Merging

- In **Pandas**, merging datasets is done using the merge() function. You can use different types of joins, such as **inner join**, **left join**, and **right join**, by specifying the how parameter in the merge() function.

- Syntax

dataframe.merge(*right*, how, on, left_on, right_on, left_index, right_index, sort, suffixes, copy, indicator, validate)

Dataset1.csv

```
id,name
1,John
2,Jane
3,Bob
4,Alice
```

Dataset2.csv

```
id,age
1,23
2,34
3,45
5,56
```

- **Inner Join (Default how value)**

An **inner join** returns only the rows that have matching keys in both datasets.

If a row in either dataset does not have a corresponding match in the other dataset, it is excluded.

```
import pandas as pd

# Load the datasets
df1 = pd.read_csv('dataset1.csv')
df2 = pd.read_csv('dataset2.csv')

# Perform an inner join
merged_inner = pd.merge(df1, df2, on='id', how='inner')

# Display the result
print(merged_inner)
```

```
   id   name  age
0   1   John   23
1   2   Jane   34
2   3    Bob   45
```

- **Left Join (Using how='left')**

A **left join** returns all rows from the left dataset (df1), and the matching rows from the right dataset (df2). If there is no match, the result will have NaN values for columns from the right dataset.

```
# Perform a left join
merged_left = pd.merge(df1, df2, on='id', how='left')

# Display the result
print(merged_left)
```

	id	name	age
0	1	John	23.0
1	2	Jane	34.0
2	3	Bob	45.0
3	4	Alice	NaN

- **Right Join (Using how='right')**

A **right join** returns all rows from the right dataset(df2), and the matching rows from the left dataset (df1). If there is no match, the result will have NaN values for columns from the left dataset.

```
# Perform a right join
merged_right = pd.merge(df1, df2, on='id', how='right')

# Display the result
print(merged_right)
```

```
   id   name  age
0   1   John   23
1   2   Jane   34
2   3    Bob   45
3   5    NaN   56
```

GROUPING

- Grouping data by a categorical column in a DataFrame allows you to aggregate or perform operations on subsets of the data.

- This is commonly done in data analysis to get summary statistics for different groups, such as averages, counts, or sums.

- In **Pandas**, you can use the groupby() function to group data by one or more categorical columns.

- Once the data is grouped, you can apply various aggregation functions to get summary statistics for each group.

```
import pandas as pd

# Sample DataFrame
data = {
   'Category': ['A', 'B', 'A', 'B', 'A', 'B', 'A', 'C', 'C'],
   'Value': [10, 20, 30, 40, 50, 60, 70, 80, 90],
   'Count': [1, 2, 3, 4, 5, 6, 7, 8, 9]
}

df = pd.DataFrame(data)

# Display the DataFrame
print(df)
grouped = df.groupby('Category’)
# Apply aggregation function (sum in this case) on the grouped object
grouped_sum = grouped.sum()
# Print the result
 print(grouped_sum)
```

Category	Value	Count
A	160	16
B	120	12
C	170	17

Common aggrgate functions with GroupBy:

- **sum()**: Sums the values for each group.
- **mean()**: Computes the average for each group.
- **count()**: Counts the number of non-NA values for each group.
- **agg()**: Allows you to apply multiple aggregation functions at once.