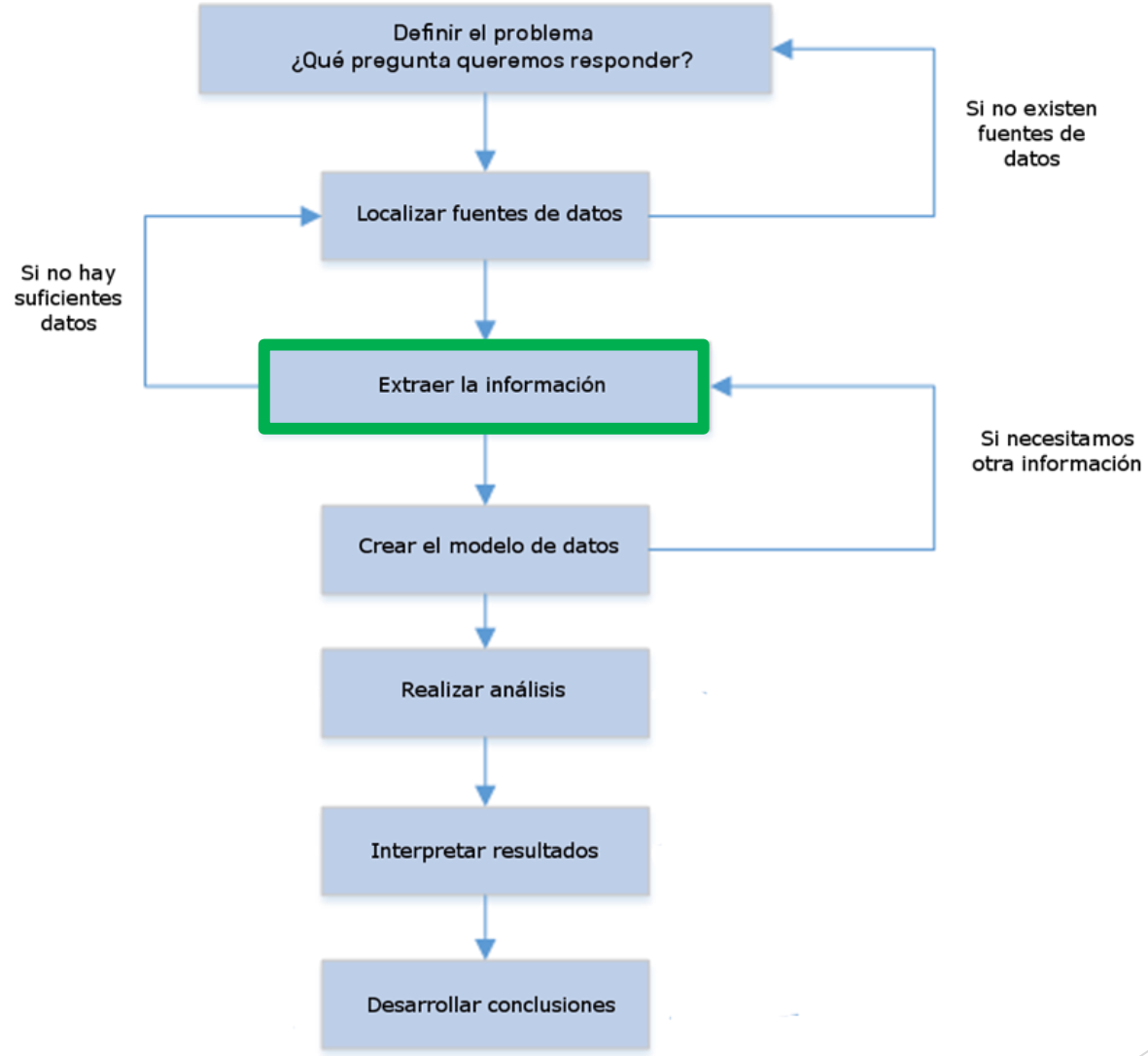
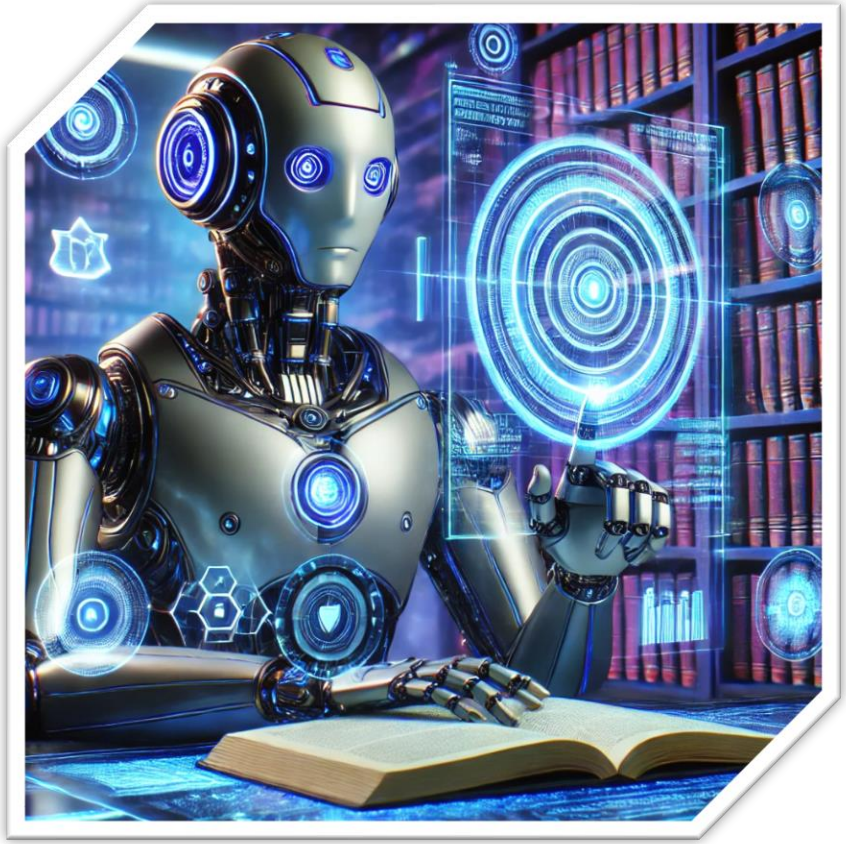
Extracción de datos

Dr. Francisco E. Cabrera

El proceso de un análisis de datos



El acceso a los datos



- Para poder extraer datos, lo primero es tener acceso a ellos.
- Existen distintos métodos para obtener los datos.

El acceso a los datos

Distintas formas de acceder a los datos

- Archivos.
- Bases de datos SQL.
- Bases de datos NoSQL.
- APIs.
- Fuentes RSS.
- OCR.
- Logs y otros archivos de registro.
- Web Scraping.
- Protocolos específicos (IoT).
- Streaming de datos.

Archivos descargados

- La descarga de archivos consiste en obtener datos que ya están almacenados en un archivo en formato CSV, JSON o XML.
- Estos son formatos comunes para el intercambio de datos, y muchas aplicaciones y bases de datos pueden exportar datos en estos formatos..

- Utilidades y aplicaciones
  - **Extracción de datos tabulares**: Los archivos CSV son ampliamente utilizados para almacenar datos estructurados.
  - **Intercambio de datos entre sistemas**: Los formatos **JSON** y **XML** son populares para APIs, servicios web y exportaciones de muchas aplicaciones.
  - **Migración de datos**: Facilita la exportación e importación de datos entre diferentes plataformas o aplicaciones.

Archivos descargados

- **Dificultades**
  - **Manejo de grandes volúmenes**: Los archivos grandes pueden consumir mucha memoria, especialmente si no están correctamente optimizados.
  - **Inconsistencias en los datos**: Los archivos CSV o JSON pueden contener datos mal formateados o erróneos que pueden causar problemas al momento de cargarlos.
  - **Compatibilidad de formatos**: Es posible que el formato de archivo no sea compatible o que sea necesario convertirlo a otro formato.

Archivos descargados

¿Cuándo usarlo?

- **Fuentes de datos que ya están almacenadas** en archivos como CSV, JSON o XML.
- **Cuando no se necesita procesamiento en tiempo real**, sino solo la extracción de datos almacenados previamente.

¿Cuándo no usarlo?

- **Cuando los datos cambian frecuentemente** o requieren actualización en tiempo real.

Archivos descargados

Lectura de un fichero de texto genérico

- Método nativo de Python open()
  - https://docs.python.org/3/library/functions.html#open

Archivos descargados

Lectura de un fichero de texto genérico

- Método nativo de Python open()
  - https://docs.python.org/3/library/functions.html#open

```
with open(“archivo.txt”, “r”, encoding=“utf-8”) as f:
    contenido = f.read()
```

Archivos descargados

- Librería nativa csv
  - https://docs.python.org/3/library/csv.html

Archivos descargados

- Librería nativa csv
  - https://docs.python.org/3/library/csv.html

```
import csv

with open("datos.csv", newline='', encoding="utf-8") as csvfile:
    reader = csv.reader(csvfile)
    for row in reader:
        print(row)
```

Archivos descargados

Ejemplo con Pandas

- Mostrar la primera línea de un archivo CSV con Pandas.

Archivos descargados

Ejemplo con Pandas

- Mostrar la primera línea de un archivo CSV con Pandas.

```
import pandas as pd

# Cargar en un DataFrame el archivo
df = pd.read_csv("alumnos.csv")

# Mostrar las primeras filas
print(df.head())
```

Archivos JSON

- Librería nativa JSON
  - https://docs.python.org/3/library/json.html

Archivos JSON

- Librería nativa JSON
  - https://docs.python.org/3/library/json.html

```
import json

with open("datos.json", "r", encoding="utf-8") as f:
    data = json.load(f)

#Si ya teníamos temenos el JSON como string
jsonstr = "{\"key1\":1, \"key2\": \"adios\"}"
data = json.loads(jsonstr)
```

Archivos XML

- Librería xml
  - xml.etree.ElementTree
  - https://docs.python.org/3/library/xml.etree.elementtree.html

Archivos XML

- Librería xml
  - xml.etree.ElementTree
  - https://docs.python.org/3/library/xml.etree.elementtree.html

```
from xml.etree import ElementTree

tree = ElementTree.parse("datos.xml")
root = tree.getroot()
print(root.tag) # Mostrar nodo raiz

for child in root:
    # Mostrar cada uno de los nodos hijos.
    print(f“{child.tag} -> {child.text}”)
```

Archivos serializados “pickle”

- Pickle implementa un protocolo binario para serializar y deserializar estructuras de Python.
  - https://docs.python.org/3/library/pickle.html

Archivos serializados “pickle”

- Pickle implementa un protocolo binario para serializar y deserializar estructuras de Python.
  - https://docs.python.org/3/library/pickle.html

```
import pickle

# Definir e instanciar la clase un libro
class Libro:
    def __init__(self, titulo, autor):
        self.titulo = titulo
        self.autor = autor

mi_libro = Libro(“Un estudio en escarlata”,
“Arthur Conan Doyle”)
```

```
# Guardar un archivo binario
with open(“libro.pkl”, “wb”) as f:
    pickle.dump(mi_libro, f)


# Recuperar un archivo binario
with open(“libro.pkl”, “rb”) as f:
    libro_recuperado = pickle.load(f)
```

Imágenes y Audio

Imagen

- Librería Pillow
  - https://pillow.readthedocs.io/en/stable/

- Librería OpenCV
  - https://docs.opencv.org/4.x/d6/d00/tutorial_py_root.html

Audio

- Librería librosa
  - https://librosa.org/doc/latest/index.html

Fuentes RSS

- Las **fuentes RSS** son una manera sencilla y estructurada de distribuir contenido web, como noticias, blogs, o artículos.
- RSS permite que los usuarios reciban actualizaciones automáticas cuando se publica nuevo contenido en sitios web que ofrecen estas fuentes.
- La ventaja es que podemos acceder a contenido sin tener que navegar por el sitio web directamente.
- Utilidades y aplicaciones
  - **Seguimiento de noticias**: Puedes obtener actualizaciones automáticas de sitios como blogs, periódicos, o sitios de tecnología.
  - **Agregadores de contenido**: Muchos servicios de noticias o aplicaciones de agregación utilizan RSS para distribuir contenido de manera eficiente.
  - **Monitoreo de actualizaciones en tiempo real**: RSS es ideal para mantenerse al tanto de eventos recurrentes, como la publicación de nuevos artículos.

Fuentes RSS

- **Dificultades**
  - **Limitación de información**: La cantidad de información en un feed RSS es limitada, generalmente solo los títulos y resúmenes del contenido.
  - **No siempre está disponible**: No todos los sitios web ofrecen RSS, y algunos no lo actualizan con regularidad.
  - **Formatos y estándares inconsistentes**: Aunque RSS tiene un formato estándar, algunos sitios pueden no seguirlo estrictamente.

Fuentes RSS

¿Cuándo usarlo?

- **Para obtener actualizaciones automáticas de sitios web** que ofrecen feeds RSS (noticias, blogs, etc.).
- **Para agregar contenido en tiempo real** a tu aplicación o sistema sin tener que acceder directamente a un sitio web.

¿Cuándo no usarlo?

- **No usarlo si el sitio no tiene una fuente RSS** o si necesitas datos detallados que no están disponibles en el feed.

Fuentes RSS

- Suelen ser habituales en noticias, foros y blogs.
- Es fácil leer el XML, pero algunas fuentes cambian y desaparecen.
- Hay que monitorizar la disponibilidad de los feeds.

```
import feedparser

rss_url = "http://feeds.bbci.co.uk/news/rss.xml"
feed = feedparser.parse(rss_url)

# Mostrar 5 articulos de la BBC
for entry in feed.entries[:5]:
    print(entry.title)
```

Web Scraping

- **Dificultades**
  - **Legalidad y Términos de Servicio**: Muchos sitios prohíben explícitamente el scraping en sus **Términos y Condiciones**.
  - **Anti-bots**: Algunas páginas implementan medidas para bloquear bots (uso de captchas, headers HTTP especiales, etc.).
  - **Estructura cambiante de sitios**: Si el diseño del sitio web cambia, el scraper podría dejar de funcionar.
  - **Carga en los servidores**: Si no se configura correctamente, el scraping puede generar una carga excesiva en los servidores del sitio, lo que puede ser considerado un abuso.

Web Scraping

¿Cuándo usarlo?

- Recolección de datos públicos de sitios sin restricciones explícitas.
- Datos no disponibles a través de APIs

¿Cuándo no usarlo?

- Sitios con términos de servicio que lo prohíban
- Cuando se pueda usar una API oficial.

Librería requests y beautifulsoup4

- Requests
  - https://docs.python-requests.org/en/v2.0.0/
- Beautifulsoup
  - https://tedboy.github.io/bs4_doc/

Web Scraping

- La dificultad del scraping dependerá de diseño del sitio web.
- Ejemplo con el sitio de noticias Hacker News

```
import requests
from bs4 import BeautifulSoup

# Sitio de noticias Hackernews
url = "https://news.ycombinator.com/"
response = requests.get(url)
soup = BeautifulSoup(response.text, "html.parser")

titles = soup.find_all("a", class_="titlelink")
for title in titles[:5]:
    print(title.text)
```

Bases de Datos SQL

- Las **bases de datos SQL** como **MySQL** y **PostgreSQL** son sistemas de gestión de bases de datos relacionales (RDBMS) que almacenan datos en tablas interrelacionadas.
- Usan el lenguaje **SQL** (Structured Query Language) para realizar consultas y obtener datos.
- Utilidades y aplicaciones
  - **Almacenamiento estructurado** de datos en tablas.
  - **Consultas complejas**: Capacidad de realizar búsquedas, filtros y operaciones de agregación utilizando SQL.
  - **Integridad referencial**: Asegura que los datos se mantengan consistentes y organizados con relaciones entre tablas.

Bases de Datos SQL

- **Dificultades**
  - **Escalabilidad limitada**: A medida que los volúmenes de datos crecen, las bases de datos SQL pueden volverse más lentas, aunque existen soluciones como **particionamiento** y **replicación**.
  - **Complejidad en esquemas grandes**: Manejar bases de datos con estructuras complejas o relaciones entre muchas tablas puede ser difícil.
  - **Rendimiento en consultas muy grandes**: Para consultas complejas o con grandes volúmenes de datos, las bases de datos SQL pueden ser más lentas que otras soluciones NoSQL.

Bases de Datos SQL

¿Cuándo usarlo?

- **Cuando se necesita estructura y relaciones claras** entre los datos.
- **Cuando se requiere realizar consultas complejas o transacciones** (por ejemplo, en sistemas bancarios o CRM).

¿Cuándo no usarlo?

- **No usarlo cuando los datos son semi-estructurados o no estructurados** (como documentos o registros de logs).

Bases de Datos SQL

- MySQL, PostgreSQL, etc.
- Se puede utilizar el lenguaje SQL, que es un estándar desde 1986.

```
import sqlite3

conn = sqlite3.connect("midatabase.db")

cursor = conn.cursor()
cursor.execute("SELECT * FROM users")
data = cursor.fetchall()

print(data)
```

Bases de Datos NoSQL

- Las **bases de datos NoSQL** son bases de datos no relacionales que almacenan datos en una forma más flexible que las bases de datos SQL.
- La forma de acceder a los datos es distinta para diferentes soluciones.
- Ejemplos populares son **MongoDB** (documentos JSON) y **Firebase** (clave-valor).
- Utilidades y aplicaciones
  - **Flexibilidad** en los datos: No es necesario tener una estructura fija como en las bases de datos SQL.
  - **Escalabilidad horizontal**: Las bases de datos NoSQL son muy buenas para manejar grandes volúmenes de datos distribuidos.
  - **Eficiencia en datos semi-estructurados**: Son muy eficaces para manejar datos como JSON, logs, y otros formatos que no se ajustan bien a un modelo relacional.

Bases de Datos NoSQL

- **Dificultades**
  - **Falta de soporte para transacciones complejas**: Algunas bases de datos NoSQL no tienen soporte completo para transacciones, lo que puede ser un problema en sistemas críticos.
  - **Consulta limitada**: Aunque algunas bases de datos NoSQL soportan consultas complejas, generalmente no son tan potentes como las bases de datos SQL para estos casos.
  - **Consistencia eventual**: En algunos sistemas distribuidos de NoSQL, los datos pueden ser inconsistentes temporalmente, lo que puede ser problemático en ciertos escenarios.

Bases de Datos NoSQL

¿Cuándo usarlo?

- **Cuando necesitas escalar horizontalmente** y manejar grandes volúmenes de datos distribuidos.
- **Cuando los datos no son estrictamente relacionales** y pueden tener diferentes esquemas (por ejemplo, registros de usuarios, logs).

¿Cuándo no usarlo?

- **No usarlo para datos que requieran transacciones complejas** o en aplicaciones que necesiten garantizar una fuerte consistencia de los datos.

Bases de Datos NoSQL

- MongoDB, Firebase, etc.
- La forma de trabajar con estas bases de datos dependerá de cada solución particular.

```
from pymongo import MongoClient

client = MongoClient("mongodb://localhost:27017/")

db = client["mi_base"]
collection = db["usuarios"]
data = collection.find_one()

print(data)
```

API REST

- Las **APIs (Interfaz de Programación de Aplicaciones)** permiten que diferentes sistemas se comuniquen entre sí.
- Las más comunes son **REST** (basadas en HTTP y recursos) y **GraphQL** (que permite consultas más flexibles). También existen APIs de **streaming** que permiten recibir datos en tiempo real.
- Utilidades y aplicaciones
  - **Acceso a servicios de terceros**: Puedes obtener datos de aplicaciones como **Twitter**, **Google Maps**, **APIs bancarias**, etc.
  - **Interacción entre servicios**: Las APIs permiten integrar diferentes servicios y crear aplicaciones más potentes.
  - **Automatización de procesos**: Integrar servicios a través de APIs puede ahorrar tiempo y recursos en tareas repetitivas.

API REST

- **Dificultades**
  - **Límites de uso**: Muchas APIs públicas tienen limitaciones en cuanto a la cantidad de solicitudes que puedes hacer, lo que puede ser problemático en grandes volúmenes de datos.
  - **Autenticación**: Algunas APIs requieren autenticación, lo que puede agregar complejidad al proceso.
  - **Dependencia de terceros**: Si la API cambia o se cae, tu aplicación podría verse afectada.

API REST

¿Cuándo usarlo?

- **Cuando necesites obtener datos de servicios externos** como redes sociales, servicios de pagos, mapas, etc.
- **Cuando no puedas o no quieras almacenar los datos tú mismo** y prefieras obtenerlos en tiempo

¿Cuándo no usarlo?

- No usarlo si necesitas datos internos o **tienes bases de datos propias con la información que necesitas.**

API REST

- REST, GraphSQL, Streaming.
- Suele requerir autenticación y manejo de peticiones.
- Puede haber límites de uso o cambios en la API.

API REST

- Verbos HTTP
  - GET: Solicitar datos de servidor.
  - POST: Enviar datos al servidor (normalmente para crear nuevos elementos)
    - Aunque en la práctica se utiliza para más cosas.
  - PUT y PATCH: Alterar algún elemento que ya está en el servidor.
  - DELETE: Eliminar un elemento en el servidor.
    - Suele devolver un 204
  - OPTION: Solicita información sobre el funcionamiento de la API.

API REST

- REST, GraphSQL, Streaming.
- Suele requerir autenticación y manejo de peticiones.
- Puede haber límites de uso o cambios en la API.

```
import requests

url = “https://catfact.ninja/fact”
response = requests.get(url)

# Obtener datos curiosos sobre gatos
if response.status_code == 200:
    data = response.json()
    print(“Interesting cat fact: ”, data[“fact”])
else:
    print(“Error”)
```

Logs y archivos de registro

- Los **logs** o **archivos de registro** son archivos donde las aplicaciones y sistemas almacenan información sobre su funcionamiento, eventos y errores. Estos archivos son esenciales para monitorear el estado de un sistema y para la depuración.
- Los logs pueden contener desde simples mensajes de estado hasta detalles sobre fallos y comportamientos no deseados. Generalmente contienen datos sobre nuestros propios sistemas.
- Utilidades y aplicaciones
  - **Monitoreo y diagnóstico de sistemas**: Los logs permiten a los administradores de sistemas realizar un seguimiento de la actividad y detectar problemas de manera temprana.
  - **Análisis de seguridad**: Se pueden usar para identificar intentos de acceso no autorizados o comportamientos sospechosos en sistemas.
  - **Mejora del rendimiento**: Al analizar los logs de las aplicaciones, se pueden identificar cuellos de botella o ineficiencias en el código o infraestructura.
  - **Auditoría de eventos**: Permiten realizar auditorías y mantener un registro histórico de todas las acciones dentro de un sistema.

Logs y archivos de registro

- **Dificultades**
  - **Volumen masivo de datos**: Los archivos de logs pueden crecer rápidamente, lo que hace que la lectura, almacenamiento y procesamiento sean un reto.
  - **Estructura inconsistente**: Los logs pueden no seguir un formato estructurado, lo que hace difícil procesarlos automáticamente sin un preprocesamiento adecuado.
  - **Filtrado y clasificación**: Extraer datos relevantes de grandes volúmenes de logs sin perder información importante es un reto, ya que los logs pueden contener millones de entradas.
  - **Análisis en tiempo real**: Los logs generados en tiempo real deben procesarse y analizarse rápidamente para poder tomar decisiones inmediatas.

Logs y archivos de registro

¿Cuándo usarlo?

- **Para monitorear aplicaciones y servidores**: Ideal para obtener información detallada sobre el funcionamiento interno de sistemas y aplicaciones.
- **Cuando se necesiten realizar auditorías o análisis de seguridad**: Los logs son esenciales para auditar accesos y detectar posibles brechas de seguridad.
- **En escenarios de análisis de rendimiento**: Al monitorear logs de aplicaciones, servidores y redes se pueden detectar problemas de rendimiento y optimizar los recursos.

¿Cuándo no usarlo?

- **No usarlo si los logs no contienen información relevante.**

Logs y archivos de registro

- Dependen del formato específico que tengan los logs.
- Útiles para aplicaciones webs propias y ciberseguridad.

```
import re

with open("server.log", "r") as log_file:
    logs = log_file.readlines()

# Mostrar todas las líneas que contengan “ERROR” en un fichero
error_logs = [log for log in logs if re.search(r"ERROR", log)]
print(error_logs)
```

OCR (Reconocimiento óptico de caracteres)

- El **OCR** (Reconocimiento Óptico de Caracteres) es una tecnología que convierte documentos escaneados, imágenes o archivos PDF que contienen texto en **datos estructurados**.
- Se utiliza comúnmente para digitalizar documentos físicos, leer textos de imágenes y convertirlos en datos procesables.
- Utilidades y aplicaciones
  - **Digitalización de documentos**: Convierte documentos físicos o imágenes escaneadas en texto editable.
  - **Extracción de texto de imágenes**: Útil para leer texto desde fotografías de libros, carteles o pantallas.
  - **Automatización del procesamiento de documentos**: Se puede aplicar a la automatización de tareas como el llenado de formularios, la clasificación de documentos, o la extracción de datos de facturas.

OCR (Reconocimiento óptico de caracteres)

- **Dificultades**
  - **Precisión limitada**: El OCR no siempre es 100% preciso, especialmente con textos escritos a mano o imágenes de baja calidad.
  - **Requiere imágenes de buena calidad**: Las imágenes deben ser claras y bien enfocadas para obtener buenos resultados.
  - **Problemas con textos complejos**: El OCR puede tener dificultades con fuentes no estándar o con caracteres especiales.

OCR (Reconocimiento óptico de caracteres)

¿Cuándo usarlo?

- **Cuando necesitas extraer texto de imágenes o documentos escaneados.**
- **Cuando los documentos están en formato físico y necesitan ser digitalizados** para ser procesados de manera automatizada.

¿Cuándo no usarlo?

- **No usarlo si el documento es de mala calidad** o la imagen es poco clara.
- Si la **precisión es crítica.**

Librería Teseract para OCR

- Tesseract
  - https://pytesseract.readthedocs.io/en/latest/
- Implementación en Python
  - https://pypi.org/project/pytesseract/

OCR (Reconocimiento óptico de caracteres)

- Reconocimiento de caracteres en imágenes.
- El tipo de letra y otros aspectos pueden hacer que no se reconozcan bien los caracteres.

```
import pytesseract
from PIL import Image

img = Image.open("texto.png")
text = pytesseract.image_to_string(img)
print(text)
```

Protocolos de IoT

- Los protocolos de IoT como MQTT y WebSockets permiten la comunicación en tiempo real entre dispositivos.
  - MQTT es un protocolo ligero de publicación y suscripción, especialmente diseñado para dispositivos con recursos limitados.
  - WebSockets permite comunicación bidireccional persistente entre cliente y servidor en tiempo real, ideal para aplicaciones web interactivas.
- La implementación dependerá del dispositivo que queramos monitorizar.
- Se pueden usar para IoT en tiempo real, Smart Homes e Industria 4.0

Protocolos de IoT

- **Dificultades**
  - **Conexión constante**: Los dispositivos deben estar siempre conectados para transmitir datos en tiempo real.
  - **Seguridad**: Es fundamental implementar medidas de seguridad como encriptación y autenticación para proteger los datos y evitar accesos no autorizados.
  - **Manejo de grandes volúmenes de datos**: Los sistemas IoT pueden generar grandes cantidades de datos rápidamente, por lo que es necesario tener una infraestructura de almacenamiento y procesamiento eficiente.
  - **Latencia**: Los retrasos en la transmisión de datos pueden afectar aplicaciones críticas, como sistemas de salud o seguridad.

Protocolos de IoT

¿Cuándo usarlo?

- **Sistemas que requieren comunicación en tiempo real** (smart homes, vehículos autónomos, dispositivos conectados).
- **Aplicaciones que requieren baja latencia** y alta confiabilidad en la entrega de datos.

¿Cuándo no usarlo?

- Cuando no se requiere comunicación constante.
- No es ideal para grandes bases de datos tradicionales.

Streaming de datos

- El **streaming de datos** permite el procesamiento de datos en tiempo real a medida que se generan.
- Herramientas como **Apache Kafka**, **Apache Flink** y **Apache Spark Streaming** son esenciales para procesar **flujos de datos** a gran escala.
- Utilidades y aplicaciones:
  - **Análisis en tiempo real**: Procesamiento de transacciones bancarias, logs de servidores, clics de usuarios en sitios web, etc.
  - **Monitoreo de eventos**: Detectar eventos críticos en tiempo real, como fraudes o intrusiones en sistemas.
  - **Procesamiento de flujos masivos de datos**: Ideal para empresas que generan grandes volúmenes de datos (como **streaming de video o audio**).

Streaming de datos

- **Dificultades**
  - **Escalabilidad**: Las soluciones de **streaming** deben ser capaces de escalar de manera eficiente para manejar flujos de datos masivos.
  - **Complejidad de la infraestructura**: Configurar y mantener un sistema de procesamiento en tiempo real como **Apache Kafka** o **Apache Flink** puede ser desafiante.
  - **Latencia y consistencia**: Es importante garantizar la consistencia de los datos mientras se procesan en tiempo real.
  - **Manejo de errores**: Los sistemas de streaming deben ser capaces de manejar errores y fallos de manera eficiente para evitar pérdida de datos.

Streaming de datos

¿Cuándo usarlo?

- **Sistemas que requieren procesamiento en tiempo real** como transacciones financieras, análisis de datos de sensores IoT, o análisis de redes sociales.
- **Análisis de grandes volúmenes de datos** generados constantemente (logs de servidores, tráfico web, etc.).

¿Cuándo no usarlo?

- Cuando el procesamiento batch tradicional es suficiente.
- Para sistemas con bajos requerimientos de latencia.