MANIPULACIÓN DE DATOS

UNIDAD 2

Fundamentos de Ciencia de Datos 2024


SAY IT AGAIN DEXTER
I'M A DATA SCIENTIST

> ¿Qué son los DATOS?

Los datos son **piezas de información** de algún tema o área en particular.

Pueden venir en tres formas diferentes:

- Estructurados
- No estructurados
- Semiestructurados

<

DATOS ESTRUCTURADOS

Son aquellos en los que los *data points* comparten los mismos campos o atributos.

Los datos estructurados tienen un **formato estandarizado** que permite tanto al software como a las personas acceder a la información de una manera eficiente. Por lo general, suelen organizarse **en forma de tablas con filas y columnas** que definen claramente sus atributos.

DATOS ESTRUCTURADOS - características

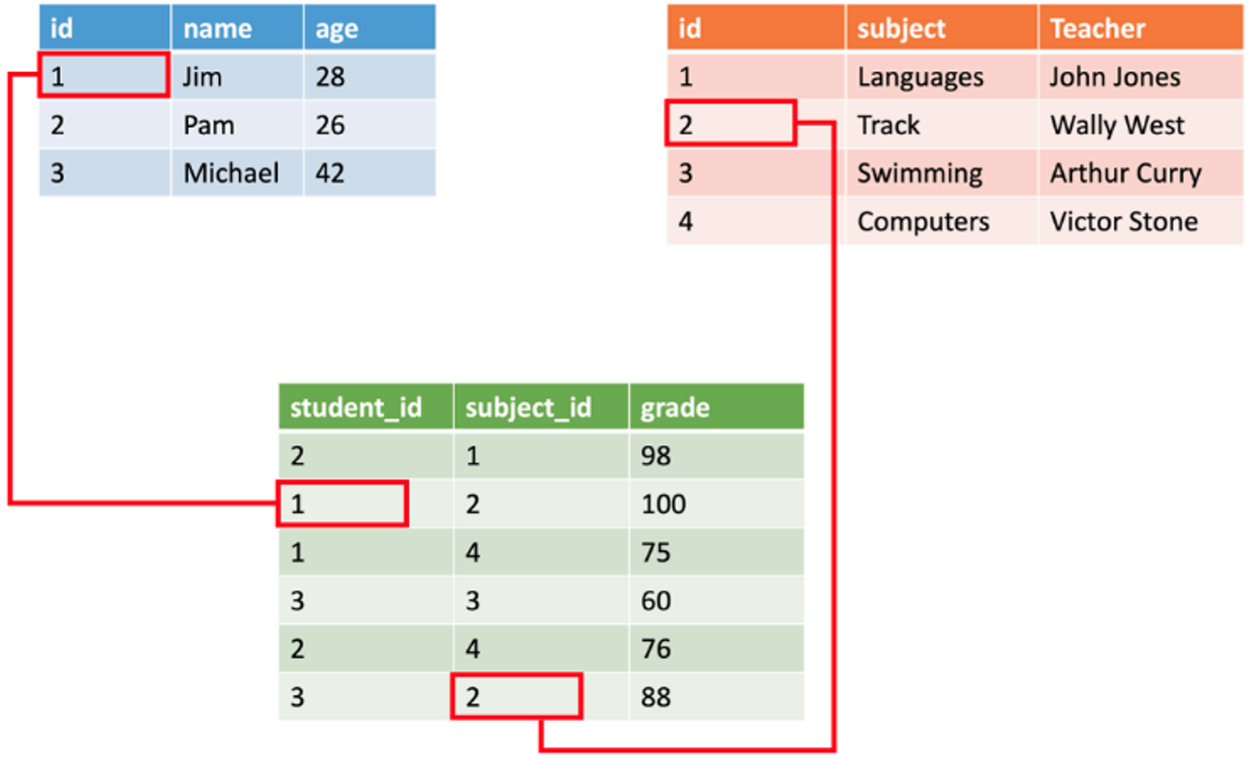
Atributos definibles

Los datos estructurados tienen los mismos atributos para los diferentes registros.

DATOS ESTRUCTURADOS - características

Atributos relacionales

Las tablas de datos estructurados tienen información común que permite vincular entre sí diferentes conjuntos de datos.



id	name	age
1	Jim	28
2	Pam	26
3	Michael	42

id	subject	Teacher
1	Languages	John Jones
2	Track	Wally West
3	Swimming	Arthur Curry
4	Computers	Victor Stone

student_id	subject_id	grade
2	1	98
1	2	100
1	4	75
3	3	60
2	4	76
3	2	88

DATOS ESTRUCTURADOS - características

Almacenamiento

Se pueden almacenar datos estructurados de manera conveniente en una **base de datos relacional** y administrarlos y consultarlos en forma sencilla con lenguajes como SQL (*Structured Query Language*).

DATOS NO ESTRUCTURADOS

Los datos no estructurados son información **sin un modelo de datos establecido** o son datos que **no están ordenados de una manera predefinida**. Generalmente no pueden representarse con filas y columnas.

Ejemplos: archivos de texto y video, informes, e-mails, chats, imágenes.

DIFERENCIAS

Facilidad de análisis Es **mucho más difícil organizar, limpiar y analizar datos que no tengan un modelo de datos predefinido** y muy pocas herramientas disponibles en el mercado pueden hacer esto.

Capacidad de búsqueda Es **fácil hacer búsquedas en los datos estructurados** ya que siguen diversas reglas predefinidas.

Flexibilidad **Los datos no estructurados tienen menos restricciones en su estructura**, por lo que es más fácil agregar nueva información en comparación con un conjunto de datos estructurados.

DATOS SEMI-ESTRUCTURADOS

Son una **categoría intermedia entre los datos estructurados y los no estructurados**. No pueden considerarse datos totalmente estructurados porque carecen de un modelo de datos relacional o tabular específico.

Utilizan ***tags*** para darle organización y jerarquías a los datos y se construyen con lenguajes de serialización, como **XML, JSON o YAML**.

XML (*Extensible Markup Language*)

Está basado en texto, lo que lo hace fácil de leer para humanos y computadoras.

El siguiente código representa el registro de una persona. **¿Qué información contiene?**

```
<Person Age="23">
    <FirstName>Quinn</FirstName>
    <LastName>Anderson</LastName>
    <Hobbies>
        <Hobby Type="Sports">Golf</Hobby>
        <Hobby Type="Leisure">Reading</Hobby>
        <Hobby Type="Leisure">Guitar</Hobby>
   </Hobbies>
</Person>
```

XML (*Extensible Markup Language*)

XML utiliza ***tags*** para darles forma a los datos, los cuales pueden ser:

- **Elementos**, como `<First Name>`.
- **Atributos**, como `Age='23'` .

A su vez, los elementos pueden tener elementos secundarios o hijos que permiten expresar relaciones, como `Hobby` dentro del elemento `Hobbies`.

XML (*Extensible Markup Language*)

VENTAJAS

- Es un lenguaje de marcado ampliamente utilizado y bien documentado, lo que lo hace **fácil de implementar en diversos lenguajes de programación.**
- Es flexible y extensible.

XML (*Extensible Markup Language*)

DESVENTAJAS

- El formato XML puede ser propenso a la **redundancia de datos**, lo que puede aumentar el tamaño del archivo.
- XML **puede ser difícil de manejar para archivos grandes**, ya que puede requerir más recursos de procesamiento y almacenamiento que otros formatos.
- La sintaxis puede tornarse compleja.

JSON

Es un formato de archivo liviano que se utiliza para almacenar e intercambiar datos y se basa en el lenguaje de programación **JavaScript**.

Utiliza una **estructura clave-valor** para representar datos y soporta el manejo de listas y jerarquías.

JSON usa llaves `{}` para indicar la estructura de los datos.

JSON

Ejemplo conocido...

```json
{
    "firstName": "Quinn",
    "lastName": "Anderson",
    "age": "23",
    "hobbies": [
        { "type": "Sports", "value": "Golf" },
        { "type": "Leisure", "value": "Reading" },
        { "type": "Leisure", "value": "Guitar" }
    ]
}
```

JSON

VENTAJAS

- Es un lenguaje **legible por humanos y fácil de entender**.
- Se puede **analizar y manipular fácilmente** con diferentes lenguajes de programación.
- **Admite tipos de datos complejos**, como matrices y objetos anidados.

JSON

DESVENTAJAS

- Puede ser menos eficiente para almacenar y procesar grandes conjuntos de datos en comparación con otros formatos binarios.
- Soporte limitado para la compresión de datos.

YAML

YAML Ain’t Markup Language (YAML) es un lenguaje de serialización con una gran legibilidad.

La estructura de los datos se da por la **separación de las líneas** y la **indentación** (se elimina la dependencia de caracteres especiales).

```yaml
firstName: Quinn
lastName: Anderson
age: 23
hobbies:
    - type: Sports
      value: Golf
    - type: Leisure
      value: Reading
    - type: Leisure
      value: Guitar
```

YAML

VENTAJAS

- Es **fácil de leer y editar por humanos**.
- Es un formato de texto plano, lo que lo hace **compatible con una amplia variedad de herramientas y lenguajes de programación.**
- Es **fácil de usar para estructurar y organizar datos complejos**.
- A diferencia de JSON, **permite utilizar comentarios y resulta más compacto en tamaño**.

YAML

DESVENTAJAS

- **No es eficiente para el almacenamiento de grandes cantidades de datos**, ya que puede ser propenso a la redundancia de datos.
- YAML **puede tener problemas de compatibilidad en diferentes lenguajes de programación y plataformas.**
- La falta de estándares para el formato YAML puede generar **ambigüedad en la interpretación de los datos.**

DATOS TABULARES

Los datos utilizados en el análisis de datos están generalmente representados en **forma tabular** (compuestos por filas y columnas).

Para guardar los datos se pueden usar diferentes tipos de archivos: **.csv, .json, .txt, .html, .parquet.**

ARCHIVOS ORIENTADOS A FILAS/COLUMNAS

ORIENTADOS A FILAS

Los datos **se organizan en registros** y todos los datos asociados a un registro se guardan juntos en la memoria. Por lo tanto, realizar consultas sobre el valor de un atributo para diferentes registros resulta ineficiente ya que se debe cargar todo el registro con datos innecesarios.

ARCHIVOS ORIENTADOS A FILAS/COLUMNAS

ORIENTADOS A COLUMNAS

Los datos **se organizan por columna/campo/variable** y todos los datos de la columna se guardan juntos en la memoria. Por lo tanto, cuando queremos consultar los valores de una columna, sólo necesitamos cargar esa columna sin necesidad de leer todo el archivo.

Además, al ser todos los datos de una misma columna del mismo tipo, la compresión del archivo es mejor.

ARCHIVOS ORIENTADOS A FILAS/COLUMNAS

Supongamos que tenemos la siguiente tabla:

dni	nombre	apellido	anio_nacimiento
40576890	Pedro	Aguirre	1995
32492645	Julia	Martinez	1988

¿Qué forma tendrá el archivo si dicha información se guarda orientada a filas?

¿Y si se guarda orientada a columnas?

ARCHIVOS ORIENTADOS A FILAS/COLUMNAS

Si la info se guarda **orientada a filas**:

row	value
row 1	40576890
	Pedro
	Aguirre
	1995
row 2	32492645
	Julia
	Martinez
	1988

ARCHIVOS ORIENTADOS A FILAS/COLUMNAS

Si la info se guarda **orientada a columnas**:

column	value
dni	40576890
	32492645
nombre	Pedro
	Julia
apellido	Aguirre
	Martinez
año_nacimiento	1995
	1988

ARCHIVOS PARA ALMACENAMIENTO DE DATOS TABULARES

A continuación, veremos diferentes formatos de archivos para almacenar datos tabulares, con los que trabajaremos a lo largo del semestre:

- **.csv**
- **.json**
- **.txt**
- **.html**
- **.parquet**

CSV (*Comma-Separated Values*)

Los diferentes registros (las filas) se separan entre sí mediante saltos de líneas, mientras que los atributos/variables (las columnas) se separan usando la **coma** (también se pueden usar otros caracteres como el **punto y coma** o el ***tab***). Hoy en día es uno de los formatos más utilizados en el análisis de datos.

```
Name, Age, Gender
John, 25, Male
Jane, 30, Female
Bob, 40, Male
```

CSV (*Comma-Separated Values*)

VENTAJAS

- Casi todos los softwares que realizan tratamiento de datos pueden leerlos y escribirlos con facilidad, y además son fáciles de leer para las personas.
- Es sencillo generarlos desde casi cualquier lenguaje de programación.

CSV (*Comma-Separated Values*)

DESVENTAJAS

- No es eficiente para almacenar grandes conjuntos de datos con tipos de datos complejos.
- Puede provocar la pérdida de datos si los valores contienen comas o saltos de línea.
- El formato está **orientado a filas**. Por lo tanto, realizar consultas de agregación tiende a ser poco eficiente.
- Los archivos .csv **no guardan información acerca del tipo de dato** de su contenido puesto que todo se guarda en texto simple.
- Cuando tienen un tamaño considerable su lectura puede volverse bastante lenta.

TXT

Es uno de los formatos más simples y ampliamente utilizados para el almacenamiento de datos.

Cuando se almacenan datos tabulares, los archivos TXT suelen ser muy similares a los CSV.

Aplicaciones: se utiliza comúnmente para almacenar grandes conjuntos de datos de texto, como documentos, transcripciones, registros de chat, mensajes de correo electrónico, etc. También se utiliza en el procesamiento de lenguaje natural (NLP) para almacenar y analizar conjuntos de texto, como textos médicos, noticias, redes sociales, entre otros.

TXT

VENTAJAS

- Poseen una **gran facilidad de uso y compatibilidad** con una amplia variedad de herramientas y lenguajes de programación.
- El formato es **legible por humanos**, lo que permite una fácil inspección y edición de datos.
- Los archivos de texto sin formato son adecuados para el **intercambio de datos entre sistemas**, dada la simplicidad para leerlos y generarlos, aunque es necesario conocer como están estructurados los datos para poder luego procesar su información.

TXT

DESVENTAJAS

- La falta de estructura en los archivos de texto sin formato puede **dificultar su procesamiento automatizado**. Esto puede hacer que el análisis de datos sea más lento y menos preciso en algunos casos y que se requiera de validaciones o de la construcción de funciones especiales para interpretarlos (***parseo***).
- El formato .txt **no es adecuado para almacenar datos complejos o información multimedia**, como imágenes, audio o video. Tampoco resulta eficiente para almacenar grandes cantidades de datos con una alta densidad de información.

APACHE PARQUET

Es un formato para el almacenamiento de datos tabulares **orientado a columnas** y está optimizado para **grandes cargas de datos**. Es de uso común en sistemas de procesamiento de *Big Data* basados en Hadoop**,** como Hive, Impala y Spark. Fue desarrollado por Cloudera y Twitter en 2013 como un proyecto de código abierto.

APACHE PARQUET

Parquet se basa en una representación de datos en **columnas comprimidas**, lo que lo hace muy eficiente para consultas analíticas que involucran grandes cantidades de datos. Es un formato popular para el procesamiento de *Big Data*.

APACHE PARQUET

VENTAJAS

- **Compresión eficiente:** Parquet es muy eficiente en lo que respecta a la compresión, lo que reduce los requisitos de almacenamiento y mejora el rendimiento de las consultas.
- **Orientado a columnas:** Parquet almacena datos en columnas en lugar de filas, lo que lo hace más eficiente para consultas analíticas que generalmente implican leer sólo un subconjunto de columnas de un gran conjunto de datos.

APACHE PARQUET

VENTAJAS

- **Evolución del esquema:** Parquet admite la evolución del esquema, lo que significa que puede agregar, eliminar o modificar columnas sin romper la compatibilidad con los datos existentes. Esto facilita la actualización de los modelos de datos a lo largo del tiempo.
- **Soporte multiplataforma:** Parquet es un proyecto de código abierto y es compatible con una variedad de sistemas de procesamiento de *Big Data*.

APACHE PARQUET

DESVENTAJAS

- **Rendimiento de escritura:** el formato de almacenamiento en columnas de Parquet puede ser más lento que los formatos basados en filas para escrituras, especialmente cuando se agregan datos a columnas existentes.
- **No apto para conjuntos de datos pequeños:** Parquet está optimizado para consultas analíticas a gran escala y no es adecuado para conjuntos de datos pequeños.

HTML

El formato HTML (Lenguaje de Marcado de Hipertexto o *HyperText Markup Language*) es un lenguaje utilizado principalmente para crear páginas web y documentos de hipertexto. En el campo de la ciencia de datos, se puede utilizar como un formato de almacenamiento de datos estructurados y no estructurados.

HTML

Ejemplo:

```
<html>
    <head></head>
    <body>
        <table id="customers">
  <tbody>
        <tr>
        <th>Company</th>
        <th>Contact</th>
        <th>Country</th>
      </tr>
      <tr>
```

HTML

VENTAJAS

- HTML es un lenguaje ampliamente utilizado y bien documentado, lo que lo hace **fácil de entender y manipular.**
- HTML es **compatible con una amplia variedad de herramientas y lenguajes de programación**, lo que lo hace una opción conveniente para la integración en flujos de trabajo de Ciencia de Datos.
- Los datos HTML se pueden analizar para extraer **información estructurada y no estructurada.**

HTML

DESVENTAJAS

- El formato HTML puede ser complejo, lo que puede dificultar la extracción de datos específicos de páginas web grandes y complejas.
- El formato HTML puede ser susceptible a cambios en la estructura de la página, lo que puede afectar la calidad y la precisión de los datos extraídos.
- HTML no es un formato de almacenamiento de datos óptimo para grandes cantidades de datos o datos no estructurados.

LECTURA DE ARCHIVOS DE DATOS TABULARES

LIBRERÍA PANDAS Con `pandas` podemos leer archivos **.csv, .txt, .xlsx o .xls, .parquet y .json**

- **CSV.** Si bien el archivo .csv sigue siendo orientado a filas, la librería se encarga de ponerlo dentro de un objeto **DataFrame**. `pd.read_csv()` también permite leer archivos .txt.

```
1 import pandas as pd
2
3 data = pd.read_csv('datasets/listings_ba.csv')
```

LECTURA DE ARCHIVOS DE DATOS TABULARES

LIBRERÍA PANDAS

- **EXCEL.** La función `read_excel()` nos permite leer archivos .xlsx o .xls. Si el archivo en cuestión tiene más de una hoja, se debe especificar el nombre de la hoja con la que se quiere trabajar en el argumento `sheet_name`.

```
1 import pandas as pd
2
3 data = pd.read_excel('datasets/lista_personas.xlsx', sheet_name = 'da
```

LECTURA DE ARCHIVOS DE DATOS TABULARES

LIBRERÍA PANDAS

- **JSON.** `pandas` cuenta con la función `read_json()` que posibilita la lectura/importación de archivos JSON al entorno de trabajo. Esta función convierte automáticamente los datos en un objeto **`pd.DataFrame`**.

```
1 import pandas as pd
2
3 data = pd.read_json('datasets/datos.json')
```

LECTURA DE ARCHIVOS DE DATOS TABULARES

LIBRERÍA JSON

- La librería nativa `json` en Python proporciona herramientas para trabajar con datos JSON. Para leer un archivo .json, primero se debe abrir el archivo en modo de lectura y luego utilizar la función `load()` de la librería `json` para cargar los datos en un objeto Python.

```
1  import json
2  import pandas as pd
3
4  # Abrir archivo JSON en modo lectura
5  with open('datasets/datos.json', 'r') as f:
6
7      # Cargar los datos JSON del archivo en un objeto Python
8      datos = json.load(f)
```

LECTURA DE ARCHIVOS DE DATOS TABULARES

LIBRERÍA PANDAS

- **APACHE PARQUET.** `pandas` cuenta con la función `read_parquet()` para la lectura de archivos con este formato. El parámetro `engine` nos permite seleccionar la librería específica de parquet para leer el archivo: io.parquet.engine (`auto`), `pyarrow`, `fastparquet`.

```
1 import pandas as pd
2
3 pd.read_parquet('datasets/datos.parquet', engine = 'auto',
4 columns = None, storage_options = None, use_nullable_dtypes = False)
```

ESCRITURA DE DATOS TABULARES EN ARCHIVOS

¿Cómo guardamos nuestros datos en distintos formatos de archivos usando **pandas**?

TIPOS DE DATOS USUALES

- **`int`**: para representar valores enteros. En general, el tipo de dato entero puede ser *byte*, *short*, *int* o *long,* y cada uno se corresponde con el volumen de memoria que puede ocupar el dato: 8, 16, 32 o 64 bits respectivamente. A partir de Python 3, todos los enteros son de formato `long`, es decir que ocupan 64 bits de memoria.
- **`float`**: para representar valores reales de coma flotante. Existe el `float` *single precision* (32 bits) y el *double precision* (64 bits).
- **`str`**: sirve para representar texto.
- **`bool`**: para representar valores booleanos de True/False
- **`NaN/None`**

`object` vs. `str`

En pandas, las columnas que contienen valores de texto se representan como el tipo de datos **`object`** en lugar del tipo de datos **`str`** de Python. Esto se debe a que **`str`** es un tipo de datos específico en Python que sólo puede contener caracteres alfanuméricos y algunos caracteres especiales, mientras que **`object`** puede contener cualquier tipo de objeto de Python, incluyendo cadenas de texto.

Además, las columnas de texto en un **DataFrame** de pandas pueden contener valores faltantes (por ejemplo, NaN), y **`object`** es un tipo de datos compatible con este tipo de datos.

MANEJO DE FECHAS

El módulo **`datetime`** de Python nos provee clases para manipular fechas y horas. Los objetos de fecha y hora pueden ser categorizados como **aware** o **naive**, dependiendo de si incluyen o no información sobre el huso horario.

Un **objeto de fecha y hora de tipo *aware*** contiene información sobre la zona horaria, lo que lo hace inequívoco en cuanto a la representación de un momento específico en el tiempo. De esta forma, para crear este tipo de objetos es necesaria la ayuda de la librería **`pytz`** .

MANEJO DE FECHAS

Ejemplo de creación de un **objeto de fecha y hora de tipo *aware***

```python
from datetime import datetime
import pytz

zona_horaria = pytz.timezone('America/Argentina/Buenos_Aires')
fecha_aware = datetime(2024, 8, 26, 18, 00, 0, tzinfo = zona_horaria)

print('Horario de la clase virtual de Fundamentos:', fecha_aware)
```

```
Horario de la clase virtual de Fundamentos: 2024-08-26 18:00:00-03:54
```

La lista de husos horarios incluida dentro del módulo `pytz` puede consultarse en el siguiente link.

MANEJO DE FECHAS

Por el contrario, un **objeto de fecha y hora *naive*** no contiene información sobre el huso horario. Representa una fecha y hora determinadas, pero no está claro a qué zona horaria se refiere.

```python
from datetime import datetime

fecha_naive = datetime(2024, 8, 26, 18, 00, 0)

print(fecha_naive)
```

```
2024-08-26 18:00:00
```

MANEJO DE FECHAS EN `pandas`

Si nuestros datos están en un archivo .csv y tienen una columna que contiene **fechas**, podemos usar el método `pd.to_datetime()` para indicarlo. Como resultado, transformarmos esa información en datos de tipo **`datetime64`**.

SOBRE `datetime64`

`datetime64` es un tipo de datos numérico que se representa internamente como un **número entero de 64 bits.**

Cada unidad de fecha y hora (año, mes, día, hora, minuto, segundo, nanosegundo) se convierte en un número entero que representa la cantidad de esa unidad desde una fecha de referencia, que es el **1 de enero de 1970**. Esta fecha de referencia se utiliza como base para el cálculo de todas las demás fechas y horas.

La precisión de **`datetime64`** se puede controlar mediante los modificadores de unidades de tiempo. Por ejemplo, **`datetime64[s]`** representa una fecha y hora con precisión de segundos, **`datetime64[ms]`** con precisión de milisegundos y **`datetime64[us]`** con precisión de microsegundos.


So what's your idea of a perfect date?
IG @PunHubOnline
YYYY-MM-DD
I find other formats a bit confusing.
Pun hub