codeDealingWithDuplicateData

October 8, 2024

5.1 Introduction to Dealing with Duplicate Data

In the world of data management, one of the most common and challenging issues is dealing with duplicate data. Duplicate data can arise from a variety of sources: merging datasets from different systems, data entry errors, or repeated data collection processes. While duplicates may seem harmless at first glance, they can lead to significant issues in data analysis, reporting, and decision-making, including inflated metrics, skewed results, and increased storage costs. Dealing with Duplicate Data is a critical step in the data cleaning process, ensuring that each record in the dataset is unique and accurately represents the information it contains.

Definition

Duplicate Data refers to instances where the same data point or record appears more than once in a dataset. This can include exact duplicates, where all fields match perfectly, or partial duplicates, where some fields match but others differ due to minor variations or errors.

Dealing with Duplicate Data involves identifying these duplicate records and deciding on an appropriate method for handling them, whether by removing them, merging them, or flagging them for further review. This process is essential for maintaining the integrity and reliability of the dataset.

Objective

The primary objective of dealing with duplicate data is to ensure that the dataset is free from redundancy, which can distort analysis results and lead to incorrect conclusions. By identifying and removing duplicates, the goal is to create a dataset where each entry is unique, accurately reflecting the data's true value. This process improves the quality of the data, making it more reliable for analysis, reporting, and decision-making.

Importance Dealing with duplicate data is crucial for several reasons: 1. Accuracy in Analysis: Duplicates can significantly skew analysis results, leading to incorrect insights and potentially costly business decisions. Removing duplicates ensures that the analysis reflects the true state of the data. 2. Data Integrity: Ensuring that each record is unique helps maintain the integrity of the data, which is vital for any data-driven process. 3. Efficiency: Duplicate records increase the size of the dataset, leading to higher storage costs and longer processing times. By eliminating duplicates, the dataset becomes leaner and more efficient to work with. 4. Improved Decision-Making: Clean, duplicate-free data supports better decision-making by providing accurate and reliable information. 5. Consistency Across Systems: In integrated systems where data is shared across multiple platforms, dealing with duplicates ensures consistency and avoids discrepancies in reports generated from different sources.

5.2 Techniques for Handling Duplicate Data 1. Identifying Duplicates: Identifying duplicates involves detecting rows in the dataset that are exactly or nearly identical.

2. Removing Duplicates: Removing duplicates refers to the process of deleting repeated rows to clean the dataset.
3. Handling Partial Duplicates: This technique involves managing rows that have identical values in some columns but differ in others.
4. Fuzzy Matching for Near Duplicates: Fuzzy matching identifies rows that are not exactly identical but are similar enough to be considered duplicates.

5.2.1 Identifying Duplicates

Identifying duplicates is the first step in dealing with redundant data. It involves scanning the dataset to find rows that appear more than once. These duplicates can be exact copies or may only match in specific columns.

[13]:
```
import pandas as pd

# Load the data
df = pd.read_csv('D:/Projects/Data-cleaning-series/Chapter05 Dealing with␣
 ↪Duplicate Data/Products.csv')

# Example of the dataset with potential duplicates
print("Original DataFrame:")
print(df.to_string(index=False))

# Identify duplicate rows
duplicates = df[df.duplicated()]

# Display duplicate rows
print("\nDuplicate Rows in the DataFrame:")
print(duplicates.to_string(index=False))
```

```
Original DataFrame:
 Product ID Product Name  Price    Category  Stock               Description
          1     Widget A  19.99 Electronics  100.0    A high-quality widget
          2     Widget B  29.99 Electronics    NaN                       NaN
          3          NaN  15.00  Home Goods   50.0        Durable and stylish
          4     Widget D    NaN  Home Goods  200.0         A versatile widget
          5     Widget E   9.99         NaN   10.0    Compact and efficient
          6     Widget F  25.00 Electronics    0.0 Latest technology widget
          7     Widget G    NaN     Kitchen  150.0     Multi-purpose widget
          8     Widget H  39.99     Kitchen   75.0          Premium quality
          9     Widget I    NaN Electronics    NaN        Advanced features
         10     Widget J  49.99 Electronics   60.0            Best in class

Duplicate Rows in the DataFrame:
Empty DataFrame
Columns: [Product ID, Product Name, Price, Category, Stock, Description]
Index: []
```

Explanation:

The duplicated() method in pandas identifies rows that are duplicated in the DataFrame. By default, it marks all rows except the first occurrence as duplicates.

The resulting Boolean Series is then used to filter the DataFrame, displaying all duplicate rows.

5.2.2 Removing Duplicates

Once duplicates are identified, the next step is to remove them from the dataset. This process ensures that each row in the dataset is unique, which is crucial for accurate data analysis.

[14]:
```
# Remove duplicate rows
df_cleaned = df.drop_duplicates()

# Display the DataFrame after removing duplicates
print("\nDataFrame After Removing Duplicates:")
print(df_cleaned.to_string(index=False))
```

```
DataFrame After Removing Duplicates:
 Product ID Product Name  Price    Category  Stock              Description
          1     Widget A  19.99 Electronics  100.0    A high-quality widget
          2     Widget B  29.99 Electronics    NaN                      NaN
          3          NaN  15.00  Home Goods   50.0       Durable and stylish
          4     Widget D    NaN  Home Goods  200.0       A versatile widget
          5     Widget E   9.99         NaN   10.0     Compact and efficient
          6     Widget F  25.00 Electronics    0.0 Latest technology widget
          7     Widget G    NaN     Kitchen  150.0      Multi-purpose widget
          8     Widget H  39.99     Kitchen   75.0           Premium quality
          9     Widget I    NaN Electronics    NaN         Advanced features
         10     Widget J  49.99 Electronics   60.0             Best in class
```

Explanation:

The drop_duplicates() method removes duplicate rows from the DataFrame, leaving only the first occurrence of each duplicate. This method helps in cleaning the dataset, ensuring no redundant rows remain.

5.2.3 Handling Partial Duplicates

Partial duplicates occur when rows have identical values in specific columns but differ in others. Handling these requires identifying the critical columns where duplication matters and removing or managing these partial duplicates accordingly.

[15]:
```
# Identify partial duplicates based on specific columns
partial_duplicates = df[df.duplicated(subset=['Product Name', 'Category'])]

# Display partial duplicates
print("\nPartial Duplicates in the DataFrame (based on 'Product Name' and␣
 ↪'Category'):")
print(partial_duplicates.to_string(index=False))
```

```
# Remove partial duplicates based on specific columns
df_cleaned_partial = df.drop_duplicates(subset=['Product Name', 'Category'])

# Display the DataFrame after removing partial duplicates
print("\nDataFrame After Removing Partial Duplicates:")
print(df_cleaned_partial.to_string(index=False))
```

```
Partial Duplicates in the DataFrame (based on 'Product Name' and 'Category'):
Empty DataFrame
Columns: [Product ID, Product Name, Price, Category, Stock, Description]
Index: []

DataFrame After Removing Partial Duplicates:
 Product ID Product Name  Price    Category  Stock             Description
          1     Widget A  19.99 Electronics  100.0    A high-quality widget
          2     Widget B  29.99 Electronics    NaN                     NaN
          3          NaN  15.00  Home Goods   50.0       Durable and stylish
          4     Widget D    NaN  Home Goods  200.0        A versatile widget
          5     Widget E   9.99         NaN   10.0    Compact and efficient
          6     Widget F  25.00 Electronics    0.0 Latest technology widget
          7     Widget G    NaN     Kitchen  150.0     Multi-purpose widget
          8     Widget H  39.99     Kitchen   75.0          Premium quality
          9     Widget I    NaN Electronics    NaN        Advanced features
         10     Widget J  49.99 Electronics   60.0            Best in class
```

Explanation:

By specifying a subset of columns in the duplicated() method, you can identify partial duplicates based on those columns.

The drop_duplicates() method with the subset parameter removes these partial duplicates, ensuring the dataset is clean where it matters most.

5.2.4 Fuzzy Matching for Near Duplicates

Near duplicates are records that are not exact matches but are close enough in value to be considered duplicates. Fuzzy matching helps identify such records, which may occur due to minor differences in data entry or formatting.

```
[16]: import pandas as pd
from rapidfuzz import process, fuzz

# Load your dataset
df = pd.read_csv('D:/Projects/Data-cleaning-series/Chapter05 Dealing with␣
 ↪Duplicate Data/Products.csv')

# Define a function to find near duplicates using fuzzy matching
def find_near_duplicates(df, column_name, threshold=80, top_n=3):
```

```python
    # Get unique values from the specified column
    unique_values = df[column_name].dropna().unique()
    near_duplicates = []

    for value in unique_values:
        # Extract matches with the specified threshold and limit
        matches = process.extract(value, unique_values, scorer=fuzz.ratio,␣
↪limit=top_n, score_cutoff=threshold)
        for match in matches:
            if match[0] != value:
                near_duplicates.append((value, match[0], match[1]))

    return near_duplicates

# Find near duplicates in the 'Product Name' column
near_duplicates = find_near_duplicates(df, 'Product Name')

# Display near duplicates
print("\nNear Duplicates in the 'Product Name' column:")
if near_duplicates:
    for original, match, score in near_duplicates:
        print(f"Original: {original}, Match: {match}, Score: {score}")
else:
    print("No near duplicates found.")
```

```
Near Duplicates in the 'Product Name' column:
Original: Widget A, Match: Widget B, Score: 87.5
Original: Widget A, Match: Widget D, Score: 87.5
Original: Widget B, Match: Widget A, Score: 87.5
Original: Widget B, Match: Widget D, Score: 87.5
Original: Widget D, Match: Widget A, Score: 87.5
Original: Widget D, Match: Widget B, Score: 87.5
Original: Widget E, Match: Widget A, Score: 87.5
Original: Widget E, Match: Widget B, Score: 87.5
Original: Widget F, Match: Widget A, Score: 87.5
Original: Widget F, Match: Widget B, Score: 87.5
Original: Widget G, Match: Widget A, Score: 87.5
Original: Widget G, Match: Widget B, Score: 87.5
Original: Widget H, Match: Widget A, Score: 87.5
Original: Widget H, Match: Widget B, Score: 87.5
Original: Widget I, Match: Widget A, Score: 87.5
Original: Widget I, Match: Widget B, Score: 87.5
Original: Widget J, Match: Widget A, Score: 87.5
Original: Widget J, Match: Widget B, Score: 87.5
```

Explanation:

Threshold Adjustment: The threshold is set to 80 to capture more potential matches.

Conditional Output: The code checks if any near duplicates are found before printing the results.