Apunte de Estudio

Clase 03: Fundamentos de Datos Geoespaciales

Guía de trabajo autónomo

Curso: Geoinformática
Prof. Francisco Parra O.

Semestre 2, 2025

NOTA IMPORTANTE

Este apunte está diseñado para trabajo autónomo.
Lean cada sección cuidadosamente, ejecuten los códigos y completen los ejercicios.
Trabajen en grupos de 2-3 personas para discutir conceptos y resolver dudas.

Índice

1 Introducción

1.1 Objetivos de Aprendizaje

Al finalizar esta sesión de estudio, serán capaces de:

- **Distinguir** entre datos vectoriales y raster
- **Crear** y manipular geometrías básicas (puntos, líneas, polígonos)
- **Ejecutar** operaciones espaciales fundamentales
- **Aplicar** transformaciones de sistemas de coordenadas
- **Implementar** análisis que combinen vector y raster
- **Desarrollar** features espaciales para machine learning

1.2 Preparación del Ambiente

Antes de comenzar, verifiquen que tienen instaladas las siguientes librerías:

```
# Verificar instalacion
import geopandas as gpd
import shapely
import rasterio
import folium
import matplotlib.pyplot as plt
print("Ambiente listo!")
```

Si falta alguna librería, instálenla con:

```
conda install -c conda-forge geopandas rasterio folium
```

2 Parte 1: Datos Vectoriales

2.1 Conceptos Fundamentales

Modelo Vectorial

El modelo vectorial representa el mundo como **objetos discretos** con geometrías precisas. Cada objeto tiene:

- **Geometría**: La forma espacial (punto, línea, polígono)
- **Atributos**: Las características descriptivas
- **Topología**: Las relaciones espaciales con otros objetos

Piensen en un mapa de Google Maps: cada restaurant es un punto, cada calle es una línea, cada manzana es un polígono.

2.2 Puntos: Tu Primera Geometría

Los puntos son la geometría más simple. Solo necesitan coordenadas X e Y.

```python
from shapely.geometry import Point
import geopandas as gpd
import matplotlib.pyplot as plt

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

# Crear multiples puntos - Estaciones de Metro Linea 1
estaciones = [
    {'nombre': 'Universidad de Santiago', 'linea': 1,
     'geometry': Point(-70.681, -33.450)},
    {'nombre': 'Estacion Central', 'linea': 1,
     'geometry': Point(-70.678, -33.452)},
    {'nombre': 'ULA', 'linea': 1,
     'geometry': Point(-70.675, -33.453)},
    {'nombre': 'Republica', 'linea': 1,
     'geometry': Point(-70.671, -33.454)}
]

# Crear GeoDataFrame
gdf_estaciones = gpd.GeoDataFrame(estaciones)
print(gdf_estaciones)

# Visualizar
fig, ax = plt.subplots(figsize=(8, 6))
gdf_estaciones.plot(ax=ax, color='red', markersize=100)
ax.set_title('Estaciones Metro Linea 1')
plt.show()
```

Ejercicio

Ejercicio 1.1: Creen un GeoDataFrame con al menos 5 lugares importantes de su comuna (colegios, plazas, centros comerciales, etc.). Incluyan como atributos: nombre, tipo, y año de construcción.

2.3 Líneas: Conectando Puntos

Las líneas son secuencias ordenadas de puntos. La dirección importa en muchos casos (ej: sentido del tráfico).

```python
from shapely.geometry import LineString

# Crear linea conectando las estaciones
coordenadas = [
    (-70.681, -33.450),  # U. de Santiago
    (-70.678, -33.452),  # Estacion Central
    (-70.675, -33.453),  # ULA
    (-70.671, -33.454)   # Republica
]

linea_metro = LineString(coordenadas)

# Propiedades de la linea
print(f"Longitud: {linea_metro.length:.4f} grados")
print(f"Numero de vertices: {len(linea_metro.coords)}")
print(f"Es simple (no se cruza)?: {linea_metro.is_simple}")
print(f"Es cerrada?: {linea_metro.is_ring}")

# Buffer - zona de influencia de 100 metros
# OJO: Como estamos en grados, 0.001 grados ~ 111 metros
zona_influencia = linea_metro.buffer(0.001)
print(f"Area de influencia: {zona_influencia.area:.6f} grados^2")
```

Importante

Unidades: Cuando trabajamos con coordenadas geográficas (lat/lon), las unidades están en grados. Para distancias reales en metros, necesitamos proyectar a un sistema métrico como UTM.

2.4 Polígonos: Definiendo Áreas

Los polígonos son áreas cerradas. El primer y último punto deben ser iguales.

```python
from shapely.geometry import Polygon

# Campus USACH simplificado (rectangulo)
campus = Polygon([
    (-70.683, -33.448),  # Esquina NO
    (-70.679, -33.448),  # Esquina NE
    (-70.679, -33.452),  # Esquina SE
    (-70.683, -33.452),  # Esquina SO
    (-70.683, -33.448)   # Cierre (igual al primero)
])

print(f"Area: {campus.area:.6f} grados^2")
print(f"Perimetro: {campus.length:.4f} grados")

# Verificar relaciones espaciales
print(f"USACH esta dentro del campus? {campus.contains(usach)}")
print(f"La linea de metro cruza el campus? {campus.intersects(linea_metro)}")

# Poligono con hueco (donut)
exterior = [(0, 0), (10, 0), (10, 10), (0, 10), (0, 0)]
interior = [(2, 2), (8, 2), (8, 8), (2, 8), (2, 2)]
donut = Polygon(exterior, [interior])
print(f"Area del donut: {donut.area}")  # Sera 100 - 36 = 64
```

Ejercicio

Ejercicio 1.2:

1. Creen un polígono que represente su manzana o barrio
2. Verifiquen cuáles de los puntos del Ejercicio 1.1 están dentro
3. Calculen el área y perímetro

3 Parte 2: Operaciones Espaciales

3.1 Operaciones Geométricas Básicas

Operaciones Fundamentales

- **Buffer**: Zona de influencia alrededor de una geometría
- **Intersección**: Área común entre geometrías
- **Unión**: Combinación de geometrías
- **Diferencia**: Resta de una geometría de otra
- **Dissolve**: Fusión de geometrías por atributo

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

# Crear dos poligonos que se solapan
poly1 = Polygon([(0, 0), (2, 0), (2, 2), (0, 2), (0, 0)])
poly2 = Polygon([(1, 1), (3, 1), (3, 3), (1, 3), (1, 1)])

# Operaciones geometricas
interseccion = poly1.intersection(poly2)
union = poly1.union(poly2)
diferencia1 = poly1.difference(poly2)  # poly1 - poly2
diferencia2 = poly2.difference(poly1)  # poly2 - poly1

print(f"Area poly1: {poly1.area}")
print(f"Area poly2: {poly2.area}")
print(f"Area interseccion: {interseccion.area}")
print(f"Area union: {union.area}")
print(f"Area diferencia1: {diferencia1.area}")
print(f"Area diferencia2: {diferencia2.area}")

# Visualizar
fig, axes = plt.subplots(2, 3, figsize=(12, 8))

# Poligonos originales
gpd.GeoSeries([poly1]).plot(ax=axes[0,0], color='blue', alpha=0.5)
gpd.GeoSeries([poly2]).plot(ax=axes[0,0], color='red', alpha=0.5)
axes[0,0].set_title('Originales')

# Interseccion
gpd.GeoSeries([interseccion]).plot(ax=axes[0,1], color='purple')
axes[0,1].set_title('Interseccion')

# Union
gpd.GeoSeries([union]).plot(ax=axes[0,2], color='green')
axes[0,2].set_title('Union')

# Diferencias
gpd.GeoSeries([diferencia1]).plot(ax=axes[1,0], color='blue')
axes[1,0].set_title('Poly1 - Poly2')

gpd.GeoSeries([diferencia2]).plot(ax=axes[1,1], color='red')
axes[1,1].set_title('Poly2 - Poly1')

# Buffer
buffer = poly1.buffer(0.5)
gpd.GeoSeries([buffer]).plot(ax=axes[1,2], color='orange')
```

```
gpd.GeoSeries([poly1]).plot(ax=axes[1,2], color='blue')
axes[1,2].set_title('Buffer 0.5')

plt.tight_layout()
plt.show()
```

3.2 Predicados Espaciales

Los predicados espaciales evalúan relaciones entre geometrías y retornan True/False.

```
# Crear geometrias de ejemplo
punto = Point(1.5, 1.5)
linea = LineString([(0, 0), (3, 3)])
poligono = Polygon([(0, 0), (2, 0), (2, 2), (0, 2), (0, 0)])

# Predicados espaciales
print("=== Relaciones Espaciales ===")
print(f"Punto dentro del poligono? {punto.within(poligono)}")
print(f"Poligono contiene punto? {poligono.contains(punto)}")
print(f"Linea cruza poligono? {linea.crosses(poligono)}")
print(f"Linea intersecta poligono? {linea.intersects(poligono)}")
print(f"Poligono toca linea? {poligono.touches(linea)}")
print(f"Distancia punto a poligono: {punto.distance(poligono):.4f}")

# Aplicacion con GeoDataFrame
puntos_random = gpd.GeoSeries([
    Point(0.5, 0.5), Point(1.5, 1.5), Point(2.5, 2.5),
    Point(0, 3), Point(3, 0)
])

# Filtrar puntos dentro del poligono
dentro = puntos_random[puntos_random.within(poligono)]
fuera = puntos_random[~puntos_random.within(poligono)]

print(f"\nPuntos dentro: {len(dentro)}")
print(f"Puntos fuera: {len(fuera)}")
```

Ejercicio

Ejercicio 2.1:

1. Creen un buffer de 500 metros alrededor de las estaciones de metro
2. Encuentren la intersección de todas las zonas de influencia
3. Identifiquen qué puntos del Ejercicio 1.1 están dentro de estas zonas

3.3 Spatial Joins

Los spatial joins combinan datos basándose en relaciones espaciales.

```
# Crear datos de ejemplo
# Comunas (poligonos)
comunas = gpd.GeoDataFrame({
    'comuna': ['Santiago', 'Providencia', 'Las Condes'],
    'poblacion': [400000, 150000, 300000],
    'geometry': [
        Polygon([(-70.65, -33.45), (-70.64, -33.45),
                 (-70.64, -33.44), (-70.65, -33.44), (-70.65, -33.45)]),
        Polygon([(-70.64, -33.44), (-70.63, -33.44),
                 (-70.63, -33.43), (-70.64, -33.43), (-70.64, -33.44)]),
```

```
        Polygon([(-70.63, -33.43), (-70.62, -33.43),
                 (-70.62, -33.42), (-70.63, -33.42), (-70.63, -33.43)])
    ]
})

# Colegios (puntos)
colegios = gpd.GeoDataFrame({
    'nombre': ['Colegio A', 'Colegio B', 'Colegio C', 'Colegio D'],
    'tipo': ['Municipal', 'Particular', 'Subvencionado', 'Municipal'],
    'geometry': [
        Point(-70.645, -33.445),
        Point(-70.635, -33.435),
        Point(-70.625, -33.425),
        Point(-70.648, -33.448)
    ]
})

# Spatial join - encontrar en que comuna esta cada colegio
colegios_con_comuna = gpd.sjoin(colegios, comunas,
                                predicate='within', how='left')

print(colegios_con_comuna[['nombre', 'tipo', 'comuna', 'poblacion']])

# Contar colegios por comuna
colegios_por_comuna = colegios_con_comuna.groupby('comuna').size()
print("\nColegios por comuna:")
print(colegios_por_comuna)
```

4 Parte 3: Datos Raster

4.1 Conceptos Fundamentales

Modelo Raster

El modelo raster divide el espacio en una **grilla regular** de celdas (píxeles). Cada celda tiene un valor que representa:

- Elevación (DEM - Digital Elevation Model)
- Temperatura
- Reflectancia espectral (imágenes satelitales)
- Clasificación de cobertura del suelo
- Cualquier variable continua en el espacio

4.2 Creando y Manipulando Rasters

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

# Crear un raster sintetico de elevacion
# Simular una montana con ruido
x = np.linspace(-5, 5, 100)
y = np.linspace(-5, 5, 100)
X, Y = np.meshgrid(x, y)

# Formula de montana gaussiana + ruido
elevacion = 1000 * np.exp(-(X**2 + Y**2)/10) + \
            np.random.normal(0, 20, (100, 100))

# Visualizar
fig, axes = plt.subplots(1, 3, figsize=(15, 4))

# Imagen basica
im1 = axes[0].imshow(elevacion, cmap='terrain')
axes[0].set_title('Elevacion (m)')
plt.colorbar(im1, ax=axes[0])

# Contornos
contours = axes[1].contour(X, Y, elevacion, levels=10, colors='black')
axes[1].clabel(contours, inline=True, fontsize=8)
axes[1].set_title('Curvas de Nivel')

# 3D
from mpl_toolkits.mplot3d import Axes3D
ax3d = fig.add_subplot(133, projection='3d')
ax3d.plot_surface(X, Y, elevacion, cmap='terrain', alpha=0.8)
ax3d.set_title('Vista 3D')
ax3d.set_xlabel('X')
ax3d.set_ylabel('Y')
ax3d.set_zlabel('Elevacion (m)')

plt.tight_layout()
plt.show()

# Estadisticas del raster
```

```
print(f"Elevacion minima: {elevacion.min():.2f} m")
print(f"Elevacion maxima: {elevacion.max():.2f} m")
print(f"Elevacion promedio: {elevacion.mean():.2f} m")
print(f"Desviacion estandar: {elevacion.std():.2f} m")
```

4.3 Resolución y Remuestreo

La resolución determina el nivel de detalle. Mayor resolución = más detalle pero archivos más grandes.

```
# Diferentes resoluciones
from scipy import ndimage

# Raster original 100x100
original = elevacion

# Reducir resolucion (downsampling)
baja_res_10x10 = ndimage.zoom(original, 0.1, order=1)
media_res_50x50 = ndimage.zoom(original, 0.5, order=1)

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

axes[0].imshow(original, cmap='terrain')
axes[0].set_title(f'Alta Res: {original.shape[0]}x{original.shape[1]}')

axes[1].imshow(media_res_50x50, cmap='terrain')
axes[1].set_title(f'Media Res: {media_res_50x50.shape[0]}x{media_res_50x50.shape
    [1]}')

axes[2].imshow(baja_res_10x10, cmap='terrain')
axes[2].set_title(f'Baja Res: {baja_res_10x10.shape[0]}x{baja_res_10x10.shape
    [1]}')

plt.tight_layout()
plt.show()

# Tamaño en memoria
print(f"Tamaño alta res: {original.nbytes / 1024:.2f} KB")
print(f"Tamaño media res: {media_res_50x50.nbytes / 1024:.2f} KB")
print(f"Tamaño baja res: {baja_res_10x10.nbytes / 1024:.2f} KB")
```

Importante

Trade-off Resolución vs Tamaño:

- Doblar la resolución = 4x más píxeles = 4x más memoria
- Para análisis regional: 30m (Landsat) o 10m (Sentinel)
- Para análisis urbano: 1-5m
- Para inspección detallada: ¡1m

4.4 Índices Espectrales: NDVI

El NDVI (Normalized Difference Vegetation Index) es el índice más usado para detectar vegetación.

```
# Simular bandas espectrales
```

```
np.random.seed(42)

# Banda roja (Band 4 en Landsat/Sentinel)
# Vegetacion absorbe mucho rojo
red = np.random.uniform(0.05, 0.15, (100, 100))

# Banda infrarrojo cercano (Band 5/8)
# Vegetacion refleja mucho NIR
nir = np.random.uniform(0.30, 0.50, (100, 100))

# Agregar patrones espaciales (parches de vegetacion)
for i in range(5):
    x, y = np.random.randint(20, 80, 2)
    r = 15
    mask = (X - x)**2 + (Y - y)**2 < r**2
    red[mask] *= 0.5  # Menos rojo en vegetacion
    nir[mask] *= 1.5  # Mas NIR en vegetacion

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

# Clasificar
agua = ndvi < 0
suelo = (ndvi >= 0) & (ndvi < 0.2)
vegetacion_baja = (ndvi >= 0.2) & (ndvi < 0.4)
vegetacion_alta = ndvi >= 0.4

# Visualizar
fig, axes = plt.subplots(2, 3, figsize=(12, 8))

# Bandas originales
axes[0,0].imshow(red, cmap='Reds')
axes[0,0].set_title('Banda Roja')

axes[0,1].imshow(nir, cmap='YlGn')
axes[0,1].set_title('Banda NIR')

# NDVI
im = axes[0,2].imshow(ndvi, cmap='RdYlGn', vmin=-1, vmax=1)
axes[0,2].set_title('NDVI')
plt.colorbar(im, ax=axes[0,2])

# Clasificacion
clasificacion = np.zeros_like(ndvi)
clasificacion[agua] = 0
clasificacion[suelo] = 1
clasificacion[vegetacion_baja] = 2
clasificacion[vegetacion_alta] = 3

from matplotlib.colors import ListedColormap
colors = ['blue', 'brown', 'lightgreen', 'darkgreen']
cmap = ListedColormap(colors)

axes[1,0].imshow(clasificacion, cmap=cmap, vmin=0, vmax=3)
axes[1,0].set_title('Clasificacion')

# Histograma NDVI
axes[1,1].hist(ndvi.flatten(), bins=50, color='green', alpha=0.7)
axes[1,1].axvline(0, color='blue', linestyle='--', label='Agua')
axes[1,1].axvline(0.2, color='brown', linestyle='--', label='Suelo')
axes[1,1].axvline(0.4, color='darkgreen', linestyle='--', label='Veg. Alta')
axes[1,1].set_xlabel('NDVI')
axes[1,1].set_ylabel('Frecuencia')
```

```
axes[1,1].legend()
axes[1,1].set_title('Distribucion NDVI')

# Estadisticas por clase
axes[1,2].bar(['Agua', 'Suelo', 'Veg.Baja', 'Veg.Alta'],
              [agua.sum(), suelo.sum(), vegetacion_baja.sum(),
               vegetacion_alta.sum()],
              color=colors)
axes[1,2].set_ylabel('Pixeles')
axes[1,2].set_title('Pixeles por Clase')

plt.tight_layout()
plt.show()
```

Ejercicio

Ejercicio 3.1:

1. Creen un raster sintético que represente temperatura (hint: usen gradientes)
2. Apliquen diferentes niveles de suavizado (smoothing)
3. Clasifiquen en categorías: Muy frío, Frío, Templado, Cálido, Muy cálido

5 Parte 4: Sistemas de Coordenadas (CRS)

5.1 Por Qué Importa el CRS

Sistema de Referencia de Coordenadas

El CRS define cómo las coordenadas se relacionan con lugares en la Tierra. Incluye:

- **Datum**: Modelo matemático de la forma de la Tierra
- **Proyección**: Método para aplanar la superficie curva
- **Unidades**: Grados (geográficas) o metros (proyectadas)

Para Chile:

- WGS84 (EPSG:4326): Coordenadas geográficas, usado por GPS
- UTM Zona 19S (EPSG:32719): Sistema métrico para Chile continental
- SIRGAS-Chile (EPSG:5361): Sistema oficial de Chile

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

# Crear punto en coordenadas geograficas (grados)
santiago_geo = gpd.GeoSeries([Point(-70.65, -33.45)],
                             crs='EPSG:4326')

print("=== Coordenadas Geograficas (WGS84) ===")
print(f"CRS: {santiago_geo.crs}")
print(f"Coordenadas: {santiago_geo[0].x:.3f}, {santiago_geo[0].y:.3f}")

# Proyectar a UTM 19S (metros)
santiago_utm = santiago_geo.to_crs('EPSG:32719')

print("\n=== Coordenadas UTM 19S ===")
print(f"CRS: {santiago_utm.crs}")
print(f"Coordenadas: {santiago_utm[0].x:.0f} E, {santiago_utm[0].y:.0f} N")

# Diferencia en calculos de distancia
punto1_geo = Point(-70.65, -33.45)
punto2_geo = Point(-70.64, -33.44)

# Distancia en grados (incorrecto para distancia real)
dist_grados = punto1_geo.distance(punto2_geo)
print(f"\nDistancia en grados: {dist_grados:.4f}")

# Convertir a GeoSeries para proyectar
puntos_geo = gpd.GeoSeries([punto1_geo, punto2_geo], crs='EPSG:4326')
puntos_utm = puntos_geo.to_crs('EPSG:32719')

# Distancia en metros (correcto)
dist_metros = puntos_utm[0].distance(puntos_utm[1])
print(f"Distancia real en metros: {dist_metros:.2f} m")

# Factor de conversion aproximado en Santiago
factor = dist_metros / dist_grados
print(f"\nFactor conversion (m/grado) en Santiago: {factor:.0f}")
```

Importante

Regla de Oro:

- Para visualización y almacenamiento: WGS84 (EPSG:4326)
- Para cálculos de distancia y área: UTM local
- SIEMPRE verificar el CRS antes de hacer análisis
- NUNCA mezclar datos con diferentes CRS sin reproyectar

Ejercicio

Ejercicio 4.1:

1. Creen un polígono (cuadrado de 1km x 1km) en coordenadas UTM
2. Proyéctenlo a WGS84
3. Comparen las áreas calculadas en ambos sistemas
4. ¿Por qué son diferentes?

6 Parte 5: Machine Learning Espacial

6.1 Feature Engineering Espacial

El ML espacial requiere features especiales que capturen la naturaleza espacial de los datos.

Primera Ley de Tobler

"Todo está relacionado con todo lo demás, pero las cosas cercanas están más relacionadas que las cosas distantes."
Esto significa que:

- Los valores cercanos tienden a ser similares (autocorrelación espacial)
- No podemos usar train_test_split aleatorio
- Necesitamos features que capturen contexto espacial

```python
import pandas as pd
import geopandas as gpd
from shapely.geometry import Point
import numpy as np
from sklearn.ensemble import RandomForestRegressor
from sklearn.model_selection import train_test_split

# Crear dataset sintetico de propiedades
np.random.seed(42)
n_propiedades = 200

# Generar propiedades aleatorias en Santiago
propiedades = {
    'x': np.random.uniform(-70.70, -70.60, n_propiedades),
    'y': np.random.uniform(-33.50, -33.40, n_propiedades),
    'habitaciones': np.random.randint(1, 6, n_propiedades),
    'banos': np.random.randint(1, 4, n_propiedades),
    'superficie': np.random.uniform(40, 300, n_propiedades),
    'ano': np.random.randint(1960, 2024, n_propiedades)
}

# Crear GeoDataFrame
gdf = gpd.GeoDataFrame(propiedades)
gdf['geometry'] = [Point(x, y) for x, y in zip(gdf['x'], gdf['y'])]
gdf = gdf.set_crs('EPSG:4326')

# Precio base (funcion de caracteristicas + ubicacion)
# Mas caro hacia el nororiente (Las Condes, Vitacura)
gdf['precio_base'] = (
    gdf['habitaciones'] * 50 +
    gdf['banos'] * 30 +
    gdf['superficie'] * 0.5 +
    (2024 - gdf['ano']) * (-0.5) +  # Depreciacion
    (gdf['x'] + 70.65) * 5000 +      # Mas caro hacia el este
    (-gdf['y'] - 33.45) * 3000       # Mas caro hacia el norte
)

# Agregar ruido
gdf['precio'] = gdf['precio_base'] + np.random.normal(0, 50, n_propiedades)

# FEATURE ENGINEERING ESPACIAL
# 1. Distancia a punto central (Plaza de Armas)
plaza_armas = Point(-70.65, -33.44)
```

```
gdf['dist_centro'] = gdf.geometry.distance(plaza_armas) * 111  # km aprox

# 2. Distancia a punto premium (Las Condes)
las_condes = Point(-70.58, -33.41)
gdf['dist_premium'] = gdf.geometry.distance(las_condes) * 111

# 3. Densidad de vecinos
# Contar propiedades en radio de 1km
gdf['vecinos_1km'] = gdf.geometry.apply(
    lambda x: sum(gdf.geometry.distance(x) < 0.009)  # ~1km
)

# 4. Precio promedio de los 5 vecinos mas cercanos (lag espacial)
from sklearn.neighbors import NearestNeighbors

coords = np.column_stack([gdf['x'], gdf['y']])
nbrs = NearestNeighbors(n_neighbors=6, algorithm='ball_tree').fit(coords)
distances, indices = nbrs.kneighbors(coords)

# Calcular precio promedio de vecinos (excluyendo el punto mismo)
precio_vecinos = []
for idx in indices:
    vecinos_idx = idx[1:6]  # Excluir el primero (si mismo)
    precio_vecinos.append(gdf.iloc[vecinos_idx]['precio'].mean())

gdf['precio_lag'] = precio_vecinos

# 5. Cuadrante (feature categorica)
gdf['cuadrante'] = 'SO'  # Default
gdf.loc[(gdf['x'] > -70.65) & (gdf['y'] > -33.45), 'cuadrante'] = 'NE'
gdf.loc[(gdf['x'] <= -70.65) & (gdf['y'] > -33.45), 'cuadrante'] = 'NO'
gdf.loc[(gdf['x'] > -70.65) & (gdf['y'] <= -33.45), 'cuadrante'] = 'SE'

# One-hot encoding del cuadrante
gdf = pd.get_dummies(gdf, columns=['cuadrante'], prefix='zona')

print("=== Features Creadas ===")
print(gdf[['dist_centro', 'dist_premium', 'vecinos_1km',
           'precio_lag']].describe())
```

6.2 Modelo con Validación Espacial

```
# Preparar features y target
feature_cols = ['habitaciones', 'banos', 'superficie', 'ano',
                'dist_centro', 'dist_premium', 'vecinos_1km',
                'precio_lag', 'zona_NE', 'zona_NO', 'zona_SE', 'zona_SO']

X = gdf[feature_cols]
y = gdf['precio']

# SPLIT INCORRECTO (aleatorio) - NO USAR
X_train_bad, X_test_bad, y_train_bad, y_test_bad = train_test_split(
    X, y, test_size=0.3, random_state=42
)

# SPLIT CORRECTO (espacial) - dividir por zona
train_mask = gdf['x'] < -70.64  # Zona oeste para train
test_mask = ~train_mask         # Zona este para test

X_train = X[train_mask]
X_test = X[test_mask]
y_train = y[train_mask]
```

```
y_test = y[test_mask]

print(f"Train: {len(X_train)} propiedades (zona oeste)")
print(f"Test: {len(X_test)} propiedades (zona este)")

# Entrenar modelo
modelo = RandomForestRegressor(n_estimators=100, random_state=42)

# Modelo con split aleatorio (INCORRECTO)
modelo_bad = RandomForestRegressor(n_estimators=100, random_state=42)
modelo_bad.fit(X_train_bad, y_train_bad)
score_bad = modelo_bad.score(X_test_bad, y_test_bad)

# Modelo con split espacial (CORRECTO)
modelo.fit(X_train, y_train)
score_good = modelo.score(X_test, y_test)

print(f"\n=== Comparacion de Validacion ===")
print(f"R  con split aleatorio (INCORRECTO): {score_bad:.3f}")
print(f"R  con split espacial (CORRECTO): {score_good:.3f}")
print(f"Diferencia: {(score_bad - score_good):.3f}")
print("\nEl split aleatorio da resultados demasiado optimistas!")

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

print("\n=== Feature Importance ===")
print(importancia.head(10))

# Visualizar predicciones
gdf['prediccion'] = modelo.predict(X)
gdf['error'] = gdf['prediccion'] - gdf['precio']

fig, axes = plt.subplots(1, 3, figsize=(15, 5))

# Precio real
gdf.plot(column='precio', cmap='YlOrRd', legend=True,
         ax=axes[0], markersize=20)
axes[0].set_title('Precio Real')

# Precio predicho
gdf.plot(column='prediccion', cmap='YlOrRd', legend=True,
         ax=axes[1], markersize=20)
axes[1].set_title('Precio Predicho')

# Error
gdf.plot(column='error', cmap='RdBu', legend=True,
         ax=axes[2], markersize=20, vmin=-100, vmax=100)
axes[2].set_title('Error (Pred - Real)')

plt.tight_layout()
plt.show()
```

Importante

Features Espaciales Clave:

- Distancia a puntos de interés (metro, parques, centro)
- Densidad de features en buffers
- Lag espacial (valor promedio de vecinos)
- Coordenadas X, Y (capturan tendencias)
- Índices de accesibilidad
- Variables ambientales de rasters (temperatura, NDVI)

7 Parte 6: Integración Vector-Raster

7.1 Estadísticas Zonales

Las estadísticas zonales calculan resúmenes de raster dentro de polígonos.

```
import numpy as np
import geopandas as gpd
from shapely.geometry import Polygon
import matplotlib.pyplot as plt

# Crear raster de temperatura (100x100)
np.random.seed(42)
x = np.linspace(-70.70, -70.60, 100)
y = np.linspace(-33.50, -33.40, 100)
X, Y = np.meshgrid(x, y)

# Temperatura con gradiente + islas de calor
temperatura = 20 + (X + 70.65) * 10 + (Y + 33.45) * 5
# Agregar islas de calor urbanas
for i in range(3):
    cx, cy = np.random.uniform(-70.68, -70.62), np.random.uniform(-33.48,
    -33.42)
    isla_calor = 5 * np.exp(-((X - cx)**2 + (Y - cy)**2) / 0.001)
    temperatura += isla_calor

# Crear comunas (poligonos)
comunas = gpd.GeoDataFrame({
    'nombre': ['Comuna A', 'Comuna B', 'Comuna C', 'Comuna D'],
    'geometry': [
        Polygon([(-70.70, -33.50), (-70.65, -33.50),
                 (-70.65, -33.45), (-70.70, -33.45), (-70.70, -33.50)]),
        Polygon([(-70.65, -33.50), (-70.60, -33.50),
                 (-70.60, -33.45), (-70.65, -33.45), (-70.65, -33.50)]),
        Polygon([(-70.70, -33.45), (-70.65, -33.45),
                 (-70.65, -33.40), (-70.70, -33.40), (-70.70, -33.45)]),
        Polygon([(-70.65, -33.45), (-70.60, -33.45),
                 (-70.60, -33.40), (-70.65, -33.40), (-70.65, -33.45)])
    ]
})

# Funcion para estadisticas zonales manuales
def zonal_stats_manual(raster, x_coords, y_coords, polygon):
    """Calcula estadisticas de un raster dentro de un poligono"""
    from matplotlib.path import Path

    # Crear mascara del poligono
    xx, yy = np.meshgrid(x_coords, y_coords)
    points = np.column_stack([xx.ravel(), yy.ravel()])

    # Vertices del poligono
    vertices = list(polygon.exterior.coords)
    path = Path(vertices)

    # Puntos dentro del poligono
    mask = path.contains_points(points).reshape(raster.shape)

    # Extraer valores dentro del poligono
    valores = raster[mask]

    # Calcular estadisticas
    stats = {
        'mean': valores.mean(),
```

```
        'min': valores.min(),
        'max': valores.max(),
        'std': valores.std(),
        'count': len(valores)
    }

    return stats

# Calcular estadisticas zonales para cada comuna
for idx, comuna in comunas.iterrows():
    stats = zonal_stats_manual(temperatura, x, y, comuna.geometry)
    for key, value in stats.items():
        comunas.loc[idx, f'temp_{key}'] = value

print("=== Estadisticas Zonales de Temperatura ===")
print(comunas[['nombre', 'temp_mean', 'temp_min', 'temp_max', 'temp_std']])

# Visualizar
fig, axes = plt.subplots(1, 3, figsize=(15, 5))

# Raster de temperatura
im = axes[0].imshow(temperatura, extent=[-70.70, -70.60, -33.50, -33.40],
                    cmap='hot', origin='lower')
axes[0].set_title('Temperatura ( C )')
plt.colorbar(im, ax=axes[0])

# Comunas
comunas.plot(ax=axes[1], column='temp_mean', cmap='YlOrRd',
             legend=True, edgecolor='black')
axes[1].set_title('Temperatura Media por Comuna')

# Grafico de barras
comunas.plot(kind='bar', x='nombre', y='temp_mean',
             ax=axes[2], color='orange', legend=False)
axes[2].set_title('Comparacion de Temperatura Media')
axes[2].set_ylabel('Temperatura ( C )')
axes[2].set_xlabel('Comuna')

plt.tight_layout()
plt.show()

# Comuna mas caliente y mas fria
mas_caliente = comunas.loc[comunas['temp_mean'].idxmax()]
mas_fria = comunas.loc[comunas['temp_mean'].idxmin()]

print(f"\nComuna mas caliente: {mas_caliente['nombre']} ({mas_caliente['
    temp_mean']:.1f} C )")
print(f"Comuna mas fria: {mas_fria['nombre']} ({mas_fria['temp_mean']:.1f} C )")
```

7.2 Extracción de Valores en Puntos

```
# Crear puntos de muestreo
puntos_muestreo = gpd.GeoDataFrame({
    'id': range(20),
    'tipo': np.random.choice(['urbano', 'parque', 'residencial'], 20),
    'geometry': [Point(np.random.uniform(-70.70, -70.60),
                       np.random.uniform(-33.50, -33.40))
                 for _ in range(20)]
})

# Extraer valores de temperatura en cada punto
def extract_raster_value(point, raster, x_coords, y_coords):
```

```
    """Extrae valor de raster en un punto"""
    # Encontrar indices mas cercanos
    x_idx = np.argmin(np.abs(x_coords - point.x))
    y_idx = np.argmin(np.abs(y_coords - point.y))
    return raster[y_idx, x_idx]

# Extraer temperaturas
puntos_muestreo['temperatura'] = puntos_muestreo.geometry.apply(
    lambda pt: extract_raster_value(pt, temperatura, x, y)
)

print("=== Temperaturas en Puntos de Muestreo ===")
print(puntos_muestreo.groupby('tipo')['temperatura'].agg(['mean', 'std']))

# Visualizar
fig, ax = plt.subplots(figsize=(10, 8))

# Raster de fondo
im = ax.imshow(temperatura, extent=[-70.70, -70.60, -33.50, -33.40],
               cmap='hot', origin='lower', alpha=0.7)

# Puntos coloreados por temperatura
scatter = ax.scatter(puntos_muestreo.geometry.x,
                     puntos_muestreo.geometry.y,
                     c=puntos_muestreo['temperatura'],
                     cmap='hot', s=100, edgecolor='black')

# Comunas
comunas.boundary.plot(ax=ax, color='black', linewidth=2)

plt.colorbar(im, ax=ax, label='Temperatura ( C )')
ax.set_title('Extraccion de Temperatura en Puntos')
ax.set_xlabel('Longitud')
ax.set_ylabel('Latitud')

plt.tight_layout()
plt.show()
```

Ejercicio

Ejercicio 6.1 - Proyecto Integrador:
Analicen la ubicación óptima para un nuevo parque urbano:

1. Creen un raster de densidad poblacional (pueden simularlo)
2. Creen polígonos de manzanas/barrios
3. Identifiquen parques existentes (puntos)
4. Calculen para cada manzana:
   - Densidad poblacional promedio (del raster)
   - Distancia al parque más cercano
   - Temperatura promedio (del raster de temperatura)
5. Creen un índice de prioridad: alta densidad + lejos de parques + alta temperatura
6. Visualicen y recomienden las 3 mejores ubicaciones

8 Ejercicios Integradores

8.1 Proyecto 1: Análisis de Accesibilidad a Servicios

Ejercicio

Analicen la accesibilidad a servicios de salud en una comuna:

1. **Datos necesarios**:
   - Puntos: Centros de salud (hospitales, consultorios)
   - Líneas: Red vial principal
   - Polígonos: Manzanas censales con población
   - Raster: Densidad poblacional o población por edad
2. **Análisis a realizar**:
   - Buffer de 500m, 1km y 2km alrededor de centros de salud
   - Calcular población dentro de cada buffer
   - Identificar zonas sin cobertura
   - Proponer ubicación para nuevo centro
3. **Entregables**:
   - Mapa de cobertura actual
   - Tabla con población por nivel de accesibilidad
   - Mapa con propuesta de nuevo centro
   - Justificación basada en datos

8.2 Proyecto 2: Monitoreo de Cambios en Cobertura Vegetal

Ejercicio

Detecten cambios en vegetación urbana:

1. **Simular datos**:
   - Dos rasters NDVI (tiempo 1 y tiempo 2)
   - Polígonos de parques y plazas
   - Puntos de árboles urbanos
2. **Análisis**:
   - Calcular diferencia NDVI
   - Identificar pérdidas y ganancias
   - Estadísticas zonales por parque
   - Clasificar magnitud del cambio
3. **Machine Learning**:
   - Features: NDVI_t1, NDVI_t2, distancia a vías, densidad urbana
   - Target: Tipo de cambio (pérdida/estable/ganancia)
   - Modelo: Random Forest o SVM
   - Validación espacial

8.3 Proyecto 3: Optimización de Rutas de Recolección

Ejercicio

Optimicen rutas de recolección de residuos:

1. **Crear red**:
   - Líneas: Calles con sentido y velocidad
   - Puntos: Contenedores o puntos de recolección
   - Polígonos: Zonas de servicio
2. **Análisis**:
   - Clustering espacial de puntos de recolección
   - Asignación de zonas a camiones
   - Cálculo de ruta óptima por zona
   - Estimación de tiempo y distancia
3. **Optimización**:
   - Minimizar distancia total
   - Balancear carga entre camiones
   - Considerar horarios y tráfico
   - Proponer mejoras

9 Recursos y Referencias

9.1 Documentación Oficial

- **GeoPandas**: `https://geopandas.org`
- **Shapely**: `https://shapely.readthedocs.io`
- **Rasterio**: `https://rasterio.readthedocs.io`
- **Folium**: `https://python-visualization.github.io/folium/`
- **GDAL/OGR**: `https://gdal.org`

9.2 Datos para Practicar

- **Natural Earth**: Datos vectoriales globales
  `https://www.naturalearthdata.com`
- **OpenStreetMap**: Datos urbanos detallados
  `https://www.openstreetmap.org`
- **IDE Chile**: Datos oficiales de Chile
  `https://www.ide.cl`
- **Sentinel Hub**: Imágenes satelitales
  `https://www.sentinel-hub.com`
- **Earth Explorer**: Landsat y otros
  `https://earthexplorer.usgs.gov`

9.3 Tutoriales Recomendados

- **Automating GIS Processes** (Universidad de Helsinki):
  Excelente curso completo con notebooks
- **Earth Data Science** (Universidad de Colorado):
  Enfocado en análisis ambiental
- **Geocomputation with Python**:
  Libro online gratuito y actualizado
- **PyGIS**:
  Recursos en español para SIG con Python

9.4 Cheat Sheet: Operaciones Comunes

```
# === VECTORES ===
# Leer/Guardar
gdf = gpd.read_file('archivo.shp')
gdf.to_file('salida.geojson', driver='GeoJSON')

# CRS
gdf.crs                          # Ver CRS
gdf = gdf.to_crs('EPSG:32719')  # Reproyectar

# Operaciones
buffer = gdf.buffer(100)         # Buffer
union = gdf.unary_union          # Unir todo
```

```python
dissolved = gdf.dissolve(by='campo')  # Agrupar

# Spatial join
resultado = gpd.sjoin(puntos, poligonos, predicate='within')

# === RASTER ===
# Leer
import rasterio
with rasterio.open('imagen.tif') as src:
    data = src.read(1)  # Banda 1
    meta = src.meta     # Metadata

# Estadisticas zonales
import rasterstats
stats = rasterstats.zonal_stats(poligonos, 'raster.tif',
                                stats=['mean', 'max', 'min'])

# === VISUALIZACION ===
# Estatico
gdf.plot(column='campo', cmap='viridis', legend=True)

# Interactivo
m = folium.Map(location=[-33.45, -70.65], zoom_start=11)
folium.GeoJson(gdf.to_json()).add_to(m)
m.save('mapa.html')
```

Importante

Para el Laboratorio:

- Traigan laptop con ambiente configurado
- Descarguen datos de ejemplo de IDE Chile
- Formen grupos de 2-3 personas
- Preparen preguntas sobre los conceptos