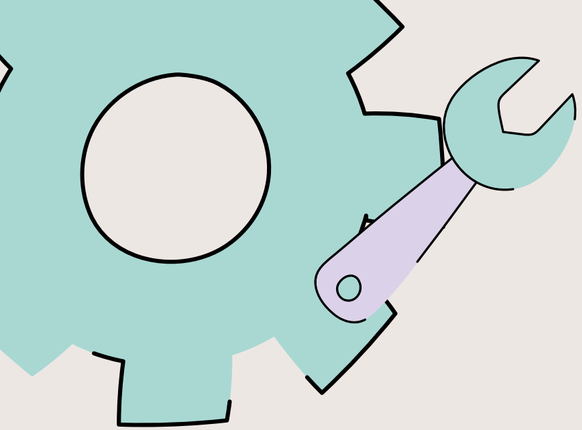
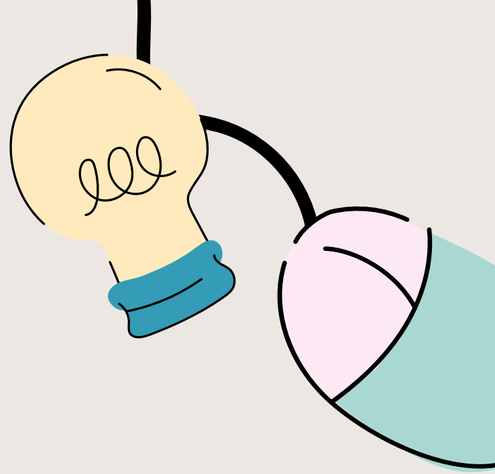
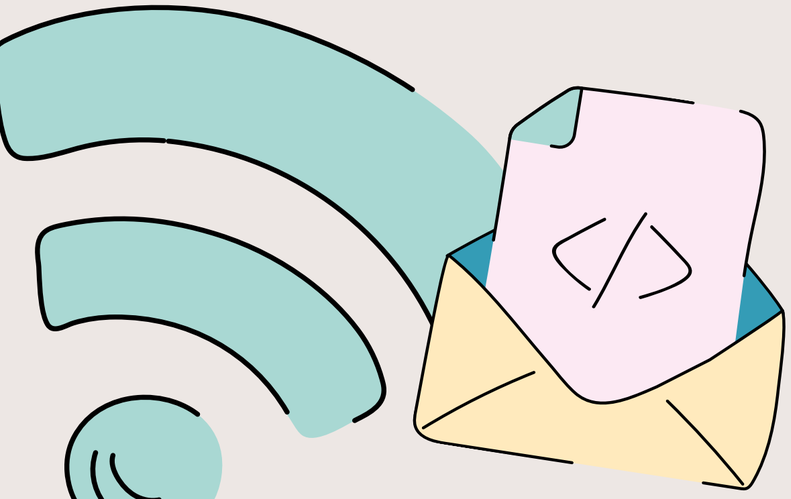
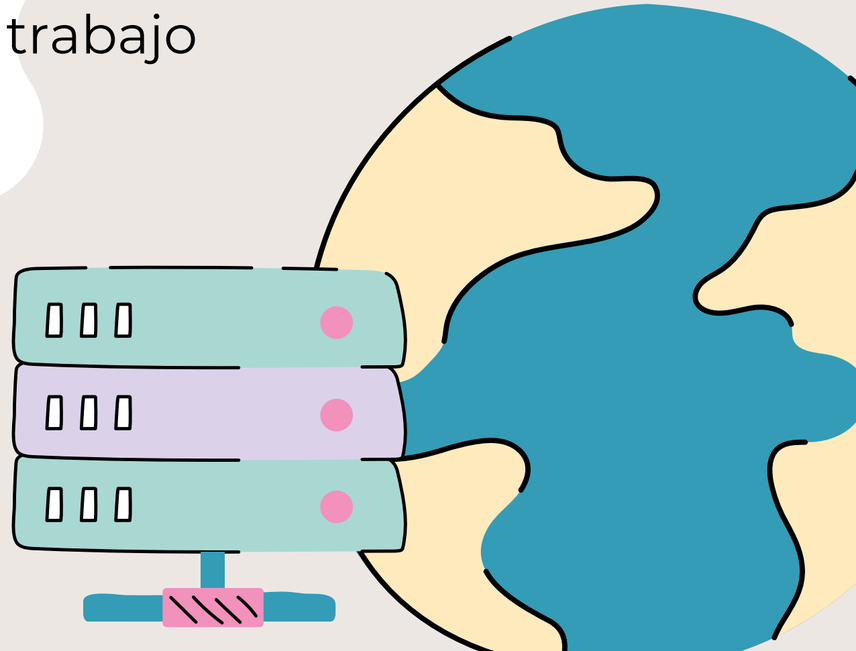




Buenas prácticas en Python

Optimiza tu código y mejora tu flujo de trabajo





Por Juan Duran

Introducción

Python es el lenguaje de referencia en el mundo del **análisis de datos**, gracias a su facilidad de uso, su versatilidad y la gran cantidad de librerías especializadas que facilitan el trabajo con datos. No obstante, escribir código sin seguir **buenas prácticas** puede generar problemas a largo plazo, como dificultades en la **interpretación**, problemas de **rendimiento** o incluso errores inesperados.

En esta presentación, veremos las mejores prácticas para trabajar con Python en análisis de datos. Aprenderemos a **estructurar** nuestro código de manera ordenada, **utilizar** herramientas adecuadas para mejorar la **eficiencia** y **documentar** correctamente cada parte del proceso. Al aplicar estas prácticas, garantizamos que nuestro código sea fácil de leer, **reutilizable** y **escalable**, lo que beneficiará tanto a nivel personal como en entornos colaborativos.

Pros

- Código más **legible** y comprensible.
- Facilita la **colaboración** en equipos de trabajo.
- Mejor **rendimiento** y eficiencia en el procesamiento de datos.
- **Reducción** de **errores** y mayor facilidad para depurar.
- Mejor **reutilización** del código.

versus

Contras

- Puede requerir más **tiempo** inicial de desarrollo.
- Requiere **aprendizaje** y adaptación.
- Algunas prácticas pueden parecer **innecesarias** en proyectos pequeños.
- Posible **sobreingeniería**.
- Puede ser difícil de aplicar en código **legado**.

Puntos clave

Estructura de código

- **Organizar** el código en funciones y módulos reutilizables.
- **Utilizar** nombres de variables y funciones descriptivos.
- **Seguir** la convención PEP8.

Eficiencia y optimización

- **Elegir** la estructura de datos adecuada.
- **Evitar** loops innecesarios y preferir operaciones vectorizadas.
- **Manejar** la memoria de manera eficiente.

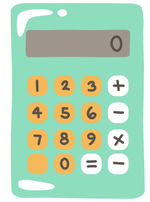
Documentación y comentarios

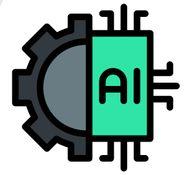
- **Escribir** docstrings en cada función.
- **Utilizar** comentarios claros en el código.
- **Crear** notebooks de Jupyter con explicaciones detalladas y ejemplos.

Librerías especializadas

Python ofrece una variedad de **librerías** especializadas que facilitan el análisis de datos. Es fundamental conocerlas y utilizarlas correctamente para mejorar la eficiencia del trabajo. Algunas de las más utilizadas incluyen:

- **Pandas**: permite trabajar con estructuras de datos flexibles y realizar manipulaciones avanzadas de datos tabulares.
- **NumPy**: optimiza los cálculos matemáticos con matrices multidimensionales y operaciones vectorizadas, reduciendo la necesidad de loops ineficientes.
- **Matplotlib** y **Seaborn**: herramientas esenciales para la visualización de datos, permitiendo crear gráficos informativos y personalizables que facilitan la interpretación de tendencias y patrones.
- **Scikit-learn**: conjunto de herramientas para machine learning, que incluye algoritmos de clasificación, regresión y clustering, fundamentales para modelos predictivos en análisis de datos.





Control de versiones con Git

El uso de **control** de **versiones** es esencial para **gestionar** proyectos de análisis de datos de manera **organizada**.

Git permite rastrear cambios, colaborar en equipo y evitar la pérdida de trabajo.

- Uso de **repositorios**: permite mantener un historial de cambios y revertir modificaciones en caso de errores.
- Creación de **ramas**: organiza el desarrollo y facilita la colaboración en equipo sin afectar la versión principal.
- **Commits** descriptivos: cada cambio debe ir acompañado de un mensaje claro que explique su finalidad.
- Uso de **GitHub**: plataforma para alojar repositorios, colaborar en proyectos y realizar revisiones de código mediante pull requests.

Manejo de datos eficiente

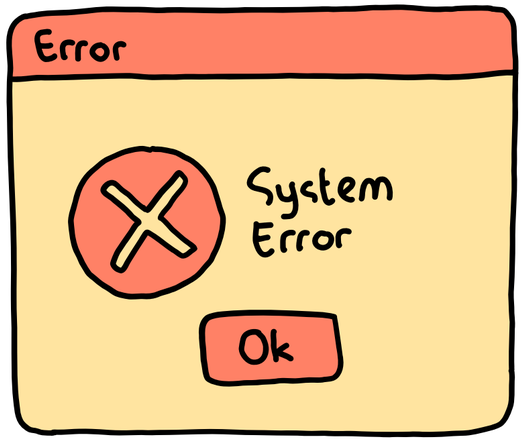
Trabajar con grandes **volúmenes** de **datos** requiere estrategias eficientes para evitar cuellos de botella en el procesamiento.

- **Cargar** solo los datos necesarios: evitar la carga de **datasets** completos si solo se requiere un subconjunto.
- **Optimizar** formatos de archivo: usar **Parquet** en lugar de CSV para almacenamiento eficiente y lectura rápida.
- **Filtrado** y agregación antes de procesar: reducir la cantidad de datos a procesar aplicando filtros y agregaciones previas.
- Uso de **estructuras** de datos eficientes: convertir columnas categóricas en **CategoricalDtype** en Pandas reduce el consumo de memoria.

Manejo de errores y excepciones

Identificar y **manejar errores** es clave para evitar fallos en los análisis de datos.

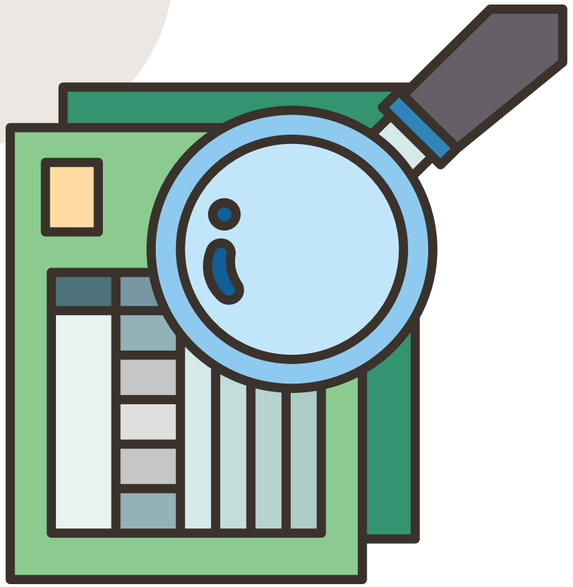
- **Uso** de **try-except**: captura y maneja errores sin interrumpir la ejecución del programa.
- **Mensajes** de error claros: proporcionar mensajes informativos que faciliten la solución de problemas.
- **Registro** de errores (logging): guardar errores en archivos de log permite su análisis y resolución posterior.
- **Validación** de datos: verificar que los datos cumplen ciertos criterios antes de procesarlos reduce la posibilidad de errores inesperados.



Testing y validación de código

El **testing** es crucial para asegurar la **calidad** del **código** en análisis de datos.

- **Pruebas** unitarias con **pytest**: verificar que cada función produce los resultados esperados.
- **Automatización** de pruebas: ejecutar pruebas automáticas reduce el riesgo de errores en nuevas implementaciones.
- **Validación** de resultados: comprobar que los análisis y modelos generan predicciones lógicas y precisas.



Optimización de código

Trabajar con **big data** requiere optimizaciones avanzadas:

- Uso de **Dask** y **Vaex**: aternativas a Pandas que permiten procesamiento distribuido y optimizado.
- **Procesamiento** en paralelo: distribuir la carga de trabajo en varios núcleos mejora el rendimiento.
- **Uso** de **generadores**: evitar cargar grandes datasets en memoria permite mayor eficiencia.
- **Indexación** y **ordenamiento** eficiente: facilita la búsqueda y acceso rápido a datos específicos.

Conclusiones

Organización del código

Un código estructurado facilita la **escalabilidad** y el **mantenimiento**.

Librerías adecuadas

Elegir la **herramienta** correcta optimiza el análisis de datos.

Optimización

Aplicar buenas prácticas reduce los tiempos de **ejecución** y el **consumo** de recursos.

Documentación clara

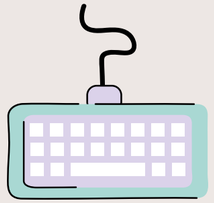
Un código bien documentado mejora la **colaboración** y **comprensión**.

Control de versiones

Usar **Git** permite un desarrollo ordenado y controlado.

Validación y testing

Las pruebas aseguran la **fiabilidad** y **calidad** de los resultados.



Gracias

Por Juan Duran

“Coding, Gaming and Leveling Up”