**Pandas Tutorial**

Last Updated : 09 Jan, 2025

Pandas is a **powerful data manipulation and analysis library for Python**. It provides data structures like series and dataframes to effectively easily clean, transform, and analyze large datasets and integrates seamlessly with other python libraries, such as numPy and matplotlib. It offers powerful functions for data transformation, aggregation, and visualization, which are crucial for effective analysis.

Pandas revolves around two primary [data structures:](https://www.geeksforgeeks.org/data-structures-in-pandas/) series (1D) for single columns and dataframe (2D) for tabular data, enabling efficient data manipulation.

**Section 1: Installing and Importing Pandas**

To install Pandas, run the following command:

*pip install pandas*

To use Pandas in your code, import it with:

*import pandas as pd*

This imports the Pandas library and gives it the **alias pd** for convenience.

**Section 2: Creating DataFrames**

* DataFrame is a two-dimensional table-like data structure **with labeled rows and columns**, where each column can have a different data type (e.g., integers, strings, floats). It can be created from [Python data structures](https://www.geeksforgeeks.org/python-data-structures/) like lists, dictionaries, or a list of dictionaries.

Example: Create DataFrame using [dictionary](https://www.geeksforgeeks.org/python-dictionary/):



1

import pandas as pd

2

​

3

data = {'Name': ['Alice', 'Bob', 'Charlie'],'Age': [25, 30, 35]}

4

df = pd.DataFrame(data)

5

print(df)

**Output**

Name Age

0 Alice 25

1 Bob 30

2 Charlie 35

In this example, a dictionary named data is created with keys representing column names (Name, Age) and values as lists containing the respective data.**pd.DataFrame() function** is then used to convert this dictionary into a DataFrame, which is stored in the variable **df.**

*To learn Different ways to create dataframe, refer:* [*Creating a Pandas DataFrame*](https://www.geeksforgeeks.org/creating-a-pandas-dataframe/) *page.*

* **Series in Pandas** is 1-dimensional labeled array capable of holding any data type (integers, strings, floats, etc.). Each element is associated with an index, either default (0, 1, 2…) or custom labels. It can be created from lists, [NumPy arrays](https://www.geeksforgeeks.org/basics-of-numpy-arrays/), dictionaries, or scalar values.

**Note**: All elements must be of the same data type.

**Example: Creating a Series**



1

import pandas as pd

2

age\_series = pd.Series([25, 30, 35], index=['Alice', 'Bob', 'Charlie'])

3

print(age\_series)

**Output**

Alice 25

Bob 30

Charlie 35

dtype: int64

*To Learn more ways to creating pandas series, refer:* [*Creating a Pandas Series*](https://www.geeksforgeeks.org/creating-a-pandas-series/)*.*

**Section 3 : Reading CSV Files**

[CSV (Comma Separated Values)](https://www.geeksforgeeks.org/csv-file-format/)files are a common format for storing large datasets in plain text. The Pandas library in Python provides,[read\_csv() function](https://www.geeksforgeeks.org/python-read-csv-using-pandas-read_csv/), to load these files into a DataFrame. For our example we will use [people.csv](https://media.geeksforgeeks.org/wp-content/uploads/20241121154629307916/people_data.csv).



1

import pandas as pd

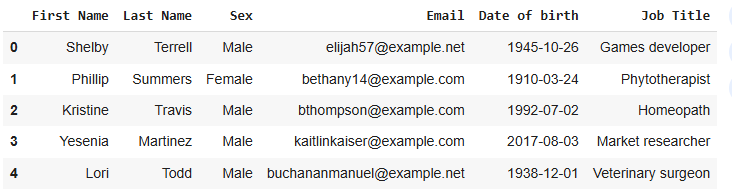
2

df = pd.read\_csv("people.csv")

3

print(df)

**Output:**



*Pandas Read CSV*

**Note:**With this line, pandas will display only the first and last 5 rows if the DataFrame is large.

* Using [to\_string()](https://www.geeksforgeeks.org/python-pandas-dataframe-to_string/) prints the entire DataFrame. Pandas has options to control how much of the DataFrame is displayed.
* You can check or set these using: [pd.options.display.max\_rows](https://www.geeksforgeeks.org/pandas-set_option-function-in-python/).

Pandas provides flexible options to work with various file formats, including CSV and [JSON (JavaScript Object Notation)](https://www.geeksforgeeks.org/javascript-json/). Using read\_json() function in pandas, we can read JSON files. To load a JSON file into a Pandas DataFrame, use the following code:

**import** **pandas** **as** **pd**

*# Load the JSON file into a DataFrame*

df = pd.read\_json('data.json')

print(df)

*Note: To run the code and learn about how the read\_json() function handles different JSON structures with various parameters, refer to the page:* [*Reading JSON Files*](https://www.geeksforgeeks.org/how-to-read-json-files-with-pandas/)*.*

**Section 4: Understanding and Analyzing the Data Frame**

After creating or loading a DataFrame, inspecting and summarizing the data is an important step in understanding dataset. Pandas provides various functions to help you view and analyze the data.

* [**head()**:](https://www.geeksforgeeks.org/python-pandas-dataframe-series-head-method/) View the first n rows of the DataFrame (default is 5 rows).
* [**tail()**:](https://www.geeksforgeeks.org/python-pandas-dataframe-series-tail-method/)View the last n rows of the DataFrame (default is 5 rows).
* [**info()**:](https://www.geeksforgeeks.org/python-pandas-dataframe-info/) This method provides a concise summary of the DataFrame, including the number of non-null entries, column names, and data types. Let’s see an example demonstrating the use of .head(), .tail(), and .info() methods:



1

import pandas as pd

2

​

3

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

4

'Age': [24, 27, 22, 32, 29],

5

'City': ['New York', 'Los Angeles', 'Chicago', 'Houston', 'Phoenix']}

6

​

7

df = pd.DataFrame(data)

8

​

9

# View the first 3 rows of the DataFrame

10

print("First 3 rows using head():")

11

print(df.head(3))

12

​

13

# View the last 2 rows of the DataFrame

14

print("\nLast 2 rows using tail():")

15

print(df.tail(2))

16

​

17

# Get a concise summary of the DataFrame

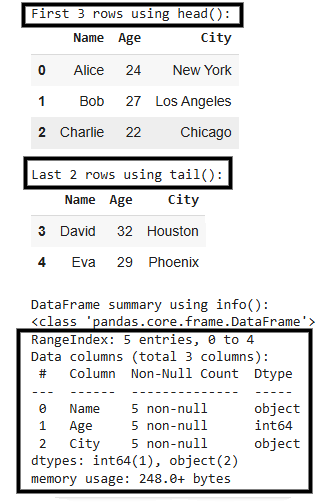
18

print("\nDataFrame summary using info():")

19

df.info()

**Output:**



*Viewing and Inspecting Data With Pandas*

Pandas offers versatile tools for viewing and inspecting data, enabling to quickly summarize or dive deeper into specific data attributes. To explore more advanced functionalities in Pandas, you can follow the links below:

* [View basic statistical details](https://www.geeksforgeeks.org/python-pandas-dataframe-describe-method/)
* [Get information about the DataFrame (data types, non-null counts, etc.)](https://www.geeksforgeeks.org/python-pandas-dataframe-info/)
* [Check for missing data](https://www.geeksforgeeks.org/count-nan-or-missing-values-in-pandas-dataframe/)
* [Inspect the DataFrame shape (rows and columns)](https://www.geeksforgeeks.org/python-pandas-df-size-df-shape-and-df-ndim/)
* [Checking rows with minimum and maximum values](https://www.geeksforgeeks.org/select-row-with-maximum-and-minimum-value-in-pandas-dataframe/)

***Note:******describe*** *generates descriptive statistics for numerical columns (such as count, mean, standard deviation, etc.). You can apply it to the entire DataFrame or specific columns. (You can also use .describe(include='all') to include non-numeric data in the summary.)*

**Section 5: Indexing in Pandas**

**Indexing in Pandas** refers to process of **accessing and selecting data from a Pandas DataFrame or Series.**There are multiple ways to do this. We will cover how to to basic indexing, select specific columns , apply slicing, and use Boolean indexing to filter data efficiently.

**Example :**Basic Indexing**( Selecting a single column ) with use of [ ] operator:**



1

import pandas as pd

2

​

3

data = {'Name': ['Alice', 'Bob', 'Charlie'],'Age': [25, 30, 35]}

4

df = pd.DataFrame(data)

5

​

6

# Select a single column

7

age\_column = df['Age']

8

print(age\_column)

We can also select multiple columns by passing a list of column names. For example, you can select both the ‘Name’ and ‘Age’ columns by providing a list [‘Name’, ‘Age’]. **When it comes to selecting rows:** we can use **.loc[] to select rows by label, meaning we refer to the row index or label directly**. Alternatively, .iloc[] allows for position-based indexing, where we select rows by their integer positions.



1

import pandas as pd

2

​

3

data = {'Name': ['Alice', 'Bob', 'Charlie'], 'Age': [25, 30, 35]}

4

df = pd.DataFrame(data)

5

​

6

# Using .loc[] to select rows by label

7

row\_by\_label = df.loc[1] # Selects the row with index label 1 (Bob's data)

8

​

9

# Using .iloc[] to select rows by position

10

row\_by\_position = df.iloc[1] # Selects the second row (Bob's data)

11

​

12

print("Row by label:\n", row\_by\_label)

13

print("Row by position:\n", row\_by\_position)

For more examples where we have covered selecting single and multiple columns and row based selection: [Indexing and Selecting Data with Pandas](https://www.geeksforgeeks.org/indexing-and-selecting-data-with-pandas/). Explore the following resources for advanced indexing techniques:

* [*Using loc[] and iloc[] for Advanced Row and Column Selection*](https://www.geeksforgeeks.org/select-rows-columns-by-name-or-index-in-pandas-dataframe-using-loc-iloc/)
* [*Boolean Indexing in Pandas*](https://www.geeksforgeeks.org/boolean-indexing-in-pandas/)
* [*Slicing dataFrames and series*](https://www.geeksforgeeks.org/slicing-indexing-manipulating-and-cleaning-pandas-dataframe/) *: Part of indexing in Pandas, refers to selecting a subset of rows or columns from a DataFrame or Series by specifying a range of indices.*

**Section 6: Selecting and Filtering Data – Techniques to Filter Based on Conditions**

For selecting and filtering the data after indexing: is about narrowing down the data by applying conditions to select only those rows or columns that meet certain criteria.



1

import pandas as pd

2

​

3

data = {'Name': ['Alice', 'Bob', 'Charlie'],'Age': [25, 30, 35]}

4

df = pd.DataFrame(data)

5

​

6

# Filtering rows where Age is greater than 30

7

filtered\_df = df[df['Age'] > 28]

8

print(filtered\_df)

In this example, we filter the rows where the Age column is greater than 30. This is a simple condition that helps narrow down the data to only those rows that meet the condition. For more methods, refer to our page : [Selecting Rows from a Dataframe based on Column Values](https://www.geeksforgeeks.org/how-to-select-rows-from-a-dataframe-based-on-column-values/). You can also refer to following resources for advanced techniques and more conditions in filtering and selection:

* [*Select Rows With Multiple Filters in Pandas*](https://www.geeksforgeeks.org/select-rows-with-multiple-filters-in-pandas/)
* [*Filtering by Column Value*](https://www.geeksforgeeks.org/ways-to-filter-pandas-dataframe-by-column-values/)
* [*Filter rows based on a list of values*](https://www.geeksforgeeks.org/python-pandas-dataframe-isin/)
* [*Filter Pandas Dataframe with multiple conditions*](https://www.geeksforgeeks.org/filter-pandas-dataframe-with-multiple-conditions/)

**Section 7: Dealing with Rows and Columns in Pandas DataFrame**

Involves various data manipulation techniques in Pandas including adding and deleting columns, truncating data, iterating over DataFrames, and sorting data. [Dealing with Rows and Columns in Pandas DataFrame](https://www.geeksforgeeks.org/dealing-with-rows-and-columns-in-pandas-dataframe/) – every concept and step is explained in this post.

**1. Adding a New Column to DataFrame**: There are several methods available, each suitable for specific use cases. We can easily add new columns by assigning values to them by direct assignment.



1

import pandas as pd

2

​

3

df = pd.DataFrame({'Name': ['Alice', 'Bob', 'Charlie'], 'Age': [25, 30, 35]})

4

​

5

# Adding a new column with a list of values

6

df['Salary'] = [50000, 60000, 70000]

7

print(df)

**Output**

Name Age Salary

0 Alice 25 50000

1 Bob 30 60000

2 Charlie 35 70000

*There are multiple methods, for that refer to:* [*Adding new column to existing dataFrame in pandas*](https://www.geeksforgeeks.org/adding-new-column-to-existing-dataframe-in-pandas/)*. If you want to add columns from one dataFrame to another, refer to* [*Adding Columns from Another DataFrame*](https://www.geeksforgeeks.org/how-to-add-column-from-another-dataframe-in-pandas/)*.*

**2.**[Renaming columns](https://www.geeksforgeeks.org/how-to-rename-columns-in-pandas-dataframe/) using the rename() method or by directly modifying the columns attribute.

* Use rename() for selective renaming.
* Use columns attribute for renaming all columns at once.



1

import pandas as pd

2

​

3

df = pd.DataFrame({'A': [1, 2, 3], 'B': [4, 5, 6], 'C': [7, 8, 9]})

4

​

5

# Renaming columns

6

df.rename(columns={'A': 'X', 'B': 'Y', 'C': 'Z'}, inplace=True)

7

print(df)

**Output**

X Y Z

0 1 4 7

1 2 5 8

2 3 6 9

*We can also:* [***Rename column by index in Pandas***](https://www.geeksforgeeks.org/rename-column-by-index-in-pandas/)***,*** *refer to the article for examples.*

**3.**[**Reindexing Data with Pandas**](https://www.geeksforgeeks.org/reindexing-in-pandas-dataframe/): allows to change the row and column labels of a DataFrame or Series. This operation is useful for aligning data with a new set of labels, handling missing data, and reorganizing datasets. The[reindex() method](https://www.geeksforgeeks.org/python-pandas-dataframe-reindex/) can change the row indices or column labels of a DataFrame.

**Note: If the new index includes labels not present in the original DataFrame, the corresponding values will be set to NaN by default.**



1

import pandas as pd

2

​

3

# Sample DataFrame

4

data = {'A': [10, 20, 30], 'B': [40, 50, 60]}

5

df = pd.DataFrame(data)

6

​

7

# Reindex rows

8

new\_index = [0, 1, 2, 3]

9

df\_reindexed = df.reindex(new\_index)

10

print("Reindexed Rows:\n", df\_reindexed)

**Output**

Reindexed Rows:

A B

0 10.0 40.0

1 20.0 50.0

2 30.0 60.0

3 NaN NaN

*What if you want to reset the index? for that refer to:* [*Convert Index to Column in Pandas Dataframe*](https://www.geeksforgeeks.org/how-to-convert-index-to-column-in-pandas-dataframe/)

**Section 8: Handling Missing Data in Pandas**

[Working with Missing Data in Pandas](https://www.geeksforgeeks.org/working-with-missing-data-in-pandas/) is one of the most common tasks in data manipulation. Pandas provides various functions to identify, fill, and drop missing values efficiently. To explore the procedure hand-to-hand in detail refer to below steps:

**1. Identifying Missing Data With Pandas**using:

* [isnull()](https://www.geeksforgeeks.org/python-pandas-isnull-and-notnull/): Returns True where values are missing (NaN), and False where values are present.
* [notnull()](https://www.geeksforgeeks.org/python-pandas-isnull-and-notnull/): Returns True where ues are present, and False where they are missing.

Here’s an example:

**import** **pandas** **as** **pd**

**import** **numpy** **as** **np**

df = pd.DataFrame({'Col1': [1, 2, np.nan],'Col2': [3, np.nan, np.nan]})

*# Check for missing values*

print(df.isnull())

*# Output:*

*# Col1 Col2*

*# 0 False False*

*# 1 False True*

*# 2 True True*

**2. Filling Missing Data**: For Replace missing values with a specific value, [**fillna()**](https://www.geeksforgeeks.org/python-pandas-dataframe-fillna-to-replace-null-values-in-dataframe/) is used. For example: you want to fill missing values with default value 0 you can perform it like : **df.fillna(0).**



1

import pandas as pd

2

import numpy as np

3

df = pd.DataFrame({'Col1': [1, 2, np.nan],'Col2': [3, np.nan, np.nan]})

4

df.fillna(0)

5

print(df)

We can also fill missing values with below techniques:

* [*Replacing with the mean, median, or mode of the column*](https://www.geeksforgeeks.org/replacing-missing-values-using-pandas-in-python/)
* [*forward fill – Fill missing values with the last valid observation*](https://www.geeksforgeeks.org/python-pandas-dataframe-fillna-to-replace-null-values-in-dataframe/)
* [*backward fill- Fill missing values with the next valid observation*](https://www.geeksforgeeks.org/python-pandas-dataframe-bfill/#:~:text=bfill()%20is%20used%20to,present%20in%20the%20pandas%20dataframe.&text=limit%20%3A%20integer%20value%2C%20No.,na%20cells%20to%20be%20populated.)

**3. Dropping Missing Data With Pandas:**We can also remove rows or columns that contain missing values **using .**[**dropna()**](https://www.geeksforgeeks.org/python-pandas-dataframe-dropna/)**. For example:**



1

import pandas as pd

2

data = {'Name': ['Alice', 'Bob', None], 'Age': [25, None, 35]}

3

df = pd.DataFrame(data)

4

​

5

# Drop rows with missing values

6

df\_dropped = df.dropna()

7

print(df\_dropped)

We can also use **df.dropna(axis=1) to remove columns** or **df.dropna(subset=[‘Column1’, ‘Column2’])** to drop rows with missing values in specific columns. For understand various ways for which we can use drop refer to:

* [Dropping rows from Pandas dataframe with missing values](https://www.geeksforgeeks.org/drop-rows-from-pandas-dataframe-with-missing-values-or-nan-in-columns/)
* Dropping one or multiple columns

**Section 9: Aggregation and Grouping With Pandas**

Aggregation and grouping in Pandas are powerful tools for analyzing and summarizing data. **Grouping allows to segment your data into categories, while aggregation performs operations (like sum, mean, or count) on these groups to derive insights.**[groupby() function](https://www.geeksforgeeks.org/python-pandas-dataframe-groupby/)is commonly used for grouping data, followed by aggregation methods like sum(), mean(), or custom functions for statistical analysis.

**Examples of operations:**

**import** **pandas** **as** **pd**

data = {'Category': ['A', 'B', 'A', 'B'], 'Value': [10, 20, 30, 40]}

df = pd.DataFrame(data)

*# Group data by 'Category' and calculate the sum*

grouped\_sum = df.groupby('Category').sum()

print("Sum:**\n**", grouped\_sum)

*# Compute the mean for each group*

grouped\_mean = df.groupby('Category')['Value'].mean()

print("**\n**Mean:**\n**", grouped\_mean)

*# Aggregate using multiple functions*

grouped\_agg = df.groupby('Category').agg(['sum', 'mean'])

print("**\n**Aggregated:**\n**", grouped\_agg)

In the example above:

* groupby('Category'): Groups data by the ‘Category’ column.
* .sum(), .mean(), and .size(): Perform aggregation on the grouped data.
* [*Grouping and Aggregating with Pandas*](https://www.geeksforgeeks.org/grouping-and-aggregating-with-pandas/) *– this is the main page where in-depth explanations and examples are given.*
* *For more advanced operations, you can use .agg() to* [*apply custom aggregation functions.*](https://www.geeksforgeeks.org/write-custom-aggregation-function-in-pandas/)
* [*Grouping Rows in pandas*](https://www.geeksforgeeks.org/grouping-rows-in-pandas/)

**Section 10: Merging and Concatenating Data in Pandas**

This section covers techniques for **combining multiple DataFrames or Series into a single DataFrame**. These operations are essential for integrating datasets and can be performed in several ways:

**1. Merging DataFrames**: Combines data based on common column or index using functions like merge or join. **There are 4 types of joins:**

* **Inner Join:** Keep rows that match in both DataFrames.
* **Left Join:** Keep all rows from the left DataFrame and match data from the right.
* Right Join and Outer Join are similar but differ in data retention rules.

*Refer :* [*Different types of Joins in pandas*](https://www.geeksforgeeks.org/different-types-of-joins-in-pandas/)

**Example of operations:**



1

import pandas as pd

2

data = {'Name': ['Alice', 'Charlie', 'Edward', 'Grace'],

3

'Years\_Experience': [2, 3, 4, 6],

4

'Role': ['Manager', 'Analyst', 'Developer', 'HR']}

5

​

6

df = pd.DataFrame(data)

7

​

8

# Merging DataFrames on the 'Name' column (inner join)

9

df\_merged\_inner = pd.merge(df1, df3, on='Name', how='inner')

10

​

11

# Merging DataFrames with outer join

12

df\_merged\_outer = pd.merge(df1, df3, on='Name', how='outer')

13

​

14

print("\nInner Merge (Intersection of Names):\n", df\_merged\_inner)

15

print("\nOuter Merge (Union of Names):\n", df\_merged\_outer)

For examples of all operations:

* [Merging Dataframes using merge](https://www.geeksforgeeks.org/joining-two-pandas-dataframes-using-merge/)
* Joining the dataframes – Join method

**2. Concatenating Data**: Refers to stacking DataFrames either vertically (adding rows) or horizontally (adding columns). This can be achieved using the pd.concat() function. Let’s create two dataFrames and concatenate it with the original one:



1

import pandas as pd

2

data = {'Name': ['Alice', 'Charlie', 'Edward', 'Grace'],'Years\_Experience': [2, 3, 4, 6],'Role': ['Manager', 'Analyst', 'Developer', 'HR']}

3

df = pd.DataFrame(data)

4

​

5

# New DataFrame to concatenate

6

new\_data = {'Name': ['John', 'Lily'],'Years\_Experience': [5, 3],'Role': ['Designer', 'Developer']}

7

new\_df = pd.DataFrame(new\_data)

8

​

9

# Concatenate the original and new DataFrames along rows (axis=0)

10

concatenated\_df = pd.concat([df, new\_df], axis=0, ignore\_index=True)

11

​

12

print("Concatenated DataFrame:\n", concatenated\_df)

**In the code:**

* The axis=0 parameter specifies concatenation along rows (vertically).
* ignore\_index=True resets the index so it doesn’t retain the original indices from both DataFrames.

*Learn* [*Concatenate Two or More Pandas DataFrames*](https://www.geeksforgeeks.org/how-to-concatenate-column-values-in-pandas-dataframe/) *with all operations and multiple examples. For more, Go Through below articles:*

* [*Concatenate values in different columns to one column*](https://www.geeksforgeeks.org/how-to-concatenate-column-values-in-pandas-dataframe/)
* [*Merge two dataFrames on certain columns*](https://www.geeksforgeeks.org/merge-two-pandas-dataframes-on-certain-columns/)
* [*Merge multiple CSV Files into single dataframe*](https://www.geeksforgeeks.org/how-to-merge-multiple-csv-files-into-a-single-pandas-dataframe/)

**Section 11: Reshaping Data in Pandas**

Reshaping data involves changing the structure of rows and columns to better organize or analyze data. **Common operations include:**

**1. Pivot Tables in Pandas**: Reshape data based on column values. The .[pivot\_table() method](https://www.geeksforgeeks.org/python-pandas-pivot_table/) is particularly powerful for creating aggregated views of data.



1

import pandas as pd

2

data = {'Date': ['2024-10-01', '2024-10-01', '2024-10-02', '2024-10-02'],'Category': ['A', 'B', 'A', 'B'],'Values': [10, 20, 15, 25]}

3

df = pd.DataFrame(data)

4

​

5

pivot\_table = df.pivot\_table(values='Values', index='Date', columns='Category', aggfunc='sum')

6

print(pivot\_table)

*For more examples and in-depth understanding:* [*Pivoting tables in pandas*](https://www.geeksforgeeks.org/pivot-tables-in-pandas/)

**2. Melting and Unmelting: M**ultiple columns are combined into a single key-value pair.

***When you melt data:***

* *The column names become the* ***keys****.*
* *The values in the columns become the* ***values****.*

Note: To **melt** data, you must specify columns that act as “identifiers” (id\_vars) and others that need to be melted (value\_vars).



1

import pandas as pd

2

data = {'Date': ['2024-10-01', '2024-10-01', '2024-10-02', '2024-10-02'],'Category': [10, None, 15, None],'Category\_B': [None, 20, None, 25]}

3

df = pd.DataFrame(data)

4

​

5

# Melt data from wide to long format

6

melt\_df = df.melt(id\_vars='Date', var\_name='Category', value\_name='Values')

7

print(melt\_df)

*More examples with :* [*Reshaping – Melt and Unmelt*](https://www.geeksforgeeks.org/reshaping-pandas-dataframes-using-melt-and-unmelt/)

**3. Stacking and Unstacking With Pandas**: R**eshape the data** by changing its rows and columns, particularly when working with **MultiIndex DataFrames**. These techniques help pivot the layout of your data.

* **Stacking**: Moves the data from **columns** to **rows** (making the DataFrame taller).
* **Unstacking**: Moves the data from **rows** to **columns** (making the DataFrame wider). Let’s take a simple DataFrame with multiple levels of columns.



1

import pandas as pd

2

data = {

3

('Sales', 'Q1'): [100, 150],

4

('Sales', 'Q2'): [200, 250],

5

('Expenses', 'Q1'): [50, 70],

6

('Expenses', 'Q2'): [80, 90]

7

}

8

​

9

index = ['Product\_A', 'Product\_B']

10

df = pd.DataFrame(data, index=index)

11

print(df)

12

​

13

stacked\_df = df.stack() # Columns → Rows

14

print(stacked\_df)

15

​

16

unstacked\_df = stacked\_df.unstack() # Rows → Columns

17

print(unstacked\_df)

**Explanation:**

* Columns are stacked into rows.
* Now, the data has a MultiIndex for rows:
  + First level: Product (e.g., Product\_A, Product\_B)
  + Second level: Category (e.g., Sales, Expenses).
* Q1 and Q2 remain as columns

*Let’s compare and understand in-depth with more examples :* [*stack,unstack and melt method*](https://www.geeksforgeeks.org/reshape-a-pandas-dataframe-using-stackunstack-and-melt-method/)*. We can also use transpose method for* [*Swapping Rows and Columns*](https://www.geeksforgeeks.org/python-pandas-dataframe-transpose/)

**Section 12: Handling Large Datasets With Pandas**

When working with large datasets or performing intensive computations, **optimizing performance in Pandas** is required. Below are some techniques to improve the efficiency of your data processing workflows.

1. [**Working with Chunks of Data**](https://www.geeksforgeeks.org/how-to-load-a-massive-file-as-small-chunks-in-pandas/): When dealing with large datasets that may not fit into memory, you can use the chunksize parameter in pd.read\_csv() to load the data in chunks that allows to process the data in manageable pieces **rather than loading everything at once.**
2. [**Optimizing Memory Usage by Changing Data Types**](https://www.geeksforgeeks.org/pandas-memory-management/): By default, Pandas may use larger data types than necessary (e.g., int64 instead of int8), which can lead to inefficient memory usage. Changing the data types with [.astype()](https://www.geeksforgeeks.org/python-pandas-dataframe-astype/) can optimize memory consumption significantly, especially with large datasets.

[For more methods](https://www.geeksforgeeks.org/handling-large-datasets-in-pandas/), click on link.

Refer to [**Pandas Exercises and Programs**](https://www.geeksforgeeks.org/pandas-practice-excercises-questions-and-solutions/) for hands-on practice to reinforce your understanding of key concepts, including data manipulation, cleaning, and analysis.