Logotipo

El contenido generado por IA puede ser incorrecto.Programación TP final

Maestría en Ciencia de Datos

Grupo 22

Integrantes:

Natalia Gabriel

Raul Marusca

Pablo Moreira

**Informe de Diseño y Funcionamiento del Sistema de Análisis de Notas**

**Visión General del Sistema**

Este sistema abarca desde la generación de información de prueba hasta el análisis y la presentación de resultados. El diseño prioriza la modularidad, la claridad del código, la integridad de los datos y una salida de información estructurada.

**Estructura del Proyecto**

El proyecto sigue una estructura de directorios clara y organizada, facilitando la separación de componentes y la gestión del proyecto.

Texto, Carta

El contenido generado por IA puede ser incorrecto.

* proyecto\_integrador/: El directorio raíz del proyecto.
* datos/: Contiene los archivos de datos generados o utilizados por el sistema.
  + estudiantes.csv: El archivo CSV generado con los datos de los estudiantes.
* src/: Contiene el código fuente principal de la aplicación.
  + \_\_init\_\_.py: Indica que src es un paquete Python, permitiendo la importación de sus módulos.
  + estudiante.py: Define la clase Estudiante y su lógica de negocio.
  + analizador.py: Contiene la clase AnalizadorNota para el procesamiento y análisis de datos.
  + main.py: El punto de entrada de la aplicación, que orquesta las operaciones.
* requirements.txt: Lista las dependencias necesarias para ejecutar el proyecto, asegurando la reproducibilidad del entorno.

Esta estructura modular es una decisión de diseño que contribuye a la organización, la mantenibilidad y la escalabilidad del sistema.

**Decisiones de Diseño Clave**

Las siguientes decisiones de diseño fueron fundamentales para la construcción y el funcionamiento del sistema:

1. Modelado de Entidades con Orientación a Objetos:
   * Se optó por crear una clase Estudiante (estudiante.py) para encapsular la información y el comportamiento asociado a cada alumno (nombre, apellido, notas, cálculo de promedio, situación de aprobación). La programación orientada a objetos (POO) mejora la cohesión y la mantenibilidad.
   * La validación de datos en el constructor de Estudiante asegura que cada objeto creado sea consistente y válido, previniendo errores lógicos en etapas posteriores del procesamiento.
   * El uso de métodos estáticos para la validación (\_validar\_nombre, \_validar\_nota) se utiliza ya que estas funciones no requieren acceso a la instancia del objeto, haciéndolas reutilizables y claras.
   * La sobreescritura del método \_\_str\_\_ en Estudiante permite una representación legible y útil del objeto al imprimirlo, facilitando la depuración y la generación de informes individuales.
2. Manejo Robusto y Flexible de Datos:
   * Para la generación de datos sintéticos (generador.py, aunque no esté explícitamente en src/ se asume su rol de creación del input), se eligió la librería Faker (configurada para español) junto con random. Esto permite crear un conjunto de datos realistas y variados de manera programática, esencial para pruebas sin depender de datos reales.
   * El formato CSV (datos/estudiantes.csv) se seleccionó para el almacenamiento de los datos generados debido a su universalidad, simplicidad y facilidad de lectura/escritura.
   * La lectura inicial de datos con pandas en la clase AnalizadorNota (analizador.py) es una decisión estratégica. pandas es una herramienta robusta y eficiente para el manejo de datos tabulares, lo que agiliza la carga y pre-procesamiento del CSV.
   * A pesar de usar pandas, se decidió transformar los datos del DataFrame en objetos Estudiante (\_crear\_estudiantes en analizador.py). Esta es una decisión híbrida que permite aprovechar la potencia de pandas para la E/S y, al mismo tiempo, aplicar la lógica de negocio y las validaciones encapsuladas en la clase Estudiante.
3. Modularidad y Organización del Código:
   * El sistema está dividido en módulos con responsabilidades claras, reflejadas en la estructura de directorios (src/ para el código, datos/ para el input/output).
   * Esta separación de responsabilidades mejora la legibilidad, facilita la depuración y promueve la reutilización de código, haciendo el sistema más mantenible y escalable.
   * La decisión de almacenar el archivo CSV en una subcarpeta datos/ es una práctica aconsejada al organizar proyectos, ya que mantiene los datos separados del código fuente.
4. Manejo de Errores y Robustez:
   * Se implementaron mecanismos de manejo de excepciones (try-except) en el constructor de AnalizadorNota para capturar errores de archivo (FileNotFoundError) o problemas generales durante la lectura del CSV. Esto hace que el sistema sea más robusto frente a entradas inválidas o ausentes.
   * Las excepciones (TypeError, ValueError) lanzadas por los métodos de validación en la clase Estudiante contribuyen a la integridad de los datos y permiten un control de flujo claro en caso de datos malformados.
5. Presentación Clara de Resultados:
   * El script main.py actúa como un controlador central que maneja el flujo de la aplicación.
   * Se decidió estructurar la salida por consola en secciones claras ("INFORME INDIVIDUAL" y "ESTADÍSTICAS GENERALES") para mejorar la legibilidad y facilitar la interpretación de los resultados por parte del usuario. El uso de to\_string(index=False) para los DataFrames asegura una impresión limpia.

En conjunto, estas decisiones de diseño resultan en un sistema que cumple su función de análisis de notas, y utiliza buenas prácticas de programación, desde el diseño de clases hasta la gestión de archivos y la presentación de resultados.