

Module 5

Chapter 2

Data Aggregation and Group Operations: GroupBy Mechanics, Data Aggregation, Group-wise Operations and Transformations, Pivot Tables and Cross-Tabulation.

GroupBy Mechanics:

The `groupby()` function in Pandas is important for data analysis as it allows us to group data by one or more categories and then apply different functions to those groups. This technique is used for handling large datasets efficiently and performing operations like aggregation, transformation and filtration on grouped data.

The **groupby()** operation is divided into three main steps:

Step 1: Splitting Data into Groups

Step 2: Applying Functions to Groups

Step 3: Combining Results

Step 1: Splitting Data into Groups

The splitting process refers to dividing the dataset into groups based on a particular condition or key. This can be done using the `groupby()` function by passing one or more columns as keys.

Methods for splitting data are as follows:

1. **Group data by a single key:** In order to group data with one key we pass only one key as an argument in `groupby` function. Here we will group the data by the `Name` column.

```
import pandas as pd

data1 = {'Name': ['Jai', 'Anuj', 'Jai', 'Princi',
                  'Gaurav', 'Anuj', 'Princi', 'Abhi'],
         'Age': [27, 24, 22, 32,
                 33, 36, 27, 32],
         'Address': ['Nagpur', 'Kanpur', 'Allahabad', 'Kannuaj',
                     'Jaunpur', 'Kanpur', 'Allahabad', 'Aligarh'],
         'Qualification': ['Msc', 'MA', 'MCA', 'Phd',
                           'B.Tech', 'B.com', 'Msc', 'MA']}

df = pd.DataFrame(data1)
print(df)

df.groupby('Name')
print(df.groupby('Name').groups)
```

Output:

```
   Name  Age  Address  Qualification
0   Jai   27   Nagpur         Msc
1  Anuj   24   Kanpur          MA
2   Jai   22  Allahabad        MCA
3 Princi  32  Kannauj         Phd
4 Gaurav  33  Jaunpur        B.Tech
5  Anuj   36   Kanpur        B.com
6 Princi  27  Allahabad        Msc
7  Abhi   32  Aligarh          MA
{'Abhi': [7], 'Anuj': [1, 5], 'Gaurav': [4], 'Jai': [0, 2], 'Princi': [3, 6]}
```

Custom dataset

Now we print the first entries in all the groups formed.

```
gk = df.groupby('Name')
gk.first()
```

Output:

	Age	Address	Qualification
Name			
Abhi	32	Aligarh	MA
Anuj	24	Kanpur	MA
Gaurav	33	Jaunpur	B.Tech
Jai	27	Nagpur	Msc
Princi	32	Kannauj	Phd

2. Grouping data with multiple keys : In order to group data with multiple keys, we pass multiple keys in groupby function. Here we group a data of "Name" and "Qualification" together using multiple keys in groupby function.

```
df.groupby(['Name', 'Qualification'])
print(df.groupby(['Name', 'Qualification']).groups)
```

Output:

```
{('Abhi', 'MA'): [7], ('Anuj', 'B.com'): [5], ('Anuj', 'MA'): [1], ('Gaurav', 'B.Tech'): [4], ('Jai', 'MCA'): [2], ('Jai', 'Msc'): [0], ('Princi', 'Msc'): [6], ('Princi', 'Phd'): [3]}
```

3. Grouping data by sorting keys: Group keys are sorted by default using the groupby operation. Now we apply groupby() using sort.

```
df.groupby('Name')['Age'].sum()
```

Output:

Age	
Name	
Abhi	32
Anuj	60
Gaurav	33
Jai	49
Princi	59

Now we apply `groupby()` without using `sort` (we pass `sort=False`).

```
df.groupby(['Name'], sort=False)['Age'].sum()
```

Output:

Age	
Name	
Jai	49
Anuj	60
Princi	59
Gaurav	33
Abhi	32

4. Grouping data with object attributes: Grouping data with object attributes in Pandas allows us to treat the groups attribute as a [dictionary](#) where the keys are unique group values and values are the indices (row labels) corresponding to each group.

```
df.groupby('Name').groups
```

Output:

```
{'Abhi': [7], 'Anuj': [1, 5], 'Gaurav': [4], 'Jai': [0, 2], 'Princi': [3, 6]}
```

5. Iterating through groups: In order to iterate an element of groups, we can iterate through the object.

```
grp = df.groupby('Name')
for name, group in grp:
    print(name)
    print(group)
    print()
```

Output:

```
Abhi
  Name  Age  Address  Qualification
7  Abhi   32  Aligarh              MA

Anuj
  Name  Age  Address  Qualification
1  Anuj   24  Kanpur              MA
5  Anuj   36  Kanpur              B.com

Gaurav
  Name  Age  Address  Qualification
4  Gaurav  33  Jaunpur              B.Tech

Jai
  Name  Age  Address  Qualification
0  Jai   27   Nagpur              Msc
2  Jai   22  Allahabad              MCA

Princi
  Name  Age  Address  Qualification
3  Princi  32  Kannauj              Phd
6  Princi  27  Allahabad              Msc
```

6. Selecting a group: In order to select a group, we can select group using **GroupBy.get_group()**. We can select a group by applying a function GroupBy.get_group this function select a single group.

```
grp = df.groupby('Name')
grp.get_group('Jai')
```

Output:

	Name	Age	Address	Qualification
0	Jai	27	Nagpur	Msc
2	Jai	22	Allahabad	MCA

Now we select an object grouped on multiple columns.

```
grp = df.groupby(['Name', 'Qualification'])
grp.get_group(('Jai', 'Msc'))
```

Output:

	Name	Age	Address	Qualification
0	Jai	27	Nagpur	Msc

Data Aggregation:

Aggregation means applying a mathematical function to summarize data. It can be used to get a summary of columns in our dataset like getting sum, minimum, maximum etc. from a particular column of our dataset. The function used for aggregation is **agg()** the parameter is the function we want to perform. Some functions used in the aggregation are:

Function	Description
sum()	Compute sum of column values
min()	Compute min of column values
max()	Compute max of column values
mean()	Compute mean of column
size()	Compute column sizes
describe()	Generates descriptive statistics
first()	Compute first of group values
last()	Compute last of group values
count()	Compute count of column values
std()	Standard deviation of column
var()	Compute variance of column
sem()	Standard error of the mean of column

Step 2: Applying Functions to Groups

After splitting a data into a group we apply a function to each group, in order to do that we have to perform some operations which are as follows:

1. **Aggregation:** It allows us to find a summary statistic for each group like summing or averaging values.

```
grp1 = df.groupby('Name')
grp1['Age'].aggregate(np.sum)
```

Output:

Age	
Name	
Abhi	32
Anuj	60
Gaurav	33
Jai	49
Princi	59

Now we perform aggregation on a group containing multiple keys.

```
grp1 = df.groupby(['Name', 'Qualification'])
grp1['Age'].aggregate(np.sum)
```

Output:

Age		
Name	Qualification	
Abhi	MA	32
Anuj	B.com	36
	MA	24
Gaurav	B.Tech	33
Jai	MCA	22
	Msc	27
Princi	Msc	27
	Phd	32

When we can apply a multiple functions at once by passing a list or dictionary of functions.

```
grp = df.groupby('Name')
grp['Age'].agg([np.sum, np.mean, np.std])
```

Output:

	sum	mean	std
Name			
Abhi	32	32.0	NaN
Anuj	60	30.0	8.485281
Gaurav	33	33.0	NaN
Jai	49	24.5	3.535534
Princi	59	29.5	3.535534

2. Transformation: It is a process in which we perform some group-specific computations and return a result with the same index as the original data.

Here we are using some different datasets which is created randomly below.

```
import pandas as pd
data2 = {'Name': ['Jai', 'Anuj', 'Jai', 'Princi',
                  'Gaurav', 'Anuj', 'Princi', 'Abhi'],
         'Age': [27, 24, 22, 32,
                  33, 36, 27, 32],
         'Address': ['Nagpur', 'Kanpur', 'Allahabad', 'Kannuaj',
                     'Jaunpur', 'Kanpur', 'Allahabad', 'Aligarh'],
         'Qualification': ['Msc', 'MA', 'MCA', 'Phd',
                           'B.Tech', 'B.com', 'Msc', 'MA'],
         'Score': [23, 34, 35, 45, 47, 50, 52, 53]}
grp2 = df2.groupby('Name')
sc = lambda x: (x - x.mean()) / x.std()
grp2['Age'].transform(sc)
```

Output:

	Age
0	0.707107
1	-0.707107
2	-0.707107
3	0.707107
4	NaN
5	0.707107
6	-0.707107
7	NaN

3. Filtration: It is a process which is used to discard groups based on a condition. For example we can filter out groups where the number of records is less than a certain threshold.

```
grp2 = df2.groupby('Name')
grp2.filter(lambda x: len(x) >= 2)
```

Output:

	Name	Age	Address	Qualification	Score
0	Jai	27	Nagpur	Msc	23
1	Anuj	24	Kanpur	MA	34
2	Jai	22	Allahabad	MCA	35
3	Princi	32	Kannuaj	Phd	45
5	Anuj	36	Kanpur	B.com	50
6	Princi	27	Allahabad	Msc	52

With the ability to group data by one or more keys and apply various operations, groupby simplifies tasks such as summarizing data, transforming values or filtering groups, making it essential for efficient and scalable data analysis.

Pivot Tables: A pivot table is a table of statistics that summarizes the data of a more extensive table (such as from a database, spreadsheet, or business intelligence program). This summary might include sums, averages, or other statistics, which the pivot table groups together in a meaningful way.

Steps Needed

- Import Library (Pandas)
- Import / Load / Create data.
- Use Pandas.pivot_table() method with different variants.

Here, we will discuss some variants of pivot table over the dataframe shown below :


```

# import packages
import pandas as pd

# create data
df = pd.DataFrame({'ID': {0: 23, 1: 43, 2: 12,
                          3: 13, 4: 67, 5: 89,
                          6: 90, 7: 56, 8: 34},
                  'Name': {0: 'Ram', 1: 'Deep', 2: 'Yash',
                          3: 'Aman', 4: 'Arjun', 5: 'Aditya',
                          6: 'Akash', 7: 'Chalsea',
                          8: 'Divya'},
                  'Marks': {0: 89, 1: 97, 2: 45,
                          3: 78, 4: 56, 5: 76,
                          6: 81, 7: 87, 8: 100},
                  'Grade': {0: 'B', 1: 'A', 2: 'F',
                          3: 'C', 4: 'E', 5: 'C',
                          6: 'B', 7: 'B', 8: 'A'}})

# view data
display(df)

```

Output:

	ID	Name	Marks	Grade
0	23	Ram	89	B
1	43	Deep	97	A
2	12	Yash	45	F
3	13	Aman	78	C
4	67	Arjun	56	E
5	89	Aditya	76	C
6	90	Akash	81	B
7	56	Chalsea	87	B
8	34	Divya	100	A

Example 1: Simple use of pivot_table() method.

```
# import packages
import pandas as pd

# create data
df = pd.DataFrame({'ID': {0: 23, 1: 43, 2: 12,
                          3: 13, 4: 67, 5: 89,
                          6: 90, 7: 56, 8: 34},
                  'Name': {0: 'Ram', 1: 'Deep', 2: 'Yash',
                          3: 'Aman', 4: 'Arjun', 5: 'Aditya',
                          6: 'Akash', 7: 'Chalsea',
                          8: 'Divya'},
                  'Marks': {0: 89, 1: 97, 2: 45,
                          3: 78, 4: 56, 5: 76,
                          6: 81, 7: 87, 8: 100},
                  'Grade': {0: 'B', 1: 'A', 2: 'F',
                          3: 'C', 4: 'E', 5: 'C',
                          6: 'B', 7: 'B', 8: 'A'}})

# simple use pivot_table() method
print(pd.pivot_table(df, index = ["ID"]))
```

Output :

Marks	
ID	
12	45
13	78
23	89
34	100
43	97
56	87
67	56
89	76
90	81

Cross-Tabulation:

Cross-tabulation is a statistical technique used to summarize the relationship between categorical variables in a table format. It displays data in rows and columns, where each cell represents the frequency of a category combination.

How Does Cross-Tabulation Organize Data?

Cross-tabulation structures categorical data into a table by counting occurrences for each combination of categories. This makes relationships and patterns between variables easy to compare and interpret.

- Load the dataset using Pandas
- Select variables for rows and columns
- Use `pd.crosstab()` to compute frequencies
- Analyze the table for patterns and relationships

Here we implement cross-tabulation in Python using the Pandas library to analyze relationships between categorical variables.

Syntax

```
pd.crosstab(index, columns, values=None, aggfunc=None,
            margins=False, normalize=False)
```

Parameters Explanation

Parameter	Description
index	Row categories
columns	Column categories
values	Numerical values for aggregation
aggfunc	Aggregation function like sum, mean
margins	Adds row/column totals
normalize	Shows percentages instead of counts

Why Cross-Tabulation is Important in Data Analytics

Cross-tabulation helps in:

1. Data Summarization

Condenses large datasets into meaningful tables.

2. Pattern Identification

Finds relationships between variables.

3. Decision Making

Used in business analytics, survey analysis, healthcare, education.

4. Statistical Analysis

3. Real-World Example

Suppose a college wants to analyze:

- Gender distribution
- Course preference

Dataset:

Student	Gender	Course
A	Male	Python
B	Female	Java
C	Male	Python
D	Female	Python
E	Male	Java

Question:

- How many males/females selected Python or Java?

4. Basic Cross-Tabulation Example

Program 1: Frequency Count

```
import pandas as pd

# Create dataset
data = {
    'Gender': ['Male', 'Female', 'Male', 'Female', 'Male', 'Female'],
    'Course': ['Python', 'Java', 'Python', 'Python', 'Java', 'Java']
}

df = pd.DataFrame(data)

print("Dataset:")
print(df)

# Cross-tabulation
result = pd.crosstab(df['Gender'], df['Course'])

print("\nCross Tabulation:")
print(result)
```

Output

Python

Dataset:

	Gender	Course
0	Male	Python
1	Female	Java
2	Male	Python
3	Female	Python
4	Male	Java
5	Female	Java

Cross Tabulation:

Course	Java	Python
Gender		
Female	2	1
Male	1	2

Explanation

Rows = Gender

Columns = Course

Interpretation:

- Female students:

- Java = 2
 - Python = 1
- Male students:
 - Java = 1
 - Python = 2

Cross-Tabulation with Margins (Totals)

Margins add totals.

Program 2

```
import pandas as pd

data = {
    'Gender': ['Male', 'Female', 'Male', 'Female', 'Male', 'Female'],
    'Course': ['Python', 'Java', 'Python', 'Python', 'Java', 'Java']
}

df = pd.DataFrame(data)

result = pd.crosstab(df['Gender'], df['Course'], margins=True)

print(result)
```

Output

Python

Course	Java	Python	All
Gender			
Female	2	1	3
Male	1	2	3
All	3	3	6

Explanation

- Total Females = 3
- Total Males = 3
- Total Java = 3
- Total Python = 3
- Total students = 6