Integrantes

- Montoya Solórzano, Leonardo Alfredo
- Salazar Medina, Breysi Fernanda
- Villarreal Falcón, Mishelle Stephany

Aquisición de datos - Parte 1 (Web scraping)

Este script realiza la extracción automatizada de registros sísmicos del portal oficial del Instituto Geofísico del Perú (IGP), utilizando Selenium WebDriver para navegar dinámicamente por la interfaz web.

🔹 **Objetivo**

Recolectar todos los eventos sísmicos registrados por el IGP entre los años 2020 y 2025, incluyendo información detallada de cada sismo (fecha, magnitud, ubicación, latitud, longitud y profundidad), para su posterior análisis y visualización.

🔹 **Flujo general del proceso**

1. Inicialización del navegador
    - Se configura webdriver.Chrome() y se define la URL base del portal de sismos.
2. Recorrido por años (2025–2020)
    - Se selecciona cada año desde el menú desplegable.
    - Luego, se cambia la cantidad de registros mostrados por página a 100.
3. Extracción de datos por página
    - Se recorren todas las páginas de resultados mediante el botón “Siguiente”, hasta completar el conjunto anual.
    - De cada fila de la tabla se extraen: reporte, referencia, fecha_hora, magnitud y el enlace (href) al reporte detallado.
4. Obtención de detalles adicionales
    - Para cada reporte individual, el script abre el enlace en una nueva pestaña y extrae: latitud, longitud y profundidad.
    - Estos valores se buscan en los elementos de la página de detalle.
5. Construcción del dataset final
    - Todos los registros se consolidan en un DataFrame de Pandas.
    - Se exporta el resultado a un archivo CSV llamado sismos_igp.csv.
6. Cierre del proceso
    - Se cierra el navegador y se imprime el número total de registros extraídos.

```
import pandas as pd
import numpy as np
import time
from selenium import webdriver
from selenium.webdriver.common.by import By
from selenium.webdriver.support.ui import WebDriverWait
from selenium.webdriver.support import expected_conditions as EC
from selenium.webdriver.support.ui import Select
```

```
# Definir la URL
url = 'https://ultimosismo.igp.gob.pe/ultimo-sismo/sismos-reportados'

# Configurar Selenium WebDriver
driver = webdriver.Chrome()
wait = WebDriverWait(driver, 15)

driver.get(url)

# Lista de años (de 2025 a 2020)
years = ["2025", "2024", "2023", "2022", "2021", "2020"]

entries = []

for year in years:
    print(f"\n🔹 Procesando año {year}...")

    # Seleccionar el año en el combo <select id="year">
    wait.until(EC.presence_of_element_located((By.ID, "year")))
    # Crear el objeto Select y elegir el año
    select_year = Select(driver.find_element(By.ID, "year"))
    select_year.select_by_value(year)

    # Esperar a que cambie el contenido de la tabla
    wait.until(EC.presence_of_element_located((By.ID, "show")))
    time.sleep(2)

    # Crear el objeto Select y elegir el valor "12"
    wait.until(EC.presence_of_element_located((By.ID, "show")))
    select_show = Select(driver.find_element(By.ID, "show"))
    select_show.select_by_value("12")

    # Crear el objeto Select y elegir el valor "100"
    wait.until(EC.presence_of_element_located((By.ID, "show")))
    select_show = Select(driver.find_element(By.ID, "show"))
    select_show.select_by_value("100")

    # Esperar a que la tabla de resultados esté presente
    wait.until(EC.presence_of_element_located((By.CSS_SELECTOR, "table tbody tr")))
    time.sleep(2)

    while True:
        # Esperar a que la tabla de resultados esté presente
        wait.until(EC.presence_of_element_located((By.CSS_SELECTOR, "table tbody tr")))

        # Extraer de la tabla los datos visibles
        rows = driver.find_elements(By.CSS_SELECTOR, "table tbody tr")

        for r in rows:
            try:
                cols = r.find_elements(By.TAG_NAME, "td")
                reporte = cols[0].text.strip()
                referencia = cols[1].text.strip()
                fecha_hora = cols[2].text.strip()
                magnitud = cols[3].text.strip()

                # Busca todos los enlaces en la columna de descargas
                anchors = cols[4].find_elements(By.TAG_NAME, "a")

                href = None

                # Recorremos los enlaces
```

```
                for a in anchors:
                    txt = (a.text or "").lower()
                    h = a.get_attribute("href")
                    # preferimos el que tiene 'reporte' en el texto
                    if 'reporte' in txt:
                        href = h
                        break

                # Agregamos los datos del evento a la lista de entradas
                entries.append({
                    "anio": year,
                    "reporte": reporte,
                    "referencia": referencia,
                    "fecha_hora": fecha_hora,
                    "magnitud": magnitud,
                    "href": href
                })

            except Exception as e:
                print("Salteando fila por error:", e)
                continue

        # Intentar encontrar y hacer clic en "Siguiente"
        try:
            next_button = driver.find_element(By.XPATH, "//button[contains(., 'Siguiente')]")

            # Si el botón está deshabilitado, terminamos
            if not next_button.is_enabled():
                print(f"Fin de páginas del año {year}.")
                break

            # Hacer clic en el botón "Siguiente"
            driver.execute_script("arguments[0].click();", next_button)
            print("Pasando a la siguiente página...")
            # Esperar que se actualice la tabla (detectando cambio de primera celda)
            wait.until(EC.presence_of_element_located((By.CSS_SELECTOR, "table tbody tr td")))
            time.sleep(3)  # esperar que cargue la nueva tabla

        except Exception as e:
            print("No se encontró el botón 'Siguiente' o ya no hay más páginas:", e)
            break

print(f"Total registros encontrados: {len(entries)}")

# Abrir cada link en una pestaña nueva y extraer latitud-longitud y profundidad
data = []

# Guardar el identificador de la ventana principal
main_handle = driver.current_window_handle

# Iterar sobre las entradas y abrir cada link
for ent in entries:
    href = ent["href"]
    lat_lon = ""
    profundidad = ""

    if not href:
        # no hay link, agregamos vacío y seguimos
        data.append([ent["reporte"], ent["referencia"], ent["fecha_hora"], ent["magnitud"], lat_lon, profundidad, None])
        continue

    # abrir nueva pestaña con el href
```

```
    driver.execute_script("window.open(arguments[0]);", href)
    # cambiar el control a la pestaña nueva (-1 es la ultima)
    driver.switch_to.window(driver.window_handles[-1])

    try:
        # esperar que cargue la página de detalle (b.text-dark existe)
        wait.until(EC.presence_of_all_elements_located((By.CSS_SELECTOR, "b.text-dark")))

        # Buscar los <b class="text-dark">
        b_elems = driver.find_elements(By.CSS_SELECTOR, "b.text-dark")

        # Recorremos los <b> para encontrar latitud y profundidad
        for b in b_elems:
            txt = (b.text or "").lower()
            try:
                if "latitud" in txt:
                    # Si contiene "latitud", tomar el siguiente <span>
                    lat_lon = b.find_element(By.XPATH, "./following-sibling::span").text.strip()
                    # Si contiene "profundidad", tomar el siguiente <span>
                if "profundidad" in txt:
                    profundidad = b.find_element(By.XPATH, "./following-sibling::span").text.strip()
            except Exception:
                # si no encuentra el sibling o hay otro formato, ignorar
                pass

        # Agregar los datos extraídos a la data general
        data.append([ent["reporte"], ent["referencia"], ent["fecha_hora"], ent["magnitud"], lat_lon, profundidad, href])

    # En caso de error, agregar la entrada con lat_lon y profundidad vacíos
    except Exception as e:
        print("Error al abrir detalle:", href, e)
        data.append([ent["reporte"], ent["referencia"], ent["fecha_hora"], ent["magnitud"], lat_lon, profundidad, href])

    # Cerrar pestaña de detalle
    driver.close()

    # volver a la ventana principal
    driver.switch_to.window(main_handle)

    # Esperar que la tabla esté presente antes de pasar al siguiente sismo
    wait.until(EC.presence_of_element_located((By.CSS_SELECTOR, "table tbody tr")))

# 3) Guardar a DataFrame / CSV
df = pd.DataFrame(data, columns=["reporte","referencia","fecha_hora","magnitud","lat_lon","profundidad","href"])
df.to_csv("../data/processed/sismos_igp.csv", index=False, encoding="utf-8-sig")

print("Extraído:", len(df), "filas. CSV -> sismos_igp.csv")

driver.quit()
```

```
Pasando a la siguiente página...
Pasando a la siguiente página...
Fin de páginas del año 2025.

🔹 Procesando año 2024...
Pasando a la siguiente página...
```

```
🔹 Procesando año 2023...
Pasando a la siguiente página...
Pasando a la siguiente página...
Pasando a la siguiente página...
Pasando a la siguiente página...
Pasando a la siguiente página...
Pasando a la siguiente página...
Pasando a la siguiente página...
Fin de páginas del año 2023.

🔹 Procesando año 2022...
Pasando a la siguiente página...
Pasando a la siguiente página...
Pasando a la siguiente página...
Pasando a la siguiente página...
Pasando a la siguiente página...
Pasando a la siguiente página...
Pasando a la siguiente página...
Pasando a la siguiente página...
Fin de páginas del año 2022.

🔹 Procesando año 2021...
Pasando a la siguiente página...
Pasando a la siguiente página...
Pasando a la siguiente página...
Pasando a la siguiente página...
Pasando a la siguiente página...
Pasando a la siguiente página...
Pasando a la siguiente página...
Pasando a la siguiente página...
Fin de páginas del año 2021.

🔹 Procesando año 2020...
Pasando a la siguiente página...
Pasando a la siguiente página...
Pasando a la siguiente página...
Pasando a la siguiente página...
Pasando a la siguiente página...
Pasando a la siguiente página...
Pasando a la siguiente página...
Pasando a la siguiente página...
Fin de páginas del año 2020.
Total registros encontrados: 4732
Extraído: 4732 filas  CSV  > sismos_igp.csv
```

```
# Verificar el CSV cargando el DataFrame
df = pd.read_csv("../data/processed/sismos_igp.csv")
df.head()
```

	reporte	referencia	fecha_hora	magnitud	lat_lon	profundidad	href
0	IGP/CENSIS/RS\n2025-0718	42 km al NO de Satipo, Satipo - Junín	29/10/2025 05:45:16	3.6	-10.91, -74.81	16 Km	https://ultimosismo.igp.gob.pe/evento/2025-0718
1	IGP/CENSIS/RS\n2025-0717	46 km al NO de Satipo, Satipo - Junín	29/10/2025 04:45:21	3.8	-10.89, -74.84	18 Km	https://ultimosismo.igp.gob.pe/evento/2025-0717
2	IGP/CENSIS/RS\n2025-0716	71 km al SO de Tacna, Tacna - Tacna	29/10/2025 03:22:58	3.8	-18.56, -70.6	37 Km	https://ultimosismo.igp.gob.pe/evento/2025-0716
3	IGP/CENSIS/RS\n2025-0715	16 km al NO de Zorritos, Contralmirante Villar...	28/10/2025 15:44:06	3.8	-3.61, -80.79	20 Km	https://ultimosismo.igp.gob.pe/evento/2025-0715
4	IGP/CENSIS/RS\n2025-0714	23 km al O de Marcona, Nasca - Ica	26/10/2025 23:52:15	4.1	-15.42, -75.37	34 Km	https://ultimosismo.igp.gob.pe/evento/2025-0714

Aquisición de datos - Parte 2 (INEI)

Esta parte constituye la segunda fase del proyecto de extracción y análisis sísmico, enfocada en enriquecer los datos obtenidos del IGP mediante la integración de información geográfica (distritos) y la proyección sociodemográfica (2018-2025) obtenida del INEI. Su propósito es localizar espacialmente cada sismo dentro del territorio peruano, asignarle su distrito correspondiente, y vincularlo con variables territoriales y poblacionales.

🔹 **Objetivo general**

Combinar los datos sísmicos con capas geográficas y censales para obtener un dataset georreferenciado y contextualizado, que permita realizar análisis espaciales, estimaciones de riesgo y visualizaciones por distrito.

🔹 **Flujo general del proceso**

1. Carga y preparación inicial
    - Se carga el archivo sismos_igp.csv generado en la fase anterior.
    - Se separa la columna lat_lon en dos nuevas columnas: latitud y longitud.
    - Se genera una columna geométrica geometry con objetos Point(longitud, latitud).
2. Creación del GeoDataFrame
    - Se convierte el DataFrame en un GeoDataFrame (gdf_sismos) con el sistema de referencia geográfico EPSG:4326 (WGS84), estándar para coordenadas GPS.
3. Asignación de distritos mediante spatial join
    - Se carga la capa geográfica DISTRITO.gpkg (GeoPackage oficial de distritos del Perú).
    - Se reproyecta tanto en grados (EPSG:4326) como en metros (EPSG:32718) para cálculos de área.
    - Cada sismo se asocia con el distrito dentro del cual ocurrió.
    - Se renombran los campos geográficos: departamento, provincia, distrito, y ubigeo.
4. Cálculo del área distrital
    - En la proyección métrica (EPSG:32718), se calcula el área de cada distrito en kilómetros cuadrados (km^2).
    - Se crea una tabla con ubigeo, departamento, provincia, distrito y area_km2.
5. Integración demográfica
    - Se carga el archivo reporte_estadist.xlsx con información poblacional por distrito desde el 2018 hasta el 2025.
    - Se ordena el encabezado, se renombran columnas y se estandariza el código ubigeo a texto de 6 dígitos.
    - Se realiza un merge con los datos sísmicos por el campo común ubigeo.
    - Se identifica el año en que ocurrió el sismo y se utiliza para identificar la población (**En caso que el valor de la población en ese año sea nulo se busca en el año más próximo del mismo distrito**)
    - Finalmente, eliminamos las columas con la población de cada año, quedándonos con las columnas anterior a este proceso y la nueva columna adicional del población.
6. Generación del dataset final
    - El DataFrame resultante (sismos_poblacion) combina información de:
    - Sismo: magnitud, fecha, profundidad, coordenadas
    - Ubicación geográfica: distrito, provincia, departamento, área km^2
    - Demografía: población del distrito

```
import osmnx as ox
import geopandas as gpd
from shapely.geometry import Point
import os
```

```
# Crear una copia del DataFrame original
sismos = df.copy()
```

```
# Separar la columan "lat_lon" en dos columnas "latitud" y "longitud"
coordenates = sismos["lat_lon"].str.split(",", expand=True)

# Renombrar las nuevas columnas
coordenates.columns = ["latitud", "longitud"]

# Unir con el DataFrame original
sismos = pd.concat([sismos, coordenates], axis=1)
```

```
# Crear geometría (latitud y longitud deben estar en columnas)
sismos["geometry"] = sismos.apply(lambda row: Point(row["longitud"], row["latitud"]), axis=1)
gdf_sismos = gpd.GeoDataFrame(sismos, geometry="geometry", crs="EPSG:4326")
```

```
# Cargar el GeoPackage de distritos
distritos_general = gpd.read_file("../data/raw/DISTRITO.gpkg")
distritos = distritos_general.to_crs("EPSG:4326")
distritos_km = distritos_general.to_crs("EPSG:32718")
```

```
# Hacer el join espacial para asignar distritos a los sismos
sismos_con_dist = gpd.sjoin(gdf_sismos, distritos, how="left", predicate="within")
sismos_con_dist.rename(columns={
    "NOMBDEP": "departamento",
    "NOMBPROV": "provincia",
    "NOMBDIST": "distrito",
    "UBIGEO": "ubigeo"
}, inplace=True)

# Seleccionar solo las columnas necesarias
sismos_con_dist = sismos_con_dist[[
    "reporte", "referencia", "fecha_hora", "magnitud", "profundidad",
    "latitud", "longitud", "ubigeo", "geometry"
]]
```

```
# Calcular área en km²
distritos_km["area_km2"] = distritos_km.geometry.area / 1e6
```

```
# Seleccionar y renombrar columnas relevantes
distritos_km = distritos_km.rename(columns={
    "NOMBDEP": "departamento",
    "NOMBPROV": "provincia",
    "NOMBDIST": "distrito",
    "UBIGEO": "ubigeo"
})[["departamento", "provincia", "distrito", "ubigeo", "area_km2"]]

# Verificar
distritos_km.head()
```

	departamento	provincia	distrito	ubigeo	area_km2
0	APURIMAC	ANDAHUAYLAS	JOSE MARIA ARGUEDAS	030220	175.238434
1	APURIMAC	AYMARAES	TINTAY	030415	142.422772
2	APURIMAC	AYMARAES	LUCRE	030409	103.775032
3	APURIMAC	ANDAHUAYLAS	SAN MIGUEL DE CHACCRAMPA	030214	84.908979
4	APURIMAC	ANDAHUAYLAS	HUAYANA	030206	95.301065

```
# Unir sismos con km² de distritos
sismos_poblacion = sismos_con_dist.merge(distritos_km, on="ubigeo", how="left")
```

```
# Verificar la union de tablas
sismos_poblacion.head()
```

	reporte	referencia	fecha_hora	magnitud	profundidad	latitud	longitud	ubigeo	geometry	departamento	provincia	distrito	area_km2
0	IGP/CENSIS/RS\n2025-0718	42 km al NO de Satipo, Satipo - Junín	29/10/2025 05:45:16	3.6	16 Km	-10.91	-74.81	120303	POINT (-74.81 -10.91)	JUNIN	CHANCHAMAYO	PICHANAQUI	1242.381349
1	IGP/CENSIS/RS\n2025-0717	46 km al NO de Satipo, Satipo - Junín	29/10/2025 04:45:21	3.8	18 Km	-10.89	-74.84	120303	POINT (-74.84 -10.89)	JUNIN	CHANCHAMAYO	PICHANAQUI	1242.381349
2	IGP/CENSIS/RS\n2025-0716	71 km al SO de Tacna, Tacna - Tacna	29/10/2025 03:22:58	3.8	37 Km	-18.56	-70.6	NaN	POINT (-70.6 -18.56)	NaN	NaN	NaN	NaN

```
# Cargar el archivo de población
poblacion = pd.read_excel('../data/raw/reporte_estadist.xlsx')
```

```
# Eliminar filas con Ubigeo nulo
poblacion = poblacion[poblacion['Ubigeo'].notna()]

# Renombrar columnas
poblacion = poblacion.rename(columns={
    "Ubigeo": "ubigeo",
    "Unnamed: 8": "2020",
    "Unnamed: 9": "2021",
    "Unnamed: 10": "2022",
    "Unnamed: 11": "2023",
    "Unnamed: 12": "2024",
    "Unnamed: 13": "2025"
})

# Mantener solo las columnas necesarias
poblacion = poblacion[["ubigeo", "2020", "2021", "2022", "2023", "2024", "2025"]]

# Reemplazar valores 0 por None
poblacion.replace({0: np.nan}, inplace=True)
```

```
# Convertir ubigeo a entero y luego a texto de 6 dígitos
poblacion["ubigeo"] = (
    poblacion["ubigeo"]
    .astype(float)
    .astype("Int64")
    .astype(str)
    .str.zfill(6)
)
```

```
# Unir los datos de población con los sismos
sismos_poblacion = sismos_poblacion.merge(
    poblacion,
    on="ubigeo",
    how="left"
)
```

```
# Verificar la union de tablas
sismos_poblacion.head()
```

	reporte	referencia	fecha_hora	magnitud	profundidad	latitud	longitud	ubigeo	geometry	departamento	provincia	distrito	area_km2	2020	2021	2022	202
0	IGP/CENSIS/RS\n2025-0718	42 km al NO de Satipo, Satipo - Junín	29/10/2025 05:45:16	3.6	16 Km	-10.91	-74.81	120303	POINT (-74.81 -10.91)	JUNIN	CHANCHAMAYO	PICHANAQUI	1242.381349	42869.0	42209.0	41498.0	40752.
1	IGP/CENSIS/RS\n2025-0717	46 km al NO de Satipo, Satipo - Junín	29/10/2025 04:45:21	3.8	18 Km	-10.89	-74.84	120303	POINT (-74.84 -10.89)	JUNIN	CHANCHAMAYO	PICHANAQUI	1242.381349	42869.0	42209.0	41498.0	40752.
2	IGP/CENSIS/RS\n2025-0716	71 km al SO de Tacna, Tacna - Tacna	29/10/2025 03:22:58	3.8	37 Km	-18.56	-70.6	NaN	POINT (-70.6 -18.56)	NaN	NaN	NaN	NaN	NaN	NaN	NaN	Na
3	IGP/CENSIS/RS\n2025-0715	16 km al NO de Zorritos, Contralmirante Villar...	28/10/2025 15:44:06	3.8	20 Km	-3.61	-80.79	NaN	POINT (-80.79 -3.61)	NaN	NaN	NaN	NaN	NaN	NaN	NaN	Na
4	IGP/CENSIS/RS\n2025-0714	23 km al O de Marcona, Nasca - Ica	26/10/2025 23:52:15	4.1	34 Km	-15.42	-75.37	NaN	POINT (-75.37 -15.42)	NaN	NaN	NaN	NaN	NaN	NaN	NaN	Na

```
# Agregar columna de año extraída de fecha_hora
sismos_poblacion["anio"] = pd.to_datetime(sismos_poblacion["fecha_hora"], errors="coerce", dayfirst=True).dt.year
```

```
def obtener_poblacion_fila(row):
    year = row["anio"]
    columnas_anio = [2020, 2021, 2022, 2023, 2024, 2025]

    # Si el año del sismo está dentro del rango
    if year in columnas_anio and not pd.isna(row[str(year)]):
        return row[str(year)]

    # Si no hay dato exacto, buscar el más cercano con valor no nulo
    disponibles = [a for a in columnas_anio if not pd.isna(row[str(a)])]
    if not disponibles:
        return None
    closest_year = min(disponibles, key=lambda x: abs(x - year))
    return row[str(closest_year)]
```

```
# Aplicar la función
sismos_poblacion["poblacion"] = sismos_poblacion.apply(obtener_poblacion_fila, axis=1)
```

```
# Eliminar columnas de años y anio
sismos_poblacion.drop(columns=["2020", "2021", "2022", "2023", "2024", "2025","anio"], inplace=True)
```

```
# Verificar el DataFrame final
sismos_poblacion.head()
```

	reporte	referencia	fecha_hora	magnitud	profundidad	latitud	longitud	ubigeo	geometry	departamento	provincia	distrito	area_km2	poblacion
0	IGP/CENSIS/RS\n2025-0718	42 km al NO de Satipo, Satipo - Junín	29/10/2025 05:45:16	3.6	16 Km	-10.91	-74.81	120303	POINT (-74.81 -10.91)	JUNIN	CHANCHAMAYO	PICHANAQUI	1242.381349	39213.0
1	IGP/CENSIS/RS\n2025-0717	46 km al NO de Satipo, Satipo - Junín	29/10/2025 04:45:21	3.8	18 Km	-10.89	-74.84	120303	POINT (-74.84 -10.89)	JUNIN	CHANCHAMAYO	PICHANAQUI	1242.381349	39213.0
2	IGP/CENSIS/RS\n2025-0716	71 km al SO de Tacna, Tacna - Tacna	29/10/2025 03:22:58	3.8	37 Km	-18.56	-70.6	NaN	POINT (-70.6 -18.56)	NaN	NaN	NaN	NaN	NaN
3	IGP/CENSIS/RS\n2025-0715	16 km al NO de Zorritos, Contralmirante Villar...	28/10/2025 15:44:06	3.8	20 Km	-3.61	-80.79	NaN	POINT (-80.79 -3.61)	NaN	NaN	NaN	NaN	NaN
4	IGP/CENSIS/RS\n2025-0714	23 km al O de Marcona, Nasca - Ica	26/10/2025 23:52:15	4.1	34 Km	-15.42	-75.37	NaN	POINT (-75.37 -15.42)	NaN	NaN	NaN	NaN	NaN

Adquisición de datos - Parte 3 (OpenStreetMap)

Este script amplía la información sísmica y geográfica previa mediante la incorporación de información sobre infraestructuras sensibles y de respuesta ante emergencias disponibles en el entorno de cada evento sísmico. El objetivo es evaluar la proximidad de servicios esenciales (como hospitales, estaciones de policía o bomberos) respecto a los puntos donde ocurrieron los sismos, usando datos abiertos del proyecto OpenStreetMap (OSM).

🔹 **Objetivo general**

Determinar la cantidad y tipo de instituciones clave ubicadas en un radio de 10 km alrededor de cada sismo, con el fin de medir la exposición de infraestructuras críticas a eventos sísmicos y facilitar la planificación territorial y de emergencia.

🔹 **Flujo general del proceso**

1. Carga de datos sísmicos
    - Se importa el archivo sismos_igp_2.csv (producto de la fase anterior).
    - Se construye la columna geometry y se transforma en un GeoDataFrame con el sistema de referencia EPSG:4326 (coordenadas geográficas).
2. Definición de categorías de infraestructura
    - Se establece un diccionario categorias con los principales grupos de servicios a analizar, definidos por etiquetas (amenity) de OpenStreetMap:
        - Salud: hospitales, clínicas, policlínicos, SISOL, postas y otros centros de salud.
        - Bomberos: estaciones de bomberos.
        - Policía: comisarías o unidades policiales.
3. Obtención de instituciones cercanas (OSM)

- Mediante la función obtener_instituciones_cercanas, se consulta la API de OSMnx (ox.features_from_point) para cada categoría y cada punto sísmico.
- Se define un radio de búsqueda de 10 km (10 000 metros) alrededor de la ubicación del sismo.
- Nos aseguramos que el lugar encontrado tenga terminos que nos aseguren que corresponden a los centros mencionados en el anterior punto.
- Si no se encuentran instituciones en una categoría, se maneja la excepción InsufficientResponseError para continuar sin interrupciones.
- **Nota 1: El asegurar el nombre de los centros encontrados genera una mayor demora en la extracción de los datos, pero nos ayuda a asegurar que el establecimiento encontrado tenga más posibilidades de soportar concentrtaciones de personas antes sismos de magnitud considerable**
- **Nota 2: Se validó de manera manual varios casos obtenidos por el OSM, todos esos conincidieron con lo encontrado en google maps**

4. Procesamiento y conteo de instituciones
   - La función procesar_sismo ejecuta los siguientes pasos para cada sismo:
     - Verifica si el punto ya fue procesado (para evitar reprocesos).
     - Obtiene las instituciones dentro del radio definido.
     - Calcula el conteo por categoría (salud, bomberos, policia).
     - Registra los resultados en un archivo incremental conteo_instituciones.csv para evitar reprocesos.
5. Ejecución iterativa
   - Se recorre cada sismo del GeoDataFrame para aplicar el proceso de búsqueda y conteo.
   - Este enfoque modular permite pausar y continuar el proceso sin perder los resultados previos.
6. Integración final
   - Se cargan los conteos resultantes y se realiza un merge con los datos sísmicos, conservando las coordenadas (latitud, longitud) como clave de unión.
   - El resultado es un nuevo dataframe, que agrega a cada evento sísmico los valores de:
     - Número de hospitales o clínicas cercanas (salud)
     - Estaciones de bomberos (bomberos)
     - Comisarías o unidades policiales (policia)

```
# Sacamos una copia del dataframe
sismos = sismos_poblacion.copy()
```

```
# Asegurarse de que latitud y longitud sean numéricos
gdf_sismos["latitud"] = pd.to_numeric(gdf_sismos["latitud"], errors="coerce")
gdf_sismos["longitud"] = pd.to_numeric(gdf_sismos["longitud"], errors="coerce")
```

```
# Definir categorías de interés
categorias = {
    "salud": {
        "amenity": ["hospital", "clinic", "polyclinic"],
        "healthcare": ["hospital", "clinic"]
    },
    "bomberos": {
```

```
        "amenity": ["fire_station"]
    },
    "policia": {
        "amenity": ["police"]
    }
}
```

```
# Función para obtener instituciones cercanas
def obtener_instituciones_cercanas(lat, lon, categorias, radio=10000):
    gdfs = []
    for nombre_cat, tags in categorias.items():
        # Obtener datos de OSM
        try:
            # Buscar instituciones cercanas
            gdf = ox.features_from_point((lat, lon), tags=tags, dist=radio)
            if not gdf.empty:
                gdf["categoria"] = nombre_cat
                gdfs.append(gdf)

        # Manejar el caso cuando no se encuentran resultados
        except ox._errors.InsufficientResponseError:
            print(f"No se encontraron resultados para {nombre_cat}")

    if not gdfs:
        print("No se encontraron instituciones en este radio.")
        return gpd.GeoDataFrame(columns=["name", "amenity", "categoria", "geometry"])

    # Combinar resultados
    instituciones_cerca = gpd.GeoDataFrame(pd.concat(gdfs, ignore_index=True))

    # Limpiar datos

    # Asegurar columnas clave
    for col in ["name", "amenity", "categoria", "geometry"]:
        if col not in instituciones_cerca.columns:
            instituciones_cerca[col] = None

    # Mantener registros con nombre o etiqueta significativa
    instituciones_cerca = instituciones_cerca[
        instituciones_cerca["name"].notna() | instituciones_cerca["amenity"].notna()
    ]

    # Mantener puntos y polígonos principales
    instituciones_cerca = instituciones_cerca[
        instituciones_cerca.geometry.type.isin(["Point", "Polygon"])
    ]

    # Eliminar duplicados por nombre
    instituciones_cerca = instituciones_cerca.drop_duplicates(subset=["name"])

    # Definir patrones de búsqueda para cada categoría
    filtros = {
        "salud": r"hospital|clínica|clinica|policlínico|policlinico|sisol|posta|minsa|essalud|centro de salud|centro medico|centro médico|puesto de salud|centro de atencion|establecimi
        "policia": r"comisaría|comisaria|policial",
        "bomberos": r"bomberos|compañia|compañía"
    }

    # Aplicar filtros específicos por categoría
    filtradas = []
    for cat, patron in filtros.items():
        subset = instituciones_cerca[instituciones_cerca["categoria"] == cat]
```

```
        if "name" in subset.columns:  # Asegurar que la columna exista
            subset = subset[subset["name"].str.contains(patron, case=False, na=False)] # Filtrar por patrón
        filtradas.append(subset)

    # Si la lista no está vacía, crear el GeoDataFrame final
    if filtradas:
        instituciones_filtradas = gpd.GeoDataFrame(pd.concat(filtradas, ignore_index=True))
    # Si no hay filtradas, retornar GeoDataFrame vacío con columnas correctas
    else:
        instituciones_filtradas = gpd.GeoDataFrame(columns=["name", "amenity", "categoria", "geometry"])

    return instituciones_filtradas
```

```
# Función para procesar un sismo y guardar conteo en CSV
def procesar_sismo(sismo, categorias, radio=10000, archivo_csv="../data/raw/conteo_instituciones.csv"):
    lat, lon = sismo.latitud, sismo.longitud

    # Verificar si ya fue procesado y si el archivo no está vacío
    if os.path.exists(archivo_csv):
        df_existente = pd.read_csv(archivo_csv)
        if ((df_existente["latitud"] == lat) & (df_existente["longitud"] == lon)).any():
            print(f"Punto ({lat}, {lon}) ya procesado, se omite.")
            return

    # Obtener instituciones cercanas
    instituciones_cerca = obtener_instituciones_cercanas(lat, lon, categorias, radio)

    # Conteo de instituciones por categoría, manejar caso vacío
    conteo = instituciones_cerca["categoria"].value_counts() if not instituciones_cerca.empty else pd.Series()
    # Reordenar e incluir todas las categorias, incluso si el conteo es 0
    conteo = conteo.reindex(categorias.keys(), fill_value=0)

    # Crear fila con lat/lon + conteos
    fila = {"latitud": lat, "longitud": lon}
    for cat in categorias.keys():
        fila[cat] = conteo[cat]

    # Guardar resultados en CSV
    df_fila = pd.DataFrame([fila])
    if os.path.exists(archivo_csv):
        df_fila.to_csv(archivo_csv, mode="a", header=False, index=False)
    else:
        df_fila.to_csv(archivo_csv, index=False)

    print(f"Procesado sismo en ({lat}, {lon}) — resultados guardados en {archivo_csv}")
```

```
# Procesar cada sismo
for _, sismo in gdf_sismos.iterrows():
    procesar_sismo(sismo, categorias, radio=10000)
```

```
# Cargar el conteo final de instituciones
instituciones = pd.read_csv("../data/raw/conteo_instituciones.csv")
```

```
# Verificar el dataframe cargado
instituciones.head()
```

	latitud	longitud	salud	bomberos	policia
0	-10.91	-74.81	1	0	1
1	-10.89	-74.84	1	0	1
2	-18.56	-70.60	0	0	0
3	-3.61	-80.79	0	0	0
4	-15.42	-75.37	0	0	0

```
# Asegurarse de que latitud y longitud sean numéricos para luego unir
sismos["latitud"] = pd.to_numeric(sismos["latitud"], errors="coerce")
sismos["longitud"] = pd.to_numeric(sismos["longitud"], errors="coerce")
```

```
# Merge de los datos de sismos con el conteo de instituciones
sismos_poblacion_conteo = sismos.merge(
    instituciones,
    on=["latitud", "longitud"],
    how="left"
)
```

```
# Verificar el dataframe final
sismos_poblacion_conteo.head()
```

	reporte	referencia	fecha_hora	magnitud	profundidad	latitud	longitud	ubigeo	geometry	departamento	provincia	distrito	area_km2	poblacion	salud	bomberos	p
0	IGP/CENSIS/RS\n2025-0718	42 km al NO de Satipo, Satipo - Junín	29/10/2025 05:45:16	3.6	16 Km	-10.91	-74.81	120303	POINT (-74.81 -10.91)	JUNIN	CHANCHAMAYO	PICHANAQUI	1242.381349	39213.0	1	0	
1	IGP/CENSIS/RS\n2025-0717	46 km al NO de Satipo, Satipo - Junín	29/10/2025 04:45:21	3.8	18 Km	-10.89	-74.84	120303	POINT (-74.84 -10.89)	JUNIN	CHANCHAMAYO	PICHANAQUI	1242.381349	39213.0	1	0	
2	IGP/CENSIS/RS\n2025-0716	71 km al SO de Tacna, Tacna - Tacna	29/10/2025 03:22:58	3.8	37 Km	-18.56	-70.60	NaN	POINT (-70.6 -18.56)	NaN	NaN	NaN	NaN	NaN	0	0	
3	IGP/CENSIS/RS\n2025-0715	16 km al NO de Zorritos, Contralmirante Villar...	28/10/2025 15:44:06	3.8	20 Km	-3.61	-80.79	NaN	POINT (-80.79 -3.61)	NaN	NaN	NaN	NaN	NaN	0	0	

Limpieza de datos

En esta fase se realiza el proceso de depuración y normalización final del conjunto de datos sísmicos integrados (sismos_igp_3.csv), con el propósito de obtener una base de datos lista.

🔹 **Objetivo general**

Optimizar la calidad del dataset mediante la eliminación de redundancias, tratamiento de valores faltantes, normalización de unidades y creación de variables derivadas que mejoren la capacidad analítica del conjunto de datos.

🔹 **Flujo general del proceso**

1. depuración inicial
    - Se eliminan columnas que no aportan valor directo al análisis, tales como: reporte, referencia, geometry, ubigeo.

2. Corrección de ubicaciones sin departamento
    - Algunos eventos sísmicos se ubican fuera del territorio continental peruano o sobre el mar, generando valores NaN en los campos de ubicación.
    - Se define un rango geográfico aproximado del mar peruano (lat: -19 a -3, lon: -84 a -70) para clasificar dichos eventos:
        - MAR: sismos ocurridos dentro del rango marítimo peruano.
        - EXTRANJERO_{PAIS}: Se definen promedios de los limites fronterizos de cada país limítrofe y su dominio marítimo.
    - Estas etiquetas se asignan a las columnas departamento, provincia y distrito para una categorización homogénea.
3. Cálculo de densidad poblacional
    - Se crea la variable densidad como el cociente entre población y área:

      $$ densidad = \frac {poblacion} {area_{km^2}} $$
    - Esta métrica permite estimar la exposición poblacional potencial ante eventos sísmicos según la zona.
4. Clasificación de magnitud
    - Se construye una variable categórica rango_magnitud con tres niveles de intensidad:
        - Leve: $magnitud < 4.5$
        - Moderada: $4.5 \ge magnitud < 6$
        - Fuerte: $magnitud \ge 6$
    - Esta clasificación facilita el análisis cualitativo y la segmentación de riesgos.
5. Tratamiento de valores faltantes y formatos
    - Los valores ausentes en area_km2, poblacion y densidad se reemplazan por 0.
    - Se eliminan textos como " Km" de la columna profundidad para convertirla a tipo numérico (int).
    - Los valores de area_km2 y densidad se redondean a 4 decimales para mantener consistencia.
    - Se convierte fecha_hora a tipo datetime usando el formato día/mes/año.
6. Conversión de tipos de datos
    - Las variables numéricas (poblacion, densidad, profundidad) se transforman a tipos enteros o flotantes según corresponda, asegurando compatibilidad con análisis estadísticos y modelado.
7. Exportación del dataset final
    - Se guarda el resultado como IGP_clean.csv, con codificación utf-8-sig.
    - Este archivo constituye la versión limpia, normalizada y enriquecida del conjunto sísmico completo listo para análisis exploratorio y visualización.

```
# Sacamos una copia del dataframe
df = sismos_poblacion_conteo.copy()
```

```
# Eliminamos las columnas innecesarias
df.drop(columns=['reporte','referencia', 'geometry', 'ubigeo'], inplace=True)
```

```
# Definir los límites aproximados del mar peruano
lon_min, lon_max = -84, -70
lat_min, lat_max = -19, -3
```

```
# crear mascara de mar peruano
es_mar = (
    df["departamento"].isna() &
    df["longitud"].between(lon_min, lon_max) &
    df["latitud"].between(lat_min, lat_max)
)

# definir extranjero general
es_extranjero = df["departamento"].isna() & (~es_mar)

# mascaras con las fronteras
es_chile = es_extranjero & (df["latitud"] < -17.5) & (df["longitud"] > -71.5)
es_bolivia = es_extranjero & (df["latitud"].between(-18.0, -14.0)) & (df["longitud"] > -70.2)
es_brasil = es_extranjero & (df["latitud"].between(-12.0, -5.0)) & (df["longitud"] > -70.0)
es_ecuador = es_extranjero & (df["latitud"] > -4.0) & (df["longitud"] > -81.3)
es_colombia = es_extranjero & (df["latitud"] > -3.0) & (df["longitud"] > -76.0)

# Asignar etiquetas
df.loc[es_mar, ["departamento", "provincia", "distrito"]] = ["MAR", "MAR", "MAR"]
df.loc[es_chile, ["departamento", "provincia", "distrito"]] = ["FRONTERA_CHILE", "FRONTERA_CHILE", "FRONTERA_CHILE"]
df.loc[es_bolivia, ["departamento", "provincia", "distrito"]] = ["FRONTERA_BOLIVIA", "FRONTERA_BOLIVIA", "FRONTERA_BOLIVIA"]
df.loc[es_brasil, ["departamento", "provincia", "distrito"]] = ["FRONTERA_BRASIL", "FRONTERA_BRASIL", "FRONTERA_BRASIL"]
df.loc[es_ecuador, ["departamento", "provincia", "distrito"]] = ["FRONTERA_ECUADOR", "FRONTERA_ECUADOR", "FRONTERA_ECUADOR"]
df.loc[es_colombia, ["departamento", "provincia", "distrito"]] = ["FRONTERA_COLOMBIA", "FRONTERA_COLOMBIA", "FRONTERA_COLOMBIA"]

# Los que quedan como extranjero
df.loc[es_extranjero & ~(es_chile | es_bolivia | es_brasil | es_ecuador | es_colombia),
       ["departamento", "provincia", "distrito"]] = ["EXTRANJERO", "EXTRANJERO", "EXTRANJERO"]
```

```
# Verificar conteos por frontera
df[df['departamento'].str.startswith('FRONTERA')]['departamento'].value_counts()
```

```
departamento
FRONTERA_CHILE      112
FRONTERA_ECUADOR     37
FRONTERA_BOLIVIA     22
Name: count, dtype: int64
```

```
# Calcular densidad poblacional
df["densidad"] = df["poblacion"] / df["area_km2"]

# Clasificar magnitud
df["rango_magnitud"] = df["magnitud"].map(
    lambda x: "Fuerte" if x >= 6 else "Leve" if x < 4.5 else "Moderada"
)
```

```
# Llenar valores NaN en area_km² con 0
df["area_km2"] = df["area_km2"].fillna(0)
```

```
# Eliminar ' Km' de la columna 'profundidad'
df['profundidad'] = df['profundidad'].str.replace(' Km', '')
```

```
# Redondear valores de area_km2 y densidad a 4 decimales
df['area_km2'] = df['area_km2'].round(4)
df['densidad'] = df['densidad'].round(4)
```

```
# Convertir 'fecha_hora' a formato datetime
df['fecha_hora'] = pd.to_datetime(df['fecha_hora'], errors='coerce', dayfirst=True)
```

```
# Convertir 'profundidad' a entero
df['profundidad'] = df['profundidad'].astype(float).astype(int)

# Llenar valores NaN en 'poblacion' con 0 y convertir a entero
df['poblacion'] = df['poblacion'].fillna(0).astype(int)

# Llenar valores NaN en 'poblacion' con 0
df['densidad'] = df['densidad'].fillna(0)
```

```
# Verificar el dataframe final
df.head()
```

	fecha_hora	magnitud	profundidad	latitud	longitud	departamento	provincia	distrito	area_km2	poblacion	salud	bomberos	policia	densidad	rango_magnitud
0	2025-10-29 05:45:16	3.6	16	-10.91	-74.81	JUNIN	CHANCHAMAYO	PICHANAQUI	1242.3813	39213	1	0	1	31.5628	Leve
1	2025-10-29 04:45:21	3.8	18	-10.89	-74.84	JUNIN	CHANCHAMAYO	PICHANAQUI	1242.3813	39213	1	0	1	31.5628	Leve
2	2025-10-29 03:22:58	3.8	37	-18.56	-70.60	MAR	MAR	MAR	0.0000	0	0	0	0	0.0000	Leve
3	2025-10-28 15:44:06	3.8	20	-3.61	-80.79	MAR	MAR	MAR	0.0000	0	0	0	0	0.0000	Leve
4	2025-10-26 23:52:15	4.1	34	-15.42	-75.37	MAR	MAR	MAR	0.0000	0	0	0	0	0.0000	Leve

```
# Guardar el DataFrame final con la data ya limpia a un nuevo archivo CSV
df.to_csv("IGP_clean.csv", index=False, encoding="utf-8-sig")
```