UNIT IV

Data Cleaning and Preparation: Handling Missing Data, Data Transformation, String Manipulation.

Data Wrangling: Join, Combine and Reshape: Hierarchical Indexing, Combining and Merging Datasets, Reshaping and Pivoting.

**Data Cleaning and Preparation**

Data cleaning is an essential step in any data analysis workflow, ensuring your data is complete, consistent, and ready for processing. This section focuses on **handling missing data, data transformation, and string manipulation**.

**1. Handling Missing Data**

Datasets often contain **missing values**. Handling these missing values correctly is crucial to avoid inaccurate analyses.

**Methods to Handle Missing Data:**

1. **Detect Missing Values:** Using isnull() or notnull() methods.
2. **Remove Missing Data:** Using dropna().
3. **Fill Missing Values:** Using fillna() to replace them with a specific value or an aggregate (like mean or median).

**Example: Handling Missing Data**

import pandas as pd

import numpy as np

# Creating a DataFrame with missing values

data = {'Name': ['Alice', 'Bob', 'Charlie', 'David'],

'Age': [24, np.nan, 22, np.nan],

'Score': [85, 90, np.nan, 70]}

df = pd.DataFrame(data)

print("Original DataFrame:\n", df)

# Detecting missing values

print("\nMissing Values:\n", df.isnull())

# Dropping rows with missing values

df\_dropped = df.dropna()

print("\nAfter Dropping Missing Values:\n", df\_dropped)

# Filling missing values with mean

df\_filled = df.fillna(df.mean(numeric\_only=True))

print("\nAfter Filling Missing Values:\n", df\_filled)

**Explanation:**

* **dropna()** removes any rows or columns with missing values.
* **fillna()** replaces missing values with a specific value or an aggregate like mean, median, etc.
* **isnull()** detects missing data by returning a boolean mask.

**2. Data Transformation**

Data transformation involves **modifying data values** to improve their consistency and usability. Common operations include:

1. **Scaling data** for machine learning algorithms.
2. **Renaming columns** for better clarity.
3. **Applying functions** to manipulate column values using apply() or map().

**Example: Data Transformation**

# Renaming columns

df\_renamed = df.rename(columns={'Score': 'Marks'})

print("\nRenamed DataFrame:\n", df\_renamed)

# Applying a function to transform data

df\_renamed['Marks'] = df\_renamed['Marks'].apply(lambda x: x if not np.isnan(x) else 50)

print("\nAfter Applying Transformation:\n", df\_renamed)

**Explanation:**

* Transformation techniques, such as renaming columns or applying functions, make data easier to understand and use.
* **apply()** allows you to apply a function to every element in a column.

**3. String Manipulation**

String manipulation involves modifying text data within a DataFrame. Pandas provides built-in methods to handle strings efficiently.

**Example: String Manipulation**

# Creating a DataFrame with string data

data = {'Name': ['Alice', 'Bob', 'Charlie'], 'Department': ['IT', 'hr', 'Finance']}

df = pd.DataFrame(data)

# Converting text to uppercase

df['Department'] = df['Department'].str.upper()

print("\nUppercase Department Names:\n", df)

# Checking if a string contains a substring

print("\nNames Containing 'li':\n", df[df['Name'].str.contains('li')])

**Explanation:**

* **str.upper()** converts strings to uppercase.
* **str.contains()** checks for the presence of a substring within text data.

**Data Wrangling: Join, Combine, and Reshape Data**

Data wrangling involves **manipulating datasets** to make them more suitable for analysis. Key tasks include **merging, joining, and reshaping datasets**.

**1. Hierarchical Indexing**

Hierarchical indexing allows **multiple levels of indexing**, making it easy to manage and manipulate complex datasets.

**Example: Hierarchical Indexing**

# Creating a DataFrame with multi-level indexing

data = {'State': ['AP', 'AP', 'TS', 'TS'],

'City': ['Vijayawada', 'Guntur', 'Hyderabad', 'Warangal'],

'Population': [1200000, 800000, 7000000, 800000]}

df = pd.DataFrame(data).set\_index(['State', 'City'])

print("\nDataFrame with Hierarchical Indexing:\n", df)

**Explanation:**

* **Hierarchical indexing** provides a way to work with data that has multiple dimensions, such as state and city data.

**2. Combining and Merging Datasets**

Merging allows **combining two or more datasets** based on a key column. Pandas supports:

* **merge()**: Combines data based on key columns.
* **concat()**: Stacks datasets either **vertically** or **horizontally**.

**Example: Merging Datasets**

# Creating two DataFrames to merge

df1 = pd.DataFrame({'ID': [1, 2, 3], 'Name': ['Alice', 'Bob', 'Charlie']})

df2 = pd.DataFrame({'ID': [1, 2, 4], 'Score': [85, 90, 75]})

# Merging on 'ID' column

df\_merged = pd.merge(df1, df2, on='ID', how='inner')

print("\nMerged DataFrame:\n", df\_merged)

**Explanation:**

* **Inner join** merges only the matching rows.
* **Outer join** includes all rows, filling unmatched values with NaN.

**3. Reshaping and Pivoting**

Pandas provides tools for reshaping data to suit your analysis needs.

* **pivot()**: Converts rows into columns.
* **melt()**: Converts columns into rows.
* **stack() and unstack()**: Rearrange hierarchical indexes.

**Example: Reshaping and Pivoting**

# Creating a DataFrame for pivoting

data = {'Date': ['2024-10-01', '2024-10-01', '2024-10-02'],

'City': ['Vijayawada', 'Guntur', 'Vijayawada'],

'Temperature': [30, 32, 29]}

df = pd.DataFrame(data)

# Pivoting the DataFrame

df\_pivot = df.pivot(index='Date', columns='City', values='Temperature')

print("\nPivoted DataFrame:\n", df\_pivot)

**Explanation:**

* **pivot()** reshapes the data by turning unique values from a column into separate columns.
* **melt()** does the opposite by converting columns into rows.

**Summary and Key Takeaways**

In this section, we covered:

1. **Handling Missing Data:** Using dropna() and fillna() to manage incomplete data.
2. **Data Transformation:** Applying functions and renaming columns to prepare data.
3. **String Manipulation:** Using string methods to modify text data.
4. **Hierarchical Indexing:** Managing multi-level indexed data.
5. **Combining and Merging Datasets:** Using merge() and concat() to join datasets.
6. **Reshaping and Pivoting:** Changing the structure of data using pivot() and melt().