





and DataFrame Operations

Class Objective

The session's primary focus was on understanding and practicing DataFrame operations using Pandas, especially concerning merging datasets, slicing, and other critical Pandas functionalities.

Topics Covered:

1. DataFrame Merging
2. Data Slicing
3. Using `apply()` Function
4. Concatenation vs. Merge
5. Pandas Built-in Operations

DataFrame Merging

Basic Merging Concept:

- Merging is an essential operation to combine DataFrames on a common key, similar to SQL joins.
- Merges can be performed in different ways - inner, left, right, and full joins.

Types of Joins:

- **Inner Join:** Keeps only the keys that are present in both DataFrames .
- **Left Join:** Keeps all keys from the left DataFrame, disregarding what's absent in the right .
- **Right Join:** Opposite to left join; keeps all keys from the right.
- **Full Join:** Retains all keys from both DataFrames .

Common Key Consideration:

- You must have a common key to perform a merge; otherwise, a concatenation needs to be considered .



```
merged_data = users.merge(messages, on='user_id', how='inner') #
```

Data Slicing

Slicing with `loc` and `iloc`:

- **`loc`**: Label-based slicing, useful when you have a clear identifier for rows.
 - Syntax: `df.loc[row_labels, column_labels]`
- **`iloc`**: Index-based slicing, useful for direct positional access to rows and columns.
 - Syntax: `df.iloc[row_indices, column_indices]`

Step Slicing:

- You can specify a step in slicing to skip intervals .
 - Example: `df.iloc[0:10:2]` will select every second row from 0 to 10 .
-

The `apply()` Function

- Used to apply functions along an axis (row/column-wise) in a DataFrame .
- Example for converting a column:

```
def encode_gender(gender):  
    return 0 if gender == 'Male' else 1  
  
df['gender_encoded'] = df['gender'].apply(encode_gender)
```

- **Row vs. Column Application:**
 - `axis=0` for columns (default).
 - `axis=1` for rows.
-

Concatenation vs. Merge



- Simply stacking multiple DataFrames without consideration of common keys.
- Use `pd.concat([df1, df2], axis=0)` for vertical stacking.
- Use `pd.concat([df1, df2], axis=1)` for horizontal stacking .

Merge for Data Integrity:

- Preferred for relational data handling where logical combinations of data are necessary .

Built-in Operations

Common Built-in Functions:

- `mean()` , `sum()` , `count()` , `max()` , `min()`
- These simplify everyday analytics workflows and eliminate the need for explicit loop constructs .

Sorting:

- Sorting a DataFrame using `sort_values()` .
- You can sort on multiple columns, and specify ascending/descending order .

Example:

```
sorted_df = df.sort_values(by='age', ascending=True)
```

By following these notes, students should gain a comprehensive understanding of essential Pandas operations as discussed in the class. The focus was on practical applications and understanding the nuances of different methods available to manipulate data effectively and efficiently.