Assignment No. 1

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 :

In data analysis and manipulation, working with data stored in various file formats is a frequent task. Let's explore three specific types of files we'll concentrate on:

1. **Text Files (.txt) -**

- Store data in a simple, unstructured format with each line representing a data point.
- Utilize built-in Python functions like open(), read(), and write() for easy handling.
- Ideal for lightweight storage of smaller datasets or logs.
- Not suitable for complex data structures due to lack of inherent formatting.
- Pay attention to encoding (e.g., UTF-8) for proper character display.
- Basic string operations may become cumbersome for large datasets.

2. **Comma-Separated Values (.csv) -**

- Structured tabular data with rows and columns.
- Entries are separated by commas for easy parsing.
- Handled efficiently with pandas functions like read_csv() and to_csv().
- Easily created and edited using text editors or spreadsheet software.
- No support for complex data types and may misinterpret data types.
- Can grow large with extensive datasets, affecting performance.

3. **Extensible Markup Language (.xml) -**

- Organizes data in a hierarchical format using descriptive tags.
- Often used for seamless data exchange between different systems.
- Read and write XML files with Python's built-in xml library or third-party libraries like lxml.
- Supports more intricate data relationships than .txt and .csv formats.
- Tags enhance data clarity and provide context.
- Allows for schema definitions (like DTD or XSD) to ensure data integrity.

Libraries –

1. **Pandas –**

- Open-source Python library for data manipulation and analysis.
- Provides Data Frames and Series for structured data handling.
- Supports reading and writing various formats, including CSV, Excel, JSON, and XML.
- Offers functions for efficient filtering and processing of datasets.
- Optimized for performance with large datasets.
- Strong community support and extensive documentation.
- Integrates well with libraries like NumPy and Matplotlib.

2. **xml.etree.ElementTree –**

- Standard Python module for parsing, creating, and modifying XML.
- Simplifies navigation and manipulation of XML using a hierarchical tree structure.
- Supports reading from files and strings, as well as writing back to files.
- User-friendly for basic XML tasks, perfect for smaller projects.
- Lightweight and included in the standard library with no external dependencies.
- Allows searching for elements and attributes using methods like find().
- Ideal for quick XML processing without complex overhead.

3. **lxml –**

- Actively maintained for compatibility with recent Python versions.
- Supports XML schema validation for data integrity.
- Offers a user-friendly API for easy integration.
- Python library for parsing and creating XML and HTML documents.

- Facilitates manipulation of XML using XPath and XSLT, ideal for complex data.

Algorithm (if any to achieve the objective)

1. **Reading and Writing .txt Files:**
   - **Open File:** Use open("fileName.txt", "r") to read the file.
   - **Read Content:** Read the content using content = file.read() or line by line.
   - **Process Content:** Manipulate the content as needed.
   - **Close File:** Close the file with file.close().
   - **Write to File (if needed):** Open in write mode with open("fileName.txt", "w") and use file.write(content).
   - **Close File:** Close the file again after writing.

2. **Reading and Writing .csv Files:**
   - **Import Library:** Import pandas with import pandas as pd.
   - **Read CSV:** Load data using df = pd.read_csv("fileName.csv") or from a URL.
   - **Process Data:** Use pandas functions for data manipulation.
   - **Write to CSV:** Save changes using df.to_csv("newFileName.csv", index=False).
   - **Handle Exceptions:** Include error handling for file access issues.
   - **Close Resources:** Ensure all resources are closed after processing.

Code & Output :

```
# Importing the pandas library for data manipulation and analysis
import pandas as pd

# Loading the Iris dataset from a CSV file into a DataFrame for further analysis
df = pd.read_csv('C:/Users/vishal_2/Assignments/Datasets/iris.csv')

# Displaying the first 5 rows of the DataFrame to get an overview of the dataset
df.head()
```

```
# Defining the file path for the text file containing dataset information
file_path = 'C:/Users/vishal_2/Assignments/Datasets/iris.txt'

# Using 'with' to open the file, ensuring it will be properly closed after reading
with open(file_path, 'r') as file:
    # Reading the entire content of the file and storing it in the variable 'data'
    data = file.read()

# Printing the contents of the file to the console for review
print(data)
```

```
Id,SepalLengthCm,SepalWidthCm,PetalLengthCm,PetalWidthCm,Species
1,5.1,3.5,1.4,0.2,Iris-setosa
2,4.9,3.0,1.4,0.2,Iris-setosa
3,4.7,3.2,1.3,0.2,Iris-setosa
4,4.6,3.1,1.5,0.2,Iris-setosa
5,5.0,3.6,1.4,0.2,Iris-setosa
6,5.4,3.9,1.7,0.4,Iris-setosa
7,4.6,3.4,1.4,0.3,Iris-setosa
8,5.0,3.4,1.5,0.2,Iris-setosa
9,4.4,2.9,1.4,0.2,Iris-setosa
10,4.9,3.1,1.5,0.1,Iris-setosa
11,5.4,3.7,1.5,0.2,Iris-setosa
12,4.8,3.4,1.6,0.2,Iris-setosa
13,4.8,3.0,1.4,0.1,Iris-setosa
14,4.3,3.0,1.1,0.1,Iris-setosa
15,5.8,4.0,1.2,0.2,Iris-setosa
16,5.7,4.4,1.5,0.4,Iris-setosa
17,5.4,3.9,1.3,0.4,Iris-setosa
18,5.1,3.5,1.4,0.3,Iris-setosa
19,5.7,3.8,1.7,0.3,Iris-setosa
20,5.1,3.8,1.5,0.3,Iris-setosa
21,5.4,3.4,1.7,0.2,Iris-setosa
22,5.1,3.7,1.5,0.4,Iris-setosa
23,4.6,3.6,1.0,0.2,Iris-setosa
24,5.1,3.3,1.7,0.5,Iris-setosa
...
148,6.5,3.0,5.2,2.0,Iris-virginica
149,6.2,3.4,5.4,2.3,Iris-virginica
150,5.9,3.0,5.1,1.8,Iris-virginica
```

Output is truncated. View as a scrollable element or open in a text editor. Adjust cell output settings...

```python
# Importing the xml.etree.ElementTree library for XML data parsing and manipulation
# This library treats an XML document as a tree structure for easier navigation and processing
import xml.etree.ElementTree as ET

# Loading and parsing the XML file to create an ElementTree object
tree = ET.parse('C:/Users/vishal_2/Assignments/Datasets/iris.xml')

# Retrieving the root element of the XML document for traversal
root = tree.getroot()

# Iterating through all elements in the XML tree and printing their tags, attributes, and text content
for elem in root.iter():
    print(elem.tag, elem.attrib, elem.text)
```

```
root {}

row {}

Id {} 1
SepalLengthCm {} 5.1
SepalWidthCm {} 3.5
PetalLengthCm {} 1.4
PetalWidthCm {} 0.2
Species {} Iris-setosa
row {}

Id {} 2
SepalLengthCm {} 4.9
SepalWidthCm {} 3
PetalLengthCm {} 1.4
PetalWidthCm {} 0.2
Species {} Iris-setosa
row {}

Id {} 3
SepalLengthCm {} 4.7
SepalWidthCm {} 3.2
PetalLengthCm {} 1.3
PetalWidthCm {} 0.2
...
SepalWidthCm {} 3
PetalLengthCm {} 5.1
PetalWidthCm {} 1.8
Species {} Iris-virginica
```

Output is truncated. View as a scrollable element or open in a text editor. Adjust cell output settings...

```
!pip install lxml
```

```
Requirement already satisfied: lxml in c:\users\dnyan\appdata\local\programs\python\python312\lib\site-packages (5.3.0)
```

+ Code + Markdown

```
# Importing the pandas library for data manipulation and analysis
import pandas as pd

# Loading the XML file into a pandas DataFrame for easier data handling
df = pd.read_xml('C:/Users/vishal_2/Assignments/Datasets/iris.xml')

# Displaying the first few rows of the DataFrame to verify the data has been loaded correctly
print(df.head())
```

```
   Id  SepalLengthCm  SepalWidthCm  PetalLengthCm  PetalWidthCm      Species
0   1            5.1           3.5            1.4           0.2  Iris-setosa
1   2            4.9           3.0            1.4           0.2  Iris-setosa
2   3            4.7           3.2            1.3           0.2  Iris-setosa
3   4            4.6           3.1            1.5           0.2  Iris-setosa
4   5            5.0           3.6            1.4           0.2  Iris-setosa
```

```python
# Creating a dictionary to hold data for the Iris dataset
data = {
    'sepal_length': [5.1, 4.9, 4.7, 4.6],  # Length of the sepal in centimeters
    'sepal_width': [3.5, 3.0, 3.2, 3.1],   # Width of the sepal in centimeters
    'petal_length': [1.4, 1.4, 1.3, 1.5],  # Length of the petal in centimeters
    'petal_width': [0.2, 0.2, 0.2, 0.2],    # Width of the petal in centimeters
    'species': ['setosa', 'setosa', 'setosa', 'setosa']  # Species of the Iris flower
}

# Creating a DataFrame from the dictionary for structured data analysis
df = pd.DataFrame(data)
```

```python
df
```

```python
# Write the DataFrame to a CSV file
df.to_csv('C:/Users/vishal_2/Assignments/Datasets/iris.csv', index=False)
```

```python
# Write the DataFrame to a TXT file (using tab as a delimiter)
df.to_csv('C:/Users/vishal_2/Assignments/Datasets/iris.txt', sep='\t', index=False)
```

```python
# Write the DataFrame to an XML file
df.to_xml('C:/Users/vishal_2/Assignments/Datasets/iris.xml', index=False)
```

References :

- https://www.kaggle.com/code/hamelg/python-for-data-10-reading-and-writing-data
- https://docs.python.org/3/library/csv.html

Conclusion :

This assignment focused on the techniques for reading and writing various dataset formats in Python. We successfully handled text files, CSV files, and XML files stored locally. By using libraries like pandas for CSV operations and xml.etree.ElementTree for XML processing, we enhanced the efficiency of our data management tasks, simplifying the overall workflow.