Clase 04: Representación de datos geográficos

Trabajando con R y Python

Profesor: Francisco Parra O.

27 de agosto de 2025

USACH - Ingeniería Civil en Informática

Agenda

Paquete sf en R

Paquete sf en R: Introducción

¿Qué es sf?

- Simple Features para R
- Estándar OGC/ISO
- Reemplazo moderno de sp
- Integración con tidyverse

Ventajas:

- Sintaxis consistente
- Compatible con dplyr
- Visualización con ggplot2
- Manejo eficiente de CRS

Instalación y carga:

```r
# Instalaci n
install.packages("sf")

# Carga
library(sf)
library(tidyverse)

# Verificar instalaci n
sf_extSoftVersion()

```

sf: Estructura de datos

Objeto sf = data.frame + geometría

```r
# Crear puntos desde coordenadas
df <- data.frame(
  nombre = c("USACH", "UC", "U.Chile"),
  lat = c(-33.449, -33.441, -33.442),
  lon = c(-70.681, -70.640, -70.650)
)

# Convertir a sf
puntos_sf <- st_as_sf(df,
  coords = c("lon", "lat"),
  crs = 4326)

# Ver estructura
print(puntos_sf)
class(puntos_sf)

```

Componentes:

- `geometry`: columna especial
- Atributos: columnas regulares
- CRS: sistema de referencia
- Bbox: límites espaciales

Tipos de geometría:

- POINT
- LINESTRING
- POLYGON
- MULTIPOINT
- MULTILINESTRING
- MULTIPOLYGON
- GEOMETRYCOLLECTION

sf: Lectura y escritura

```r
# Lectura de archivos espaciales
comunas <- st_read("comunas_chile.shp")
comunas <- st_read("comunas.geojson")
comunas <- st_read("comunas.gpkg")

# Informaci n b sica
st_geometry_type(comunas)  # Tipo de geometr a
st_crs(comunas)            # Sistema de referencia
st_bbox(comunas)           # L mites espaciales
nrow(comunas)              # N mero de features

# Escritura de archivos
st_write(comunas, "output.shp")
st_write(comunas, "output.geojson")
st_write(comunas, "output.gpkg", layer = "comunas")

# Transformaci n de CRS
comunas_utm <- st_transform(comunas, crs = 32719)

```

sf: Operaciones con dplyr

```r
# Filter
rm_comunas <- comunas %>%
  filter(region == "RM")

# Select
comunas_min <- comunas %>%
  select(nombre, poblacion, geometry)

# Mutate
comunas <- comunas %>%
  mutate(
    area_km2 = st_area(.) / 1e6,
    densidad = poblacion / area_km2
  )

# Group by + summarize
regiones <- comunas %>%
  group_by(region) %>%
  summarize(
    poblacion_total = sum(poblacion),
    n_comunas = n()
  )

```

Operaciones espaciales:

```r
# Buffer
buffer_1km <- st_buffer(puntos_sf,
                        dist = 1000)

# Intersecci n
intersec <- st_intersection(a, b)

# Uni n
union_all <- st_union(comunas)

# Diferencia
diff <- st_difference(a, b)

# Contains
st_contains(poligono, puntos)

# Within
st_within(puntos, poligono)

```

sf: Visualización con ggplot2

```r
# Mapa b sico
ggplot(comunas) +
  geom_sf() +
  theme_minimal()

# Mapa con colores por variable
ggplot(comunas) +
  geom_sf(aes(fill = poblacion)) +
  scale_fill_viridis_c() +
  labs(title = "Poblaci n por comuna",
       fill = "Habitantes") +
  theme_minimal()

# M ltiples capas
ggplot() +
  geom_sf(data = comunas,
          fill = "lightgray") +
  geom_sf(data = puntos_sf,
          color = "red", size = 3) +
  coord_sf(crs = 4326)

```

```r
# Mapa interactivo con tmap
library(tmap)
tmap_mode("view")

tm_shape(comunas) +
  tm_polygons("poblacion",
    palette = "Blues",
    title = "Poblaci n") +
  tm_shape(puntos_sf) +
  tm_dots(size = 0.1, col = "red")

# Facets espaciales
ggplot(comunas) +
  geom_sf(aes(fill = densidad)) +
  facet_wrap(~region) +
  scale_fill_gradient(
    low = "white",
    high = "darkred"
  ) +
  theme_minimal()

```

GeoPandas en Python

GeoPandas en Python: Introducción

¿Qué es GeoPandas?

- Extensión espacial de pandas
- Built on Shapely, Fiona, pyproj
- DataFrames con geometría
- Análisis espacial simplificado

Ecosistema:

- **Shapely**: geometrías
- **Fiona**: I/O de archivos
- **pyproj**: proyecciones
- **rtree**: índices espaciales

Instalación y carga:

```
# Instalaci n
pip install geopandas
# o con conda
conda install -c conda-forge geopandas

# Importaci n
import geopandas as gpd
import pandas as pd
from shapely.geometry import Point
import matplotlib.pyplot as plt

```

GeoPandas: GeoDataFrame

```python
# Crear GeoDataFrame desde puntos
df = pd.DataFrame({
    'ciudad': ['Santiago', 'Valparaíso',
               'Concepción'],
    'lat': [-33.45, -33.04, -36.82],
    'lon': [-70.66, -71.61, -73.04],
    'poblacion': [5614000, 295113, 223574]
})

# Crear geometrías
geometry = [Point(xy) for xy in
            zip(df['lon'], df['lat'])]

# Crear GeoDataFrame
gdf = gpd.GeoDataFrame(df,
    geometry=geometry,
    crs='EPSG:4326')

print(gdf.head())
print(gdf.crs)

```

Atributos importantes:

```python
# Geometría activa
gdf.geometry

# Bounds
gdf.bounds
gdf.total_bounds

#  Área  y longitud
gdf.area  # para polígonos
gdf.length  # para líneas

# Centroide
gdf.centroid

# Tipo de geometría
gdf.geom_type

# Sistema de referencia
gdf.crs

```

GeoPandas: Lectura y escritura

```python
# Lectura de archivos espaciales
gdf = gpd.read_file("comunas.shp")
gdf = gpd.read_file("data.geojson")
gdf = gpd.read_file("database.gpkg", layer='comunas')

# Lectura con filtro espacial
bbox = (-71, -34, -70, -33)  # minx, miny, maxx, maxy
gdf = gpd.read_file("chile.shp", bbox=bbox)

# Escritura
gdf.to_file("output.shp")
gdf.to_file("output.geojson", driver='GeoJSON')
gdf.to_file("output.gpkg", layer='mi_capa', driver="GPKG")

# Formatos adicionales
gdf.to_csv("datos.csv")  # Sin geometr a
gdf.to_parquet("datos.parquet")  # Con geometr a (GeoParquet)

```

GeoPandas: Operaciones espaciales

```python
# Transformaci n de CRS
gdf_utm = gdf.to_crs(epsg=32719)
gdf_wgs = gdf.to_crs('EPSG:4326')

# Buffer
gdf['buffer_1km'] = gdf.buffer(1000)

# Operaciones geom tricas
union = gdf.unary_union
dissolved = gdf.dissolve(by='region')

# Spatial join
points_in_polys = gpd.sjoin(
    points, polygons,
    how='inner',
    predicate='intersects'
)

# Clip
clipped = gpd.clip(gdf, mask)

```

```python
# Predicados espaciales
gdf.intersects(other)
gdf.contains(point)
gdf.within(polygon)
gdf.crosses(line)
gdf.touches(boundary)
gdf.overlaps(other)

# Operaciones overlay
result = gpd.overlay(
    gdf1, gdf2,
    how='intersection'
)
# how: 'intersection', 'union',
#      'difference',
#      'symmetric_difference'

# Distancias
gdf['dist'] = gdf.distance(point)

```

GeoPandas: Visualización

```python
# Plot b sico
gdf.plot()
plt.show()

# Plot con colores por variable
gdf.plot(column='poblacion',
         cmap='Blues',
         legend=True,
         figsize=(10, 6))

# M ltiples capas
fig, ax = plt.subplots(figsize=(10, 8))
comunas.plot(ax=ax, color='lightgray',
             edgecolor='black')
ciudades.plot(ax=ax, color='red',
              markersize=50)
ax.set_title('Mapa de Chile')
plt.show()
```

```python
# Mapa interactivo con Folium
import folium

# Crear mapa base
m = folium.Map(
    location=[-33.45, -70.66],
    zoom_start=10
)

# Agregar capa
folium.GeoJson(
    gdf.to_json(),
    name='comunas'
).add_to(m)

# Guardar
m.save('mapa.html')

# Con explore (m s simple)
gdf.explore(column='poblacion',
            cmap='Blues')
```

Importación y exportación de datos

Formatos de datos espaciales

Formatos vectoriales:

- **Shapefile**: Legacy, múltiples archivos
- **GeoJSON**: JSON, legible
- **GeoPackage**: SQLite, moderno
- **KML/KMZ**: Google Earth
- **GML**: XML-based
- **PostGIS**: PostgreSQL

Formatos raster:

- **GeoTIFF**: Con georreferencia
- **NetCDF**: Datos científicos
- **HDF5**: Multidimensional
- **COG**: Cloud Optimized GeoTIFF

Consideraciones:

Shapefile:

- Nombres max 10 caracteres
- Sin valores NULL
- Límite 2GB

GeoPackage:

- Un solo archivo
- Múltiples capas
- Sin límites de tamaño
- Soporte completo SQL

Importación: Diferentes fuentes

Python - GeoPandas:

```python
# Desde archivo local
gdf = gpd.read_file("data.shp")

# Desde URL
url = "https://ejemplo.com/data.geojson"
gdf = gpd.read_file(url)

# Desde PostGIS
from sqlalchemy import create_engine
engine = create_engine(
    'postgresql://user:pass@host/db'
)
sql = "SELECT * FROM tabla WHERE region='RM'"
gdf = gpd.read_postgis(sql, engine,
                       geom_col='geom')

# Desde CSV con coordenadas
df = pd.read_csv("puntos.csv")
gdf = gpd.GeoDataFrame(df,
    geometry=gpd.points_from_xy(
        df.lon, df.lat))

```

R - sf:

```r
# Desde archivo local
sf_obj <- st_read("data.shp")

# Desde URL
url <- "https://ejemplo.com/data.json"
sf_obj <- st_read(url)

# Desde PostGIS
library(RPostgreSQL)
con <- dbConnect(PostgreSQL(),
    dbname="db", host="host")
sf_obj <- st_read(con,
    query = "SELECT * FROM tabla")

# Desde CSV
df <- read.csv("puntos.csv")
sf_obj <- st_as_sf(df,
    coords = c("lon", "lat"),
    crs = 4326)

```

Exportación: Opciones y optimización

Python:

```python
# Shapefile con encoding
gdf.to_file("output.shp",
            encoding='utf-8')

# GeoJSON simplificado
gdf_simple = gdf.copy()
gdf_simple.geometry = gdf.simplify(
    tolerance=0.001
)
gdf_simple.to_file("simple.geojson",
                   driver='GeoJSON')

# GeoPackage con capas
gdf1.to_file("data.gpkg", layer='capa1',
             driver="GPKG")
gdf2.to_file("data.gpkg", layer='capa2',
             driver="GPKG", mode='a')

# Parquet para big data
gdf.to_parquet("data.parquet")
```

R:

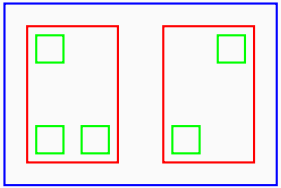
```r
# Opciones de escritura
st_write(sf_obj, "output.shp",
         delete_dsn = TRUE)

# Simplificaci n
sf_simple <- st_simplify(sf_obj,
    preserveTopology = TRUE,
    dTolerance = 100)
st_write(sf_simple, "simple.json")

# GeoPackage multicapa
st_write(sf1, "data.gpkg",
         layer = "capa1")
st_write(sf2, "data.gpkg",
         layer = "capa2",
         append = TRUE)

# Formato eficiente
library(arrow)
write_parquet(sf_obj, "data.parquet")
```

Estructuras de datos espaciales

Estructuras de datos: Índices espaciales

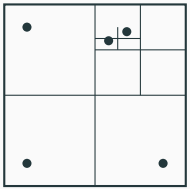
R-tree:

- Estructura jerárquica
- Bounding boxes anidados
- Búsqueda O(log n)
- Usado en GeoPandas/PostGIS



Quadtree:

- División recursiva en 4
- Adaptativo al contenido
- Eficiente para puntos
- Usado en indexación raster



Índices espaciales: Implementación

Python - Uso de índices:

```python
# GeoPandas usa R-tree autom ticamente
gdf.sindex  # Acceso al  ndice

# B squeda eficiente
from shapely.geometry import box
bbox = box(minx, miny, maxx, maxy)

# Candidatos potenciales
possible_matches_index = list(
    gdf.sindex.intersection(bbox.bounds)
)
possible_matches = gdf.iloc[
    possible_matches_index
]

# Verificaci n precisa
precise_matches = possible_matches[
    possible_matches.intersects(bbox)
]

```

R - Índices espaciales:

```r
# sf usa  ndices  autom ticamente
# en operaciones espaciales

# Crear  ndice  expl cito
library(sf)
library(s2)

# Para datos planares
sf_use_s2(FALSE)

# Operaci n con  ndice
result <- st_intersects(
    puntos,
    poligonos,
    sparse = TRUE  # matriz sparse
)

# B squeda por bbox
bbox <- st_bbox(area_interes)
subset <- st_crop(gdf, bbox)

```

Topología y relaciones espaciales

Modelo DE-9IM:

- Interior (I)
- Boundary (B)
- Exterior (E)

	I(B)	B(B)	E(B)
I(A)	dim	dim	dim
B(A)	dim	dim	dim
E(A)	dim	dim	dim

Predicados espaciales:

- `equals`: Misma geometría
- `disjoint`: Sin intersección
- `intersects`: Alguna intersección
- `touches`: Solo boundaries
- `crosses`: Interior cruza
- `within`: A dentro de B

```python
# Python - Relaciones
a.intersects(b)
a.contains(b)
a.within(b)
a.touches(b)
a.crosses(b)
a.overlaps(b)
a.equals(b)
a.disjoint(b)

# Matriz DE-9IM
a.relate(b)
# Returns: '212101212'

# Pattern matching
a.relate_pattern(b,
    'T*F**F***')

```

```r
# R - Predicados
st_intersects(a, b)
st_contains(a, b)
st_within(a, b)
st_touches(a, b)
st_crosses(a, b)
st_overlaps(a, b)

```

Validación y reparación de geometrías

Problemas comunes:

- Self-intersection
- Anillos no cerrados
- Orden incorrecto de vértices
- Geometrías duplicadas
- Slivers (polígonos delgados)

Python - Validación:

```python
# Verificar validez
gdf['is_valid'] = gdf.is_valid

# Ver problemas
invalid = gdf[~gdf.is_valid]
print(invalid)

# Explicaci n del problema
from shapely.validation import explain_validity
for idx, row in invalid.iterrows():
    print(explain_validity(row.geometry))

```

Reparación:

```python
# Python - Reparar
from shapely.geometry import Polygon
from shapely.validation import make_valid

# M todo 1: buffer(0)
gdf['geometry'] = gdf.buffer(0)

# M todo 2: make_valid
gdf['geometry'] = gdf.apply(
    lambda x: make_valid(x.geometry),
    axis=1
)

# R - Reparaci n
sf_obj <- st_make_valid(sf_obj)

# Verificar
st_is_valid(sf_obj)

# Simplificar para eliminar slivers
sf_clean <- st_simplify(
    sf_obj,
    dTolerance = 0.001
)

```

Integración R-Python

R desde Python (rpy2):

```
Cargar sf en R ro.r('library(sf)')
Convertir GeoDataFrame a sf r_sf = pandas2ripy2rpy(gdf)
Ejecutar funcin R result = ro.r['st_buffer
  '](r_sf, 1000)
Volver a Python gdf_buffer = pandas2rirpy2py(result).
```

Python desde R (reticulate):

```
Importar geopandas gpd <- import(
"geopandas")
Leer con geopandas gdf <- gpdreadfile("datashp.")
Convertir a sf sf_obj <- −stassf(gdf)
Operacin en R buffer <- st_buffer(sf_obj, 1000)
Volver a Python py_gdf <- −ritopy(buffer)
```

Optimización y rendimiento

Estrategias de optimización:

- Usar índices espaciales
- Simplificar geometrías
- Filtrar por bbox primero
- Operaciones vectorizadas
- Formato Parquet/Arrow

```
# Simplificar geometr as
gdf_simple = gdf.copy()
gdf_simple.geometry = gdf.simplify(
    tolerance=10,  # metros
    preserve_topology=True
)

# Filtro por bbox antes de operaci n
bbox = target.total_bounds
candidates = source.cx[
    bbox[0]:bbox[2], bbox[1]:bbox[3]
]

```

Procesamiento paralelo:

```
# Python - Dask-GeoPandas
import dask_geopandas as dgpd

# Particionar datos
ddf = dgpd.from_geopandas(
    gdf, npartitions=4
)

# Operaci n paralela
result = ddf.map_partitions(
    lambda x: x.buffer(100)
).compute()

# R - parallel processing
library(parallel)
library(foreach)

cl <- makeCluster(4)
result <- foreach(
    i = 1:nrow(sf_obj),
    .packages = 'sf'
) %dopar% {
    st_buffer(sf_obj[i,], 100)
}

```

Mejores prácticas

Recomendaciones generales:

- Validar geometrías al importar
- Mantener CRS consistente
- Usar proyecciones locales para cálculos
- Documentar transformaciones
- Versionar datos espaciales

Elección de herramienta:

- **R/sf**: Análisis estadístico
- **Python/GeoPandas**: Integración ML
- **PostGIS**: Grandes volúmenes
- **QGIS**: Exploración visual

Flujo de trabajo típico:

1. Importar y validar datos
2. Establecer CRS apropiado
3. Crear índices espaciales
4. Realizar análisis
5. Validar resultados
6. Exportar en formato óptimo

Tip: Siempre trabaja con copias de los datos originales y documenta cada transformación.

Recursos y documentación

Documentación oficial:

- sf documentation
- GeoPandas docs
- Shapely manual
- pyproj reference

Tutoriales recomendados:

- Geocomputation with R
- Python Geospatial Development
- PostGIS in Action
- Spatial Data Science with R

Datasets de práctica:

- Natural Earth Data
- OpenStreetMap
- GADM boundaries
- NASA Earthdata
- Copernicus Open Access Hub

Comunidades:

- r-spatial GitHub
- GeoPandas Discussions
- GIS Stack Exchange
- OSGeo mailing lists

Ejercicio integrado

Análisis de accesibilidad a servicios:

```
# 1. Cargar datos
comunas = gpd.read_file('comunas_rm.geojson')
hospitales = gpd.read_file('hospitales.geojson')
poblacion = pd.read_csv('poblacion_comunas.csv')

# 2. Unir datos de poblaci n
comunas = comunas.merge(poblacion, on='cod_comuna')

# 3. Calcular distancia al hospital m s cercano
from shapely.ops import nearest_points
def distancia_minima(geom, puntos):
    punto_cercano = nearest_points(geom, puntos.unary_union)[1]
    return geom.distance(punto_cercano)

comunas['dist_hospital'] = comunas.geometry.apply(
    lambda x: distancia_minima(x.centroid, hospitales.geometry)
)

# 4. Clasificar accesibilidad y visualizar
comunas['accesibilidad'] = pd.cut(comunas['dist_hospital'],
                                  bins=[0, 2000, 5000, 10000, float('inf')],
                                  labels=['Muy Alta', 'Alta', 'Media', 'Baja'])

```

Cierre

¿Preguntas?

Próxima clase:

Análisis espacial y geoestadística

Tarea:

Implementar pipeline completo de importación, transformación y exportación con datos reales