```
import numpy as np
import pandas as pd
```

Series is 1D and DataFrames are 2D objects

- But why?
- And what exactly is index?

```
# 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)]
a = pd.Series([1,2,3,4,5,6,7,8],index=index_val)
a
```

```
(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
```

```
# The problem?
a['cse']
```

```
---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
/usr/local/lib/python3.8/dist-packages/pandas/core/indexes/base.py in get_loc(self, key, method,
tolerance)
   3360             try:
-> 3361                 return self._engine.get_loc(casted_key)
   3362             except KeyError as err:

                              5 frames
pandas/_libs/hashtable_class_helper.pxi in pandas._libs.hashtable.PyObjectHashTable.get_item()

pandas/_libs/hashtable_class_helper.pxi in pandas._libs.hashtable.PyObjectHashTable.get_item()

KeyError: 'cse'

The above exception was the direct cause of the following exception:

KeyError                                  Traceback (most recent call last)
/usr/local/lib/python3.8/dist-packages/pandas/core/indexes/base.py in get_loc(self, key, method,
tolerance)
   3361                 return self._engine.get_loc(casted_key)
   3362             except KeyError as err:
-> 3363                 raise KeyError(key) from err
   3364
   3365         if is_scalar(key) and isna(key) and not self.hasnans:

KeyError: 'cse'
```

```
# The solution -> multiindex series(also known as Hierarchical Indexing)
# multiple index levels within a single index
```

```
# 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[1]
# 2. pd.MultiIndex.from_product()
pd.MultiIndex.from_product([['cse','ece'],[2019,2020,2021,2022]])
```

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

```
# level inside multiindex object
```

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

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

```
# how to fetch items from such a series
s['cse']
```

```
2019    1
2020    2
2021    3
2022    4
dtype: int64
```

```
# a logical question to ask
```

```
# unstack
temp = s.unstack()
temp
```

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

```
# stack
temp.stack()
```

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

```
# Then what was the point of multiindex series?
```

```
# multiindex dataframe
```

```
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
```

		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

```
branch_df1['students']
```

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

```
# Are columns really different from index?
```

```
# multiindex df from columns perspective
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
```

	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

```
branch_df2.loc[2019]
```

```
delhi   avg_package    1
        students       2
mumbai  avg_package    0
        students       0
Name: 2019, dtype: int64
```

```
# Multiindex df in terms of both cols and index

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
```

		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

```
branch_df3.stack().stack()
```

```
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
```

Working with multiindex dataframes

```
# head and tail
branch_df3.head()
# shape
branch_df3.shape
# info
branch_df3.info()
# duplicated -> isnull
branch_df3.duplicated()
branch_df3.isnull()
```

```
<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
```

		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

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

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

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

		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

```
# using iloc
branch_df3.iloc[0:5:2]
```

		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

```
# Extracting cols
branch_df3['delhi']['students']
```

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

```
branch_df3.iloc[:,1:3]
```

		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

```
# Extracting both
branch_df3.iloc[[0,4],[1,2]]
```

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

```
# sort index
# both -> descending -> diff order
# based on one level
branch_df3.sort_index(ascending=False)
branch_df3.sort_index(ascending=[False,True])
branch_df3.sort_index(level=0,ascending=[False])
```

		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

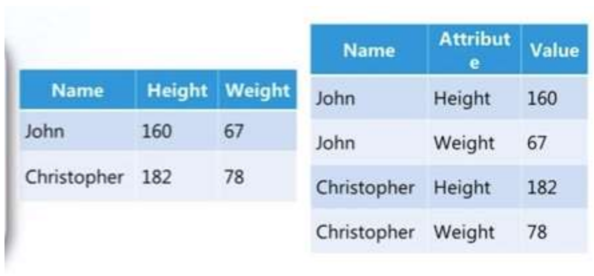
```
# multiindex dataframe(col) -> transpose
branch_df3.transpose()
```

		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

```
# swaplevel
branch_df3.swaplevel(axis=1)
```

		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

Long Vs Wide Data



Name	Height	Weight
John	160	67
Christopher	182	78

Name	Attribute	Value
John	Height	160
John	Weight	67
Christopher	Height	182
Christopher	Weight	78

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

	variable	value
0	cse	120

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

	branch	num_students
0	cse	120
1	ece	100
2	mech	50

```
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')
```

	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

```
# melt -> real world example
death = pd.read_csv('/content/time_series_covid19_deaths_global.csv')
confirm = pd.read_csv('/content/time_series_covid19_confirmed_global.csv')
```

```
death.head()
```

	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/
0	NaN	Afghanistan	33.93911	67.709953	0	0	0	0	0	0	...	7845	7846	
1	NaN	Albania	41.15330	20.168300	0	0	0	0	0	0	...	3595	3595	
2	NaN	Algeria	28.03390	1.659600	0	0	0	0	0	0	...	6881	6881	
3	NaN	Andorra	42.50630	1.521800	0	0	0	0	0	0	...	165	165	
4	NaN	Angola	-11.20270	17.873900	0	0	0	0	0	0	...	1928	1928	

5 rows × 1081 columns

```
confirm.head()
```

	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/
0	NaN	Afghanistan	33.93911	67.709953	0	0	0	0	0	0	...	207310	207399	2
1	NaN	Albania	41.15330	20.168300	0	0	0	0	0	0	...	333749	333749	3
2	NaN	Algeria	28.03390	1.659600	0	0	0	0	0	0	...	271194	271198	2
3	NaN	Andorra	42.50630	1.521800	0	0	0	0	0	0	...	47686	47686	
4	NaN	Angola	-11.20270	17.873900	0	0	0	0	0	0	...	104973	104973	1

5 rows × 1081 columns

```
death = death.melt(id_vars=['Province/State','Country/Region','Lat','Long'],var_name='date',value_name='num_deaths')
confirm = confirm.melt(id_vars=['Province/State','Country/Region','Lat','Long'],var_name='date',value_name='num_cases')
```

```
death.head()
```

	Province/State	Country/Region	Lat	Long	date	num_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

```
confirm.merge(death,on=['Province/State','Country/Region','Lat','Long','date'])[['Country/Region','date','num_cases','num_deaths']]
```

	Country/Region	date	num_cases	num_deaths
1	Albania	1/22/20	0	0
2	Algeria	1/22/20	0	0
4	Angola	1/22/20	0	0

311253 rows × 4 columns

Pivot Table

The pivot table takes simple column-wise data as input, and groups the entries into a two-dimensional table that provides a multidimensional summarization of the data.