captura stdout
# - Maneja gráficos y texto
function _unal_core_executor(code, is_plot, filename, dpi, w, h, fs)
    @capture_out begin
        if is_plot
            # Parámetros gráficos homogéneos (DPI, tamaño, fuentes)
            default(dpi=dpi, size=(w, h), titlefontsize=fs+2,
                    guidefontsize=fs, tickfontsize=fs-2, legendfontsize=fs-1)
        end
        pos = 1
        while pos <= lastindex(code)
            start_idx = pos
            try
                ex, pos = Meta.parse(code, pos)
                cmd_part = strip(code[start_idx:prevind(code, pos)])
                if !isempty(cmd_part)
                    println("julia> ", cmd_part)
                    res = eval(ex)
                    if res !== nothing && !(res isa Plots.Plot)
                        show(stdout, MIME("text/plain"), res)
                        println()
                    end
                    println()
                end
            catch e
                println("julia> Error: ", e)
                break
            end
        end
        # Guardado del gráfico si aplica
        if is_plot && current() !== nothing; savefig(current(), filename); end
    end
end
'

# ============================================================
# Inicialización segura de Julia
# ------------------------------------------------------------
# - Verifica que JuliaConnectoR esté disponible
# - Inyecta el ejecutor solo una vez por sesión
# ============================================================
.ensure_julia_ready <- function() {
  if (!requireNamespace("JuliaConnectoR", quietly = TRUE)) stop("JuliaConnectoR missing")
  if (!JuliaConnectoR::juliaEval('isdefined(Main, :_unal_core_executor)')) {
    JuliaConnectoR::juliaEval(.unal_julia_code)
  }
}

# ============================================================
# j_eval(): ejecutar código Julia (solo texto)
# ------------------------------------------------------------
# Uso típico:
# j_eval("1 + 1")
# ============================================================
j_eval <- function(cmd) {
  .ensure_julia_ready()
  cat(JuliaConnectoR::juliaCall("_unal_core_executor", cmd, FALSE, "", 72, 800, 500, 12))
}

# ============================================================
# j_plot(): ejecutar código Julia con gráficos
# ------------------------------------------------------------
# - Guarda el gráfico en PNG
# - Lo renderiza directamente en R
# ============================================================
j_plot <- function(cmd, n = "tmp_plot.png", dpi = 300, w = 800, h = NULL, ratio = 1.6, fontsize = 12) {
  .ensure_julia_ready()
  if (is.null(h)) h <- round(w / ratio)
  log_out <- JuliaConnectoR::juliaCall("_unal_core_executor", cmd, TRUE, n, dpi, as.integer(w),
 ↪  as.integer(h), as.integer(fontsize))
  if (nchar(log_out) > 0) cat(log_out)
  if (file.exists(n)) {
    img <- png::readPNG(n)
    grid::grid.newpage()
    grid::grid.raster(img)
  }
}
```

```
# ============================================================
# --- 2. CARGA DE LIBRERÍAS Y DISPOSITIVOS ---
# ------------------------------------------------------------
# Librerías base para renderizar imágenes
# ============================================================
library(png)
library(grid)

if (interactive()) {

  # ============================================================
  # Visor gráfico httpgd (VSCode / navegador)
  # ------------------------------------------------------------
  # Permite gráficos interactivos persistentes
  # ============================================================
  if (requireNamespace("httpgd", quietly = TRUE)) {
    options(device = "httpgd", httpgd.host = "0.0.0.0", httpgd.port = 8787, httpgd.token = FALSE)
  }

  # ============================================================
  # Hook Python → R (Matplotlib)
  # ------------------------------------------------------------
  # Captura gráficos de matplotlib y los muestra en R
  # usando el backend custom definido en Docker
  # ============================================================
  setHook(packageEvent("reticulate", "onLoad"), function(...) {
    try({
      ret_py <- reticulate::import("reticulate", delay_load = TRUE)
      reticulate::py_set_attr(ret_py, "r_graphic_command", function(path) {
        if (file.exists(path)) {
          img <- png::readPNG(path)
          grid::grid.newpage()
          grid::grid.raster(img)
        }
      })
      reticulate::py_run_string("import matplotlib;
 ↪  matplotlib.use('module://reticulate.matplotlib.backend')")
    }, silent = TRUE)
  })
}
EOF


# ============================================================
# Compatibilidad multi-R (opcional)
# ------------------------------------------------------------
# Permite que R instalado en rutas alternativas use
# exactamente la misma configuración
# ============================================================
#RUN cp /usr/lib/R/etc/Rprofile.site /etc/R/Rprofile.site


# ============================================================
# 10. Finalización del contenedor
# ------------------------------------------------------------
# Directorio de trabajo compartido
# Permisos amplios para docencia
# ============================================================
WORKDIR /home/rstudio/work
RUN chmod -R 777 /home/rstudio/work

# Puertos:
# 8888 → JupyterLab
# 8787 → httpgd / RStudio-like viewer
EXPOSE 8888
EXPOSE 8787

# Arranque por defecto: JupyterLab
CMD ["jupyter", "lab", "--ip=0.0.0.0", "--port=8888", "--no-browser", "--allow-root",
 ↪  "--NotebookApp.token='geomatica2025'"]
```

D.7 Arrancar y detener las imágenes

D.7.1 Arrancar las imágenes

Los comandos `build` o `load` una vez se terminan satisfactoriamente, no necesitan volver a ejecutarse. El principal comando para subir el servicio de las imágenes instaladas se muestra a continuación.

Use este comando siempre antes de iniciar a trabajar con los contenedores instalados. *Esto activará los dos contenedores: uno para análisis geoespacial y otro para la base de datos.*:

```
docker compose up -d
```

Si detuvo los contenedores con `docker compose stop`, use este comando para **reiniciarlos**:

```
docker compose start
```

D.7.2 Detener las imágenes

Para **apagar** los contenedores sin borrar datos:

```
docker compose stop
```

Si necesita **borrar todo** (por ejemplo para reinstalar) (los datos de la DB se mantienen en el volumen):

```
docker compose down
```

D.8 Verificar logs de instalación

Verificación de Logs: Si desea ver el log de instalación de una imagen compilada, ejecute el siguiente comando, sin embargo la url de acceso a **Jupyter Lab** mostrada después de ejecutar este comando puede estar errónea. Para acceder a **Jupyter Lab** vea (Sección B.7)

```
docker logs contenedor_sig_unal
```

D.9 Comandos docker

Use la siguiente tabla como referencia para gestionar sus contenedores desde la terminal de su sistema anfitrión (Windows/Mac/Linux).

Acción	Comando	Propósito
Construcción Inicial	`docker-compose up --build -d`	Compila el Dockerfile y levanta los servicios (Solo una vez).
Inicio Diario	`docker-compose up -d`	Inicia los servicios de forma instantánea si ya fueron construidos.
Monitoreo de Procesos	`docker logs -f entorno_unal_sig`	Sigue los logs en vivo (ideal para ver pre-compilaciones).
Ver logs recientes	`docker logs --tail 20 entorno_unal_sig`	Muestra las últimas 20 líneas (ideal para buscar el token).

Acción	Comando	Propósito
Apagado	`docker-compose down`	Detiene los servicios y libera recursos del sistema.
Prueba de R (sf)	`docker exec entorno_unal_sig R -e "library(sf); print('R-Spatial detectado')"`	Confirmar que R reconoce los drivers geoespaciales del sistema.
Prueba de Python	`docker exec entorno_unal_sig python -c "import shapely; import geopandas; print('Python OK')"`	Confirmar que el stack de Python (GeoPandas/Shapely) está instalado.
Prueba de Julia	`docker exec entorno_unal_sig julia -e "using LibGEOS; println('Julia OK')"`	Confirmar que Julia tiene acceso a los binarios geoespaciales.

D.10 Compilación avanzada

Para trabajar con entornos SIG, a menudo necesitamos reconstruir imágenes sin basura previa.

- **Compilación Limpia (Sin Cache) con Log:** Para asegurar que Docker descargue todas las librerías desde cero y guarde un registro detallado de los errores: `docker build --no-cache -t mi_sig_env:v1 . --progress=plain > build_details.log 2>&1`
- **Subir Imagen a Repositorio:** `docker tag mi_sig_env:v1 usuario_dockerhub/mi_sig_env:v1` `docker push usuario_dockerhub/mi_sig_env:v1`
- **Cargar en VSCode:** Una vez el contenedor esté corriendo, use la extensión **Dev Containers** -> Botón verde inferior izquierdo -> *Attach to Running Container*.

D.11 Cambio de ruta de almacenamiento (Windows)

Si el disco `C:` se agota debido a las imágenes de Docker, siga estos pasos en **Docker Desktop**: 1. Vaya a **Settings** (engranaje). 2. **Resources** > **Advanced**. 3. En **Disk image location**, cambie la ruta a un disco con mayor capacidad (ej. `D:\DockerImages`). 4. Presione **Apply & Restart**.

D.12 Optimización de memoria RAM y swap

Cuando procesamos datos masivos (como imágenes Sentinel-2 o rásters globales), la RAM física de 16GB suele ser insuficiente. Para evitar que el sistema aborte los procesos con el error **"Killed"**, debemos configurar un "pulmón" de emergencia en dos niveles.

D.12.1 1. El "pulmón" de Windows: memoria virtual

Obligamos a Windows a usar el disco duro como si fuera RAM de reserva. Si tiene un disco sólido (SSD) secundario con mucho espacio libre (ej. Disco D:), es el lugar ideal para configurarlo.

1. En el buscador de Windows, escriba: **"Ajustar la apariencia y rendimiento de Windows"**.

2. Vaya a la pestaña **Opciones avanzadas** > sección **Memoria virtual** > clic en **Cambiar**.
3. Desmarque la opción "Administrar automáticamente el tamaño del archivo de paginación para todas las unidades".
4. Seleccione la unidad de disco (C: o D:) y marque **Tamaño personalizado**.
5. Establezca los siguientes valores (recomendados para este curso):
   - **Tamaño inicial**: `16384` MB (16 GB).
   - **Tamaño máximo**: `32768` MB (32 GB).
6. Haga clic en **Establecer**, luego en **Aceptar** y **reinicie su computadora** para aplicar los cambios.

D.12.2 2. El "túnel" de docker: swap de WSL2

Docker Desktop corre sobre WSL2 (*Windows Subsystem for Linux*), el cual tiene su propio "presupuesto" limitado. Por defecto, este túnel es estrecho (máximo 4GB de Swap). Si no ampliamos esto, el contenedor nunca podrá aprovechar realmente el espacio que le asignamos a Windows.

Cómo ampliar la tubería: 1. Presione `Win + R`, escriba `%UserProfile%` y presione Enter. 2. Busque el archivo `.wslconfig`. Si no existe, créelo con el Bloc de Notas. 3. Pegue el siguiente contenido:

```
[wsl2]
memory=12GB # RAM máxima que le permitimos usar a Linux/Docker
swap=16GB   # El nuevo tamaño de swap que verá el comando 'top' en la terminal
```

4. **Aplicar cambios**: Guarde el archivo, abra una terminal (PowerShell) y escriba `wsl --shutdown`. Luego, inicie Docker Desktop nuevamente.

D.13 Higiene y limpieza de choque

Para garantizar que un renderizado de Quarto llegue al 100% sin colapsar el contenedor, aplique estas medidas de higiene:

- **Cierre "Vampiros"**: Aplicaciones como Chrome, Edge, Teams y Slack consumen RAM de forma agresiva. Ciérrelas antes de procesos pesados.
- **Limpieza de disco**: Use el Liberador de espacio en disco (`cleanmgr`) para vaciar archivos de volcado de memoria.
- **La "Escoba" en el Código**: Use comandos de limpieza entre procesos de diferentes lenguajes:
  - **R**: `rm(obj); gc(full = TRUE)`
  - **Python**: `del var; gc.collect()`
  - **Julia**: `GC.gc()`

> ⚠ Importante para Visualización
>
> En Julia y R, evite generar gráficos interactivos pesados dentro de bucles de procesamiento. La acumulación de objetos visuales en la memoria de VSCode es la causa número uno de colapsos en el contenedor.

Apéndice E

Git y GitHub: Sincronización Local-Remota

E.1 Instalación de Git

E.2 Creación de cuenta en GitHub

E.3 Control de versiones - flujo de trabajo en PowerShell

Para gestionar el repositorio de forma robusta y tener control total sobre el historial de cambios, utilice la siguiente secuencia de comandos desde la terminal del contenedor:

Tabla E.1: Comandos esenciales para la gestión de Git en terminal

Acción	Comando	Propósito
Verificar estado	`git status`	Muestra la "verdad absoluta" local: archivos modificados o listos para commit.
Sincronizar índices	`git remote update`	Consulta el servidor para saber si hay cambios nuevos en la nube sin descargarlos aún.
Preparar cambios	`git add .`	Agrega todos los cambios detectados al área de preparación (*Staging*).
Confirmar versión	`git commit -m "mensaje"`	Crea un punto de control permanente en el historial local.
Enviar al remoto	`git push`	Sube los *commits* locales al servidor (GitHub).

Acción	Comando	Propósito
Actualizar local	`git pull`	Descarga y fusiona los cambios del servidor en su entorno de trabajo.
Ver historial	`git log --oneline`	Muestra un resumen simplificado de la línea de tiempo del proyecto.

E.4 Diagnóstico de archivos "invisibles" o errores de rastreo

Si ha modificado un archivo pero este no aparece listado tras ejecutar `git status`, las causas técnicas suelen ser las siguientes:

1. **Interferencia de Repositorios Anidados:** Si clonó una carpeta que ya contenía un directorio `.git`, el repositorio principal la tratará como un objeto opaco (submódulo) y no rastreará sus archivos internos.
   - **Solución:** Verifique la presencia de carpetas ocultas con `ls -a` y elimine la carpeta `.git` interna si desea que el repositorio raíz tome el control: `rm -rf nombre_carpeta/.git`.
2. **Filtros en el archivo `.gitignore`:** El archivo puede tener reglas que excluyan carpetas enteras por nombre o extensión.
   - **Verificación:** Use el comando `git status --ignored` para listar todos los archivos que Git está omitiendo deliberadamente.
3. **Estado del sistema de archivos:** Git opera sobre los datos escritos en el disco duro. Asegúrese de que el editor de texto haya guardado efectivamente los cambios en el archivo físico antes de consultar el estado.

> **!** Persistencia de Credenciales
>
> Si la terminal le solicita usuario y contraseña en cada `push` o `pull`, puede configurar Git para que recuerde sus credenciales temporalmente en la memoria del contenedor con el siguiente comando: `git config --global credential.helper cache`

E.5 Subír el contenido (carpeta local hacia GitHub)

Si ya tiene sus archivos en Windows y quiere subirlos a un repositorio recién creado en GitHub:

1. **Instalar Git**

- **Descarga**: Acceda a git-scm.com/download/win.
- **Instalación**: Ejecute el instalador manteniendo las opciones por defecto; asegúrese de que la opción **"Git from the command line and also from 3rd-party software"** esté activa.
- **Verificación**: Ejecute `git --version` en una terminal para confirmar la instalación.

2. **Crear una cuenta en GitHub**

3. **Abrir Terminal en la carpeta local:**

4. **Incicializar Git** en la carpeta local:

```
git init
```

5. Ver el **estado actual** del repositorio Git:

```
git status
```

6. **Archivo `.gitignore`**: Crear un archivo de texto `.gitignore` para indicarle a Git los archivos y carperas que nunca deben ser registrados con Git cuando se use el comando `git add` (siguiente item del procedimiento). Estos archivos que pueden estar en la raíz de la carpeta o dentro de otras sub-carpetas no se registrarán automáticamente cuando ejecute el comando `git add .` (ver mas adelante). Sin embargo puede forzar una carpeta o algunos tipos de archivos que estén marcados para ignorar dentro del `.gitignore` (ver mas adelante).

```
.quarto/
_freeze/
.ipynb_checkpoints/
__pycache__/
notebooks/
scripts/
data/
```

7. **Registrar (add)** el archivo `.gitignore` con Git:

```
git add .gitignore
```

8. Revise nuevamente el **estátus** y verifique cambios. Puede incluir el listado de los ignorados:

```
git status --ignored
```

9. **Salvar (commit)** el archivo en Git (local). Note que en el comando, el texto "*Archivo .gitignore*" es un comentario que describe lo que está salvando:

```
git commit -m "Archivo .gitignore"
```

10. Cambiar al nombre de **rama (branch)** a **principal (main)**. Pueden haber diferentes ramas con diferentes versiones del mismo archivo. Para cambiar a **main**:

```
git branch -M main
```

11. **Conectar** el repositorio local (Git) con nuestro repositorio en GitHub.

Para conectar Git con GitHub, puede hacerlo con SSH (debe realizar el proceso de instalación de llaves Sección E.6) o con HTTPS (debe generar un PAT desde `github.com` que usará como clave de autenticación). En este caso usaremos HTTPS.

Cree un nuevo repositorio en GitHub y copie la url HTTP del repositorio. Es algo como: `https://github.com/usuario/reposit`

```
git remote add origin https://github.com/usuario/repositorio.git
```

12. Procedimiento para **generar el PAT** (Personal Access Token) en GitHub.com {#ancla-github_pat}

- En lugar de escribir tu contraseña cuando el terminal te la pida, debes pegar un token generado en tu cuenta.
- Ve a tu cuenta de GitHub.com: `Settings` > `Developer settings` > `Personal access tokens` > `Tokens (classic)`.
- Genera un nuevo token **con el permiso repo activado.**
- Copia el token (no lo volverás a ver).

13. **Subir (push)** la información guardada localmente en Git a nuestro repositorio en GitHub:

```
git push -u origin main
# Cuanto solicite el password use el PAT/token generado previamente en github.com
```

Puedes ver el estatus `git status` y realizar un procedimiento similar para registrar (`add`), salvar (`commit`) y subir (`push`) mas archivos. Otras opciones de Git se describen a continuación:

- Registrar archivos específicos con Git (local):

```
git add file1 file2 ...
```

- **Forzar archivos** y/o carpetas que están marcados para ignorar (bandera `-f`): el comando a continuación sincronizará con Git toda la carpeta `_site/site_libs/` y su contenido, así contenga archivos o carpetas marcados para ignorar en `.gitignore`. {#ancla-git_add_minus_f}

```
git add -f _site/site_libs/
```

- Registrar **todos los archivos y carpetas** disponibles y sin registrar, sin embargo todos los archivos y carpetas dentro de `.gitignore` no serán tenidos en cuenta {#ancla-git_add_punto}:

```
git add .
```

- En **resumen** estos son los comandos que comúnmente usarás para mentener sincronizados tus archivos con Git y tu Git con GitHub, es decir el proceso de subir (push) archivos a la nube.
  - `git add` (agregar archivos a Git)
  - `git commit` (guardar archivos en Git)
  - `git push` (subir archivos a GitHub)
- Los otros comandos vistos solo se ejecutan **una vez** o de acuerdo con **necesitades específicas**:
  - `git init` (una vez)
  - `git remote add origin` (una vez)
  - `git branch -M main` (lo debes usar para cambiar a main en caso de tener múltiples ramas)

E.6 Autenticación segura SSH - Windows

La autenticación mediante SSH permite interactuar con GitHub de forma segura sin necesidad de ingresar credenciales manualmente en cada operación. Siga este procedimiento detallado para configurar su acceso.

E.6.1 Requisito previo: creación de cuenta en GitHub

El correo que use para crear esta cuenta **debe ser el mismo correo** usado para crear las llaves SSH para conexión automática.

E.6.2 Requisito previo: instalación de Git

Es indispensable que su sistema operativo cuente con el motor de Git antes de iniciar la configuración:

E.6.3 Preparación del entorno local

- **Navegación**: Diríjase mediante el Explorador de Archivos a la carpeta raíz de su proyecto (ejemplo: carpeta `ID Unal`).
- **Terminal**: Abra una ventana de PowerShell en esa ubicación (clic derecho -> **"Abrir en Terminal"**).
- **Inicialización**: Prepare el repositorio local mediante el comando:

```
git init
```

E.6.4 Gestión de llaves SSH en Windows

SSH (*Secure Shell*) es un protocolo de red diseñado para el acceso, administración y transferencia de datos entre un equipo local y un servidor remoto mediante un canal cifrado. En nuestro flujo de trabajo, implementamos el algoritmo **Ed25519**, un estándar de vanguardia basado en criptografía de curva elíptica (EdDSA). Este método no solo ofrece un nivel de seguridad superior frente a ataques de fuerza bruta, sino que permite una **autenticación transparente y sin contraseña**, actuando como un "pasaporte digital" que vincula de forma única su estación de trabajo con GitHub.

> ¿Por qué Ed25519?
>
> A diferencia de los estándares antiguos (como RSA), **Ed25519** genera llaves más cortas, rápidas y significativamente más seguras, optimizando la latencia en cada comunicación con el repositorio.

- **Comprobación de llaves existentes**: Verifique si su usuario ya posee llaves configuradas:

```
ls ~/.ssh
```

```
* Si el directorio está vacío o solo contiene `known_hosts`, proceda a generar una nueva.
* Si visualiza los archivos `id_ed25519` e `id_ed25519.pub`, puede saltar al paso de registro en GitHub.
```

- **Generación de llaves**: Cree un par de llaves seguras utilizando el algoritmo Ed25519:

El correo que use en el siguiente paso, **debe** ser el correo con el que creó la cuenta de GitHub: "su_correo@domimio.com"

```
ssh-keygen -t ed25519 -C "su_correo@domimio.com"
```

```
* **Nota**: Presione `ENTER` en todas las solicitudes para aceptar las rutas por defecto y no asignar ur
```

> **Seguridad de la Identidad**
>
> La **llave pública** (`id_ed25519.pub`) es la que se entrega a GitHub, mientras que la **llave privada** (`id_ed25519`) funciona como su sello personal secreto y nunca debe salir de su equipo.

E.6.5 Registro de la identidad en GitHub

- **Copiar la firma pública**: Obtenga el contenido de su llave para registrarla:

```
cat $env:USERPROFILE\.ssh\id_ed25519.pub
```

```
*Seleccione y copie toda la cadena de texto resultante.*
```

- **Configuración en la plataforma**:
  - Acceda a su cuenta en GitHub.com.
  - Entre a **Settings** -> **SSH and GPG keys**.
  - Seleccione **New SSH key**.
  - Asigne el título `Windows-UNAL-2025`, pegue el contenido en el campo **Key** y presione **Add SSH key**.

E.6.6 Activación del agente SSH del sistema

Para que Windows gestione su identidad de forma transparente, debe habilitar el servicio correspondiente:

- **Modo Administrador**: Cierre su terminal actual y abra una nueva sesión de **PowerShell como Administrador**.
- **Habilitar Servicio**: Ejecute los siguientes comandos para automatizar el arranque del agente:

```
Set-Service ssh-agent -StartupType Automatic
Start-Service ssh-agent
```

- **Cargar la llave privada**: Registre su llave en el agente activo:

```
ssh-add $env:USERPROFILE\.ssh\id_ed25519
```

E.6.7 Verificación de la autenticación

- **Prueba de conexión**: Compruebe que el túnel de comunicación con los servidores de GitHub funciona correctamente:

```
ssh -T git@github.com
```

- **Resultado esperado**: Si la configuración es exitosa, recibirá un mensaje similar a: `Hi usuario! You've successfully authenticated, but GitHub does not provide shell access.`

E.6.7.1 ¡Túnel SSH Activo y Configurado!

Has establecido una identidad digital segura entre tu equipo y GitHub. A partir de este momento, la comunicación para sincronizar tus repositorios será transparente y automática.

- **Identidad Permanente**: Esta configuración se realiza **una sola vez** por computador; tu "firma digital" ya reside en el sistema.
- **Adiós a las Interrupciones**: GitHub ya no solicitará tu usuario, contraseña o tokens personales de acceso en cada operación.
- **Compatibilidad Total**: Esta llave es reconocida automáticamente por **PowerShell**, **VSCode**, **Positron**, **RStudio** y cualquier otra terminal científica que utilices.
- **Seguridad de Grado Profesional**: Tus envíos (`push`) viajan ahora por un canal cifrado bajo el estándar Ed25519.

E.7 Autenticación segura SSH - Linux (Docker)

La autenticación mediante **SSH** permite interactuar con GitHub de forma segura sin ingresar credenciales manualmente. En contenedores Docker, esto soluciona errores de validación SSL y facilita el uso de extensiones de Quarto.

E.7.1 Preparación del Entorno Global

Antes de generar las llaves, configura tu identidad de Git y verifica que el contenedor tenga salida a internet.

- **Configurar identidad**:

```
git config --global user.email "correo@dominio.com"
git config --global user.name "usuario_github"
```

- **Verificar conexión**:

```
curl -I https://github.com
```

> ⚠ Advertencia
>
> Si recibes el error * "Could not resolve host"*, revisa la configuración de red de tu contenedor.

E.7.2 Gestión de llaves SSH (Algoritmo Ed25519)

Implementamos **Ed25519**, el estándar actual por su seguridad y velocidad.

- **Comprobar llaves existentes**:

```
ls -la ~/.ssh
```

- **Generar nueva llave**: Ejecuta el comando y presiona `ENTER` en todas las solicitudes (sin contraseña).

```
ssh-keygen -t ed25519 -C "alexyshr@gmail.com"
```

> **Nota**
>
> La **llave pública** (`id_ed25519.pub`) se registra en GitHub; la **llave privada** (`id_ed25519`) es tu firma secreta.

E.7.3 Registro de Identidad en GitHub

1. **Visualizar y copiar la llave pública**:

```
cat ~/.ssh/id_ed25519.pub
```

2. **Configurar en la web**:
   - Ve a **Settings** -> **SSH and GPG keys** en tu cuenta de GitHub.
   - Haz clic en **New SSH key**.
   - Título: `Docker-RStudio-Geomatica`.
   - Pega el contenido y guarda.

E.7.4 Activación del Agente SSH

Para que el sistema use tu llave automáticamente, inicia el agente y regístrala:

```
# Iniciar agente
eval "$(ssh-agent -s)"

# Cargar la llave privada
ssh-add ~/.ssh/id_ed25519
```

E.7.5 Verificación Final

Comprueba que el túnel de comunicación funciona correctamente:

```
ssh -T git@github.com
```

Resultado esperado: > `Hi usuario_github! You've successfully authenticated, but GitHub does not provide shell access.`

E.7.5.1 ¡Configuración Exitosa!

Tu contenedor ahora es un entorno de confianza para GitHub. Comandos como `quarto add` o `git push` funcionarán de forma transparente.

E.8 Descargar contenido (GitHub hacia carpeta local)

Para explicar este procedimiento, usaremos el repositorio GitHub con el material de la clase. Los contenidos de la clase están en `https://github.com/alexyshr/ProgramacionSIG2026.git` (HTTPS) o `git@github.com:alexyshr/ProgramacionSIG2026.git` (SSH). La elección de uno de estos dos protocolos (HTTPS o SSH), depende de si ha configurado su identidad digital, ver Sección E.6.

> ⚠ ¡Advertencia sobre la Estructura de Carpetas!
>
> Es fundamental para el orden del curso que el repositorio con los contenidos de la clase se conecte a una **carpeta independiente**.
> **No sincronice estos archivos dentro de su carpeta local con los contenedores `ID UNAL`.** Mantener directorios separados garantiza que sus ejercicios personales y los materiales de lectura del profesor no se mezclen, evitando errores de sobreescritura y conflictos de Git al realizar actualizaciones del material.

E.8.1 Opción A: sincronizar mediante SSH (recomendado)

Utilice este método si ya configuró sus llaves siguiendo los pasos de la Sección E.6.

> ❗ Url SSH del Repositorio para la Clase
>
> `git@github.com:alexyshr/ProgramacionSIG2026.git`

- **Vincular el repositorio remoto**: Si ya inicializó su carpeta local (no la de sus contenedores Docker) con `git init`, ejecute el siguiente comando para establecer el origen:

```
git remote add origin git@github.com:alexyshr/ProgramacionSIG2026.git
```

- **Descargar el contenido (Pull)**: Para traer los archivos del servidor a su computadora por primera vez:

```
git pull origin main
```

- **Verificación de estado**: Confirme que la carpeta local está sincronizada correctamente:

```
git status
```

Resultado esperado: `On branch main. Your branch is up to date. working tree clean.`

E.8.2 Opción B: sincronizar mediante HTTPS

Utilice este método si aún no ha configurado llaves SSH o prefiere una descarga rápida. A diferencia del método anterior, este suele solicitar un **Personal Access Token (PAT)** (ver líneas arriba) para realizar operaciones de subida (*push*).

> **!** Url HTTPS del Repositorio para la Clase
>
> https://github.com/alexyshr/ProgramacionSIG2026.git

E.8.2.1 Opción B-1: Descarga rápida mediante `git clone`

Nota: Este procedimiento tiene la desventaja que el nombre de la carpeta es definido por el nombre del respositorio en GitHub, no por el usuario (usted).

- Este comando descarga el repositorio completo y crea automáticamente una subcarpeta con el nombre del proyecto:

```
git clone https://github.com/alexyshr/ProgramacionSIG2026.git
```

- Para chequear por actualizaciones en el repositorio remoto:

```
git remote -v
```

- Para descargar las actualizaciones a la carpeta local, es decir sincronizar el repositorio remoto con la carpeta local:

```
git pull
```

E.8.2.2 Opción B-2: Sincronizar repositorio GitHub con carpeta local existente

Utilice este procedimiento si usted ya creó una carpeta localmente (por ejemplo, `ProgramacionSIG2026`) y desea vincularla al repositorio oficial de GitHub para recibir actualizaciones sin perder sus archivos actuales.

Ejecute los siguientes comandos desde la terminal dentro de su carpeta de trabajo:

```
# 1. Iniciar el repositorio localmente
git init

# 2. Vincular la carpeta local con el servidor de GitHub
git remote add origin https://github.com/alexyshr/ProgramacionSIG2026.git

# 3. Asegurar que la rama principal se llame 'main'
git branch -M main

# 4. Sincronizar (traer los archivos del servidor)
# Usamos --allow-unrelated-histories para permitir la fusión de carpetas
# que no "nacieron" del mismo commit original.
git pull origin main --allow-unrelated-histories
```

! Manejo de conflictos en el primer pull

Al ejecutar el `git pull` en una carpeta existente, es posible que Git le pida resolver conflictos si usted tiene archivos con el mismo nombre que los del repositorio (ej. un `README.md` genérico).
Si desea forzar que sus archivos locales sean sobrescritos por los del repositorio oficial, puede usar: `git fetch --all` `git reset --hard origin/main`

Verificación de la conexión

Para confirmar que su carpeta quedó bien vinculada, ejecute: `git remote -v`
Debería ver la URL de su repositorio tanto para `fetch` como para `push`.

E.9 Notas de flujo de trabajo

- **Persistencia**: Una vez clonado o vinculado el repositorio, Git recordará la ruta del servidor (ya sea SSH o HTTPS). No es necesario repetir los comandos de `remote add` en sesiones futuras.
- **Seguridad**: Si opta por HTTPS, recuerde que GitHub ya no acepta contraseñas básicas por terminal; deberá generar un PAT (Personal Access Token) en la configuración de su cuenta. Por esta razón, se recomienda priorizar el uso de **SSH**.

Apéndice F

Visual Studio Code (VScode): Extensiones e Interfaz

VScode es un editor de código para programadores gratuito, de código abierto y multiplataforma (Microsoft Corporation, 2026). Lo usaremos para editar archivos `.qmd` y para tener acceso a los contenedores Docker instalados para el curso.

F.1 Instalación

Descargue e instale las versión oficial de **VSCode IDE** correspondiente al sistema operativo de su máquina local desde el siguiente enlace:

- **URL**: https://code.visualstudio.com/
- **Instalación**: Seleccione el instalador `.exe` más reciente y que coincida con la arquitectura de su máquina.

F.2 Acceso y configuración en VSCode

Ver Sección B.5.

F.3 Inicialización del visor gráfico

Ver Sección B.6.

F.4 Atajos

F.4.1 La paleta de comandos VSCode

La mayoría de las funciones avanzadas de VSCode no tienen un botón visible. Se acceden mediante la **Paleta de Comandos** presionando `Ctrl + Shift + P`. A continuación, los comandos más utilizados en este curso:

Comando (Command Palette)	Propósito	¿Cuándo usarlo?
`Developer: Reload Window`	Reinicia la interfaz de VSCode sin cerrar el contenedor.	Cuando una extensión (como la de R o Julia) deja de responder.
`Dev Containers: Rebuild Container`	Reconstruye la imagen desde el Dockerfile.	Cuando se realizan cambios en la configuración del entorno.
`Quarto: Preview`	Abre una ventana lateral con la previsualización del documento.	Para ver los cambios en el `.qmd` en tiempo real.
`R: Create R terminal`	Fuerza la apertura de una nueva consola de R vinculada.	Si el prompt `>` no aparece automáticamente.
`Julia: Start REPL`	Inicia la consola interactiva de Julia.	Al abrir el contenedor por primera vez para ejecutar código Julia.
`Python: Select Interpreter`	Permite elegir la versión de Python del contenedor.	Si VSCode no detecta automáticamente `/usr/bin/python3`.
`PostgreSQL: New Query`	Abre un editor SQL en blanco.	Para realizar consultas a la base de datos espacial.
`Git: Pull` / `Git: Push`	Sincroniza cambios con GitHub.	Para actualizar el repositorio de la clase o subir tareas.

! El comando de "Primer Auxilio"

Si notas que el autocompletado desaparece o que los gráficos no cargan en **httpgd**, el primer paso siempre es ejecutar `Developer: Reload Window`. Esto refresca todas las conexiones de las extensiones con el contenedor sin interrumpir los procesos que Docker está ejecutando de fondo.

F.4.2 Atajos de teclado rápidos VSCode

Además de la paleta, memorice estos tres para su flujo diario: * `Ctrl + Shift +` (acento grave): Abre una nueva Terminal. * `Ctrl + J`: Mostrar la terminal/consola existente. * `Ctrl + Enter`: Envía la línea de código actual desde el editor al REPL (R, Julia o Python). * `Ctrl + Shift + V`: Previsualiza archivos Markdown (.md) de forma rápida.

F.4.3 Atajos de teclado JupyterLab (notebooks)

Acción	Atajo (Esc Mode)
Ejecutar Celda	`Ctrl + Enter`
Ejecutar y pasar a la siguiente	`Shift + Enter`
Convertir a Markdown	`M`
Convertir a Código	`Y`
Crear celda arriba / abajo	`A` / `B`

F.5 Instalación Tinytex

TinyTeX es un motor ligero de LaTeX necesario para compilar reportes profesionales (ej. en Quarto) en formato PDF (Xie, 2019). Si desea instalar TinyTeX para generar PDF por fuera del entorno Docker, desde la terminal de VSCode use el comando:

```
quarto install tinytex
```

La herramienta `Tinytex` viene instalada dentro del contenedor `contenedor_sig_unal`, así que no necesita instalarla.

F.6 Instalación de extensiones

Para que el entorno de desarrollo sea plenamente funcional, es necesario instalar un conjunto de extensiones específicas. Inicialmente todas las extensiones se pueden instalar para el entorno local (ej: Windows), sin embargo esas mismas extensiones deben estar instaladas también dentro del contenedor `contenedor_sig_unal`.

Una vez que haya realizado el proceso de **"Dev Containers: Attach to Running Container ..."** (`Ctrl + Shift + P` en VSCode), asegúrese de que estas herramientas estén instaladas **dentro del contenedor** (busque el botón azul *Install in ...* en la pestaña de extensiones).

Tabla F.3: Extensiones de VSCode para el entorno de Geomática y Programación SIG

Categoría	Herramienta	Propósito	ID de la Extensión
Infraestructura	**Dev Containers**	Conexión y gestión del contenedor activo	`ms-vscode-remote.remote-containers`
	Docker / Container Tools	Control de imágenes y recursos del contenedor	`ms-azuretools.vscode-docker`
Ecosistema R	**R (REditorSupport)**	Soporte de sintaxis, LSP y terminal de R	`REditorSupport.r`
	R Debugger & Tools	Depuración de scripts y herramientas de desarrollo	`RDebugger.r-debugger`
	R Extension Pack	Pack de utilidades (por Yuki Ueda)	`yukiueda.r-extension-pack`
Ecosistema Python	**Python (Microsoft)**	IntelliSense (Pylance) y depuración avanzada	`ms-python.python`
	Python Debugger	Motor de depuración específico para Python	`ms-python.debugpy`
	Jupyter (Microsoft)	Soporte para Notebooks e interactividad nativa	`ms-toolsai.jupyter`

Categoría	Herramienta	Propósito	ID de la Extensión
Ecosistema Julia	**Julia**	Soporte integral para ejecución y gráficos	`julialang.language-julia`
Base de Datos	**PostgreSQL**	Cliente para explorar la DB (Chris Kolkman)	`ckolkman.vscode-postgres`
	psql	Herramienta de consulta SQL (doublefint)	`doublefint.psql`
Publicación	**Quarto**	Renderizado y preview de archivos `.qmd`	`quarto.quarto`
Versión	**GitHub Repositories**	Manejo de archivos remotos sin clonar todo	`github.remotehub`
Utilidad	**Live Server**	Servidor local con recarga automática para previsualizar HTML	`ritwickdey.liveserver`
	Code Spell Checker	Corrector ortográfico para código y comentarios	`streetsidesoftware.code-spell-checker`
	Spanish - Code Spell Checker	Diccionario en español para el corrector ortográfico	`streetsidesoftware.code-spell-checker-spanish`

F.6.1 Notas sobre herramientas específicas

- **PostgreSQL (Chris Kolkman)**: Esta extensión le permitirá conectarse a la base de datos espacial del contenedor sin salir de VSCode. Podrá ejecutar queries SQL y visualizar tablas directamente.
- **Jupyter Ecosystem**: Incluimos el *Jupyter Keymap* para aquellos acostumbrados a los atajos de teclado de Jupyter Notebooks. Esto facilita la transición al trabajar con kernels de Python dentro de Quarto.
- **Sobre `psql`**: La herramienta de línea de comandos `psql` ya viene preinstalada **dentro del sistema operativo del contenedor** (PostgreSQL Client). No requiere instalación adicional para funcionar en la terminal del contenedor. Sin embargo, la instalación de `psql` y dentro de VSCode garantiza soporte adicional, ver Sección F.7 líneas abajo.
- **Live Server (Ritwick Dey)**: Esta extensión le permite lanzar un servidor de desarrollo local con un solo clic. Es extremadamente útil para previsualizar exportaciones de mapas interactivos (**Leaflet** u **OpenLayers**) o sitios web estáticos generados por **Quarto**. La ventaja principal es la "recarga en vivo" (*live reload*): cada vez que guarde un cambio en su archivo HTML o CSS, el navegador se actualizará automáticamente sin necesidad de recargar la página manualmente.
- **Code Spell Checker**: Esta extensión es esencial para evitar errores tipográficos en el código y, sobre todo, en la documentación de archivos `.qmd`. Para que reconozca términos en nuestro idioma, se recomienda instalar también el paquete **Spanish - Code Spell Checker** (`streetsidesoftware.code-spell-checker-spanish`).

! Instalación Local vs. Remota

Recuerde que extensiones como **Dev Containers** y **Docker** deben estar instaladas en su VSCode **local** (en Windows), mientras que las de análisis (R, Python, Julia, PostgreSQL) deben estar instaladas **dentro de la sesión del contenedor** para acceder a los compiladores y librerías de la imagen.

i Soporte del Lenguaje en R

Para que el autocompletado y la documentación en tiempo real funcionen, el contenedor requiere el paquete `languageserver`. Aunque la imagen ya lo incluye por defecto, si nota que el resaltado de sintaxis falla, puede reinstalarlo ejecutando: `install.packages("languageserver")` en su consola de R.

¿Cuándo usar Live Server?

Úselo cuando necesite probar cómo se comporta un mapa interactivo exportado fuera del entorno de **Quarto**. Simplemente haga clic derecho sobre su archivo `.html` en el explorador de **VSCode** y seleccione **"Open with Live Server"**. La opción de búsqueda en la página del curso (archivo `_site\index.html`) solo funcionará si la ejecuta dentro de **Live Server**.

F.6.1.1 Configuración del corrector ortográfico

Para habilitar corrección ortográfica en idioma español:

- Active el idioma: Presione `Ctrl + Shift + P`, escriba "Spell", seleccione `Enable Spanish Spell Checker Dictionary` y finalmente seleccione `User` de la lista.

Para que el corrector funcione simultáneamente con términos técnicos en inglés y redacción en español, puede añadir la siguiente configuración en su archivo `settings.json` de **VSCode**:

```
{
    "cSpell.language": "en,es",
    "cSpell.enabled": true,
    "cSpell.spellCheckDelayMs": 500,
    "cSpell.diagnosticLevel": "Information"
}
```

Diccionario técnico

A medida que trabaje con librerías como `geopandas` o `ArchGDAL`, es posible que el corrector marque estas palabras como errores. Puede agregarlas a su "Diccionario de Usuario" haciendo clic derecho sobre la palabra y seleccionando **Add to User Dictionary**.

F.7 Sinergia de las extensiones de PostgreSQL en VSCode

Contar con ambas extensiones instaladas dentro del contenedor no es redundancia, sino una estrategia para optimizar el flujo de trabajo con datos espaciales. Cada una ofrece una "capa" de interacción distinta que, al

combinarse, potencia la productividad.

Extensión	Fortaleza técnica	Utilidad principal
PostgreSQL (Chris Kolkman)	Exploración de objetos	Navegar visualmente por esquemas, ver la lista de tablas y verificar columnas de **PostGIS**.
psql (doublefint)	Ejecución de consultas	Ejecutar bloques de código SQL directamente desde scripts `.sql` con atajos de teclado rápidos.

F.7.1 Ventajas de la instalación dual

- **Validación visual inmediata**: Mientras escribe una consulta compleja en el editor (usando la extensión de **doublefint**), puede consultar el panel lateral de **Chris Kolkman** para verificar nombres de columnas o tipos de datos geográficos sin tener que hacer un `SELECT *`.
- **Soporte de sintaxis robusto**: La combinación garantiza que **VSCode** reconozca perfectamente las palabras clave de **SQL** y proporcione sugerencias automáticas (IntelliSense) precisas.
- **Resultados en pestañas independientes**: Permite mantener los resultados de una consulta abiertos en una pestaña mientras se sigue editando el código en otra, algo fundamental cuando se comparan estadísticas de diferentes capas ráster o vectoriales.

> **Recomendación de flujo**
>
> 1. Use el panel de **Chris Kolkman** para conectarse y "ver" qué hay en la base de datos.
> 2. Escriba sus scripts de análisis espacial en archivos `.sql` y ejecútelos usando la extensión de **doublefint** para obtener una respuesta rápida en la consola.

> **Atajo de teclado (Shortcut)**
>
> Para la mayoría de estas extensiones, puede resaltar una consulta SQL y presionar `Ctrl + E` (o `Cmd + E` en Mac) para ejecutarla inmediatamente y ver los resultados en una tabla formateada.

F.8 Verificación de conectividad

Una vez instaladas las extensiones, verifique que su entorno esté correctamente configurado siguiendo estos puntos:

- **Estado del Contenedor**: En la esquina inferior izquierda de VSCode, debe aparecer un recuadro azul con el texto: **Dev Container: contenedor_sig_unal**.
- **Activación de Kernels**: Al abrir un archivo `.qmd`, en la esquina superior derecha debe poder seleccionar el kernel deseado (R, Python 3 o Julia).
- **Terminal Unificada**: Al abrir una nueva terminal (`Ctrl + Shift + `), la línea de comandos debe estar identificada como `rstudio@contenedor_sig_unal:/home/rstudio/work$`.

Sincronización Automática

Recuerde que cualquier cambio que realice en la configuración de estas extensiones mientras esté conectado al contenedor se guardará en su perfil local de VSCode, facilitando las conexiones futuras.

Apéndice G

Temas técnicos sobre Windows

G.1 Herramientas de administración

Acceso Directo	Parámetro / Comando	Descripción
Windows + R	`%UserProfile%`	Abre tu carpeta personal de usuario (donde se crea el archivo `.wslconfig`).
Windows + R	`cleanmgr`	Abre el **Liberador de espacio en disco** para purgar basura del sistema.
Windows + R	`taskmgr`	[**Ctrl + Shift + Esc**] Abre el **Administrador de Tareas** para ver procesos.
Windows + R	`resmon`	Abre el **Monitor de Recursos** (ideal para ver el uso real de RAM y Swap).
Windows + R	`sysdm.cpl`	[**Win + Pausa**] Propiedades del sistema (configuración de Memoria Virtual).
Windows + R	`%temp%`	Abre la carpeta de **Archivos temporales del usuario** actual.
Windows + R	`temp`	Abre la carpeta de **Archivos temporales del sistema** (raíz).
Windows + R	`prefetch`	Carpeta de precarga de Windows (caché que puedes vaciar para liberar espacio).
Windows + R	`diskmgmt.msc`	Abre la **Administración de discos** (para vigilar tus unidades C: y D:).

Acceso Directo	Parámetro / Comando	Descripción
Windows + R	`appdata`	Acceso a carpetas de configuración de aplicaciones (Roaming/Local).
Windows + R	`cmd`	**[Win + X, luego C]** Abre el Símbolo del Sistema (Terminal clásica).
Windows + R	`powershell`	**[Win + X, luego A]** Abre PowerShell con permisos de administrador.
Windows + R	`control`	Abre el **Panel de Control clásico** de Windows.
Windows + R	`msconfig`	Configuración del sistema (gestión de servicios y arranque).
Windows + R	`ncpa.cpl`	Panel de **Conexiones de red** (configuración de adaptadores e IPs).
Nativo	**Win + E**	Abre instantáneamente el **Explorador de Archivos**.
Nativo	**Win + I**	Abre el panel de **Configuración de Windows** (Settings).
Nativo	**Win + D**	**Minimiza/Restaura todo** para ir al Escritorio rápidamente.
Nativo	**Win + V**	Abre el **Historial del Portapapeles** (muy útil para copiar varios códigos).

- Procesos de **WSL2** (Windows Subsystem for Linux 2): Docker y Vmmem

G.2 Comandos (PowerShell)

- **Ver uso de RAM:** `Get-Process | Sort-Object WorkingSet -Descending | Select-Object -First 10`
- **Ver estado de WSL:** `wsl --list --verbose`
- **Limpiar Cache de DNS:** `ipconfig /flushdns`

Apéndice H

Temas Técnicos sobre Linux

H.1 Comandos adicionales antes de trabajar con los contenedores

Ver Sección B.1.

H.2 Comandos comúnes de terminal

Al trabajar dentro de un contenedor Docker, la terminal es su línea directa con el sistema operativo (Ubuntu/Debian). Estos comandos le permitirán verificar que los motores geoespaciales y las librerías de sistema están correctamente instalados.

Comando	Propósito	¿Para qué sirve en el curso?
`dpkg -l | grep -E "libpng|libgdal|libproj|libgeos"`	Lista librerías instaladas	Verificar que los drivers de mapas (GDAL, PROJ, GEOS) están activos.
`gdalinfo --version`	Versión de GDAL	Confirmar que el motor de traducción de datos geográficos funciona.
`df -h`	Espacio en disco	Evitar errores de "Disk Full" al descargar imágenes satelitales pesadas.
`htop` (o `top`)	Monitor de recursos	Ver si un proceso de Julia o R está consumiendo toda la RAM o CPU.
`ls -la`	Listado detallado	Revisar permisos de archivos y carpetas (clave para errores de Git).
`psql --version`	Versión de Postgres	Confirmar que el cliente de base de datos está listo para usarse.

Comando	Propósito	¿Para qué sirve en el curso?
`which r` / `which julia`	Ruta del ejecutable	Localizar dónde están instalados los lenguajes para configurar el Path.
`fc-list | grep -i "Symbola"`	Buscar fuentes del sistema	Localizar la ruta exacta de fuentes necesarias para emojis y símbolos en el PDF.

H.3 Verificación de librerías geoespaciales

El comando `dpkg -l | grep -E "libpng|libgdal|libproj|libgeos"` es su principal herramienta de diagnóstico. Si al ejecutarlo no obtiene resultados, las funciones de mapeo en R (`sf`), Python (`geopandas`) y Julia (`ArchGDAL`) fallarán, ya que todas dependen de estos binarios del sistema.

> **Tip de Productividad**
>
> En la terminal de VSCode, puede usar la tecla **Tab** para autocompletar nombres de archivos o comandos, y las **flechas arriba/abajo** para navegar por el historial de comandos ejecutados.

H.3.1 ¿Cómo leer la salida de `dpkg`?

Al ejecutar el comando de verificación, verá una lista con el prefijo `ii`. Esto significa: * **i**: *Desired state* (Install) * **i**: *Current state* (Installed)

Si ve algo diferente a `ii`, el paquete tiene problemas de instalación.

H.4 La cirugía de librerías: el "cambiazo" de OpenSSL

Al trabajar con contenedores de última generación, a veces nos encontramos con un choque de versiones entre lo que el sistema operativo ofrece y lo que los lenguajes de programación esperan. En nuestro caso, realizamos una "cirugía" técnica utilizando enlaces simbólicos (`ln -sf`).

H.4.1 El conflicto: ¿3.0.x o 3.3.x?

La respuesta corta es: **Físicamente tiene instalada la versión 3.0.x, pero le mentimos al software diciéndole que es la 3.3.x.**

- **La Realidad (Sistema)**: Su contenedor corre sobre **Ubuntu 24.04 (Noble Numbat)**. Esta versión viene de fábrica con **OpenSSL 3.0.x**, que es la versión estable y oficial del sistema.
- **La Expectativa (Julia)**: Algunos paquetes potentes de Julia (como los binarios JLL de `GDAL` o `NCDatasets`) fueron compilados en entornos más recientes que ya usaban **OpenSSL 3.3.x**.

Cuando Julia intenta arrancar, busca una "etiqueta" interna llamada `OPENSSL_3.3.0`. Al no encontrarla en la librería de Ubuntu, el sistema lanza un error y bloquea la carga de mapas.

H.4.2 La solución: ¿qué hace exactamente `ln -sf`?

Imagine que Julia es un guardia de seguridad que busca una llave maestra etiquetada como **"Llave 3.3"**. Al revisar la caja fuerte (el sistema), solo ve la **"Llave 3.0"** y se niega a abrir. Lo que hicimos fue un "cambiazo" estratégico.

Ejecutamos este comando de "cirugía":

```
find /root/.julia/artifacts -name "libssl.so*" -exec ln -sf /usr/lib/x86_64-linux-gnu/libssl.so.3 {} \;
```

Desglose de la operación: 1. `find ... -name "libssl.so*"`: Localiza todos los archivos de librería que Julia descargó por su cuenta (sus propios binarios internos). 2. `-exec ln -sf ... {}`: Por cada archivo encontrado, borra el original (la `-f` de *force*) y crea un **enlace simbólico** (la `-s` de *symbolic*). 3. **El Destino**: El enlace apunta directamente a la librería real de Ubuntu (`/usr/lib/.../libssl.so.3`).

H.4.3 ¿por qué funciona si las versiones son distintas?

Funciona gracias a que **OpenSSL 3** mantiene algo llamado **compatibilidad binaria** entre sus versiones menores. Las funciones internas se llaman exactamente igual; lo único que cambió fue el "nombre del archivo" que Julia buscaba.

Al redirigir el nombre, Julia utiliza la librería de Ubuntu pensando que es la suya, y como las instrucciones internas son compatibles, el sistema arranca sin errores y con total estabilidad.

> ⚠ Importancia para el SIG
>
> Sin esta cirugía, paquetes como `ArchGDAL` en Julia no podrían cargar los controladores para leer archivos `.tif` o conectarse a bases de datos espaciales, ya que la comunicación segura (SSL) fallaría al inicio.

H.4.4 La "factura" de la cirugía: el adiós a `JuliaCall`

Como en toda operación de emergencia, el "cambiazo" de librerías tuvo un efecto secundario importante: la ruptura definitiva de la compatibilidad con `JuliaCall`, el puente más común entre R y Julia.

- **El Conflicto**: `JuliaCall` intenta embeber a Julia directamente dentro del proceso de R (comparten la misma memoria). Al detectar que las librerías de OpenSSL han sido redirigidas mediante enlaces simbólicos, el proceso de R entra en un conflicto de seguridad y colapsa inmediatamente (*Segmentation Fault*), impidiendo cualquier comunicación.
- **La Solución**: Para no renunciar a la integración de ambos lenguajes, cambiamos de estrategia y adoptamos `JuliaConnectoR`. A diferencia del método anterior, este paquete trata a Julia como un servicio independiente que se comunica mediante *sockets*. Esto lo hace inmune al conflicto de OpenSSL, ya que cada lenguaje corre en su propia parcela de memoria.

H.4.5 El nacimiento de `j_eval` y `j_plot`

Para que este cambio de arquitectura fuera invisible para el usuario final, se desarrollaron dos funciones "envolventes" (*wrappers*) que automatizan la traducción entre lenguajes:

1. `j_eval()`: Toma comandos escritos en R, los envía al motor de Julia de forma segura y captura el resultado (numérico o textual) para devolverlo a la consola de R. Es el cerebro detrás de nuestros benchmarks.
2. `j_plot()`: Resuelve el problema del renderizado. Julia genera la visualización geoespacial "tras bambalinas", y esta función captura la salida gráfica para insertarla automáticamente en el reporte de Quarto.

Conclusión Técnica

Gracias a esta combinación estratégica (**Cirugía OpenSSL** + **JuliaConnectoR** + **Funciones j_**), logramos lo que parecía imposible en un entorno Docker restrictivo: un ecosistema donde **R, Python y Julia conviven en paz**, permitiéndonos procesar datos masivos sin colapsos de memoria ni errores de librerías.

Apéndice I

Adicionales Sobre Python

I.1 Stack instalado de Python

Para el análisis geoespacial avanzado, en los contenedores se instalaron un conjunto de librerías especializadas. El núcleo de este entorno es **`geopandas`**, cuya instalación gestiona automáticamente dependencias críticas como **`numpy`** (cálculo numérico), **`pandas`** (manipulación de datos) y **`shapely`** (operaciones geométricas). Complementariamente, se instaló **`rasterio`** para la gestión profesional de datos ráster y **`rasterstats`** para la extracción de estadísticas zonales.

Tabla I.1: Paquetes principales de Python

Paquete	Funcionalidad	Sitio web
`numpy`	Arreglos	https://numpy.org/
`pandas`	Tablas	https://pandas.pydata.org/
`shapely`	Geometrías vectoriales	https://shapely.readthedocs.io/
`geopandas`	Capas vectoriales	https://geopandas.org/
`rasterio`	Ráster	https://rasterio.readthedocs.io/
`rasterstats`	Estadísticas zonales	https://github.com/perrygeo/python-rasterstats
`psycopg2`	Interfaz a PostgreSQL	https://www.psycopg.org/docs/

Como veremos, estos paquetes dependen unos de otros. Las dependencias principales se muestran en la Figura I.1:

- **`geopandas`**: Capa superior que integra a `pandas`, `shapely`, `fiona` y `pyproj`. Permite realizar consultas espaciales complejas y gestionar GeoDataFrames con una sintaxis simplificada.
- **`rasterio`**: Basada en la librería **GDAL**, es el estándar para la lectura, escritura y manipulación de formatos ráster (como GeoTIFF), permitiendo gestionar metadatos y arreglos de píxeles con alta eficiencia.
- **`scipy`**: Proporciona algoritmos avanzados de optimización, álgebra lineal y estadística necesarios para procesos de interpolación y análisis de superficies.

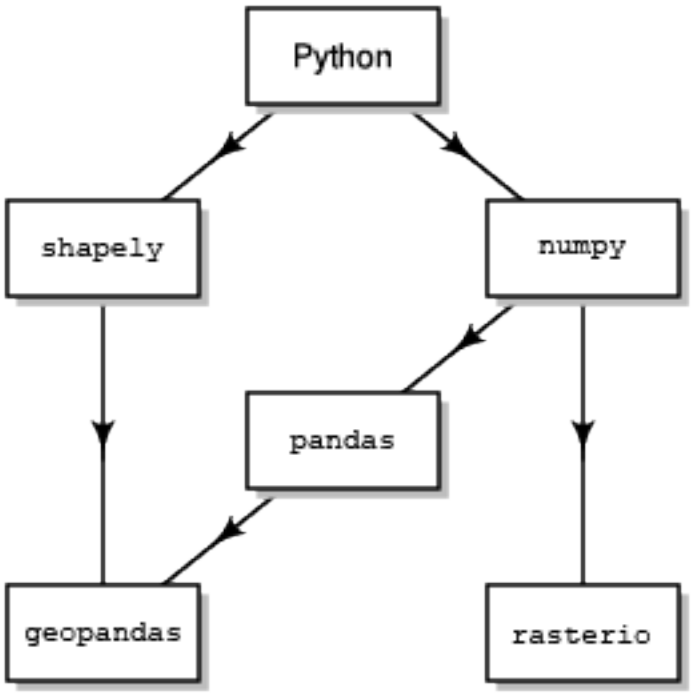


Figura I.1: Principales dependencias entre los paquetes de Python que vamos a estudiar. Adaptado de Dorman (2025).

- `rasterstats`: Herramienta específica para extraer estadísticas (medias, sumas) a partir de capas ráster basadas en geometrías vectoriales.
- `shapely`: Motor geométrico basado en el estándar *Simple Features* y puente hacia la librería **GEOS**. Se encarga de la lógica matemática de las geometrías (áreas, intersecciones, buffers y validación topográfica).
- `pyproj`: Interfaz para la librería **PROJ**. Gestiona proyecciones cartográficas, sistemas de referencia (CRS) y transformaciones de datums.
- `fiona / pyogrio`: Motores de acceso a datos vectoriales (SHP, GPKG) que actúan como interfaces hacia la librería **GDAL/OGR**.
- `xyzservices`: Repositorio de metadatos para conectar con servicios de mapas base dinámicos (como OpenStreetMap).

La siguiente tabla vincula las librerías de Python con los motores de cálculo de bajo nivel (C/C++) integrados en el contenedor Docker:

Librería Python	Motor de Sistema (C/C++)	Función Principal en Geomática
`pyproj`	**PROJ**	Gestión de Proyecciones y Sistemas de Referencia (CRS).
`shapely`	**GEOS**	Álgebra topológica (Intersecciones, Buffers, Áreas).
`fiona / pyogrio`	**GDAL/OGR**	Lectura y escritura de formatos vectoriales.
`rasterio`	**GDAL**	Manejo de datos ráster y metadatos de imágenes.

Librería Python	Motor de Sistema (C/C++)	Función Principal en Geomática
geopandas	*(Integra los 3 motores)*	Gestión integral de GeoDataFrames espaciales.

I.2 Versiones instaladas del software

Para verificar las versiones inSstaladas, puede ejecutar `pip3 list` en la terminal. Las versiones instaladas deben coincidir con la siguiente tabla:

Software	Tipo	Versión	Comando (desde terminal)	Comando (desde Python)
`Python`	Lenguaje	3.12.3	`python3 --version`	`import sys; sys.version`
`pip`	Gestor de paquetes	24.0	`pip3 --version`	`import pip; pip.__version__`
`numpy`	Paquete	2.4.1	`pip3 show numpy`	`import numpy; numpy.__version__`
`pandas`	Paquete	2.3.3	`pip3 show pandas`	`import pandas; pandas.__version__`
`geopandas`	Paquete	1.1.2	`pip3 show geopandas`	`import geopandas; geopandas.__version__`
`shapely`	Paquete	2.1.2	`pip3 show shapely`	`import shapely; shapely.__version__`
`pyproj`	Paquete	3.7.2	`pip3 show pyproj`	`import pyproj; pyproj.__version__`
`fiona`	Paquete	1.10.1	`pip3 show fiona`	`import fiona; fiona.__version__`
`rasterio`	Paquete	1.5.0	`pip3 show rasterio`	`import rasterio; rasterio.__version__`

Apéndice J

Adicionales Sobre R

J.1 Stack de R

Para el análisis geoespacial en `R` (instalado dentro de los contenedores), utilizaremos un ecosistema robusto basado en el estándar *Simple Features* y motores de alto rendimiento para datos ráster (rejilla - cuadrículas).

Tabla J.1: Paquetes principales del stack espacial en R

Paquete	Funcionalidad	Sitio web
`sf`	Geometrías vectoriales (Simple Features)	https://r-spatial.github.io/sf/
`terra`	Análisis de datos espaciales (Ráster/Vector)	https://rspatial.github.io/terra/
`stars`	Arreglos ordenados espacio-temporales (DataCubes)	https://r-spatial.github.io/stars/
`tidyverse`	Metapaquete para ciencia de datos	https://www.tidyverse.org/
`tidyterra`	Métodos de tidyverse para terra	https://dieghernan.github.io/tidyterra/

- **`sf` (Simple Features)**: Es el estándar moderno para el manejo de datos vectoriales. Representa las geometrías como una columna especial en un *data frame*, permitiendo aplicar toda la potencia de manipulación de tablas a objetos espaciales.
- **`terra`**: Motor de alta eficiencia que reemplaza al antiguo paquete `raster`. Está optimizado para el manejo de grandes volúmenes de datos mediante objetos `SpatRaster` y `SpatVector`, permitiendo operaciones de álgebra de mapas y análisis local de manera veloz.
- **`stars` (Spatiotemporal Arrays)**: Especializada en el manejo de "cubos de datos" (rejillas con dimensiones de espacio, tiempo y múltiples atributos). Es la herramienta ideal para procesar series temporales de imágenes satelitales o modelos climáticos multidimensionales.

- **tidyterra**: Extiende la gramática de ggplot2 y tidyverse hacia los objetos de terra. Permite visualizar mapas ráster de forma elegante y realizar tuberías (pipes) de datos manteniendo la integridad espacial.

J.2 Versiones instaladas del software

Versiones que deben quedar instaladas en el sistema R:

Software	Tipo	Versión	Comando (desde terminal)	Comando (desde R)
R	Lenguaje	4.5.2	R --version	R.version.string
GEOS	Librería sistema	3.12.1	geos-config --version	sf::sf_extSoftVersion()["GEOS"]
GDAL	Librería sistema	3.8.4	gdalinfo --version	sf::sf_extSoftVersion()["GDAL"]
PROJ	Librería sistema	9.4.0	(no aplica / no disponible)	sf::sf_extSoftVersion()["PROJ"]
sf	Paquete	1.0.23	R -q -e "packageVersion('sf')"	packageVersion("sf")
terra	Paquete	1.8.86	R -q -e "packageVersion('terra')"	packageVersion("terra")
stars	Paquete	0.7.0	R -q -e "packageVersion('stars')"	packageVersion("stars")

J.3 Visualización interactiva con httpgd

Dado que nuestro entorno Docker es "headless" (sin monitor propio), utilizamos **httpgd** como un servidor gráfico intermedio. Este paquete captura las señales de dibujo de R y las sirve a través de una URL que VSCode puede renderizar.

J.3.1 Inicio del servidor gráfico

Cada vez que inicie una nueva sesión de R, debe despertar al servidor manualmente:

```
# Inicia el motor gráfico
httpgd::hgd()

# Verifica el estado y el puerto asignado
httpgd::hgd_details()
```

J.3.2 Resolución de conflictos (el "fix" del puerto)

En nuestro contenedor, el puerto estándar para gráficos está mapeado al **8787**. Si al ejecutar hgd_details() nota que el sistema asignó un puerto aleatorio o si el visor aparece en blanco, fuerce la conexión con la siguiente "receta":

```r
# Forzar host y puerto para compatibilidad con Docker
httpgd::hgd(host = "0.0.0.0", port = 8787, token = FALSE)
```

J.3.3 Formas de ver sus gráficos

Una vez el servidor está corriendo, tiene dos caminos para visualizar sus mapas:

Método	Comando / Acción	Ventaja
Visor Interno	`Ctrl + Shift + P` -> **Open httpgd viewer**	Mantiene todo dentro de una pestaña de VSCode.
Navegador Externo	`httpgd::hgd_browse()`	Ideal si trabaja con dos monitores para ver mapas en pantalla completa.

> ¿Por qué `token = FALSE`?
>
> Dentro del entorno seguro del contenedor, desactivamos el *token* para facilitar la conexión inmediata entre el servidor de R y el visor de VSCode sin que el firewall interno bloquee la petición por falta de credenciales.

J.3.4 Verificación de salud

Si después de iniciar el servidor intenta graficar algo (ej: `plot(1:10)`) y no ve nada, ejecute `httpgd::hgd_details()`. Asegúrese de que: 1. **URL**: Apunte a `http://0.0.0.0:8787`. 2. **Status**: Indique que está escuchando (*listening*).

J.3.5 El Entorno interactivo (REPL) y motores gráficos

En el desarrollo científico, el **REPL** (*Read-Eval-Print Loop*) es la consola interactiva que permite la "programación exploratoria". Es el ciclo donde el sistema **lee** su instrucción, la **evalúa** mediante el motor correspondiente, **imprime** el resultado y queda en un **bucle** a la espera del siguiente comando.

En nuestro laboratorio políglota, la identidad visual del REPL es su brújula para saber en qué lenguaje está operando:

Lenguaje	¿Cómo se ve el REPL?	Características en este curso
R	`>`	Consola estándar para procesos geoespaciales con `sf` o `terra`.
Julia	`julia>`	Entorno de alto rendimiento con modos de ayuda (`?`) y terminal (`;`).

Lenguaje	¿Cómo se ve el REPL?	Características en este curso
Python	`>>>`	Consola básica; se recomienda el uso de **IPython** para interactividad.

La experiencia interactiva y el soporte gráfico dentro de **VSCode** dependen de la combinación entre el motor de ejecución y el dispositivo gráfico configurado en el contenedor:

Lenguaje	Acceso / Contexto	Motor de Ejecución	Dispositivo Gráfico	Soporte Inline
R	Nativo (Estático)	R Extension	VSCode Plots (Pane)	✔
R	Nativo (Interactivo)	R Extension	**httpgd**	✔
Julia	Nativo (REPL)	VSCodeServer	VSCode Plots (Pane)	✔
Julia	Interop (R-Bridge)	`JuliaConnectoR`	**httpgd** (vía R)	✔
Python	Interop (R-Bridge)	`reticulate`	R Device (**httpgd**)	✖ /
Python	Nativo (Jupyter)	Jupyter kernel	Jupyter Viewer	✔

J.3.6 El rol crítico de `httpgd` en R

Para R, no basta con tener la extensión instalada. El paquete **`httpgd`** actúa como un servidor web interno que captura los gráficos generados en el contenedor y los "proyecta" en VSCode.

- **¿Por qué lo usamos?** Los dispositivos gráficos tradicionales (como `X11` o `windows()`) no funcionan dentro de un contenedor Docker (*headless*). `httpgd` soluciona esto convirtiendo cada mapa en un elemento interactivo SVG/HTML renderizable.
- **Activación**: Siempre debe iniciar su sesión de R ejecutando `httpgd::hgd()` para abrir este canal de comunicación.

J.4 Paquetes para comunicación con Julia y Python

Para que este ecosistema funcione, R actúa como el "director de orquesta" utilizando un motor de ejecución y una serie de puentes que permiten la comunicación fluida de datos entre lenguajes.

Lenguaje / Función	Paquete en R	Motor / Propósito	Estado
Renderizado	`knitr`	Motor maestro que orquesta la ejecución de celdas en el documento	✔
Julia	`JuliaConnectoR`	Comunicación funcional estable con el kernel de Julia	✔
Python	`reticulate`	Interoperabilidad de objetos y ejecución de Python	✔
LSP (Soporte)	`languageserver`	Autocompletado, IntelliSense y ayuda en tiempo real	✔

J.4.1 El rol de `knitr`

Mientras que los paquetes actúan como puentes técnicos, **`knitr`** es el encargado de leer el archivo `.qmd`, enviar las instrucciones a los motores respectivos y capturar los resultados (tablas, mapas, gráficos) para integrarlos en el documento final. Sin `knitr`, la integración políglota de Quarto no sería posible.

J.4.2 Transición técnica: De `JuliaCall` a `JuliaConnectoR`

En el diseño de este entorno, se tomó la decisión técnica de descartar el paquete `JuliaCall` en favor de `JuliaConnectoR`:

1. **El Problema**: `JuliaCall` presentaba inestabilidades al gestionar dependencias dinámicas compartidas (como `libcurl` o `libstdc++`) dentro del contenedor, causando cierres inesperados (*crashes*) de la sesión de R.
2. **La Sustitución**: Se seleccionó **`JuliaConnectoR`** por su arquitectura más limpia, que no interfiere con el entorno de Julia ya configurado en el sistema operativo.
3. **La Solución**: La integración se resolvió mediante una "inyección" de código en el archivo `Rprofile.site`, configurando automáticamente la ruta del ejecutable en `$Sys.BINDIR$` y definiendo las funciones maestras `j_eval()` y `j_plot()`.

Regla de Oro del Curso

1. **Julia y R**: Se operan de manera totalmente **interactiva** en VSCode (envío de líneas con `Ctrl+Enter`).
2. **Python vía reticulate**: Se utiliza primordialmente para el **renderizado (vía knitr)** de informes finales. Para una programación interactiva pura en Python, se recomienda el uso de los **Jupyter kernels**.

Apéndice K

Adicionales Sobre Julia

K.1 Stack de Julia

Julia ofrece un rendimiento de nivel C++ con sintaxis simplificada. A continuación las principales librerías instaladas en los contenedores para el geoprocesamiento.

Tabla K.1: Paquetes principales del stack en Julia

Paquete	Funcionalidad	Sitio web
`LibPQ.jl`	Controlador PostgreSQL/PostGIS	https://juliadatabases.org/LibPQ.jl/dev/
`ArchGDAL.jl`	Interfaz de alto nivel para GDAL	https://yeesian.com/ArchGDAL.jl/
`LibGEOS.jl`	Interfaz del motor geométrico (GEOS)	https://github.com/JuliaGeo/LibGEOS.jl
`Plots.jl`	Visualización y generación de gráficos	https://docs.juliaplots.org/
`DataFrames.jl`	Manipulación de datos en memoria	https://dataframes.juliadata.org/

- **`LibPQ.jl`**: Conector de bajo nivel para **PostgreSQL**. Es la herramienta que permitirá a los estudiantes realizar ingesta y consulta de datos desde el servidor PostGIS.
- **`ArchGDAL.jl`**: Proporciona una abstracción de alto nivel para el motor GDAL. Es excelente para transformar formatos y manejar proyecciones.
- **`LibGEOS.jl`**: La interfaz directa al motor de geometrías (GEOS). Se utiliza para operaciones topológicas puras como validación de polígonos, intersecciones y cálculos de buffers.
- **`DataFrames.jl`**: El estándar para el manejo de datos tabulares en Julia, permitiendo integrar los resultados de las consultas espaciales en estructuras fáciles de analizar.

K.2 Versiones instaladas del software

Versiones de paquetes `Julia` en el sistema:

Software	Tipo	Versión	Comando (desde terminal)	Comando (desde Julia)
`Julia`	Lenguaje	1.10.4	`julia --version`	`VERSION`
`LibPQ`	Paquete	1.18.0	`julia -e "using LibPQ; println(pkgversion(LibPQ))"`	`using LibPQ; pkgversion(LibPQ)`
`ArchGDAL`	Paquete	0.10.11	`julia -e "using ArchGDAL; println(pkgversion(ArchGDAL))"`	`using ArchGDAL; pkgversion(ArchGDAL)`
`LibGEOS`	Paquete	0.9.7	`julia -e "using LibGEOS; println(pkgversion(LibGEOS))"`	`using LibGEOS; pkgversion(LibGEOS)`
`Plots`	Paquete	1.41.4	`julia -e "using Plots; println(pkgversion(Plots))"`	`using Plots; pkgversion(Plots)`
`DataFrames`	Paquete	1.8.1	`julia -e "using DataFrames; println(pkgversion(DataFrames))"`	`using DataFrames; pkgversion(DataFrames)`

Apéndice L

Adicionales Sobre QGIS

L.1 Instalación

L.1.1 OSGeo4w

L.1.2 QGIS con PIXI

Apéndice M

Adicionales sobre ArcGIS Pro

M.1 Instalación

M.1.1

M.1.2

Apéndice N

Adicionales sobre ArcGIS Pro

N.1 Instalación

N.1.1

N.1.2

Apéndice O

Errata

Esta es la errata

Referencias Bibliográficas

Bezanson, J., Edelman, A., Karpinski, S., & Shah, V. B. (2017). Julia: A fresh approach to numerical computing. *SIAM review*, *59*(1), 65-98.

Dorman, M. (2025). *Spatial Data Programming with Python*. Ben-Gurion University of the Negev. https://geobgu.xyz/py-2025/

Inc., D. (2025). *Docker Desktop Documentation*. https://docs.docker.com

Law, M., & Collins, A. (2019). Getting to know ArcGIS PRO. *(No Title)*.

Lawhead, J. (2017). *QGIS python programming cookbook*. Packt Publishing Ltd.

Microsoft Corporation. (2026). *Visual Studio Code*. https://code.visualstudio.com/

Mitchell, T. (2015). *An Introduction to Open Source Geospatial Tools*.

Neteler, M., Bowman, M. H., Landa, M., & Metz, M. (2012). GRASS GIS: A multi-purpose open source GIS. *Environmental Modelling & Software*, *31*, 124-130.

Passy, P., & Théry, S. (2018). The use of SAGA GIS modules in QGIS. *QGIS and generic tools*, *1*, 107-149.

Pebesma, E., & Bivand, R. (2023). *Spatial Data Science: With Applications in R*. Chapman; Hall/CRC. https://doi.org/10.1201/9780429459016

Pebesma, E., & Bivand, R. (2025). *Spatial Data Science: With Applications in R and Python*. https://r-spatial.org/python/

Python Software Foundation. (2025). *Python 3 Documentation*. https://docs.python.org/3/

Python, W. (2021). Python. *Python releases for windows*, *24*. http://static.softwaresuggest.com.s3.amazonaws.com/ssguides/1604480172_Python_read%20(1).pdf

R Core Team. (2025). *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing. https://www.R-project.org/

Rosas-Chavoya, M., Gallardo-Salazar, J. L., López-Serrano, P. M., Alcántara-Concepción, P. C., & León-Miranda, A. K. (2022). QGIS a constantly growing free and open-source geospatial software contributing to scientific development. *Cuadernos de Investigación Geográfica*, *48*(1), 197-213.

Toms, S. (2015). *ArcPy and ArcGIS–geospatial analysis with Python*. Packt Publishing Ltd.

Wijayaningrum, V. N., Lestari, V. A., et al. (2022). Jupyter lab platform-based interactive learning. *2022 International Conference on Electrical and Information Technology (IEIT)*, 295-301.

Wu, Q. (2023a). PyQGIS cookbook in markdown and Jupyter notebook formats. En *GitHub repository*. https://github.com/opengeos/pyqgis-cookbook; GitHub.

Wu, Q. (2023b). QGIS Notebook Plugin: Integrate Jupyter Notebooks into QGIS. En *GitHub repository* [Software]. https://github.com/opengeos/qgis-notebook-plugin; GitHub.

Xie, Y. (2019). TinyTeX: A lightweight, cross-platform, and easy-to-maintain LaTeX distribution based on TeX Live. *TUGboat*, *40*(1), 30-32.