Métodos de Captura Noviembre 2023

Tema 1 Introducción

Este tema realiza una introducción a la asignatura y presenta las definiciones necesarias sobre las que se basarán los temas posteriores. Para estudiar este tema se deben leer las Ideas clave y, si se desea información adicional sobre un concepto específico, se debería consultar las lecturas sugeridas en la bibliografía (sección A fondo).

La lectura de los casos de estudio también será de ayuda, ya que proporcionan información de las decisiones tomadas para la captura de datos en un entorno determinado. Este capítulo servirá tanto de introducción como de repaso de los conceptos básicos sobre bases de datos en general y bases de datos relacionales en específico.

En tiempos pasados, la obtención de datos solía ser un proceso costoso y a menudo requería del trabajo humano para la digitalización de documentos físicos. El continuo avance de la tecnología permite que los métodos disponibles para la captura de datos sean cada vez más diversos. Esto, en adición a la disminución del coste por unidad de almacenamiento, ha resultado en la posibilidad de almacenar grandes cantidades de datos provenientes de distintas fuentes.

A raíz de esta diversidad, es necesario contar con tres elementos que conforman la base para capturar datos de forma eficiente y adecuada al uso que se le brindará a dichos datos:

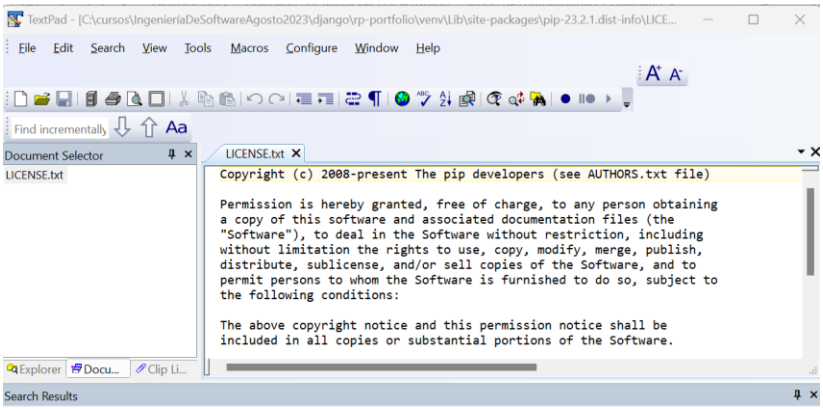
1. El primer elemento es la definición y distinción de los términos **dato, información y conocimiento**.
2. El segundo, las **dimensiones o criterios de calidad** que se analizan en un conjunto de datos.
3. El último elemento es la **experiencia de analizar casos verdaderos** en los que ha sido necesario capturar datos y almacenarlos.

El aprendizaje de formatos de ficheros de texto como **CSV y JSON** será de utilidad en los próximos temas, ya que son formatos utilizados por el sistema de almacenamiento.

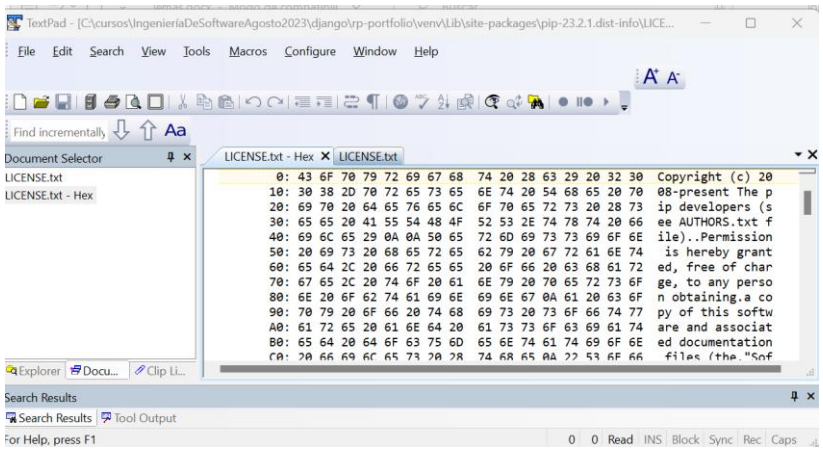
Después de capturar información y poder utilizarla de forma eficiente, será necesario almacenarla de forma permanente (**persistencia de datos**). Así, la información obtenida no residirá únicamente en la memoria volátil del ordenador, sino que estará disponible para futuras ocasiones.

El método más básico para almacenar datos es mediante el uso del sistema de ficheros del sistema operativo. Los ficheros pueden tener un **formato plano**, donde toda la información es legible para una persona; o un **formato binario**, donde la información puede escribirse y leerse de forma directa por una aplicación, pero no puede ser analizada directamente de forma manual.

Un archivo cualquiera: interpretado por un editor de texto



Y su contenido binario



Nota

El conjunto de caracteres ASCII (American Standard Code for Information Interchange) es un estándar de codificación de caracteres que se utiliza comúnmente en la informática y las comunicaciones. ASCII define un conjunto de 128 caracteres, incluyendo letras, números, signos de puntuación y caracteres de control, y asigna a cada uno de estos caracteres un valor numérico entre 0 y 127.

Los primeros 32 caracteres (0-31) son caracteres de control que se utilizan para funciones como el retorno de carro, avance de línea y tabulación, entre otros. Los caracteres del 32 al 126 son los caracteres imprimibles, que incluyen letras mayúsculas y minúsculas, números y signos de puntuación comunes. El valor numérico 127 se utiliza como el carácter "Delete" (borrar).

A continuación, algunos ejemplos de caracteres ASCII comunes:

Letras mayúsculas: A, B, C, ..., Z, Letras minúsculas: a, b, c, ..., z, Números: 0, 1, 2, ..., 9

Signos de puntuación: !, ", #, $, %, &, ', (,), *, +, ,, -, ., /, :, ;, <, =, >, ?, @, [, ,], ^, _, `, {, |, }, ~

Caracteres de control: Tabulación, Retorno de carro, Avance de línea, etc.

El conjunto de caracteres ASCII es ampliamente compatible y se utiliza en una variedad de sistemas y aplicaciones, especialmente en sistemas informáticos más antiguos. Aunque ha sido superado por conjuntos de caracteres más amplios como Unicode, el conjunto de caracteres ASCII sigue siendo fundamental en informática y se utiliza como base para muchos otros conjuntos de caracteres y codificaciones de caracteres.

Este tema proporciona un repaso de los mecanismos comúnmente utilizados para el almacenamiento de información. Se hablará de dos aproximaciones:

1. la utilización de ficheros planos, que son comúnmente utilizados para el almacenamiento y compartición de datos, y
2. las bases de datos, que dan un paso más allá y proporcionan una consistencia en la información y el hecho de poder consultar y modificar de manera eficiente un conjunto de datos en específico. Es por esto que se revisará el concepto de base de datos y, en particular, las bases de datos relacionales.

Además de ser una de las herramientas más comunes en la actualidad, se puede hacer una analogía entre estas herramientas y el sistema que se utilizarán en otros temas de la asignatura.

En situaciones informales es común usar indiscriminadamente los términos ***dato, información y conocimiento***. En ámbitos profesionales y académicos, es conveniente distinguir estos conceptos para evitar malinterpretaciones durante las distintas fases de la analítica de datos.

Existen varias aproximaciones para la distinción de estos términos. En el contexto de esta asignatura se utilizará la definición descrita por Davenport y Prusak (2000).

Nota

Thomas H. Davenport y Laurence Prusak, en su libro "Working Knowledge: How Organizations Manage What They Know" publicado en el año 2000, ofrecen una perspectiva sobre la diferencia entre datos, información y conocimiento en el contexto organizacional. Aquí está su enfoque:

- **Datos:** Los datos son hechos crudos y no procesados que representan eventos o transacciones. Son la materia prima de la información y el conocimiento. Los datos pueden ser números, palabras, símbolos o registros que no tienen significado por sí solos. Por ejemplo, números en una hoja de cálculo, letras en un documento o registros de ventas sin contexto.
- **Información:** La información es el resultado de procesar y dar sentido a los datos. Cuando los datos se organizan, se estructuran y se relacionan entre sí, adquieren significado. La información es un conjunto de datos que se utiliza para tomar decisiones, responder preguntas o realizar tareas. Por ejemplo, un informe de ventas que muestra ingresos por región es información derivada de datos de ventas.
- **Conocimiento:** El conocimiento va un paso más allá de la información. Es la comprensión y el contexto que las personas aplican a la información para tomar decisiones y resolver problemas. El conocimiento incluye la experiencia, la intuición y la capacidad de aplicar la información de manera efectiva en situaciones concretas. El conocimiento se basa en la experiencia y la comprensión acumulada a lo largo del tiempo.

Un dato puede definirse como un hecho concreto y discreto acerca de un evento. La característica de ser discreto significa que, semánticamente, es la unidad mínima que puede comunicarse o almacenarse. Por sí solos, los datos no brindan detalles significantes del entorno del que fueron obtenidos. Ejemplos de datos pueden ser:

- 2010.
- 443.
- DE.

La información puede distinguirse simplemente como un mensaje formado por la composición de varios datos. Esto significa que, a diferencia del dato, la información sí posee un significado para un receptor u observador. Por ejemplo, utilizando los datos anteriores se podría obtener la siguiente información:

- El año de establecimiento de la empresa ACME fue el 2010.
- La altura del edificio Empire State es de 443 metros.
- DE es el código ISO que identifica al idioma alemán.

Los datos deben ser transformados para añadirles valor y convertirlos en información. **Estas transformaciones** incluyen métodos como:

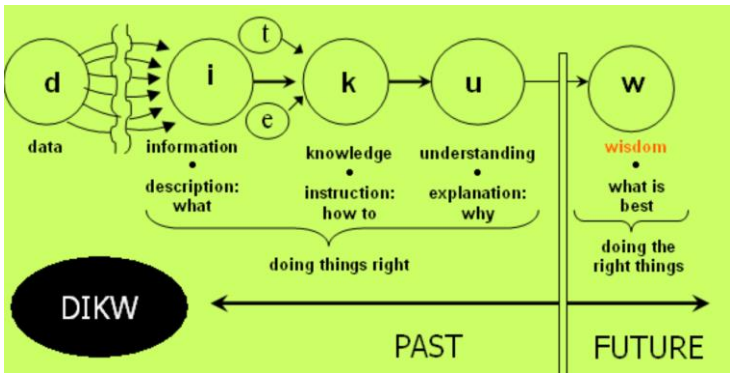
- Contextualización: conocer el propósito del dato obtenido.
- Categorización: conocer la unidad de medida y los componentes del dato.
- Cálculo: realizar una operación matemática sobre el dato.
- Corrección: eliminar errores del dato.
- Agregación: resumir o minimizar un dato de forma más concisa.

El conocimiento implica una combinación de experiencias, información contextual y relevancia sobre cierta información. Así como la información se genera a partir de datos, el conocimiento surge de la agregación de información. Ejemplos de métodos que generan esta transformación son:

- Comparación: relación entre información obtenida en distintas experiencias.
- Repercusión: implicación de la información en decisiones y acciones.
- Conexión: relación entre distintos tipos de información.
- Conversación: opinión de otras personas sobre la información.

La jerarquía del conocimiento suele representarse gráficamente por una pirámide, siendo los datos la base y el conocimiento la cima.

DIKW pyramid: Each step up the pyramid creates value based on the initial data, and can be used to answer high-level questions



Origen y calidad de los datos

Distinguir estos conceptos básicos proporciona un nivel de abstracción útil para la separación de características en el proceso de análisis. El hecho de que un dato sea inválido o erróneo debe distinguirse fácilmente de que la información que se obtiene de dicho conjunto de datos sea adecuada o no al problema que se intenta resolver.

Por ejemplo, es importante conocer si la malinterpretación de un análisis de datos se debe a un error en la fuente de datos, a un problema al combinar los datos en el proceso de análisis o a una confusión por parte del usuario final, debido a experiencias en otros contextos.

Evaluación de la calidad

Las **métricas o dimensiones utilizadas para describir la calidad** de un conjunto de datos pueden agruparse con base en los actores que interactúan con los datos. Los diseñadores y administradores de almacenes de datos tratan con métricas que afectan el diseño o esquema de los datos obtenidos y no con los datos directamente. Entre estas métricas se podría mencionar la completitud de los datos obtenidos y el minimalismo del conjunto de datos, el cual se interpreta como la eliminación de redundancias en el diseño del almacén de datos.

Los desarrolladores de software tratan con métricas específicas de la producción de productos informáticos. Si bien en esta categoría entran métricas que no están relacionadas con los datos directamente, es importante tener en cuenta que estas métricas afectan al proceso de almacenaje, acceso y manipulación de los datos.

El último actor que interactúa con los datos es el usuario final, es decir, quien utilizará los datos presentados para crear una conclusión y tomar una decisión acorde. Las métricas relevantes para este actor pueden ser el nivel de disponibilidad de los datos y el nivel de interpretación requerido para entender los datos presentados.

A partir de estas perspectivas, Jarke et al. (1998) presentan un conjunto de dimensiones para evaluar las propiedades de un conjunto de datos:

1. **La completitud** o cobertura describe el porcentaje de datos disponibles respecto a la población total que representa dichos datos. Por ejemplo, un conjunto de datos con información de 90 de 100 tratamientos médicos realizados en un hospital presenta una cobertura de 90 %. Esto también se aplica a subconjuntos de los datos obtenidos, por ejemplo, el 5 % de los tratamientos médicos no incluyen la fecha de finalización del tratamiento, lo cual disminuiría la cobertura debido a esta característica.
2. **La credibilidad** representa la fiabilidad que se le brinda al organismo que proporciona el conjunto de datos. Esta métrica puede reflejarse en el conjunto de datos por aquellas características que presenten un valor por defecto. Por ejemplo, si se habla de postres, es fácil ver que, si incluye valores por defecto como tarta, helado, fruta, etc., el dato es fiable, pero nunca lo será si se habla de pincho moruno.
3. **La precisión** indica el porcentaje de datos correctos respecto al total disponible. También se representa por medio de un porcentaje.
4. **La consistencia** describe el nivel con el que los datos son coherentes entre ellos. Un ejemplo de la aplicación de esta métrica se da en datos geográficos: se puede concluir que si una misma entidad tiene asociada una ciudad y un país que no tienen relación, existe un problema de consistencia.
5. **La interpretabilidad** define el grado en el que los datos pueden ser entendidos correctamente por una persona. Entre los atributos que definen la interpretabilidad de un conjunto de datos se puede mencionar la documentación de elementos importantes y si el formato en el que se proporcionan los datos es entendible.

Caso de estudio

A continuación, se proporciona un conjunto de datos de fuente desconocida representadas en la siguiente tabla:

Datos de ejemplo					
Marca	Modelo	Precio	Motor	Consumo	Fecha
Audi	La Ferrari	15 000 €	963 CV diésel	alto	2014
Lamborghini	Murciélago	300 000 €	580 CV 7500 rpm		2011
BMW	M6	158 000€	560 CV 4935 rpm	13,6/7,6/9,9	2012
Porsche	911 turbo	127 000 €	420 CV 7500 rpm	11 litros	2017

Tabla 1. Datos de ejemplo.

Si analizamos estos datos con respecto a los indicadores de calidad antes mencionados:

1. **Completitud**: 75 % respecto a los datos de consumo, o el mismo porcentaje con respecto a las revoluciones por minuto del motor del coche.
2. **Credibilidad:** al tratarse de una fuente desconocida, la credibilidad es baja debido a que no se puede comprobar la fuente. Además, por defecto, se sabe que Ferrari solo hace motores de gasolina y que, por defecto, no existen motores diésel de ese caballaje.
3. **Precisión:** es claro ver que los datos no son precisos, ya que el precio de un súper deportivo no puede ser de 15 000€ y las revoluciones de algunos coches son demasiado redondas. Además, un consumo «alto» no es una cifra de consumo.
4. **Consistencia:** la consistencia de los datos es baja porque a poco que se entienda algo de coches, el modelo de Ferrari nunca puede pertenecer a la marca Audi, sino a Ferrari.
5. **Interpretabilidad:** podría mejorarse, puesto que hay muchos datos que no incluyen unidades o que ofrecen un conjunto de números que si no se está familiarizado con el entorno no se conocerá su significado, como ocurre con los 3 valores de consumo.

Fuentes de información

Los métodos para la captura de información se pueden clasificar con base en las características del elemento que genera el conjunto de datos. Las cinco categorías más utilizadas en la actualidad son:

1. **La captura manual de datos** es la categoría más tradicional y también una de las más frecuentes en el contexto de investigación en ámbitos sociales y naturales. Entre los métodos que encajan en esta categoría se encuentra el uso de encuestas y las mediciones a través de observaciones.

   Si bien esta es la única categoría que no tiene una dependencia directa de las tecnologías de información, para su utilización es necesario que la información se digitalice, ya sea en el momento de la captura o en un procesado posterior.

2. **El procesado de documentos estructurados** consiste en la obtención directa de datos disponibles en documentos, cuyo fin inicial no es ser consultados como fuente de datos. Uno de los métodos más comunes en esta categoría es el procesado de páginas HTML en un sitio web, conocido como web scraping.

   Otro ejemplo es el análisis de logs o ficheros que contienen un listado secuencial de los eventos ocurridos dentro de un sistema y son creados con el objetivo de tener una bitácora y no para ser accedido por otras aplicaciones.

3. **La salida de aplicaciones** es una de las categorías más triviales. Los métodos de este tipo involucran el acceso a almacenes de datos tradicionales, tales como bases de datos relacionales, ficheros con valores separados por comas (CSV), etc.
4. **Los datos obtenidos a través de sensores**. En este conjunto se pueden mencionar ejemplos como sensores meteorológicos, sensores de ambiente (ruido o luz), sensores corporales (ritmo cardíaco o conductividad de la piel) y sensores de dispositivos móviles (acelerómetro o giroscopio).

   Actualmente existe un gran interés en el uso de sensores para la captura de datos en el contexto personal; este movimiento es conocido como ***quantified self***.

5. **El acceso a datos públicos** ya sea mediante la descarga de conjuntos de datos o a través de interfaces de programación de aplicaciones (API, por sus siglas en inglés). Por un lado, muchas de las entidades públicas (como gobiernos centrales y locales) publican catálogos de datos para que sean analizados y se desarrollen nuevas aplicaciones a partir de ellos.

   La publicación de estos datos de carácter público suele realizarse en un portal dedicado, por ejemplo http://datos.gob.es en España, http://data.gov.uk en Reino Unido y http://data.gov en Estados Unidos. Por otro lado, algunas empresas brindan acceso a datos de forma pública, siendo uno de los fines el formar parte de una plataforma de desarrollo, como, por ejemplo, los servicios de redes sociales (Facebook o Twitter) que suelen brindar un API tanto para obtener información sobre un usuario en particular, como para alterar dicha información.

Nota

El término "Quantified Self" (QS), que se traduce como "Uno Mismo Cuantificado" en español, se refiere a un movimiento y enfoque que involucra el uso de sensores y tecnología para recopilar datos sobre diversos aspectos de la vida cotidiana de una persona, como la salud, la actividad física, la nutrición, el sueño, el estado de ánimo y otros. Los datos recopilados se utilizan para cuantificar y analizar aspectos de la vida personal con el objetivo de tomar decisiones informadas y mejorar la calidad de vida.

En el contexto de sensores, el Quantified Self implica la recopilación continua de datos mediante dispositivos y sensores que se pueden llevar puestos, como dispositivos de seguimiento de la actividad, relojes inteligentes, monitores de frecuencia cardíaca, aplicaciones móviles y otros dispositivos conectados. Estos sensores registran información en tiempo real, como la frecuencia cardíaca, la cantidad de pasos, la calidad del sueño, la ingesta de alimentos, la presión arterial, y otros indicadores de salud y bienestar.

Los usuarios del movimiento Quantified Self pueden realizar un seguimiento de su progreso a lo largo del tiempo, establecer metas personales y tomar decisiones basadas en los datos recopilados. Por ejemplo, alguien podría utilizar un dispositivo de seguimiento de actividad para medir cuánto ejercicio hace durante la semana y luego ajustar su rutina de ejercicios en función de los datos recopilados.

Archivos planos

Los ficheros planos suelen ser un mecanismo utilizado para el intercambio de información entre sistemas. Una de sus ventajas es que es posible ver y editar el contenido del fichero con una herramienta de edición de texto.

Estos ficheros suelen ser mucho más verbosos que los ficheros en formato binario, lo cual implica que su tamaño en el sistema de ficheros será mayor **(pregunten por qué**), así como las operaciones necesarias para procesar el contenido desde un programa de software. Entre los formatos de fichero plano más comunes se pueden mencionar **CSV, JSON y XML**. A continuación, se describen los detalles básicos de cada uno de ellos.

Formato CSV

El formato CSV (Comma Separated Values o valores separados por coma), se documenta en la RFC 4180 y presenta las siguientes características:

- Cada registro se delimita por un cambio de línea (combinación de dos caracteres: CR y LF).
- Como su nombre indica, los valores de cada registro se separan mediante el uso de comas. Es requerido que el número de valores sea constante para todos los registros disponibles en el fichero.
- Los valores pueden estar encapsulados con comillas dobles. Esto es obligatorio en aquellos casos donde el valor incluye un cambio de línea, una coma o comillas dobles.
- Si un valor contiene comillas dobles, estas deben escaparse precediéndolas con otro carácter de comillas dobles. Por ejemplo: «Encargado de ""Business Model""»
- Opcionalmente, puede incluir una primera línea con los nombres de los campos que se incluyen en el fichero.

El archivo iris.csv visualizado con datagrip.

sepal_length	sepal_width	petal_length	petal_width	species
5.1	3.5	1.4	0.2	setosa
4.9	3	1.4	0.2	setosa
4.7	3.2	1.3	0.2	setosa
4.6	3.1	1.5	0.2	setosa
5	3.6	1.4	0.2	setosa
5.4	3.9	1.7	0.4	setosa
4.6	3.4	1.4	0.3	setosa
5	3.4	1.5	0.2	setosa
4.4	2.9	1.4	0.2	setosa
4.9	3.1	1.5	0.1	setosa
5.4	3.7	1.5	0.2	setosa
4.8	3.4	1.6	0.2	setosa
4.8	3	1.4	0.1	setosa
4.3	3	1.1	0.1	setosa
5.8	4	1.2	0.2	setosa
5.7	4.4	1.5	0.4	setosa
5.4	3.9	1.3	0.4	setosa
5.1	3.5	1.4	0.3	setosa
5.7	3.8	1.7	0.3	setosa
5.1	3.8	1.5	0.3	setosa
5.4	3.4	1.7	0.2	setosa
5.1	3.7	1.5	0.4	setosa
4.6	3.6	1	0.2	setosa
5.1	3.3	1.7	0.5	setosa

Nota

El conjunto de datos "Iris" es uno de los conjuntos de datos más conocidos y utilizados en la comunidad de aprendizaje automático y estadísticas. Contiene información sobre tres especies diferentes de iris de flores: Setosa, Versicolor y Virginica. Cada especie está representada por 50 muestras, lo que hace un total de 150 muestras en el conjunto de datos. Para cada muestra, se registran cuatro características:

Longitud del sépalo (sepal length): Longitud del sépalo de la flor en centímetros.

Anchura del sépalo (sepal width): Ancho del sépalo de la flor en centímetros.

Longitud del pétalo (petal length): Longitud del pétalo de la flor en centímetros.

Anchura del pétalo (petal width): Ancho del pétalo de la flor en centímetros.

El objetivo del conjunto de datos Iris es clasificar las tres especies de iris en función de estas cuatro características. Cada muestra se etiqueta con la especie correspondiente.

El conjunto de datos Iris es un conjunto de datos ideal para tareas de clasificación y es ampliamente utilizado para ejemplificar conceptos en aprendizaje automático y estadísticas. Es uno de los conjuntos de datos disponibles en la biblioteca de Python Scikit-Learn y se puede utilizar para probar y entrenar algoritmos de clasificación, como la regresión logística, los árboles de decisión, las máquinas de soporte vectorial y otros.

Puedes cargar y explorar el conjunto de datos Iris en Python utilizando bibliotecas como Scikit-Learn, Pandas y Matplotlib para realizar análisis de datos y visualización.

Ejemplo usando scikit-learn y pandas (Python por supuesto)

```
from sklearn import datasets
import pandas as pd
# Cargar el conjunto de datos Iris
iris = datasets.load_iris()
# Nombres de las características (columnas)
feature_names = iris.feature_names
print("Nombres de características (columnas):\n", feature_names)
# Nombres de las etiquetas de clase
target_names = iris.target_names
print("Nombres de etiquetas de clase:\n", target_names)
# Crear un DataFrame a partir de los datos de Iris
df = pd.DataFrame(data=iris.data, columns=iris.feature_names)
# Obtener las primeras tres filas utilizando el método .head()
primeras_tres_filas = df.head(3)
# Imprimir las primeras tres filas
print(primeras_tres_filas)
```

Resultado de la ejecución

```
Nombres de características (columnas):
 ['sepal length (cm)', 'sepal width (cm)', 'petal length (cm)', 'petal width (cm)']
Nombres de etiquetas de clase:
 ['setosa' 'versicolor' 'virginica']
   sepal length (cm)  sepal width (cm)  petal length (cm)  petal width (cm)
0                5.1               3.5                1.4               0.2
1                4.9               3.0                1.4               0.2
2                4.7               3.2                1.3               0.2
```

Formato JSON

El formato JSON (JavaScript Object Notation) es un formato de intercambio de datos ligero y fácil de leer que se utiliza para representar datos estructurados. Fue inspirado por la notación de objetos literales en JavaScript, pero se ha convertido en un estándar ampliamente utilizado para el intercambio de datos en aplicaciones web y en muchas otras áreas.

Sintaxis básica

- JSON utiliza una sintaxis simple y legible que consiste en pares clave-valor.
- Los pares clave-valor se separan por dos puntos : y se separan entre sí por comas ,.
- Los objetos JSON se encierran entre llaves {} y los arreglos se encierran entre corchetes [].
- Los valores pueden ser cadenas de texto, números, booleanos, objetos JSON anidados, arreglos, null o valores especiales (como true, false, null).

Un objeto JSON:

```
{
    "nombre": "John Doe",
    "edad": 30,
    "ciudad": "Nueva York",
    "activo": true,
    "intereses": ["programación", "música", "viajes"]
}
```

Un arreglo JSON:

```
["manzana", "plátano", "uva", "naranja"]
```

Cadenas de Texto:

Las cadenas de texto en JSON se encierran entre comillas dobles " y pueden contener caracteres especiales escapados (por ejemplo, \" para comillas dobles dentro de una cadena).

Números:

JSON admite números enteros y de punto flotante. No se requieren comillas para los números. Ejemplo: 42 o 3.14.

Booleanos:

JSON admite los valores true y false para representar valores booleanos.

Nulo (null):

JSON admite el valor null para representar la ausencia de un valor.

Anidamiento:

Puedes anidar objetos JSON dentro de otros objetos o arreglos dentro de arreglos para representar estructuras de datos complejas.

Comentarios:

JSON no admite comentarios, a diferencia de algunos otros formatos de datos.

Extensibilidad:

JSON es extensible y permite la creación de estructuras de datos más complejas mediante el uso de objetos y arreglos anidados.

Uso:

JSON se utiliza en una variedad de aplicaciones, como el intercambio de datos entre un servidor y un cliente web, la configuración de aplicaciones, el almacenamiento de datos, y más. También se utiliza en la comunicación de servicios web a través de API REST.

Legibilidad:

JSON se ha diseñado para que sea fácil de leer y escribir para los humanos, lo que lo convierte en una elección popular para la representación de datos en archivos de configuración y comunicación de datos en aplicaciones web.

Soporte en Lenguajes de Programación:

La mayoría de los lenguajes de programación tienen bibliotecas para parsear y generar JSON, lo que facilita la manipulación de datos JSON en aplicaciones.

Al ser estructuras de datos muy genéricas, suelen estar soportadas por la mayoría de los lenguajes de programación modernos. La definición de las estructuras sigue las siguientes condiciones:

Un objeto se delimita por llaves ({ }) y los pares nombre/valor se separan por medio de comas, y entre el nombre y el valor se coloca el carácter dos puntos (:).

Un array se delimita por corchetes ([ ]) y los valores se separan por comas.

Cada valor, tanto dentro de un objeto como de un array, puede ser una cadena de texto delimitada por comillas dobles, un número, un valor booleano (true/false), el valor nulo (null), un objeto o un array.

La representación de los primeros dos registros del ejemplo en CSV sería la siguiente en formato JSON (escriban esa porción de código como continuación del anterior:

```
dos_registros = df.head(2)
# Convertir los registros a formato JSON
json_data = dos_registros.to_json(orient='records')
# Imprimir los registros en formato JSON
print(json_data)
```

Estudiaremos en este curso los modelos orientados a documentos con mongodb. Los documentos de esta base de datos utilizan JSON y se almacenan en formato binario BSON. BSON añade otros tipos de datos al formato JSON.

Resultado:


```
[
{"sepal length (cm)":5.1,"sepal width (cm)":3.5,"petal length (cm)":1.4,"petal width (cm)":0.2},
{"sepal length (cm)":4.9,"sepal width (cm)":3.0,"petal length (cm)":1.4,"petal width (cm)":0.2}
]
```


Adelanto aquí el tema de mongo, solamente para ilustrar documentos en formato JSON de una de las colecciones de la base de datos university en mongo. Como se describe en este documento más adelante, he utilizado esa base de datos relacional para repasar los fundamentos de bases de datos relacionales y el lenguaje SQL.

Indicaré un ejercicio de convertir esa base de datos en una base de datos de documentos en mongo.

La tabla department será una colección en mongo y al leer de esa colección, visualizamos los documentos JSON que contiene.

test> use university;

switched to db university

university> show collections

advisor
classroom
course
department
estudiantes
instructor
prereq
section
student
takes
teaches
time_slot

Realizamos una consulta a la colección (algo así como una tabla) department:

university**> db.department.find()**


```
[
  {
    _id: ObjectId("63d316299ea9b8585b3123f9"),
    dept_name: 'Biology',
    building: 'Watson',
    budget: '90000.00'
  },
  {
    _id: ObjectId("63d316299ea9b8585b3123fa"),
    dept_name: 'Comp. Sci.',
    building: 'Taylor',
    budget: '100000.00'
  },
  {
    _id: ObjectId("63d316299ea9b8585b3123fb"),
    dept_name: 'Elec. Eng.',
    building: 'Taylor',
    budget: '85000.00'
  },
  {
    _id: ObjectId("63d316299ea9b8585b3123fc"),
    dept_name: 'Finance',
    building: 'Painter',
    budget: '120000.00'
  },
  {
    _id: ObjectId("63d316299ea9b8585b3123fd"),
    dept_name: 'History',
    building: 'Painter',
    budget: '50000.00'
  },
  {
    _id: ObjectId("63d316299ea9b8585b3123fe"),
    dept_name: 'Music',
    building: 'Packard',
    budget: '80000.00'
```

Formato XML

El formato XML (Extensible Markup Language) es un lenguaje de marcado que se utiliza para describir datos en un formato estructurado. Fue diseñado para ser legible tanto por humanos como por máquinas, y se utiliza comúnmente en una variedad de aplicaciones, incluyendo el intercambio de datos en la web, el almacenamiento de información, la configuración de archivos y mucho más. A continuación, describiré con detalle los elementos clave del formato XML:

Etiqueta (Tag):

- Una etiqueta es el elemento más básico de XML y se utiliza para marcar o identificar datos.
- Las etiquetas están encerradas en corchetes angulares < y >.
- Hay dos tipos de etiquetas: etiquetas de apertura <etiqueta> y etiquetas de cierre </etiqueta>.
- La etiqueta de cierre tiene una barra diagonal / antes del nombre de la etiqueta, como en </etiqueta>.

Elemento:

- Un elemento XML consta de una etiqueta de apertura, su contenido y una etiqueta de cierre.
- El contenido de un elemento puede ser texto, otros elementos anidados o una combinación de ambos.
- Ejemplo de un elemento simple:

```
<nombre>John</nombre>
```

Atributo:

- Los atributos proporcionan información adicional sobre un elemento y se incluyen en la etiqueta de apertura.
- Los atributos se definen como pares de nombre y valor.
- Se utilizan comillas (simples o dobles) para encerrar el valor del atributo.
- Ejemplo de un elemento con atributos:

```
<persona nombre="John" edad="30" sexo="masculino" />
```

Documento XML:

- Un documento XML comienza con una declaración XML opcional y contiene uno o más elementos.
- La declaración XML generalmente se ve así:

```
<?xml version="1.0" encoding="UTF-8"?>
```

A continuación, se encuentran los elementos del documento.

- Anidamiento:

  XML permite anidar elementos dentro de otros elementos, lo que crea una estructura jerárquica.

  Esto permite representar datos complejos y relaciones entre datos de manera eficaz.

  Ejemplo de anidamiento:

```
<libro>
  <titulo>El Gran Gatsby</titulo>
```

```
  <autor>F. Scott Fitzgerald</autor>
</libro>
```

- Comentarios:

  Los comentarios se pueden agregar en el documento XML para proporcionar explicaciones o notas.

  Se encierran entre <!-- y -->.

- CDATA:

  El CDATA (Character Data) permite incluir contenido de texto sin que se interprete como XML.

  Se usa para datos que pueden contener caracteres reservados de XML sin que se escape.

  Se encierra en <![CDATA[y]]>.

```
<descripcion><![CDATA[Este es un ejemplo de CDATA en XML. & no se interpreta como un carácter especial.]]></descripcion>
```

- Espacio de nombres (Namespaces):

  Los espacios de nombres permiten evitar conflictos de nombres al definir elementos y atributos en documentos XML complejos.

  Se definen utilizando el atributo xmlns en la etiqueta de apertura.

  Ejemplo:

```
<persona xmlns="http://www.ejemplo.com/persona">
  <nombre>John</nombre>
</persona>
```

A continuación, se presenta un fragmento XML del archivo iris.

```
<iris>
  <row>
    <sepal_length>5.1</sepal_length>
    <sepal_width>3.5</sepal_width>
    <petal_length>1.4</petal_length>
    <petal_width>0.2</petal_width>
  </row>
  <row>
    <sepal_length>4.9</sepal_length>
    <sepal_width>3.0</sepal_width>
    <petal_length>1.4</petal_length>
    <petal_width>0.2</petal_width>
  </row>
  <row>
    <sepal_length>4.7</sepal_length>
    <sepal_width>3.2</sepal_width>
    <petal_length>1.3</petal_length>
    <petal_width>0.2</petal_width>
  </row>
</iris>
```

Validación de archivos XML

Recuerda que todo fichero XML cumple con una jerarquía de etiquetas que garantiza su estructura. Para ello, las etiquetas estarán contenidas adecuadamente unas dentro de otras, con su cierre correspondiente. Recuerda también que el fichero XML debe incluir un único elemento raíz que contendrá las otras etiquetas que existan. El valor de los atributos, por su parte, deben estar entre comillas simples o dobles; y en lo que respecta a las etiquetas, estas son sensibles a mayúsculas y minúsculas.

Cuando un XML cumple con estas consideraciones, se dice que este está **«bien formado».** Para que se pueda asegurar que el documento XML es «válido», será necesario validarlo mediante algún otro documento que garantice dicha validez.

Validación de archivos XML mediante DTD

Uno de esos documentos se llama DTD (Document Type Definition o definición de tipo de documento). Este documento recoge las reglas que debe cumplir el contenido del XML (estructura, elementos y atributos que sí puede incluir el fichero), de ahí que sea útil para garantizar la validez de todo fichero XML que haga referencia a él.

Veamos un ejemplo.

El archivo XML

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE biblioteca [
  <!ELEMENT biblioteca (libro+)>
  <!ELEMENT libro (titulo, autor, genero, precio)>
  <!ELEMENT titulo (#PCDATA)>
  <!ELEMENT autor (#PCDATA)>
  <!ELEMENT genero (#PCDATA)>
  <!ELEMENT precio (#PCDATA)>
]>
<biblioteca>
  <libro>
    <titulo>El Gran Gatsby</titulo>
    <autor>F. Scott Fitzgerald</autor>
    <genero>Novela</genero>
    <precio>15.99</precio>
  </libro>
  <libro>
    <titulo>Matar un ruiseñor</titulo>
    <autor>Harper Lee</autor>
    <genero>Novela</genero>
    <precio>12.99</precio>
  </libro>
</biblioteca>
```

En este ejemplo, hemos creado una estructura XML que describe una biblioteca de libros con información sobre el título, el autor, el género y el precio de cada libro. Además, hemos definido una DTD que especifica las reglas de validación para esta estructura.

Código para validar contra DTD

Varias maneras de validación. El siguiente programa valida el archivo XML utilizando Python y la biblioteca lxml. Asegúrate de tener la biblioteca lxml instalada en tu entorno de Python.

```python
from lxml import etree
# Cargar y analizar el archivo XML
xml_file = "libros.xml"
dtd_file = "libros.dtd"  # El archivo DTD

xml_parser = etree.XMLParser(dtd_validation=True)
try:
    tree = etree.parse(xml_file, parser=xml_parser)
    print("El archivo XML es válido.")
except etree.XMLSyntaxError as e:
    print("Error de validación:", e)
```

Validar con un esquema

Para realizar la validación de un archivo XML utilizando un esquema XML (XML Schema Definition, XSD), primero debes definir un esquema XSD que describa la estructura y las restricciones del XML. Luego, puedes usar una biblioteca como lxml en Python para validar el archivo XML con respecto a ese esquema.

El archivo libros.xsd con el esquema

```xml
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">

  <xs:element name="biblioteca">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="libro" maxOccurs="unbounded">
          <xs:complexType>
            <xs:sequence>
              <xs:element name="titulo" type="xs:string"/>
              <xs:element name="autor" type="xs:string"/>
              <xs:element name="genero" type="xs:string"/>
              <xs:element name="precio" type="xs:decimal"/>
            </xs:sequence>
          </xs:complexType>
        </xs:element>
      </xs:sequence>
    </xs:complexType>
  </xs:element>

</xs:schema>
```

El programa para validar contra esquema

```python
from lxml import etree

# Cargar el archivo XSD
xsd_file = "libros.xsd"

# Crear un validador XSD
xmlschema = etree.XMLSchema(etree.parse(xsd_file))

# Cargar y analizar el archivo XML
xml_file = "libros.xml"

try:
    tree = etree.parse(xml_file)
    xmlschema.assertValid(tree)
    print("El archivo XML es válido según el esquema XSD.")
except etree.DocumentInvalid as e:
    print("Error de validación:", e)
```

Bases de datos

Una base de datos es un conjunto de datos persistente utilizado por un sistema de software. Siguiendo con las definiciones, y tal y como se menciona en la bibliografía, un sistema de base de datos es un sistema computarizado para el almacenamiento de registros. Se pueden mencionar varios componentes de un sistema de esta categoría:

- **Datos.** Los datos en un sistema de base de datos pueden definirse como integrados, en aquellos casos en que todos los datos se mantienen unificados y comúnmente serán accedidos por una sola persona, así como compartidos, para aquellos casos en los que se desea mantener los conjuntos de datos separados y otorgar privilegios de acceso distintos a varias personas.
- **Hardware.** Como en otros métodos de almacenamiento, los componentes de hardware que intervienen en un sistema de base de datos son los volúmenes de almacenamiento, así como los procesadores y memoria principal.
- **Software.** La capa de software entre el usuario y la base de datos física se conoce como DBMS (Database Management System o sistema gestor de la base de datos).
- Usuarios. Existen tres clases de usuarios en un sistema de bases de datos:
- **Programadores:** encargados de crear aplicaciones que permitan la interacción con la base de datos.
- **Usuarios finales:** utilizan las distintas aplicaciones y herramientas para interactuar con la base de datos.
- **Administrador de base de datos:** también conocido como DBA por sus siglas en inglés, se encarga de gestionar la estructura, disponibilidad y eficiencia del sistema de base de datos.

En el contexto de bases de datos se utiliza el **término entidad** para describir a cualquier objeto que puede almacenarse en el sistema. Por ejemplo, en una base de datos utilizada por un almacén se puede tener una **entidad producto** para describir los productos disponibles en el almacén y el **término bodega** para describir las bodegas con las que cuenta.

Además, se utilizan los términos vínculo o relación para representar las relaciones entre las entidades. En el ejemplo del almacén, puede haber una relación bodega-producto para indicar que un producto se almacena en una bodega específica.

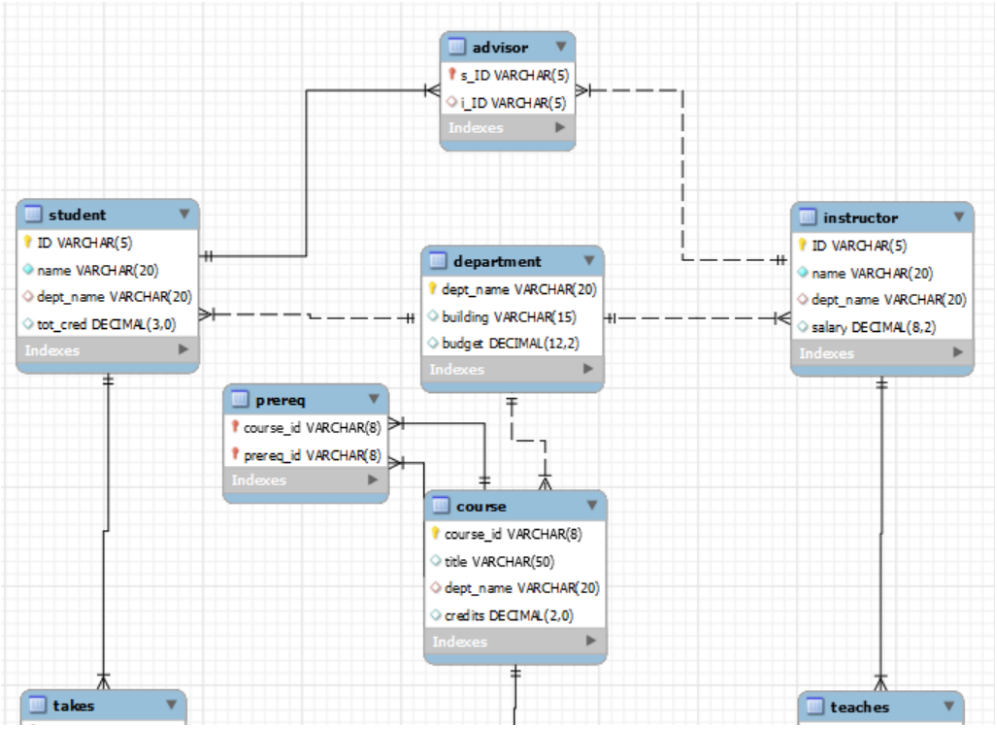
Los datos almacenados en una base de datos se pueden categorizar de forma jerárquica. La unidad más pequeña es el campo, el cual suele tener un tipo (número, fecha, etc.) y la base de datos tendrá muchas ocurrencias.

Al conjunto de datos que tienen relación entre sí se le denomina **registro**. Por ejemplo, un registro de tipo producto puede tener campos como «nombre», «precio» y «descripción». Finalmente, al conjunto de registros del mismo tipo se le denomina archivo almacenado.

Bases de datos relacionales y SQL

En un sistema de base de datos relacional, los archivos de datos son representados por tablas (relaciones en el **álgebra relacional**). Cada columna de la tabla representa un campo del archivo, mientras que cada fila representa un registro de datos. Además, cuando un usuario realiza una operación sobre una tabla, el resultado de dicha operación también será una tabla.

Para ejemplificar estos sistemas utilizaremos una base de datos que simula un sistema escolar en una universidad. Parte de su diagrama:



La relación instructor:

	ID	name	dept_name	salary
1	10101	Srinivasan	Comp. Sci.	65000.00
2	12121	Wu	Finance	90000.00
3	15151	Mozart	Music	40000.00
4	22222	Einstein	Physics	95000.00
5	32343	El Said	History	60000.00
6	33456	Gold	Physics	87000.00
7	45565	Katz	Comp. Sci.	75000.00
8	58583	Califieri	History	62000.00
9	76543	Singh	Finance	80000.00
10	76766	Crick	Biology	72000.00
11	83821	Brandt	Comp. Sci.	92000.00
12	98345	Kim	Elec. Eng.	80000.00

Para interactuar con una base de datos de manera programática, se utiliza el estándar SQL. Los comandos para la manipulación de datos se pueden resumir en cuatro:

- **SELECT**: utilizado para obtener un conjunto de datos a partir de una o varias tablas en un DBMS relacional.
- **INSERT**: comando para agregar un conjunto de registros dentro de una tabla.
- **UPDATE**: permite la modificación de un conjunto de campos sobre un conjunto de registros en una tabla específica.
- **DELETE**: como su nombre indica, permite eliminar un conjunto de registros de una tabla.

En general, las funciones básicas que se pueden llevar a cabo sobre una base de datos se conocen con el acrónimo CRUD, que significa Create, Read, Update and Delete.

Ejemplos de algunas consultas para la base de datos university

Consulta 1: Encuentra todos los cursos que un instructor con un salario superior a $90,000 está enseñando.

```
use university;
SELECT course.title, instructor.name
FROM instructor
JOIN teaches ON instructor.ID = teaches.ID
JOIN course ON teaches.course_id = course.course_id
WHERE instructor.salary > 90000;
```

Resultado:

	title	name
1	Physical Principles	Einstein
2	Game Design	Brandt
3	Game Design	Brandt
4	Image Processing	Brandt

Consulta 2: Muestra el nombre de todos los estudiantes que han tomado el curso "Introducción a la Programación" (course_id = 'CS101').

```
SELECT student.name, takes.course_id
FROM student
JOIN takes ON student.ID = takes.ID
JOIN section ON takes.course_id = section.course_id
where takes.course_id = "CS-101"
```

Resultado:

	name	course_id
1	Zhang	CS-101
2	Shankar	CS-101
3	Levy	CS-101
4	Williams	CS-101
5	Brown	CS-101
6	Bourikas	CS-101
7	Levy	CS-101
8	Zhang	CS-101
9	Shankar	CS-101
10	Levy	CS-101
11	Williams	CS-101
12	Brown	CS-101
13	Bourikas	CS-101
14	Levy	CS-101

Consulta 3: Encuentra el departamento con el presupuesto más alto.

```
SELECT dept_name
FROM department
WHERE budget = (SELECT MAX(budget) FROM department);
```

	dept_name
1	Finance

Después de leer gran parte del tema, el reto siguiente es que repases los aspectos más relevantes de los métodos de captura de información con la clase «Resumen de los métodos de captura de información». También será útil recordar las principales características de las bases de datos relacionales y NoSQL.

Para aquellos que quieran repasar los fundamentos de bases de datos relacionales y el lenguaje SQL, podrán encontrar en el Foro el script DDL y el script DML de la base de datos university.

Al final del tema se presentan también varios casos de estudio que podrán realizar y discutir en el foro de la materia.

- Análisis de logs de un servidor web
- Web scraping de listado de cursos online
- Acceso a transacciones bancarias mediante API
- Productos en formato CSV
- Información geolocalizada en formato JSON
- Información de clientes en una base de datos relacional