Programación SIG: Python, R y Julia

Alexys H. Rodríguez-Avellaneda Ph.D.

2026-02-03

Tabla de Contenidos

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:

> **Note**
>
> 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 VS Code. - QGIS - ArcGIS Pro - Conocimientos básicos de Sistemas de Información Geográfica (SIG) y manejo de datos vectoriales/raster.

Licencia

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*.

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 pre-configurado 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.

Part I

Instalación y Entornos

Chapter 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)** (Pre-configurados).	**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.

Chapter 2

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

2.1 Introducción al Entorno de Trabajo

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".

2.1.1 Preparación de Herramientas en Windows

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

1. **VSCode IDE**: Su interfaz principal para editar archivos `.qmd` y gestionar código (Microsoft Corporation, 2026). En el anexo de referencia técnica encontrará las principales extensiones (Section A.7.2)necesarias a instalar en VSCode.
2. **Docker Desktop**: Motor de virtualización que ejecutará nuestro laboratorio SIG (Inc., 2025). Durante la instalación, asegúrese de activar la opción **"Use WSL 2 instead of Hyper-V"**.

Siguiendo los pasos detallados a continuación para instalar dos contenedores Docker, en resumen ellos contendrán (no es necesario instalarlos manualmente):

1. **Python**, **R** y **Julia** con los principales paquetes/librerías para geoprocesamiento.
2. **Quarto**.
3. **TinyTeX**: 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 (opcional: fuera de clase):

```
quarto install tinytex
```

4. **PostgreSQL + PostGIS**

2.1.2 Enlaces de Descarga e Instalación

Descargue e instale las versiones oficiales desde los siguientes enlaces:

- **VSCode IDE**:
  - **URL**: https://code.visualstudio.com/
  - **Instalación**: Seleccione el instalador `.exe` más reciente para Windows.
- **Docker Desktop**:
  - **URL**: https://www.docker.com/products/docker-desktop/
  - **Instalación**: Ejecute el instalador y asegúrese de aceptar la actualización del kernel de WSL 2 si el sistema lo solicita.

2.2 Infraestructura con Docker y Docker Compose

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

2.2.1 El Manifiesto Técnico: `Dockerfile` y `docker-compose.yml`

> **i** Organización del Proyecto
>
> Para automatizar el despliegue de los dos contenedores del curso (`analisis-geo` y `db-postgis`), utilizaremos la orquestación de Docker. Mientras que el entorno de análisis se construye a medida sobre la imagen base `ghcr.io/osgeo/gdal:ubuntu-full-latest`, el servicio de base de datos utiliza la imagen especializada `kartoza/postgis`.
> Para este proceso, utilizaremos dos archivos clave: **docker-compose.yml** y **Dockerfile**.

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`.
2. Dentro de ella, guarde los archivos que se presentan a continuación.

2.2.1.1 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 Windows 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 se construye a medida (etiqueta `build: .`) utilizando las instrucciones del `Dockerfile`. 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`.

- **Servicio PostGIS (`db-postgis`)**: Identificado como `db-postgis`, este servicio descarga automáticamente 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`.

```yaml
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:
```

2.2.1.2 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.

Los pilares técnicos de este archivo 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:
# - VS Code
# - 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 ↔ VS Code.
#
# 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 (VS Code / 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'"]
```

2.2.1.3 Stack de Python

Para el análisis geoespacial avanzado, es necesario instalar 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, instalaremos **rasterio** para la gestión profesional de datos ráster y **rasterstats** para la extracción de estadísticas zonales.

Table 2.1: 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 Figure 2.1:

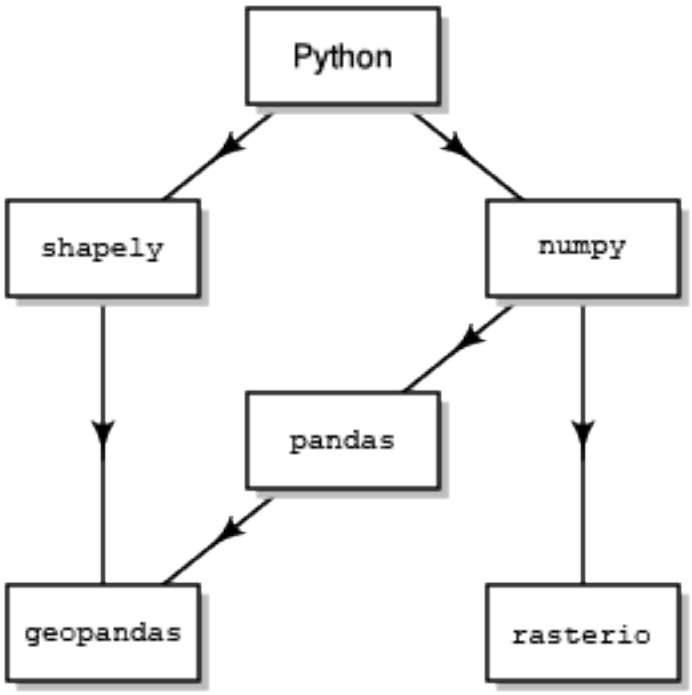


Figure 2.1: Main dependencies between the Python packages we are going to learn

- **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.
- `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.
`geopandas`	*(Integra los 3 motores)*	Gestión integral de GeoDataFrames espaciales.

2.2.1.4 Stack de R

Para el análisis geoespacial en `R`, utilizaremos un ecosistema robusto basado en el estándar *Simple Features* y motores de alto rendimiento para datos ráster (rejilla - grids).

Table 2.3: 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/

Paquete	Funcionalidad	Sitio web
`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.

2.2.1.5 Stack de Julia

Julia ofrece un rendimiento de nivel C++ con sintaxis simplificada. Debido a que utilizaremos librerías espaciales modernas instaladas vía Mamba, debemos sincronizar el entorno para que el Kernel de Jupyter pueda localizar los controladores actualizados sin colapsar.

Table 2.4: 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.

2.3 Puesta en Marcha y Acceso al Contenedor (Terminal, VSCode, JupyterLab)

Siga este procedimiento para iniciar su laboratorio por primera vez:

1. **Iniciar Docker Desktop**: Asegúrese de que el icono de la ballena está en la barra de tareas.

- **Configuración de Almacenamiento**: Debido a que la imagen políglota y la base de datos requieren un espacio considerable, asegúrese de que el disco de destino tenga al menos **30 GB de espacio libre**.
- **Procedimiento para cambiar la ubicación de las imágenes**: Si su disco principal (`C:`) está sin espacio, mueva Docker a otro disco:
    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** {#swap-ref}: Para procesar datos raster de gran tamaño sin interrupciones en R o Julia, es fundamental ampliar el archivo de intercambio (Swap). Puede consultar más detalles en la Section 2.7 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 VS Code. Esto le mostrará la memoria total y el espacio de Swap disponible tal como se configuró en el Anexo de Referencia Técnica.

2. **Organización de Archivos**: Copie los archivos `docker-compose.yml` y `Dockerfile` dentro de su carpeta de trabajo (nombrada con su ID de correo UNAL).

3. **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
```

4. **Construcción del Entorno (Build)**: **Este comando se ejecuta solo la primera vez**. Desde la terminal, ubicado en su carpeta de ID (`su_carpeta`), ejecute:

```
docker compose build --no-cache 2>&1 | tee build_sig_unal.log
```

5. **Arrancar Contenedores**: El comando `build` una vez se termina satisfactoriamente, no necesita volver a ejecutarse. El principal comando para subir el servicio de las imagenes instaladas se muestra a continuación. Use este comando siempre antes de inciar a trabajar con los contenedores instalados:

```
docker compose up -d
```

6. **Verificación de Logs**: Si desea ver el progreso o verificar errores, revise el archivo `.log` generado o ejecute el siguiente comando, sin embargo la url de acceso a **JupyterLab** mostranda después de ejecutar este comando puede estar errónea. Para acceder a **JupyterLab** vea la siguiente instrucción (**Ingreso a JupyterLab**)

```
docker logs contenedor_sig_unal
```

7. **Ingreso a JupyterLab**: Una vez el contenedor esté corriendo, abra su navegador y pegue la siguiente URL (note que usamos el puerto **8889** definido en nuestro manifiesto):

`http://127.0.0.1:8889/lab?token=geomatica2025`

8. **Localización de Archivos y Persistencia**: En el contenedor, su carpeta de Windows 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` (provista para las guías Quarto) debe residir también en esta ruta para un renderizado correcto.

9. **Acceso y Configuración en VS Code**:

Para una experiencia de desarrollo profesional, conecte VS Code directamente al contenedor:

- Ejecute `Ctrl + Shift + P` y seleccione **Dev Containers: Attach to Running Container...**.
- Seleccione el contenedor `contenedor_sig_unal`.
- **Extensiones**: Una vez conectado "dentro" del contenedor, debe habilitar/instalar las extensiones (*'Install in Container'*) de **R** (REditorSupport), **R Debugger**, **R Extension Pack** (Yuki Ueda), **Julia**, **Python**, **Quarto**, **PostgreSQL** (Chris Kolman), **psql** (doublefint), **GitHub Repositories**, **Container Tools**, **Python** (Microsoft), **Jupyter** (Microsoft). Ver Section A.7.2.
- **Inicialización del Visor Gráfico** (Solo la primera vez por sesión)

Para que los gráficos de R (y los puentes de Python/Julia) se visualicen correctamente en VS Code, debe inicializar el dispositivo gráfico. En su terminal de R, ejecute:

```
# Lanza el servidor de gráficos httpgd
httpgd::hgd()
```

- **Acceso al Visor**: VS Code 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.

10. **Apagar los servicios**:

```
docker compose down
```

2.3.1 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.

Componente	Versión / Estado	Detalles Técnicos
Cirugía SSL	✔ Aplicada	Compatibilidad OpenSSL 3.0 (Sistema) vs 3.3 (Julia).

2.4 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**

2.5 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:

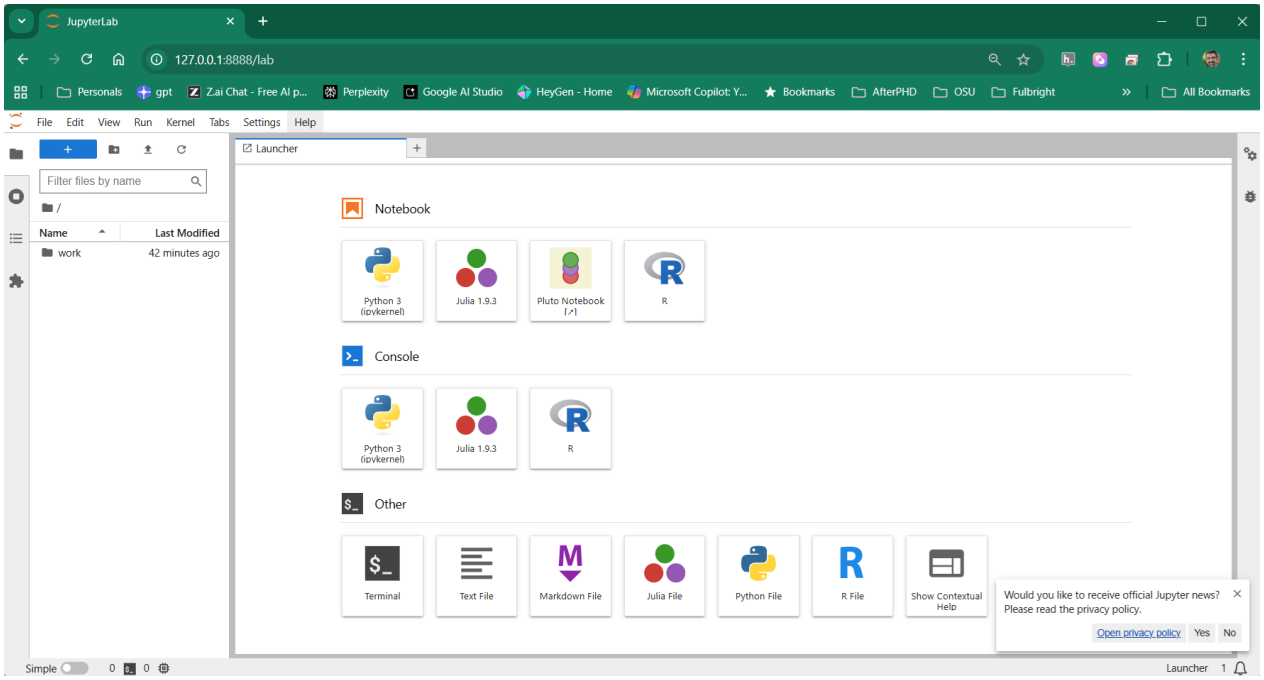


Figure 2.2: Interfaz de JupyterLab configurada para el laboratorio.

2.5.1 La Carpeta 'work' y el Espejo de Datos

En el panel izquierdo de la Figure 2.2, 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.

2.6 Compilación de la guía (documento quarto `qmd`)

Para generar el reporte, ejecute los comandos según el formato requerido:

Generar todos los formatos (HTML y PDF):

```
quarto render guia_instalacion.qmd --to all
```

Generar solo formato HTML:

```
quarto render guia_instalacion.qmd --to html
```

Generar solo formato PDF:

```
quarto render guia_instalacion.qmd --to pdf
```

> Tip
>
> En Quarto, puedes agregar tus **notas de estudio** usando esta sintaxis:
>
> ```
> ::: {.callout-tip icon="true"}
> Escribe acá!
> :::
> ```

2.7 Mantenimiento y Limpieza del Entorno

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.

2.7.1 Comandos de Rescate de Espacio

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.

2.8 Verificación de Conectividad Multilenguaje

Nota técnica: Dentro de la red de Docker, el host es `db-postgis`.

2.8.1 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 ---
```

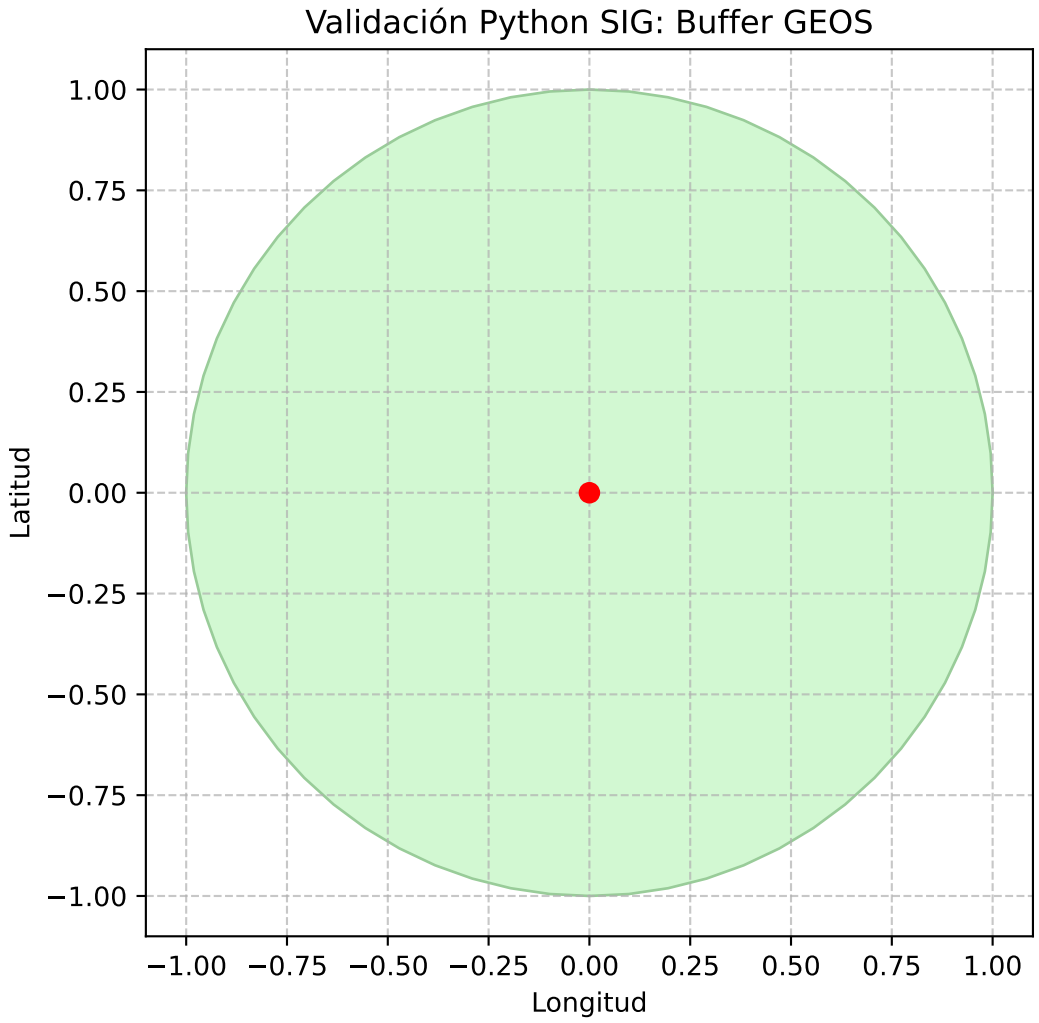
```
# 1. Prueba de conexión a la base de datos PostGIS
try:
    conn = psycopg2.connect(
        host="db-postgis",
        dbname="sig_db_unal",
        user="profe_unal",
        password="geomatica2025"
    )
    print("✅ Conexión a PostGIS: Exitosa")
    conn.close()
except Exception as e:
    print(f"❌ 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 ---
```

2.8.2 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")
```

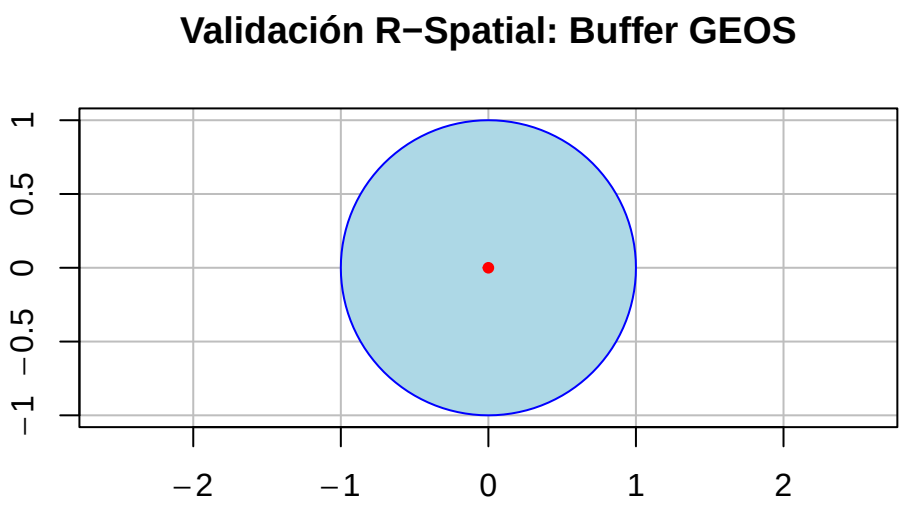
```
--- Inicio de Verificación de R-Spatial ---
```

```
# 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 ---

2.8.3 Julia

```
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 ---")
```

Chapter 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).

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 Seleccionados

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.
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 using 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
```

ℹ ¿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)

Chapter 4

Guía de Instalación. Opción C: ArcGIS Pro y ArcPy

4.1 ..

Part II

Fundamentos

Chapter 5

Fundamentos: Python, R y Julia

5.1 1. Principios de programación con Geometrías

En esta sesión inicial de la **Maestría en Geomática**, exploraremos la sintaxis básica de los tres lenguajes dominantes en la ciencia de datos espaciales. Utilizaremos el estándar *Simple Features* y la representación *Well-Known Text* (WKT) para modelar objetos en el territorio colombiano. En estos ejemplos no se define formalmente un *Sistema de Referencia de Coordenadas* (CRS por sus siglas en inglés)

5.1.1 Creación de Geometrías desde WKT

5.1.2 Python

Utilizaremos `shapely` para instanciar objetos espaciales. Nota que `shapely` por sí solo no gestiona proyecciones; solo manipula la topología y geometría en el plano.

```
# #| eval: false
import shapely

# Definición de un polígono (coordenadas cartesianas origen 0,0)
# Importante: No tiene SRC definido ni unidades específicas.
pol_wkt = 'POLYGON ((0 0, 0 -1, 7.5 -1, 7.5 0, 0 0))'
pol1 = shapely.from_wkt(pol_wkt)

print(f"Tipo de objeto: {type(pol1)}")
```

```
Tipo de objeto: <class 'shapely.geometry.polygon.Polygon'>
```

```
print(f"Representación WKT: {pol1}")
```

```
Representación WKT: POLYGON ((0 0, 0 -1, 7.5 -1, 7.5 0, 0 0))
```

```
# Visualización directa en cuadernos
pol1
```

```
<POLYGON ((0 0, 0 -1, 7.5 -1, 7.5 0, 0 0))>
```

5.1.3 R

En R, la librería fundamental es `sf`. Noten que el WKT es un estándar universal. En R, al crear un objeto con `st_as_sfc`, el parámetro `crs` queda como `NA` por defecto, indicando un sistema puramente cartesiano.

```
# #| eval: false
library(sf)
```

```
Linking to GEOS 3.12.1, GDAL 3.8.4, PROJ 9.4.0; sf_use_s2() is TRUE
```

```
pol_wkt <- 'POLYGON ((0 0, 0 -1, 7.5 -1, 7.5 0, 0 0))'
pol1 <- st_as_sfc(pol_wkt)
print(class(pol1))
```

```
[1] "sfc_POLYGON" "sfc"
```

```
print(pol1)
```

```
Geometry set for 1 feature
Geometry type: POLYGON
Dimension:     XY
Bounding box:  xmin: 0 ymin: -1 xmax: 7.5 ymax: 0
CRS:           NA

POLYGON ((0 0, 0 -1, 7.5 -1, 7.5 0, 0 0))
```

```
plot(pol1)
```

```
# El resultado mostrará: CRS: NA
```

5.1.4 Julia

En Julia, la eficiencia se logra mediante `LibGEOS.jl`.

```
# 1. Configuración del entorno (Ejecutar en la primera celda del Notebook)
using Pkg

# Forzar la activación del proyecto local y añadir los paquetes si no están
if !haskey(Pkg.project().dependencies, "LibGEOS")
    Pkg.add(["LibGEOS", "Plots", "ArchGDAL", "DataFrames", "LibPQ"])
end

# 2. Carga de librerías
using LibGEOS
using Plots

# 3. Código de verificación
pol_wkt = "POLYGON ((0 0, 0 -1, 7.5 -1, 7.5 0, 0 0))"
#pol1 = LibGEOS.readWKT(pol_wkt) # Nota: use readWKT que es el estándar de LibGEOS
# La función correcta en la API actual es LibGEOS.readgeom
pol1 = LibGEOS.readgeom(pol_wkt)

# Imprimir descripción en consola
println("Tipo: ", typeof(pol1))
println("WKT: ", pol1)

# Imprimir/Visualizar gráfico
# Esto requiere que Plots.jl esté instalado y cargado
plot(pol1, title="Polígono en Julia", fillalpha=0.5,fillcolor=:green, aspect_ratio=:equal)
```

5.2 2. Estructuras de Datos y Geometría en Colombia

5.2.1 Listas y Tuplas (Colecciones de Puntos)

Utilizaremos listas para agrupar coordenadas de ciudades principales.

5.2.2 Python

Las **Listas** agrupan geometrías. Las **Tuplas** representan coordenadas inmutables.

```
# #| eval: false
from shapely.geometry import Point

# Lista de puntos: Bogotá y Medellín
ciudades = [Point(-74.07, 4.60), Point(-75.56, 6.25)]

# Agregar Cali usando una tupla de coordenadas
coord_cali = (-76.52, 3.43)
ciudades.append(Point(coord_cali))

print(f"Número de ciudades cargadas: {len(ciudades)}")
```

```
Número de ciudades cargadas: 3
```

5.2.3 R

En R manejamos listas de geometrías (sfc) para conformar capas.

```
# #| eval: false
library(sf)
ciudades <- list(st_point(c(-74.07, 4.60)), st_point(c(-75.56, 6.25)))
# Inserción en lista
ciudades[[3]] <- st_point(c(-76.52, 3.43))
```

5.2.4 Julia

Julia destaca por su manejo de tipos en arreglos.

```
using LibGEOS
ciudades = [LibGEOS.Point(-74.07, 4.60), LibGEOS.Point(-75.56, 6.25)]
push!(ciudades, LibGEOS.Point(-76.52, 3.43))
```

5.2.5 Diccionarios (Metadatos de Capas)

5.2.6 Python

```
# #| eval: false
departamento = {
    "nombre": "Cundinamarca",
    "centroide": shapely.Point(-74.1, 4.8),
    "codigo_dane": 25
}
print(f"Entidad: {departamento['nombre']}")
```

```
Entidad: Cundinamarca
```

5.2.7 R

```
# #| eval: false
departamento <- list(
  nombre = "Cundinamarca",
  centroide = st_point(c(-74.1, 4.8)),
  codigo_dane = 25
)
print(departamento$nombre)
```

```
[1] "Cundinamarca"
```

5.2.8 Julia

```
departamento = Dict(
    "nombre" => "Cundinamarca",
    "centroide" => LibGEOS.Point(-74.1, 4.8),
    "codigo_dane" => 25
)
```

5.3 3. Funciones y Matrices de Coordenadas

5.3.1 Funciones Geométricas

5.3.2 Python

```python
# #| eval: false
def crear_zona_influencia(geom, radio=0.01):
    """Retorna un buffer geométrico."""
    return geom.buffer(radio)

buffer_bogota = crear_zona_influencia(ciudades[0])
```

5.3.3 R

```r
# #| eval: false
crear_zona_influencia <- function(geom, radio = 0.01) {
  return(st_buffer(geom, dist = radio))
}
```

5.3.4 Julia

```julia
crear_zona_influencia(geom, radio=0.01) = LibGEOS.buffer(geom, radio)
```

5.3.5 Matrices (Arrays) - Datos del IDEAM

Simularemos una matriz de estaciones meteorológicas con (Lon, Lat, Elevación).

5.3.6 Python

```python
# #| eval: false
import numpy as np

estaciones = np.array([
    [-74.0, 4.6, 2600],
    [-75.5, 6.2, 1495],
    [-76.5, 3.4, 1018]
])

# Extraer elevaciones (Todas las filas, columna 2)
elevaciones = estaciones[:, 2]
```

5.3.7 R

```r
# #| eval: false
estaciones <- matrix(c(-74.0, 4.6, 2600, -75.5, 6.2, 1495, -76.5, 3.4, 1018),
                     nrow = 3, byrow = TRUE)
elevaciones <- estaciones[, 3]
```

5.3.8 Julia

```
estaciones = [-74.0 4.6 2600;
              -75.5 6.2 1495;
              -76.5 3.4 1018]
elevaciones = estaciones[:, 3]
```

Chapter 6

Estructuras de Datos Geoespaciales

6.1 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).

6.2 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.

6.2.1 Python (Shapely)

```
# #| eval: false
from shapely.geometry import Point, LineString

# Crear punto (Longitud, Latitud)
plaza_bolivar = Point(-74.076, 4.598)

# Crear una línea (un segmento de la Séptima)
calle_septima = LineString([(-74.076, 4.598), (-74.075, 4.605)])

print(f"Tipo: {plaza_bolivar.geom_type}")
```

```
Tipo: Point
```

```
print(f"WKT: {plaza_bolivar.wkt}")
```

```
WKT: POINT (-74.076 4.598)
```

6.2.2 R (sf)

```r
# #| eval: false
library(sf)
```

```
Linking to GEOS 3.12.1, GDAL 3.8.4, PROJ 9.4.0; sf_use_s2() is TRUE
```

```r
# Crear punto
plaza_bolivar <- st_point(c(-74.076, 4.598))

# Crear línea
calle_septima <- st_linestring(rbind(c(-74.076, 4.598), c(-74.075, 4.605)))

print(plaza_bolivar)
```

```
POINT (-74.076 4.598)
```

6.2.3 Julia (LibGEOS)

```julia
using LibGEOS

# Crear punto desde formato WKT
plaza_bolivar = LibGEOS.readgeom("POINT (-74.076 4.598)")

println("Tipo: ", typeof(plaza_bolivar))
```

6.3 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`

6.3.1 Ejemplo: Ciudades Principales de Colombia

6.3.2 Python (GeoPandas)

```python
# #| eval: false
import geopandas as gpd
import pandas as pd
from shapely.geometry import Point

# Datos alfanuméricos
df = pd.DataFrame({
    'Ciudad': ['Bogotá', 'Medellín', 'Cali'],
    'Pob': [7.9, 2.5, 2.2]
})
```

```
# Geometrías
geoms = [Point(-74.08, 4.60), Point(-75.56, 6.25), Point(-76.52, 3.42)]

# Unión en GeoDataFrame
gdf = gpd.GeoDataFrame(df, geometry=geoms, crs="EPSG:4326")
print(gdf)
```

```
     Ciudad  Pob             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)
```

6.3.3 R (sf)

```
# #| eval: false
library(sf)

ciudades <- data.frame(
  Ciudad = c("Bogotá", "Medellín", "Cali"),
  Pob = c(7.9, 2.5, 2.2),
  lon = c(-74.08, -75.56, -76.52),
  lat = c(4.60, 6.25, 3.42)
)

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            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)
```

6.3.4 Julia (GeoTables)

```
using GeoTables, GeometryBasics, DataFrames

df = DataFrame(Ciudad = ["Bogotá", "Medellín", "Cali"], Pob = [7.9, 2.5, 2.2])
puntos = [Point2f(-74.08, 4.60), Point2f(-75.56, 6.25), Point2f(-76.52, 3.42)]

gt = GeoTable(df, geometry = puntos)
println(gt)
```

6.4 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.

Part III

Prácticas

Chapter 7

Benchmark de Procesamiento Geoespacial

7.1 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

7.2 Función j_eval en R

7.3 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.

7.3.1 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

7.3.1.1 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")
```

7.4 Metodología del Benchmark

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*).

7.4.1 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.

7.4.2 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.

7.4.3 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)*

7.4.4 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.

7.4.5 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

---

7.4.6 Limitaciones inevitables del benchmark

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.*

7.4.7 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.

7.4.7.1 ¿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`).

7.4.7.2 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.

7.5 Análisis de Rendimiento y Paralelismo

7.5.1 Benchmark en Python vs. Julia

7.5.1.1 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: 3.888 seg | mean = 3766.624630
```

```
# ------------------------------------------------
# 4. Plot (FUERA DEL BENCHMARK)
# ------------------------------------------------
plt.imshow(res, cmap="terrain")
```

```
<matplotlib.image.AxesImage object at 0x7f9f2edac950>
```

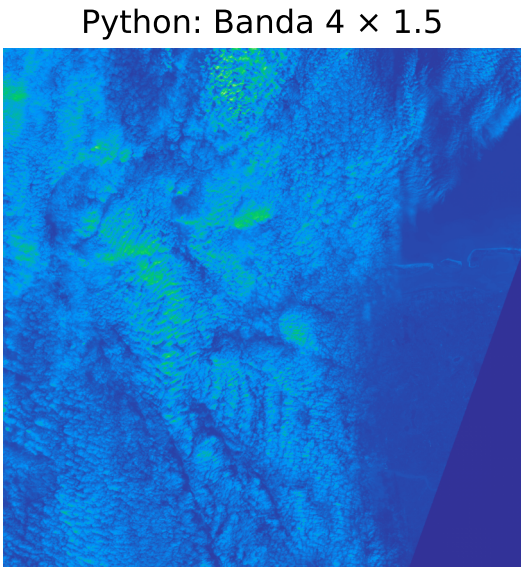
```
plt.title("Python: Banda 4 × 1.5")
```

```
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: 3.888 seg.
```

7.5.1.2 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 ...

julia> using Rasters, ArchGDAL, Statistics, Plots

julia> # Evita restricciones artificiales de memoria

julia> Rasters.checkmem!(false)
false

julia> # ------------------------------------------------

julia> # 1. Leer ruta compartida

julia> # ------------------------------------------------

julia> path = strip(read("s2_shared_path.txt", String))
"SENTINEL2_L1C:/vsizip//usr/local/lib/R/site-library/starsdata/sentinel/S2A_MSIL1C_20180220T105051_N0206
```

```
julia> # ------------------------------------------------

julia> # 2. FUNCIÓN DE BENCHMARK (proxy + mean)

julia> # ------------------------------------------------

julia> 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
process_band_mean (generic function with 1 method)

julia> # ------------------------------------------------

julia> # 3. WARM-UP (compilación)

julia> # ------------------------------------------------

julia> process_band_mean(path)
3766.6246303263756

julia> GC.gc()

julia> # ------------------------------------------------

julia> # 4. BENCHMARK REAL

julia> # ------------------------------------------------

julia> t0 = time_ns()
0x0000028403b0f8b0

julia> m_julia = process_band_mean(path)
3766.6246303263756

julia> t1 = time_ns()
0x00000286775f403d

julia> t_julia = (t1 - t0) / 1e9
```

```
10.530736013

julia> println("⏱ Julia: ", round(t_julia, digits=3), " seg | mean = ", round(m_julia, digits=6))
⏱ Julia: 10.531 seg | mean = 3766.62463

julia> # ------------------------------------------------

julia> # 5. Plot (FUERA DEL BENCHMARK, proxy)

julia> # ------------------------------------------------

julia> 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
"/home/rstudio/work/01_prog_sig/julia_plot.png"

julia> # Debe ser la última para que j_eval en R capture solo el número

julia> t_julia
10.530736013
```

```r
# 2. R muestra la imagen guardada por Julia en el HTML
knitr::include_graphics("julia_plot.png")
```

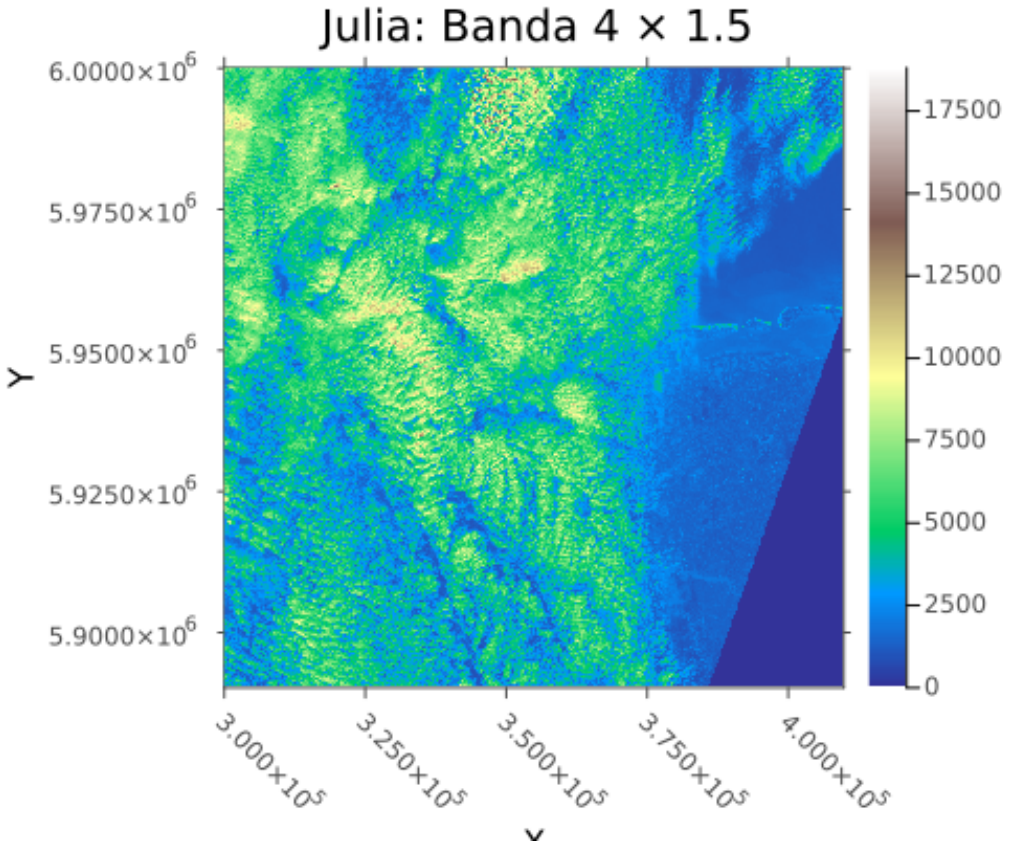


Figure 7.1: Procesamiento de alta resolución en Julia

```
# 3. Impresión desde R
print(paste("ϟ Tiempo capturado en R:", t_julia, " seg."))
```

```
[1] "ϟ Tiempo capturado en R: 10.530736013  seg."
```

7.5.2 Benchmark en R: Terra vs Stars

7.5.2.1 🏁 R: terra

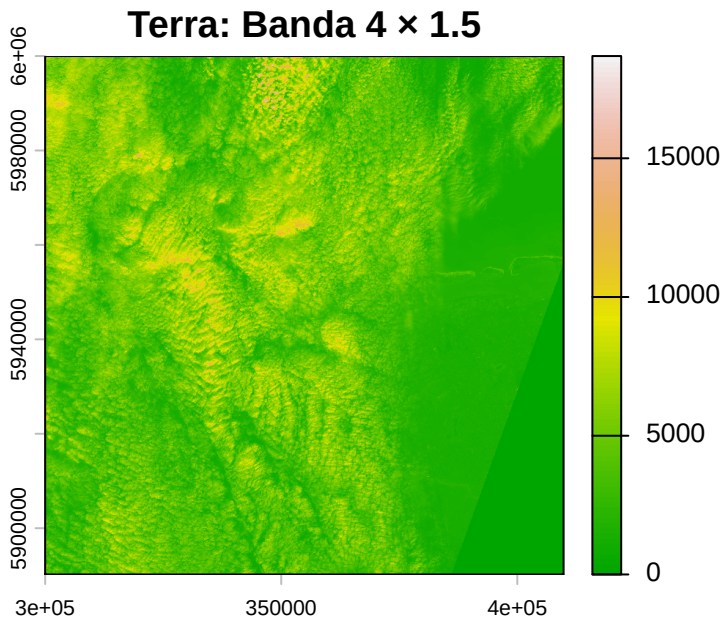
`terra` está desarrollado sobre C++. Su fortaleza es la velocidad de lectura y el manejo de memoria mediante punteros externos. **Paralelismo:** Para esta tarea (operación escalar), `terra` trabaja de forma **secuencial** (monohilo), confiando en la optimización de sus bucles en C++.

```
# 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: 4.605 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 2612888 139.6    4974351 265.7  4974351 265.7
Vcells 5101831  39.0   10497356  80.1 10412733  79.5
```

7.5.2.2 ☆ 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.

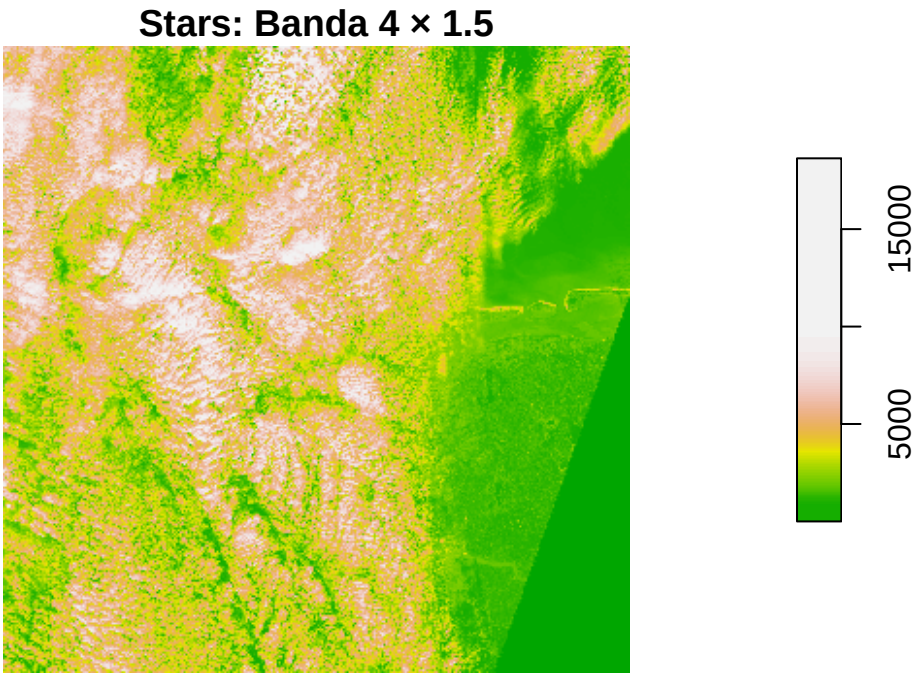
Paralelismo: En operaciones aritméticas simples —como una **multiplicación escalar** seguida de una **media global**— **ni stars ni terra garantizan paralelismo explícito por defecto**. En estos casos, el procesamiento suele realizarse de forma: - **Secuencial**, o
- **Por bloques**, dependiendo de la 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: 8.483 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   2663588 142.3    4974351  265.7   4974351  265.7
Vcells 124985186 953.6  562392541 4290.8 672236837 5128.8
```

7.6 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   4.605474
2          R: stars                  R             Monohilo   8.482853
3  Python: rasterio Python (C++/NumPy)   SIMD (Vectorizado)   3.887517
4 Julia: Rasters.jl     Julia (Nativo) Multihilo (12 hilos)  10.530736
```

Table 7.5: Duelo de Titanes: Procesamiento de 1GB Sentinel-2

Motor	Lenguaje	Paralelismo	Tiempo (s)	Eficiencia (X)
R: terra	R (C++)	Monohilo	4.605	2.29
R: stars	R	Monohilo	8.483	1.24

Python: rasterio	**Python (C++/NumPy)**	**SIMD (Vectorizado)**	**3.888**	**2.71**
Julia: Rasters.jl	Julia (Nativo)	Multihilo (12 hilos)	10.531	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.

7.6.1 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.

7.7 Más allá del benchmark: 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:

7.7.1 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.

7.7.2 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.

7.7.3 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.

7.7.4 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.*

7.8 🏁 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**.

7.8.1 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).

7.8.2 Parte A — Ejecución en JupyterLab (Notebooks)

Ejecuten los **cuatro procesos del benchmark** desde **JupyterLab**, usando el kernel adecuado para cada lenguaje.

7.8.2.1 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

---

7.8.3 Parte B — Ejecución desde VS Code (Terminal Integrada)

Ahora repitan el benchmark **fuera de Jupyter**, usando la terminal integrada de **VS Code**.

7.8.3.1 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

7.8.3.2 Ejecución esperada

Desde la terminal de VS Code:

```
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`.

7.8.4 Parte C — Ejecución desde Windows Terminal (PowerShell)

Finalmente, ejecuten los procesos **sin usar VS Code ni Jupyter**, directamente desde **Windows Terminal (PowerShell)**, trabajando con Docker.

Pueden usar **una o ambas opciones**.

7.8.4.1 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.

7.8.4.2 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 VS Code

---

7.8.5 Preguntas de Análisis

7.8.5.1 1. Entorno de ejecución

¿Notaron diferencias de tiempo entre:

- JupyterLab
- VS Code (terminal integrada)
- Windows Terminal (PowerShell)

Den **una razón técnica posible** (overhead del kernel, entorno, proceso, etc.).

7.8.5.2 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.

7.8.5.3 3. Julia y el calentamiento

¿El efecto del *warm-up* de Julia se notó más en algún entorno específico?
Expliquen brevemente por qué.

7.8.5.4 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.

7.9 Limpieza de Recursos

Part IV

Presentaciones

Chapter 8

Benchmark geoespacial: cómo leer los resultados

Una comparación de stacks, no de lenguajes

8.1 ¿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: Open GIS Reference (Simple Features))
- Librerías raster
- Modelo de ejecución
- Nivel de abstracción

El lenguaje es solo una parte del sistema.

8.2 ¿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.

8.3 ¿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.

8.4 ¿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.

8.5 Modelos de ejecución comparados

Cada stack sigue una filosofía diferente:

- **Python / rasterio + NumPy**
  Lectura completa a memoria + ejecución inmediata
- **R / terra**
  Raster algebra 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

---

8.6 ¿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.

8.7 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.

8.8 Niveles de abstracción por motor

Motor	Nivel de abstracción	Forma de trabajar
NumPy / rasterio	Baja	"Aquí tienes un array, 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.

8.9 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.

8.10 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.

8.11 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.

8.12 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**.

8.13 ¿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.

8.14 ¿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 **VS Code**
- 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.

8.15 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.

Appendix A

Referencia Técnica y Compendio de Comandos

A.1 Introducción a la Infraestructura de Datos

Este anexo constituye la guía técnico para la gestión del entorno de desarrollo. A diferencia de los capítulos teóricos, aquí nos centramos en la operatividad: cómo configurar el motor de Docker, cómo asegurar que la memoria virtual (Swap) no colapse en procesos masivos, cómo sincronizar el conocimiento mediante Git, y otros temas técnicos (ej. comandos) importantes.

A.2 Contenedores Instalados

A.2.1 Resumen de Infraestructura Docker

Antes de entrar en los detalles técnicos, es vital entender el rol de los dos archivos maestros que sostienen su laboratorio geográfico.

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`).

! Persistencia de Datos

Gracias al `docker-compose`, si su contenedor se apaga o se reinicia, los datos de su base de datos PostGIS **no se pierden**. El volumen nombrado actúa como un disco duro externo que sobrevive a cualquier caída del sistema.

A.2.2 Batería de Pruebas de Integridad del Docker (Health Checks)

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.
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).

ID	Comando de Verificación (Docker)	Descripción
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 VS Code está listo en el puerto `8787`.

A.2.3 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 VS Code	`http://localhost:8788`
PostGIS (DB)	`5434`	`5432`	Base de Datos Espacial	`localhost:5434`

A.2.4 Credenciales de la Base de Datos

Utilice estos datos para configurar sus conexiones en QGIS, DBeaver o mediante código (R/Python/Julia):

- **Base de Datos:** `sig_db_unal`
- **Usuario:** `profe_unal`
- **Contraseña:** `geomatica2025`

- **Host Interno:** `db-postgis` (Use este nombre únicamente para conexiones **dentro** de sus scripts).

A.2.5 Lógica de Arquitectura (Dockerfile y 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 VS Code al visor de gráficos de R.
   - `5434:5432`: Evita el choque con bases de datos locales (Postgres suele usar el 5432 o 5433).

A.2.6 Guía de Acceso: El concepto de "Lados"

Es fundamental entender desde dónde está intentando conectar:

A.2.6.1 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`.

A.2.6.2 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`.

A.3 Quarto: Orquestación y Configuración

Quarto se basa en un archivo de configuración centralizado donde se definen los parámetros de renderizado, bibliografía y estilos.

A.3.1 Opciones del encabezado yml

Ejemplo 1: Solo HTML

```yaml
---
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

```yaml
---
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
---
```

A.3.2 Resumen de comandos Quarto

`quarto render guia_instalacion.qmd --to all`

`quarto render guia_instalacion.qmd --to html`

`quarto render guia_instalacion.qmd --to pdf`

A.3.3 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".

Opción de Chunk	¿Se ejecuta?	¿Muestra el Código?	¿Muestra Resultados / Plots?	Uso Ideal
`#| include: false`	✔ SÍ	✖ NO	✖ NO	**Para el Setup.** Corre todo en secreto, sin mensajes ni advertencias.
`#| 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…").

A.3.3.1 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.

Tipo de Chunk	Opción recomendada	¿Por qué?
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.
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...
```

A.4 Docker: Gestión de Contenedores e Imágenes

A.4.1 Procedimiento Incial

Antes de trabajar con Docker debes:

1. Arrancar Docker Desktop
2. Usando la terminal **PowerShell** de Windows ubicarte sobre la carpeta dónde se encuentran los archivos `Dockerfile` y `docker-compose.html`

Los siguientes pasos son **compilar** (solo una vez), arrancar las imágenes y detener las imágenes:

3. **Compilación Limpia (Sin Cache) con Log:**

```
docker build --no-cache -t mi_sig_env:v1 . > build_details.log 2>&1
```

4. **Arrancar las imágenes:**

```
docker compose up -d
```

5. **Detener las imágenes:** (opcional: solo para terminar)

```
docker compose down
```

Finalmente trabajar con los contenedores dentro de VSCode:

6. "Ctrl + Shif + P": "Dev Containers: Attach to Running Container..." y seleccionar container `image_sig_unal` (`contenedor_sig_unal`)

A.4.2 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).
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.

A.4.3 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*.

A.4.4 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**.

A.4.5 Optimización de Memoria 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.

A.4.5.1 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.

A.4.5.2 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.

A.4.6 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 VS Code es la causa número uno de colapsos en el contenedor.

A.5 Git y GitHub: Sincronización Local-Remota

A.5.1 Subír el contenido existende de una carpeta local a un repositorio vacío en 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 que archivos y carperas nunca deben ser registrados con Git. A continuación un archivo de ejemplo:

```
.quarto/
_freeze/
.ipynb_checkpoints/
__pycache__/
*.pdf
*.html
notebooks/
scripts/
```

```
data/
```

7. **Registrar (add)** el archivo `.gitignore` con Git:

```
git add .gitignore
```

8. Revise nuevamente el **estátus** y verifique cambios:

```
git status
```

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 Section A.5.2) 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/reposi`

```
git remote add origin https://github.com/usuario/repositorio.git
```

12. Procedimiento para **generar el PAT** en GitHub.com {#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.

- Registrar archivos específicos con Git (local):

```
git add file1 file2 ...
```

- 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:

```
git add .
```

- En **resumen** estos son los comandos que usarás permanentemente para mentener sincronizados tus archivos con Git y tu Git con GitHub.
- `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)

A.5.2 Autenticación Segura: Configuración de Llaves SSH en GitHub

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.

A.5.2.1 Requitiso 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.

A.5.2.2 Requisito Previo: Instalación de Git

Es indispensable que su sistema operativo cuente con el motor de Git antes de iniciar la configuración:

A.5.2.3 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
```

A.5.2.4 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.

> **i** ¿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
```

> **i** 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.

A.5.2.5 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](https://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**.
```
```

```
#### 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:
```powershell
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
```

A.5.2.6 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.`

A.6 ¡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**, **VS Code**, **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.

A.6.1 Vinculación del Repositorio de Contenidos 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 Section A.5.2.
```

> ⚠ ¡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.

A.6.1.1 Opción A: Sincronizar mediante SSH (Recomendado)

Utilice este método si ya configuró sus llaves siguiendo los pasos de la Section A.5.2.

- **URL de 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.`

A.6.2 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 (**?**)) para realizar operaciones de subida (*push*).

- **URL de la Clase**: `https://github.com/alexyshr/ProgramacionSIG2026.git`
- **Instrucción**: Este comando descarga el repositorio completo y crea automáticamente una subcarpeta con el nombre del proyecto:

```
git clone https://github.com/alexyshr/ProgramacionSIG2026.git
```

A.6.3 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` o `clone` en sesiones futuras.
- **Seguridad**: Si opta por HTTPS, recuerde que GitHub ya no acepta contraseñas básicas por terminal; deberá generar un Token en la configuración de su cuenta. Por esta razón, se recomienda priorizar el uso de **SSH**.

Siguiente Paso

Con el repositorio sincronizado y el contenedor de Docker activo (como se vio en la Section 2.2), ya puede comenzar a trabajar en sus archivos `.qmd` y ver los cambios reflejados en tiempo real.

A.7 Visual Studio Code: Extensiones e Interfaz

A.7.1 Atajos de Poder: La Paleta de Comandos

La mayoría de las funciones avanzadas de VS Code 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 VS Code 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 VS Code 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.

A.7.1.1 Atajos de Teclado Rápidos

Además de la paleta, memorice estos tres para su flujo diario: * `Ctrl + Shift +` (acento grave): Abre una nueva Terminal. * `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.

A.7.2 Configuración de Extensiones en VS Code

Para que el entorno de desarrollo sea plenamente funcional, es necesario instalar un conjunto de extensiones específicas. Una vez que haya realizado el proceso de **"Attach to Running Container"**, asegúrese de que estas herramientas estén instaladas **dentro del contenedor** (busque el botón azul *Install in Dev Container* en la pestaña de extensiones).

Table A.8: Extensiones de VS Code 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`
Ecosistema Julia	**Julia**	Soporte integral para ejecución y gráficos	`julialang.language-julia`

Categoría	Herramienta	Propósito	ID de la Extensión
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`

A.7.2.1 Notas sobre herramientas específicas

- **PostgreSQL (Chris Kolkman)**: Esta extensión le permitirá conectarse a la base de datos espacial del contenedor sin salir de VS Code. 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 una extensión adicional para funcionar en la terminal.

! Instalación Local vs. Remota

Recuerde que extensiones como **Dev Containers** y **Docker** deben estar instaladas en su VS Code **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.

A.7.3 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 VS Code, 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 VS Code, facilitando las conexiones futuras.

A.8 Comandos Rápidos y Shortcuts

A.8.1 Jupyter Notebook (Dentro de VSCode)

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`

A.8.2 Comandos de Windows (PowerShell/CMD)

- **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`

A.9 Adicionales Sobre Python

A.9.1 Versiones Instaladas del Software

Para verificar las versiones instaladas, 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__`

Software	Tipo	Versión	Comando (desde terminal)	Comando (desde Python)
`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__`

A.10 Adicionales Sobre R

A.10.1 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")`

A.10.2 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 VS Code puede renderizar.

A.10.2.1 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()
```

A.10.2.2 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":

```
# Forzar host y puerto para compatibilidad con Docker
httpgd::hgd(host = "0.0.0.0", port = 8787, token = FALSE)
```

A.10.2.3 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 VS Code.
Navegador Externo	`httpgd::hgd_browse()`	Ideal si trabaja con dos monitores para ver mapas en pantalla completa.

> **i** ¿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 VS Code sin que el firewall interno bloquee la petición por falta de credenciales.

A.10.2.4 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*).

A.10.3 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 (`;`).
Python	`>>>`	Consola básica; se recomienda el uso de **IPython** para interactividad.

La experiencia interactiva y el soporte gráfico dentro de **VS Code** 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	VS Code Plots (Pane)	✔
R	Nativo (Interactivo)	R Extension	**httpgd**	✔
Julia	Nativo (REPL)	VSCodeServer	VS Code 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	✔

A.10.3.1 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 VS Code.

- **¿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.

A.10.4 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	✔

A.10.4.1 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.

A.10.4.2 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 VS Code (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**.

A.11 Adicionales Sobre Julia

A.11.1 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)`

A.12 Comandos de Terminal (Sistema Operativo)

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.

Comando	Propósito	¿Para qué sirve en el curso?
`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.
`which r` / `which julia`	Ruta del ejecutable	Localizar dónde están instalados los lenguajes para configurar el Path.

A.12.1 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 VS Code, 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.

A.12.1.1 ¿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.

A.13 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`).

A.13.1 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.

A.13.2 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`).

A.13.3 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.

A.14 Herramientas de admon de Windows

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:).
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**.

Acceso Directo	Parámetro / Comando	Descripción
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

Appendix B

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.

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.

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. In *GitHub repository*. https://github.com/opengeos/pyqgis-cookbook; GitHub.

Wu, Q. (2023b). QGIS notebook plugin: Integrate jupyter notebooks into QGIS. In *GitHub repository* [Computer 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.