

Extracción de Datos desde Archivos y Fuentes Externas con Python: CSV, Excel y Web

La obtención y extracción de datos es el primer eslabón de toda cadena de valor en ingeniería y ciencia de datos. Sin datos limpios, confiables y estructurados, ningún análisis o modelo posterior será robusto, sin importar la tecnología empleada. La habilidad de extraer datos correctamente desde archivos planos (CSV), hojas de cálculo (Excel) o páginas web (scraping de tablas HTML) es fundamental para cualquier profesional que busque tomar decisiones basadas en información. Además, la extracción de datos es transversal a áreas como business intelligence, automatización de procesos, análisis científico, economía, marketing, salud, entre otros, y es el punto de partida para tareas más complejas como integración de bases de datos, ETL, machine learning y desarrollo de dashboards.

 **por Kibernetum Capacitación S.A.**



Preguntas de activación



Retos en importación de datos

¿Cuál ha sido el mayor reto que has enfrentado al intentar importar datos en tus propios proyectos?



Importancia de la extracción de datos

¿Por qué crees que la extracción de datos se considera una de las fases más críticas dentro de un pipeline de datos?



Problemas técnicos y éticos

¿Qué problemas técnicos o éticos imaginas al obtener datos desde páginas web abiertas?

Conceptos fundamentales de extracción de datos

¿Qué significa extraer datos?

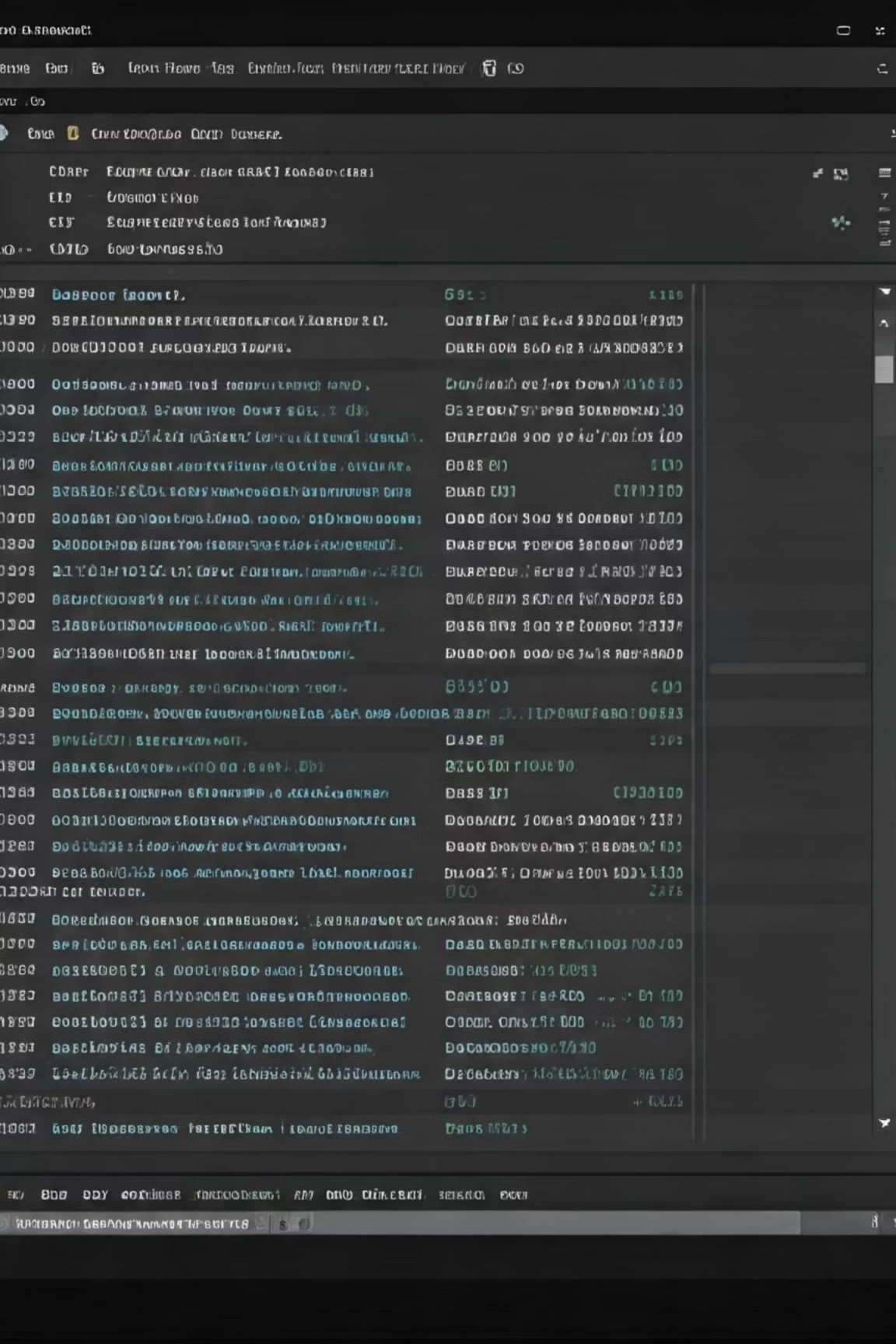
Extraer datos es el proceso de obtener información relevante desde fuentes externas para su posterior análisis o almacenamiento. Estas fuentes pueden ser archivos locales (planos o estructurados), bases de datos, APIs, documentos, sitios web, sensores IoT, entre otros.

En ingeniería de datos, la extracción suele ser el "puente" entre el mundo real y el análisis digital. Una extracción efectiva significa obtener datos íntegros, precisos y listos para ser transformados y explotados.

¿Por qué es crítica la extracción?

- El 70-80% del esfuerzo de un proyecto de datos real se dedica a la obtención y preparación de los datos.
- Los errores y malas prácticas en esta etapa suelen propagarse y magnificarse en las etapas siguientes.
- Las fuentes de datos son muy variadas: archivos CSV generados por sistemas antiguos, reportes Excel de departamentos, y datos en tiempo real publicados en web o APIs.

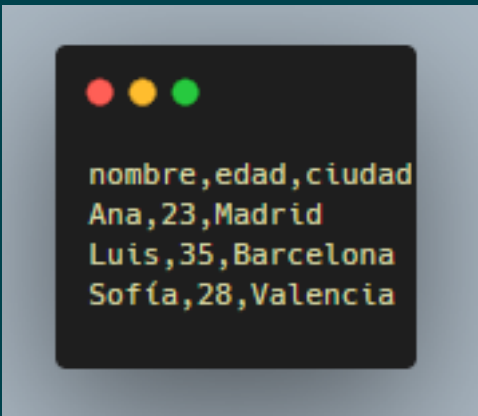
Conexión interdisciplinar: Quien domina la extracción de datos puede colaborar fluidamente con equipos de ingeniería, analistas, científicos de datos, áreas comerciales y TI.



Archivos CSV: Definición y características

Un archivo CSV (Comma Separated Values) es un formato universal de archivo plano, donde cada línea representa un registro y cada campo está separado por un delimitador (coma por defecto, pero pueden ser punto y coma, tabulador, etc.).

Ejemplo de CSV



Características técnicas

- Es texto puro, por lo que puede ser abierto y editado con cualquier editor.
- No almacena metadatos (no hay formatos, colores ni fórmulas).
- Permite el intercambio entre distintos sistemas, incluso con lenguajes de programación o software antiguo.

Ventajas y limitaciones de los archivos CSV

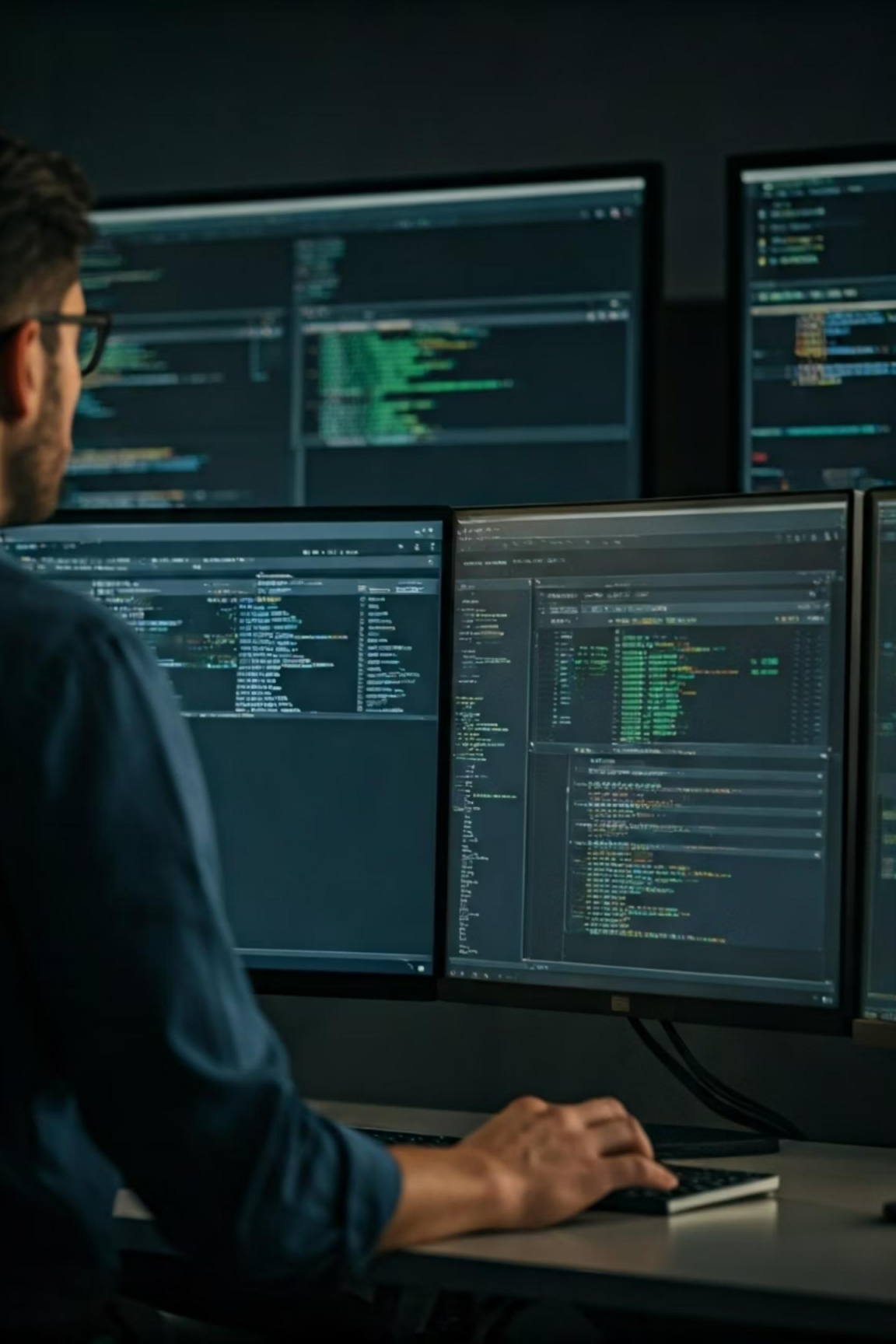
Ventajas de los CSV

- Portabilidad: Funciona igual en cualquier sistema operativo.
- Simplicidad: No requiere librerías complejas para ser leído o escrito.
- Rapidez: Ideal para transmitir datos tabulares rápidamente entre aplicaciones o servicios.
- Estandarización: Muy usado en ciencia de datos, comercio electrónico, banca, gobierno, etc.

Limitaciones de los CSV

- No soporta estructuras complejas (como tablas anidadas o jerárquicas).
- No almacena fórmulas, formatos visuales, ni tipos de datos explícitos.
- Puede generar errores si los campos contienen el mismo delimitador usado (por ejemplo, comas dentro de textos).
- Puede tener problemas de codificación y de compatibilidad regional (puntos decimales vs. comas, por ejemplo).

Analogía didáctica: Piensa en un CSV como una tabla impresa en papel: simple, entendible, fácil de pasar a otros, pero sin colores ni fórmulas ocultas.



Uso típico de CSV en ingeniería de datos



Exportación de registros

Exportar registros desde una base de datos o aplicación para su posterior procesamiento o análisis.



Intercambio entre sistemas

Compartir datos entre sistemas que no comparten un formato nativo, facilitando la interoperabilidad.

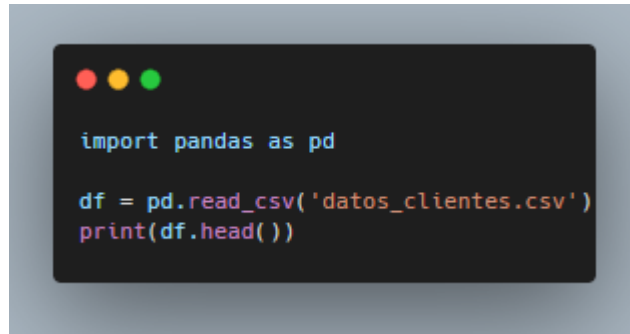


Logs automáticos

Registros automáticos generados por sensores o plataformas web que necesitan ser almacenados de forma simple.

Lectura básica de CSV con Pandas

`pandas.read_csv()` es la función estándar para leer archivos CSV en Python, creando un `DataFrame` que puede manipularse fácilmente.



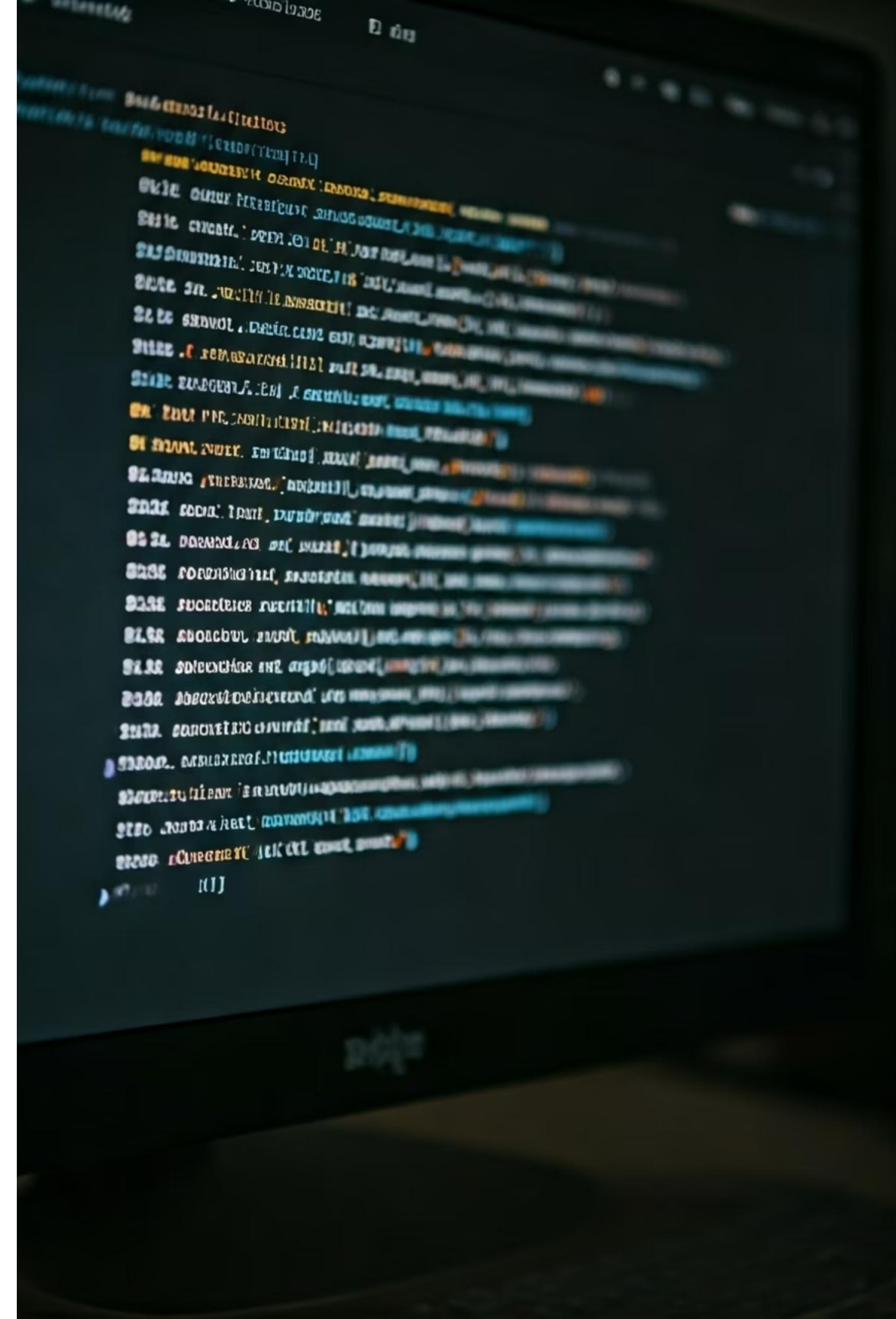
```
import pandas as pd

df = pd.read_csv('datos_clientes.csv')
print(df.head())
```

DataFrame: Es la estructura de datos de Pandas, similar a una hoja Excel, que permite manipulación tabular potente.

Parámetros clave de `pandas.read_csv()`:


- `sep`: Define el delimitador (por defecto `,`). Ejemplo: `sep=';'` para CSV europeos.
- `header`: Indica la fila que contiene los nombres de las columnas.
- `index_col`: Define una columna como índice del `DataFrame`.
- `na_values`: Especifica qué valores deben considerarse nulos o faltantes.
- `encoding`: Controla la codificación, vital para archivos con caracteres especiales.



Lectura avanzada de CSV y casos reales

Lectura con parámetros específicos

Supón que tienes un CSV generado en un sistema europeo, con punto y coma como delimitador y acentos:



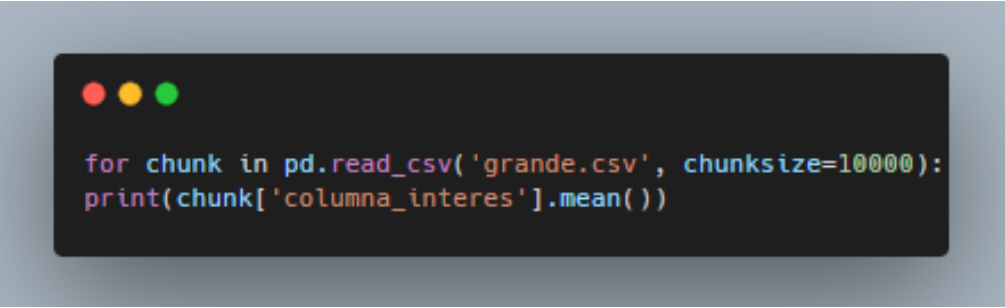
```
df = pd.read_csv(
    'clientes.csv',
    sep=';',
    encoding='latin1',
    na_values=['NA', ''],
    index_col='ID'
)
```

Aquí se adapta la lectura al contexto regional y a los posibles nulos.

Lectura de archivos grandes por partes

Problema real: A veces los archivos son tan grandes que no caben en memoria RAM.

Solución: usar chunksize para procesar en bloques (por ejemplo, 10,000 filas a la vez):



```
for chunk in pd.read_csv('grande.csv', chunksize=10000):
    print(chunk['columna_interes'].mean())
```

Esto es fundamental para "big data" o pipelines automatizados.


```
def main():
    # Cargar datos desde el archivo CSV original
    df = pd.read_csv('clientes.csv')

    # Filtrado de datos
    filtro = (df['ciudad'] == 'Madrid') & (df['ingresos'] > 3000)

    # Creación del nuevo DataFrame
    df_filtrado = df[filtro]

    # Exportación del resultado
    df_filtrado.to_csv('clientes_madrid_alto.csv', index=False)
```

Escritura y exportación de CSV con Pandas

```
df.to_csv('salida.csv', index=False, sep=',', encoding='utf-8')
```

- index=False evita agregar una columna adicional de índices.
- Puede exportarse solo parte de las columnas, filtrando previamente.

Ejemplo integrado: Extracción

Cargar los datos desde el archivo CSV original

```
df = pd.read_csv('clientes.csv')
```

Filtrado de datos

Aplicar condiciones para seleccionar solo los registros deseados

```
filtro = (df['ciudad'] == 'Madrid') & (df['ingresos'] > 3000)
```

Creación del nuevo DataFrame

Crear un nuevo DataFrame con los datos filtrados

```
df_filtrado = df[filtro]
```

Exportación del resultado

Guardar los datos filtrados en un nuevo archivo CSV

```
df_filtrado.to_csv('clientes_madrid_alto.csv', index=False)
```



Archivos Excel: Fundamentos y características

Un archivo Excel (extensiones .xls, .xlsx) es un archivo binario estructurado que permite guardar datos en múltiples hojas, incluir fórmulas, formatos, gráficos y metadatos.

Características técnicas:

- Puede tener varias hojas dentro de un mismo archivo.
- Soporta tipos de datos mixtos, celdas fusionadas, fórmulas, formatos condicionales, comentarios y validación de datos.
- Es el estándar de facto en empresas y ciencia para reportes y almacenamiento intermedio.

Analogía didáctica: Excel es como una libreta digital con hojas múltiples, gráficos y notas, ideal para análisis colaborativo pero menos eficiente para integraciones automáticas.

Ventajas y limitaciones de los archivos Excel

Ventajas:

- Riqueza estructural: Permite formatos avanzados, gráficos, y fórmulas embebidas.
- Multipropósito: Útil para compartir datos y análisis junto con los resultados en un solo archivo.
- Interoperabilidad: Muchas plataformas y sistemas exportan en formato Excel.

Limitaciones:

- Más pesado y menos eficiente que CSV para grandes volúmenes.
- Requiere librerías adicionales (openpyxl, xlrd) y cuidado con versiones.
- Su formato puede cambiar según la versión de Excel utilizada (antiguo .xls vs. moderno .xlsx).
- Puede contener celdas mezcladas, vacías o datos poco estructurados, lo que complica el procesamiento automatizado.

Manejo de Excel en Python con Pandas



Lectura básica

```
df = pd.read_excel('clientes.xlsx', sheet_name='Hoja1')
print(df.head())
```

Puedes seleccionar la hoja por nombre o número.



Lectura avanzada

```
datos = pd.read_excel('clientes.xlsx', sheet_name=None)
```

Leer todas las hojas a la vez como un diccionario de DataFrames.



Escritura y exportación

```
df.to_excel('resultado.xlsx', index=False, sheet_name='DatosLimpios')
```

Se pueden guardar varias hojas en un solo archivo usando ExcelWriter.

Introducción a los datos con Python

Series DataFrames Indexing Pandas

```
def main():
    # Leer datos desde un archivo Excel
    df = pd.read_excel('datos.xlsx', sheet_name='Hoja1')

    # Filtrar datos por una columna
    df_filtrado = df[df['columna'] > 10]

    # Agrupar datos por una columna
    df_agrupado = df.groupby('columna').sum()

    # Exportar datos a un archivo Excel
    df.to_excel('resultado.xlsx', sheet_name='DatosLimpios', index=False)
```

Python | Series | DataFrames | Indexing | Pandas | Data Science

3	3	columna1	columna2	columna3	columna4
4	3	columna1	columna2	columna3	columna4
5	3	columna1	columna2	columna3	columna4
6	3	columna1	columna2	columna3	columna4
7	2	columna1	columna2	columna3	columna4
8	3	columna1	columna2	columna3	columna4
9	0				

Compatibilidad con archivos Excel antiguos

Para trabajar con archivos Excel antiguos (.xls) es necesario usar la librería xlrd:

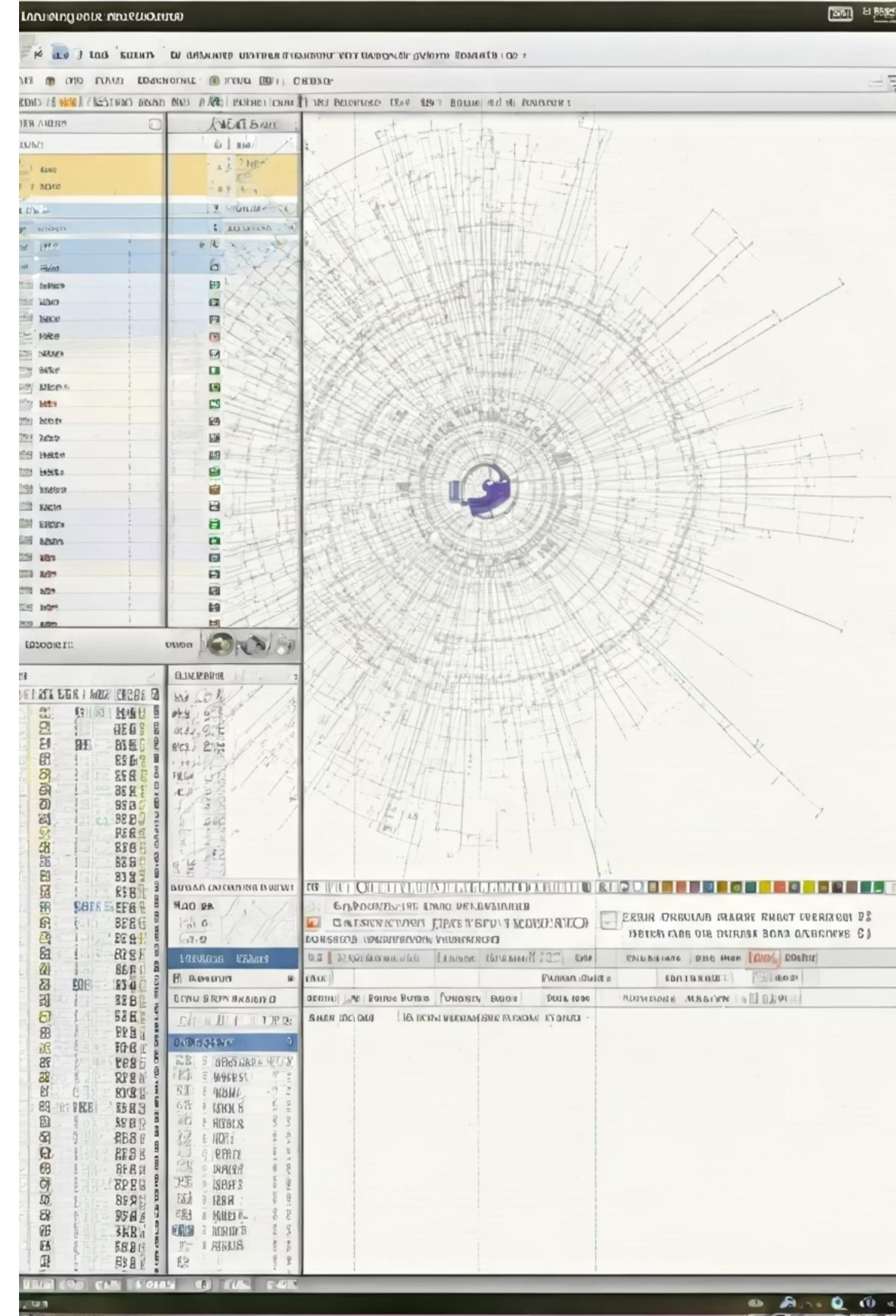
```
import xlrd

book = xlrd.open_workbook('archivo.xls')
sheet = book.sheet_by_name('Hoja1')

for row_idx in range(sheet.nrows):
    print(sheet.row_values(row_idx))
```

Errores comunes:

- Leer archivos con hojas ocultas o protegidas.
- Celdas con formatos especiales no reconocidos.
- Diferencias en la codificación de caracteres (Excel puede introducir caracteres especiales invisibles).
- Falta de actualización de librerías de Python (openpyxl, xlrd).



Extracción de tablas desde la web: Web scraping

¿Por qué extraer datos de la web?

Hoy día, muchos datos relevantes (indicadores económicos, rankings, precios, estadísticas) se publican en portales abiertos y páginas web, en tablas HTML.

¿Qué es el web scraping?

Es la técnica de extraer información automatizadamente desde páginas web, generalmente parseando (interpretando) el HTML y extrayendo solo las partes útiles (ejemplo: tablas).

Ventajas del web scraping:

- Acceso a información pública y actualizada en tiempo real.
- Permite automatizar la recolección de datos, evitando la tarea manual y repetitiva.

Limitaciones y retos:

- Las páginas pueden cambiar de estructura (rompiendo el código de extracción).
- Puede haber restricciones legales o técnicas (robots.txt, CAPTCHAs).
- No siempre se garantiza la calidad o veracidad de los datos extraídos.

Extracción automática de tablas HTML con Pandas

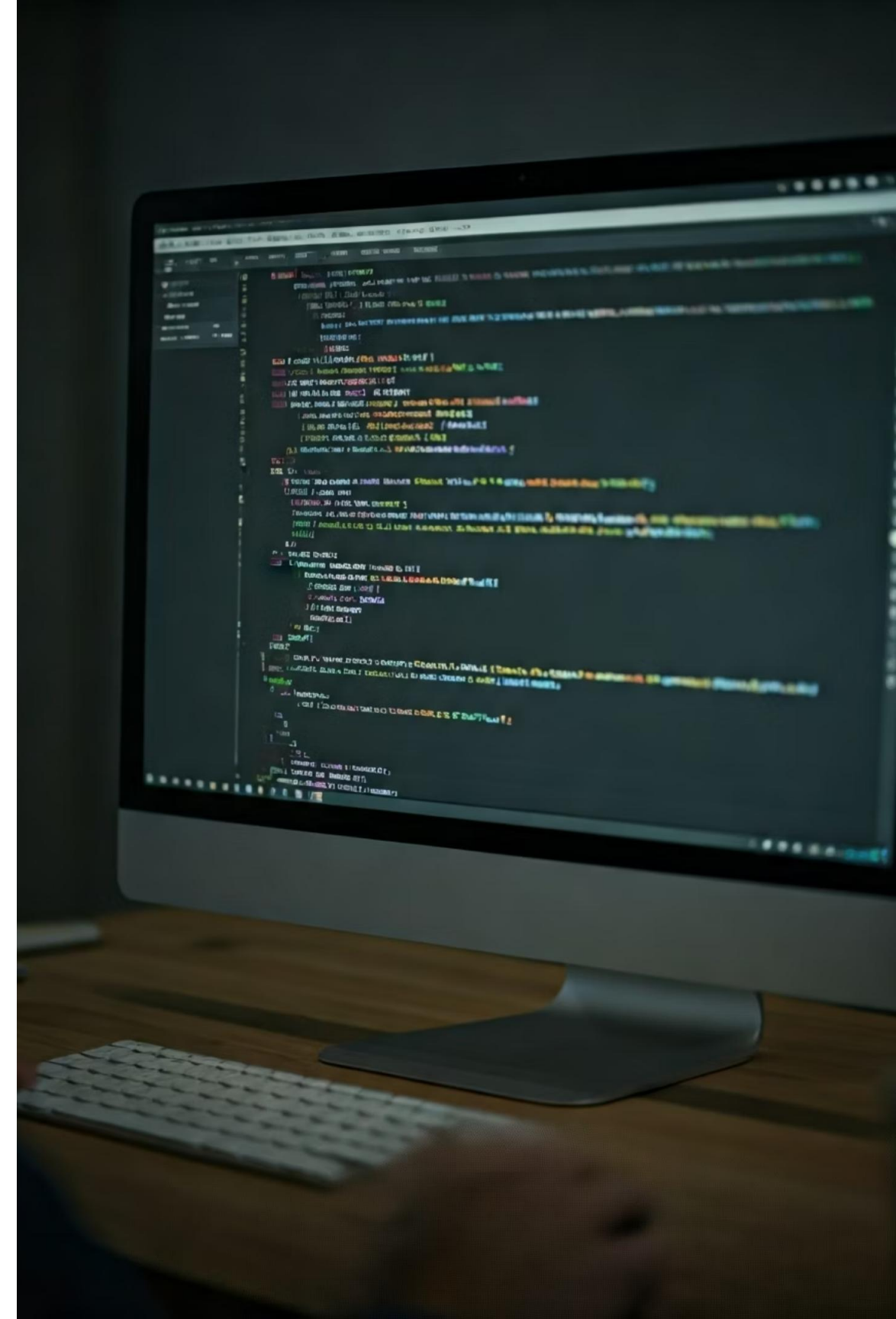
```
url = 'https://es.wikipedia.org/wiki/Anexo:Países_por_PIB'
tablas = pd.read_html(url)

print(f"Se encontraron {len(tablas)} tablas.")
print(tablas[0].head())
```

- read_html() busca automáticamente tablas <table> y las convierte en DataFrames.
- Ideal para indicadores, rankings, datos de portales abiertos, etc.

Consideraciones éticas y técnicas:

- Legalidad: Revisa siempre los términos de uso y respeta las normas del sitio (robots.txt).
- Frecuencia: No realices demasiadas solicitudes en poco tiempo.
- Validación: Chequea periódicamente que la estructura HTML no haya cambiado.



Errores frecuentes en web scraping

Tablas no encontradas

Tablas no encontradas porque la web usa JavaScript para generarlas dinámicamente (requiere técnicas avanzadas).

Datos mal interpretados

Datos mal interpretados por Pandas por encabezados confusos o estructuras complejas en las tablas HTML.

Extracción de tabla incorrecta

Extracción de la tabla incorrecta (algunas páginas contienen muchas tablas, algunas de ejemplo, otras con notas).



Comparativa de formatos y aplicaciones prácticas

Formato/Fuente	Ventajas	Limitaciones	Ejemplo de aplicación
CSV	Simple, universal, eficiente	No soporta fórmulas, sin metadatos complejos	Logs, exportaciones automáticas
Excel	Multitabla, soporta fórmulas, metadatos, formatos	Mayor peso, complejidad, dependencias de librerías	Informes empresariales, datos con formatos especiales
Web (HTML)	Acceso a información pública, datos dinámicos	Requiere validación y parsing, cambios de estructura, cuestiones legales	Rankings, scraping de portales abiertos



Desafíos frecuentes en extracción de datos



Encoding y formatos regionales

Archivos exportados desde distintos sistemas pueden tener codificaciones diferentes (utf-8, latin1) o delimitadores no estándar.



Datos faltantes y tipos incorrectos

Común encontrar campos nulos o con mezclas inesperadas de texto y números que requieren limpieza y transformación.



Archivos corruptos o con hojas múltiples

Pandas permite manejar estas situaciones con parámetros específicos, pero requiere validaciones y pruebas.

Mejores prácticas en extracción de datos



Automatización y reproducibilidad

Integra scripts en pipelines programados/notebooks para asegurar que los procesos sean repetibles y trazables.



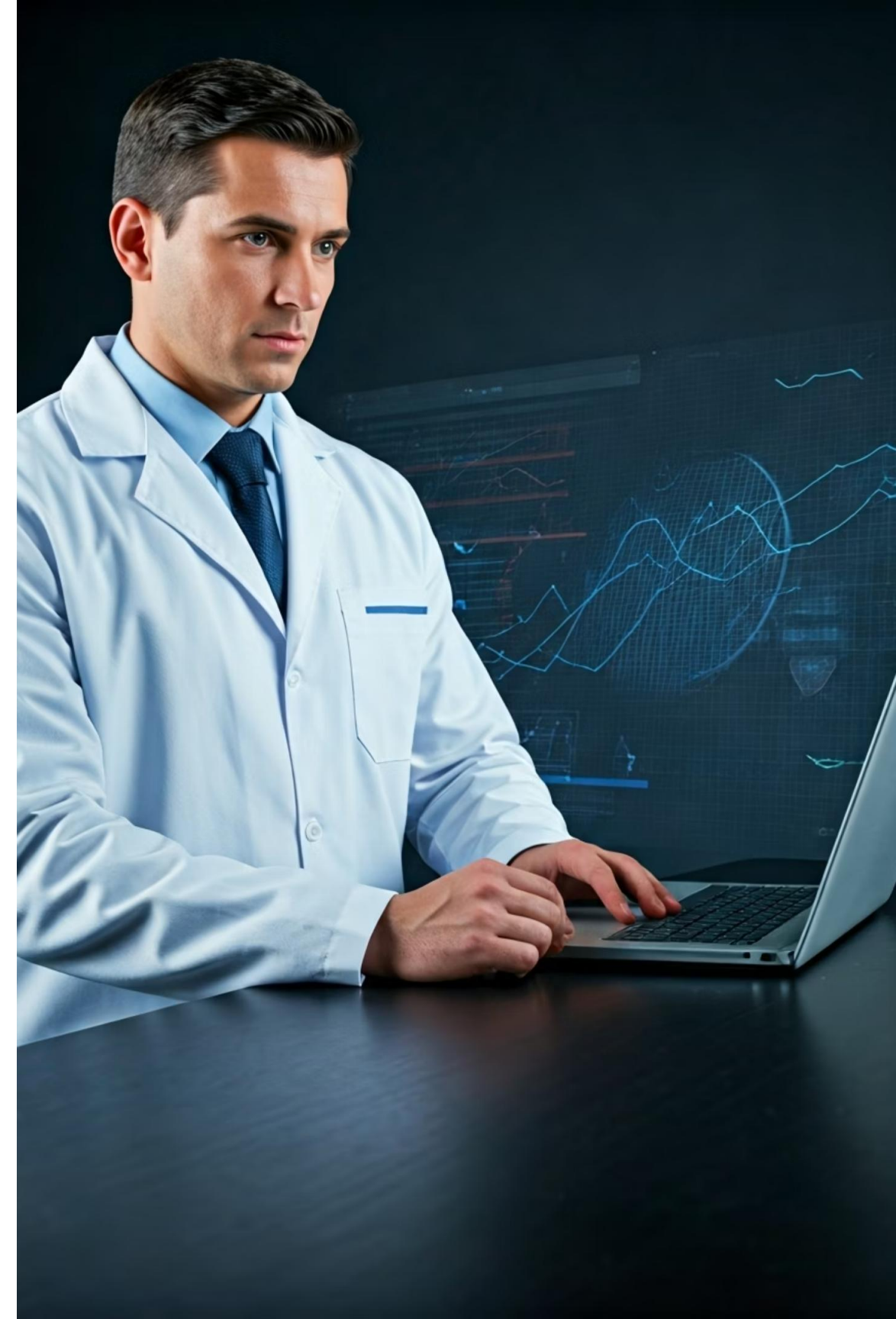
Prueba y validación continua

Siempre valida el DataFrame resultante (usando `info()`, `describe()`, inspección visual) tras cada extracción.



Documentación del proceso

Documenta cada paso del proceso de extracción, incluyendo las decisiones tomadas y los problemas encontrados.



Relación con otras áreas y aplicación interdisciplinar



Recursos recomendados



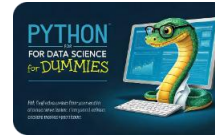
Python for Data Analysis

Wes McKinney,
O'Reilly Media.
Libro fundamental
para entender
Pandas y la
manipulación de
datos en Python.



Documentación oficial de Pandas

<https://pandas.pydata.org/docs/> -
Referencia
completa de todas
las funcionalidades
de la librería.



Python for Data Science for Dummies

Wiley. Guía
accesible para
principiantes en
ciencia de datos
con Python.



BeautifulSoup y lxml

<https://www.crummy.com/software/BeautifulSoup/bs4/doc/> - Herramientas
avanzadas para
web scraping.

Actividades prácticas

Las actividades prácticas permiten aplicar los conceptos aprendidos en situaciones reales de extracción de datos. Estas pueden adaptarse o ampliarse para profundizar en temáticas particulares como:

1 Manejo de errores de encoding

Trabajar con archivos CSV que contienen caracteres especiales o diferentes codificaciones regionales.

2 Lectura avanzada de Excel

Extraer datos de archivos Excel complejos con múltiples hojas, fórmulas y formatos especiales.

3 Scraping robusto

Desarrollar técnicas para extraer datos de páginas web que cambian su estructura o utilizan JavaScript.

4 Integración de múltiples fuentes

Combinar datos provenientes de diferentes formatos (CSV, Excel, web) en un único DataFrame para análisis.





Preguntas abiertas de reflexión



Problemas con datos reales

¿Qué problemas encontraste al trabajar con datos reales?



Ventajas de la automatización

¿Qué ventajas ofrece automatizar la extracción y limpieza de datos?



Validación de resultados

¿Cómo validarías que tus resultados son correctos y reproducibles?

Resumen – 5 puntos clave



Herramientas fundamentales

Python y Pandas permiten extraer datos de CSV, Excel y la web con eficiencia.



Selección de formato

Cada formato tiene ventajas y limitaciones; elegir según el caso de uso.



Manejo de errores

El manejo de errores, encoding y validación es crucial para datos confiables.



Automatización

Automatizar la extracción agiliza proyectos y mejora reproducibilidad.



Habilidad esencial

Dominar estas técnicas es esencial para cualquier ingeniero de datos o científico de datos.