INSTITUTO TECNOLÓGICO BELTRÁN
Centro de Tecnología e Innovación

ANÁLISIS DE CÓDIGO: TP 1 Sistema de Gestión CSV/JSON

Materia: Modelizado de Minería de Datos

Docente: Fernández, David

Alumno: Oviedo, Lucas Nahuel

Carrera: Tecnicatura en Ciencia de Datos e Inteligencia Artificial

Institución: INSTITUTO SUPERIOR DE FORMACIÓN TÉCNICA N° 197

Año: 2025

1. Introducción

Este informe presenta un análisis detallado del sistema de gestión de archivos CSV y JSON desarrollado en Python. El sistema permite realizar operaciones CRUD (Crear, Leer, Actualizar, Eliminar) sobre archivos de datos estructurados, proporcionando una interfaz de línea de comandos interactiva para la manipulación de registros. El análisis se centra en la arquitectura del código, las funciones implementadas y la lógica de negocio del programa.

2. Descripción del Problema

El problema abordado consiste en desarrollar un sistema que permita gestionar archivos de datos en formatos CSV y JSON de manera eficiente. Los usuarios necesitan poder cargar múltiples archivos, visualizar registros en formato tabular, agregar nuevos registros, modificar registros existentes y eliminar registros específicos. El sistema debe manejar diferentes tipos de datos y proporcionar una experiencia de usuario intuitiva a través de menús interactivos.

2.1 Archivos Analizados

- **mainCSV_v4.py:** Archivo principal que contiene la lógica del menú interactivo y la coordinación de operaciones
- **funcionesCSV_v3.py:** Módulo de funciones que implementa las operaciones CRUD para archivos CSV y JSON

3. Implementación del Sistema

3.1 Estructura General

El sistema está dividido en dos archivos principales: un módulo de funciones (`funcionesCSV_v3.py`) que contiene las operaciones de bajo nivel, y un archivo principal (`mainCSV_v4.py`) que maneja la interfaz de usuario y la lógica de aplicación.

3.2 Funciones del Módulo funcionesCSV_v3.py

Este módulo contiene las funciones básicas para manipular archivos CSV y JSON:

```
# Funciones para CSV
def csv_a_diccionarios(archivo):
    """Lee un archivo CSV y retorna una lista de diccionarios"""

def agregar_registro(archivo, nuevo_registro):
    """Agrega un nuevo registro al archivo CSV"""

def borrar_por_indice(archivo, indices):
    """Borra registros por sus índices"""

def modificar_interactivo(archivo):
    """Función interactiva para modificar registros"""

# Funciones para JSON
def json_a_diccionarios(archivo):
    """Lee un archivo JSON y retorna una lista de diccionarios"""

def agregar_registro_json(archivo, nuevo_registro):
    """Agrega un nuevo registro al archivo JSON"""

def borrar_por_indice_json(archivo, indices):
    """Borra registros por sus índices en JSON"""

def modificar_interactivo_json(archivo):
    """Función interactiva para modificar registros en JSON"""
```

3.3 Lógica del Programa Principal

El archivo principal implementa un menú interactivo con seis opciones principales:

```
def main():
    archivos_cargados = {}

    while True:
        mostrar_menu(archivos_cargados)

        opcion = input("\nSeleccione una opción (1-6): ").strip()

        match opcion:
            case "1":  # Cargar archivos
```

```
                nuevos_archivos = cargar_archivos()
                archivos_cargados.update(nuevos_archivos)

            case "2":  # Leer y mostrar registros
                # Implementación de lectura y visualización

            case "3":  # Agregar nuevo registro
                # Implementación de inserción

            case "4":  # Borrar registro
                # Implementación de eliminación

            case "5":  # Modificar registro
                # Implementación de modificación

            case "6":  # Salir
                break
```

3.4 Gestión de Archivos y Formatos

El sistema determina automáticamente el formato de archivo basado en la extensión:

```
def determinar_formato(archivo):
    if archivo.lower().endswith('.json'):
        return 'json'
    elif archivo.lower().endswith('.csv'):
        return 'csv'
    else:
        return None
```

3.5 Interfaz de Usuario

La función `mostrar_registros_como_tabla` crea una representación visual atractiva de los datos:

```
def mostrar_registros_como_tabla(registros, archivo):
    # Calcular anchos de columna
    anchos = {}
    for col in columnas:
        anchos[col] = max(len(col), max(len(str(registro.get(col, ''))) for registro in
registros))

    # Imprimir tabla con bordes
    imprimir_linea()
    print("|" + "|".join(f" {col:<{anchos[col]}} " for col in columnas) + "|")
    imprimir_linea()

    for registro in registros:
        fila = "|" + "|".join(f" {str(registro.get(col, '')):<{anchos[col]}} " for col in
columnas) + "|"
        print(fila)
```

3.6 Gestión de IDs y Localidades

El sistema incluye lógica especial para campos de ID y selección de localidades:

```
def pedir_datos_registro(campos, archivo_actual, localidades_data=None):
    for campo in campos:
        if campo.lower().startswith('id_') and campo != 'id_localidad':
            registros_existentes = csv_a_diccionarios(archivo_actual) if
```

```
os.path.exists(archivo_actual) else []
            nuevo_id = len(registros_existentes) + 1
            registro[campo] = str(nuevo_id)
        elif campo == 'id_localidad' and localidades_data:
            registro[campo] = seleccionar_localidad(localidades_data)
```

4. Funcionalidades y Características

4.1 Carga de Archivos

Permite cargar múltiples archivos CSV/JSON simultáneamente, con validación de existencia y creación automática si no existen.

4.2 Visualización de Datos

Muestra los registros en formato tabular con bordes y alineación automática de columnas.

4.3 Operaciones CRUD

- **Crear:** Agregar nuevos registros con validación de campos
- **Leer:** Visualizar todos los registros de un archivo
- **Actualizar:** Modificar registros existentes de forma interactiva
- **Eliminar:** Borrar registros por índice con confirmación

4.4 Opciones de Guardado

Para operaciones de modificación y eliminación, el sistema ofrece dos opciones:

- Sobrescribir el archivo original
- Crear un nuevo archivo con los cambios

4.5 Manejo de Errores

Incluye manejo robusto de excepciones y validación de entradas del usuario.

5. Conclusión

El sistema de gestión CSV/JSON analizado demuestra una implementación sólida de operaciones CRUD con una arquitectura modular bien estructurada. La separación entre la lógica de negocio (funcionesCSV_v3.py) y la interfaz de usuario (mainCSV_v4.py) facilita el mantenimiento y la extensibilidad del código. Las características implementadas como la visualización tabular, el manejo automático de IDs y las opciones flexibles de guardado hacen del sistema una herramienta práctica para la manipulación de datos estructurados. El código sigue buenas prácticas de programación con manejo adecuado de errores y una interfaz intuitiva para el usuario final.