

Disclaimer and caution: This script/file was generated via generative AI

Thursday February 13, 2025, without prejudice

Approach:

In my experience, duplicate data is in three forms which are handled differently depending on the type of duplication, and depends on defining rules on how to handle scenarios:

1. Duplicate rows
2. Duplicate values in key columns used to identify the record
3. Duplicate values in non-key columns

Prompt:

'Identify the types of duplicate data often found in csv files, and suggest an approach to deduplicate each type.'

Response:

Duplicate data in CSV files can manifest in various forms. Here are some common types of duplicates and suggested approaches to deduplicate each type: Types of Duplicate Data in CSV Files

Exact Duplicates:

Description: Rows that are identical across all columns.

Deduplication Approach: Remove duplicate rows by keeping only one instance.

Partial Duplicates:

Description: Rows that have some identical columns but differ in others. For example, two rows representing the same user but with different timestamps.

Deduplication Approach: Identify key columns that define uniqueness (e.g., user ID) and aggregate or decide which row to keep based on specific criteria (e.g., latest timestamp).

Near Duplicates:

Description: Rows that are similar but not identical, often caused by typos or variations in data entry (e.g., "John Doe" vs. "Jon Doe").

Deduplication Approach: Use string similarity algorithms (e.g., Levenshtein distance) to identify and merge near duplicates.

Duplicate Columns:

Description: Columns that contain the same data but are named differently or are redundant.

Deduplication Approach: Identify and remove or merge duplicate columns based on a defined criterion (e.g., keeping the first occurrence).

Duplicate Records with Different Formats:

Description: The same record represented in different formats (e.g., date formats, casing).

Deduplication Approach: Normalize the data formats before deduplication to ensure consistency.

Suggested Approach for Deduplication in Python

Here's a Python function that demonstrates how to deduplicate exact duplicates from a CSV file using the pandas library:

```
import pandas as pd

def deduplicate_csv(input_file: str, output_file: str) -> None:
    """
    Read a CSV file, remove exact duplicate rows, and save the result to a new CSV
    file.

    Parameters:
    - input_file (str): Path to the input CSV file.
    - output_file (str): Path to save the deduplicated CSV file.

    Raises:
    - FileNotFoundError: If the input file does not exist.
    - pd.errors.EmptyDataError: If the input file is empty.
    """
    try:
        # Load the CSV file into a DataFrame
        df = pd.read_csv(input_file)

        # Remove exact duplicate rows
        df_deduplicated = df.drop_duplicates()

        # Save the deduplicated DataFrame to a new CSV file
        df_deduplicated.to_csv(output_file, index=False)
        print(f"Deduplication complete. Output saved to {output_file}.")

    except FileNotFoundError:
        print(f"Error: The file {input_file} was not found.")
    except pd.errors.EmptyDataError:
        print("Error: The input file is empty.")
    except Exception as e:
        print(f"An unexpected error occurred: {e}")

# Example usage
deduplicate_csv('input_data.csv', 'deduplicated_data.csv')
```

Explanation of the Code

Function Definition: The `deduplicate_csv` function takes the input and output file paths as parameters.

Error Handling: The function includes error handling for file not found and empty

data scenarios.

DataFrame Operations: It uses pandas to read the CSV, remove duplicates with `drop_duplicates()`, and save the result back to a CSV file.

Comments: Each section of the code is commented to explain its purpose and functionality.

Conclusion

By identifying the types of duplicates and implementing a structured approach to deduplication, you can maintain clean and accurate data in your CSV files. The provided code can be adapted for other types of duplicates by modifying the deduplication logic accordingly.

Prompt 2:

'For a data in a python dataframe, write a code that removes all duplicate entries from the data. Print the total number of rows in the data before and after removal, to prove the working of the code.'

Response:

Certainly! Below is a well-structured and documented Python code that reads a CSV record from a URL, identifies columns with "?" values, replaces them with the mean of the respective columns, and converts the columns to float data type. The code includes error handling and follows consistent coding practices.

```
import pandas as pd
import numpy as np

def read_and_clean_csv(url):
    """
    Reads a CSV file from a given URL, replaces "?" with the mean of the
    respective columns,
    and converts those columns to float type.

    Parameters:
        url (str): The URL of the CSV file to read.

    Returns:
        pd.DataFrame: A DataFrame with cleaned data.
    """
    try:
        # Read the CSV file from the provided URL
        data = pd.read_csv(url)

        # Identify columns with "?" and replace them with NaN for easier
        # processing
        data.replace("?", np.nan, inplace=True)

        # Iterate through each column to replace NaN values with the mean
        for column in data.columns:
            if data[column].isnull().any(): # Check if there are NaN values in
            the column
```

```
        # Calculate the mean of the column, ignoring NaN values
        mean_value = data[column].astype(float).mean()

        # Replace NaN values with the calculated mean
        data[column].fillna(mean_value, inplace=True)

        # Convert the column to float type
        data[column] = data[column].astype(float)

    return data

except Exception as e:
    # Handle exceptions and provide informative error messages
    print(f"An error occurred: {e}")
    return None

# Example usage
if __name__ == "__main__":
    url = "https://example.com/data.csv" # Replace with the actual CSV URL
    cleaned_data = read_and_clean_csv(url)

    if cleaned_data is not None:
        print("Cleaned Data:")
        print(cleaned_data.head())
```

Explanation of the Code:

Imports: pandas for data manipulation and numpy for handling NaN values.

Function read_and_clean_csv: This function takes a URL as input, reads the CSV file, and cleans the data.

- It uses `pd.read_csv` to read the CSV file.
- It replaces "?" with NaN for easier handling of missing values.
- It checks each column for NaN values, calculates the mean, and replaces NaN with the mean.
- Finally, it converts the column data type to float.

Error Handling: A try-except block is included to catch and print any errors that occur during the execution of the function.

Example Usage: The code includes a main block to demonstrate how to use the function, which can be modified with the actual URL of the CSV file.

This structure ensures that the code is readable, maintainable, and adheres to recent programming practices.

Here's a simpler script response:

```
import pandas as pd
# Read CSV record from URL
url = 'https://example.com/data.csv'
data = pd.read_csv(url)
print(data.dtypes)
```

```
# Identify columns with "?" values
cols_with_question_mark = data.columns[data.isin(['?']).any()]
print(cols_with_question_mark)
# Replace "?" values with the mean value of the respective attribute
for col in cols_with_question_mark:
    mean_val = pd.to_numeric(data[col], errors='coerce').mean()
    data[col] = pd.to_numeric(data[col].replace('?', mean_val))
# Modify the data type of the attribute to float after replacement
data[cols_with_question_mark] = data[cols_with_question_mark].astype(float)
# Print the modified data
print(data.dtypes)
```