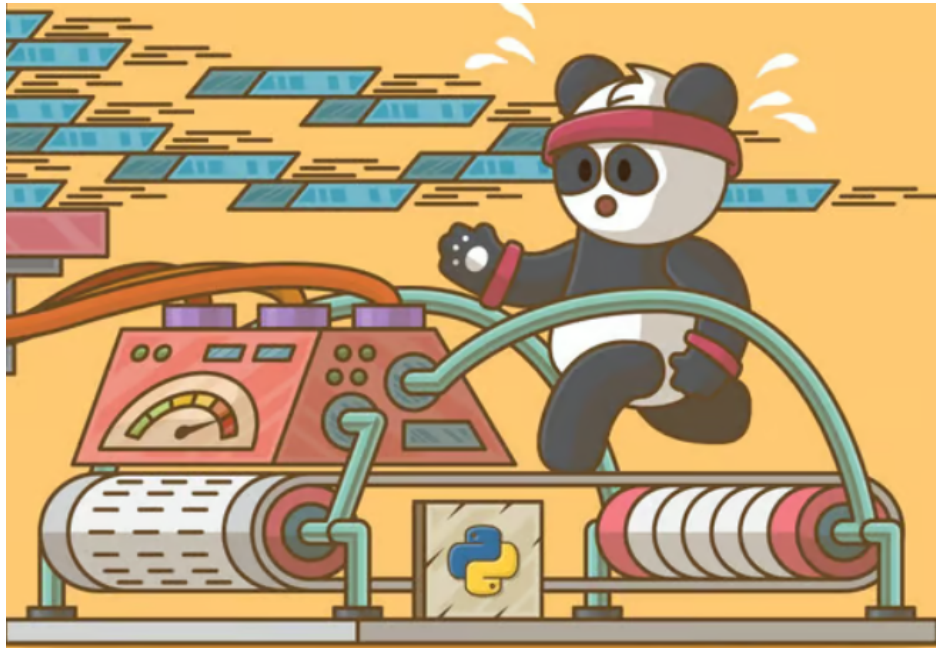


What is MultiIndex in Pandas?

In pandas, a multi-index, also known as a **hierarchical index**, is a way to represent two or more dimensions of data in a single index. This is useful when you have data that can be grouped or categorized by more than one variable.



```
In [1]: import numpy as np
import pandas as pd
```

Series is 1D and DataFrames are 2D objects

- But why?
- And what exactly is index?

```
In [2]: # can we have multiple index? Let's try
index_val = [('cse',2019),('cse',2020),('cse',2021),('cse',2022),('ece',2019),('ece',2020),('ece',2021),('ece',2022)]
data = pd.Series([1,2,3,4,5,6,7,8],index=index_val)
data
```

```
Out[2]: (cse, 2019)    1
(cse, 2020)    2
(cse, 2021)    3
(cse, 2022)    4
(ece, 2019)    5
(ece, 2020)    6
(ece, 2021)    7
(ece, 2022)    8
dtype: int64
```

```
In [3]: # The problem?
#data['cse']
```

The solution : multiindex series-2D(also known as Hierarchical Indexing)

multiple index levels within a single index

```
In [4]: # how to create multiindex object
# 1. pd.MultiIndex.from_tuples()
index_val = [('cse',2019),('cse',2020),('cse',2021),('cse',2022),('ece',2019),('ece',2020),('ece',2021),('ece',2022)]
multiindex = pd.MultiIndex.from_tuples(index_val)
multiindex.levels[0]
```

```
Out[4]: Index(['cse', 'ece'], dtype='object')
```

```
In [5]: # 2. pd.MultiIndex.from_product()
pd.MultiIndex.from_product([['cse', 'ece'], [2019, 2020, 2021, 2022]])
```

```
Out[5]: MultiIndex([('cse', 2019),
                    ('cse', 2020),
                    ('cse', 2021),
                    ('cse', 2022),
                    ('ece', 2019),
                    ('ece', 2020),
                    ('ece', 2021),
                    ('ece', 2022)],
                  )
```

```
In [6]: # creating a series with multiindex object
sample = pd.Series([1,2,3,4,5,6,7,8], index=multiindex)
sample
```

```
Out[6]: cse  2019    1
          2020    2
          2021    3
          2022    4
        ece  2019    5
          2020    6
          2021    7
          2022    8
dtype: int64
```

```
In [7]: # how to fetch items from such a series
sample[('cse', 2022)]
```

```
Out[7]: 4
```

```
In [8]: sample['cse']
```

```
Out[8]: 2019    1
        2020    2
        2021    3
        2022    4
dtype: int64
```

unstack

reshape the given Pandas DataFrame by transposing specified row level to column level

```
In [9]: temp = sample.unstack()
temp
```

```
Out[9]:
```

	2019	2020	2021	2022
cse	1	2	3	4
ece	5	6	7	8

stack

reshapes the given DataFrame by converting the column label to a row index.

```
In [10]: temp.stack()
```

```
Out[10]: cse  2019    1
          2020    2
          2021    3
          2022    4
        ece  2019    5
          2020    6
          2021    7
          2022    8
dtype: int64
```

so why we should study Multi Index

because we can convert any dataframe dimension, including 3D, 4D, 10D, and 20D, to 1Dimension (Series) and 2Dimension (Dataframes).

```
In [11]: # multi index dataframes
branch_df1 = pd.DataFrame(
    [
        [1,2],
        [3,4],
        [5,6],
        [7,8],
        [9,10],
        [11,12],
        [13,14],
        [15,16],
    ],
    index = multiindex,
    columns = ['avg_package', 'students']
)
branch_df1
```

```
Out[11]:
```

		avg_package	students
cse	2019	1	2
	2020	3	4
	2021	5	6
	2022	7	8
ece	2019	9	10
	2020	11	12
	2021	13	14
	2022	15	16

```
In [12]: branch_df1.loc['cse']
```

```
Out[12]:
```

	avg_package	students
2019	1	2
2020	3	4
2021	5	6
2022	7	8

```
In [13]: branch_df1['avg_package']
```

```
Out[13]: cse 2019    1
          2020    3
          2021    5
          2022    7
ece 2019    9
     2020   11
     2021   13
     2022   15
Name: avg_package, dtype: int64
```

```
In [14]: branch_df1['students']
```

```
Out[14]: cse 2019    2
          2020    4
          2021    6
          2022    8
ece 2019   10
     2020   12
     2021   14
     2022   16
Name: students, dtype: int64
```

In [15]: `branch_df1.loc['ece']`

Out[15]:

	avg_package	students
2019	9	10
2020	11	12
2021	13	14
2022	15	16

multiindex df from columns perspective

```
In [16]: branch_df2 = pd.DataFrame(
    [
        [1,2,0,0],
        [3,4,0,0],
        [5,6,0,0],
        [7,8,0,0],
    ],
    index = [2019,2020,2021,2022],
    columns = pd.MultiIndex.from_product([['delhi','mumbai'], ['avg_package','students']])
)
branch_df2
```

Out[16]:

	delhi		mumbai	
	avg_package	students	avg_package	students
2019	1	2	0	0
2020	3	4	0	0
2021	5	6	0	0
2022	7	8	0	0

In [17]: `branch_df2['delhi']`

Out[17]:

	avg_package	students
2019	1	2
2020	3	4
2021	5	6
2022	7	8

In [18]: `branch_df2.loc[2019]`

Out[18]:

delhi	avg_package	1
	students	2
mumbai	avg_package	0
	students	0

Name: 2019, dtype: int64

In [19]: `branch_df2.iloc[1]`

Out[19]:

delhi	avg_package	3
	students	4
mumbai	avg_package	0
	students	0

Name: 2020, dtype: int64

Multiindex df in terms of both cols and index

```
In [20]: branch_df3 = pd.DataFrame(
    [
        [1,2,0,0],
        [3,4,0,0],
        [5,6,0,0],
        [7,8,0,0],
        [9,10,0,0],
        [11,12,0,0],
        [13,14,0,0],
        [15,16,0,0],
    ],
    index = multiindex,
    columns = pd.MultiIndex.from_product([['delhi', 'mumbai'], ['avg_package', 'students']])
)

branch_df3

#here index= multiindex is a name , already we have stored data of ece and cse in above
```

Out[20]:

		delhi		mumbai	
		avg_package	students	avg_package	students
cse	2019	1	2	0	0
	2020	3	4	0	0
	2021	5	6	0	0
	2022	7	8	0	0
ece	2019	9	10	0	0
	2020	11	12	0	0
	2021	13	14	0	0
	2022	15	16	0	0

Stacking and Unstacking

```
In [21]: branch_df1
```

Out[21]:

		avg_package	students
cse	2019	1	2
	2020	3	4
	2021	5	6
	2022	7	8
ece	2019	9	10
	2020	11	12
	2021	13	14
	2022	15	16

```
In [22]: # After applying Unstack
branch_df1.unstack()
```

Out[22]:

		avg_package				students			
		2019	2020	2021	2022	2019	2020	2021	2022
cse		1	3	5	7	2	4	6	8
ece		9	11	13	15	10	12	14	16

```
In [23]: branch_df1.unstack().unstack()
```

```
Out[23]: avg_package  2019  cse    1
              ece    9
              2020  cse    3
              ece   11
              2021  cse    5
              ece   13
              2022  cse    7
              ece   15
students      2019  cse    2
              ece   10
              2020  cse    4
              ece   12
              2021  cse    6
              ece   14
              2022  cse    8
              ece   16
dtype: int64
```

The stack() method

It can be used to move the columns to the index. This means that the columns will become the rows, and the rows will become the columns.

The stack method can be used to move the columns to the index

```
In [24]: # After applying Unstack + stack
branch_df1.unstack().stack()
```

```
Out[24]:
```

		avg_package	students
cse	2019	1	2
	2020	3	4
	2021	5	6
	2022	7	8
ece	2019	9	10
	2020	11	12
	2021	13	14
	2022	15	16

```
In [25]: # applying multiple stack
branch_df1.unstack().stack().stack()
```

```
Out[25]: cse  2019  avg_package  1
              students  2
              2020  avg_package  3
              students  4
              2021  avg_package  5
              students  6
              2022  avg_package  7
              students  8
ece  2019  avg_package  9
              students  10
              2020  avg_package  11
              students  12
              2021  avg_package  13
              students  14
              2022  avg_package  15
              students  16
dtype: int64
```

```
In [26]: # Example : 2
branch_df2
```

Out[26]:

	delhi		mumbai	
	avg_package	students	avg_package	students
2019	1	2	0	0
2020	3	4	0	0
2021	5	6	0	0
2022	7	8	0	0

The Unstack()

It is method can be used to move the index to the columns. This means that the index will become the rows, and the rows will become the columns.

The unstack method can be used to move the index to the columns

```
In [27]: branch_df2.unstack()
```

Out[27]:

delhi	avg_package	2019	1
		2020	3
		2021	5
		2022	7
	students	2019	2
		2020	4
		2021	6
		2022	8
mumbai	avg_package	2019	0
		2020	0
		2021	0
		2022	0
	students	2019	0
		2020	0
		2021	0
		2022	0

dtype: int64

```
In [28]: branch_df2.stack()
```

Out[28]:

		delhi	mumbai
2019	avg_package	1	0
	students	2	0
2020	avg_package	3	0
	students	4	0
2021	avg_package	5	0
	students	6	0
2022	avg_package	7	0
	students	8	0

```
branch_df2.stack().stack()
```

```

2019  avg_package  delhi    1
      avg_package  mumbai   0
      students    delhi    2
      students    mumbai   0
2020  avg_package  delhi    3
      avg_package  mumbai   0
      students    delhi    4
      students    mumbai   0
2021  avg_package  delhi    5
      avg_package  mumbai   0
      students    delhi    6
      students    mumbai   0
2022  avg_package  delhi    7
      avg_package  mumbai   0
      students    delhi    8
      students    mumbai   0
dtype: int64

```

```
# Working on 4D data
branch_df3
```

		delhi		mumbai	
		avg_package	students	avg_package	students
cse	2019	1	2	0	0
	2020	3	4	0	0
	2021	5	6	0	0
	2022	7	8	0	0
ece	2019	9	10	0	0
	2020	11	12	0	0
	2021	13	14	0	0
	2022	15	16	0	0

```
branch_df3.stack()
```

			delhi	mumbai
cse	2019	avg_package	1	0
		students	2	0
	2020	avg_package	3	0
		students	4	0
	2021	avg_package	5	0
		students	6	0
2022	avg_package	7	0	
	students	8	0	
ece	2019	avg_package	9	0
		students	10	0
	2020	avg_package	11	0
		students	12	0
	2021	avg_package	13	0
		students	14	0
2022	avg_package	15	0	
	students	16	0	


```
In [32]: branch_df3.stack().stack()
```

```
Out[32]: cse  2019  avg_package  delhi      1
          Mumbai      0
          students  delhi      2
          Mumbai      0
          2020  avg_package  delhi      3
          Mumbai      0
          students  delhi      4
          Mumbai      0
          2021  avg_package  delhi      5
          Mumbai      0
          students  delhi      6
          Mumbai      0
          2022  avg_package  delhi      7
          Mumbai      0
          students  delhi      8
          Mumbai      0
ece  2019  avg_package  delhi      9
          Mumbai      0
          students  delhi     10
          Mumbai      0
          2020  avg_package  delhi     11
          Mumbai      0
          students  delhi     12
          Mumbai      0
          2021  avg_package  delhi     13
          Mumbai      0
          students  delhi     14
          Mumbai      0
          2022  avg_package  delhi     15
          Mumbai      0
          students  delhi     16
          Mumbai      0
dtype: int64
```

```
In [33]: # Unstacking on 4D data
branch_df3.unstack()
```

Out[33]:

	delhi								mumbai							
	avg_package				students				avg_package				students			
	2019	2020	2021	2022	2019	2020	2021	2022	2019	2020	2021	2022	2019	2020	2021	2022
cse	1	3	5	7	2	4	6	8	0	0	0	0	0	0	0	0
ece	9	11	13	15	10	12	14	16	0	0	0	0	0	0	0	0

```
In [34]: branch_df3.unstack().unstack()
```

```
Out[34]: delhi    avg_package    2019    cse      1
                                     ece      9
                                     2020    cse      3
                                     ece     11
                                     2021    cse      5
                                     ece     13
                                     2022    cse      7
                                     ece     15
          students    2019    cse      2
                                     ece     10
                                     2020    cse      4
                                     ece     12
                                     2021    cse      6
                                     ece     14
                                     2022    cse      8
                                     ece     16
          mumbai    avg_package    2019    cse      0
                                     ece      0
                                     2020    cse      0
                                     ece      0
                                     2021    cse      0
                                     ece      0
                                     2022    cse      0
                                     ece      0
          students    2019    cse      0
                                     ece      0
                                     2020    cse      0
                                     ece      0
                                     2021    cse      0
                                     ece      0
                                     2022    cse      0
                                     ece      0

dtype: int64
```

Working with multiindex dataframes

```
In [35]: # Multi index dataframes works same as normal dataframes
branch_df3
```

```
Out[35]:
```

		delhi		mumbai	
		avg_package	students	avg_package	students
cse	2019	1	2	0	0
	2020	3	4	0	0
	2021	5	6	0	0
	2022	7	8	0	0
ece	2019	9	10	0	0
	2020	11	12	0	0
	2021	13	14	0	0
	2022	15	16	0	0

```
In [36]: # head and tail
branch_df3.head()
```

```
Out[36]:
```

		delhi		mumbai	
		avg_package	students	avg_package	students
cse	2019	1	2	0	0
	2020	3	4	0	0
	2021	5	6	0	0
	2022	7	8	0	0
ece	2019	9	10	0	0

```
In [37]: # shape
branch_df3.shape
```

```
Out[37]: (8, 4)
```

```
In [38]: # info
branch_df3.info()

<class 'pandas.core.frame.DataFrame'>
MultiIndex: 8 entries, ('cse', 2019) to ('ece', 2022)
Data columns (total 4 columns):
#   Column                                Non-Null Count  Dtype
---  ---                                ---
0   (delhi, avg_package)                  8 non-null      int64
1   (delhi, students)                   8 non-null      int64
2   (mumbai, avg_package)                8 non-null      int64
3   (mumbai, students)                  8 non-null      int64
dtypes: int64(4)
memory usage: 932.0+ bytes
```

```
In [39]: # duplicated -> isnull
branch_df3.duplicated()
```

```
Out[39]: cse  2019    False
          2020    False
          2021    False
          2022    False
ece  2019    False
          2020    False
          2021    False
          2022    False
dtype: bool
```

```
In [40]: branch_df3.isnull()
```

```
Out[40]:
```

		delhi		mumbai	
		avg_package	students	avg_package	students
cse	2019	False	False	False	False
	2020	False	False	False	False
	2021	False	False	False	False
	2022	False	False	False	False
ece	2019	False	False	False	False
	2020	False	False	False	False
	2021	False	False	False	False
	2022	False	False	False	False

```
In [41]: # Extracting rows single
branch_df3.loc[('cse', 2022)]
```

```
Out[41]: delhi    avg_package    7
          students    8
mumbai  avg_package    0
          students    0
Name: (cse, 2022), dtype: int64
```

```
In [42]: # Extracting multiple rows
branch_df3.loc[('cse', 2019):('ece', 2020):2]
```

```
Out[42]:
```

		delhi		mumbai	
		avg_package	students	avg_package	students
cse	2019	1	2	0	0
	2021	5	6	0	0
ece	2019	9	10	0	0

```
In [43]: # Using iloc
branch_df3.iloc[0:5:2]
```

Out[43]:

		delhi		mumbai	
		avg_package	students	avg_package	students
cse	2019	1	2	0	0
	2021	5	6	0	0
	2022	8	8	0	0
ece	2019	9	10	0	0
	2020	11	12	0	0
	2021	13	14	0	0
	2022	15	16	0	0

```
In [44]: # Extracting single columns
branch_df3['delhi']['students']
```

Out[44]:

```
cse 2019    2
     2020    4
     2021    6
     2022    8
ece  2019   10
     2020   12
     2021   14
     2022   16
Name: students, dtype: int64
```

```
In [45]: # we want to extract delhi - students , mumbai - avg_package
branch_df3
```

Out[45]:

		delhi		mumbai	
		avg_package	students	avg_package	students
cse	2019	1	2	0	0
	2020	3	4	0	0
	2021	5	6	0	0
	2022	7	8	0	0
ece	2019	9	10	0	0
	2020	11	12	0	0
	2021	13	14	0	0
	2022	15	16	0	0

```
In [46]: #here [:] all rows ,
#columns : delhi=avg_package[0],students[1],mumbai=avg_package[2],students[3]
branch_df3.iloc[:,1:3]
```

Out[46]:

		delhi	mumbai
		students	avg_package
cse	2019	2	0
	2020	4	0
	2021	6	0
	2022	8	0
ece	2019	10	0
	2020	12	0
	2021	14	0
	2022	16	0

```
In [47]: # Extracting both rows and columns
branch_df3.iloc[[0,4],[1,2]]
```

Out[47]:

		delhi	mumbai
		students	avg_package
cse	2019	2	0
ece	2019	10	0

```
In [48]: # sort index
# both -> descending -> diff order
# based on one level

branch_df3
```

Out[48]:

		delhi		mumbai	
		avg_package	students	avg_package	students
cse	2019	1	2	0	0
	2020	3	4	0	0
	2021	5	6	0	0
	2022	7	8	0	0
ece	2019	9	10	0	0
	2020	11	12	0	0
	2021	13	14	0	0
	2022	15	16	0	0

```
In [49]: branch_df3.sort_index(ascending=False)
```

Out[49]:

		delhi		mumbai	
		avg_package	students	avg_package	students
ece	2022	15	16	0	0
	2021	13	14	0	0
	2020	11	12	0	0
	2019	9	10	0	0
cse	2022	7	8	0	0
	2021	5	6	0	0
	2020	3	4	0	0
	2019	1	2	0	0

```
In [50]: # if we want year in descending order
branch_df3.sort_index(ascending=False, True)
```

Out[50]:

		delhi		mumbai	
		avg_package	students	avg_package	students
ece	2019	9	10	0	0
	2020	11	12	0	0
	2021	13	14	0	0
	2022	15	16	0	0
cse	2019	1	2	0	0
	2020	3	4	0	0
	2021	5	6	0	0
	2022	7	8	0	0

```
In [51]: # multiindex dataframe(col) -> transpose
branch_df3.transpose()
```

Out[51]:

		cse				ece			
		2019	2020	2021	2022	2019	2020	2021	2022
delhi	avg_package	1	3	5	7	9	11	13	15
	students	2	4	6	8	10	12	14	16
mumbai	avg_package	0	0	0	0	0	0	0	0
	students	0	0	0	0	0	0	0	0

```
In [52]: # swaplevel
branch_df3
```

Out[52]:

		delhi		mumbai	
		avg_package	students	avg_package	students
cse	2019	1	2	0	0
	2020	3	4	0	0
	2021	5	6	0	0
	2022	7	8	0	0
ece	2019	9	10	0	0
	2020	11	12	0	0
	2021	13	14	0	0
	2022	15	16	0	0

```
In [53]: # On rows
branch_df3.swaplevel()
```

Out[53]:

		delhi		mumbai	
		avg_package	students	avg_package	students
2019	cse	1	2	0	0
2020	cse	3	4	0	0
2021	cse	5	6	0	0
2022	cse	7	8	0	0
2019	ece	9	10	0	0
2020	ece	11	12	0	0
2021	ece	13	14	0	0
2022	ece	15	16	0	0

```
In [54]: # on columns
branch_df3.swaplevel(axis=1)
```

Out[54]:

		avg_package	students	avg_package	students
		delhi	delhi	mumbai	mumbai
cse	2019	1	2	0	0
	2020	3	4	0	0
	2021	5	6	0	0
	2022	7	8	0	0
ece	2019	9	10	0	0
	2020	11	12	0	0
	2021	13	14	0	0
	2022	15	16	0	0

Long(Tall) Vs Wide data

“Long” format

country	year	metric
x	1960	10
x	1970	13
x	2010	15
y	1960	20
y	1970	23
y	2010	25
z	1960	30
z	1970	33
z	2010	35

“Wide” format

country	yr1960	yr1970	yr2010
x	10	13	15
y	20	23	25
z	30	33	35

Wide format is where we have a single row for every data point with multiple columns to hold the values of various attributes.

Long format is where, for each data point we have as many rows as the number of attributes and each row contains the value of a particular attribute for a given data point.

Melt -- Converting wide data to long Data.

```
In [55]: # melt -> simple example branch
# wide to Long
pd.DataFrame({'cse':[120]})
```

```
Out[55]:
   cse
0  120
```

```
In [56]: pd.DataFrame({'cse':[120]}).melt()
```

```
Out[56]:
  variable  value
0      cse    120
```

```
In [57]: # melt -> branch with year
pd.DataFrame({'cse':[120], 'ece':[100], 'mech':[50]}).melt()
```

```
Out[57]:
  variable  value
0      cse    120
1      ece    100
2     mech     50
```

```
In [58]: # we can name the variable and value
pd.DataFrame({'cse':[120], 'ece':[100], 'mech':[50]}).melt(var_name='branch', value_name='num_students')
```

Out[58]:

	branch	num_students
0	cse	120
1	ece	100
2	mech	50

```
In [59]: pd.DataFrame(
    {
        'branch':['cse','ece','mech'],
        '2020':[100,150,60],
        '2021':[120,130,80],
        '2022':[150,140,70]
    }
)
```

Out[59]:

	branch	2020	2021	2022
0	cse	100	120	150
1	ece	150	130	140
2	mech	60	80	70

```
In [60]: pd.DataFrame(
    {
        'branch':['cse','ece','mech'],
        '2020':[100,150,60],
        '2021':[120,130,80],
        '2022':[150,140,70]
    }
).melt()
```

Out[60]:

	variable	value
0	branch	cse
1	branch	ece
2	branch	mech
3	2020	100
4	2020	150
5	2020	60
6	2021	120
7	2021	130
8	2021	80
9	2022	150
10	2022	140
11	2022	70


```
In [61]: # dont include 'branch' to rows
pd.DataFrame(
    {
        'branch': ['cse', 'ece', 'mech'],
        '2020': [100, 150, 60],
        '2021': [120, 130, 80],
        '2022': [150, 140, 70]
    }
).melt(id_vars=['branch'])
```

Out[61]:

	branch	variable	value
0	cse	2020	100
1	ece	2020	150
2	mech	2020	60
3	cse	2021	120
4	ece	2021	130
5	mech	2021	80
6	cse	2022	150
7	ece	2022	140
8	mech	2022	70

the **melt()** method is used to reshape a DataFrame from wide to long format. This means that the columns of the DataFrame are converted into rows, and the values in the columns are converted into columns.

```
In [62]: # adding variable and value names.
pd.DataFrame(
    {
        'branch': ['cse', 'ece', 'mech'],
        '2020': [100, 150, 60],
        '2021': [120, 130, 80],
        '2022': [150, 140, 70]
    }
).melt(id_vars=['branch'], var_name='year', value_name='students')
```

Out[62]:

	branch	year	students
0	cse	2020	100
1	ece	2020	150
2	mech	2020	60
3	cse	2021	120
4	ece	2021	130
5	mech	2021	80
6	cse	2022	150
7	ece	2022	140
8	mech	2022	70

```
In [63]: # melt ---> Real world examples.

deaths = pd.read_csv("time_series_covid19_deaths_global.csv")
confirm = pd.read_csv("time_series_covid19_confirmed_global.csv")
```

In [64]: deaths.head(2)

Out[64]:

	Province/State	Country/Region	Lat	Long	1/22/20	1/23/20	1/24/20	1/25/20	1/26/20	1/27/20	...	12/24/22	12/25/22	12/26/22	12/27/22
0	NaN	Afghanistan	33.93911	67.709953	0	0	0	0	0	0	...	7845	7846	7846	7846
1	NaN	Albania	41.15330	20.168300	0	0	0	0	0	0	...	3595	3595	3595	3595

2 rows × 1081 columns



In [65]: `deaths.shape`

Out[65]: (289, 1081)

In [66]: `deaths = deaths.melt(id_vars=['Province/State', 'Country/Region', 'Lat', 'Long'], var_name='date', value_name='no. of deaths')`

In [67]: *# After converting columns into rows,
which is converting wide format to long format using 'melt'*
`deaths.shape`

Out[67]: (311253, 6)

In [75]: `deaths.head()`

Out[75]:


	Province/State	Country/Region	Lat	Long	date	no. of deaths
0	NaN	Afghanistan	33.93911	67.709953	1/22/20	0
1	NaN	Albania	41.15330	20.168300	1/22/20	0
2	NaN	Algeria	28.03390	1.659600	1/22/20	0
3	NaN	Andorra	42.50630	1.521800	1/22/20	0
4	NaN	Angola	-11.20270	17.873900	1/22/20	0

In [68]: `confirm.head(2)`

Out[68]:

	Province/State	Country/Region	Lat	Long	1/22/20	1/23/20	1/24/20	1/25/20	1/26/20	1/27/20	...	12/24/22	12/25/22	12/26/22	12/27/22
0	NaN	Afghanistan	33.93911	67.709953	0	0	0	0	0	0	...	207310	207399	207438	207460
1	NaN	Albania	41.15330	20.168300	0	0	0	0	0	0	...	333749	333749	333751	333751

2 rows × 1081 columns



In [69]: `confirm.shape`

Out[69]: (289, 1081)

In [73]: `confirm = confirm.melt(id_vars=['Province/State', 'Country/Region', 'Lat', 'Long'], var_name='date', value_name='no. of confi')`

In [76]: `confirm.head()`

Out[76]:

	Province/State	Country/Region	Lat	Long	date	no. of confirmed
0	NaN	Afghanistan	33.93911	67.709953	1/22/20	0
1	NaN	Albania	41.15330	20.168300	1/22/20	0
2	NaN	Algeria	28.03390	1.659600	1/22/20	0
3	NaN	Andorra	42.50630	1.521800	1/22/20	0
4	NaN	Angola	-11.20270	17.873900	1/22/20	0

In [74]: *# After converting columns into rows,
which is converting wide format to long format using 'melt'*
`confirm.shape`

Out[74]: (311253, 6)

```
In [77]: # Now merge both data frames as per desire
confirm.merge(deaths, on = ['Province/State', 'Country/Region', 'Lat', 'Long', 'date'])
```

Out[77]:

	Province/State	Country/Region	Lat	Long	date	no. of confirmed	no. of deaths
0	NaN	Afghanistan	33.939110	67.709953	1/22/20	0	0
1	NaN	Albania	41.153300	20.168300	1/22/20	0	0
2	NaN	Algeria	28.033900	1.659600	1/22/20	0	0
3	NaN	Andorra	42.506300	1.521800	1/22/20	0	0
4	NaN	Angola	-11.202700	17.873900	1/22/20	0	0
...
311248	NaN	West Bank and Gaza	31.952200	35.233200	1/2/23	703228	5708
311249	NaN	Winter Olympics 2022	39.904200	116.407400	1/2/23	535	0
311250	NaN	Yemen	15.552727	48.516388	1/2/23	11945	2159
311251	NaN	Zambia	-13.133897	27.849332	1/2/23	334661	4024
311252	NaN	Zimbabwe	-19.015438	29.154857	1/2/23	259981	5637

311253 rows × 7 columns

```
In [80]: desired columns
deaths, on = ['Province/State', 'Country/Region', 'Lat', 'Long', 'date'][['Country/Region', 'date', 'no. of confirmed', 'no. of
```

Out[80]:

	Country/Region	date	no. of confirmed	no. of deaths
0	Afghanistan	1/22/20	0	0
1	Albania	1/22/20	0	0
2	Algeria	1/22/20	0	0
3	Andorra	1/22/20	0	0
4	Angola	1/22/20	0	0
...
311248	West Bank and Gaza	1/2/23	703228	5708
311249	Winter Olympics 2022	1/2/23	535	0
311250	Yemen	1/2/23	11945	2159
311251	Zambia	1/2/23	334661	4024
311252	Zimbabwe	1/2/23	259981	5637

311253 rows × 4 columns

Pivot table -- Converting Long data to wide data.

the **Pivot table** takes simple column wise data as input, and groups as the entire Into 2 dimensional table that provides a multi dimensional summarization of the data.

Pivot table generally used on categorical data

```
In [81]: import seaborn as sns
```

```
In [83]: df = sns.load_dataset('tips')
df.head()
```

Out[83]:

	total_bill	tip	sex	smoker	day	time	size
0	16.99	1.01	Female	No	Sun	Dinner	2
1	10.34	1.66	Male	No	Sun	Dinner	3
2	21.01	3.50	Male	No	Sun	Dinner	3
3	23.68	3.31	Male	No	Sun	Dinner	2
4	24.59	3.61	Female	No	Sun	Dinner	4

```
In [85]: # On gender basis average total bill
df.groupby('sex')['total_bill'].mean()
```

```
Out[85]: sex
Male      20.744076
Female    18.056897
Name: total_bill, dtype: float64
```

```
In [88]: # On gender basis. Who smokes more? On average.
df.groupby(['sex', 'smoker'])['total_bill'].mean().unstack()
```

```
Out[88]:
```

	smoker	Yes	No
	sex		
	Male	22.284500	19.791237
	Female	17.977879	18.105185

```
In [89]: # Using Pivot table method
df.pivot_table(index='sex', columns='smoker', values='total_bill')
```

```
Out[89]:
```

	smoker	Yes	No
	sex		
	Male	22.284500	19.791237
	Female	17.977879	18.105185

```
In [90]: # Aggregate function.
# Print, the total amount smokers of the bill, Not mean Or average,
df.pivot_table(index='sex', columns='smoker', values='total_bill', aggfunc='sum')
```

```
Out[90]:
```

	smoker	Yes	No
	sex		
	Male	1337.07	1919.75
	Female	593.27	977.68

```
In [91]: # count of people
df.pivot_table(index='sex', columns='smoker', values='total_bill', aggfunc='count')
```

```
Out[91]:
```

	smoker	Yes	No
	sex		
	Male	60	97
	Female	33	54

```
In [92]: # standard deviation
df.pivot_table(index='sex', columns='smoker', values='total_bill', aggfunc='std')
```

```
Out[92]:
```

	smoker	Yes	No
	sex		
	Male	9.911845	8.726566
	Female	9.189751	7.286455

In [93]: *# ALL columns together --- gives average*

```
df.pivot_table(index='sex',columns='smoker')
```

Out[93]:

	size		tip		total_bill	
	Yes	No	Yes	No	Yes	No
sex						
Male	2.500000	2.711340	3.051167	3.113402	22.284500	19.791237
Female	2.242424	2.592593	2.931515	2.773519	17.977879	18.105185

In [95]: *# single column*

```
df.pivot_table(index='sex',columns='smoker')['tip']
```

Out[95]:

	smoker Yes	No
sex		
Male	3.051167	3.113402
Female	2.931515	2.773519

In [96]: `df.pivot_table(index='sex',columns='smoker')['size']`

Out[96]:

	smoker Yes	No
sex		
Male	2.500000	2.711340
Female	2.242424	2.592593

In [98]: *# Multi dimensional -5D*
`df.head(2)`

Out[98]:

	total_bill	tip	sex	smoker	day	time	size
0	16.99	1.01	Female	No	Sun	Dinner	2
1	10.34	1.66	Male	No	Sun	Dinner	3

In [100]: *# 5D - 5 Dimensional data*

```
df.pivot_table(index=['sex','smoker'],columns=['day','time'],values='total_bill')
```

Out[100]:

	day	Thur		Fri		Sat	Sun
	time	Lunch	Dinner	Lunch	Dinner	Dinner	Dinner
sex	smoker						
Male	Yes	19.171000	NaN	11.386667	25.892	21.837778	26.141333
	No	18.486500	NaN	NaN	17.475	19.929063	20.403256
Female	Yes	19.218571	NaN	13.260000	12.200	20.266667	16.540000
	No	15.899167	18.78	15.980000	22.750	19.003846	20.824286

In [102]: `df.pivot_table(index=['sex','smoker'],columns=['day','time'])`

Out[102]:

						tip						total_bill					
		Fri		Sat		Sun		Thur		Fri		Sat		Sun		Thur	
		Dinner	Lunch	Dinner	Dinner	Dinner	Lunch	Dinner	Lunch	Dinner	Lunch	Dinner	Dinner	Dinner	Dinner	Lunch	Dinner
		NaN	1.666667	2.4	2.629630	2.600000	3.058000	NaN	1.90	3.246	2.879259	3.521333	19.171000	NaN	11.386667	25.892	21.837778
		NaN	NaN	2.0	2.656250	2.883721	2.941500	NaN	NaN	2.500	3.256563	3.115349	18.486500	NaN	NaN	17.475	19.929063
		NaN	2.000000	2.0	2.200000	2.500000	2.990000	NaN	2.66	2.700	2.868667	3.500000	19.218571	NaN	13.260000	12.200	20.266667
		2.0	3.000000	2.0	2.307692	3.071429	2.437083	3.0	3.00	3.250	2.724615	3.329286	15.899167	18.78	15.980000	22.750	19.003846

```
In [103]: df.pivot_table(index=['sex', 'smoker'], columns=['day', 'time'], aggfunc={'size': 'mean', 'tip': 'max', 'total_bill': 'sum'})
```

```
Out[103]:
```

		size						tip				total_bill					
		day	Thur	Fri		Sat		Sun	Thur	Fri		Sat	Sun	Thur	Fri		
		time	Lunch	Dinner	Lunch	Dinner	Dinner	Dinner	Lunch	Dinner	Lunch	Dinner	Dinner	Dinner	Lunch	Dinner	Lunch
sex	smoker																
Male	Yes		2.300000	NaN	1.666667	2.4	2.629630	2.600000	5.00	NaN	2.20	4.73	10.00	6.5	191.71	0.00	34.16
	No		2.500000	NaN	NaN	2.0	2.656250	2.883721	6.70	NaN	NaN	3.50	9.00	6.0	369.73	0.00	0.00
Female	Yes		2.428571	NaN	2.000000	2.0	2.200000	2.500000	5.00	NaN	3.48	4.30	6.50	4.0	134.53	0.00	39.78
	No		2.500000	2.0	3.000000	2.0	2.307692	3.071429	5.17	3.0	3.00	3.25	4.67	5.2	381.58	18.78	15.98

```
In [106]: # Margins.
df.pivot_table(index='sex', columns='smoker', values='total_bill', aggfunc='sum', margins=True)
```

```
Out[106]:
```

smoker	Yes	No	All
sex			
Male	1337.07	1919.75	3256.82
Female	593.27	977.68	1570.95
All	1930.34	2897.43	4827.77

```
In [108]: # Plotting Graphs.
expense = pd.read_csv("expense_data.csv")
```

```
In [110]: expense.head(2)
```

```
Out[110]:
```

	Date	Account	Category	Subcategory	Note	INR	Income/Expense	Note.1	Amount	Currency	Account.1
0	3/2/2022 10:11	CUB - online payment	Food	NaN	Brownie	50.0	Expense	NaN	50.0	INR	50.0
1	3/2/2022 10:11	CUB - online payment	Other	NaN	To lend people	300.0	Expense	NaN	300.0	INR	300.0

```
In [115]: # Categories
expense['Category'].value_counts()
```

```
Out[115]: Food      156
Other      60
Transportation  31
Apparel     7
Household    6
Allowance    6
Social Life   5
Education     1
Salary        1
Self-development  1
Beauty        1
Gift          1
Petty cash    1
Name: Category, dtype: int64
```

In [117]: `expense.info()`

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 277 entries, 0 to 276
Data columns (total 11 columns):
#   Column                Non-Null Count  Dtype
---  ---
0   Date                  277 non-null   object
1   Account               277 non-null   object
2   Category              277 non-null   object
3   Subcategory           0 non-null     float64
4   Note                  273 non-null   object
5   INR                   277 non-null   float64
6   Income/Expense        277 non-null   object
7   Note.1                0 non-null     float64
8   Amount                277 non-null   float64
9   Currency              277 non-null   object
10  Account.1             277 non-null   float64
dtypes: float64(5), object(6)
memory usage: 23.9+ KB
```

In [119]: `# Converting integer to daytime format`
`expense['Date'] = pd.to_datetime(expense['Date'])`

In [121]: `expense.info()`

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 277 entries, 0 to 276
Data columns (total 11 columns):
#   Column                Non-Null Count  Dtype
---  ---
0   Date                  277 non-null   datetime64[ns]
1   Account               277 non-null   object
2   Category              277 non-null   object
3   Subcategory           0 non-null     float64
4   Note                  273 non-null   object
5   INR                   277 non-null   float64
6   Income/Expense        277 non-null   object
7   Note.1                0 non-null     float64
8   Amount                277 non-null   float64
9   Currency              277 non-null   object
10  Account.1             277 non-null   float64
dtypes: datetime64[ns](1), float64(5), object(5)
memory usage: 23.9+ KB
```

In [123]: `# Extracting month from the date column`

`expense['Date'].dt.month_name()`

Out[123]:

0	March
1	March
2	March
3	March
4	March
...	
272	November
273	November
274	November
275	November
276	November

Name: Date, Length: 277, dtype: object

In [124]: `expense['month'] = expense['Date'].dt.month_name()`

In [125]: `expense.head(2)`

Out[125]:

	Date	Account	Category	Subcategory	Note	INR	Income/Expense	Note.1	Amount	Currency	Account.1	month
0	2022-03-02 10:11:00	CUB - online payment	Food	NaN	Brownie	50.0	Expense	NaN	50.0	INR	50.0	March
1	2022-03-02 10:11:00	CUB - online payment	Other	NaN	To lend people	300.0	Expense	NaN	300.0	INR	300.0	March

```
In [126]: # Using pivot table
expense.pivot_table(index='month', columns='Category', values='INR', aggfunc='sum')
```

Out[126]:

Category	Allowance	Apparel	Beauty	Education	Food	Gift	Household	Other	Petty cash	Salary	Self-development	Social Life	Transportation
month													
December	11000.0	2590.0	196.0	NaN	6440.72	NaN	4800.0	1790.0	NaN	NaN	400.0	513.72	914.0
February	NaN	798.0	NaN	NaN	5579.85	NaN	2808.0	20000.0	NaN	NaN	NaN	1800.00	5078.8
January	1000.0	NaN	NaN	1400.0	9112.51	NaN	4580.0	13178.0	NaN	8000.0	NaN	200.00	2850.0
March	NaN	NaN	NaN	NaN	195.00	NaN	NaN	900.0	NaN	NaN	NaN	NaN	30.0
November	2000.0	NaN	NaN	NaN	3174.40	115.0	NaN	2000.0	3.0	NaN	NaN	NaN	331.0

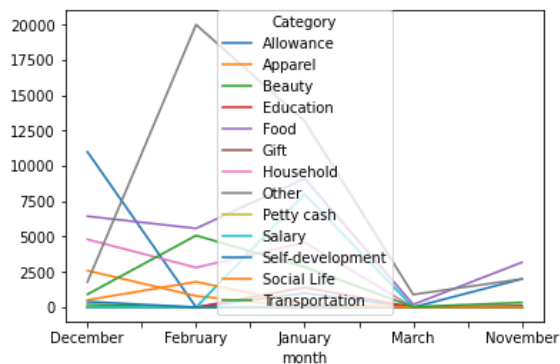
```
In [128]: # fill values of NAN
expense.pivot_table(index='month', columns='Category', values='INR', aggfunc='sum', fill_value=0)
```

Out[128]:

Category	Allowance	Apparel	Beauty	Education	Food	Gift	Household	Other	Petty cash	Salary	Self-development	Social Life	Transportation
month													
December	11000	2590	196	0	6440.72	0	4800	1790	0	0	400	513.72	914.0
February	0	798	0	0	5579.85	0	2808	20000	0	0	0	1800.00	5078.8
January	1000	0	0	1400	9112.51	0	4580	13178	0	8000	0	200.00	2850.0
March	0	0	0	0	195.00	0	0	900	0	0	0	0.00	30.0
November	2000	0	0	0	3174.40	115	0	2000	3	0	0	0.00	331.0

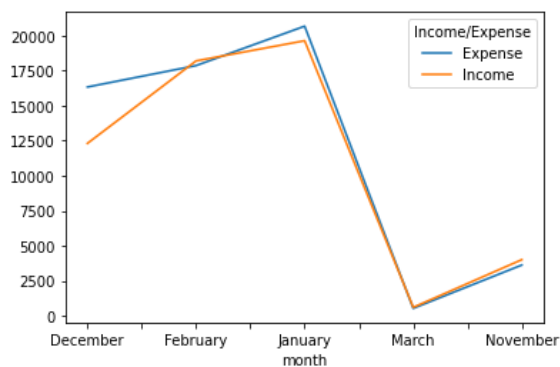
```
In [131]: # plot
expense.pivot_table(index='month', columns='Category', values='INR', aggfunc='sum', fill_value=0).plot()
```

Out[131]: <AxesSubplot:xlabel='month'>



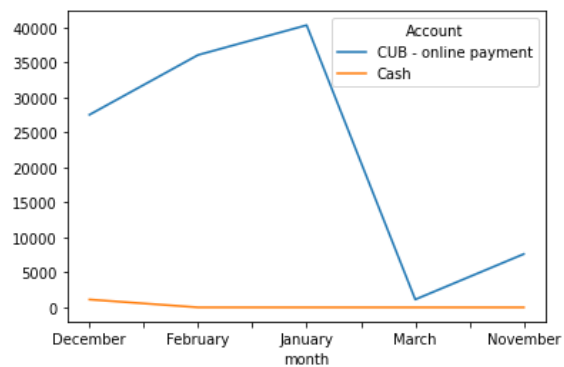
```
In [132]: expense.pivot_table(index='month', columns='Income/Expense', values='INR', aggfunc='sum', fill_value=0).plot()
```

Out[132]: <AxesSubplot:xlabel='month'>




```
In [133]: expense.pivot_table(index = 'month', columns = 'Account', values = 'INR', aggfunc = 'sum', fill_value = 0).plot()
```

```
Out[133]: <AxesSubplot:xlabel='month'>
```



```
In [ ]:
```