

Module 2: Pandas DataFrames – Basics

- 1. Creating DataFrames
- 2. DataFrame Attributes
- 3. Descriptive Methods
- 4. Selecting Rows/Columns
- 5. Modifying DataFrames
- 6. Transposing, Sorting, Resetting Index

Dataframe is like Table

1) Creating DataFrames:

- In Pandas, a DataFrame is like a table
- It contains rows, columns, labels, and supports a variety of data types.
- You can create a DataFrame from:
 - Dictionary
 - List of Series
 - List of Lists / Tuples
 - Another DataFrame
 - Reading from files (CSV, Excel, JSON etc.)

*In real time
we'll get CSV or
Excel, JSON
files*

a) From Dictionary:

all column values

No Row Labels

```

import pandas as pd
mydata = {
    'Name': ['Srinivas', 'Vas', 'Hello'],
    'Age': [25, 30, 35],
    'Course': ['Machine Learning', 'Java Fullstack', 'Deep Learning']
}

df = pd.DataFrame(data=mydata)
df

```

For testing or lab purpose we use dictionary DS

	Name	Age	Course
0	Srinivas	25	Machine Learning
1	Vas	30	Java Fullstack
2	Hello	35	Deep Learning

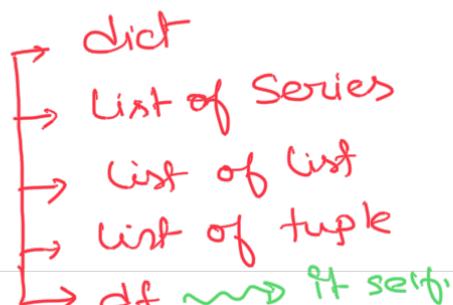
Next steps: [Generate code with df](#) [New interactive sheet](#)

▼ b) From List of Series:

- So in the below example if you see we don't have any column name or column labels. So index will be used in DataFrame.

0 1

- In DataFrame constructor we can pass below Data types:-



```

import pandas as pd
mydata = [
    pd.Series(['Srinivas', 20, 'Machine Learning']),
    pd.Series(['Vas', 25, 'Java Fullstack']),
    pd.Series(['Hello', 30, 'Deep Learning']),
]

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

print("-"*25)

df.columns = ['Name', 'Age', 'Course'] # Set column names
print(df)

```

No row label

No col label

How to add colm these label to Dataframe

Not col label

should be present for each colm values other wish it gives error

Error: ValueError

Expected axis has 3 elemnt new values have 2 elemnt

	0	1	2
0	Srinivas	20	Machine Learning
1	Vas	25	Java Fullstack
2	Hello	30	Deep Learning

	Name	Age	Course
0	Srinivas	20	Machine Learning
1	Vas	25	Java Fullstack
2	Hello	30	Deep Learning

✓ c) From List of Lists:

```
import pandas as pd
```

```
mydata = [  
    ['Srinivas', 20, 'Machine Learning'],  
    ['Vas', 25, 'Java Fullstack'],  
    ['Hello', 30, 'Deep Learning']  
]
```

```
df = pd.DataFrame(data=mydata, columns=['Name', 'Age', 'Course'])
```

```
df
```

	Name	Age	Course
0	Srinivas	20	Machine Learning
1	Vas	25	Java Fullstack
2	Hello	30	Deep Learning

Next steps:

[Generate code with df](#)[New interactive sheet](#)

Normal way

each list is single row

other way to create DataFrame with colm colbe using list.

Define the colm Label

✓ d) from another DataFrame:

```
import pandas as pd

mydata = {
    'Name': ['Srinivas', 'Vas', 'Hello'],
    'Age': [25, 30, 35],
    'Course': ['ML', 'DevOps', 'Java']
}
```

```
df1 = pd.DataFrame(mydata)
```

```
print(df1)
print("-"*25)
```

```
df2 = pd.DataFrame(data=df1)
```

```
print(df2)
print("-"*25)
```

```
df3 = df1.copy()
```

```
print(df3)
```

	Name	Age	Course
0	Srinivas	25	ML
1	Vas	30	DevOps
2	Hello	35	Java

	Name	Age	Course
0	Srinivas	25	ML
1	Vas	30	DevOps
2	Hello	35	Java

	Name	Age	Course
0	Srinivas	25	ML
1	Vas	30	DevOps
2	Hello	35	Java

} create 1 DataFrame
From other DataFrame

shallow Copy :- Both DataFrame is having same object.

/ pd.DataFrame (df) → using constructor [shallow copy]

→ using Copy method [deep copy]

? } constructor (shallow copy)

? } copy method deep copy

- Key points:

- pd.DataFrame(df1) → Creates a Shallow Copy
- df1.copy() → Creates a Deep Copy

- If you change a value in df1,

- df2 may reflect it (since it's shallow).
- df3 will not be affected.

How to get Current path:

✓ e) Reading data from files:

```
df1 = pd.read_csv('mystudents.csv')
```

```
df1
```

Read Data from external file.

	studentId	studentName	city	course	fee	
0	1	Amit Sharma	Delhi	Computer Science	55000	
1	2	Priya Mehta	Mumbai	Information Technology	60000	
2	3	Rahul Verma	Bengaluru	Electronics	52000	
3	4	Sneha Iyer	Chennai	Mechanical Engineering	58000	
4	5	Arjun Singh	Kolkata	Civil Engineering	57000	
5	6	Neha Gupta	Pune	Data Science	65000	

Next steps:

[Generate code with df1](#)

[New interactive sheet](#)

```
df2 = pd.read_excel("mystudents.xlsx")
```

```
df2
```

	studentId	studentName	city	course	fee	
0	1	Amit Sharma	Delhi	Computer Science	55000	
1	2	Priya Mehta	Mumbai	Information Technology	60000	
2	3	Rahul Verma	Bengaluru	Electronics	52000	
3	4	Sneha Iyer	Chennai	Mechanical Engineering	58000	
4	5	Arjun Singh	Kolkata	Civil Engineering	57000	
5	6	Neha Gupta	Pune	Data Science	65000	

Next steps:

[Generate code with df2](#)

[New interactive sheet](#)

```
df3 = pd.read_json("mystudents.json")
```

```
df3
```

	studentId	studentName	city	course	fee	
0	1	Amit Sharma	Delhi	Computer Science	55000.0	
1	2	Priya Mehta	Mumbai	Information Technology	60000.0	
2	3	Rahul Verma	Bengaluru	Electronics	52000.0	
3	4	Sneha Iyer	Chennai	Mechanical Engineering	58000.0	
4	5	Arjun Singh	Kolkata	Civil Engineering	57000.0	
5	6	Neha Gupta	Pune	Data Science	65000.0	
6	7	RAvi Gupta	Pune	None	NaN	

Next steps:

[Generate code with df3](#)

[New interactive sheet](#)

2) DataFrame Attributes

- Below are the attributes of Pandas DataFrame

- shape
- ndim
- size
- dtype
- columns
- values
- index

Important attributes

Shape :- No of rows & col in dataset.

ndim : Dimension of the dataset. (rows and coln so its 2D)

size : No of element in the df even for empty table also
 \rightarrow (rows x coln)

dtype : - Check below

cols : It returns the col names of the

Index : Return the colm index in the form of RangeIndex(start=0, Stop=4, Step=1)

```

import pandas as pd
import numpy as np

df1 = pd.DataFrame()
print(df1)

print("-"*25)

data = {
    'Name': ['Srinivas', 'Vas', 'Hello', "Hai"],
    'Age': [25, 30, 35, None],
    'Course': ['ML', 'DevOps', 'Java', np.nan]
}

df2 = pd.DataFrame(data)
print(df2)

```

Empty DataFrame

Columns: []

Index: []

	Name	Age	Course
0	Srinivas	25.0	ML
1	Vas	30.0	DevOps
2	Hello	35.0	Java
3	Hai	NaN	NaN

✓ a) shape

- Number of rows and columns
- Returns a tuple (rows, columns)

```

print(df1.shape)
print("-"*25)
print(df2.shape)

```

(0, 0)

(4, 3)

Dataframe → (row, coln)

✓ b) ndim

- Number of dimensions
- 1 for Series, 2 for DataFrame

```
print(df1.ndim) → Dataframe is always Table (row, col)
print("-" * 25)
print(df2.ndim)
```

↳ ndim = 2

2

2

✓ c) size

- Total number of elements (rows × columns)

```
print(df1.size) → total No of element.
print("-" * 25)
print(df2.size)

0 → As its empty.
-----
```

12

✓ d) dtypes

- Returns Data types of each column

```
print(df1.dtypes)
print("-" * 25)
print(df2.dtypes)
```

Series([], dtype: object)

Name object
Age float64
Course object
dtype: object

each coln data Type

it can be change.

✓ e) columns

- List of column names (Index object)

```
print(df1.columns)
print("-"*25)
print(df2.columns)
```

	0	1	2
0	Name	age	Course
1			
2			
3			

RangeIndex(start=0, stop=0, step=1)

Index(['Name', 'Age', 'Course'], dtype='object')

So output
is also
Index

As overall its coln index Label.

0 index = Name

1 → Age

2 → course

dtype is object

✓ f) values

- All values in the DataFrame as a 2D NumPy array

```
print(df1.values)
print("-"*25)
print(df2.values)
```

[]

[['Srinivas' 25.0 'ML']
 ['Vas' 30.0 'DevOps']
 ['Hello' 35.0 'Java']
 ['Hai' nan nan]]

}

Only the value → list of list

✓ g) index

- Row index range (Index object)

```
print(df1.index)
print("-"*25)
print(df2.index)
```

RangeIndex(start=0, stop=0, step=1)

RangeIndex(start=0, stop=4, step=1)

→ index → 0, 1, 2, 3

in RangeIndex
Form

✓ 3) Descriptive Methods in Pandas

- These are built-in methods to quickly get a summary or preview of your DataFrame
- Useful for EDA (Exploratory Data Analysis).
- Below is the List of Important methods

- head()
- tail()
- info()
- describe()
- value_counts()



Dataframe method

called Descriptive Method

```
import numpy as np
import pandas as pd

data = {
    'studentId': [101, 102, 103, 104, 105, 106, 107, 108],
    'Name': ['Srinivas', 'Vas', 'Hello', 'Srinivas', 'OK', 'Hai', 'He'],
    'Age': [25, 30, 35, None, 45, 30, 35, 28],
    'Course': ['ML', 'ML', 'ML', 'Python', 'DL', 'ML', 'DL', None],
    'City': ['Bangalore', 'Chennai', 'Bangalore', 'Bangalore', None, 'Hyderabad'],
    'Fee': [20000, 25000, 15000, 18000, 22000, 21000, 17000, np.nan]
}

df = pd.DataFrame(data)

df
```

	studentId	Name	Age	Course	City	Fee	grid icon
0	101	Srinivas	25.0	ML	Bangalore	20000.0	info icon
1	102	Vas	30.0	ML	Chennai	25000.0	edit icon
2	103	Hello	35.0	ML	Bangalore	15000.0	
3	104	Srinivas	NaN	Python	Bangalore	18000.0	
4	105	OK	45.0	DL	None	22000.0	
5	106	Hai	30.0	ML	Hyderabad	21000.0	
6	107	Hello	35.0	DL	NaN	17000.0	
7	108	Vas	28.0	None	Chennai	NaN	

Next steps: [Generate code with df](#) [New interactive sheet](#)

✓ a) head()

- Returns the first n rows of the DataFrame.
- Default: n = 5.

```
print(df.head())      # First 5 rows
```

```
print("-"*25)
```

```
print(df.head(3))      # First 3 rows
```

	studentId	Name	Age	Course	City	Fee
0	101	Srinivas	25.0	ML	Bangalore	20000.0
1	102	Vas	30.0	ML	Chennai	25000.0
2	103	Hello	35.0	ML	Bangalore	15000.0
3	104	Srinivas	NaN	Python	Bangalore	18000.0
4	105	OK	45.0	DL	None	22000.0

	studentId	Name	Age	Course	City	Fee
0	101	Srinivas	25.0	ML	Bangalore	20000.0
1	102	Vas	30.0	ML	Chennai	25000.0
2	103	Hello	35.0	ML	Bangalore	15000.0

▼ b) tail()

- Returns the last n rows of the DataFrame.
- Default: n = 5.

```
print(df.tail())      # Last 5 rows
```

```
print("-"*25)
```

```
print(df.tail(2))      # Last 2 rows
```

	studentId	Name	Age	Course	City	Fee
3	104	Srinivas	NaN	Python	Bangalore	18000.0
4	105	OK	45.0	DL	None	22000.0
5	106	Hai	30.0	ML	Hyderabad	21000.0
6	107	Hello	35.0	DL	NaN	17000.0
7	108	Vas	28.0	None	Chennai	NaN

	studentId	Name	Age	Course	City	Fee
6	107	Hello	35.0	DL	NaN	17000.0
7	108	Vas	28.0	None	Chennai	NaN

✓ c) info()

- Gives a summary of:
 - Number of rows and columns
 - Column names
 - Non-null counts → count of non-null count
 - Data types
 - Memory usage

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 8 entries, 0 to 7
Data columns (total 6 columns):
 #   Column      Non-Null Count  Dtype  
--- 
 0   studentId   8 non-null      int64  
 1   Name         8 non-null      object  
 2   Age          7 non-null      float64 
 3   Course       7 non-null      object  
 4   City          6 non-null      object  
 5   Fee           7 non-null      float64 
dtypes: float64(2), int64(1), object(3)
memory usage: 516.0+ bytes
```

6 columns there

→ each coln data type

✓ d) describe() → only displays the numeric coln. By Default

- Generates descriptive statistics for numeric columns (by default).
- These Statistics Includes:
 - count
 - mean
 - std (standard deviation)
 - min
 - 25%
 - 50% (median)
 - 75%
 - max.

2 types of coln
 * Numerical Coln (int float)
 * Categorical Coln (object)
 → String
 → date
 → boolean

→ By default all
the numeric column

```
print(df.describe()) # Numeric columns only
```

```
print("-"*25)
```

```
print(df.describe(include='all')) # All columns including object type
```

	studentId	Age	Fee	→ float64	→ float64	→ float64	type
count	8.00000	7.000000	7.000000				
mean	104.50000	32.571429	19714.285714				
std	2.44949	6.553807	3352.326839				
min	101.00000	25.000000	15000.000000				
25%	102.75000	29.000000	17500.000000				
50%	104.50000	30.000000	20000.000000				
75%	106.25000	35.000000	21500.000000				
max	108.00000	45.000000	25000.000000				
<hr/>							
count	8.00000	8	7.000000	7	6	7.000000	
unique	NaN	5	NaN	3	3	NaN	
top	NaN	Srinivas	NaN	ML	Bangalore	NaN	
freq	NaN	2	NaN	4	3	NaN	
mean	104.50000	NaN	32.571429	NaN	NaN	19714.285714	
std	2.44949	NaN	6.553807	NaN	NaN	3352.326839	
min	101.00000	NaN	25.000000	NaN	NaN	15000.000000	
25%	102.75000	NaN	29.000000	NaN	NaN	17500.000000	
50%	104.50000	NaN	30.000000	NaN	NaN	20000.000000	
75%	106.25000	NaN	35.000000	NaN	NaN	21500.000000	
max	108.00000	NaN	45.000000	NaN	NaN	25000.000000	

✓ e) value_counts()

- Returns frequency count for unique values in a Series (column).

→ No NaN i.e. → int64

↑
As Fee is
having NaN
its float64

None: No value

NaN: Not a number

```

print(df['Age'].value_counts())
print("-"*25)

print(df['City'].value_counts())
print("-"*25)

print(df['Course'].value_counts())
print("-"*25)

```

Age

30.0	2
35.0	2
25.0	1
45.0	1
28.0	1

Name: count, dtype: int64

City

Bangalore	3
Chennai	2
Hyderabad	1

Name: count, dtype: int64

Course

ML	4
DL	2
Python	1

Name: count, dtype: int64

if you want to
operate on a specific
column df['Age']

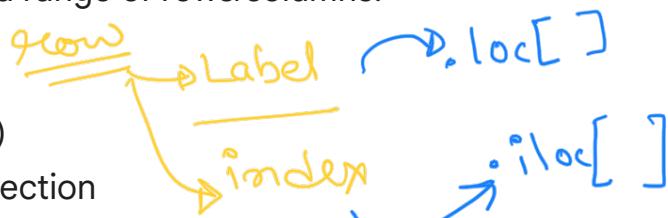
4) Selecting Rows and Columns (Indexing & Slicing)

- In Pandas
 - indexing means accessing specific rows/columns,
 - slicing means selecting a range of rows/columns.

- We can select data using:

- [] (quick column access)
- .loc[] → label-based selection
- .iloc[] → position-based selection

index



```

import pandas as pd

data = {
    'studentId': [101, 102, 103, 104, 105, 106, 107, 108],
    'Name': ['Srinivas', 'Vas', 'Hello', 'Srinivas', 'OK', 'Hai', 'He'],
    'Age': [25, 30, 35, 40, 45, 30, 35, 28],
    'Course': ['ML', 'ML', 'ML', 'Python', 'DL', 'ML', 'DL', 'ML'],
    'City': ['Bangalore', 'Chennai', 'Bangalore', 'Bangalore', 'Delhi'],
    'Fee': [20000, 25000, 15000, 18000, 22000, 21000, 17000, 24000]
}

df = pd.DataFrame(data)

df

```

	studentId	Name	Age	Course	City	Fee	
0	101	Srinivas	25	ML	Bangalore	20000	
1	102	Vas	30	ML	Chennai	25000	
2	103	Hello	35	ML	Bangalore	15000	
3	104	Srinivas	40	Python	Bangalore	18000	
4	105	OK	45	DL	Delhi	22000	
5	106	Hai	30	ML	Hyderabad	21000	
6	107	Hello	35	DL	Pune	17000	
7	108	Vas	28	ML	Chennai	24000	

Next steps: [Generate code with df](#) [New interactive sheet](#)

✓ a) Selecting Columns

```

# Single column

print(df['Name'])          # As a Series
print("-"*25)

print(df[['City']])        # As a DataFrame
print("-"*25)

# Multiple columns

```

Why Dataframe as
we are passing
df[**[city]**]

```
print(df[['Name', 'City']])
print("-"*25)

print(df[['Name', 'City', 'Course']])
```

0 Srinivas
1 Vas
2 Hello
3 Srinivas
4 OK
5 Hai
6 Hello
7 Vas
Name: Name, dtype: object

0 Bangalore
1 Chennai
2 Bangalore
3 Bangalore
4 Delhi
5 Hyderabad
6 Pune
7 Chennai

	Name	City
0	Srinivas	Bangalore
1	Vas	Chennai
2	Hello	Bangalore
3	Srinivas	Bangalore
4	OK	Delhi
5	Hai	Hyderabad
6	Hello	Pune
7	Vas	Chennai

	Name	City	Course
0	Srinivas	Bangalore	ML
1	Vas	Chennai	ML
2	Hello	Bangalore	ML
3	Srinivas	Bangalore	Python
4	OK	Delhi	DL
5	Hai	Hyderabad	ML
6	Hello	Pune	DL
7	Vas	Chennai	ML

▼ b) Selecting Rows by Index Position – .iloc[]

```
# Single row
```

```
print(df.iloc[1])      # 2nd row  
print("-"*25)
```

```
# Multiple rows
```

```
print(df.iloc[[1, 3, 6]])  # rows 1, 3, 6  
print("-"*25)
```

```
# Row slicing
```

```
print(df.iloc[2:7])    # rows from index 2 to 6
```

```
studentId      102  
Name          Vas  
Age           30  
Course        ML  
City          Chennai  
Fee          25000  
Name: 1, dtype: object
```

```
studentId      102      Name   Age  Course      City     Fee  
1            102      Vas    30    ML  Chennai  25000  
3            104  Srinivas  40  Python  Bangalore  18000  
6            107      Hello  35     DL    Pune    17000
```

```
studentId      103      Name   Age  Course      City     Fee  
2            103      Hello  35     ML  Bangalore  15000  
3            104  Srinivas  40  Python  Bangalore  18000  
4            105       OK   45     DL    Delhi    22000  
5            106       Hai  30     ML  Hyderabad  21000  
6            107      Hello  35     DL    Pune    17000
```

Series

Dataframe

[1, 2]

✓ c) Selecting Rows by Label – .loc[]

```
# Single row
print(df.loc[2])      # row with label 2
print("-"*25)

# Multiple rows

print(df.loc[[2, 5, 7]])  # rows with labels 2, 5, 7
print("-"*25)

# Row slicing

print(df.loc[2:7])      # rows from index 2 to 7
```

studentId	103
Name	Hello
Age	35
Course	ML
City	Bangalore
Fee	15000
Name:	2, dtype: object

	studentId	Name	Age	Course	City	Fee
2	103	Hello	35	ML	Bangalore	15000
5	106	Hai	30	ML	Hyderabad	21000
7	108	Vas	28	ML	Chennai	24000

	studentId	Name	Age	Course	City	Fee
2	103	Hello	35	ML	Bangalore	15000
3	104	Srinivas	40	Python	Bangalore	18000
4	105	OK	45	DL	Delhi	22000
5	106	Hai	30	ML	Hyderabad	21000
6	107	Hello	35	DL	Pune	17000
7	108	Vas	28	ML	Chennai	24000

✓ d) Selecting Rows & Columns Together

- # Selects rows from 2 to 6 and cols from 0 to 2

```
df.iloc[2:7, 0:3]
```

Row

what all col you want (col index)

	studentId	Name	Age	
2	103	Hello	35	
3	104	Srinivas	40	
4	105	OK	45	
5	106	Hai	30	
6	107	Hello	35	

Selects rows from 2 to 6 and 2 cols – Name and Course

```
df.loc[2:7, ['Name', 'Course']]
```

	Name	Course	
2	Hello	ML	
3	Srinivas	Python	
4	OK	DL	
5	Hai	ML	
6	Hello	DL	
7	Vas	ML	

*If you know Col Names
or Col Labels are present*

✓ e) Slicing Columns

```
# first two columns  
print(df.iloc[:, 1:4])  
print("-"*25)
```

coln slicing

```
# columns from Name to Course  
print(df.loc[:, 'Name':'Course'])
```

	Name	Age	Course
0	Srinivas	25	ML
1	Vas	30	ML
2	Hello	35	ML
3	Srinivas	40	Python
4	OK	45	DL
5	Hai	30	ML
6	Hello	35	DL
7	Vas	28	ML

	Name	Age	Course
0	Srinivas	25	ML
1	Vas	30	ML
2	Hello	35	ML
3	Srinivas	40	Python
4	OK	45	DL
5	Hai	30	ML
6	Hello	35	DL
7	Vas	28	ML

✓ f) Negative Indexing with iloc

only row
(-1) means last coln

```
print(df.iloc[-1])          # last row
print("-"*25)

print(df.iloc[-3:])         # last 3 rows
print("-"*25)

print(df.iloc[:, -1])       # last column → For all row
print(df.iloc[-3:, -3:])    # last 3 rows and last 3 cols
```

```
studentId      108
Name           Vas
Age            28
Course          ML
City          Chennai
Fee           24000
Name: 7, dtype: object
```

```
-----
```

	studentId	Name	Age	Course	City	Fee
5	106	Hai	30	ML	Hyderabad	21000
6	107	Hello	35	DL	Pune	17000
7	108	Vas	28	ML	Chennai	24000

```
-----
```

```
0    20000
1    25000
2    15000
3    18000
4    22000
5    21000
6    17000
7    24000
```

```
Name: Fee, dtype: int64
      Course      City      Fee
      5    ML   Hyderabad  21000
      6    DL        Pune  17000
      7    ML   Chennai  24000
```

✓ g) Other Selections

```
# Alternative Rows and All Cols
print(df.iloc[::2])
```

```
print("-"*25)
```

```
# Alternative Rows and Alternative Cols
```

```
print(df.iloc[::2, ::2])
```

Alternate row

start, stop, step

row	All col
1	
3	
5	
7	
9	

	studentId	Name	Age	Course	City	Fee
0	101	Srinivas	25	ML	Bangalore	20000
2	103	Hello	35	ML	Bangalore	15000
4	105	OK	45	DL	Delhi	22000
6	107	Hello	35	DL	Pune	17000

	studentId	Age	City
0	101	25	Bangalore
2	103	35	Bangalore
4	105	45	Delhi
6	107	35	Pune

```
# df[0] - Error
# df["Age"]
df.columns=[0,1,2,3,4,5]
# df["Age"]
print(df)
df[0]
```

As column labeling already done
so coln index available

if you want to select it
and want indexes.

	0	1	2	3	4	5
0	101	Srinivas	25	ML	Bangalore	20000
1	102	Vas	30	ML	Chennai	25000
2	103	Hello	35	ML	Bangalore	15000
3	104	Srinivas	40	Python	Bangalore	18000
4	105	OK	45	DL	Delhi	22000
5	106	Hai	30	ML	Hyderabad	21000
6	107	Hello	35	DL	Pune	17000
7	108	Vas	28	ML	Chennai	24000

0
0 101
1 102
2 103
3 104
4 105
5 106
6 107
7 108

dtype: int64

5) Modifying DataFrames

- We can change, add, rename, or delete rows/columns in a DataFrame.

```
import pandas as pd

data = {
    'Name': ['Srinivas', 'Vas', 'Hello', "Sri", "Hai"],
    'Age': [25, 30, 35, 40, 45],
    'Course': ['ML', 'DevOps', 'Java', "DL", "ML"]
}

df = pd.DataFrame(data)
print(df)
```

	Name	Age	Course
0	Srinivas	25	ML
1	Vas	30	DevOps
2	Hello	35	Java
3	Sri	40	DL
4	Hai	45	ML

✓ a) Renaming Columns and Index

If you want to change the coln name single or multiple
coln name

```
# Rename a column
df.rename(columns={'Course': 'MyCourse'}, inplace=True)
print(df)
print("-"*25)

# Rename multiple columns
df.rename(columns={'Name': 'FullName', 'Age': 'Years'}, inplace=True)
print(df)
print("-"*25)

# Rename index
df.rename(index={0: 'Row1', 1: 'Row2'}, inplace=True)
print(df)
print("-"*25)
```

```
Name    Age  MyCourse
0  Srinivas   25      ML
1        Vas   30  DevOps
2     Hello   35      Java
3       Sri   40      DL
4       Hai   45      ML
```

```
FullName  Years  MyCourse
0  Srinivas   25      ML
1        Vas   30  DevOps
2     Hello   35      Java
3       Sri   40      DL
4       Hai   45      ML
```

```
FullName  Years  MyCourse
Row1  Srinivas   25      ML
Row2        Vas   30  DevOps
2     Hello   35      Java
3       Sri   40      DL
4       Hai   45      ML
```

▼ b) Adding a New Column

```
df['Marks'] = [85, 90, 88, 60, 72]
print(df)
print("-"*25)

df['Passed'] = df['Marks'] >= 85
print(df)
```

I do checking like this
it will add the colm Passed
with value checked
recensivly:

	FullName	Years	MyCourse	Marks
Row1	Srinivas	25	ML	85
Row2	Vas	30	DevOps	90
2	Hello	35	Java	88
3	Sri	40	DL	60
4	Hai	45	ML	72

85>=85	✓
90>=85	✓
88>=85	✓
60>=85	✗
72>=85	✗

	FullName	Years	MyCourse	Marks	Passed
Row1	Srinivas	25	ML	85	True
Row2	Vas	30	DevOps	90	True
2	Hello	35	Java	88	True
3	Sri	40	DL	60	False
4	Hai	45	ML	72	False

c) Modifying Existing Column Values

```
df['Years'] = df['Years'] + 1
print(df)
print("-"*25)

df['MyCourse'] = df['MyCourse'].str.upper()
print(df)
```

you can modify the
existing values.

All

	FullName	Years	MyCourse	Marks	Passed
Row1	Srinivas	26	ML	85	True
Row2	Vas	31	DevOps	90	True
2	Hello	36	Java	88	True
3	Sri	41	DL	60	False
4	Hai	46	ML	72	False

	FullName	Years	MyCourse	Marks	Passed
Row1	Srinivas	26	ML	85	True
Row2	Vas	31	DEVOPS	90	True
2	Hello	36	JAVA	88	True
3	Sri	41	DL	60	False
4	Hai	46	ML	72	False

✓ d) Adding a New Row

```
# Using loc
df.loc[5] = ['Kiran', 40, 'PYTHON', 95, True]
print(df)
```

So you can add new coln record in existing DF

	FullName	Years	MyCourse	Marks	Passed
Row1	Srinivas	26	ML	85	True
Row2	Vas	31	DEVOPS	90	True
2	Hello	36	JAVA	88	True
3	Sri	41	DL	60	False
4	Hai	46	ML	72	False
5	Kiran	40	PYTHON	95	True

✓ e) Updating Specific Values

```
df.at[2, 'Marks'] = 100      # by label
print(df)
print("-"*25)
```

```
df.iat[3, 1] = 50           # by position
print(df)
```

df.at[row, coln]
 ↓
 index Based Label/ Index based

	FullName	Years	MyCourse	Marks	Passed
Row1	Srinivas	26	ML	85	True
Row2	Vas	31	DEVOPS	90	True
2	Hello	36	JAVA	100	True
3	Sri	41	DL	60	False
4	Hai	46	ML	72	False
5	Kiran	40	PYTHON	95	True

	FullName	Years	MyCourse	Marks	Passed
Row1	Srinivas	26	ML	85	True
Row2	Vas	31	DEVOPS	90	True
2	Hello	36	JAVA	100	True
3	Sri	50	DL	60	False
4	Hai	46	ML	72	False
5	Kiran	40	PYTHON	95	True

df.iat[row, coln]
 ↑
 index index

✓ f) Deleting Columns

df.drop(columns=['Passed'], inplace=True)

```
print(df)
print("-"*25)

df.drop(columns=['Passed'], inplace=True)
print(df)
```

	FullName	Years	MyCourse	Marks	Passed
Row1	Srinivas	26	ML	85	True
Row2	Vas	31	DEVOPS	90	True
2	Hello	36	JAVA	100	True
3	Sri	50	DL	60	False
4	Hai	46	ML	72	False
5	Kiran	40	PYTHON	95	True

	FullName	Years	MyCourse	Marks
Row1	Srinivas	26	ML	85
Row2	Vas	31	DEVOPS	90
2	Hello	36	JAVA	100
3	Sri	50	DL	60
4	Hai	46	ML	72
5	Kiran	40	PYTHON	95

✓ g) Deleting Rows → `df.drop(index=[2,3], inplace=True)`

```
df.drop(index=[2, 3], inplace=True)
print(df)
print("-"*25)
```

2nd and 3rd row will
be dropped

	FullName	Years	MyCourse	Marks
Row1	Srinivas	26	ML	85
Row2	Vas	31	DEVOPS	90
4	Hai	46	ML	72
5	Kiran	40	PYTHON	95

```
df.drop(index=["Row1", "Row2", 4], inplace=True)
```

inplace = true

- means update happen in the same DataFrame

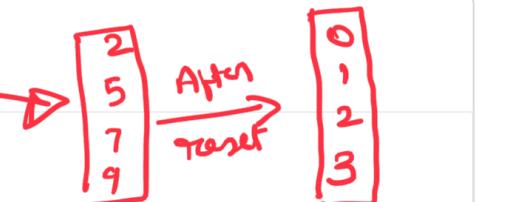
	FullName	Years	MyCourse	Marks
5	Kiran	40	PYTHON	95

*- if inplace = false
↳ New DF will be created with updated info*

```
df.reset_index(drop=True, inplace=True)
```

*After reset
row value
it been reset*

	FullName	Years	MyCourse	Marks
0	Kiran	40	PYTHON	95



6) Transposing, Sorting, Resetting Index

- These are common operations to rearrange DataFrames for better analysis.

```
import pandas as pd

data = {
    'Name': ['Srinivas', 'Vas', 'Hello', 'Hai', 'Sri'],
    'Age': [40, 25, 30, 45, 35],
    'Course': ['ML', 'DevOps', 'Java', 'Python', 'DL']
}

df = pd.DataFrame(data)
print(df)
```

	Name	Age	Course
0	Srinivas	40	ML
1	Vas	25	DevOps
2	Hello	30	Java
3	Hai	45	Python
4	Sri	35	DL

✓ a) Transposing a DataFrame

- Swaps rows ↔ columns.

```
print(df)
print("-"*25)

# Transpose
mydf = df.T → By default inplace=False

print(mydf)
print("-"*25)

print(df)
print("-"*25)
```

	Name	Age	Course
0	Srinivas	40	ML
1	Vas	25	DevOps
2	Hello	30	Java
3	Hai	45	Python
4	Sri	35	DL

	0	1	2	3	4
Name	Srinivas	Vas	Hello	Hai	Sri
Age	40	25	30	45	35
Course	ML	DevOps	Java	Python	DL

	Name	Age	Course
0	Srinivas	40	ML
1	Vas	25	DevOps
2	Hello	30	Java
3	Hai	45	Python
4	Sri	35	DL

✓ b) Sorting

df.sort_values(by='Age')

- Sort by ASC order or DESC order

```
# Sort by a column
print(df)
print("-"*25)

print(df.sort_values(by='Age'))
print("-"*25)

print(df.sort_values(by='Age', ascending=False))
print("-"*25)
```

```
Name    Age   Course
0 Srinivas  40      ML
1 Vas        25  DevOps
2 Hello      30      Java
3 Hai        45  Python
4 Sri        35      DL
```

```
Name    Age   Course
1 Vas        25  DevOps
2 Hello      30      Java
4 Sri        35      DL
0 Srinivas  40      ML
3 Hai        45  Python
```

```
Name    Age   Course
3 Hai        45  Python
0 Srinivas  40      ML
4 Sri        35      DL
2 Hello      30      Java
1 Vas        25  DevOps
```

```
# Sort by multiple columns
```

```
# Sort Course by ASC and Age by ASC
```

```
print(df.sort_values(by=['Course', 'Age']))
```

```
# print(df.sort_values(by=['Course', 'Age']), ascending=[True, True]))
```

```
print("-"*25)
```

```
# Sort Course by DESC and Age by DESC
```

```
print(df.sort_values(by=['Course', 'Age'], ascending=False))
```

```
# print(df.sort_values(by=['Course', 'Age']), ascending=[False, False]))
```

```
print("-"*25)
```

```
# Sort Course by ASC and Age by DESC
```

```
df.sort_values(by=['Course', 'Age'], ascending=[True, False])
```

If both are
false then
make it
single
student

ASC
DESC

	Name	Age	Course
4	Sri	35	DL
1	Vas	25	DevOps
2	Hello	30	Java
0	Srinivas	40	ML
3	Hai	45	Python

	Name	Age	Course
3	Hai	45	Python
0	Srinivas	40	ML
2	Hello	30	Java
1	Vas	25	DevOps
4	Sri	35	DL

	Name	Age	Course	
4	Sri	35	DL	
1	Vas	25	DevOps	
2	Hello	30	Java	
0	Srinivas	40	ML	
3	Hai	45	Python	

```
# Sort by index
df = df.sort_values(by=['Course', 'Age'], ascending=False)
```

```
print(df)
print("-"*25)
```

```
print(df.sort_index())
print("-"*25)
```

```
print(df.sort_index(ascending=False))
print("-"*25)
```

```
print(df)
```

	Name	Age	Course
3	Hai	45	Python
0	Srinivas	40	ML
2	Hello	30	Java
1	Vas	25	DevOps
4	Sri	35	DL

	Name	Age	Course
0	Srinivas	40	ML
1	Vas	25	DevOps
2	Hello	30	Java
3	Hai	45	Python
4	Sri	35	DL

	Name	Age	Course
4	Sri	35	DL
3	Hai	45	Python
2	Hello	30	Java
1	Vas	25	DevOps
0	Srinivas	40	ML

	Name	Age	Course
3	Hai	45	Python
0	Srinivas	40	ML
2	Hello	30	Java
1	Vas	25	DevOps
4	Sri	35	DL

grow index wise sorting Asc order

new df created

inplace=False, so new df will be created

grow index wise sorting Desc order

c) Resetting Index

- Sometimes after filtering or dropping rows, the index is not continuous.
- We can reset it.

```
print(df)
print("-"*25)

#
df.drop(index=[1, 3], inplace=True)

# Before reset:
print(df)

# Reset index without keeping old index
df.reset_index(drop=True, inplace=True)

# After reset:
print(df)
```

	Name	Age	Course
3	Hai	45	Python
0	Srinivas	40	ML
2	Hello	30	Java
1	Vas	25	DevOps
4	Sri	35	DL

} Before Reset

	Name	Age	Course
0	Srinivas	40	ML
2	Hello	30	Java
4	Sri	35	DL

} dropped few Records

	Name	Age	Course
0	Srinivas	40	ML
1	Hello	30	Java
2	Sri	35	DL

} Then do a reset

