Proyecto final de Tópicos avanzados de bases de datos

Instalación de librerias

[ ] ↳ 1 celda oculta

Importación de librerias

```
import requests
import pandas as pd
import pandas_geojson as pdg
from google.colab import userdata
from pymongo.mongo_client import MongoClient
from pymongo.server_api import ServerApi
from pymongo import InsertOne
from google.colab import drive
import json
import geopandas as gpd
import numpy as np
import folium
from folium.plugins import HeatMapWithTime
from shapely.geometry import Point, LineString, MultiLineString
from shapely import wkt
import random
import datetime
import urllib
from folium.plugins import TimestampedGeoJson
import math
```

URLs

[ ] ↳ 1 celda oculta

Creación de cliente de MongoDB

```
MONGO_DB_USER = 'admin'
MONGO_DB_PASSWORD = 'admin'

API_URL = 'https://services6.arcgis.com/kyerLIHvrND0OSya/arcgis/rest/services/PeajesODAGOL/FeatureServer/0/query?outFields=*&where=
MONGO_URI = f'mongodb+srv://admin:{MONGO_DB_PASSWORD}@vias-cluster.kkrz7nv.mongodb.net/?retryWrites=true&w=majority&appName=vias-cl
DB_NAME = 'vias_db'
COLLECTION = 'vias_collection'


from pymongo.mongo_client import MongoClient
from pymongo.server_api import ServerApi

# Create a new client and connect to the server
client = MongoClient(MONGO_URI, server_api=ServerApi('1'))

# Send a ping to confirm a successful connection
try:
    client.admin.command('ping')
    print("Pinged your deployment. You successfully connected to MongoDB!")
except Exception as e:
    print(e)
```

```
Pinged your deployment. You successfully connected to MongoDB!
```

Descarga de datos

[ ] ↳ 5 celdas ocultas

Transformar datos (merge)

[ ] ↳ 5 celdas ocultas

Cargar datos

```
db = client['vias_db']
collection = db['vias_collection']
```

```
from pymongo import InsertOne
def load_data_in_mongo_db(data):
    # Check if collection exists
    collection.drop()
    data_to_load = [InsertOne(document={**data[doc_key]}) for doc_key in data]
    print("data_to_load: ", data_to_load[1])
    collection.bulk_write(data_to_load)
```

```
drive.mount("/content/drive")
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=True)
```

```
def store_data_in_google_drive(data):
  #Store data in json format
  with open("/content/drive/MyDrive/InviasData/data.json", "w") as f:
    data_as_json = json.dumps(data)
    f.write(data_as_json)
```

```
load_data_in_mongo_db(final_data)
store_data_in_google_drive(final_data)
```

```
data_to_load:  InsertOne({'objectid': 5, 'categoria': '4', 'codigotramo': '45CSE', 'postereferenciainicial': 0, 'distanciainici
```

Visualizar datos

```
def get_map_data():
  red_vial_line_string_data = {
      'ruta': [],
      'geometry': []
  }

  red_vial_multi_line_string_data = {
      'ruta': [],
      'geometry': []
  }

  peajes_time_series_data = []
  peajes_points_data = {
      'ruta': [],
      'tramo': [],
      'geometry': []
  }

  for ruta in final_data.values():
    if ruta["geometry"]["type"] == "LineString":
      red_vial_line_string_data["ruta"].append(ruta["nombreruta"])
      red_vial_line_string_data["geometry"].append(LineString(ruta["geometry"]["coordinates"]))
    else:
      red_vial_multi_line_string_data["ruta"].append(ruta["nombreruta"])
      red_vial_multi_line_string_data["geometry"].append(MultiLineString(ruta["geometry"]["coordinates"]))
    for tramo in ruta["tramos"]:
      for peaje in ruta["tramos"][tramo]["peajes"]:
```

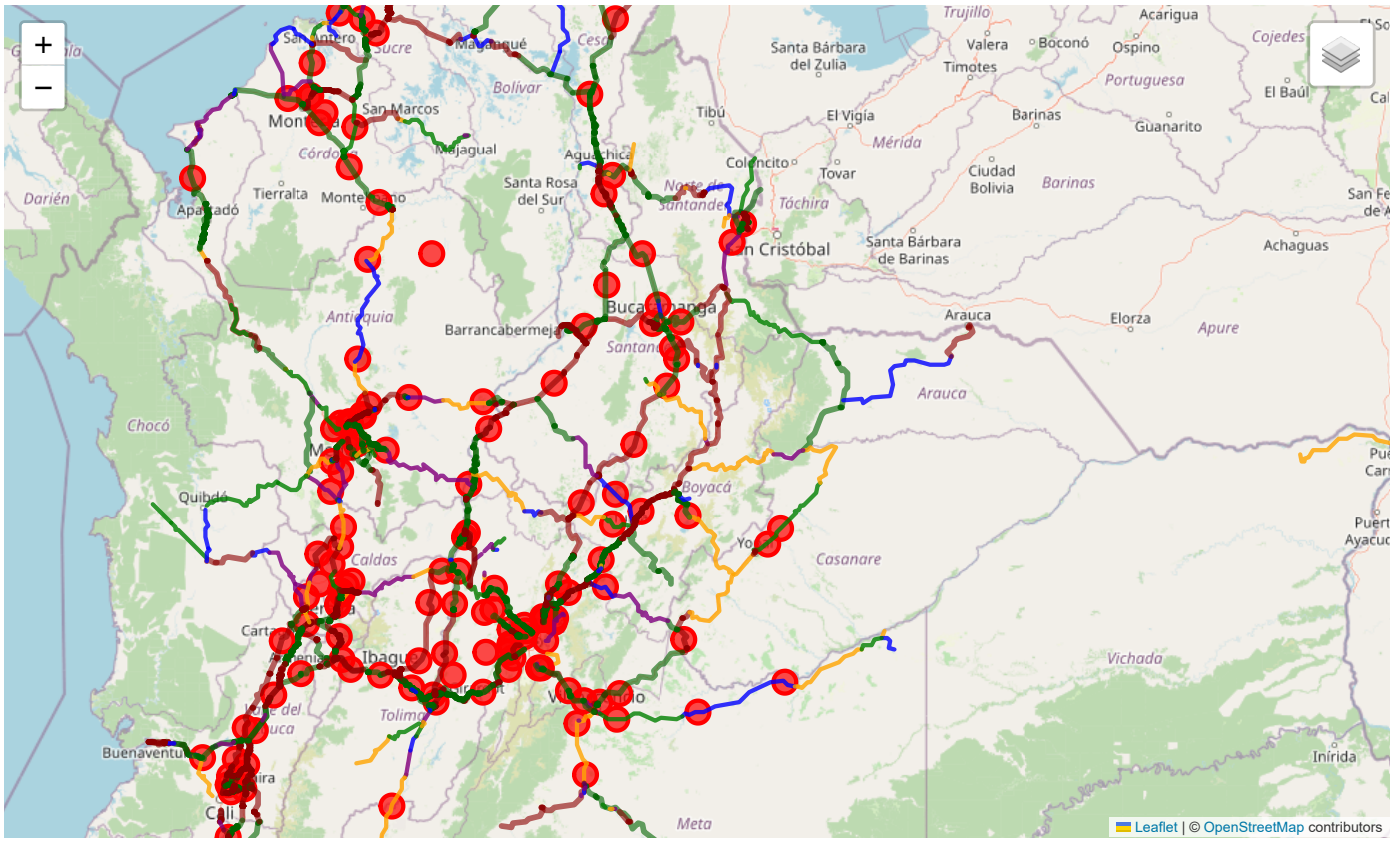
```
        peaje_data = get_data_by_peaje(peaje["nombrepeaje"].strip())

        for peaje_time_data in peaje_data:
          since_time = datetime.datetime.strptime(peaje_time_data["desde"].split("T")[0], "%Y-%m-%d")
          to_time = datetime.datetime.strptime(peaje_time_data["hasta"].split("T")[0], "%Y-%m-%d")

          if not since_time or not to_time:
            print(f"Since the merge feature since time is {since_time} and to time is {to_time}")

          peajes_time_series_data.append({ **peaje, **peaje_time_data, "nombreruta": ruta["nombreruta"], "nombretramo": tramo, "sir

        peajes_points_data["ruta"].append(ruta["nombreruta"])
        peajes_points_data["tramo"].append(tramo)
        peajes_points_data["geometry"].append(Point(peaje["geometry"]["coordinates"]))

  return gpd.GeoDataFrame(peajes_points_data, crs='EPSG:4326'), gpd.GeoDataFrame(red_vial_line_string_data, crs='EPSG:4326'), gpd.G


def create_map(peajes_points_data, red_vial_line_string_data, red_vial_multi_line_string_data, peajes_time_series_data):
  all_bounds = []
  for gdf in [peajes_points_data, red_vial_line_string_data, red_vial_multi_line_string_data]:
      bounds = gdf.total_bounds
      all_bounds.extend([(bounds[1], bounds[0]), (bounds[3], bounds[2])])

  center_lat = np.mean([bound[0] for bound in all_bounds])
  center_lon = np.mean([bound[1] for bound in all_bounds])

  m = folium.Map(
        location=[center_lat, center_lon],
        zoom_start=4,
        tiles='OpenStreetMap'
    )

  peajes_layer = folium.FeatureGroup(name='Points (Peajes)')

  for idx, row in peajes_points_data.iterrows():
      folium.CircleMarker(
          location=[row.geometry.y, row.geometry.x],
          radius=8,
          popup=f"<b>Ruta: {row['ruta']}</b><br>Tramo: {row['tramo']}",
          color='red',
          fill=True,
          fillColor='red',
          fillOpacity=0.7
      ).add_to(peajes_layer)


  red_vial_line_string_layer = folium.FeatureGroup(name='Red vial line (Red vial)')
  colors = ['blue', 'green', 'purple', 'orange']
  for idx, row in red_vial_line_string_data.iterrows():
      coords = [[point[1], point[0]] for point in row.geometry.coords]
      folium.PolyLine(
          locations=coords,
          color=colors[idx % len(colors)],
          weight=3,
          opacity=0.8,
          popup=f"<b>Ruta {row['ruta']}</b>"
      ).add_to(red_vial_line_string_layer)

  red_vial_multi_line_string_layer = folium.FeatureGroup(name='Red vial multiline (Red vial)')
  multi_colors = ['darkred', 'darkgreen']
  for idx, row in red_vial_multi_line_string_data.iterrows():
      for line in row.geometry.geoms:
          coords = [[point[1], point[0]] for point in line.coords]
          folium.PolyLine(
              locations=coords,
              color=multi_colors[idx % len(multi_colors)],
              weight=4,
              opacity=0.6,
              popup=f"<b>Ruta {row['ruta']}</b>"
          ).add_to(red_vial_multi_line_string_layer)
```

```
    peajes_layer.add_to(m)
    red_vial_line_string_layer.add_to(m)
    red_vial_multi_line_string_layer.add_to(m)
    folium.LayerControl().add_to(m)

    return m


def get_map_data_for_tolls_and_lines(tolls_data, lines_data):
    line_data = {
        'ruta_name': [],
        'type': [],
        'geometry': []
    }
    peajes_data = {
        'peaje_name': [],
        'geometry': []
    }

    for line in lines_data['features']:
      if line["geometry"]["type"] == "LineString":
        line_data['ruta_name'].append(line['properties']['nombreruta'])
        line_data['geometry'].append(LineString(line['geometry']['coordinates']))
        line_data['type'].append("LineString")
      else:
        line_data['ruta_name'].append(line['properties']['nombreruta'])
        line_data['geometry'].append(MultiLineString(line['geometry']['coordinates']))
        line_data['type'].append("MultiLineString")

    for peaje in tolls_data['features']:
      peajes_data['peaje_name'].append(peaje['properties']['nombrepeaje'])
      peajes_data['geometry'].append(Point(peaje['geometry']['coordinates']))

    return gpd.GeoDataFrame(peajes_data, crs='EPSG:4326'), gpd.GeoDataFrame(line_data, crs='EPSG:4326')



def create_map_for_tolls_and_lines(tolls_data, lines_data):
    all_bounds = []
    for gdf in [tolls_data, lines_data]:
        bounds = gdf.total_bounds
        all_bounds.extend([(bounds[1], bounds[0]), (bounds[3], bounds[2])])

    center_lat = np.mean([bound[0] for bound in all_bounds])
    center_lon = np.mean([bound[1] for bound in all_bounds])

    m = folium.Map(
        location=[center_lat, center_lon],
        zoom_start=4,
        tiles='OpenStreetMap'
    )

    peajes_layer = folium.FeatureGroup(name='Points (Peajes)')
    for idx, row in tolls_data.iterrows():
        folium.CircleMarker(
            location=[row.geometry.y, row.geometry.x],
            radius=8,
            popup=f"<b>Peaje: {row['peaje_name']}</b>",
            color='red',
            fill=True,
            fillColor='red',
            fillOpacity=0.7
        ).add_to(peajes_layer)


    lines_layes = folium.FeatureGroup(name='Red vial line (Red vial)')

    colors = ['blue', 'green', 'purple', 'orange']
    multi_colors = ['darkred', 'darkgreen']
    for idx, row in lines_data.iterrows():
        if row['type'] == "LineString":
          coords = [[point[1], point[0]] for point in row.geometry.coords]
```

```
            folium.PolyLine(
                locations=coords,
                color=colors[idx % len(colors)],
                weight=3,
                opacity=0.8,
                popup=f"<b>Ruta: {row['ruta_name']}</b>"
            ).add_to(lines_layes)
          else:
            for line in row.geometry.geoms:
              coords = [[point[1], point[0]] for point in line.coords]
              folium.PolyLine(
                  locations=coords,
                  color=multi_colors[idx % len(multi_colors)],
                  weight=4,
                  opacity=0.6,
                  popup=f"<b>Ruta: {row['ruta_name']}</b>"
              ).add_to(lines_layes)

      peajes_layer.add_to(m)
      lines_layes.add_to(m)
      folium.LayerControl().add_to(m)

      return m


    from itertools import groupby
    def prepare_heap_map(peajes_time_data_df):
        if peajes_time_data_df is None or peajes_time_data_df.empty:
            return [], []

        if 'dual_time_label' not in peajes_time_data_df.columns:
            print(f"Column 'dual_time_label' not found. Available columns: {peajes_time_data_df.columns.tolist()}")
            return [], []

        heatmap_data = []
        time_index = []

        time_groups = peajes_time_data_df.groupby('dual_time_label')

        for count, (time_label, group) in enumerate(time_groups):
            time_index.append(time_label)
            points = []

            try:
                for _, row in group.iterrows():
                    points.append([row['lat'], row['lon']])
                heatmap_data.append(points)  # Move this outside the loop
            except Exception as e:
                print(f"Error in group {count}: {e}")
                print(f"Group data: {group}")

        return heatmap_data, time_index



    def create_temporal_map(peajes_points_data, red_vial_line_string_data, red_vial_multi_line_string_data, peajes_time_series_data):
      all_bounds = []
      for gdf in [peajes_points_data, red_vial_line_string_data, red_vial_multi_line_string_data]:
          bounds = gdf.total_bounds
          all_bounds.extend([(bounds[1], bounds[0]), (bounds[3], bounds[2])])

      center_lat = np.mean([bound[0] for bound in all_bounds])
      center_lon = np.mean([bound[1] for bound in all_bounds])

      m = folium.Map(
            location=[center_lat, center_lon],
            zoom_start=4,
            tiles='OpenStreetMap'
        )

      # peajes_layer = folium.FeatureGroup(name='Points (Peajes)')
      print("Total points for peajes:", len(list(peajes_time_series_data.iterrows())))
      colors_green_to_red = [
```

```
    '#008000',  # Green
    '#32CD32',  # Lime Green
    '#9ACD32',  # Yellow-Green
    '#FFFF00',  # Yellow
    '#FFD700',  # Gold
    '#FFA500',  # Orange
    '#FF8C00',  # Dark Orange
    '#FF4500',  # Red-Orange
    '#DC143C',  # Crimson
    '#FF0000'   # Red
  ]
  max_valortarifa = peajes_time_series_data['valortarifa'].max()
  # Get max from valortarifa
  # for index, row in peajes_time_series_data.iterrows():
  #   sinceTime = row['since_time']
  #   index_color = round((int(row['valortarifa']) / int(max_valortarifa)) * 10) - 1
  #   folium.Marker(
  #           location=[row['lat'], row['lon']],
  #           popup=f"""
  #               Nombre: {row['nombrepeaje'] if row['nombrepeaje'] else 'NOT_FOUND'}<br>
  #               Ruta: {row['nombreruta'] if row['nombreruta'] else 'NOT_FOUND'}<br>
  #               Tramo: {row['nombretramo'] if row['nombretramo'] else 'NOT_FOUND'}<br>
  #               Desde: {row['since_time'] if row['since_time'] else 'NOT_FOUND'}<br>
  #               Hasta: {row['to_time'] if row['to_time'] else 'NOT_FOUND'}<br>
  #               Trafico: {row['cantidadtrafico'] if row['cantidadtrafico'] else 'NOT_FOUND'}<br>
  #               Tarifa: {row['valortarifa'] if row['valortarifa'] else 'NOT_FOUND'}<br>

  #           """,
  #           icon=folium.Icon(color=colors_green_to_red[index_color], icon='info-sign', prefix='glyphicon'),
  #       ).add_to(peajes_layer)


  # Get minimal date from since_time
  min_since_time = peajes_time_series_data['since_time'].min()
  # Get max date from to_time
  max_to_time = peajes_time_series_data['to_time'].max()

  # Create datafrom ordering the records by since_time and to_time
  peajes_time_series_data = peajes_time_series_data.sort_values(by=['since_time', 'to_time'], ascending=True)

  features = []

  for (since_time, to_time), group in peajes_time_series_data.groupby(by=['since_time', 'to_time']):

    since_time = datetime.datetime.strptime(str(since_time).split(" ")[0], '%Y-%m-%d')
    to_time = datetime.datetime.strptime(str(to_time).split(" ")[0], '%Y-%m-%d')

    time_difference = since_time - to_time

    sum_datetime = since_time + time_difference
    # valor_tarifa = float('-inf')
    popup_text = f"<br>Time: {sum_datetime}</br> Nombre peaje: {group['nombrepeaje'].iloc[0]} </br> Ruta: {group['nombreruta'].iloc
    for _, row in group.iterrows():
      # valor_tarifa = max(valor_tarifa, int(row['valortarifa']))
      popup_text += f"Categoria: {row['categoriatarifa']} - Cantidad de trafico: {row['cantidadtrafico']} - Valor de tarifa: {row['
    # index_color = math.floor((valor_tarifa / int(max_valortarifa)) * 10) - 1
    feature = {
              'type': 'Feature',
              'geometry': {
                  'type': 'Point',
                  'coordinates': [row['lon'], row['lat']],
              },
              'properties': {
                  'time': sum_datetime.isoformat(),
                  'popup': popup_text,
                  'icon': 'circle',
                  'iconstyle': {
                      'fillColor': 'red',
                      'fillOpacity': 0.6,
                      'stroke': 'true',
                      'radius': 7
                  }
```

```
                }
            }
        features.append(feature)


    TimestampedGeoJson({
        'type': 'FeatureCollection',
        'features': features,
    }, period='P1D', add_last_point=True, auto_play=False).add_to(m)


    red_vial_line_string_layer = folium.FeatureGroup(name='Red vial line (Red vial)')
    colors = ['blue', 'green', 'purple', 'orange']
    for idx, row in red_vial_line_string_data.iterrows():
        coords = [[point[1], point[0]] for point in row.geometry.coords]
        folium.PolyLine(
            locations=coords,
            color=colors[idx % len(colors)],
            weight=3,
            opacity=0.8,
            popup=f"<b>Ruta {row['ruta']}</b>"
        ).add_to(red_vial_line_string_layer)

    red_vial_multi_line_string_layer = folium.FeatureGroup(name='Red vial multiline (Red vial)')
    multi_colors = ['darkred', 'darkgreen']
    for idx, row in red_vial_multi_line_string_data.iterrows():
        for line in row.geometry.geoms:
            coords = [[point[1], point[0]] for point in line.coords]
            folium.PolyLine(
                locations=coords,
                color=multi_colors[idx % len(multi_colors)],
                weight=4,
                opacity=0.6,
                popup=f"<b>Ruta {row['ruta']}</b>"
            ).add_to(red_vial_multi_line_string_layer)


    # heatmap_data, time_index = prepare_heap_map(peajes_time_series_data)

    # heatmap = HeatMapWithTime(
    #       data=heatmap_data,
    #       index=time_index,
    #       auto_play=True,
    #       max_opacity=0.8,
    #       radius=15,
    #       gradient={0.2: 'blue', 0.4: 'lime', 0.6: 'orange', 1: 'red'},
    #       min_opacity=0.2,

    #   )

    # peajes_layer.add_to(m)
    red_vial_line_string_layer.add_to(m)
    red_vial_multi_line_string_layer.add_to(m)
    # heatmap.add_to(m)

    folium.LayerControl().add_to(m)

    print("Start to draw map")

    return m


tolls_data, lines_data = get_map_data_for_tolls_and_lines(response_peajes_json, geo_json_red_vial.to_dict())


create_map_for_tolls_and_lines(tolls_data, lines_data)
```




```
peajes_points_data, red_vial_line_string_data, red_vial_multi_line_string_data, peajes_time_series_data = get_map_data()

create_map(peajes_points_data, red_vial_line_string_data, red_vial_multi_line_string_data, peajes_time_series_data)
```

```
create_temporal_map(peajes_points_data, red_vial_line_string_data, red_vial_multi_line_string_data, peajes_time_series_data)
```

```
Total points for peajes: 354885
Start to draw map
```