# Guía de Proyecto

Sistema de Valoración Inmobiliaria Geoespacial

Documento de Orientación y Recursos

Para: Jaime Riquelme, Felipe Baeza, Valentina Barria,
Catalina López, Byron Caices

Profesor: Francisco Parra O.

Agosto 2025

**OBJETIVO DEL DOCUMENTO**

Este documento proporciona una guía detallada para desarrollar un Sistema de Valoración Inmobiliaria
con componente geoespacial, incluyendo problemática específica, fuentes de datos, metodología sugerida y código de ejemplo.

# Índice

# 1 Definición de la Problemática

## 1.1 Contexto del Problema

La valoración inmobiliaria en Chile presenta múltiples desafíos:

- **Asimetría de información**: Compradores y vendedores no tienen acceso a la misma información
- **Valoración subjetiva**: Los tasadores pueden tener sesgos o información incompleta
- **Factores espaciales ignorados**: Muchos modelos no consideran adecuadamente la ubicación
- **Cambios temporales**: El mercado inmobiliario es dinámico y los valores cambian constantemente
- **Externalidades no capturadas**: Contaminación, ruido, proyectos futuros no se reflejan en el precio

## 1.2 Problema Específico Propuesto

Problemática Central

**¿Cómo desarrollar un modelo de valoración inmobiliaria que integre eficientemente factores espaciales, temporales y de entorno para predecir con precisión el valor de propiedades en el Gran Santiago?**
Sub-problemas:

1. ¿Cuáles son los factores espaciales más relevantes para el precio?
2. ¿Cómo cuantificar el impacto de amenidades y des-amenidades urbanas?
3. ¿Cómo incorporar la autocorrelación espacial en el modelo?
4. ¿Cómo hacer el modelo interpretable para usuarios no técnicos?

## 1.3 Alcance Sugerido

Para hacer el proyecto manejable, sugiero enfocarse en:

1. **Área geográfica**: 3-4 comunas contiguas del Gran Santiago (ej: Providencia, Las Condes, Vitacura, Ñuñoa)
2. **Tipo de propiedad**: Departamentos (más homogéneos que casas)
3. **Período**: Últimos 2-3 años de datos
4. **Producto final**:
   - Modelo predictivo con $R^2$ ¿0.75
   - Dashboard interactivo con mapa
   - API REST para consultas
   - Reporte de factores más influyentes

# 2 Fuentes de Datos

## 2.1 Datos Primarios de Propiedades

| Fuente | Tipo de Datos | Acceso |
|---|---|---|
| Portal Inmobiliario | Precios, características | Web scraping (legal) |
| Yapo.cl | Arriendos y ventas | API no oficial |
| TocToc.com | Proyectos nuevos | Web scraping |
| Conservador Bienes Raíces | Transacciones reales | Pago/presencial |
| SII (Avalúo fiscal) | Valores fiscales | Portal SII |

## 2.2 Datos Geoespaciales

| Dato | Fuente | Formato |
|---|---|---|
| Límites comunales | IDE Chile / INE | Shapefile/GeoJSON |
| Red de Metro | DTPM / Metro de Santiago | KML/Shapefile |
| Paraderos de bus | DTPM | CSV con coordenadas |
| Áreas verdes | MINVU / Municipalidades | Shapefile |
| Colegios | MINEDUC | CSV con direcciones |
| Centros de salud | MINSAL / DEIS | Shapefile/Excel |
| Centros comerciales | OpenStreetMap | GeoJSON vía Overpass API |
| Delitos | Subsecretaría Prevención | CSV por cuadrante |
| Ruido ambiental | MMA / Municipalidades | Raster/puntos |

## 2.3 Datos Socioeconómicos

- **Censo 2017**: Datos demográficos por manzana censal
  - Fuente: `https://www.ine.cl/estadisticas/sociales/censos-de-poblacion-y-vivienda`
  - Incluye: Población, educación, ocupación, vivienda
- **Casen**: Encuesta de caracterización socioeconómica
  - Fuente: `http://observatorio.ministeriodesarrollosocial.gob.cl`
  - Incluye: Ingresos, pobreza, educación por comuna
- **IDE Observatorio de Ciudades UC**:
  - Fuente: `https://ideocuc-ocuc.hub.arcgis.com/`
  - Incluye: Múltiples indicadores urbanos georreferenciados

## 2.4 Código para Obtener Datos

### 2.4.1. Web Scraping de Portal Inmobiliario

```
import requests
from bs4 import BeautifulSoup
import pandas as pd
import time
from selenium import webdriver
```

```python
from selenium.webdriver.common.by import By

def scrape_portal_inmobiliario(comuna, tipo='departamento'):
    """
    Scraping responsable de Portal Inmobiliario
    IMPORTANTE: Respetar robots.txt y no sobrecargar el servidor
    """

    # URL base
    base_url = f"https://www.portalinmobiliario.com/venta/departamento/{comuna}"

    # Configurar Selenium (necesario para sitios con JS)
    options = webdriver.ChromeOptions()
    options.add_argument('--headless')
    driver = webdriver.Chrome(options=options)

    propiedades = []

    try:
        driver.get(base_url)
        time.sleep(3)  # Esperar carga de p gina

        # Obtener listado de propiedades
        listings = driver.find_elements(By.CLASS_NAME, "ui-search-result")

        for listing in listings[:10]:  # Limitar para ejemplo
            try:
                # Extraer informaci n
                precio = listing.find_element(By.CLASS_NAME, "price-tag-fraction
    ").text
                ubicacion = listing.find_element(By.CLASS_NAME, "ui-search-
    item__location").text
                atributos = listing.find_elements(By.CLASS_NAME, "ui-search-card
    -attributes__attribute")

                # Parsear atributos
                attrs = {}
                for attr in atributos:
                    texto = attr.text
                    if 'm  ' in texto:
                        attrs['superficie'] = texto
                    elif 'dormitorio' in texto:
                        attrs['dormitorios'] = texto
                    elif 'ba o' in texto:
                        attrs['banos'] = texto

                propiedades.append({
                    'precio': precio,
                    'ubicacion': ubicacion,
                    **attrs
                })

            except Exception as e:
                continue

    finally:
        driver.quit()

    return pd.DataFrame(propiedades)

# Uso
# df_props = scrape_portal_inmobiliario('providencia')
# print(df_props.head())
```

### 2.4.2. Datos de OpenStreetMap

```python
import osmnx as ox
import geopandas as gpd

def obtener_amenidades_osm(comuna, tipo_amenidad):
    """
    Obtener amenidades desde OpenStreetMap
    """
    # Definir  rea  de b squeda
    place = f"{comuna}, Santiago, Chile"

    # Obtener amenidades
    amenidades = ox.geometries_from_place(
        place,
        tags={'amenity': tipo_amenidad}
    )

    # Convertir a GeoDataFrame
    gdf = gpd.GeoDataFrame(amenidades)

    # Filtrar solo puntos (algunos pueden ser pol gonos)
    gdf_puntos = gdf[gdf.geometry.type == 'Point']

    return gdf_puntos[['name', 'amenity', 'geometry']]

# Obtener diferentes amenidades
restaurantes = obtener_amenidades_osm('Providencia', 'restaurant')
colegios = obtener_amenidades_osm('Providencia', 'school')
parques = obtener_amenidades_osm('Providencia', 'park')
hospitales = obtener_amenidades_osm('Providencia', 'hospital')

print(f"Restaurantes: {len(restaurantes)}")
print(f"Colegios: {len(colegios)}")
print(f"Parques: {len(parques)}")
print(f"Hospitales: {len(hospitales)}")
```

# 3 Metodología Propuesta

## 3.1 Pipeline de Procesamiento

1. **Recolección de Datos**
   - Web scraping de portales inmobiliarios
   - Descarga de datos geoespaciales oficiales
   - Geocodificación de direcciones
2. **Limpieza y Preparación**
   - Eliminar duplicados y outliers
   - Imputación de valores faltantes
   - Estandarización de formatos
   - Validación de geometrías
3. **Feature Engineering Espacial**
   - Distancias a puntos de interés
   - Densidad de amenidades en buffers
   - Índices de accesibilidad
   - Variables de entorno (ruido, contaminación)
   - Lag espacial de precios
4. **Modelamiento**
   - Modelos base: Random Forest, XGBoost
   - Modelos espaciales: GWR, Spatial Lag
   - Ensemble de modelos
   - Validación espacial (no aleatoria)
5. **Visualización y Deployment**
   - Dashboard interactivo con Streamlit
   - Mapa de calor de precios
   - API REST con FastAPI
   - Documentación completa

## 3.2 Feature Engineering Detallado

```
import geopandas as gpd
import numpy as np
from shapely.geometry import Point
from sklearn.neighbors import NearestNeighbors

class FeatureEngineerEspacial:
    """
    Clase para generar features espaciales para valoraci n inmobiliaria
    """

    def __init__(self, propiedades_gdf, amenidades_dict):
        """
```

```
        propiedades_gdf: GeoDataFrame con propiedades
        amenidades_dict: Dict con GeoDataFrames de amenidades
        """
        self.propiedades = propiedades_gdf
        self.amenidades = amenidades_dict

    def distancia_mas_cercana(self, tipo_amenidad):
        """Distancia a la amenidad m s cercana"""
        amenidad_gdf = self.amenidades[tipo_amenidad]

        distancias = []
        for idx, prop in self.propiedades.iterrows():
            dist_min = amenidad_gdf.geometry.distance(prop.geometry).min()
            distancias.append(dist_min * 111000)  # Convertir a metros

        return np.array(distancias)

    def densidad_en_radio(self, tipo_amenidad, radio_metros):
        """Cantidad de amenidades en un radio dado"""
        amenidad_gdf = self.amenidades[tipo_amenidad]
        radio_grados = radio_metros / 111000

        densidades = []
        for idx, prop in self.propiedades.iterrows():
            buffer = prop.geometry.buffer(radio_grados)
            dentro = amenidad_gdf[amenidad_gdf.geometry.within(buffer)]
            densidades.append(len(dentro))

        return np.array(densidades)

    def indice_accesibilidad(self, pesos=None):
        """ ndice  compuesto de accesibilidad"""
        if pesos is None:
            pesos = {
                'metro': 0.3,
                'bus': 0.2,
                'colegio': 0.2,
                'hospital': 0.15,
                'parque': 0.15
            }

        indice = np.zeros(len(self.propiedades))

        for amenidad, peso in pesos.items():
            if amenidad in self.amenidades:
                # Normalizar distancia (inversa)
                dist = self.distancia_mas_cercana(amenidad)
                dist_norm = 1 / (1 + dist/1000)  # Decaimiento con distancia
                indice += peso * dist_norm

        return indice

    def lag_espacial_precio(self, k_vecinos=5):
        """Precio promedio de los k vecinos m s cercanos"""
        coords = np.array([[p.x, p.y] for p in self.propiedades.geometry])

        # KNN para encontrar vecinos
        nbrs = NearestNeighbors(n_neighbors=k_vecinos+1)
        nbrs.fit(coords)
        distances, indices = nbrs.kneighbors(coords)

        # Calcular lag (excluyendo la propiedad misma)
        lag_prices = []
```

```
        for idx_list in indices:
            vecinos = idx_list[1:]  # Excluir el primero (s   mismo)
            precio_promedio = self.propiedades.iloc[vecinos]['precio'].mean()
            lag_prices.append(precio_promedio)

        return np.array(lag_prices)

    def crear_features(self):
        """Generar todas las features espaciales"""
        features = pd.DataFrame()

        # Distancias
        features['dist_metro'] = self.distancia_mas_cercana('metro')
        features['dist_colegio'] = self.distancia_mas_cercana('colegio')
        features['dist_hospital'] = self.distancia_mas_cercana('hospital')
        features['dist_parque'] = self.distancia_mas_cercana('parque')

        # Densidades
        features['restaurantes_500m'] = self.densidad_en_radio('restaurant',
    500)
        features['colegios_1km'] = self.densidad_en_radio('colegio', 1000)
        features['parques_1km'] = self.densidad_en_radio('parque', 1000)

        #   ndices   compuestos
        features['indice_accesibilidad'] = self.indice_accesibilidad()
        features['precio_lag'] = self.lag_espacial_precio()

        # Coordenadas (para capturar tendencias espaciales)
        features['lat'] = [p.y for p in self.propiedades.geometry]
        features['lon'] = [p.x for p in self.propiedades.geometry]

        return features

# Uso
# fe = FeatureEngineerEspacial(propiedades_gdf, amenidades_dict)
# features_espaciales = fe.crear_features()
```

### 3.3 Modelo de Machine Learning

```
from sklearn.model_selection import KFold
from sklearn.ensemble import RandomForestRegressor
from xgboost import XGBRegressor
from sklearn.metrics import mean_absolute_error, r2_score
import numpy as np

class ModeloValoracion:
    """
    Modelo de valoraci n inmobiliaria con validaci n espacial
    """

    def __init__(self):
        self.modelos = {
            'rf': RandomForestRegressor(
                n_estimators=200,
                max_depth=15,
                min_samples_split=5,
                random_state=42
            ),
            'xgb': XGBRegressor(
                n_estimators=200,
                max_depth=8,
                learning_rate=0.05,
```

```
                random_state=42
            )
        }
        self.mejor_modelo = None
        self.features_importance = None

    def validacion_espacial(self, X, y, coords, n_splits=5):
        """
        Validaci n por bloques espaciales, no aleatoria
        """
        from sklearn.cluster import KMeans

        # Crear clusters espaciales
        kmeans = KMeans(n_clusters=n_splits, random_state=42)
        clusters = kmeans.fit_predict(coords)

        scores = {nombre: [] for nombre in self.modelos.keys()}

        for cluster_test in range(n_splits):
            # Split espacial
            mask_test = clusters == cluster_test
            mask_train = ~mask_test

            X_train, X_test = X[mask_train], X[mask_test]
            y_train, y_test = y[mask_train], y[mask_test]

            # Entrenar y evaluar cada modelo
            for nombre, modelo in self.modelos.items():
                modelo.fit(X_train, y_train)
                y_pred = modelo.predict(X_test)
                score = r2_score(y_test, y_pred)
                scores[nombre].append(score)

        # Promediar scores
        mean_scores = {nombre: np.mean(s) for nombre, s in scores.items()}

        # Seleccionar mejor modelo
        mejor_nombre = max(mean_scores, key=mean_scores.get)
        self.mejor_modelo = self.modelos[mejor_nombre]

        print("=== Resultados Validaci n Espacial ===")
        for nombre, score in mean_scores.items():
            print(f"{nombre}: R   = {score:.3f}")
        print(f"\nMejor modelo: {mejor_nombre}")

        return mean_scores

    def entrenar_modelo_final(self, X, y):
        """Entrenar modelo final con todos los datos"""
        self.mejor_modelo.fit(X, y)

        # Feature importance
        if hasattr(self.mejor_modelo, 'feature_importances_'):
            self.features_importance = self.mejor_modelo.feature_importances_

        return self

    def predecir(self, X):
        """Hacer predicciones"""
        return self.mejor_modelo.predict(X)

    def explicar_prediccion(self, X_single, feature_names):
        """
```

```
        Explicar una predicci n individual usando SHAP
        """
        import shap

        # Crear explicador SHAP
        explainer = shap.TreeExplainer(self.mejor_modelo)
        shap_values = explainer.shap_values(X_single)

        # Crear dataframe con contribuciones
        contribuciones = pd.DataFrame({
            'feature': feature_names,
            'valor': X_single.flatten(),
            'impacto': shap_values.flatten()
        })

        contribuciones = contribuciones.sort_values('impacto',
                                                    key=abs,
                                                    ascending=False)

        return contribuciones

# Uso
# modelo = ModeloValoracion()
# scores = modelo.validacion_espacial(X, y, coords)
# modelo.entrenar_modelo_final(X, y)
# predicciones = modelo.predecir(X_test)
```

# 4 Implementación del Sistema

## 4.1 Arquitectura Propuesta

SISTEMA DE VALORACIÓN

| Data Layer | Process Layer | Model Layer | Present Layer |
|---|---|---|---|
| • Web Scraping | • Limpieza | • RF/XGBoost | • Dashboard |
| • APIs | • Geocoding | • GWR | • API REST |
| • Archivos | • Features | • Ensemble | • Reportes |
| • PostgreSQL | • Validación | • SHAP | • Mapas |

## 4.2 Dashboard Interactivo con Streamlit

```python
import streamlit as st
import pandas as pd
import geopandas as gpd
import folium
from streamlit_folium import folium_static
import plotly.express as px

# Configuraci n de la p gina
st.set_page_config(
    page_title="Valoraci n Inmobiliaria Santiago",
    page_icon="    ",
    layout="wide"
)

st.title("     Sistema de Valoraci n Inmobiliaria")
st.markdown("---")

# Sidebar para inputs
with st.sidebar:
    st.header("Par metros de B squeda")

    comuna = st.selectbox(
        "Comuna",
        ["Providencia", "Las Condes", "Vitacura", "  uoa  "]
    )

    tipo_propiedad = st.selectbox(
        "Tipo de Propiedad",
        ["Departamento", "Casa", "Oficina"]
    )

    superficie = st.slider(
        "Superficie (m  )",
        min_value=20,
        max_value=500,
        value=(50, 150)
    )

    dormitorios = st.slider(
        "Dormitorios",
        min_value=1,
        max_value=6,
        value=2
```

```
    )

    banos = st.slider(
        "Ba os",
        min_value=1,
        max_value=4,
        value=1
    )

    st.markdown("---")
    st.header("Factores de Entorno")

    cerca_metro = st.checkbox("Cerca de Metro (<500m)")
    cerca_parque = st.checkbox("Cerca de Parque (<300m)")
    cerca_colegio = st.checkbox("Cerca de Colegio (<1km)")

# Layout principal
col1, col2 = st.columns([2, 1])

with col1:
    st.subheader("Mapa de Valoraci n")

    # Crear mapa base
    m = folium.Map(
        location=[-33.45, -70.65],
        zoom_start=12,
        tiles='OpenStreetMap'
    )

    # Agregar capa de calor de precios
    # (aqu  ir an los datos reales)
    from folium.plugins import HeatMap

    # Datos de ejemplo
    heat_data = [
        [-33.42, -70.61, 0.9],  # Las Condes alto
        [-33.43, -70.63, 0.7],  # Providencia
        [-33.45, -70.65, 0.5],  # Santiago Centro
        [-33.47, -70.67, 0.3],  # Estaci n Central
    ]

    HeatMap(heat_data).add_to(m)

    # Agregar marcadores de propiedades
    # (datos de ejemplo)
    propiedades_ejemplo = [
        {"lat": -33.425, "lon": -70.615, "precio": 6500, "dir": "Av. Apoquindo
    3000"},
        {"lat": -33.435, "lon": -70.625, "precio": 4500, "dir": "Providencia
    1234"},
        {"lat": -33.445, "lon": -70.635, "precio": 3500, "dir": "Manuel Montt
    567"},
    ]

    for prop in propiedades_ejemplo:
        folium.Marker(
            [prop["lat"], prop["lon"]],
            popup=f"${prop['precio']} UF<br>{prop['dir']}",
            icon=folium.Icon(color='green', icon='home')
        ).add_to(m)

    # Mostrar mapa
    folium_static(m)
```

```
104
105 with col2:
106     st.subheader("Valoraci n Estimada")
107
108     # C lculo de precio (simulado)
109     precio_base = 3000  # UF
110
111     # Ajustes por comuna
112     ajuste_comuna = {
113         "Las Condes": 1.3,
114         "Vitacura": 1.4,
115         "Providencia": 1.2,
116         "  uoa  ": 1.0
117     }
118
119     precio_estimado = precio_base * ajuste_comuna[comuna]
120
121     # Ajustes por caracter sticas
122     if cerca_metro:
123         precio_estimado *= 1.15
124     if cerca_parque:
125         precio_estimado *= 1.10
126     if cerca_colegio:
127         precio_estimado *= 1.05
128
129     # Mostrar precio
130     st.metric(
131         label="Precio Estimado",
132         value=f"{precio_estimado:.0f} UF",
133         delta=f"{(precio_estimado/precio_base - 1)*100:.1f}% vs promedio"
134     )
135
136     st.markdown("---")
137
138     # Factores que influyen
139     st.subheader("Factores Principales")
140
141     factores = pd.DataFrame({
142         'Factor': ['Ubicaci n', 'Metro', 'Superficie', 'Parques', 'Colegios'],
143         'Impacto': [35, 25, 20, 10, 10]
144     })
145
146     fig = px.bar(
147         factores,
148         x='Impacto',
149         y='Factor',
150         orientation='h',
151         color='Impacto',
152         color_continuous_scale='RdYlGn'
153     )
154
155     st.plotly_chart(fig, use_container_width=True)
156
157 # Secci n de comparaci n
158 st.markdown("---")
159 st.subheader("Comparaci n con Propiedades Similares")
160
161 # Tabla de propiedades similares (datos de ejemplo)
162 similares = pd.DataFrame({
163     'Direcci n': ['Av. Providencia 123', 'Los Leones 456', 'Tobalaba 789'],
164     'Superficie': [85, 92, 78],
165     'Dormitorios': [2, 2, 2],
166     'Precio UF': [4200, 4500, 3900],
```

```
    'Precio/m ': [49.4, 48.9, 50.0]
})

st.dataframe(similares, use_container_width=True)

# Gr fico de tendencia
st.markdown("---")
st.subheader("Tendencia de Precios ( ltimos  12 meses)")

# Datos de ejemplo
meses = pd.date_range('2024-01-01', periods=12, freq='M')
precios_promedio = [3000, 3050, 3100, 3150, 3200, 3180,
                    3250, 3300, 3350, 3400, 3420, 3450]

tendencia = pd.DataFrame({
    'Mes': meses,
    'Precio_UF': precios_promedio
})

fig_tendencia = px.line(
    tendencia,
    x='Mes',
    y='Precio_UF',
    title=f'Evoluci n Precio Promedio - {comuna}',
    markers=True
)

st.plotly_chart(fig_tendencia, use_container_width=True)

# Footer
st.markdown("---")
st.caption("Sistema desarrollado por Grupo de Valoraci n Inmobiliaria -
    Geoinform tica 2025")
```

### 4.3 API REST con FastAPI

```
from fastapi import FastAPI, HTTPException
from pydantic import BaseModel
from typing import Optional
import pandas as pd
import joblib

app = FastAPI(title="API Valoraci n Inmobiliaria")

# Cargar modelo entrenado
modelo = joblib.load('modelo_valoracion.pkl')
scaler = joblib.load('scaler.pkl')

class PropiedadInput(BaseModel):
    """Schema de entrada para valoraci n"""
    comuna: str
    superficie: float
    dormitorios: int
    banos: int
    estacionamientos: Optional[int] = 0
    piso: Optional[int] = 1
    orientacion: Optional[str] = "Norte"
    lat: float
    lon: float

class ValoracionOutput(BaseModel):
    """Schema de salida con valoraci n"""
```

```
    precio_estimado_uf: float
    precio_estimado_clp: float
    rango_min_uf: float
    rango_max_uf: float
    confianza: float
    factores_principales: dict

@app.get("/")
def read_root():
    return {
        "mensaje": "API de Valoraci n Inmobiliaria",
        "version": "1.0",
        "endpoints": ["/valorar", "/comparar", "/tendencia"]
    }

@app.post("/valorar", response_model=ValoracionOutput)
async def valorar_propiedad(propiedad: PropiedadInput):
    """
    Endpoint para valorar una propiedad
    """
    try:
        # Preparar features
        features = prepare_features(propiedad)
        features_scaled = scaler.transform(features)

        # Predecir
        precio_uf = modelo.predict(features_scaled)[0]

        # Calcular intervalo de confianza
        if hasattr(modelo, 'predict_proba'):
            std = modelo.predict_std(features_scaled)[0]
            rango_min = precio_uf - 1.96 * std
            rango_max = precio_uf + 1.96 * std
            confianza = min(1.0, 1 / (1 + std/precio_uf))
        else:
            rango_min = precio_uf * 0.9
            rango_max = precio_uf * 1.1
            confianza = 0.85

        # Factores principales (simulado, idealmente con SHAP)
        factores = {
            "ubicacion": 0.35,
            "superficie": 0.25,
            "dormitorios": 0.15,
            "amenidades": 0.15,
            "otros": 0.10
        }

        return ValoracionOutput(
            precio_estimado_uf=round(precio_uf, 0),
            precio_estimado_clp=round(precio_uf * 40000, 0),
            rango_min_uf=round(rango_min, 0),
            rango_max_uf=round(rango_max, 0),
            confianza=round(confianza, 2),
            factores_principales=factores
        )

    except Exception as e:
        raise HTTPException(status_code=400, detail=str(e))

@app.get("/comparar/{comuna}")
async def comparar_comuna(comuna: str, superficie_min: int = 50):
    """
```

```
    Comparar precios en una comuna
    """
    # Aqu  ir a la l gica real de comparaci n
    return {
        "comuna": comuna,
        "precio_promedio_uf": 3500,
        "precio_min_uf": 2000,
        "precio_max_uf": 8000,
        "total_propiedades": 1234,
        "variacion_mensual": 2.3
    }

@app.get("/tendencia/{comuna}/{periodo}")
async def obtener_tendencia(comuna: str, periodo: str = "12m"):
    """
    Obtener tendencia de precios
    """
    # Datos de ejemplo
    return {
        "comuna": comuna,
        "periodo": periodo,
        "tendencia": [
            {"mes": "2024-01", "precio_promedio": 3000},
            {"mes": "2024-02", "precio_promedio": 3050},
            {"mes": "2024-03", "precio_promedio": 3100},
        ],
        "proyeccion_3m": 3250,
        "crecimiento_anual": 8.5
    }

def prepare_features(propiedad: PropiedadInput):
    """
    Preparar features para el modelo
    """
    # Aqu  ir a la l gica real de preparaci n
    features = pd.DataFrame([{
        'superficie': propiedad.superficie,
        'dormitorios': propiedad.dormitorios,
        'banos': propiedad.banos,
        'lat': propiedad.lat,
        'lon': propiedad.lon,
        # ... m s features
    }])
    return features

# Para ejecutar:
# uvicorn main:app --reload
```

# 5 Análisis de Resultados Esperados

## 5.1 Métricas de Evaluación

| Métrica | Objetivo | Interpretación |
|---|---|---|
| $R^2$ Score | $> 0{,}75$ | Varianza explicada por el modelo |
| MAE (UF) | $< 200$ | Error absoluto promedio |
| MAPE (%) | $< 10\,\%$ | Error porcentual promedio |
| RMSE (UF) | $< 300$ | Penaliza errores grandes |

## 5.2 Visualizaciones Clave

1. **Mapa de calor de precios**: Identificar zonas premium y económicas
2. **Scatter plot predicción vs real**: Evaluar precisión del modelo
3. **Gráfico SHAP**: Explicar predicciones individuales
4. **Serie temporal**: Evolución de precios por zona
5. **Boxplot por comuna**: Distribución de precios

# 6 Cronograma Sugerido

| Semana | Fase | Actividades |
|---|---|---|
| 1-2 | Recolección | • Web scraping inicial<br>• Descarga datos geoespaciales<br>• Configuración ambiente |
| 3-4 | Preparación | • Limpieza de datos<br>• Geocodificación<br>• Integración fuentes |
| 5-6 | Features | • Cálculo distancias<br>• Índices de accesibilidad<br>• Variables de entorno |
| 7-8 | Modelado | • Entrenamiento modelos<br>• Validación espacial<br>• Optimización hiperparámetros |
| 9-10 | Sistema | • Desarrollo dashboard<br>• Implementación API<br>• Testing |
| 11-12 | Finalización | • Documentación<br>• Presentación<br>• Deployment |

# 7 Entregables Finales

## 7.1 Código y Documentación

- **Repositorio GitHub** con:
  - README completo
  - Requirements.txt
  - Notebooks documentados
  - Scripts modulares
  - Tests unitarios
- **Documentación técnica**:
  - Descripción de features
  - Metodología de modelado
  - API documentation
  - Manual de usuario

## 7.2 Aplicación Web

- Dashboard interactivo con:
  - Mapa de valoraciones
  - Calculadora de precio

    - Comparador de propiedades
    - Tendencias del mercado
- API REST con endpoints para:
    - Valoración individual
    - Comparación por zona
    - Tendencias históricas
    - Recomendaciones

### 7.3 Presentación

- **Slides** (15-20) cubriendo:
    - Problema y motivación
    - Datos y metodología
    - Resultados y métricas
    - Demo en vivo
    - Conclusiones y trabajo futuro
- **Poster científico** formato A1
- **Video demo** (3-5 minutos)

# 8 Referencias y Recursos Adicionales

## 8.1 Papers Relevantes

1. Bourassa, S. C., Cantoni, E., & Hoesli, M. (2007). *Spatial dependence, housing submarkets, and house price prediction*. The Journal of Real Estate Finance and Economics.

2. Yoo, S., Im, J., & Wagner, J. E. (2012). *Variable selection for hedonic model using machine learning approaches*. Landscape and Urban Planning.

3. Čeh, M., Kilibarda, M., Lisec, A., & Bajat, B. (2018). *Estimating the performance of random forest versus multiple regression for predicting prices of the apartments*. ISPRS International Journal of Geo-Information.

## 8.2 Bibliotecas Python Especializadas

- **PySAL**: Análisis espacial avanzado
- **GeoPy**: Geocodificación
- **OSMnx**: Datos de OpenStreetMap
- **Folium**: Mapas interactivos
- **SHAP**: Explicabilidad de modelos
- **Prophet**: Series temporales

## 8.3 Datasets Públicos de Referencia

- **Boston Housing**: Dataset clásico para practicar
- **King County**: Casas en Seattle con componente espacial
- **Ames Housing**: Dataset detallado de Iowa

## 8.4 Herramientas Complementarias

- **QGIS**: Para análisis espacial visual
- **PostgreSQL + PostGIS**: Base de datos espacial
- **Apache Superset**: Dashboards empresariales
- **MLflow**: Tracking de experimentos
- **DVC**: Versionado de datos

**CONTACTO Y SOPORTE**

Para dudas específicas sobre implementación, pueden contactar:
francisco.parra.o@usach.cl

Horario de consulta: Jueves 14:00-16:00
Office Hours virtuales: Previa coordinación