Assignment : 01

Problem Statement:

Reading and writing different types of datasets.

Objective:

The objective of this assignment is to familiarize ourselves with reading and writing different types of datasets including .txt, .csv, and .xml from the web and local disk storage. We will

explore how to load these datasets into memory, process them, and save them to a specific location on

the disk.

Prerequisite:

1. A Python environment set up with libraries like pandas, xml.etree.ElementTree, and requests (for web access).
2. Internet connection (for reading datasets from the web).
3. Text editor and basic knowledge of file handling in Python.

Theory:

1. Text Files (.txt)

Text files are one of the most basic and widely used file types. They contain raw, unformatted text and are often used for logs, configuration files, or small datasets. Since text files store data sequentially, their structure is typically simple, with each new line representing a new piece of data.

Reading and Writing:
Python provides simple file handling functions for reading and writing text files. To read, you open the file, process its contents line by line (or in bulk), and close the file when done. Writing can involve appending data or completely replacing the file's current content.

Use Cases:
Text files are ideal for storing logs, simple configuration data, or any basic text-based data that

doesn't require a complex structure.

---

2. Comma-Separated Values (.csv)

CSV (Comma-Separated Values) files are commonly used for tabular data, where each row represents a record, and each field in the record is separated by a comma. This format is frequently used for exporting data from databases, spreadsheets, and other tabular datasets.

Structure:
Each line in a CSV file represents a record, with individual fields separated by commas. The first line typically holds column names or headers. CSV files are straightforward to read, write, and edit using programs like Excel or Google Sheets.

Reading and Writing:
Python's `pandas` library simplifies reading and manipulating CSV files. You can easily load, filter, sort, and analyze data with `pandas`.

Use Cases:
CSV files are perfect for storing structured tabular data, such as sales reports, grade sheets, or financial data. Their simplicity makes them readable both programmatically and by humans.

---

3. XML Files (.xml)

XML (Extensible Markup Language) is a versatile file format often used for data exchange and configuration purposes. It organizes data in a hierarchical, tree-like structure using tags, which allows for the representation of more complex relationships between data elements.

Structure:
XML data is enclosed within tags, where each element can contain attributes and text. Although XML is machine-readable, it can also be interpreted by humans, though it tends to be more verbose compared to formats like JSON.

Reading and Writing:
Parsing and manipulating XML files require specific tools, such as Python's `xml.etree.ElementTree` library. Writing XML involves organizing data into a tree structure and saving it to a file.

Use Cases:
XML is commonly used for configuration files, data exchange between different systems (e.g., via web services like SOAP), or structured data where hierarchy is important, such as metadata

and RSS feeds.

---

4. Reading Datasets from the Web

Modern applications often need to access datasets hosted online, which can come in a variety of formats like `.txt`, `.csv`, `.xml`, or `.json`.

Web Access:
Using Python's `requests` library, datasets can be fetched directly from URLs. After downloading, the data can be processed just like any locally stored file.

Challenges:
Working with online datasets introduces potential challenges, such as slow network connections, incomplete downloads, or inconsistent data structures depending on the source.

Use Cases:
Reading web-based datasets is helpful when dealing with frequently updated or large datasets that cannot be stored locally, such as stock market data, weather updates, or data fetched from APIs.

Algorithm:

Identify File Type: Determine the format of the file (e.g., `.txt`, `.csv`, `.xml`).

Select Appropriate Library or Method: Choose the suitable tool or library based on the file type.

Reading Dataset: Load the dataset using the selected library or method.

Preprocess the Data (Optional): Optionally clean or modify the data for further use.

Writing the Dataset: Save the processed data back into the desired file format.

Reference:

1) Pandas Documentation

2) Python official Documentation

Conclusion:

Mastering dataset handling with Python's libraries like `pandas`, `xml.etree.ElementTree`, and `requests` simplifies working with diverse data sources, forming a foundation for data analysis and machine learning.