Introduction to Data Manipulation using Python

Day 2

Céline Lemarinier

Eviden

Py4SHS 2023

Organization of the lecture

We will begin by:

- Introducing datasets, variables and statistical estimators;
- Learning about databases and REST API;
- 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** (or **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.

Variables and features

Variable

A **variable** is a **measurement** computed on **an individual**.

Example: Weight of individual.

Variables and features

Variable

A **variable** is a **measurement** computed on **an individual**.

Example: Weight of individual.

Feature

A **feature** is a **vector** descriptive of **an individual**.

Example: BMI of individual.

Variable types

Variable analysis highly depends on its type !

Variable types

Variable analysis highly depends on its type !

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)**.

Univariate and multivariate analysis

Univariate analysis

Univariate analysis consists in performing the analysis of a **single variable**.

Example: Analyze the petal width.

Univariate and multivariate analysis

Univariate analysis

Univariate analysis consists in performing the analysis of a **single variable**.

Example: Analyze the petal width.

Multivariate analysis

Multivariate analysis consists in analyzing of **multiple variables simultaneously**, to understand relationships and patterns among variables.

Example: Analyze the petal width within each specie.

Variables analysis

To **analyze variables**, you can perform:

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

Variables analysis

To **analyze variables**, you can perform:

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

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

Variables analysis

Question

Can anyone give me:

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

Variables analysis

Question

Can anyone give me:

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

Variables 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.
  $\mathrm{var}(X) = \frac{1}{N}\sum_{i=1}^{N} (x_i - \bar{X})^2$
  $\sigma(X) = \sqrt{\mathrm{var}(X)}$

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*!

Analyzing numeric variables

Question

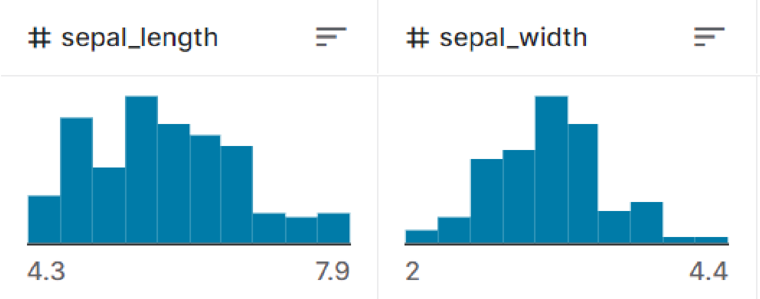
For the vector $[0, 3, 4]$, compute:

- The mean;
- The variance;
- The median.

Representing numeric variables: histograms

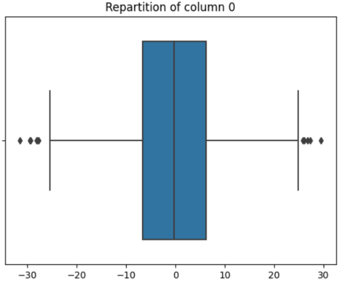
Histograms* consist in:

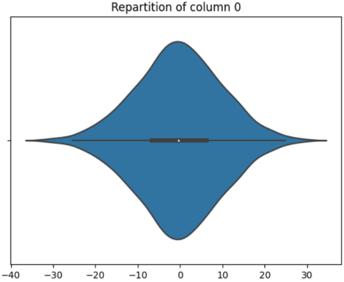
- Dividing the numerical space into intervals of regular length
- Computing the frequency of values per interval



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).





Representing numerical variables

Question

Given the vector $[0, 1, 3, 5, 7, 10, 12, 15]$:

- Plot the histogram;
- Plot the boxplot.

Analyzing and representing categorical variables

Categorical variables are often **harder** to study.

Statistical estimators are:

Analyzing and representing categorical variables

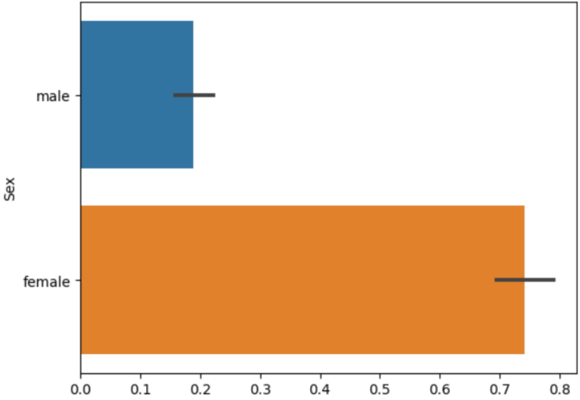
Categorical variables are often **harder** to study.

Statistical estimators are:

- Counts;
- Frequencies.

Analyzing and representing categorical variables

Usual graphical representation can be **bar graphs**.



Analyzing and representing categorical variables

Question

For vector [*"hot"*, *"hot"*, *"hot"*, *"cold"*, *"cold"*]:

- Compute count;
- Compute frequency;
- Plot bar graph.

Multivariate analysis

Previous slides focused on providing tools for the **analysis of a single variable**:

Multivariate analysis

Previous slides focused on providing tools for the **analysis of a single variable**:

How can we perform the analysis of **several variables at the same time** ?

Multivariate analysis

Previous slides focused on providing tools for the **analysis of a single variable**:

How can we perform the analysis of **several variables at the same time** ?

Similarly !

- Statistical estimators;
- Graphical representation.

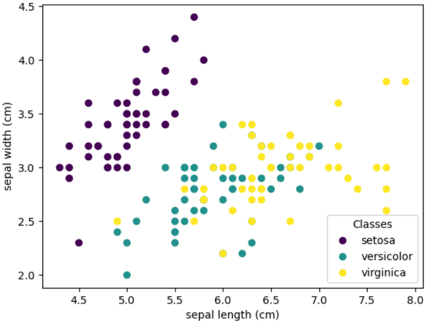
Multivariate analysis: 2 numerical variables

For 2 numerical variables: how does one variable evolve with the other one ?

Multivariate analysis: 2 numerical variables

For 2 numerical variables: how does one variable evolve with the other one ?

- **Correlation**: several ways to compute (Pearson, Spearman ...);
- **Scatter plot**: visualize one variable in respect to another.



Multivariate analysis: 1 numerical variable and 1 categorical

For 1 numerical variable and 1 categorical: how does the value of the numerical variable evolve within each category ?

Multivariate analysis: 1 numerical variable and 1 categorical

For 1 numerical variable and 1 categorical: how does the value of the numerical variable evolve within each category ?

Compute **statistical estimators** for each category, called a **grouped by** operation.

Table: Average Sepal Width for Different Iris Species

Species	Sepal Width (cm)
Setosa	3.428
Versicolor	2.770
Virginica	2.974

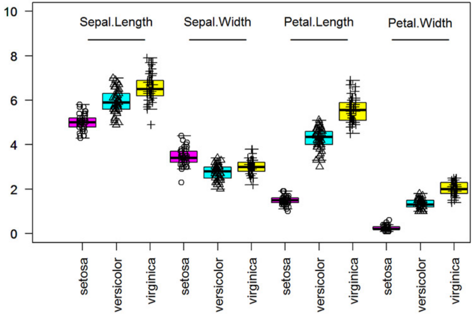
Multivariate analysis: 1 numerical variable and 1 categorical

For 1 numerical variable and 1 categorical: how does the value of the numerical variable evolve within each category ?

Multivariate analysis: 1 numerical variable and 1 categorical

For 1 numerical variable and 1 categorical: how does the value of the numerical variable evolve within each category ?

Perform numerical graphs for each category.



Multivariate analysis: 2 categorical variables

How do **distribution of individuals differ within each category** ?

Possible statistical estimator is count/frequency per category, summarized in a **contingency table**.

	Dog	Cat	Total
Male	42	10	52
Female	9	39	48
Total	51	49	100

Multivariate analysis: 2 categorical variables

How do **distribution of individuals differ within each category** ?

Possible graphical representation is count plot per category.

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 to compute statistical estimators**.

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

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 ?

Data Exchange via HTTP Request

Outline

What is a REST API?

REST API

A REST API (Representational State Transfer) is a **programming interface** that allows computer systems to **communicate with each other** via the **HTTP** protocol.

What is a REST API?

REST API

A REST API (Representational State Transfer) is a **programming interface** that allows computer systems to **communicate with each other** via the **HTTP** protocol.

- Uses standard HTTP operations (GET, POST, PUT, DELETE) to **perform actions on resources**;

What is a REST API?

REST API

A REST API (Representational State Transfer) is a **programming interface** that allows computer systems to **communicate with each other** via the **HTTP** protocol.

- Uses standard HTTP operations (GET, POST, PUT, DELETE) to **perform actions on resources**;
- Operates on a **stateless client-server principle**;

What is a REST API?

REST API

A REST API (Representational State Transfer) is a **programming interface** that allows computer systems to **communicate with each other** via the **HTTP** protocol.

- Uses standard HTTP operations (GET, POST, PUT, DELETE) to **perform actions on resources**;
- Operates on a **stateless client-server principle**;
- Data is usually exchanged in **JSON** or **XML format**.

Characteristics of REST APIs

- **Client-server architecture**: Communication takes place between a **client** and a **server**.
- **Stateless**: Each request from the client to the server **must contain all the information necessary** to understand and process the request.
- **Uniform interface**: **Consistent** use of resources and HTTP operations.

Usage Example

- **GET**: Retrieve data from a resource.
- **POST**: Create a new resource.
- **PUT**: Update an existing resource.
- **DELETE**: Delete a resource.

Usage example

Example of a GET request:

```
GET /users/123
```

Example of corresponding data (JSON):


```
{
  "id": 123,
  "name": "John Doe",
  "email": "john.doe@example.com",
  "username": "johndoe",
  "profile": {
    "age": 30,
    "gender": "male",
    "location": "New York, NY"
  }
```

Usage example

Example of a GET request:

```
GET /users/123
```

Example of corresponding data (XML):


```
<user>
  <id>123</id>
  <name>John Doe</name><email>john.doe@example.com</email>
  <username>johndoe</username>
  <profile>
    <age>30</age>
    <gender>male</gender>
    <location>New York, NY</location>
  </profile>
</user>
```

Questions?

Questions regarding the API 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 for 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 for 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 for 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;

Core Libraries

- **NumPy**: Core library for numerical computations with multidimensional arrays;
- **pandas**: Provides data structures like DataFrames and Series for data manipulation and analysis;

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;

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.

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!

Practical use: pandas and seaborn

Outline

Goal

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

- Pandas
- seaborn

We will use as a practice dataset a **dataset containing the different lines of Shakespeare's play Romeo and Juliet**.

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 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).

Workflow of working with pandas

Usually, you can decompose your Data Science task in:

1. **Loading the data** into RAM;

Workflow of working with pandas

Usually, you can decompose your Data Science task in:

1. **Loading the data** into RAM;
2. **Performing pre-processing** (filtering, sorting, creating new columns);

Workflow of working with pandas

Usually, you can decompose your Data Science task in:

1. **Loading the data** into RAM;
2. **Performing pre-processing** (filtering, sorting, creating new columns);
3. **Computing basic estimators** regarding the dataset (number of individuals, of variables, name of columns ...);

Workflow of working with pandas

Usually, you can decompose your Data Science task in:

1. **Loading the data** into RAM;
2. **Performing pre-processing** (filtering, sorting, creating new columns);
3. **Computing basic estimators** regarding the dataset (number of individuals, of variables, name of columns ...);
4. **Computing statistical estimators** for analysis of data:
    - Standard estimators;
    - Estimators conditioned by other values;

Workflow of working with pandas

Usually, you can decompose your Data Science task in:

1. **Loading the data** into RAM;
2. **Performing pre-processing** (filtering, sorting, creating new columns);
3. **Computing basic estimators** regarding the dataset (number of individuals, of variables, name of columns ...);
4. **Computing statistical estimators** for analysis of data:
    - Standard estimators;
    - Estimators conditioned by other values;
5. Creating **meaningful plots** to explore the data.

Creating DataFrames

The first step of your data processing pipeline is **to load your data in your RAM** by **creating a pandas DataFrame**.

Creating DataFrames

The first step of your data processing pipeline is **to load your data in your RAM** by **creating a pandas DataFrame**.

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 preprocessing: Selection in Pandas

Being able to select columns and/or rows is one of the great advantage of pandas.

- Pandas provides various methods to select specific rows and columns from a DataFrame.
- This allows for targeted data extraction and manipulation based on your analysis needs.

Data preprocessing: Selecting Rows in Pandas

- **Selecting Rows**:
  - Using integer-based indexing (iloc):

    ```
    # Selecting the first row
    first_row = df.iloc[0]

    # Selecting multiple rows
    subset = df.iloc[3:7]
    ```

  - Using label-based indexing (loc):

    ```
    # Selecting a row by label
    row_data = df.loc['row_label']
    ```

Data preprocessing: Selecting Columns in Pandas

- **Selecting Columns**:
  - Using integer-based indexing (iloc):

```
# Selecting a single column by position
column_data = df.iloc[:, 0]

# Selecting multiple columns by positions
subset = df.iloc[:, [1, 3, 5]]
```

  - Using label-based indexing (loc):

```
# Selecting a single column by label
column_data = df['column_name']

# Selecting multiple columns by labels
subset = df[['col1', 'col2', 'col3']]
```

Data pre-processing: Filtering Rows by Single Condition

Pandas DataFrame can be filtered on rows by using a *boolean* indexer.

- **Filtering by Single Condition**:
  - Using comparison operators to filter rows.

```
# Filter rows where Age is greater than 25
adults = df[df['Age'] > 25]

# Filter rows where Name is 'John'
johns = df[df['Name'] == 'John']
```

Data preprocessing: Filtering Rows by Multiple Conditions

- **Filtering by Multiple Conditions**:
  - Combining conditions using logical operators (&, |).

```
# Filter rows where Age is greater than 25
and Gender is 'M'
adult_males = df[(df['Age'] > 25)
& (df['Gender'] == 'M')]

# Filter rows where Age is less than 18
or greater than 60
minors_or_seniors = df[(df['Age'] < 18)
| (df['Age'] > 60)]
```

Data preprocessing: Filtering Rows Using String Methods

- **Filtering Using String Methods**:
  - Using string methods to filter rows based on text data.

```
# Filter rows where Name starts with 'A'
a_names = df[df['Name'].str.startswith('A')]

# Filter rows where Email contains 'example.com'
example_emails = df[df['Email']
.str.contains('example.com')]
```

Data preprocessing: Sorting Data

- **Sorting**: Sorting data based on column values.
  - `df.sort_values(by='Age')` - Sort DataFrame by Age column.

Example

```
# Sort DataFrame by Age in ascending order
sorted_df = df.sort_values(by='Age')
```

Data preprocessing: Adding and Dropping Columns

- **Adding and Dropping**: Adding or dropping columns or rows.
  - `df['Gender'] = ['F', 'M', 'M']` - Adding a new column 'Gender'.
  - `df.drop('Gender', axis=1, inplace=True)` - Dropping the 'Gender' column.

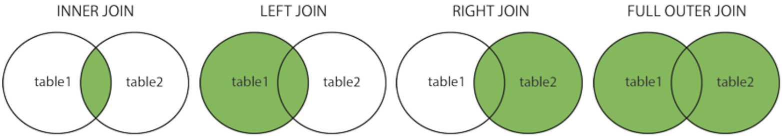
Data preprocessing: Manipulating two datasets

In many cases, **information is split** across several datasets and we need **merging** to retrieve information.

- Merging is a fundamental operation for combining datasets.
- Pandas provides powerful tools to merge DataFrames in various ways.
- Key functions: `pd.merge()`, `df.join()`, and `df.concat()`.

Data preprocessing: Different Types of Joins

- **Inner Join**: Only includes rows with keys present in both DataFrames.
- **Outer Join**: Includes all rows from both DataFrames, filling in `NaN` for missing matches.
- **Left Join**: Includes all rows from the left DataFrame and matching rows from the right DataFrame.
- **Right Join**: Includes all rows from the right DataFrame and matching rows from the left DataFrame.



pd.merge()

- `pd.merge()`: Merges two DataFrames based on a key column(s).
- Can perform inner, outer, left, and right joins.

```
import pandas as pd

# Example DataFrames
left = pd.DataFrame({
    'key': ['A', 'B', 'C', 'D'],
    'value': [1, 2, 3, 4]
})

right = pd.DataFrame({
    'key': ['B', 'D', 'E', 'F'],
    'value': [5, 6, 7, 8]
})
```

Result of pd.merge()

- The result of the inner merge:

```
  key  value_x  value_y
0   B        2        5
1   D        4        6
```

Example of Outer Join

```
# Outer merge on 'key'
outer_merged_df = pd.merge(left, right, on='key',
how='outer')
```

- Result:

```
  key  value_x  value_y
0   A      1.0      NaN
1   B      2.0      5.0
2   C      3.0      NaN
3   D      4.0      6.0
4   E      NaN      7.0
5   F      NaN      8.0
```

Example of Left Join

Example

```
# Left merge on 'key'
left_merged_df = pd.merge(left, right, on='key',
how='left')
```

- Result:

```
  key  value_x  value_y
0   A        1      NaN
1   B        2      5.0
2   C        3      NaN
3   D        4      6.0
```

Example of Right Join

```
# Right merge on 'key'
right_merged_df = pd.merge(left, right, on='key',
how='right')
```

- Result:

```
  key  value_x  value_y
0   B      2.0        5
1   D      4.0        6
2   E      NaN        7
3   F      NaN        8
```

Introduction to Missing Values in Pandas

Values can be missing in datasets for different reasons: abnormal values, failure of measurement...

- Pandas provides various methods to handle missing data effectively.

Detecting Missing Values

- **Detecting Missing Values**:
  - Use the `isnull()` method to detect missing values.

    ```
    # Detect missing values
    missing_values = df.isnull()

    # Summarize missing values per column
    missing_summary = df.isnull().sum()
    ```

  - Use the `notnull()` method to detect non-missing values.

    ```
    # Detect non-missing values
    non_missing_values = df.notnull()
    ```

Handling Missing Values

- **Handling Missing Values**:
  - Use the dropna() method to remove missing values.

```
# Drop rows with any missing values
df_cleaned = df.dropna()

# Drop columns with any missing values
df_cleaned_columns = df.dropna(axis=1)
```

  - Use the fillna() method to fill missing values.

```
# Fill missing values with a specific value
df_filled = df.fillna(0)

# Fill missing values with the mean of the column
df_filled_mean = df.fillna(df.mean())
```

Replacing Missing Values

- **Replacing Missing Values**:
  - Use the `replace()` method to replace specific values.

```
# Replace missing values with a specific value
df_replaced = df.replace(np.nan, 0)

# Replace specific values with other values
df_replaced_specific = df.replace(to_replace=np.nan,
value=-1)
```

Computing basic estimators

The second step is the **computation of basic estimators on the dataset**.

Computing basic estimators

The second step is the **computation of basic estimators on the dataset**.

- Display first/last rows of DataFrame: `df.head()` et `df.tail()`;
- Number of rows and columns: `df.shape`
- Names of columns: `df.columns`

Generating Summary Statistics

The third step is the **computation of statistical estimators and general statistical information** on the dataset.

- Generate summary statistics for numerical columns: `describe();`
- Count occurrences of unique values in a column: `df['Age'].value_counts()`

Aggregation and Grouping

- **Aggregation**: Performing group-wise operations and aggregations.
  - `df.groupby('Gender')['Age'].mean()` - Calculate mean Age by Gender.

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()`