Practical 2

**Aim:** The aim of the practical implementation is to showcase the conversion of various data formats (CSV, XML, JSON, Database, Image, Video, and Audio) into the Homogeneous Ontology for Recursive Uniform Schema (HORUS) format. The code demonstrates how to transform diverse data sources into a standardized structure, enabling a hub-and-spoke data transformation approach.

**Theory:** The implementation follows a consistent approach for each data type:

1. **CSV to HORUS:**
   * Reads a CSV file and applies processing rules (dropping columns, renaming, setting index, sorting).
   * Outputs the processed data in the HORUS format to a new CSV file.
2. **XML to HORUS:**
   * Reads an XML file, converts it to a DataFrame, and applies similar processing rules as CSV.
   * Outputs the processed data in the HORUS format to a new CSV file.
3. **JSON to HORUS:**
   * Reads a JSON file and applies processing rules similar to CSV and XML.
   * Outputs the processed data in the HORUS format to a new CSV file.
4. **Database to HORUS:**
   * Connects to a SQLite database, retrieves data from a specified table, and applies processing rules.
   * Outputs the processed data in the HORUS format to a new CSV file.
5. **Picture (JPEG) to HORUS:**
   * Reads an image file, flattens and reshapes the data, and creates a DataFrame.
   * Outputs the processed data (containing pixel information) in the HORUS format to a new CSV file.
6. **Video to HORUS:**
   * Extracts frames from a video file and processes each frame similarly to the Picture to HORUS.
   * Outputs the processed data (pixel information for each frame) in the HORUS format to a new CSV file.
7. **Audio to HORUS:**
   * Reads different audio files (2-channel, 4-channel, 6-channel, 8-channel), displays information, and converts them to DataFrames.
   * Outputs the processed audio data in the HORUS format to separate CSV files for each channel configuration.

**Conclusion:** The practical implementation demonstrates the versatility of the HORUS methodology in standardizing data from diverse sources. By converting different data formats into a uniform structure, the framework allows for seamless transformations between formats. This approach enhances interoperability and simplifies the development of data processing pipelines, making it easier to work with varied data sources in a consistent manner.

Practical 3

**Aim:** The aim of the provided practical examples is to showcase various data preprocessing techniques and utilities commonly used in data science. These techniques include data cleaning, binning, averaging, outlier detection, and logging. The practical examples are implemented in Python, using libraries such as pandas, matplotlib, and scipy.

**Theory and Conclusion:**

1. **Fixers Utilities (Part A):**
   * **Removing Spaces:** Leading or lagging spaces in data entries can be problematic. The **strip()** method is used to remove these spaces.
   * **Removing Nonprintable Characters:** Nonprintable characters can be removed using a filter with the **string.printable** set.
   * **Reformatting Date Entries:** Date entries are reformatted from YYYY/MM/DD to DD Month YYYY.

*Conclusion:* These fixers utilities are crucial for ensuring data quality by addressing common issues like spaces, nonprintable characters, and date format discrepancies.

1. **Data Binning or Bucketing (Part B):**
   * A histogram is created to showcase data binning. It demonstrates how continuous values can be grouped into bins.
   * The example uses random data with a normal distribution, plotting a histogram with a best-fit line.

*Conclusion:* Binning is useful for visualizing data distribution and identifying patterns in datasets.

1. **Averaging of Data (Part C):**
   * The example demonstrates how to load data, extract relevant columns, and calculate the mean of latitude values for specific groups (Country and Place\_Name).

*Conclusion:* Averaging data is a common preprocessing step to summarize information, reducing data volume for effective processing.

1. **Outlier Detection (Part D):**
   * Outliers are detected in latitude values for a specific location (London) using mean and standard deviation.
   * Outliers are categorized as higher, lower, and not outliers based on the calculated bounds.

*Conclusion:* Outlier detection is crucial for identifying data points that deviate significantly from the norm, potentially indicating errors or anomalies.

1. **Logging (Part E):**
   * A logging system is set up using Python's logging module, creating log files for different companies and layers.
   * Log entries are created at various levels (debug, info, warning, error) to showcase different log messages.

*Conclusion:* Logging is essential for monitoring and auditing data science processes, providing insights into the execution flow and potential issues.

**Overall Conclusion:** The practical examples cover essential aspects of data preprocessing, including data cleaning, visualization, summarization, outlier detection, and logging. These techniques are fundamental for preparing and understanding data before applying machine learning models or further analysis in a data science workflow.

Top of Form