UNIT-1

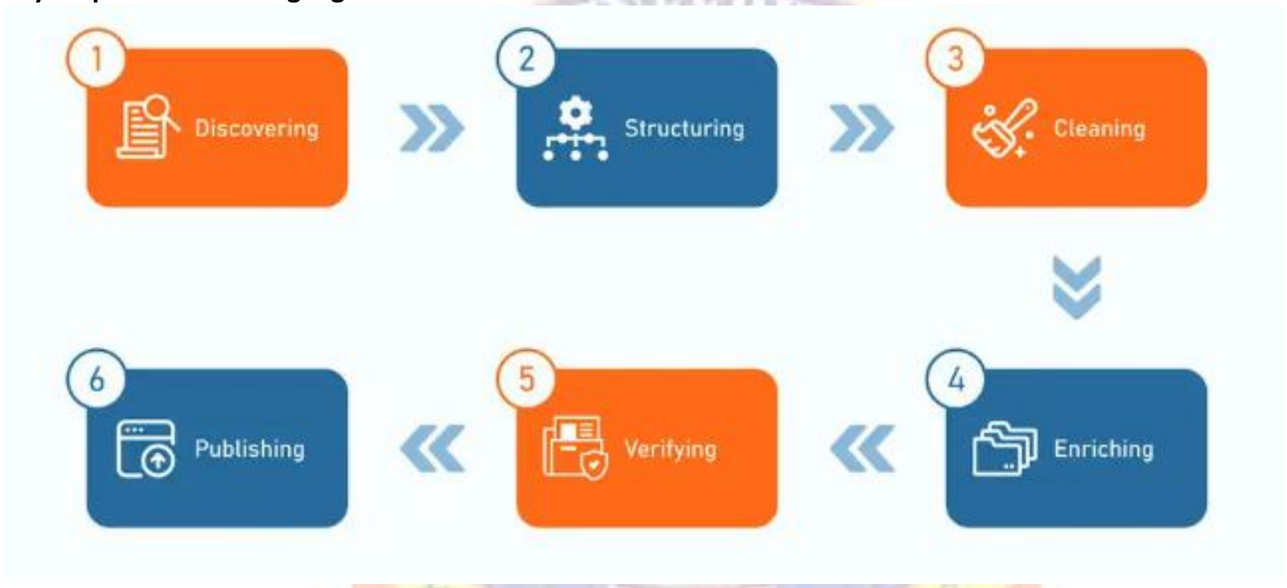
Introduction to Data Wrangling: What Is Data Wrangling?- Importance of Data Wrangling -How is Data Wrangling performed?- Tasks of Data Wrangling-Data Wrangling Tools-Introduction to Python-Python Basics-Data Meant to Be Read by Machines-CSV Data-JSON Data-XML Data.

Introduction to Data Wrangling

What is Data Wrangling?

Data Wrangling, also known as data cleaning, is the process of organizing, and transforming raw data into a structured and usable format for analysis. This critical step ensures that data is accurate, consistent, and ready for decision-making, predictive modeling, or visualization.

Key Steps in Data Wrangling



Data Collection: Gathering data from various sources such as databases, APIs and raw files.

Data Structuring: Organizing data into a structured format (e.g., rows and columns) that aligns with the requirements models.

Data Cleaning: Handling missing values, removing duplicates, and fixing data.

Data Enrichment: Enhancing the data by integrating additional information from other sources to provide context

Data Validation: Verifying the accurate and consistency of the data to ensure it meets quality standards.

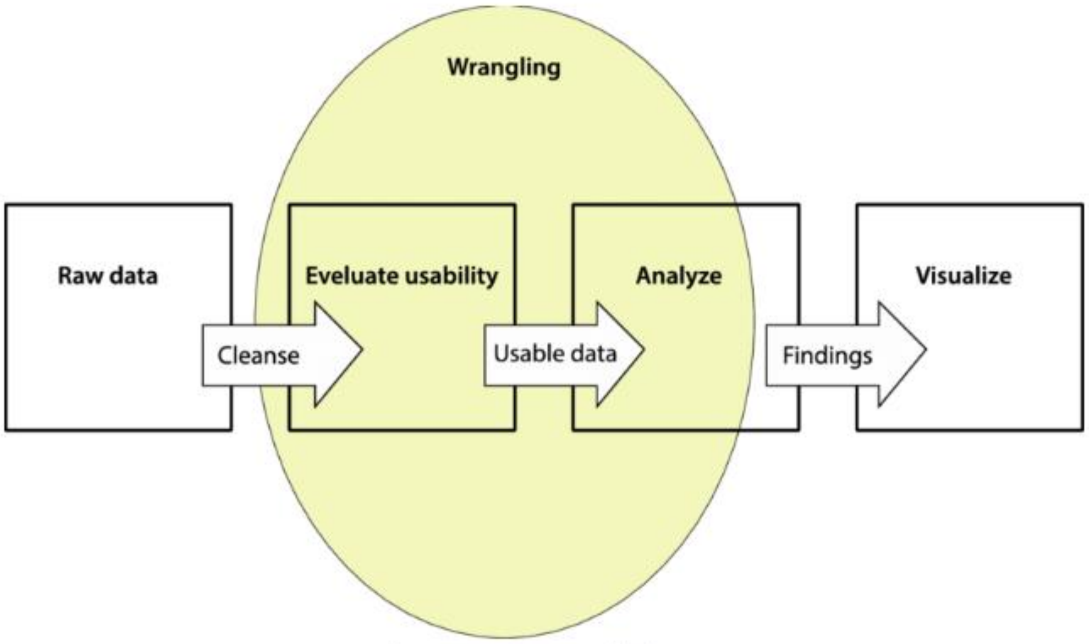
Data Transformation: Converting data into a uniform format, including parsing dates, standardizing text, and normalization techniques.

Why is Data Wrangling Important?
Improves Data Quality: Ensures that insights are based on accurate and reliable information.
Saves Time: Streamlines the analytical process by preparing the data in advance.
Supports Better Decision-Making: High-quality data leads to more accurate analysis and predictions.
Scalability: Prepares data for integration with advanced systems like AI and machine learning.

Tools for Data Wrangling
Python Libraries: Pandas, NumPy, PySpark
R: dplyr, tidyr
Specialized Tools: Talend, Alteryx, Trifacta
Spreadsheet Software: Excel, Google Sheets

Importance of Data Wrangling

Data wrangling acts as the bridge between data collection and meaningful insights, making it a crucial process in any data-driven initiative.
Data wrangling is a critical step in the data analysis process, as it ensures raw data is transformed into a clean, structured, and usable format.
This process is foundational for generating accurate insights, building reliable models, and making data-driven decisions.



Key Reasons Why Data Wrangling is Important

1. Improves Data Quality
   - Ensures datasets are accurate, complete, and consistent by addressing issues like missing values, duplicates, and errors.

2. Enhances Decision-Making
   - Clean and structured data provides a reliable basis for making informed and effective decisions.

3. Saves Time and Resources
   - Automating and standardizing the wrangling process reduces time spent on repetitive data preparation tasks, allowing teams to focus on analysis and strategy.

4. Supports Advanced Analytics
   - Prepares data for complex analytical methods, including predictive modeling, machine learning, and AI applications.

5. Facilitates Integration
   - Harmonizes data from multiple sources, enabling unified analysis and a comprehensive view of operations.

6. Increases Efficiency
   - Streamlined and organized data processing ensures smoother workflows and reduces the risk of errors during analysis.

7. Maximizes Data Usability
   - Makes data accessible and actionable for analysts, scientists, and business users by converting raw data into a standardized format.

8. Drives Better Business Outcomes
   - High-quality data leads to more accurate insights, improved customer experiences, and better operational strategies.

How is Data Wrangling Performed?

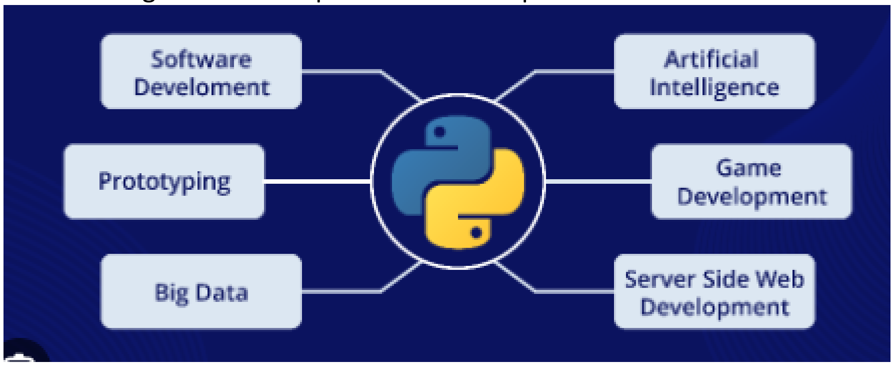
Data wrangling is a step-by-step process of cleaning, transforming, and organizing raw data into a usable format.

Here's how it is typically performed:

1. Data Collection
   -Goal: Gather raw data from various sources such as databases, APIs, spreadsheets, or files.
   - Tasks:
     - Querying databases using SQL.
     - Accessing data through APIs or web scraping.
     - Importing data from CSV, Excel, or JSON files.
2. Data Exploration
   - Goal: Understand the dataset to identify patterns, inconsistencies, and issues.
   - Tasks:
     - Checking data types, missing values, and duplicate entries.
     - Generating summary statistics (e.g., mean, median).
     - Visualizing data distributions using charts or graphs.
3. Data Cleaning
   - Goal: Fix errors and inconsistencies to improve data quality.
   - Tasks:
     - Removing or imputing missing values.
     - Standardizing data formats (e.g., dates, text case).
     - Handling outliers or anomalies.
     - Removing duplicate records.
4. Data Transformation
   - Goal: Reformat and modify data for analysis.
   - Tasks:
     - Converting data types (e.g., strings to integers).
     - Normalizing or scaling numerical values.
     - Splitting or merging columns.
     - Parsing and formatting text (e.g., extracting information).
5. Data Integration
   - Goal: Combine data from multiple sources into a cohesive dataset.
   - Tasks:
     - Merging datasets based on a common key (e.g., user ID).
     - Removing inconsistencies across sources.
     - Aligning data structures and formats.

Introduction to Python

Python is a powerful, high-level, interpreted programming language known for its simplicity and versatility. It is widely used for web development, data analysis, automation, machine learning, artificial intelligence, and more. Python's readable syntax, extensive libraries, and large community support make it an excellent choice for beginners and experienced developers alike.



Python Basics

Variables and Data Types:

Python supports a variety of data types, such as:

- Strings: Text data, enclosed in quotes ("Hello").
- Integers: Whole numbers (10).
- Floats: Decimal numbers (3.14).
- Booleans: True or False (True or False).
- Lists: Ordered collections of items ([1, 2, 3]).
- Dictionaries: Key-value pairs ({"name": "John", "age": 30}).
- Control Structures:Conditional Statements: if, else, elif.
- Loops: for and while loops for iteration.
- Functions:
  A function is a block of reusable code. Defined with def:

Data Meant to Be Read by Machines

Data that is meant to be processed by machines is typically stored in formats that are easy for software to read and manipulate. These formats are structured for easy processing. Common formats include:

- CSV (Comma Separated Values): A plain text format where each line represents a record, and fields are separated by commas. It's widely used in data analysis and data storage.
- JSON (JavaScript Object Notation): A lightweight data format that is easy to read and write, and is widely used for exchanging data between a server and a web application.
- XML (eXtensible Markup Language): A flexible, structured data format used to represent hierarchical data, commonly used in web services and configurations.

CSV Data
CSV is one of the most common data formats used in data storage and data analysis.
It consists of rows of data where each field is separated by a comma, and each line represents a record.
Used for Data storage, analysis, and sharing.

Reading CSV in Python:

Csv code:

```
Name,mobilenumber
adithya,9705655102
shankar,865987959
```

CSV in Python

```
 import csv
with open('data.csv', mode='r') as file:
 csv_reader = csv.reader(file)
 for row in csv_reader:
 print(row)
```

Explanation:

1. `open('data.csv', mode='r')`: Opens the file in read mode.
2. `csv.reader(file)`: Reads the CSV file.
3. `for row in csv_reader`: Iterates through each row, where `row` is a list of values.

Note : create data csv file

Writing to a CSV File:

```
import csv
data = [['Name', 'Age',],['adithya', 30,], ['shankar', 30,] ]
with open('exp.csv', mode='a', newline='') as file:
    writer = csv.writer(file)
    writer.writerows(data)
```

Explanation:

`mode='a'`Appends data to the file without overwriting existing content.Use `'w'` mode to overwrite the file or create a new one if it doesn't exist.

JSON Data
JSON is a lightweight data interchange format that's easy for humans to read and write, and easy for machines to parse and generate.
It stores data in key-value pairs, similar to Python dictionaries.

Reading JSON in Python:
Json code:

```
   {
    "name": "adithya",
    "age":   30,
    "city": "vizag"
   }
```

JSON in Python

```
import json
with open('data.json', mode='r') as file:
 data = json.load(file)
print(data)
print(data['age'])
```

Explanation:

- Use `json.load()` to parse JSON files.
- Access data with `data['key']`. Use `.get('key')` for safe access.
- Ensure the file exists and contains valid JSON.

Note : create data JSON file

Writing to a JSON File

```
import json
data = {

    "name": "Adithya",
    "age": 30,
    "city": "vizag",
     "mobile": "9705655102",

}
with open('csd.json', mode='w') as file:
    json.dump(data, file,indent=4)  # 'indent=4' makes the file readable
```

XML Data
XML is used to represent hierarchical data, and it is often used in web services, configuration files, and document storage.
It uses tags to define the structure of the data.

Reading XML in Python:
Xmlcode:
```xml
<employees>
    <employee>
        <name>lavanya</name>
        <age>25</age>
    </employee>
    <employee>
        <name>Tharun</name>
        <age>30</age>
    </employee>
</employees>
```
XML in Python
```python
import xml.etree.ElementTree as ET

# Parse the XML file
tree = ET.parse('data.xml')
root = tree.getroot()

for employee in root.findall('employee'):
    name = employee.find('name').text
    age = employee.find('age').text
    print(f"Name: {name}, Age: {age}")
```

Writing to an XML File
```python
import xml.etree.ElementTree as ET

# Create the root element
root = ET.Element("employees")

employee1 = ET.SubElement(root, "employee")
name1 = ET.SubElement(employee1, "name")
name1.text = "adithya"
age1 = ET.SubElement(employee1, "age")
age1.text = "28"

tree = ET.ElementTree(root)
with open("employees.xml", "wb") as file:
    tree.write(file, encoding="utf-8", xml_declaration=True)
```

Feature	**CSV**	**JSON**	**XML**
Format	Flat (Tabular)	Key-value pairs (Hierarchical)	Tag-based (Hierarchical)
Data Types	Simple (Strings, Numbers)	Supports strings, numbers, booleans, null, arrays, objects	Supports complex structures
Human Readable	Yes	Yes	Yes
Ease of Use	Simple, Easy to parse	Moderate, easy with libraries	More complex, tag-based parsing
Support for Complex Data	No	Yes	Yes
Use Cases	Spreadsheets, Data Analysis	Web APIs, Configuration Files	Document Storage, Data Interchange
Advantages	- Simple and lightweight. - Easy to read and edit. - Widely supported in data tools and spreadsheets.	- Supports complex, nested data. - Lightweight and easy to parse. - Human-readable. - Easily integrated into JavaScript applications.	- Highly flexible and extensible. - Descriptive, human-readable tags. - Suitable for complex, structured data.
Disadvantages	- Limited to flat/tabular data. - No support for complex/nested structures. - No data types like boolean or null.	- Can be more verbose than CSV for simple data. - Not as efficient for large datasets. - Doesn't support comments.	- Verbose and bulky. - Parsing can be more complex. - Slower than CSV for large datasets.