کارگاه یادگیری ماشین مقدماتی

کتابخانه Pandas

Dr. *Ali Valinejad*

valinejad@umz.ac.ir

pip install pandas

conda install pandas

https://pandas.pydata.org

Data Wrangling

with pandas Cheat Sheet

http://pandas.pydata.org

Pandas API Reference Pandas User Guide

Tidy Data – A foundation for wrangling in pandas

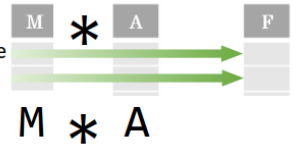
In a tidy data set:

&

Each **variable** is saved in its own **column**

Each **observation** is saved in its own **row**

Tidy data complements pandas's vectorized operations. pandas will automatically preserve observations as you manipulate variables. No other format works as intuitively with pandas.



Creating DataFrames

	a	b	c
1	4	7	10
2	5	8	11
3	6	9	12

```
df = pd.DataFrame(
        {"a" : [4, 5, 6],
         "b" : [7, 8, 9],
         "c" : [10, 11, 12]},
       index = [1, 2, 3])
```

Specify values for each column.

```
df = pd.DataFrame(
    [[4, 7, 10],
     [5, 8, 11],
     [6, 9, 12]],
    index=[1, 2, 3],
    columns=['a', 'b', 'c'])
```

Specify values for each row.

		a	b	c
N	v			
D	1	4	7	10
	2	5	8	11
e	2	6	9	12

```
df = pd.DataFrame(
        {"a" : [4 ,5, 6],
         "b" : [7, 8, 9],
         "c" : [10, 11, 12]},
index = pd.MultiIndex.from_tuples(
        [('d', 1), ('d', 2),
         ('e', 2)], names=['n', 'v']))
```

Create DataFrame with a MultiIndex

Method Chaining

Most pandas methods return a DataFrame so that another pandas method can be applied to the result. This improves readability of code.

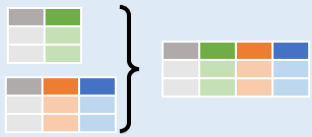
```
df = (pd.melt(df)
        .rename(columns={
                'variable':'var',
                'value':'val'})
        .query('val >= 200')
    )
```

Reshaping Data – Change layout, sorting, reindexing, renaming

pd.melt(df)
Gather columns into rows.

df.pivot(columns='var', values='val')
Spread rows into columns.

pd.concat([df1,df2])
Append rows of DataFrames



pd.concat([df1,df2], axis=1)
Append columns of DataFrames

df.sort_values('mpg')
Order rows by values of a column (low to high).

df.sort_values('mpg', ascending=False)
Order rows by values of a column (high to low).

df.rename(columns = {'y':'year'})
Rename the columns of a DataFrame

df.sort_index()
Sort the index of a DataFrame

df.reset_index()
Reset index of DataFrame to row numbers, moving index to columns.

df.drop(columns=['Length', 'Height'])
Drop columns from DataFrame

Subset Observations - rows

df[df.Length > 7]
Extract rows that meet logical criteria.

df.drop_duplicates()
Remove duplicate rows (only considers columns).

df.sample(frac=0.5)
Randomly select fraction of rows.

df.sample(n=10) Randomly select n rows.

df.nlargest(n, 'value')
Select and order top n entries.

df.nsmallest(n, 'value')
Select and order bottom n entries.

df.head(n)
Select first n rows.

df.tail(n)
Select last n rows.

Subset Variables - columns

df[['width', 'length', 'species']]
Select multiple columns with specific names.

df['width'] *or* **df.width**
Select single column with specific name.

df.filter(regex='regex')
Select columns whose name matches regular expression *regex*.

Using query

query() allows Boolean expressions for filtering rows.

```
df.query('Length > 7')
df.query('Length > 7 and Width < 8')
df.query('Name.str.startswith("abc")',
         engine="python")
```

Subsets - rows and columns

Use **df.loc[]** and **df.iloc[]** to select only rows, only columns or both.
Use **df.at[]** and **df.iat[]** to access a single value by row and column.
First index selects rows, second index columns.

df.iloc[10:20]
Select rows 10-20.

df.iloc[:, [1, 2, 5]]
Select columns in positions 1, 2 and 5 (first column is 0).

df.loc[:, 'x2':'x4']
Select all columns between x2 and x4 (inclusive).

df.loc[df['a'] > 10, ['a', 'c']]
Select rows meeting logical condition, and only the specific columns .

df.iat[1, 2] Access single value by index

df.at[4, 'A'] Access single value by label

Logic in Python (and pandas)			
<	Less than	!=	Not equal to
>	Greater than	df.column.isin(*values*)	Group membership
==	Equals	pd.isnull(*obj*)	Is NaN
<=	Less than or equals	pd.notnull(*obj*)	Is not NaN
>=	Greater than or equals	&,|,~,^,df.any(),df.all()	Logical and, or, not, xor, any, all

regex (Regular Expressions) Examples	
'\.'	Matches strings containing a period '.'
'Length$'	Matches strings ending with word 'Length'
'^Sepal'	Matches strings beginning with the word 'Sepal'
'^x[1-5]$'	Matches strings beginning with 'x' and ending with 1,2,3,4,5
'^(?!Species$).*'	Matches strings except the string 'Species'

Cheatsheet for pandas (http://pandas.pydata.org/ originally written by Irv Lustig, Princeton Consultants, inspired by Rstudio Data Wrangling Cheatsheet

Summarize Data

`df['w'].value_counts()`
Count number of rows with each unique value of variable

`len(df)`
# of rows in DataFrame.

`df.shape`
Tuple of # of rows, # of columns in DataFrame.

`df['w'].nunique()`
# of distinct values in a column.

`df.describe()`
Basic descriptive and statistics for each column (or GroupBy).

pandas provides a large set of summary functions that operate on different kinds of pandas objects (DataFrame columns, Series, GroupBy, Expanding and Rolling (see below)) and produce single values for each of the groups. When applied to a DataFrame, the result is returned as a pandas Series for each column. Examples:

`sum()`
Sum values of each object.

`count()`
Count non-NA/null values of each object.

`median()`
Median value of each object.

`quantile([0.25,0.75])`
Quantiles of each object.

`apply(function)`
Apply function to each object.

`min()`
Minimum value in each object.

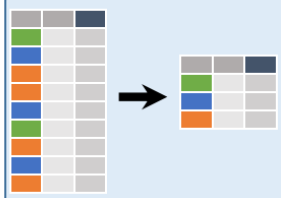
`max()`
Maximum value in each object.

`mean()`
Mean value of each object.

`var()`
Variance of each object.

`std()`
Standard deviation of each object.

Group Data



`df.groupby(by="col")`
Return a GroupBy object, grouped by values in column named "col".

`df.groupby(level="ind")`
Return a GroupBy object, grouped by values in index level named "ind".

All of the summary functions listed above can be applied to a group. Additional GroupBy functions:

`size()`
Size of each group.

`agg(function)`
Aggregate group using function.

The examples below can also be applied to groups. In this case, the function is applied on a per-group basis, and the returned vectors are of the length of the original DataFrame.

`shift(1)`
Copy with values shifted by 1.

`rank(method='dense')`
Ranks with no gaps.

`rank(method='min')`
Ranks. Ties get min rank.

`rank(pct=True)`
Ranks rescaled to interval [0, 1].

`rank(method='first')`
Ranks. Ties go to first value.

`shift(-1)`
Copy with values lagged by 1.

`cumsum()`
Cumulative sum.

`cummax()`
Cumulative max.

`cummin()`
Cumulative min.

`cumprod()`
Cumulative product.

Windows

`df.expanding()`
Return an Expanding object allowing summary functions to be applied cumulatively.

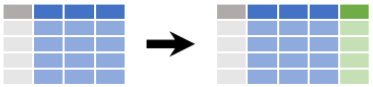
`df.rolling(n)`
Return a Rolling object allowing summary functions to be applied to windows of length n.

Handling Missing Data

`df.dropna()`
Drop rows with any column having NA/null data.

`df.fillna(value)`
Replace all NA/null data with value.

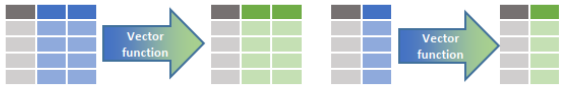
Make New Columns



`df.assign(Area=lambda df: df.Length*df.Height)`
Compute and append one or more new columns.

`df['Volume'] = df.Length*df.Height*df.Depth`
Add single column.

`pd.qcut(df.col, n, labels=False)`
Bin column into n buckets.



pandas provides a large set of **vector functions** that operate on all columns of a DataFrame or a single selected column (a pandas Series). These functions produce vectors of values for each of the columns, or a single Series for the individual Series. Examples:

`max(axis=1)`
Element-wise max.

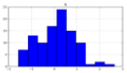
`clip(lower=-10,upper=10)`
Trim values at input thresholds

`min(axis=1)`
Element-wise min.

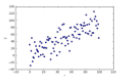
`abs()`
Absolute value.

Plotting

`df.plot.hist()`
Histogram for each column



`df.plot.scatter(x='w',y='h')`
Scatter chart using pairs of points



Combine Data Sets

adf

x1	x2
A	1
B	2
C	3

+

bdf

x1	x3
A	T
B	F
D	T

=

Standard Joins

x1	x2	x3
A	1	T
B	2	F
C	3	NaN

`pd.merge(adf, bdf, how='left', on='x1')`
Join matching rows from bdf to adf.

x1	x2	x3
A	1.0	T
B	2.0	F
D	NaN	T

`pd.merge(adf, bdf, how='right', on='x1')`
Join matching rows from adf to bdf.

x1	x2	x3
A	1	T
B	2	F

`pd.merge(adf, bdf, how='inner', on='x1')`
Join data. Retain only rows in both sets.

x1	x2	x3
A	1	T
B	2	F
C	3	NaN
D	NaN	T

`pd.merge(adf, bdf, how='outer', on='x1')`
Join data. Retain all values, all rows.

Filtering Joins

x1	x2
A	1
B	2

`adf[adf.x1.isin(bdf.x1)]`
All rows in adf that have a match in bdf.

x1	x2
C	3

`adf[~adf.x1.isin(bdf.x1)]`
All rows in adf that do not have a match in bdf.

ydf

x1	x2
A	1
B	2
C	3

+

zdf

x1	x2
B	2
C	3
D	4

=

Set-like Operations

x1	x2
B	2
C	3

`pd.merge(ydf, zdf)`
Rows that appear in both ydf and zdf (Intersection).

x1	x2
A	1
B	2
C	3
D	4

`pd.merge(ydf, zdf, how='outer')`
Rows that appear in either or both ydf and zdf (Union).

x1	x2
A	1

```
pd.merge(ydf, zdf, how='outer',
         indicator=True)
.query('_merge == "left_only"')
.drop(columns=['_merge'])
```
Rows that appear in ydf but not zdf (Setdiff).

Cheatsheet for pandas (http://pandas.pydata.org/) originally written by Irv Lustig, Princeton Consultants, inspired by Rstudio Data Wrangling Cheatsheet

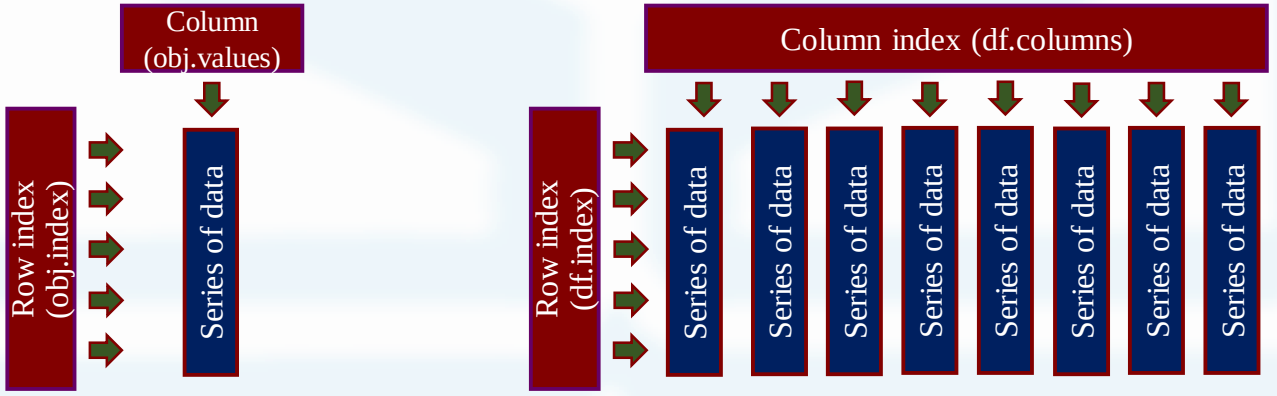
Pandas Data Structures

Series

- 1D labeled homogeneously-typed array

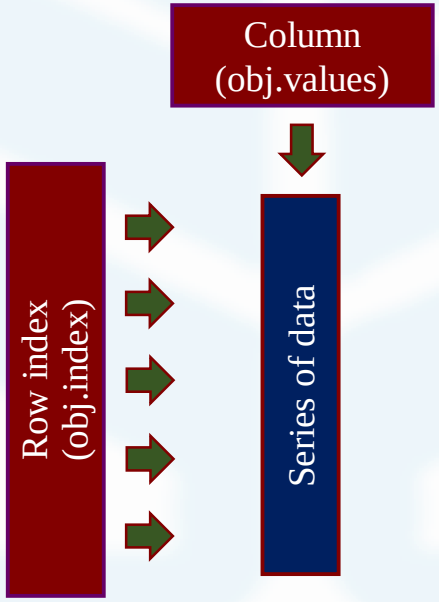
DataFrame

- General 2D labeled, size-mutable tabular structure with potentially heterogeneously-typed column



Series

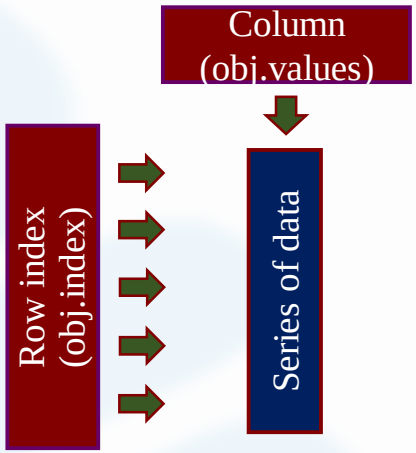
A Series is a one-dimensional array-like object containing a sequence of values (of similar types to NumPy types) and an associated array of data labels, called its index .



Pandas Data Structures : Series

❖ Attributes of Pandas Series:

- name
- index.name
- values
- index
- size
- Empty
- …



❖ Methods of Pandas Series:

- head(n)
- count()
- tail(n)
- …

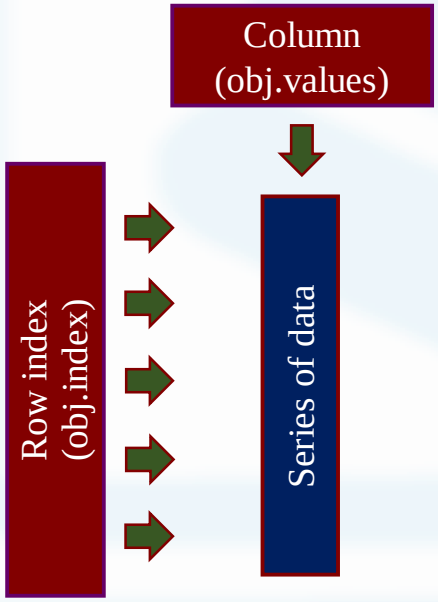
Pandas Data Structures : Series

Creation of Series

❖ Creation of Series from Scalar Values

```
import pandas as pd
s = pd.Series([30, 50, 130, 400])
```

```
from pandas import Series
s = Series([30, 50, 130, 400])
```



index	values
0	30
1	50
2	130
3	400

s[0]: 30

s[1]: 50

s[2]: 130

s[3]: 400

s.index: Index([0, 1, 2, 3], dtype='object')

s.values : [30, 50, 130, 400]

```
print( type(s) )
```

```
<class 'pandas.core.series.Series'>
```

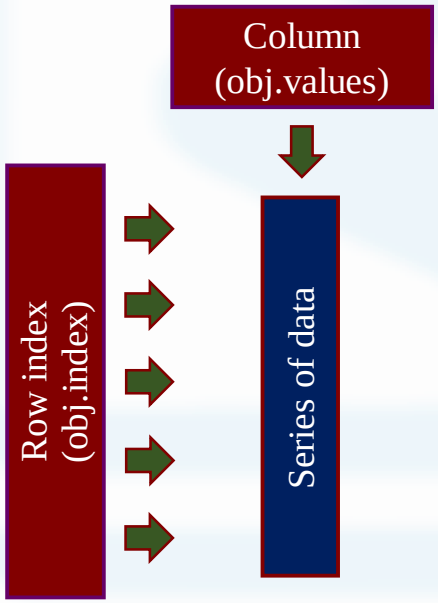
Pandas Data Structures : Series

Creation of Series

❖ Creation of Series from Scalar Values

```
import pandas as pd
s = pd.Series([30, 50, 130, 400], index=[ 'A' , 'B' , 'C' , 'D' ])
```

```
from pandas import Series
s = Series([30, 50, 130, 400], index=[ 'A' , 'B' , 'C' , 'D' ])
```



index	values
'A'	30
'B'	50
'C'	130
'D'	400

s['A']: 30

s['B']: 50

s['C']: 130

s['D']: 400

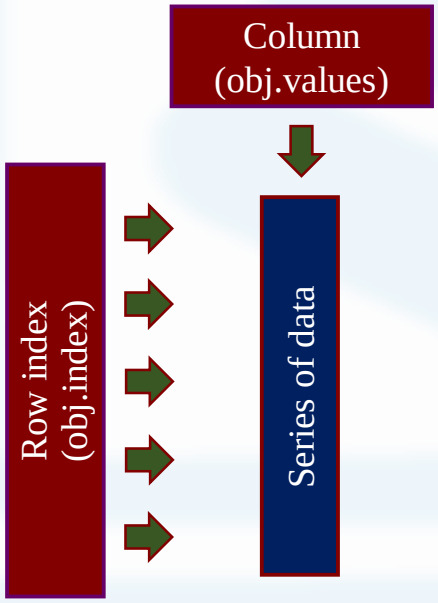
s.index: Index(['A', 'B', 'C', 'D'], dtype='object')

s.values : [30, 50, 130, 400]

Pandas Data Structures : Series

❖ Creation of Series from NumPy Arrays

```
import numpy as np
from pandas import Series
a = np.array([30, 50, 130, 400])
s = Series(a)
```



index	values
0	30
1	50
2	130
3	400

s[0]: 30

s[1]: 50

s[2]: 130

s[3]: 400

s.index: Index([0, 1, 2, 3], dtype='object')

s.values : [30, 50, 130, 400]

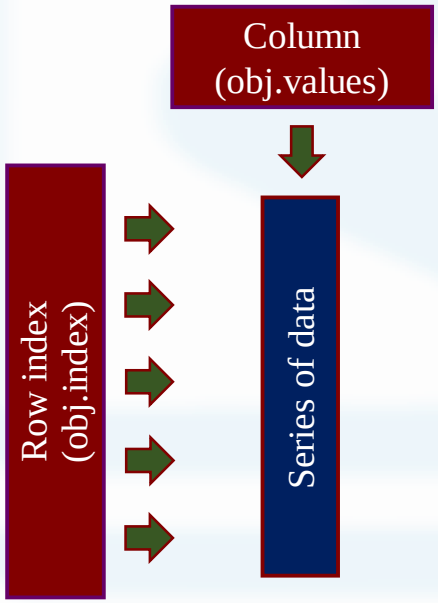
Pandas Data Structures : Series

Creation of Series

❖ Creation of Series from Dictionary

```
import pandas as pd
sdata = {'A':30, 'B':50, 'C':130, 'D':400}
s = pd.Series(sdata)
```

```
from pandas import Series
sdata = {'A':30, 'B':50, 'C':130, 'D':400}
s = Series(sdata)
```



index	values
'A'	30
'B'	50
'C'	130
'D'	400

s['A']: 30

s['B']: 50

s['C']: 130

s['D']: 400

s.index: Index(['A', 'B', 'C', 'D'], dtype='object')

s.values : [30, 50, 130, 400]

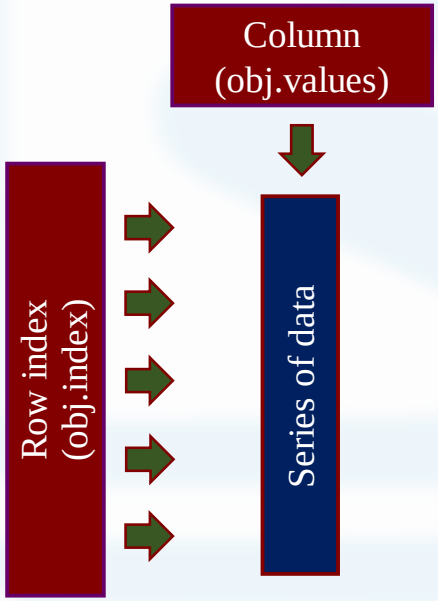
Pandas Data Structures : Series

Creation of Series

❖ Creation of Series from Dictionary

```
import pandas as pd
sdata = {‘A’:30, ‘B’:50, ‘C’:130, ‘D’:400}
cities=[‘D’, ‘B’, ‘A’]
s = pd.Series(sdata, index=cities)
```

```
from pandas import Series
sdata = {‘A’:30, ‘B’:50, ‘C’:130, ‘D’:400}
cities=[‘D’, ‘B’, ‘A’]
s = Series(sdata, index=cities)
```



index	values
‘D’	400
‘B'	50
‘A’	30

s[‘D’]: 400

s[‘B’]: 50

s[‘A’]: 30

s.index: Index([‘D', ‘B', ‘A'], dtype='object')

s.values : [400, 50, 30]

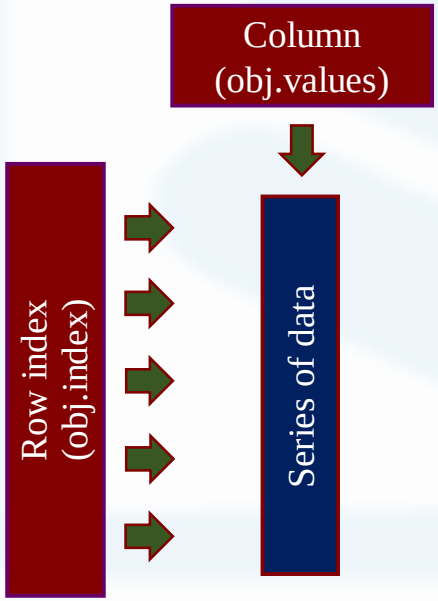
Pandas Data Structures : Series

Creation of Series

❖ Creation of Series from Dictionary

```
import pandas as pd
sdata = {'A':30, 'B':50, 'C':130, 'D':400}
cities=['D', 'B', 'A', 'F']
s = pd.Series(sdata, index=cities)
```

```
from pandas import Series
sdata = {'A':30, 'B':50, 'C':130, 'D':400}
cities=['D', 'B', 'A', 'F']
s = Series(sdata, index=cities)
```



Index	Values	
'D'	400	s['D']: 400
'B'	50	s['B']: 50
'A'	30	s['A']: 30
'F'	nan	s['F']: nan

s.index: Index(['D', 'B', 'A', 'F'], dtype='object')
s.values : [400, 50, 30, nan]

pd.isnull (series_object_name) pd.isnull(s)

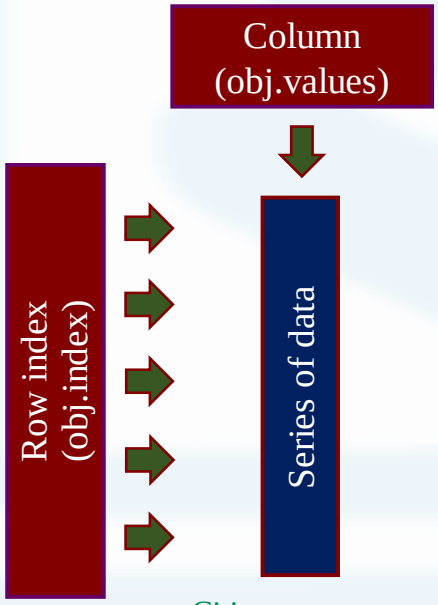
pd.notnull (series_object_name) pd.notnull(s)

Pandas Data Structures : Series

Creation of Series

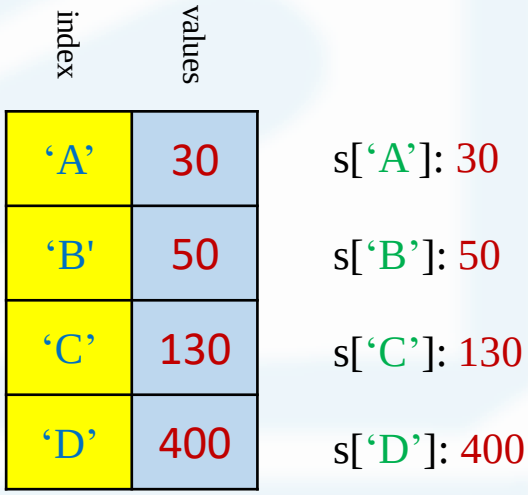
❖ obj.name, obj.index.name

```
import pandas as pd
cities = {‘A’:30, ‘B’:50, ‘C’:130, ‘D’:400}
s = pd.Series(cities)
s.name=‘Distance’
s.index.name=‘Cities’
print(s)
```




```
Cities
A      30
B      50
C     130
D     400
Name: Distance, dtype: int64
```

```
from pandas import Series
s = pd.Series([30, 50, 130, 400], index=[ ‘A' , ‘B' , ‘C' , ‘D' ])
s.name=‘Distance’
s.index.name=‘Cities’
print(s)
```



s[‘A’]: 30

s[‘B’]: 50

s[‘C’]: 130

s[‘D’]: 400

s.index: Index([‘A', ‘B', ‘C', ‘D'], dtype='object')

s.values : [30, 50, 130, 400]

Pandas Data Structures : Series

Creation of Series

❖ obj.size, obj.empty

```
import pandas as pd
cities = {'A':30, 'B':50, 'C':130, 'D':400}
s.name='Distance'
s.index.name='Cities'
s = pd.Series(cities)
```

print(s)

```
Cities
  A        30
  B        50
  C       130
  D       400
Name: Distance, dtype: int64
```

print(s.size)

```
4
```

print(s.empty)

```
False
```

Pandas Data Structures : Series

Indexing

❖ Accessing Elements of a Series

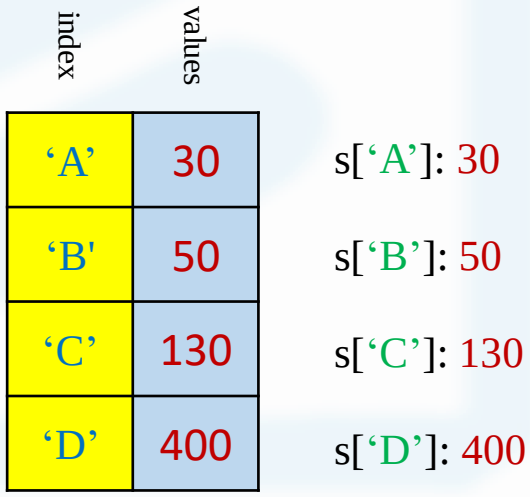
```
import pandas as pd
cities = {‘A’:30, ‘B’:50, ‘C’:130, ‘D’:400}
s = pd.Series(cities)
s.name=‘Distance’
s.index.name=‘Cities’
print(s)
print(s[ [‘A’,’C’] ])
print(s[ [0,2] ])
```

```
Cities
  A        30
  B        50
  C        130
  D        400
Name: Distance, dtype: int64

Cities
A    30
C    130
Name: Distance, dtype: int64

Cities
A    30
C    130
Name: Distance, dtype: int64
```

```
from pandas import Series
s = pd.Series([30, 50, 130, 400], index=[ ‘A' , ‘B' , ‘C' , ‘D' ])
s.name=‘Distance’
s.index.name=‘Cities’
print(s)
```



s.index: Index([‘A', ‘B', ‘C', ‘D'], dtype='object')
s.values : [30, 50, 130, 400]

Pandas Data Structures : Series

Slicing

❖ Accessing Elements of a Series

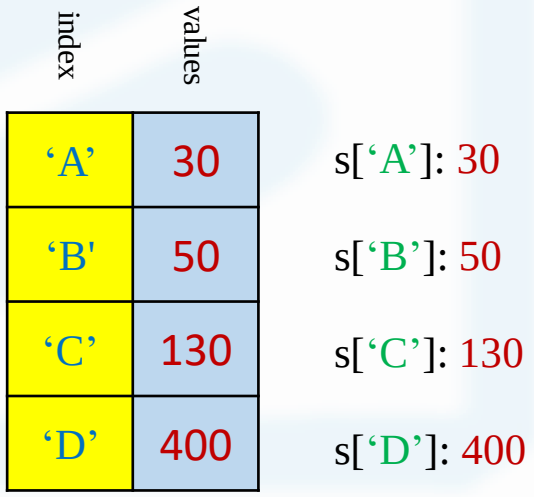
```
import pandas as pd
cities = {'A':30, 'B':50, 'C':130, 'D':400}
s = pd.Series(cities)
s.name='Distance'
s.index.name='Cities'
print(s)
print(s[ 'A':'C' ])
print(s[ 1:3 ])
```

```
Cities
A      30
B      50
C     130
D     400
Name: Distance, dtype: int64

Cities
A    30
B    50
C   130
Name: Distance, dtype: int64

Cities
B    50
C   130
Name: Distance, dtype: int64
```

```
from pandas import Series
s = pd.Series([30, 50, 130, 400], index=[ 'A' , 'B' , 'C' , 'D' ])
s.name='Distance'
s.index.name='Cities'
print(s)
```




```
s.index:    Index(['A', 'B', 'C', 'D'], dtype='object')
s.values :  [30, 50, 130, 400]
```

Pandas Data Structures : Series

Slicing

❖ Modifying values of a Series

```
import pandas as pd
cities = {'A':30, 'B':50, 'C':130, 'D':400}
s = pd.Series(cities)
s.name='Distance'
s.index.name='Cities'
print(s)
s[ 'B':'C' ]=125   # or s[ 1:3 ]=125
print(s)
```

index	values
'A'	30
'B'	50
'C'	130
'D'	400

s['A']: 30

s['B']: 50

s['C']: 130

s['D']: 400

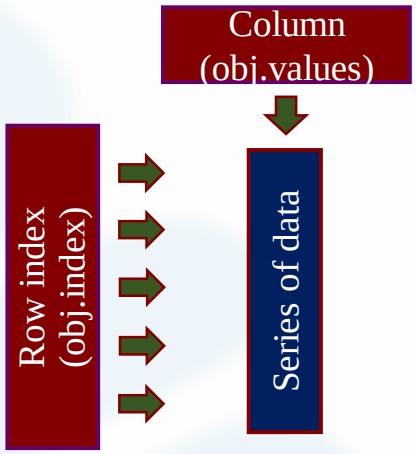
```
Cities
  A        30
  B        50
  C        130
  D        400
Name: Distance, dtype: int64
```

```
Cities
A    30
B    125
C    125
D    400
Name: Distance, dtype: int64
```

Pandas Data Structures : Series

❖ Attributes of Pandas Series:

- .name
- .index.name
- .values
- .index
- .size
- .empty
- …



❖ Methods of Pandas Series:

- head(n)
- count()
- tail(n)
- …

Pandas Data Structures : Series

head

❖ Methods of Pandas Series:

```
import numpy as np
import pandas as pd
seriesTenTwenty=pd.Series(np.arange( 10,20, 1 ))
```

```
print(seriesTenTwenty)
```

```
0    10
1    11
2    12
3    13
4    14
5    15
6    16
7    17
8    18
9    19
dtype: int32
```

```
seriesTenTwenty.head(2)
```

```
0    10
1    11
dtype: int32
```

```
seriesTenTwenty.head()
```

```
0    10
1    11
2    12
3    13
4    14
dtype: int32
```

Pandas Data Structures : Series

count

❖ Methods of Pandas Series:

```
import numpy as np
import pandas as pd
seriesTenTwenty=pd.Series(np.arange( 10,20, 1 ))
```

```
print(seriesTenTwenty)
0    10
1    11
2    12
3    13
4    14
5    15
6    16
7    17
8    18
9    19
dtype: int32
```

```
seriesTenTwenty.count()
10
```

Pandas Data Structures : Series

tail

❖ Methods of Pandas Series:

```
import numpy as np
import pandas as pd
seriesTenTwenty=pd.Series(np.arange( 10,20, 1 ))
```

```
print(seriesTenTwenty)
```

```
0    10
1    11
2    12
3    13
4    14
5    15
6    16
7    17
8    18
9    19
dtype: int32
```

```
seriesTenTwenty.tail(2)
```

```
8    18
9    19
dtype: int32
```

```
seriesTenTwenty.tail()
```

```
5    15
6    16
7    17
8    18
9    19
dtype: int32
```

Addition of two Series

❖ Mathematical Operations on Series

```
import pandas as pd

seriesA = pd.Series([10,20,30,40,50], index =['a', 'b', 'c', 'd', 'e'])
seriesB = pd.Series([100,200,-100,-500,500], index = ['z', 'y', 'a', 'c', 'e'])
print(seriesA + seriesB)
```

```
a    -90.0
b      NaN
c   -470.0
d      NaN
e    550.0
y      NaN
z      NaN
dtype: float64
```

index	value from seriesA	value from seriesB	seriesA + seriesB
a	10	-100	-90.0
b	20		NaN
c	30	-500	-470.0
d	40		NaN
e	50	500	550.0
y		200	NaN
z		100	Nan

Addition of two Series

❖ Mathematical Operations on Series

```
import pandas as pd

seriesA = pd.Series([10,20,30,40,50], index =['a', 'b', 'c', 'd', 'e'])
seriesB = pd.Series([100,200,-100,-500,500], index = ['z', 'y', 'a', 'c', 'e'])
seriesC=seriesA.add(seriesB, fill_value=1000)
print(seriesC )
```

```
a     -90.0
b    1020.0
c    -470.0
d    1040.0
e     550.0
y    1200.0
z    1100.0
dtype: float64
```

index	value from seriesA	value from seriesB	seriesA + seriesB
a	10	-100	-90.0
b	20	1000	1020.0
c	30	-500	-470.0
d	40	1000	1040
e	50	500	550.0
y	1000	200	1200.0
z	1000	100	1100.0

❖ Mathematical Operations on Series

Addition of two Series

```
seriesC= seriesA + seriesB
seriesC=seriesA.add(seriesB, fill_value=1000)
```

Subtraction of two Series

```
seriesC= seriesA - seriesB
seriesC=seriesA.sub(seriesB, fill_value=1000)
```

Multiplication of two Series

```
seriesC= seriesA * seriesB
seriesC=seriesA.mul(seriesB, fill_value=1000)
```

Division of two Series

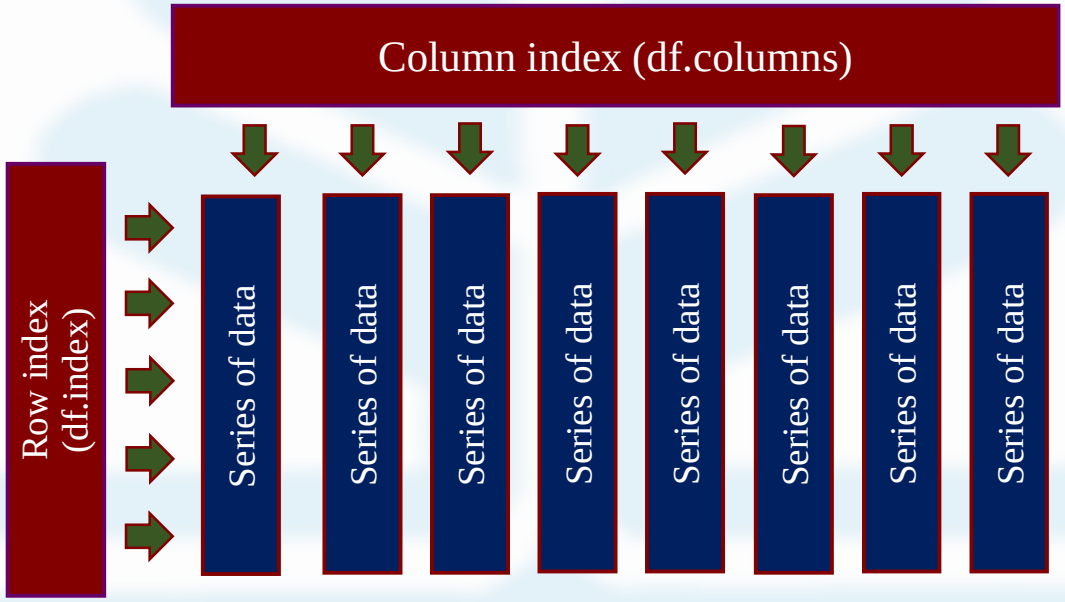
```
seriesC= seriesA / seriesB
seriesC=seriesA.div(seriesB, fill_value=1000)
```

Pandas Data Structures : DataFrame

DataFrame

A DataFrame represents a rectangular table of data and contains an ordered collection of columns, each of which can be a different value type (numeric, string, boolean, etc.).

The DataFrame has both a row and column index; it can be thought of as a dict of Series all sharing the same index.




```
import pandas as pd
```

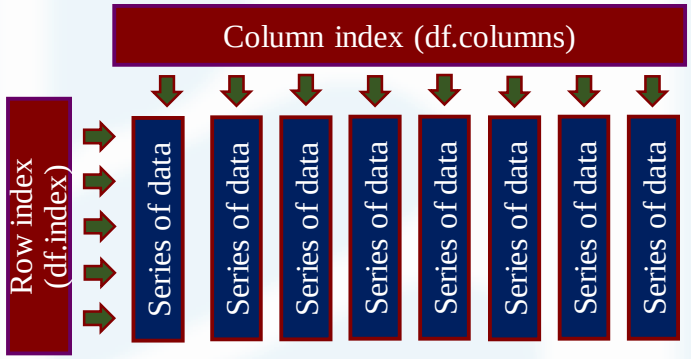
```
from pandas import DataFrame
```

A DataFrame is a two-dimensional labelled data structure like a table of MySQL. It contains rows and columns, and therefore has both a row and column index. Each column can have a different type of value such as numeric, string, boolean, etc., as in tables of a database.

Pandas Data Structures : DataFrame

❖ Attributes of Pandas DataFrame:

- .index
- .colomuns
- .values
- .dtypes
- .size
- .shape
- .T (to transpose)
- …



❖ Methods of Pandas DataFrame:

- head(n)
- count()
- tail(n)
- …

Pandas Data Structures : DataFrame

Creation of DataFrame

❖ Creation of an empty DataFrame

	CFP	NLA	ML
991	26	32	6
992	24	26	8
4001	28	39	7

```
import pandas as pd
df = pd.DataFrame()
print(df)
```

```
Empty DataFrame
Columns: []
Index: []
```

Pandas Data Structures : DataFrame

Creation of DataFrame

❖ Creation of DataFrame from NumPy ndarrays

	CFP	NLA	ML
991	26	32	6
992	24	26	8
4001	28	39	7

```
import numpy as np
import pandas as pd
Array_991 = np.array([26,32,6])
Array_992 = np.array([24,26,8])
Array_4001 = np.array([28,39,7, 31])
df = pd.DataFrame([Array_991, Array_992, Array_4001],
                  index=[991, 992, 4001],
                  columns=[ ‘CFP', ‘NLA', ‘ML', 'DS'])
```

print(df)

	CFP	NLA	ML	DS
991	26	32	6	NaN
992	24	26	8	NaN
4001	28	39	7	31.0

print(df.index)

RangeIndex(start=0, stop=3, step=1)

print(df.values)

```
[ [26.  32.   6.  nan]
  [24.  26.   8.  nan]
  [28.  39.   7.  31. ] ]
```

print(df.columns)

Index([‘CFP', ‘NLA', ‘ML', 'DS'], dtype='object')

Creation of DataFrame

❖ Creation of DataFrame from Lists of Lists

	CFP	NLA	ML
991	26	32	6
992	24	26	8
4001	28	39	7

```
import pandas as pd
# Specify values for each row:
df = pd.DataFrame([ [26, 32, 6],
                    [24, 26, 8],
                    [28, 39, 7]],
      index=[991, 992, 4001], columns=['CFP', 'NLA', 'ML'])
```

```
print(df)
```

	CFP	NLA	ML
991	26	32	6
992	24	26	8
4001	28	39	7

```
print(df.index)
Int64Index([991, 992, 4001], dtype='int64')
```

```
print(df.values)
[ [ 26  32  6]
  [24  26  8]
  [28  39  7] ]
```

```
print(df.columns)
Index(['CFP', 'NLA', 'ML'], dtype='object')
```

Pandas Data Structures : DataFrame

Creation of DataFrame

❖ Creation of DataFrame from Dictionary of Lists

	CFP	NLA	ML
991	26	32	6
992	24	26	8
4001	28	39	7

```
import pandas as pd
# Specify values for each column.
df = pd.DataFrame({ "CFP" : [26, 24, 28],
                    "NLA" : [32, 26, 39],
                    "ML" : [6, 8, 7] },  index = [991, 992, 4001])
```

```
print(df)
```

```
      CFP  NLA  ML
991    26   32   6
992    24   26   8
4001   28   39   7
```

```
print(df.index)
```

```
Int64Index([991, 992, 4001], dtype='int64')
```

```
print(df.values)
```

```
[ [ 26   32   6]
  [24   26   8]
  [28   39   7] ]
```

```
print(df.columns)
```

```
Index(['CFP', 'NLA', 'ML'], dtype='object')
```

Pandas Data Structures : DataFrame

Creation of DataFrame

❖ Creation of DataFrame from List of Dictionaries

	CFP	NLA	ML
991	26	32	6
992	24	26	8
4001	28	39	7

```
import pandas as pd
List_dict=[{ "CFP" : 26, "NLA" : 32, "ML" : 6 },
           { "CFP" : 24, "NLA" : 26, "ML" : 8 },
           { "CFP" : 28, "NLA" : 39, "ML" : 7 } ]
df = pd.DataFrame(list_dict,  index = [991, 992, 4001] )
```

print(df)

```
      CFP  NLA  ML
991   26   32    6
992   24   26    8
4001  28   39    7
```

print(df.index)

```
Int64Index([991, 992, 4001], dtype='int64')
```

print(df.values)

```
[ [ 26   32   6]
  [24   26   8]
  [28   39   7] ]
```

print(df.columns)

```
Index(['CFP', 'NLA', 'ML'], dtype='object')
```

Pandas Data Structures : DataFrame

Creation of DataFrame

❖ Creation of DataFrame from Series

	CFP	NLA	ML
991	26	32	6
992	24	26	8
4001	28	39	7

```
import pandas as pd
series991=pd.Series([26,32,6], index=['CFP','NLA','ML'])
series992=pd.Series([24,26,8], index=['CFP','NLA','ML'])
series4002=pd.Series([28,39,7], index=['CFP','NLA','ML'])
df = pd.DataFrame([series991,series992,series4002], index=[991,992,4002] )
```

print(df)

```
      CFP  NLA  ML
991    26   32   6
992    24   26   8
4001   28   39   7
```

print(df.index)

```
Int64Index([991, 992, 4001], dtype='int64')
```

print(df.values)

```
[ [ 26   32   6]
  [24   26   8]
  [28   39   7] ]
```

print(df.columns)

```
Index(['CFP', 'NLA', 'ML'], dtype='object')
```

Pandas Data Structures : DataFrame

Creation of DataFrame

❖ Creation of DataFrame from Dictionary of Series

	CFP	NLA	ML
991	26	32	6
992	24	26	8
4001	28	39	7

```
import pandas as pd
dict_series = {'CFP': pd.Series([26, 24,28], index=[991,992,4001] ,
                'NLA': pd.Series([32,26,39], index =[991,992,4001] ,
                'ML':   pd.Series([6,8,7],      index =[991,992,4001] }

df = pd.DataFrame(dict_series)
```

print(df)

```
      CFP  NLA  ML
991   26   32    6
992   24   26    8
4001  28   39    7
```

print(df.values)

```
[ [ 26   32   6]
  [24   26   8]
  [28   39   7] ]
```

print(df.index)

```
Int64Index([991, 992, 4001], dtype='int64')
```

print(df.columns)

```
Index(['CFP', 'NLA', 'ML'], dtype='object')
```

type(df.CFP)

```
<class 'pandas.core.series.Series'>
```

Pandas Data Structures : DataFrame

❖ Modifying a Column of a DataFrame

	CFP	NLA	ML
991	26	32	6
992	24	26	8
4001	28	39	7

```
import pandas as pd
df = pd.DataFrame([ [26, 32, 6],
                    [24, 26, 8],
                    [28, 39, 7]],
      index=[991, 992, 4001], columns=['CFP', 'NLA', 'ML'])
df['ML'] = [9,12,11]
```

print(df)

```
      CFP  NLA  ML
991   26   32   9
992   24   26   12
4001  28   39   11
```

print(df.index)

```
Int64Index([991, 992, 4001], dtype='int64')
```

print(df.columns)

```
Index(['CFP', 'NLA', 'ML'], dtype='object')
```

print(df.values)

```
[ [ 26   32   9]
  [24   26   12]
  [28   39   11] ]
```

❖ Adding a New Column to a DataFrame

	CFP	NLA	ML
991	26	32	6
992	24	26	8
4001	28	39	7

```
import pandas as pd
df = pd.DataFrame([ [26, 32, 6],
                    [24, 26, 8],
                    [28, 39, 7]],
     index=[991, 992, 4001], columns=['CFP', 'NLA', 'ML'])
df['DS'] = [34,32,37]
```

```
print(df)
```

	CFP	NLA	ML	DS
991	26	32	6	34
992	24	26	8	32
4001	28	39	7	37

```
print(df.index)
Int64Index([991, 992, 4001], dtype='int64')

print(df.columns)
Index(['CFP', 'NLA', 'ML', 'DS'], dtype='object')
```

```
print(df.values)
[ [ 26   32   6   34]
  [24   26   8   32]
  [28   39   7   37] ]
```

Pandas Data Structures : DataFrame

Operations on rows and columns in DataFrames

❖ Modifying a New Row to a DataFrame

	CFP	NLA	ML
991	26	32	6
992	24	26	8
4001	28	39	7

```
import pandas as pd
df = pd.DataFrame([ [26, 32, 6],
                    [24, 26, 8],
                    [28, 39, 7]],
     index=[991, 992, 4001], columns=['CFP', 'NLA', 'ML'])
df.loc[992] = [25,25,25]   # df.loc[992] = 25
```

```
print(df)
```

```
      CFP  NLA  ML
991    26   32   6
992    25   25  25
4001   28   39   7
```

```
print(df.index)
Int64Index([991, 992, 4001], dtype='int64')
```

```
print(df.columns)
Index(['CFP', 'NLA', 'ML'], dtype='object')
```

```
print(df.values)
```

```
[ [ 26   32   9]
  [25   25   5]
  [28   39   11] ]
```

Pandas Data Structures : DataFrame

❖ Modifying a New Row to a DataFrame

```
import pandas as pd
df = pd.DataFrame([ [26, 32, 6],
                    [24, 26, 8],
                    [28, 39, 7]],
    index=[991, 992, 4001], columns=['CFP', 'NLA', 'ML'])
```

	CFP	NLA	ML
991	26	32	6
992	24	26	8
4001	28	39	7

df.loc[992] = 0

	CFP	NLA	ML
991	26	32	6
992	0	0	0
4001	28	39	7

df.loc[:] = 0

	CFP	NLA	ML
991	0	0	0
992	0	0	0
4001	0	0	0

❖ Adding a New Row to a DataFrame

	CFP	NLA	ML
991	26	32	6
992	24	26	8
4001	28	39	7

```
import pandas as pd
df = pd.DataFrame([ [26, 32, 6],
                    [24, 26, 8],
                    [28, 39, 7]],
      index=[991, 992, 4001], columns=['CFP', 'NLA', 'ML'])
df.loc[4002] = [34,36,10]
```

print(df)

	CFP	NLA	ML
991	26	32	6
992	25	25	25
4001	28	39	7
4002	34	36	10

print(df.index)

```
Int64Index([991, 992, 4001,4002], dtype='int64')
```

print(df.columns)

```
Index(['CFP', 'NLA', 'ML'], dtype='object')
```

print(df.values)

```
[ [ 26   32   9 ]
  [25   25   5 ]
  [28   39   11]
  [34   36   10] ]
```

Pandas Data Structures : DataFrame

Operations on rows and columns in DataFrames

❖ Deleting Columns from a DataFrame

	CFP	NLA	ML
991	26	32	6
992	24	26	8
4001	28	39	7

```
import pandas as pd
df = pd.DataFrame([ [26, 32, 6],
                    [24, 26, 8],
                    [28, 39, 7]],
      index=[991, 992, 4001], columns=['CFP', 'NLA', 'ML'])
newdf = df.drop("NLA", axis='columns')     #  newdf = df.drop("NLA", axis=1)
```

```
print(new_df)
```

```
      CFP  ML
991   26    6
992   25    8
4001  28    7
```

```
print(new_df.index)
Int64Index([991, 992, 4001], dtype='int64')
```

```
print(new_df.columns)
Index(['CFP', 'ML'], dtype='object')
```

```
print(new_df.values)
```

```
[ [ 26   6 ]
  [25   8 ]
  [28   7 ] ]
```

```
newdf = df.drop(["NLA“, “ML”], axis='columns')
newdf = df.drop(["NLA“, “ML”], axis=1')
```

❖ Deleting Rows from a DataFrame

	CFP	NLA	ML
991	26	32	6
992	24	26	8
4001	28	39	7

```
import pandas as pd
df = pd.DataFrame([ [26, 32, 6],
                    [24, 26, 8],
                    [28, 39, 7]],
      index=[991, 992, 4001], columns=['CFP', 'NLA', 'ML'])
newdf = df.drop(992, axis='cols')     #  newdf = df.drop(992, axis=0)
```

```
print(new_df)
```

	CFP	NLA	ML
991	26	32	6
4001	28	39	7

```
print(new_df.index)
Int64Index([991, 4001], dtype='int64')
```

```
print(new_df.columns)
Index(['CFP', 'NLA', 'ML'], dtype='object')
```

```
print(new_df.values)
[ [ 26   32   6]
  [28   39   7] ]
```

```
newdf = df.drop([991,992], axis='rows')
newdf = df.drop([991,992], axis=0)
```

❖ Renaming Row Labels of a DataFrame

	CFP	NLA	ML
991	26	32	6
992	24	26	8
4001	28	39	7

```
import pandas as pd
df = pd.DataFrame([ [26, 32, 6],
                    [24, 26, 8],
                    [28, 39, 7]],
      index=[991, 992, 4001], columns=['CFP', 'NLA', 'ML'])
new_df=df.rename({992:'1399_2',4001:'1400_2'}, axis='index')
```

```
print(new_df)
```

	CFP	NLA	ML
991	26	32	6
1399_2	24	26	8
1400_2	28	39	7

```
print(new_df.index)
Index([991, '1399_2', '1400_2'], dtype='object')
```

```
print(new_df.columns)
Index(['CFP', 'NLA', 'ML'], dtype='object')
```

```
print(new_df.values)
[[26 32  6]
 [24 26  8]
 [28 39  7]]
```

❖ Renaming Column Labels of a DataFrame

	CFP	NLA	ML
991	26	32	6
992	24	26	8
4001	28	39	7

```
import pandas as pd
df = pd.DataFrame([ [26, 32, 6],
                    [24, 26, 8],
                    [28, 39, 7]],
      index=[991, 992, 4001], columns=['CFP', 'NLA', 'ML'])
new_df=df.rename({'NLA':'NumLA'}, axis='columns')
```

```
print(new_df)
```

	CFP	NumLA	ML
991	26	32	6
992	24	26	8
4002	28	39	7

```
print(new_df.values)
```

```
[[26 32  6]
 [24 26  8]
 [28 39  7]]
```

```
print(new_df.index)
Index([991, '992', '4002'], dtype='object')
```

```
print(new_df.columns)
Index(['CFP', 'NumLA', 'ML'], dtype='object')
```

Pandas Data Structures : DataFrame

Accessing DataFrames Element through Indexing

❖ Label Based Indexing

	CFP	NLA	ML
991	26	32	6
992	24	26	8
4001	28	39	7

```
import pandas as pd
df = pd.DataFrame([ [26, 32, 6],
                    [24, 26, 8],
                    [28, 39, 7]],
      index=[991, 992, 4001], columns=['CFP', 'NLA', 'ML'])
```

```
print(df.loc[992])
```

```
CFP    24
NLA    26
ML     8
Name: 992, dtype: int64
```

```
print(df.loc[:,'ML'])
```

```
991     6
992     8
4001    7
Name: ML, dtype: int64
```

Accessing DataFrames Element through Indexing

❖ Boolean Indexing

	CFP	NLA	ML
991	26	32	6
992	24	26	8
4001	28	39	7

```
import pandas as pd
df = pd.DataFrame([ [26, 32, 6],
                    [24, 26, 8],
                    [28, 39, 7]],
      index=[991, 992, 4001], columns=['CFP', 'NLA', 'ML'])
```

print(df.loc[992]<15)

```
CFP     False
NLA     False
ML      True
Name: 992, dtype: bool
```

print(df.loc[:,'ML']>=7)

```
991     False
992     True
4001    True
Name: ML, dtype: bool
```

print(df['ML']>=7)

```
991     False
992     True
4001    True
Name: ML, dtype: bool
```

Pandas Data Structures : DataFrame

Accessing DataFrames Element through Slicing

❖ Slicing

	CFP	NLA	ML
991	26	32	6
992	24	26	8
4001	28	39	7

```
import pandas as pd
df = pd.DataFrame([ [26, 32, 6],
                    [24, 26, 8],
                    [28, 39, 7]],
      index=[991, 992, 4001], columns=['CFP', 'NLA', 'ML'])
```

print(df.loc[992:4002])

```
      CFP  NLA  ML
992    24   26   8
4002   28   39   7
```

	CFP	NLA	ML
991	26	32	6
992	24	26	8
4001	28	39	7

print(df.loc[992:4002,'ML'])

```
992     8
4001    7
Name: ML, dtype: int64
```

	CFP	NLA	ML
991	26	32	6
992	24	26	8
4001	28	39	7

print(df.loc[992:4002,'NLA':'ML'])

```
      NLA  ML
992    26   8
4001   39   7
Name: ML, dtype: int64
```

	CFP	NLA	ML
991	26	32	6
992	24	26	8
4001	28	39	7

Pandas Data Structures : DataFrame

Accessing DataFrames Element through Slicing

❖ Filtering Rows in DataFrames

	CFP	NLA	ML
991	26	32	6
992	24	26	8
4001	28	39	7

```
import pandas as pd
df = pd.DataFrame([ [26, 32, 6],
                    [24, 26, 8],
                    [28, 39, 7]],
      index=[991, 992, 4001], columns=['CFP', 'NLA', 'ML'])

new_df=df.loc[ [True, False, True] ]
print(new_df)
```

	CFP	NLA	ML
991	26	32	6
4002	28	39	7

	CFP	NLA	ML
991	26	32	6
4001	28	39	7

Pandas Data Structures : DataFrame

❖ Joining of DataFrames

```
import pandas as pd
df1 = pd.DataFrame([ [26, 32, 6],
                     [24, 26, 8],
                     [28, 39, 7]],
      index=[991, 992, 4001], columns=['CFP', 'NLA', 'ML'])
```

	CFP	NLA	ML
991	26	32	6
992	24	26	8
4001	28	39	7

```
df2 = pd.DataFrame([ [21,27,12],
                     [18,29,9]],
      index=[992, 4002], columns=['CFP', 'DS', 'DL'])
```

	CFP	DS	DL
992	21	27	12
4002	18	29	9

```
df=df1.append(df2)
print(df)
```

	CFP	NLA	ML	DS	DL
991	26	32	6	NaN	NaN
992	24	26	8	NaN	NaN
4001	28	39	7	NaN	NaN
992	21	NaN	NaN	27	12
4002	18	NaN	NaN	29	9

df=df1.append(df2, sort=False)

Pandas Data Structures : DataFrame

Joining, Merging and Concatenation of DataFrames

❖ Joining of DataFrames(ignore_index=True)

```
import pandas as pd
df1 = pd.DataFrame([ [26, 32, 6],
                     [24, 26, 8],
                     [28, 39, 7]],
      index=[991, 992, 4001], columns=['CFP', 'NLA', 'ML'])
```

	CFP	NLA	ML
991	26	32	6
992	24	26	8
4001	28	39	7

```
df2 = pd.DataFrame([ [21,27,12],
                     [18,29,9]],
      index=[992, 4002], columns=['CFP', 'DS', 'DL'])
```

	CFP	DS	DL
992	21	27	12
4002	18	29	9

```
df=df1.append(df2, ignore_index=True)
print(df)
```

	CFP	NLA	ML	DS	DL
0	26	32	6	NaN	NaN
1	24	26	8	NaN	NaN
2	28	39	7	NaN	NaN
3	21	NaN	NaN	27	12
4	18	NaN	NaN	29	9

Pandas Data Structures : DataFrame

pd.melt(df)

Gather columns into rows.

	CFP	NLA	ML
991	26	32	6
992	24	26	8

	variable	value
991	CFP	26
992	CFP	24
991	NLA	32
992	NLA	26
991	ML	6
992	ML	8

```
import pandas as pd
df1 = pd.DataFrame([ [26, 32, 6],
                     [24, 26, 8]],
          index=[991, 992], columns=['CFP', 'NLA', 'ML'])
```

```
df3=pd.melt(df, ignore_index=False)
print(df3)
```

Pandas Data Structures : DataFrame

df.pivot(columns='variable', values='value')

Spread rows into columns.

	variable	value
991	CFP	26
992	CFP	24
991	NLA	32
992	NLA	26
991	ML	6
992	ML	8

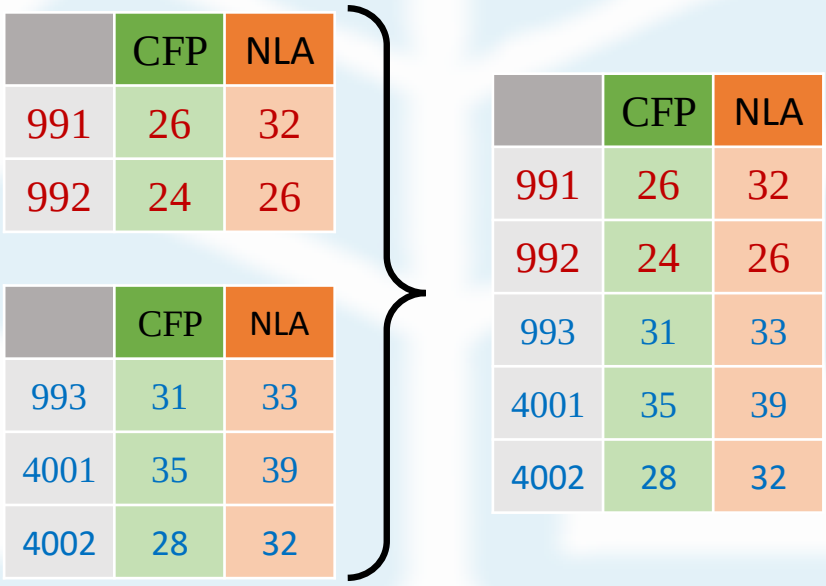
	CFP	NLA	ML
991	26	32	6
992	24	26	8

```
df4=df3.pivot(columns='variable', values='value')
print(df4)
```

Pandas Data Structures : DataFrame

pd.concat([df1,df2])

Append rows of DataFrames



	CFP	NLA
991	26	32
992	24	26

	CFP	NLA
993	31	33
4001	35	39
4002	28	32

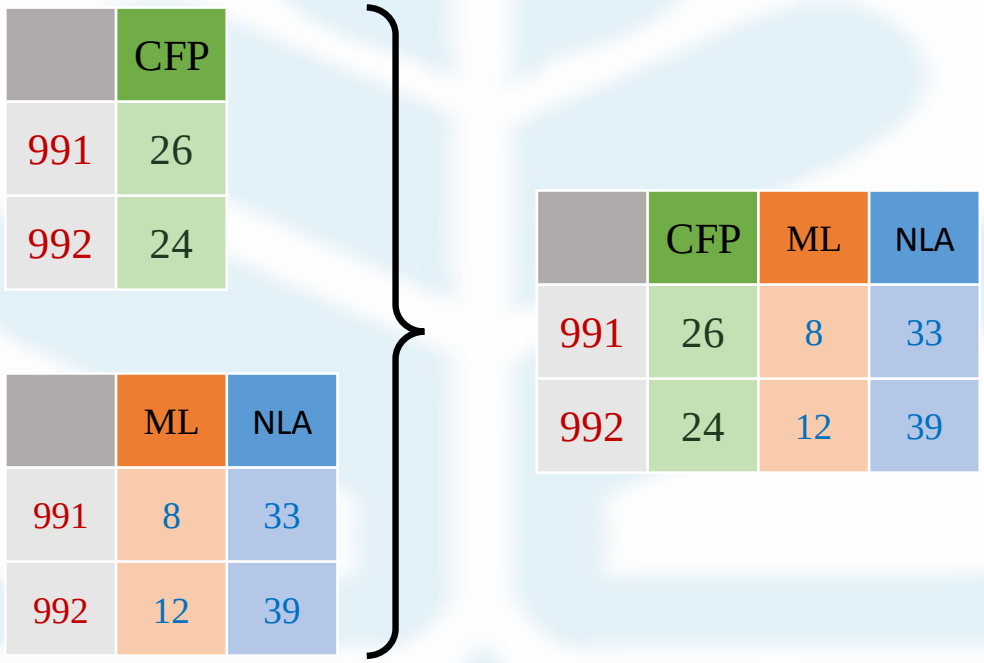
	CFP	NLA
991	26	32
992	24	26
993	31	33
4001	35	39
4002	28	32

Pandas Data Structures : DataFrame

Joining, Merging and Concatenation of DataFrames

```
pd.concat([df1,df2], axis=1)
```

Append columns of DataFrames



	CFP
991	26
992	24

	ML	NLA
991	8	33
992	12	39

	CFP	ML	NLA
991	26	8	33
992	24	12	39

Pandas Data Structures : DataFrame

Attributes

❖ **.T, .shape, .size, .dtypes**

	CFP	NLA	ML
991	26	32	6.0
992	24	26	8.0
4001	28	39	7.0

```
import pandas as pd
df = pd.DataFrame([ [26, 32, 6],
                    [24, 26, 8],
                    [28, 39, 7]],
      index=[991, 992, 4001], columns=['CFP', 'NLA', 'ML'])
```

print(df.T)

	991	992	4001
CFP	26	24	28
NLA	32	26	39
ML	6.0	8.0	7.0

print(df.shape)

```
(3,3)
```

print(df.size)

```
9
```

	991	**992**	**4001**
CFP	26	24	28
NLA	32	26	39
ML	6.0	8.0	7.0

print(df.dtypes)

```
CFP      int64
NLA      int64
ML     float64
dtype: object
```

Pandas Data Structures : DataFrame

methods

❖ .head(n), .tail(n)

	CFP	NLA	ML
991	26	32	6.0
992	24	26	8.0
4001	28	39	7.0
4002	32	35	8.0
981	23	40	7
982	36	32	8

```
import pandas as pd
df = pd.DataFrame([ [26, 32, 6],
                    [24, 26, 8],
                    [28, 39, 7],
                    [32,35,8.0],
                    [23,40,7.0],
                    [36,32,8]],
    index=[991, 992, 4001,4002,981,982], columns=['CFP', 'NLA', 'ML'])
```

print(df.head(2)

	CFP	NLA	ML
991	26	32	6
992	24	26	8

print(df.head()

	CFP	NLA	ML
991	26	32	6.0
992	24	26	8.0
4001	28	39	7.0
4002	32	35	8.0
981	23	40	7.0

print(df.tail(3)

	CFP	NLA	ML
4002	32	35	8.0
981	23	40	7.0
982	36	32	8.0

Pandas Data Structures : DataFrame

Importing and exporting data between .xlsx files and dataFrames

❖ Exporting data (.xlsx)

	CFP	NLA	ML
991	26	32	6.0
992	24	26	8.0
4001	28	39	7.0
4002	32	35	8.0
981	23	40	7
982	36	32	8

```
import pandas as pd
#ID=pd.ser
# creating the DataFrame
df = pd.DataFrame([ [26, 32, 6],
                    [24, 26, 8],
                    [28, 39, 7],
                    [32,35,8.0],
                    [23,40,7.0],
                    [36,32,8]],
    index=[991, 992, 4001,4002,981,982], columns=['CFP', 'NLA', 'ML'])

# Exporting a Pandas DataFrame to an Excel file
file_name = ‘Course.xlsx'
df.to_excel(file_name)
print('DataFrame is written to Excel File successfully.')
```

Pandas Data Structures : DataFrame

Importing and exporting data between .xlsx files and dataFrames

❖ Importing data (.xlsx)

```
# Import an Excel File into Python using Pandas
data = pd.read_excel(r'Course.xlsx')
df = pd.DataFrame(data)
print (df)
```

		CFP	NLA	ML
0	991	26	32	6
1	992	24	26	8
2	4001	28	39	7
3	4002	32	35	8
4	981	23	40	7
5	982	36	32	8

	CFP	NLA	ML
991	26	32	6.0
992	24	26	8.0
4001	28	39	7.0
4002	32	35	8.0
981	23	40	7
982	36	32	8

Pandas Data Structures : DataFrame

Importing and exporting data between .xlsx files and dataFrames

❖ Exporting data (csv)

```
import pandas as pd
#ID=pd.ser
# creating the DataFrame
df = pd.DataFrame([ [26, 32, 6],
                    [24, 26, 8],
                    [28, 39, 7],
                    [32,35,8.0],
                    [23,40,7.0],
                    [36,32,8]],
    index=[991, 992, 4001,4002,981,982], columns=['CFP', 'NLA', 'ML'])

# Exporting a Pandas DataFrame to an Excel file
file_name = 'Course.csv'
df.to_csv(file_name)
print('DataFrame is written to csv File successfully.')
```

	CFP	NLA	ML
991	26	32	6.0
992	24	26	8.0
4001	28	39	7.0
4002	32	35	8.0
981	23	40	7
982	36	32	8

Pandas Data Structures : DataFrame

Importing and exporting data between .xlsx files and dataFrames

❖ Importing data (.csv)

```
# Import an Excel File into Python using Pandas
data = pd.read_csv(r'Course.csv')
df = pd.DataFrame(data)
print (df)
```

		CFP	NLA	ML
0	991	26	32	6
1	992	24	26	8
2	4001	28	39	7
3	4002	32	35	8
4	981	23	40	7
5	982	36	32	8

	CFP	NLA	ML
991	26	32	6.0
992	24	26	8.0
4001	28	39	7.0
4002	32	35	8.0
981	23	40	7
982	36	32	8

مراجع

1- https://pandas.pydata.org.

2- Jason, Brownlee. **Basics for Linear Algebra for Machine Learning Discover the Mathematical Language of Data in Python**, 2018.

3- Stephen Klosterman. **Data Science Projects with Python A case study approach to successful data science projects using Python, pandas, and scikit-learn**, 2019.

4- https://ncert.nic.in/textbook/pdf/leip102.pdf.

https://github.com/Abhishek-Arora/Image-Classification-Using-SVM