Introduction to Data Manipulation using Python

Day 2

Céline Lemarinier

Eviden

Py4SHS 2023

Organization of the lecture

We will begin by:

- Introducing datasets and variables
- Learning about databases and retrieving datasets using SQL queries
- Learning about the PyData ecosystem which allows the manipulation in RAM of datasets.

Bibliography

- *Learning SQL: Generate, Manipulate, and Retrieve Data*, 3rd Edition, Alan Beaulieu, March 2020, O'Reilly Media, Inc.
- *Python for Data Analysis: Data Wrangling with pandas, NumPy, and Jupyter*, Wes McKinney, 2022, O'Reilly Media, Inc.
- W3C tutorial on SQL : `https://www.w3schools.com/sql/`.
- Pandas documentation: `https://pandas.pydata.org/`.
- Seaborn documentation: `https://seaborn.pydata.org/`.

Introduction to datasets and variables

Outline

Datasets

Datasets

A **dataset*** can be thought of as a matrix $M = (x_{i,j})_{1 \leq i \leq n, 1 \leq j \leq m}$ with n the number of individuals in the population and m the number of variables.

Datasets

Datasets

A **dataset*** can be thought of as a matrix $M = (x_{i,j})_{1 \leq i \leq n, 1 \leq j \leq m}$ with n the number of individuals in the population and m the number of variables.

Columns of a table represents a **particular variable** (also called **feature**), and each row corresponds to a given **record** of the data set in question for an **individual**.

Datasets

Example:

Individual	Variable 1	Variable 2	Variable 3
ID1	5	4	1
ID2	2	3	1

Question:
Give the value for:
$x_{1,3} =$
$x_{2,1} =$
Variable 1 for individual 1
All data regarding individual 2

Example of dataset

The Iris dataset was introduced by the British statistician and biologist Ronald Fisher in his 1936 paper *The use of multiple measurements in taxonomic problems*.

ID	Sepal length	Sepal width	Petal length	Specie
1	2.1	3.1	4.1	Setosa
2	3.1	1.1	2.1	Setosa
3	4.1	5.1	3.1	Versicolor
4	1.1	2.1	2.1	Virginica

Example of dataset

ID	Sepal length	Sepal width	Petal length	Specie
1	2.1	3.1	4.1	Setosa
2	3.1	1.1	2.1	Setosa
3	4.1	5.1	3.1	Versicolor
4	1.1	2.1	2.1	Virginica

The names of the variables are:
There are _____ individuals.
There are _____ variables.

Variable types

Question

Can anyone list the different types of variables that can be encountered in datasets ?

Variable types

Let's consider a dataset $M = (x_{i,j})_{1 \leq n, 1 \leq m}$, with n individuals and m variables.

Variable types

Let's consider a dataset $M = (x_{i,j})_{1 \leq n, 1 \leq m}$, with n individuals and m variables.
A variable j can be:

- **Numeric**: $(x_{i,j})_{1 \leq i \leq n} \in \mathbb{R}^n$.
  Example: **Petal width**.

Variable types

Let's consider a dataset $M = (x_{i,j})_{1 \leq n, 1 \leq m}$, with n individuals and m variables.
A variable j can be:

- **Numeric**: $(x_{i,j})_{1 \leq i \leq n} \in \mathbb{R}^n$.
  Example: **Petal width**.
- **Categorical**: $(x_{i,j})_{1 \leq i \leq n} \in \mathcal{X}^n$, with $\mathcal{X}$ a set of distinct values.
  A special case of categorical variables often encountered .
  Example: **Flower specie**.

Variable types

Let's consider a dataset $M = (x_{i,j})_{1 \leq n, 1 \leq m}$, with n individuals and m variables.
A variable j can be:

- **Numeric**: $(x_{i,j})_{1 \leq i \leq n} \in \mathbb{R}^n$.
  Example: **Petal width**.
- **Categorical**: $(x_{i,j})_{1 \leq i \leq n} \in \mathcal{X}^n$, with $\mathcal{X}$ a set of distinct values.
  A special case of categorical variables often encountered .
  Example: **Flower specie**.
- **Ordinal**: $(x_{i,j})_{1 \leq i \leq n} \in \mathcal{X}^n$, with $\mathcal{X}$ a set of **ordered** distinct values.
  Example: **Performance (low, medium, high)**.

Dataset analysis

To **analyze a dataset**, you can perform:

- A **visual*** analysis: use graphs to better understand the dataset.
- A **statistical*** analysis: use statistical estimators to better understand the dataset.

Dataset analysis

To **analyze a dataset**, you can perform:

- A **visual*** analysis: use graphs to better understand the dataset.
- A **statistical*** analysis: use statistical estimators to better understand the dataset.

Analysis depends on the variable type !
A poor analysis of variables can cause misinterpretation of data.

Dataset analysis

Question

Can anyone give me:

- Possible **graphical representation** of **numeric** and **categorical** variables ?

Dataset analysis

Question

Can anyone give me:

- Possible **graphical representation** of **numeric** and **categorical** variables ?
- Possible **estimators** of **numeric** and **categorical variables** ?

Dataset analysis

Question

Can anyone give me:

- Possible **graphical representation** of **numeric** and **categorical** variables ?
- Possible **estimators** of **numeric** and **categorical variables** ?

ID	Sepal length	Sepal width	Petal length	Specie
1	2.1	3.1	4.1	Setosa
2	3.1	1.1	2.1	Setosa
3	4.1	5.1	3.1	Versicolor
4	1.1	2.1	2.1	Virginica

Analyzing numeric variables

Usual indicators include:

- **Arithmetical mean**: summarize to better understand the overall value.
  $\bar{X} = \frac{1}{N} \sum_{i=1}^{N} x_i$

Analyzing numeric variables

Usual indicators include:

- **Arithmetical mean**: summarize to better understand the overall value.
  $\bar{X} = \frac{1}{N} \sum_{i=1}^{N} x_i$
- **Variance and standard error**: measures the **dispersion of the data** compared to the mean.
  $\text{var}(X) = \frac{1}{N} \sum_{i=1}^{N} (x_i - \bar{X})^2$
  $\sigma(X) = \sqrt{\text{var}(X)}$

Analyzing numeric variables

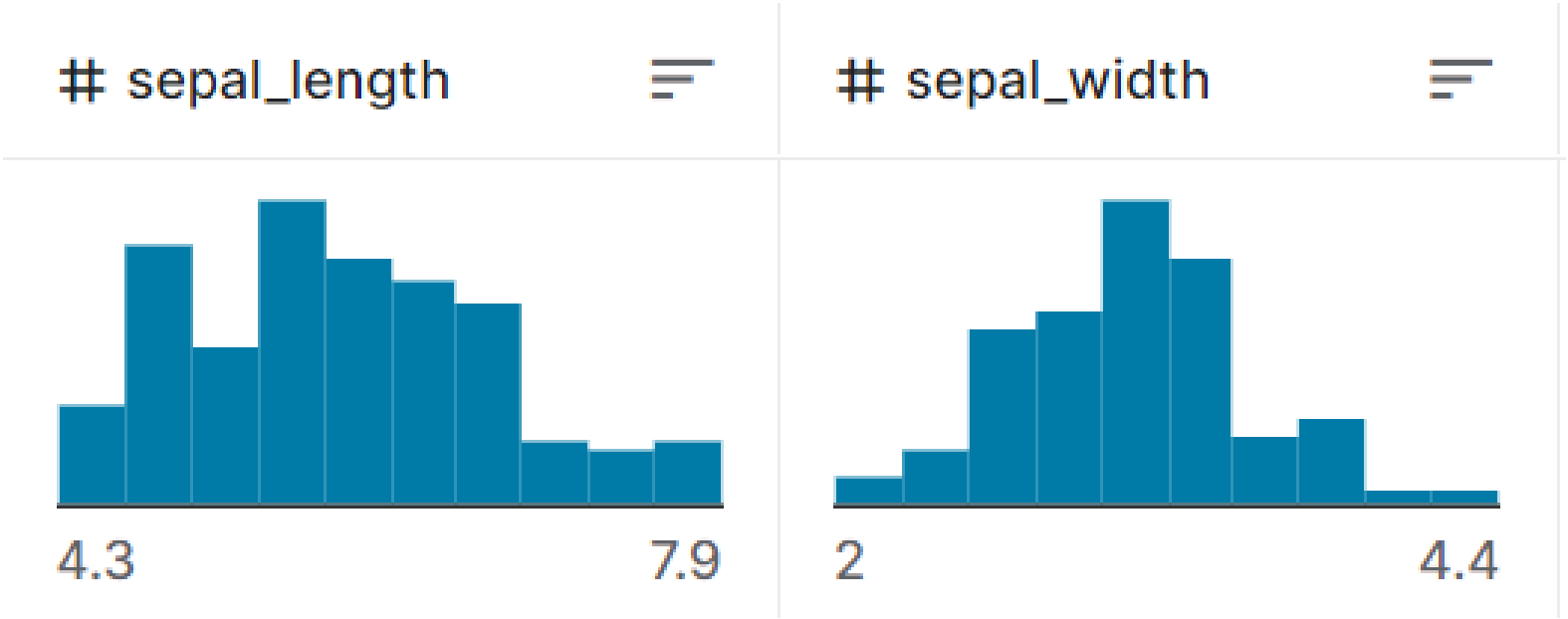
Usual indicators include:

- **Arithmetical mean**: summarize to better understand the overall value.
  $\bar{X} = \frac{1}{N} \sum_{i=1}^{N} x_i$
- **Variance and standard error**: measures the **dispersion of the data** compared to the mean.
  $\text{var}(X) = \frac{1}{N} \sum_{i=1}^{N} (x_i - \bar{X})^2$
  $\sigma(X) = \sqrt{\text{var}(X)}$
- **Quantiles**: divide the ordered vectors into equal parts of same 1/4 quantiles, median
  Very useful for datasets with a lot of outliers*!

Representing numeric variables: histograms

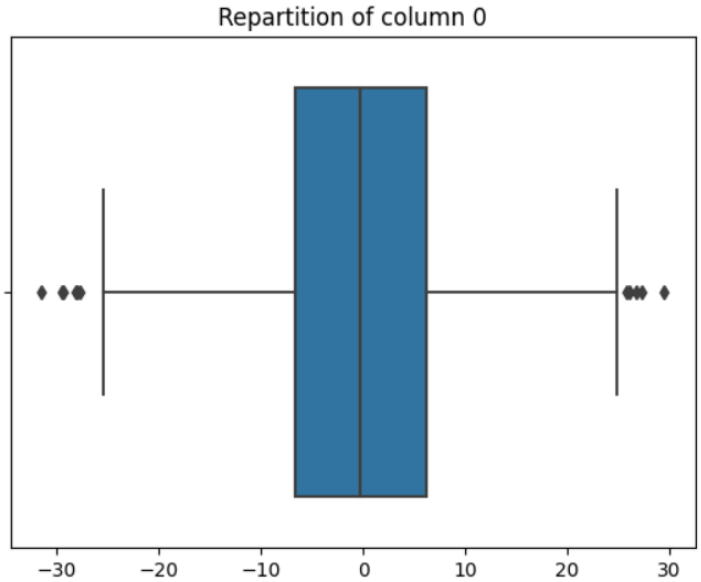
Histograms* consist in:

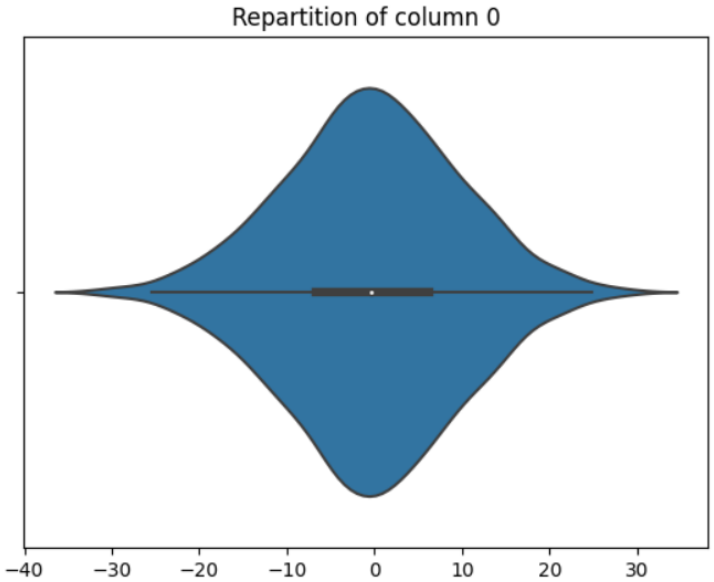
- Dividing the numerical space into intervals of regular length
- Computing the frequency of values per interval



Representing numeric variables: boxplots

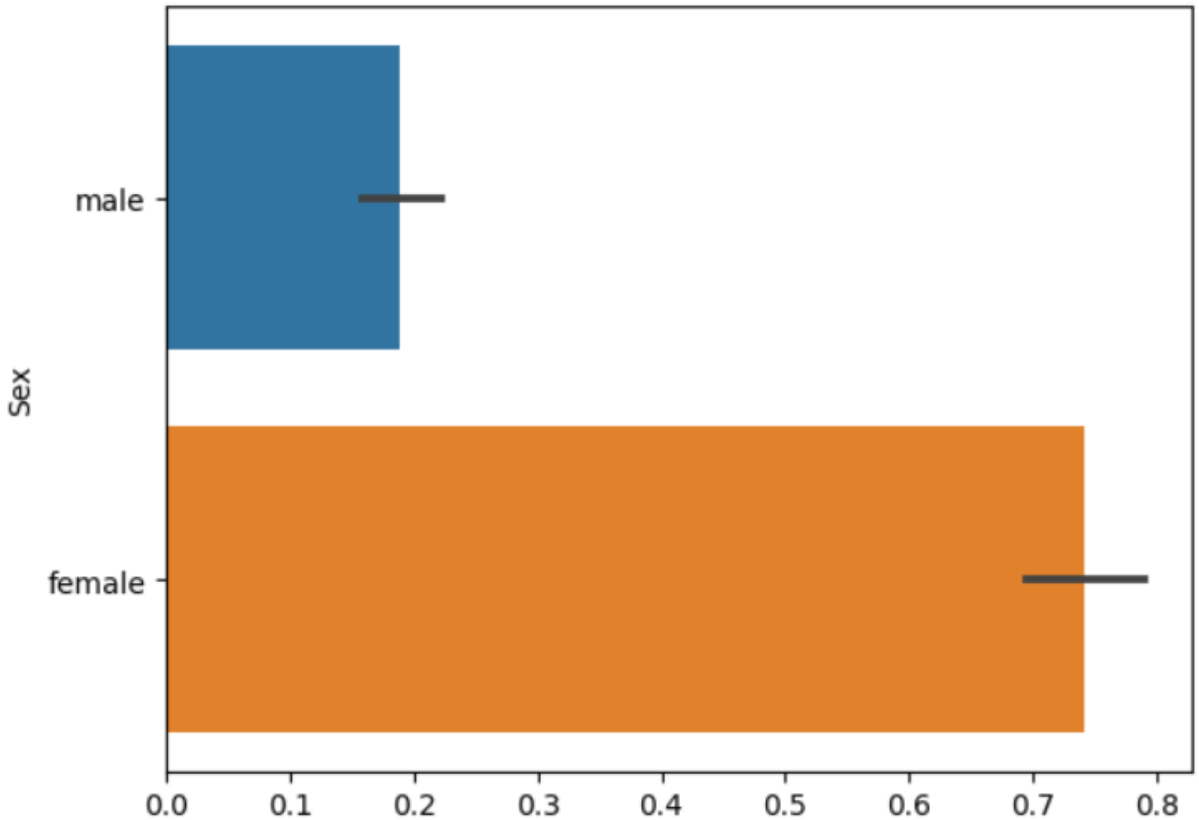
Boxplots* and **violin plots*** consist in representing all the values of the variables and their statistical indicators (usually, quantiles and medians).





Analyzing and representing categorical variables

Categorical variables are often **harder** to study.
Usual indicators are **counts** and **frequency**.
Usual graphical representation can be **bar graphs**.



Questions

Questions ?

Retrieving datasets: introduction to databases

Outline

Retrieving datasets

In order to manipulate the data as presented in the first section, **you need to access this data**.

Retrieving datasets

In order to manipulate the data as presented in the first section, **you need to access this data**.
Data can be available:

Retrieving datasets

In order to manipulate the data as presented in the first section, **you need to access this data**.
Data can be available:

- As a flat file (such as a CSV or an Excel file)

Retrieving datasets

In order to manipulate the data as presented in the first section, **you need to access this data**.
Data can be available:

- As a flat file (such as a CSV or an Excel file)
- Through an API (for example HTTP)

Retrieving datasets

In order to manipulate the data as presented in the first section, **you need to access this data**.
Data can be available:

- As a flat file (such as a CSV or an Excel file)
- Through an API (for example HTTP)
- Through a database.

Retrieving datasets

In order to manipulate the data as presented in the first section, **you need to access this data**.
Data can be available:

- As a flat file (such as a CSV or an Excel file)
- Through an API (for example HTTP)
- Through a database.

Today, we will learn how to manipulate single flat files and more complex data stored in a SQL database using Python.

What is a database ?

Database

A **database*** is an **organized** collection of structured information, or data. This database is usually controlled by a database management system (DBMS).

What is a database ?

Database

A **database*** is an **organized** collection of structured information, or data. This database is usually controlled by a database management system (DBMS).

The data and the DBMS, along with the applications that are associated with them, are referred to as a database system, often shortened simply **to database**.

What is a database ?

- Data within the most common types of databases in operation today is typically modeled in **rows and columns** in a series of tables to make processing and data querying efficient.
- The data can then be easily accessed, managed, modified, updated, controlled, and organized.

What is a database ?

- Data within the most common types of databases in operation today is typically modeled in **rows and columns** in a series of tables to make processing and data querying efficient.
- The data can then be easily accessed, managed, modified, updated, controlled, and organized.

Can you remind me the link between rows, columns, individual, features and variables seen in the first part ?

Relational databases

Relational databases

Relational databases* present the data to the user as **relations** (a presentation in tabular form) and provide relational operators to manipulate the data in tabular from.

Relational databases

Relational databases

Relational databases* present the data to the user as **relations** (a presentation in tabular form) and provide relational operators to manipulate the data in tabular from.

- Relational databases became dominant in the 1980s.

Relational databases

Relational databases

Relational databases* present the data to the user as **relations** (a presentation in tabular form) and provide relational operators to manipulate the data in tabular from.

- Relational databases became dominant in the 1980s.
- Items in a relational database are organized as a set of tables with columns and rows.

Relational databases

Relational databases

Relational databases* present the data to the user as **relations** (a presentation in tabular form) and provide relational operators to manipulate the data in tabular from.

- Relational databases became dominant in the 1980s.
- Items in a relational database are organized as a set of tables with columns and rows.
- Relational database technology provides the most efficient and flexible way to access structured information.

Relational databases

Relational databases

Relational databases* present the data to the user as **relations** (a presentation in tabular form) and provide relational operators to manipulate the data in tabular from.

- Relational databases became dominant in the 1980s.
- Items in a relational database are organized as a set of tables with columns and rows.
- Relational database technology provides the most efficient and flexible way to access structured information.
- Most relational databases use structured query language (SQL) for writing and querying data.

Examples of tables

This is the kind of data you could find in a Veterinary Clinic.

Table name : **Patients**

Pat_ID	Name	Species	type	age	gender
1	Mittens	cat	Tabby	3	male
2	Loba	dog	Greyhound	10	female
3	Coco	parrot	Cockatoo	2	female
4	Ben	dog	Poodle	5	male

Table name : **Procedures**

Proc_ID	Pat_ID	procedure	date
1	1	neutering	20 April 2021
2	2	fix broken leg	05 June 2021
3	3	vaccination	10 July 2021
4	1	kidney stone removal	12 March 2022

Keys

SQL Keys

Keys are fields in a table that are used to identify specific row(s) and to find or **create relationship between tables**.

Keys allow to:

- Create **relationships** between two tables.
- Keep data **consistent** and valid in the database.
- Maintain **uniqueness** in a table

SQL supports various types of keys:
Primary Key, Candidate Key, Unique Key, Composite Key, Super Key, Alternate Key, **Foreign Key**

Primary Keys

A **Primary Key** is a key selected to identify each record **uniquely** in a table. Columns marked as primary keys **aren't allowed to have null values**. It keeps unique values throughout the column.

Creating a table with a primary key

```
CREATE TABLE Patients(
  Pat_ID int not null PRIMARY KEY,
  Name varchar(200) not null,
  Species varchar(200) not null,
  Type varchar(200) not null,
  Age int,
  Gender varchar(200),
)
```

Foreign Keys

A **Foreign Key** is a column in a table that is used as the Primary key in another table.
It can accept **multiple nulls and duplicate values**.

creating a table with a foreign key

```
CREATE TABLE Procedures (
  Proc_ID int primary key not null,
  Pat_ID int FOREIGN KEY REFERENCES Patients(Pat_ID)
  Name varchar(100),
  date DATE
)
```

Examples of primary and foreign key

Can you identify what is the primary key and what could be a foreign key in the following table ?

Table name : Patients

Pat_ID	Name	Species	type	age	gender
1	Mittens	cat	Tabby	3	male
2	Loba	dog	Greyhound	10	female
3	Coco	parrot	Cockatoo	2	female
4	Ben	dog	Poodle	5	male

Table name : Procedures

Proc_ID	Pat_ID	procedure	date
1	1	neutering	20 April 2021
2	2	fix broken leg	05 June 2021
3	3	vaccination	10 July 2021
4	1	kidney stone removal	12 March 2022

Example of primary and foreign key

Solution

In the Procedures table, the **Pat_ID** column identifies as the foreign key, because it is the primary key in the Patients table and **it relates each Procedure to a Patient** (Each Procedure must be linked to a Patient).

Writing SQL queries

Structured Query Language

Structured Query Language (SQL)* lets you access and manipulate data stored with relational databases.

Writing SQL queries

Structured Query Language

Structured Query Language (SQL)* lets you access and manipulate data stored with relational databases.

What do we use SQL for?

- SQL can retrieve data from a database
- SQL can insert records in a database
- SQL can update records in a database
- SQL can delete records from a database

Writing SQL queries

Structured Query Language

Structured Query Language (SQL)* lets you access and manipulate data stored with relational databases.

What do we use SQL for?

- SQL can retrieve data from a database
- SQL can insert records in a database
- SQL can update records in a database
- SQL can delete records from a database

It became a standard of the American National Standards Institute (ANSI) in 1986, and of the International Organization for Standardization (ISO) in 1987.

SQL can also:

- Create new databases
- Create new tables in a database
- Create stored procedures in a database
- Create views in a database
- Set permissions on tables, procedures and views
- ...

SQL queries : SELECT

SELECT statement

The **SELECT statement*** is used to select data from a table within a database.

The data returned is stored in a result table, called the result-set.

SELECT ***column(s)*** FROM ***table*** WHERE ***filter***

where:

- **column** is a set of columns in the database
- **table** is the table containing these columns
- **filter** is a logical filter.

SQL queries : SELECT example

Table name : **patients**

Pat_ID	Name	Species	type	age	gender
1	Mittens	cat	Tabby	3	male
2	Loba	dog	Greyhound	10	female
3	Coco	parrot	Cockatoo	2	female
4	Ben	dog	Poodle	5	male

Question

```
SELECT * FROM patients WHERE age > 4
```

- Identify the columns, the table and the filter in the query.
- What view will this query return ?

SQL queries : SELECT answer

Question

What view will this table return ?

```
SELECT * FROM patients WHERE age > 4
```

Pat_ID	Name	Species	type	age	gender
2	Loba	dog	Greyhound	10	female
4	Ben	dog	Poodle	5	male

SELECT specific columns

Question

What view will this query return ?

```
SELECT name FROM patients WHERE age > 4
```

SELECT specific columns

Question

What view will this query return ?

```
SELECT name FROM patients WHERE age > 4
```

Name
Loba
Ben

Joining Tables

Sometimes you need to access data from several tables.
For example, you need the information about the **age** and the **procedures** for one individual.

JOIN clause

A **JOIN clause*** is used to combine rows from two or more tables, based on a **related column between them** usually a key.

Joining Tables

Sometimes you need to access data from several tables.
For example, you need the information about the **age** and the **procedures** for one individual.

JOIN clause

A **JOIN clause*** is used to combine rows from two or more tables, based on a **related column between them** usually a key.

```
SELECT patients.name, patients.age, procedures.name
FROM patients
INNER JOIN procedures
ON patients.Pat_ID = procedures.Pat_ID
```

Joining Tables

Table name : **patients**

Pat_ID	Name	Species	type	age	gender
1	Mittens	cat	Tabby	3	male
2	Loba	dog	Greyhound	10	female
3	Coco	parrot	Cockatoo	2	female
4	Ben	dog	Poodle	5	male

Table name : **procedures**

Proc_ID	Pat_ID	name	date
1	1	neutering	20 April 2021
2	2	fix broken leg	05 June 2021
3	3	vaccination	10 July 2021
4	1	kidney stone removal	12 March 2022

Joining Tables

Query result

What is the result of the following query on the previous tables:

```
SELECT patients.name, patients.age, procedures.name
FROM patients
INNER JOIN procedures
ON patients.Pat_ID = procedures.Pat_ID
```

Joining Tables

Query result

What is the result of the following query on the previous tables:

```
SELECT patients.name, patients.age, procedures.name
FROM patients
INNER JOIN procedures
ON patients.Pat_ID = procedures.Pat_ID
```

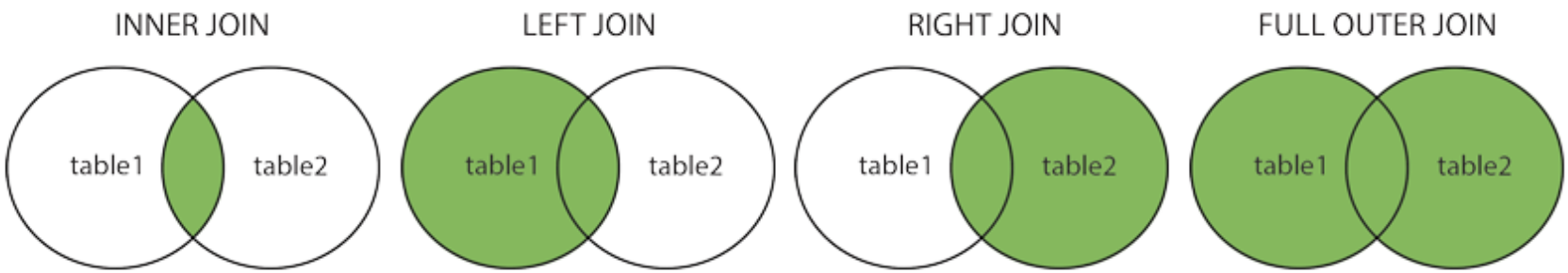
patients.name	patients.age	procedures.name
Mittens	3	neutering
Mittens	3	kidney stone removal
Loba	10	fix broken leg
Coco	2	vaccination

Different possible JOINs

Different Types of SQL JOINs

Here are the different types of the JOINs in SQL:

- `(INNER) JOIN`: Returns records that have matching values in both tables
- `LEFT (OUTER) JOIN`: Returns all records from the left table, and the matched records from the right table
- `RIGHT (OUTER) JOIN`: Returns all records from the right table, and the matched records from the left table
- `FULL (OUTER) JOIN`: Returns all records when there is a match in either left or right table



source: w3schools

Practice

question

What would be the result of that query with:

- A LEFT JOIN
- A RIGHT JOIN
- An OUTER JOIN

Solution

LEFT JOIN

patients.name	patients.age	procedures.name
Mittens	3	neutering
Mittens	3	kidney stone removal
Loba	10	fix broken leg
Coco	2	vaccination
Ben	5	null

Solution

RIGHT JOIN

patients.name	patients.age	procedures.name
Mittens	3	neutering
Mittens	3	kidney stone removal
Loba	10	fix broken leg
Coco	2	vaccination

Solution

OUTER JOIN

patients.name	patients.age	procedures.name
Mittens	3	neutering
Mittens	3	kidney stone removal
Loba	10	fix broken leg
Coco	2	vaccination
Ben	5	null

GROUP BY

GROUP BY statement

The **GROUP BY** statement groups rows that have the same values into **summary rows**, like "find the number of procedures for each animal".

GROUP BY

GROUP BY statement

The **GROUP BY** statement groups rows that have the same values into **summary rows**, like "find the number of procedures for each animal".

The GROUP BY statement is often used with **aggregate functions** (COUNT(), MAX(), MIN(), SUM(), AVG()) to group the result-set by one or more columns.

GROUP BY

GROUP BY statement

The **GROUP BY** statement groups rows that have the same values into **summary rows**, like "find the number of procedures for each animal".

The GROUP BY statement is often used with **aggregate functions** (COUNT(), MAX(), MIN(), SUM(), AVG()) to group the result-set by one or more columns.

```
SELECT column_name(s)
FROM table_name
WHERE condition
GROUP BY column_name(s)
ORDER BY column_name(s) DESC
```

GROUP BY query

```
SELECT patients.name, count(procedures.name)
FROM patients
LEFT JOIN procedures
ON Patients.Pat_ID=Procedures.Pat_ID
GROUP BY patients.name
ORDER BY count(procedures.name) desc
```

GROUP BY query

```
SELECT patients.name, count(procedures.name)
FROM patients
LEFT JOIN procedures
ON Patients.Pat_ID=Procedures.Pat_ID
GROUP BY patients.name
ORDER BY count(procedures.name) desc
```

patients.name	count(procedures.name)
Mittens	2
Loba	1
Coco	1
Ben	0

NoSQL databases

NoSQL databases

A **NoSQL*** (Not Only SQL), or non-relational database, allows unstructured and semi-structured data to be stored and manipulated (in contrast to a relational database, which defines how all data inserted into a table must have the same columns).

NoSQL databases grew popular as applications and data became more common and more complex: there are particularly well-suited for use cases like big data, real-time analytics, and web applications.

Types of NoSQL Databases

- **Document Stores**: Stores data in a document format (e.g., JSON or XML). Examples: MongoDB and Couchbase.
- **Key-Value Stores**: Stores data as key-value pairs. Examples: Redis and Amazon DynamoDB.
- **Column-Family Stores**: Data is organized into columns and column families rather than rows and tables. Examples: Apache Cassandra and HBase.
- **Graph Databases**: Designed for storing and querying graph-like data structures. Examples: Neo4j and Amazon Neptune.

Example of NoSQL data format: MongoDB Collections

MongoDB

MongoDB is a widely used open-source NoSQL database, which falls under the category of document-oriented databases. It stores data in a **flexible, JSON-like documents with dynamic schemas**.

For example, we could have a collection named "animals".
One document in this collection would be the equivalent of an individual.

Examples of collections

Data is stored into a JSON like format, the previously seen SQL tables would look like this:

```
{
    "_id": {"$oid":"456gf5465"},
    "name": "Mittens",
    "Species": "cat",
    "type": "Tabby",
    "age": 3,
    "gender": "male",
    "procedures": [
        {
            name: "neutering",
            date: "20 April 2021"
        },
    ]
}
```

Example of collections

Question

How would you store the different procedures of the patients using SQL and No SQL ? Can you list the weaknesses and strengths of each approach ?

Advantages of using SQL

- **Well-Defined Schema:** SQL databases enforce a structured schema, ensuring data consistency and integrity.

Advantages of using SQL

- **Well-Defined Schema:** SQL databases enforce a structured schema, ensuring data consistency and integrity.
- **ACID Transactions**: Support for Atomicity, Consistency, Isolation, and Durability ensures reliable data management.

Advantages of using SQL

- **Well-Defined Schema:** SQL databases enforce a structured schema, ensuring data consistency and integrity.
- **ACID Transactions**: Support for Atomicity, Consistency, Isolation, and Durability ensures reliable data management.
- **Mature Query Language**: SQL provides a standardized language for complex queries and aggregations.

Advantages of using SQL

- **Well-Defined Schema:** SQL databases enforce a structured schema, ensuring data consistency and integrity.
- **ACID Transactions**: Support for Atomicity, Consistency, Isolation, and Durability ensures reliable data management.
- **Mature Query Language**: SQL provides a standardized language for complex queries and aggregations.
- **Joins**: Ability to perform efficient joins across multiple tables, aiding complex data retrieval.

Advantages of using SQL

- **Well-Defined Schema:** SQL databases enforce a structured schema, ensuring data consistency and integrity.
- **ACID Transactions**: Support for Atomicity, Consistency, Isolation, and Durability ensures reliable data management.
- **Mature Query Language**: SQL provides a standardized language for complex queries and aggregations.
- **Joins**: Ability to perform efficient joins across multiple tables, aiding complex data retrieval.
- **Data Integrity**: Foreign key constraints maintain relationships between tables.

Advantages of NoSQL

- **Flexible Schema**: NoSQL databases allow dynamic and schema-less data structures, accommodating evolving data needs.

Advantages of NoSQL

- **Flexible Schema**: NoSQL databases allow dynamic and schema-less data structures, accommodating evolving data needs.
- **Scalability**: Designed for horizontal scalability, enabling handling of massive data volumes and high traffic.

Advantages of NoSQL

- **Flexible Schema**: NoSQL databases allow dynamic and schema-less data structures, accommodating evolving data needs.
- **Scalability**: Designed for horizontal scalability, enabling handling of massive data volumes and high traffic.
- **High Performance**: NoSQL databases excel at read and write operations, especially for specific use cases.

Advantages of NoSQL

- **Flexible Schema**: NoSQL databases allow dynamic and schema-less data structures, accommodating evolving data needs.
- **Scalability**: Designed for horizontal scalability, enabling handling of massive data volumes and high traffic.
- **High Performance**: NoSQL databases excel at read and write operations, especially for specific use cases.
- **Unstructured Data**: Ideal for managing unstructured or semi-structured data like documents, multimedia, and user-generated content.

Advantages of NoSQL

- **Flexible Schema**: NoSQL databases allow dynamic and schema-less data structures, accommodating evolving data needs.
- **Scalability**: Designed for horizontal scalability, enabling handling of massive data volumes and high traffic.
- **High Performance**: NoSQL databases excel at read and write operations, especially for specific use cases.
- **Unstructured Data**: Ideal for managing unstructured or semi-structured data like documents, multimedia, and user-generated content.
- **Rapid Development**: NoSQL databases simplify development by eliminating the need for complex schema migrations.

When to choose SQL or NoSQL

SQL is great for:

- **Well-Defined Data**: Use SQL for structured, highly relational data where consistency is critical.
- **Complex Queries**: SQL's rich query language is advantageous for analytical and reporting tasks.
- **Transactions**: When data integrity and transaction management are of utmost importance.

When to choose SQL or NoSQL

SQL is great for:

- **Well-Defined Data**: Use SQL for structured, highly relational data where consistency is critical.
- **Complex Queries**: SQL's rich query language is advantageous for analytical and reporting tasks.
- **Transactions**: When data integrity and transaction management are of utmost importance.

NoSQL is great for:

- **Unstructured Data**: Opt for NoSQL to manage diverse and unstructured data types effectively.
- **Scalability**: Choose NoSQL when dealing with massive amounts of data or high traffic loads.
- **Rapid development**: as schema is not required, development can be started faster (but this can easily backfire in complex project !)

Questions regarding the database section ?

The PyData ecosystem

Outline

Manipulating dataset

After dataset retrieval (either from a database or a flat file), you want to:

Manipulating dataset

After dataset retrieval (either from a database or a flat file), you want to:

- Manipulate this data to suit your need

Manipulating dataset

After dataset retrieval (either from a database or a flat file), you want to:

- Manipulate this data to suit your need
- Perform analysis to better understand this data

Manipulating dataset

After dataset retrieval (either from a database or a flat file), you want to:

- Manipulate this data to suit your need
- Perform analysis to better understand this data
- Display and share this analysis (reports, web applications ...)

Manipulating dataset

After dataset retrieval (either from a database or a flat file), you want to:

- Manipulate this data to suit your need
- Perform analysis to better understand this data
- Display and share this analysis (reports, web applications ...)

To do this, Python is very well-suited and provides powerful tools to do so, called the **PyData ecosystem**.

The PyData Ecosystem

The PyData ecosystem

The PyData ecosystem* is a collection of open-source libraries and tools built around the Python programming language for data analysis, visualization, and machine learning.

The PyData Ecosystem

The PyData ecosystem

The PyData ecosystem* is a collection of open-source libraries and tools built around the Python programming language for data analysis, visualization, and machine learning.

It provides a comprehensive suite of packages for every step of the data science pipeline.

The PyData Ecosystem

The PyData ecosystem

The PyData ecosystem* is a collection of open-source libraries and tools built around the Python programming language for data analysis, visualization, and machine learning.

It provides a comprehensive suite of packages for every step of the data science pipeline.

Question

Can you remind me from yesterday's lecture what is the syntax to install a package in Python ?

Core Libraries

- **NumPy**: Core library for numerical computations with multidimensional arrays.
- **pandas**: Provides data structures like DataFrames and Series for data manipulation and analysis.
- **matplotlib**: A powerful plotting library for creating static, interactive, and animated visualizations.

Packages for Data Visualization

- **Seaborn**: Built on top of matplotlib, it provides a high-level interface for creating attractive statistical graphics.
- **Plotly**: Offers interactive and web-based visualizations for data exploration and communication.

Because we cannot learn everything in a week, we will focus this afternoon on using **pandas** and **seaborn**.

Packages for Machine Learning

- **scikit-learn**: Comprehensive library for traditional machine learning algorithms and tools.
- **Keras**/**TensorFlow** and **PyTorch**: Deep learning frameworks for building and training neural networks.

Tomorrow, we will use Scikit-learn for our lab session!

Afternoon lab

Outline

Goal

The goal of this lab is to learn **how to manipulate and plot data using**:

- SQL queries
- Pandas

We will use as a practice dataset a database containing different catalogued ship wrecks, but feel free to use your own data !

Pandas core component

Pandas

Pandas is a Python package for **easy data manipulation**, which provides support for reading and writing data in various formats (CSV, SQL, XML ...)

Pandas provides 2 main objects:

- **DataFrame**: A table-like data structure (similar to the datasets we saw at the beginning of the lecture);
- **Series**: A one-dimensional labeled array (you can think of it as a column in a DataFrame).

Creating DataFrames

- DataFrames can be created from various data sources:
  - From dictionaries
  - From lists of dictionaries or tuples
  - From CSV, Excel, SQL databases, XML ...

Creating a DataFrame from a dictionary

```
import pandas as pd
data = {'Name': ['Alice', 'Bob', 'Charlie'], 'Age':
[25, 30, 22]}
df = pd.DataFrame(data)
```

Data Exploration

- **head()**: Display the first few rows of a DataFrame. `df.head()`
- **shape**: Get the dimensions (rows, columns) of the DataFrame. `df.shape`
- **describe()**: Generate summary statistics for numerical columns. `df.describe()`
- **value_counts()**: Count occurrences of unique values in a column. `df['Age'].value_counts()`
- **groupby()**: Perform grouped operations and aggregations. `df.groupby('Age').mean()`

Data Manipulation

- **Selection**: Indexing and selecting data using labels, positions, or conditions. `df['Name']`
  `df.loc[0]`
  `df[df['Age'] > 25]`
- **Filtering**: Applying conditions to filter rows or columns.
  `df[df['Age'] > 25]`
- **Sorting**: Sorting data based on column values.
  `df.sort_values(by='Age')`
- **Adding and Dropping**: Adding or dropping columns or rows.
  `df['Gender'] = ['F', 'M', 'M']`
  `df.drop('Gender', axis=1, inplace=True)`
- **Aggregation**: Performing group-wise operations and aggregations. `df.groupby('Gender')['Age'].mean()`

Introduction to Seaborn

Seaborn

Seaborn is a Python data visualization library relying on pandas, which simplifies the process of creating various types of plots with minimal code..

Key Features of Seaborn

- Seaborn provides a wide range of statistical plots:
  - Scatter plots
  - Line plots
  - Histograms
  - Bar plots
  - Box plots
  - Heatmaps
  - Pair plots

Question

Can you remind me to what variable type each plot corresponds ?

Basic Scatter Plot

Scatter plots show the relationship between two variables.

```
import seaborn as sns
import matplotlib.pyplot as plt

# Create a DataFrame
data = sns.load_dataset("iris")

# Create a scatter plot
sns.scatterplot(x="sepal_length",
                y="sepal_width",
                data=data)
plt.show()
```

Bar Plot

Bar plots display the distribution of a categorical variable.

```
# Create a bar plot with error bars
sns.barplot(x="species",
            y="petal_length",
            data=data)
plt.show()
```

Customizing Plots

- Seaborn allows customization of plots for better communication:
  - Titles and labels: `plt.title()`, `plt.xlabel()`, `plt.ylabel()`
  - Color palettes: `sns.set_palette()`
  - Themes: `sns.set_theme()`
  - Grid and background: `sns.grid()`
  - Plot styles: `sns.set_style()`