Geo-Dashboard para microdatos móviles de medio ambiente

Herramientas y Dependencias ocupadas

El Geo-Dashboard para microdatos móviles de medio ambiente utiliza un stack tecnológico enfocado en análisis de datos geoespaciales y visualización interactiva. La selección de herramientas está optimizada para el procesamiento eficiente de datos georreferenciados y la generación de interfaces web responsivas.

Lenguaje principal
Python es el lenguaje base del proyecto, elegido por su ecosistema robusto para ciencia de datos y análisis geoespacial. Ofrece las librerías especializadas necesarias para manejar datos ambientales con componentes geográficos.

Contenedorización
Docker se utiliza para garantizar la portabilidad y consistencia del entorno de ejecución. Esto es especialmente importante cuando se trabaja con dependencias geoespaciales que pueden variar entre sistemas operativos.

Dependencias principales

- Dash (2.18.2): Framework principal para crear la aplicación web interactiva. Permite construir dashboards reactivos sin necesidad de JavaScript, manteniendo todo el desarrollo en Python.

- GeoPandas (1.0.1): Librería fundamental para manejo de datos geoespaciales. Extiende pandas con capacidades geográficas, permitiendo operaciones espaciales como intersecciones, buffers y análisis de proximidad sobre los microdatos ambientales.

- Matplotlib (3.10.0): Motor de visualización base para generar gráficos estáticos. Proporciona control granular sobre elementos visuales y se integra perfectamente con geopandas para mapas.

- Seaborn (0.13.2): Librería de visualización estadística que complementa matplotlib. Simplifica la creación de gráficos complejos y mejora la estética por defecto de las visualizaciones.

Dependencias automáticas
Pandas se instala automáticamente como dependencia de geopandas, proporcionando las estructuras de datos fundamentales (DataFrames) para el manejo eficiente de los microdatos tabulares.

Framework

El Geo-Dashboard está construido sobre Dash, un framework de Python especializado en la creación de aplicaciones web analíticas interactivas. Esta elección tecnológica es fundamental para el proyecto, ya que permite desarrollar una interfaz web completa utilizando exclusivamente Python, sin necesidad de conocimientos en HTML, CSS o JavaScript.

¿Qué es Dash?
Dash es un framework open-source desarrollado por Plotly que está específicamente diseñado para crear aplicaciones web con visualizaciones de datos intensivas. El framework está construido sobre tres tecnologías principales: Flask, Plotly.js y React.js, lo que le proporciona tanto la robustez del backend como la interactividad moderna del frontend.

Arquitectura del framework
Dash abstrae completamente las tecnologías y protocolos web complejos, permitiendo que los desarrolladores se enfoquen únicamente en la lógica de datos y visualización. Las aplicaciones Dash se componen de dos partes fundamentales:

- Layout (Diseño): Define la apariencia y estructura visual de la aplicación
- Callbacks (Interactividad): Describe cómo responde la aplicación a las interacciones del usuario

Ventajas para microdatos ambientales
Para el procesamiento de microdatos móviles de medio ambiente, Dash ofrece ventajas específicas:

- Interactividad en tiempo real: Los callbacks permiten que los gráficos y mapas reaccionen instantáneamente a las interacciones del usuario, crucial para explorar datos geoespaciales.

- Integración nativa con Plotly: Facilita la creación de mapas interactivos y visualizaciones geoespaciales complejas que son esenciales para mostrar trayectorias GPS y datos ambientales georeferenciados.

- Flexibilidad de datos: Dash se integra perfectamente con múltiples fuentes de datos, incluyendo archivos CSV como los que procesa el geo-dashboard.

- Escalabilidad: Desde aplicaciones simples hasta dashboards analíticos complejos, Dash crece con las necesidades del proyecto.

Comunicación cliente-servidor
El framework utiliza un servidor Flask que se comunica con componentes React del frontend mediante paquetes JSON sobre peticiones HTTP. Esta arquitectura permite que la aplicación mantenga el estado y responda dinámicamente a las interacciones del usuario mientras procesa los microdatos ambientales.

Funcionamiento de una aplicación hecha con dash

Una aplicación Dash funciona mediante un flujo de datos reactivo que conecta la interfaz de usuario con el procesamiento de datos a través de componentes interconectados. El funcionamiento se basa en un ciclo continuo donde los datos fluyen desde la entrada del usuario hasta la visualización actualizada.

Estructura básica del funcionamiento
Una aplicación Dash opera con cinco componentes fundamentales que trabajan en conjunto:

1. Datos: Fuente de información que alimenta la aplicación, pueden ser archivos CSV, bases de datos o APIs.
2. Layout: Define la estructura visual usando componentes HTML y elementos interactivos.
3. Callbacks: Funciones que manejan la interactividad y conectan las entradas con las salidas.
4. Función de generar gráfico: Procesa los datos y crea visualizaciones.
5. Componente HTML: Elementos visuales que componen la interfaz de usuario.

Flujo de funcionamiento

```
import dash
from dash import html, dcc

# Instanciación de la aplicación
app = dash.Dash(__name__)

# Layout: estructura visual
app.layout = html.Div([
    dcc.Dropdown(),
    dcc.Graph()
])

# Callbacks: interactividad
@app.callback()
def update_graph():
    # Función de generar gráfico
    return figura_actualizada

# Ejecución
```

```
app.run_server()
```

Componentes HTML y Layout
El layout define la apariencia y estructura de la aplicación. Los componentes HTML más comunes incluyen:

- html.Div(): Contenedor principal para agrupar elementos
- html.H1(), html.H2(), html.H3(): Encabezados de diferentes niveles
- html.P(): Párrafos de texto
- dcc.Graph(): Gráficos interactivos
- dcc.Dropdown(): Menús desplegables

Sistema de Callbacks
Los callbacks son el corazón de la interactividad. Son funciones que se ejecutan automáticamente cuando cambia una propiedad de un componente de entrada:

```
@app.callback(
    Output(component_id='mi-grafico', component_property='figure'),
    Input(component_id='mi-dropdown', component_property='value')
)
def actualizar_grafico(valor_seleccionado):
    # Procesar datos
    # Generar gráfico
    return nueva_figura
```

Arquitectura subyacente
Dash utiliza Flask como backend, Plotly para visualizaciones y React para el frontend. Esta arquitectura permite que la aplicación funcione como una aplicación web de una sola página donde:

- El servidor Flask maneja las peticiones HTTP y procesa los datos
- Los componentes React gestionan la interfaz de usuario
- Plotly.js renderiza las visualizaciones interactivas

Ciclo de actualización
- El funcionamiento sigue un patrón reactivo:
- El usuario interactúa con un componente (dropdown, slider, botón)
- Dash detecta el cambio y ejecuta el callback correspondiente
- La función procesa los datos y genera una nueva visualización
- El componente de salida se actualiza automáticamente en la interfaz

Estructuración de la aplicación

El Geo-Dashboard sigue una arquitectura modular y organizada que separa responsabilidades en directorios especializados. Esta estructura facilita el mantenimiento, escalabilidad y colaboración en el desarrollo del proyecto.

```
dash/
├── dashboard/
│   ├── app.py
│   ├── layout.py
│   ├── assets/
│   │   ├── custom.css
│   │   └── styles.css
│   ├── components/
│   │   ├── corr_section.py
│   │   ├── decade_distribution.py
│   │   ├── download_file.py
│   │   ├── existing_data_graph.py
│   │   ├── file_upload.py
│   │   ├── header.py
│   │   ├── map.py
│   │   ├── records_hours.py
│   │   ├── stats_section.py
│   │   ├── table_section.py
│   │   └── time_series.py
│   └── utils/
│       ├── data_processing.py
│       └── process_file.py
├── Dockerfile
├── mexico.json
├── requirements.txt
└── README.md
```

Componentes principales

app.py - Orquestador principal

Es el archivo central que conecta todos los componentes del dashboard. Contiene:

- Instanciación de la aplicación Dash
- Todos los callbacks que manejan la interactividad
- Funciones de procesamiento de archivos y generación de gráficos
- Lógica de descarga del CSV procesado
- Configuración del servidor (host, puerto, debug)

layout.py - Definición de interfaz

Define la estructura visual completa de la aplicación. Importa y organiza todos los componentes HTML en un diseño coherente, estableciendo la disposición de elementos en la página.

assets/ - Estilos personalizados
Contiene archivos CSS que no se pueden aplicar directamente en Plotly:
Dash automáticamente carga estos archivos para personalizar la apariencia visual.

components/ - Componentes modulares
Directorio que separa los elementos HTML en archivos individuales para evitar amontonamiento en el layout principal:

utils/ - Procesamiento de datos
Contiene la lógica de procesamiento separada de la interfaz:

- data_processing.py: Clase principal DataProcessing con toda la lógica de limpieza y procesamiento de microdatos ambientales
- process_file.py: Funciones auxiliares de procesamiento

Arquivos de configuración
- Dockerfile: Define el entorno containerizado para garantizar portabilidad.
- mexico.json: Archivo GeoJSON con límites geográficos de México para validación de coordenadas GPS.
- requirements.txt: Lista de dependencias del proyecto.

Procesamiento de archivos al cargarlos

El procesamiento de archivos es una de las funciones más críticas del Geo-Dashboard, ya que transforma los microdatos ambientales crudos en información estructurada y georreferenciada lista para visualización. Este proceso involucra múltiples etapas de validación, limpieza y estandarización que se ejecutan automáticamente cuando el usuario carga archivos CSV.

Flujo de procesamiento

1. Callback de validación inicial

```
python
```

```python
@app.callback(Output('output-datos', 'children'),
              Input('upload-data', 'contents'),
              State('upload-data', 'filename'))
def return_filename(contents, filename):
    if not contents:
```

```
        return "Sube un archivo csv. Aegurate que tenga
headers.\nPara llenar todos los campos se necesitan las
etiquetas timestamp, latitude y longituude"
    return filename
```

Este callback valida la presencia de archivos y proporciona retroalimentación inmediata al usuario. Específicamente:

- Verifica que se haya subido contenido antes de proceder
- Informa al usuario sobre los requisitos de formato de archivo
- Especifica las columnas obligatorias: timestamp, latitude y longitude
- Retorna el nombre del archivo para confirmación visual

2. Función auxiliar de procesamiento

python

```
def helper(content_list):
    pipeline = DataProcessing(objects=[content_list],
is_object=True)
    pipeline.read_files()
    pipeline.process_data()
    return pipeline.get_final_csv()
```

La función `helper` encapsula todo el pipeline de procesamiento para cada archivo individual:

- Instancia la clase DataProcessing configurándola para trabajar con objetos en memoria (no archivos del disco)
- Ejecuta la lectura mediante `read_files()` que maneja la detección automática de headers
- Ejecuta el procesamiento completo mediante `process_data()` que incluye limpieza y validación
- Retorna el DataFrame procesado listo para uso

3. Callback principal de procesamiento

python

```
@app.callback([Output('stored-clean-csv', 'data'),
               Output('gps-datapoints', 'data')],
```

```
              Input('upload-data', 'contents'))
def format_data(contents_list):
```

Este callback orquesta todo el procesamiento de múltiples archivos y es el más complejo del sistema.

Etapas específicas del procesamiento

Decodificación de archivos

python

```
for contents in contents_list:
    _, content_string = contents.split(',')
    decoded = base64.b64decode(content_string)
    decoded = io.StringIO(decoded.decode('latin1'))
    content_list.append(decoded)
```

Convierte los archivos subidos desde el frontend:

- Separa el header del contenido usando el formato base64 de Dash
- Decodifica desde base64 para obtener el contenido binario original
- Convierte a StringIO para crear un objeto similar a archivo en memoria
- Usa codificación latin1 para manejar caracteres especiales comunes en datos ambientales

Procesamiento paralelo

python

```
with Pool(cpu_count()) as pool:
    df_list = pool.map(helper, content_list)
```

Optimiza el procesamiento mediante paralelización:

- Utiliza todos los núcleos del CPU disponibles
- Aplica la función helper a cada archivo simultáneamente
- Reduce significativamente el tiempo de procesamiento para múltiples archivos
- Retorna una lista de DataFrames procesados individualmente
-

Concatenación y preparación final

python

```
df = pd.concat(df_list, ignore_index=True)
coordinates = find_coordinates(df.columns)
if coordinates:
    temp = df.dropna(subset=[coordinates[0], coordinates[1]])
    return df.to_dict('records'), temp.to_dict('records')
```

Unifica todos los datos procesados:

- Concatena múltiples DataFrames en una estructura única
- Reinicia los índices para mantener coherencia
- Identifica las columnas de coordenadas GPS automáticamente
- Crea dos conjuntos de datos: uno completo y otro solo con coordenadas válidas
- Convierte a formato de registros para almacenamiento en componentes Dash

Procesamiento interno de la clase DataProcessing

Detección automática de headers

La clase identifica automáticamente si el archivo tiene headers usando el método `_is_data_row()`:

python

```
def _is_data_row(self, row) -> bool:
    data_count = sum(
        str(val).replace('.', '', 1).isdigit() or
isinstance(val, (int, float))
        for val in row
    )
    return data_count >= len(row) * 0.7
```

Analiza la primera fila para determinar si contiene datos numéricos (≥70%) o nombres de columnas.

Limpieza de coordenadas GPS

El procesamiento incluye validación geoespacial sofisticada:

1. Carga el archivo GeoJSON de México (`mexico.json`)

2. Convierte coordenadas a GeoDataFrame usando GeoPandas
3. Verifica que cada punto esté dentro de territorio mexicano usando intersección espacial
4. Aplica un filtro por umbral para eliminar estados con pocos puntos (< 10 por defecto)
5. Mantiene solo coordenadas válidas y marca las demás como NaN

Limpieza de valores numéricos

python

```
def clean_big_numbers(self) -> None:
    cols_to_clean = [col for col in numeric_cols if col not in
['Unix Time', coordinates]]
    self.final_df[cols_to_clean] =
self.final_df[cols_to_clean].map(
        lambda x: np.nan if x > 2_000_000 else x
    )
```

Elimina valores anómalos que excedan 2,000,000 en columnas numéricas, excluyendo Unix Time y coordenadas.

Estandarización de timestamps

python

```
def fix_timestamp(self) -> None:
    timestamp_column = self.find_timestamp()
    self.final_df[timestamp_column] =
pd.to_datetime(self.final_df[timestamp_column], errors="coerce")
```

Convierte automáticamente las columnas de tiempo a formato datetime de pandas, manejando errores mediante coerción a NaT.

Salidas del procesamiento

El sistema genera dos conjuntos de datos:

1. `stored-clean-csv`: Dataset completo con todas las filas procesadas y limpiadas
2. `gps-datapoints`: Subset solo con filas que tienen coordenadas GPS válidas para mapeo

Estos datos quedan almacenados en componentes dcc.Store para uso posterior en todas las visualizaciones del dashboard, evitando reprocesamiento y garantizando consistencia entre gráficos.

El procesamiento es completamente automático y robusto, manejando archivos con o sin headers, múltiples formatos de fecha, coordenadas inválidas y valores anómalos, proporcionando al usuario datos limpios y listos para análisis sin intervención manual.

Gráficos y su generación

El Geo-Dashboard genera múltiples tipos de visualizaciones especializadas para el análisis de microdatos ambientales móviles. Cada gráfico cumple una función específica en el proceso de exploración y comprensión de los datos georreferenciados.

Gráfico de Datos Nulos vs Datos Existentes

Tipo: Gráfico de barras agrupadas
Función: Proporciona una visión inmediata de la calidad del dataset mostrando la cantidad de datos nulos y existentes para cada columna. Este gráfico es fundamental para identificar variables con alta completitud de datos versus aquellas con muchos valores faltantes, permitiendo al usuario tomar decisiones informadas sobre qué columnas utilizar en análisis posteriores.

Gráfico de Barras por Hora

Tipo: Gráfico de barras simple
Función: Visualiza la distribución temporal de la recolección de datos mostrando cuántos registros se capturaron por hora para un día específico seleccionado por el usuario. Es especialmente útil para identificar patrones de recolección, períodos de mayor actividad del sensor móvil, o detectar interrupciones en la captura de datos durante el día.

Serie de Tiempo

Tipo: Gráfico de dispersión (scatter plot)
Función: Muestra la evolución temporal de cualquier variable numérica seleccionada por el usuario. Permite observar tendencias, patrones estacionales, valores anómalos y comportamientos temporales específicos de parámetros ambientales como PM10, temperatura, humedad, o presión atmosférica a lo largo del tiempo de muestreo.

Distribución por Decenas

Tipo: Histograma con estadísticas superpuestas

Función: Presenta la distribución estadística de valores para cualquier columna numérica seleccionada. El histograma incluye líneas de referencia para la media y mediana, además de un box plot marginal que muestra la distribución de cuartiles. Es fundamental para entender la naturaleza estadística de los datos ambientales y detectar distribuciones sesgadas o valores atípicos.

Mapa Interactivo

Tipo: Mapa de dispersión georreferenciado
Función: Visualiza la trayectoria GPS del sensor móvil sobre un mapa interactivo de OpenStreetMap. Permite aplicar filtros de color basados en cualquier variable numérica para mostrar cómo varía un parámetro ambiental a lo largo de la ruta. También soporta etiquetas personalizables que aparecen al hacer hover, mostrando múltiples variables simultáneamente en cada punto geográfico.

Matriz de Correlación

Tipo: Mapa de calor (heatmap) con anotaciones
Función: Genera una matriz de correlación visual entre las columnas numéricas seleccionadas por el usuario. Utiliza una escala de colores rojo-azul donde los valores cercanos a +1 (azul intenso) indican correlación positiva fuerte, valores cercanos a -1 (rojo intenso) indican correlación negativa fuerte, y valores cercanos a 0 (blanco) indican ausencia de correlación lineal. Las celdas están anotadas con los valores numéricos de correlación para precisión.

Características generales de visualización

Todos los gráficos están optimizados para tema oscuro con fondo negro (#121212), texto blanco (#f5f5f5) y elementos visuales contrastantes. Utilizan la biblioteca Plotly para interactividad completa, permitiendo zoom, pan, selección y exportación. Los gráficos se actualizan reactivamente mediante callbacks de Dash cuando el usuario modifica filtros, selecciones o sube nuevos datos.
Los gráficos están diseñados para complementarse entre sí, proporcionando desde análisis exploratorio básico (calidad de datos, distribuciones) hasta análisis avanzado (correlaciones, patrones geoespaciales), cubriendo el espectro completo de necesidades analíticas para microdatos ambientales móviles.