Manual del Profesor

Clase 03: Fundamentos de Datos Geoespaciales

Tipos y estructuras de datos espaciales

Prof. Francisco Parra O.
Geólogo, PhD en Informática
francisco.parra.o@usach.cl

Semestre 2, 2025
Duración: 80 minutos

Índice

1 Introducción

Este manual proporciona una guía detallada para dictar la Clase 03 del curso de Geoinformática. Está diseñado para ser completamente autocontenido, permitiendo que cualquier profesor pueda conducir la clase exitosamente.

1.1 Objetivos de Aprendizaje

Al finalizar esta clase de 80 minutos, los estudiantes serán capaces de:

1. **Comprender** las diferencias fundamentales entre datos vectoriales y raster
2. **Identificar** los componentes de puntos, líneas y polígonos
3. **Analizar** la estructura de datos raster, resolución y bandas espectrales
4. **Reconocer** los formatos de archivo geoespaciales más comunes
5. **Ejecutar** operaciones básicas con GeoPandas y Rasterio
6. **Aplicar** conceptos de atributos y geometrías en datos espaciales

1.2 Materiales Necesarios

- Presentación: `clase03_martes_fundamentos_de_datos_ajustada.pdf`
- Computador con proyector
- Acceso a internet para demostraciones en vivo
- Python con librerías: GeoPandas, Shapely, Rasterio
- Datos de ejemplo (comunas de Santiago, imagen satelital)

1.3 Preparación Previa

NOTA

30 minutos antes de clase:

- Verificar proyector y computador
- Abrir Jupyter Notebook o VS Code
- Cargar datos de ejemplo en carpeta accesible
- Tener terminal abierta para demostraciones
- Revisar que las librerías estén instaladas

2 Estructura de la Clase

2.1 Distribución del Tiempo

Sección	**Tiempo**	**Actividad**
Introducción	5 min	Agenda y objetivos
Datos Vectoriales	20 min	Puntos, líneas, polígonos + demo
Datos Raster	15 min	Grillas, resolución, bandas, DEM
Formatos de Archivos	10 min	Shapefile, GeoJSON, GeoTIFF, COG
Atributos y Geometrías	15 min	Operaciones, R-tree, validación, CRS
Machine Learning Espacial	10 min	Features espaciales, autocorrelación
Integración Vector-Raster	5 min	Zonal stats, extracción
Total	**80 min**	

3 Guión Detallado de la Clase

3.1 Apertura

[5 min]

DECIR

Buenos días/tardes. Hoy en la Clase 03 vamos a explorar los fundamentos de datos geoespaciales. Esta es una clase crucial porque establece las bases para todo el trabajo que haremos en el curso.
Vamos a entender cómo se representa el mundo real en sistemas computacionales, desde un simple punto GPS hasta complejas imágenes satelitales.

HACER

- Mostrar slide 1: Portada
- Mostrar slide 2: Agenda
- Hacer énfasis en que hoy combinamos teoría con código práctico

NOTA

Si hay estudiantes nuevos, pedirles que se presenten brevemente (30 segundos cada uno).

3.2 Datos Vectoriales

[20 min]

3.2.1. Modelo Vectorial: Fundamentos

[5 min]

DECIR

El modelo vectorial representa el mundo como objetos discretos con geometrías precisas. Piensen en un mapa de Google Maps: cada restaurant es un punto, cada calle es una línea, cada manzana es un polígono.
La ventaja principal es la precisión y eficiencia en almacenamiento. Un punto solo necesita 2 números (X,Y), no importa el zoom que hagamos.

HACER

- Mostrar slide 3: Modelo Vectorial Fundamentos
- Dibujar en pizarra: punto (x,y), línea como secuencia, polígono cerrado
- Mencionar: "Todo objeto espacial tiene geometría + atributos"

ALERTA

Pregunta común: "¿Por qué no usar siempre vectores si son más precisos?Respuesta: No todo se puede representar eficientemente como vectores (ej: temperatura, elevación).

3.2.2. Puntos: Localizaciones Discretas

[5 min]

DECIR

Los puntos son la geometría más simple pero muy poderosa. Cada punto de interés en su celular, cada dirección geocodificada, cada sensor IoT es un punto.
Vamos a crear algunos puntos en Python para ver qué tan simple es.

HACER

- Mostrar slide 4: Puntos
- Ejecutar código en vivo:

```python
from shapely.geometry import Point
import geopandas as gpd

# Crear punto USACH
usach = Point(-70.681, -33.450)
print(f"Coordenadas USACH: X={usach.x}, Y={usach.y}")

# Crear GeoDataFrame con múltiples puntos
estaciones_metro = gpd.GeoDataFrame({
    'nombre': ['Universidad de Santiago', 'Estación Central', 'ULA'],
    'linea': [1, 1, 1],
    'geometry': [
        Point(-70.681, -33.450),
        Point(-70.678, -33.452),
        Point(-70.675, -33.453)
    ]
})

# Visualizar
estaciones_metro.plot(markersize=100, color='red')
```

NOTA

Si hay problemas con la visualización, tener screenshot de respaldo del resultado esperado.

3.2.3. Líneas: Conexiones y Redes

[5 min]

DECIR

Las líneas representan features lineales como calles, ríos, rutas de transporte. Son fundamentales para análisis de redes y routing.
Una línea es simplemente una secuencia ordenada de puntos conectados. La dirección puede ser importante (ej: sentido del tráfico).

HACER

- Mostrar slide 5: Líneas
- Demostrar creación de línea conectando las estaciones de metro anteriores:

```python
from shapely.geometry import LineString

# Crear línea de metro
coordenadas = [(-70.681, -33.450), (-70.678, -33.452), (-70.675, -33.453)]
linea_metro = LineString(coordenadas)

print(f"Longitud: {linea_metro.length} grados")
print(f"Número de vértices: {len(linea_metro.coords)}")

# Buffer de 100 metros alrededor de la línea
zona_influencia = linea_metro.buffer(0.001)  # ~100m
```

3.2.4. Polígonos: Áreas y Regiones

[5 min]

DECIR

Los polígonos representan áreas. Son esenciales para límites administrativos, zonificación, parcelas. Un polígono es un anillo cerrado, el último punto conecta con el primero. Pueden tener huecos (piensen en un donut) que se representan como anillos interiores.

HACER

- Mostrar slide 6: Polígonos
- Ejecutar ejemplo de creación y análisis de polígono:

```python
from shapely.geometry import Polygon

# Campus USACH simplificado
campus = Polygon([
    (-70.683, -33.448),
    (-70.679, -33.448),
    (-70.679, -33.452),
    (-70.683, -33.452),
    (-70.683, -33.448)
])

print(f" rea : {campus.area} grados ")
print(f"Perímetro: {campus.length} grados")

# Verificar si el metro está dentro del campus
print(f" Metro  dentro del campus? {campus.contains(usach)}")
```

ALERTA

Error común: Olvidar cerrar el polígono (repetir primer punto al final). GeoPandas/Shapely lo maneja automáticamente, pero otros software no.

3.3 Datos Raster

[15 min]

3.3.1. Modelo Raster: Fundamentos

[5 min]

DECIR

El modelo raster divide el espacio en una grilla regular de celdas o píxeles. Cada celda tiene un valor. Es como una foto digital, pero cada píxel puede representar temperatura, elevación, tipo de vegetación, etc.
Es ideal para fenómenos continuos donde necesitamos un valor en cada punto del espacio.

HACER

- Mostrar slide 7: Modelo Raster Fundamentos
- Dibujar en pizarra: grilla 5x5 con valores
- Enfatizar: "Trade-off entre resolución y tamaño de archivo"

3.3.2. Resolución Espacial

[5 min]

DECIR

La resolución determina el nivel de detalle. Un píxel de 30m (Landsat) vs 10m (Sentinel) vs 0.3m (WorldView). Mayor resolución = más detalle pero archivos más grandes.
Para Santiago completo: 30m = 1 MB, 10m = 9 MB, 1m = 900 MB aproximadamente.

HACER

- Mostrar slide 8: Resolución Espacial
- Demostrar lectura de raster (usar archivo de ejemplo o simular):

```python
import numpy as np
import matplotlib.pyplot as plt

# Simular raster de elevación
elevacion = np.random.randint(500, 1500, size=(100, 100))

# Diferentes resoluciones
fig, axes = plt.subplots(1, 3, figsize=(12, 4))

# Alta resolución
axes[0].imshow(elevacion, cmap='terrain')
axes[0].set_title('Resolución: 1m')

# Media resolución (promedio 10x10)
media = elevacion.reshape(10, 10, 10, 10).mean(axis=(1, 3))
axes[1].imshow(media, cmap='terrain')
axes[1].set_title('Resolución: 10m')

# Baja resolución (promedio 20x20)
baja = elevacion.reshape(5, 20, 5, 20).mean(axis=(1, 3))
axes[2].imshow(baja, cmap='terrain')
axes[2].set_title('Resolución: 20m')

plt.show()
```

3.3.3. Bandas Espectrales

[5 min]

DECIR

Las imágenes satelitales capturan múltiples bandas del espectro electromagnético. No solo RGB visible, sino infrarrojo, térmico, etc. Cada banda revela información diferente.
El NDVI es el índice más usado: vegetación sana refleja mucho NIR y poco rojo.

HACER

- Mostrar slide 9: Bandas Espectrales
- Explicar cálculo de NDVI con diagrama en pizarra
- Mostrar código de cálculo:

```
# Simular bandas espectrales
red = np.random.rand(100, 100) * 0.3  # Baja reflectancia en rojo
nir = np.random.rand(100, 100) * 0.7  # Alta reflectancia en NIR

# Calcular NDVI
ndvi = (nir - red) / (nir + red + 1e-10)

# Clasificar
vegetacion_densa = ndvi > 0.6
vegetacion_media = (ndvi > 0.3) & (ndvi <= 0.6)
suelo_desnudo = ndvi <= 0.3

print(f"Píxeles con vegetación densa: {vegetacion_densa.sum()}")
```

NOTA

NDVI valores típicos: - Agua: ¡0 - Suelo desnudo: 0 - 0.2 - Vegetación escasa: 0.2 - 0.4 - Vegetación densa: ¿0.4

3.4 Formatos de Archivos

[10 min]

3.4.1. Formatos Vectoriales

[5 min]

DECIR

Cada formato tiene sus ventajas. Shapefile es el más universal pero antiguo. GeoPackage es moderno y eficiente. GeoJSON es perfecto para web. PostGIS para producción.
Vamos a ver las diferencias prácticas.

HACER

- Mostrar slides 11-13: Formatos vectoriales
- Demostrar conversión entre formatos:

```python
# Leer diferentes formatos
comunas_shp = gpd.read_file('comunas.shp')
comunas_json = gpd.read_file('comunas.geojson')

# Guardar en diferentes formatos
comunas_shp.to_file('comunas.gpkg', driver='GPKG')  # Recomendado
comunas_shp.to_file('comunas.json', driver='GeoJSON')  # Para web
comunas_shp.to_file('comunas_new.shp')  # Legacy

# Comparar tamaños
import os
for file in ['comunas.shp', 'comunas.gpkg', 'comunas.json']:
    if os.path.exists(file):
        size = os.path.getsize(file) / 1024
        print(f"{file}: {size:.1f} KB")
```

ALERTA

Limitaciones del Shapefile: - Nombres de campo máximo 10 caracteres - Sin soporte para valores NULL - Múltiples archivos (.shp, .shx, .dbf, .prj mínimo)

3.4.2. Formatos Raster

[5 min]

DECIR

GeoTIFF es el estándar para raster. Cloud Optimized GeoTIFF (COG) es la evolución para la nube, permite leer porciones sin descargar todo el archivo.
NetCDF y HDF5 son para datos multidimensionales (tiempo, profundidad, etc).

HACER

- Mostrar slides 14-15: Formatos raster
- Explicar ventajas de COG con diagrama
- Mencionar compresión: LZW (sin pérdida) vs JPEG (con pérdida)

NOTA

COG permite: - Streaming desde S3/HTTP - Overviews (pirámides) para zoom rápido - Tiles internos para acceso parcial

3.5 Atributos y Geometrías

[15 min]

3.5.1. Operaciones Espaciales Fundamentales

[5 min]

DECIR

Las operaciones espaciales son el corazón del análisis geoespacial. Vamos a ver las operaciones geométricas básicas y las relaciones espaciales que podemos evaluar.
Buffer crea zonas de influencia, overlay combina capas, dissolve agrupa geometrías. Estas son herramientas fundamentales que usarán constantemente.

HACER

- Mostrar slide 18: Operaciones Espaciales Fundamentales
- Dibujar en pizarra: buffer, intersección, unión, diferencia
- Ejecutar código de operaciones:

```python
from shapely.ops import unary_union
import geopandas as gpd

# Buffer - zona de influencia
zonas_influencia = gdf.buffer(100)  # 100 metros

# Intersección - área común
interseccion = gdf1.overlay(gdf2, how='intersection')

# Unión - combinar geometrías
union_total = unary_union(gdf.geometry)

# Diferencia - quitar área
diferencia = gdf1.overlay(gdf2, how='difference')

# Dissolve - agrupar por atributo
regiones = comunas.dissolve(by='REGION')

# Consultas espaciales con predicados
dentro = puntos[puntos.within(poligono)]
cruzan = lineas[lineas.crosses(limite)]
tocan = poligonos[poligonos.touches(otro_poligono)]
```

ALERTA

Cuidado con las unidades: si el CRS está en grados, buffer(100) significa 100 grados, no metros. Siempre verificar las unidades del CRS.

3.5.2. Indexación Espacial R-tree

[3 min]

DECIR

El R-tree es una estructura de datos fundamental para hacer eficientes las consultas espaciales. Sin índice espacial, encontrar qué puntos están dentro de un polígono requiere verificar TODOS los puntos. Con R-tree, solo verificamos los candidatos probables.
Es como un índice en base de datos, pero para geometrías. Reduce la complejidad de $O(n^2)$ a $O(n \log n)$.

HACER

- Mostrar slide 19: Indexación Espacial
- Explicar el diagrama del R-tree
- Mencionar: GeoPandas crea índices automáticamente para spatial joins

NOTA

El R-tree agrupa geometrías cercanas en rectángulos (bounding boxes) jerárquicos. Primero verifica si los bounding boxes se intersectan (rápido), solo entonces verifica la geometría real (lento).

3.5.3. Validación y Limpieza Geométrica

[3 min]

DECIR

Los datos geoespaciales del mundo real suelen tener problemas: polígonos que se cruzan a sí mismos, gaps microscópicos entre parcelas, duplicados. Es crucial validar y limpiar las geometrías antes del análisis.
El truco buffer(0) es mágico: arregla muchos problemas topológicos automáticamente.

HACER

- Mostrar slide 20: Validación y Limpieza
- Ejecutar validación rápida:

```
# Verificar validez
print(f"Geometrías inválidas: {(~gdf.geometry.is_valid).sum()}")

# Corregir con buffer(0) - el truco mágico
gdf['geometry'] = gdf.geometry.buffer(0)

# Eliminar slivers (polígonos microscópicos)
gdf = gdf[gdf.geometry.area > 0.001]
```

3.5.4. Sistemas de Referencia (CRS)

[4 min]

DECIR

El CRS es crítico. Es la diferencia entre que tus mapas se alineen perfectamente o estén desplazados 200 metros. Chile usa principalmente UTM zona 19S, pero los datos web vienen en WGS84.
Siempre, SIEMPRE verificar el CRS antes de hacer análisis. Es el error número 1 en geoespacial.

HACER

- Mostrar slide 21: CRS
- Mostrar códigos EPSG para Chile
- Demostrar reproyección:

```python
# Verificar CRS
print(f"CRS actual: {gdf.crs}")

# Reproyectar a UTM 19S (métrico)
gdf_utm = gdf.to_crs('EPSG:32719')

# Ahora las unidades son metros
area_m2 = gdf_utm.geometry.area
buffer_500m = gdf_utm.buffer(500)  # 500 metros reales
```

```python
# Crear datos de ejemplo
ciudades = gpd.GeoDataFrame({
    'ciudad': ['Santiago', 'Valparaíso', 'Concepción'],
    'poblacion': [5000000, 300000, 220000],
    'geometry': [
        Point(-70.65, -33.45),
        Point(-71.62, -33.04),
        Point(-73.05, -36.82)
    ]
})

# Buffer de 50km alrededor de cada ciudad
ciudades['area_influencia'] = ciudades.buffer(0.5)

# Join espacial (encontrar puntos en polígonos)
# Intersección de geometrías
# Union de todas las áreas
area_total = ciudades['area_influencia'].unary_union

print(f" rea  total cubierta: {area_total.area}")
```

ALERTA

Las operaciones espaciales pueden ser costosas computacionalmente. Siempre usar índices espaciales (R-tree) para datasets grandes.

3.6 Machine Learning Espacial

[10 min]

DECIR

El Machine Learning espacial tiene particularidades importantes. La primera ley de geografía de Tobler dice: "Todo está relacionado con todo, pero las cosas cercanas están más relacionadas". Esto viola el supuesto de independencia del ML tradicional.
Vamos a ver cómo crear features espaciales y por qué necesitamos cross-validation espacial especial.

HACER

- Mostrar slide 22: Machine Learning Espacial
- Explicar autocorrelación espacial con ejemplo de precios de casas
- Ejecutar código de feature engineering espacial:

```python
from sklearn.ensemble import RandomForestRegressor
import geopandas as gpd
from libpysal.weights import KNN

# Feature engineering espacial
gdf['dist_centro'] = gdf.distance(centro)
gdf['dist_metro'] = gdf.distance(metro_cercano)
gdf['dist_parque'] = gdf.distance(parque_cercano)

# Contar vecinos en radio de 500m
gdf['n_vecinos_500m'] = gdf.buffer(500).apply(
    lambda x: puntos.within(x).sum()
)

# Lag espacial - promedio de vecinos
w = KNN.from_dataframe(gdf, k=5)
gdf['precio_lag'] = w.lag(gdf['precio'])

# Modelo con features espaciales
features = ['area', 'habitaciones', 'dist_centro',
            'dist_metro', 'n_vecinos_500m', 'precio_lag']
X = gdf[features]
y = gdf['precio']

# Train con spatial cross-validation
from sklearn.model_selection import KFold
# NO usar KFold regular! Usar bloques espaciales

modelo = RandomForestRegressor(n_estimators=100)
modelo.fit(X, y)

# Feature importance
importancia = pd.DataFrame({
    'feature': features,
    'importance': modelo.feature_importances_
}).sort_values('importance', ascending=False)

print(importancia)
```

ALERTA

Error común: usar train_test_split regular ignora la autocorrelación espacial. Los resultados serán demasiado optimistas. Siempre usar validación por bloques espaciales o leave-one-region-out.

NOTA

Features espaciales típicas que mejoran modelos:

- Distancia a puntos de interés
- Densidad de features en buffers
- Lag espacial (valor promedio de vecinos)
- Coordenadas X, Y (capturan tendencias espaciales)
- Índices de accesibilidad

DECIR

Las aplicaciones son enormes: predicción de precios inmobiliarios, interpolación de datos climáticos, clasificación de cobertura del suelo, detección de hot spots criminales. Todo mejora con features espaciales bien diseñadas.

3.7 Integración Vector-Raster

[5 min]

DECIR

La verdadera potencia viene de combinar vector y raster. Podemos extraer valores de raster en puntos, calcular estadísticas zonales por polígono, o usar polígonos como máscaras para análisis raster.
Este es el tipo de análisis que harán en proyectos reales.

HACER

- Mostrar slide 23: Resumen Vector vs Raster
- Mostrar slide 24: Ejercicio Integrador
- Ejecutar ejemplo de zonal statistics:

```python
import rasterstats
import geopandas as gpd

# Estadísticas zonales - temperatura promedio por comuna
comunas = gpd.read_file('comunas.shp')
stats = rasterstats.zonal_stats(
    comunas.geometry,
    'temperatura.tif',
    stats=['mean', 'min', 'max', 'std']
)

# Agregar resultados al GeoDataFrame
for stat in ['mean', 'min', 'max', 'std']:
    comunas[f'temp_{stat}'] = [s[stat] for s in stats]

# Comuna más calurosa
mas_caliente = comunas.loc[comunas['temp_mean'].idxmax()]
print(f"Comuna más calurosa: {mas_caliente['nombre']}")
print(f"Temperatura promedio: {mas_caliente['temp_mean']:.1f} C ")

```

```python
# Extraer valores en puntos
from rasterio import sample
puntos = gpd.read_file('estaciones.geojson')
coords = [(p.x, p.y) for p in puntos.geometry]

with rasterio.open('elevacion.tif') as src:
    puntos['elevacion'] = [val[0] for val in sample(src, coords)]
```

NOTA

La biblioteca 'rasterstats' es excelente para estadísticas zonales. Para extracciones puntuales, 'rasterio.sample' es más eficiente que 'rasterstats.point$_{q}uery$'.

3.8 Cierre y Vista al Laboratorio

[5 min]

DECIR

Hoy cubrimos los fundamentos de datos geoespaciales. Vimos cómo el mundo real se representa en dos modelos complementarios: vectorial para objetos discretos, raster para fenómenos continuos.
En el jueves aplicaremos estos conceptos en el Laboratorio 1. Trabajaremos con datos reales de Santiago: comunas, estaciones de metro, e imágenes satelitales.

HACER

- Mostrar slide 29: Actividades prácticas
- Mostrar slide 30: Cierre
- Abrir espacio para preguntas
- Recordar: Lab 1 el jueves, traer laptops

NOTA

Tareas sugeridas para reforzar: 1. Descargar datos de su comuna desde IDE Chile 2. Instalar QGIS para visualización 3. Explorar OpenStreetMap para su barrio

4 Demostraciones y Ejercicios

4.1 Demo 1: Análisis de Comunas de Santiago

```python
import geopandas as gpd
import matplotlib.pyplot as plt

# Cargar datos
comunas = gpd.read_file('comunas_santiago.shp')

# Exploración básica
print(f"Número de comunas: {len(comunas)}")
print(f"CRS: {comunas.crs}")
print(comunas.columns.tolist())

# Análisis
comunas['area_km2'] = comunas.geometry.area * 111 * 111
comuna_mayor = comunas.loc[comunas['area_km2'].idxmax()]
print(f"Comuna más grande: {comuna_mayor['nombre']}")

# Visualización temática
fig, ax = plt.subplots(figsize=(10, 10))
comunas.plot(column='poblacion',
             cmap='YlOrRd',
             legend=True,
             ax=ax)
ax.set_title('Población por Comuna')
plt.show()
```

4.2 Demo 2: Cálculo de NDVI Real

```python
import rasterio
import numpy as np

# Abrir bandas Sentinel-2
with rasterio.open('B04_red.tif') as red_src:
    red = red_src.read(1).astype(float)
    profile = red_src.profile

with rasterio.open('B08_nir.tif') as nir_src:
    nir = nir_src.read(1).astype(float)

# Calcular NDVI
ndvi = (nir - red) / (nir + red + 1e-10)

# Guardar resultado
profile.update(dtype=rasterio.float32, count=1)
with rasterio.open('ndvi_output.tif', 'w', **profile) as dst:
    dst.write(ndvi.astype(np.float32), 1)

# Estadísticas
print(f"NDVI mínimo: {ndvi.min():.3f}")
print(f"NDVI máximo: {ndvi.max():.3f}")
print(f"NDVI promedio: {ndvi.mean():.3f}")
```

4.3 Demo 3: Operaciones Espaciales Complejas

```python
# Análisis multicriterio para ubicación óptima
import geopandas as gpd
```

```python
from shapely.ops import unary_union

# Datos
colegios = gpd.read_file('colegios.geojson')
vias = gpd.read_file('vias_principales.shp')
manzanas = gpd.read_file('manzanas.shp')

# Criterio 1: Alejado de colegios existentes (500m)
buffer_colegios = colegios.geometry.buffer(500/111000)  # grados
zona_exclusion = unary_union(buffer_colegios)

# Criterio 2: Cerca de vías principales (100m)
buffer_vias = vias.geometry.buffer(100/111000)
zona_acceso = unary_union(buffer_vias)

# Encontrar manzanas candidatas
candidatas = manzanas.copy()
candidatas = candidatas[~candidatas.intersects(zona_exclusion)]
candidatas = candidatas[candidatas.intersects(zona_acceso)]

print(f"Manzanas candidatas: {len(candidatas)}")

# Ranking por densidad poblacional
candidatas['score'] = candidatas['poblacion'] / candidatas.geometry.area
mejor_ubicacion = candidatas.loc[candidatas['score'].idxmax()]

print(f"Mejor ubicación: Manzana {mejor_ubicacion['id']}")
```

5 Problemas Comunes y Soluciones

5.1 Errores Frecuentes

1. **CRS no coincidentes**
   - Síntoma: Capas no se alinean al visualizar
   - Solución: `gdf.to_crs('EPSG:4326')`
2. **Geometrías inválidas**
   - Síntoma: Error en operaciones espaciales
   - Solución: `gdf.geometry = gdf.geometry.buffer(0)`
3. **Memoria insuficiente con rasters grandes**
   - Síntoma: MemoryError o kernel crash
   - Solución: Leer por ventanas (windows) o usar Dask
4. **Shapefile con caracteres especiales**
   - Síntoma: Nombres con símbolos raros
   - Solución: `encoding='latin1'` o `'utf-8'`

5.2 Preguntas Frecuentes de Estudiantes

P: ¿Cuándo usar vector vs raster?
R: Vector para objetos discretos con límites definidos (parcelas, caminos). Raster para fenómenos continuos (temperatura, elevación) o imágenes.

P: ¿Por qué mi shapefile tiene varios archivos?
R: Es el diseño del formato desde 1990s. El .shp tiene geometrías, .dbf los atributos, .shx el índice. Todos son necesarios.

P: ¿Cómo elijo la resolución correcta?
R: Depende del fenómeno estudiado y la escala de análisis. Para vegetación urbana: 1-5m. Para bosques regionales: 10-30m.

P: ¿Qué es mejor: GeoJSON o GeoPackage?
R: GeoJSON para compartir en web y datasets pequeños. GeoPackage para producción, datasets grandes, y cuando necesitas múltiples capas.

P: ¿Por qué el área calculada no coincide con el valor oficial?
R: Probablemente están en coordenadas geográficas (grados). Reproyectar a UTM o usar `geodesic=True` en GeoPandas.

6 Material Complementario

6.1 Recursos para Profundizar

- **Documentación oficial:**
  - GeoPandas: `https://geopandas.org`
  - Shapely: `https://shapely.readthedocs.io`
  - Rasterio: `https://rasterio.readthedocs.io`
- **Tutoriales recomendados:**
  - Earth Data Science (U. Colorado): Excelente para raster
  - Automating GIS Processes (U. Helsinki): Completo para vector
  - Geocomputation with Python: Libro gratuito online
- **Datasets para practicar:**
  - Natural Earth: Datos vectoriales globales
  - OpenStreetMap: Datos urbanos detallados
  - Sentinel Hub: Imágenes satelitales gratuitas
  - IDE Chile: Datos oficiales de Chile

6.2 Preparación para el Laboratorio 1

El jueves los estudiantes implementarán:

1. **Carga y exploración de datos**
   - Shapefile de comunas de Santiago
   - GeoJSON de estaciones de metro
   - Raster de temperatura o NDVI
2. **Análisis vectorial básico**
   - Cálculo de áreas y perímetros
   - Buffer analysis
   - Spatial join
3. **Análisis raster básico**
   - Lectura de bandas
   - Cálculo de índices
   - Estadísticas zonales
4. **Integración vector-raster**
   - Extracción de valores
   - Máscaras con polígonos
   - Visualización combinada

6.3 Evaluación del Aprendizaje

Indicadores de comprensión exitosa:

- ✓ Estudiantes pueden explicar diferencias vector vs raster
- ✓ Identifican cuándo usar cada formato de archivo
- ✓ Ejecutan código básico de GeoPandas sin errores
- ✓ Hacen preguntas sobre aplicaciones específicas
- ✓ Relacionan conceptos con ejemplos del mundo real

ALERTA

Si más del 30 % de la clase tiene dificultades con el código, considerar:

- Sesión extra de instalación de librerías
- Tutorial paso a paso más detallado
- Pair programming en el laboratorio

7 Anexos

7.1 Instalación de Ambiente de Trabajo

```
# Crear ambiente conda
conda create -n geo python=3.10
conda activate geo

# Instalar librerías espaciales
conda install -c conda-forge geopandas
conda install -c conda-forge rasterio
conda install -c conda-forge folium
conda install -c conda-forge jupyter

# Verificar instalación
python -c "import geopandas; print(geopandas.__version__)"
python -c "import rasterio; print(rasterio.__version__)"
```

7.2 Script de Prueba Completo

```
"""
Script de verificación de ambiente
Ejecutar antes de la clase para verificar que todo funciona
"""

import sys
print(f"Python version: {sys.version}")

try:
    import geopandas as gpd
    print(f"    GeoPandas {gpd.__version__}")
except ImportError:
    print("    GeoPandas no instalado")

try:
    import shapely
    print(f"    Shapely {shapely.__version__}")
except ImportError:
    print("    Shapely no instalado")

try:
    import rasterio
    print(f"    Rasterio {rasterio.__version__}")
except ImportError:
    print("    Rasterio no instalado")

try:
    import folium
    print(f"    Folium {folium.__version__}")
except ImportError:
    print("    Folium no instalado")

try:
    import matplotlib.pyplot as plt
    print(f"    Matplotlib instalado")
except ImportError:
    print("    Matplotlib no instalado")

# Test básico de funcionalidad
try:
    from shapely.geometry import Point
```

```
    p = Point(0, 0)
    gdf = gpd.GeoDataFrame({'geometry': [p]})
    print("    Creación de geometrías funciona")
except Exception as e:
    print(f"    Error en funcionalidad: {e}")

print("\nAmbiente listo para la clase!" if all else "Revisar instalaciones")
```

7.3 Glosario de Términos

Vector Modelo de datos que representa features como puntos, líneas y polígonos

Raster Modelo de datos basado en grilla regular de celdas/píxeles

Shapefile Formato vectorial legacy de ESRI, múltiples archivos

GeoJSON Formato vectorial basado en JSON para web

GeoPackage Formato moderno basado en SQLite

GeoTIFF Formato raster con georeferenciación embebida

COG Cloud Optimized GeoTIFF, optimizado para streaming

CRS Coordinate Reference System, sistema de referencia espacial

EPSG European Petroleum Survey Group, códigos estándar de CRS

NDVI Normalized Difference Vegetation Index

Buffer Área de influencia alrededor de una geometría

Overlay Operación que combina dos capas espaciales

Spatial Join Une atributos basándose en relación espacial