1. ¿Qué es la captación de datos?

La captación de datos es el proceso de **obtener información desde distintas fuentes** para poder usarla en:

- análisis,
- modelos de machine learning,
- soluciones de inteligencia artificial.

En IA, **los datos son el punto de partida**.
Un buen dataset = mejores modelos; malos datos = malos resultados.

¿Por qué es tan importante?

- Afecta a la **calidad** del modelo.
- Determina la **variedad y representatividad** de los datos.
- Condiciona cuánto trabajo necesitaremos en **limpieza y preparación**.

2. Tipos de fuentes de datos

A. Datos internos

Son los datos que ya tiene una organización.

Incluye:

- **Bases de datos internas** (datos almacenados en sistemas propios).
- **Logs de aplicaciones** (registros automáticos que generan los sistemas, como quién entra, qué consulta, errores...).
- **Datos transaccionales** (información de operaciones: compras, reservas, movimientos, etc.).
- **Sensores / IoT (Internet de las Cosas)**:
  dispositivos que captan datos del entorno (temperatura, tráfico, consumo eléctrico, etc.).

B. Datos públicos o abiertos

Son datos que cualquier persona puede consultar o descargar.

Ejemplos:

- Portales de datos abiertos de ayuntamientos, comunidades autónomas o ministerios.
- Datos publicados por universidades o centros de investigación.
- Conjuntos de datos que se ofrecen en plataformas educativas.

C. Datos obtenidos desde el exterior mediante técnicas programáticas

Aquí entran **las herramientas que permiten obtener datos directamente de internet.**

1. Consumo de APIs

Una **API** (Interfaz de Programación de Aplicaciones) permite pedir datos a un servidor de forma estructurada.

Conceptualmente:

- Tú haces una "pregunta" a un servidor.
- El servidor te responde normalmente en **JSON.**

Ventajas:

- La información ya viene ordenada.
- Es estable.
- Está pensada para ser usada por programas.
- Suele respetar normativa y legalidad.

2. Web Scraping

El **web scraping** consiste en obtener datos directamente del contenido de una página web (HTML).

Conceptualmente:

- Tu programa “visita” una web.
- Descarga su contenido.
- Extrae solo lo que necesitas (por ejemplo, precios, títulos, opiniones).

Ventajas:

- Permite obtener datos que no están disponibles mediante API.
- Muy flexible.

Inconvenientes:

- Puede romperse si el sitio cambia su estructura.
- Hay que tener en cuenta la legalidad (robots.txt y condiciones de la web).

3. Rastreo automático mediante enlaces (crawling)

Este es el método es similar a lo que hacen buscadores como Google.

Conceptualmente:

- Un programa “visita” una página.
- Detecta todos los enlaces que contiene.
- Visita esos enlaces también.
- Repite el proceso creando un “mapa” o índice de contenidos.

La diferencia entre crawling y scraping:

- **Crawling** = recorrer y descubrir páginas.
- **Scraping** = extraer datos de cada página.

El crawling suele ser una etapa previa cuando se quiere:

- recopilar una gran cantidad de páginas,
- descubrir nuevas fuentes,
- construir bases de datos de URLs,
- o preparar un scraping masivo.

¿En qué categoría encaja?
Forma parte de **las técnicas programáticas de captación de datos**, igual que las APIs y el web scraping.

3. ¿Cuándo usar cada técnica?

API

- Si existe y da la información que necesitas, **úsala siempre**.
- Es la opción más limpia y estable.

Web scraping

- Cuando no existe API.
- Cuando la página muestra información adicional que la API no ofrece.

Crawling

- Cuando necesitas descubrir automáticamente muchas páginas relacionadas.
- Cuando quieres construir un conjunto amplio de URLs antes de extraer datos.
- Cuando analizas la estructura de un sitio o sector.

4. Ética y legalidad

Es importante, antes de captar datos de un determinado sitio, analizar el estado legal de dichos datos.

- Revisar siempre el archivo **robots.txt** de la web.
- No saturar servidores (pausas entre peticiones).
- Consultar términos y condiciones de cada sitio.
- Preferir API si está disponible.
- No usar datos sensibles que identifiquen personas sin permiso.

El archivo robots.txt (normas para robots)

El **archivo robots.txt** es un pequeño fichero público que se encuentra en https://www.nombre-de-la-web.com/robots.txt.

Sirve para **indicar a los robots/crawlers qué partes del sitio pueden visitar y cuáles no**.

Se interpreta mediante reglas muy simples:

- **User-agent:** indica a qué robot va dirigida una regla.
  (Ej.: User-agent: * = todos los robots).
- **Disallow:** páginas o carpetas que el robot **no** debe visitar.
- **Allow:** zonas a las que sí puede acceder.
- **Sitemap:** ubicación del mapa del sitio.

No es un sistema de seguridad, sino un **manual de instrucciones para robots bienintencionados** (Google, Bing, rastreadores académicos, etc.).
Sin embargo, es importante **respetarlo siempre** en un proyecto educativo o profesional.

5. Cómo encaja la captación de datos en un proyecto de IA/ML

1. **Captación de datos** (tema de esta lección).
2. **Limpieza y preparación de datos** (tratamiento de valores nulos, tipos, normalización...).
3. **Análisis exploratorio** (EDA).
4. **Entrenamiento del modelo**.
5. **Evaluación del modelo**.
6. **Implementación**.
7. **Monitorización y mejora continua**.

6. Consumo de APIs en la captación de datos

Una **API (Interfaz de Programación de Aplicaciones)** es un servicio que permite **pedir datos a un servidor de forma estructurada**, sin necesidad de interactuar con la web de manera manual.

En Machine Learning e IA, las APIs son muy útiles porque **proporcionan datos limpios y organizados**, listos para análisis o entrenamiento de modelos.

6.1 Cómo se consume una API

Consumir una API significa **hacer una petición** al servidor y **recibir una respuesta** con los datos.

Esto se hace normalmente mediante **HTTP (HyperText Transfer Protocol)**, que es **el protocolo que usan los navegadores y aplicaciones para comunicarse con servidores en Internet**.
En otras palabras, cada vez que hacemos una petición a una API, estamos "hablando" con el servidor mediante HTTP, que define **cómo enviar la solicitud, cómo recibir la respuesta y qué significan los códigos que devuelve**.

Principales métodos de petición HTTP

Método	Qué hace	Ejemplo conceptual
GET	Solicita información o datos del servidor	"Dame todos los Pokémon de la PokeAPI"
POST	Envía información para crear un recurso nuevo	"Crear un nuevo usuario en un servicio"
PUT	Actualiza un recurso existente	"Modificar la información del Pokémon X"
DELETE	Elimina un recurso	"Borrar un registro del servidor"

Para captación de datos normalmente usamos **GET**, porque solo necesitamos **leer información**, no modificar nada.

6.2 Códigos de estado HTTP

Cuando hacemos una petición, el servidor responde con un **código de estado** que indica si todo fue correcto o si hubo algún problema:

Código	Significado
200 OK	Todo correcto, la respuesta contiene los datos

201 Created	Recurso creado correctamente (usualmente tras POST)
400 Bad Request	La petición es incorrecta o mal formada
401 Unauthorized	No tienes permiso, necesitas autenticación o token
403 Forbidden	Acceso prohibido a ese recurso
404 Not Found	No se encontró el recurso solicitado
500 Internal Server Error	Error en el servidor

En captación de datos para ML, los **200** son los que queremos. Si aparece otro código, hay que revisar la URL o la documentación de la API.

6.3 Formato de los datos

- La PokeAPI devuelve datos en **JSON** (JavaScript Object Notation).
- JSON es **estructurado y legible**, con objetos y listas que podemos transformar fácilmente en un **diccionario o dataframe** en Python.

Ejemplo conceptual de respuesta JSON de PokeAPI:

```
{
 "name": "pikachu",
 "id": 25,
 "types": [
  {"slot": 1, "type": {"name": "electric"}}
 ]
}
```

Esto indica que:

- El Pokémon se llama **Pikachu**
- Su ID es 25
- Su tipo es **electric**

6.4 Ejemplo paso a paso: obtener Pokémon usando PokeAPI

1. **Seleccionar el endpoint de la API**

URL para un Pokémon concreto:

https://pokeapi.co/api/v2/pokemon/pikachu

2. **Hacer una petición GET** al servidor

El servidor devuelve los datos del Pokémon en formato JSON.

3. **Interpretar los datos**
    a. Nombre
    b. ID
    c. Tipo
    d. Estadísticas, habilidades, etc.
4. **Usar los datos en Python** (ejemplo expuesto en la presente clase)
    a. Convertir JSON en diccionario o dataframe.
    b. Guardar la información para análisis o entrenamiento de modelos.

6.5 Buenas prácticas al usar APIs

- Leer siempre la **documentación oficial** de la API: endpoints, parámetros, límites y formato de datos.
- Respetar **límites de peticiones** (rate limits) para no sobrecargar el servidor.
- Manejar errores y códigos de estado (reintentar si hay un 500, avisar si hay un 404).
- Guardar los datos localmente si se van a usar muchas veces, evitando llamadas innecesarias.