D.A.V. COLLEGE MANAGING COMMITTEE

ओ३म्

ESTD 1886

MRA DAV PUBLIC SCHOOL

INFORMATICS PRACTICES

Code No-065

CLASS-XII

2022-2023

Blue Print:

Unit No	Unit Name	Marks
1	Data Handling using Pandas and Data Visualization	30
2	Database Query using SQL	25
3	Introduction to Computer Networks	7
4	Societal Impacts	8
	Practical	30
	Total	100

Data Handling Using Pandas - I

By

Rajesh Verma

What is Pandas?

Pandas is a software **library** for the **Python** programming language written by Wes McKinney for data manipulation and analysis. The name Pandas is derived from the term “Panel Data”. It is an open source and free to use (under a BSD license). It takes data (like a CSV or TSV file, or a SQL database) and creates a Python object with rows and columns called data frame that looks very similar to Excel.

Pandas Data Types

1. Series
2. DataFrame
3. Panel

Data Structure	Dimensions	Description
Series	1	1D labelled homogeneous, data-mutable, size-immutable array.
DataFrame	2	2D labelled heterogeneous, data-mutable, size-mutable array.
Panel	3	3D labelled, data-mutable, size-mutable array.

KEY FEATURES OF PANDAS

✓Fast and efficient DataFrame object with default and customized indexing.

✓Tools for loading data from different file formats.

✓Data alignment and integrated handling of missing data.

✓Reshaping and pivoting of data sets.

✓Label-based slicing and indexing of large data sets.

✓Deletion/Insertion of columns from/to a data structure.

✓Group by data for aggregation and transformations.

✓High performance merging and joining of data.

INSTALLING PANDAS

To install Pandas from command line, type:

```
pip install pandas
```

Introduction To Series

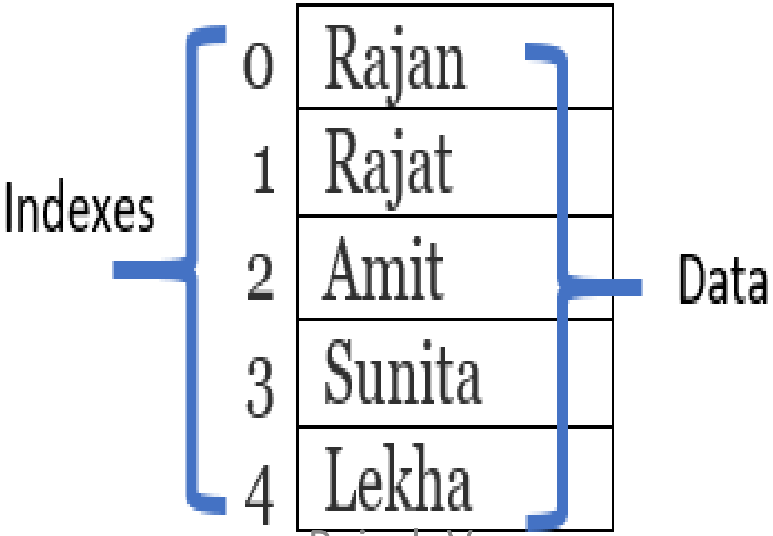
- A Series is a list of values with default integer index.
- It is one-dimensional homogeneously-typed array.

Syntax:

```
<Series Name>=<pd>.Series(<list name>, ...)
```

Characteristics of Series

1. Series is a one-dimensional labelled array capable of holding homogenous data of any type (integer, string, float etc.).
2. The data labels in series are numeric starting from 0 by default. The data labels are called as indexes.
3. The data in series is mutable i.e. it can be changed but the size of series is immutable i.e. size of the series cannot be changed.



Indexes

0	Rajan
1	Rajat
2	Amit
3	Sunita
4	Lekha

Data

CREATING A SERIES

CREATING A SERIES

Pandas Series can be created from the lists, dictionary, and from a scalar value etc.

SYNTAX

```
pandas.Series( data, index, name)
```

Where:

- **data:** takes various forms like ndarray, list, constants/scalar values, dictionary, mathematical expression
- **index:** are unique and hashable with same length as data. Default is np.arrange(n) if no index is passed.
- **name:** allows you to give a name to a Series object

Creation of Series

Creation of Series:

We can create a pandas series in following ways-

- From arrays
- From Lists
- From Dictionaries
- From scalar value

Creating a series from array

To create a series from array, first import a numpy and pandas module. Use array() function to create an array and then use Series() function to create a series from the array. By default, indexes are created from 0 till n-1 where n is the length of the array.

Example:

```
import pandas as pd

import numpy as np

data = np.array (['W','e','l','c','o','m','e'])

series1 = pd.Series(data)

print(series1)
```

OUTPUT

```
0   W

1    e

2    l

3    c

4    o

5    m

6    e

dtype: object
```

Creating a series from array with customized index values

In order to create a series from array with customized index values, we pass index as parameter in the Series() function with same number of elements as is present in the array.

Example:

```
import pandas as pd

import numpy as np

data = np.array([5400,2500,7634,8725])

# providing an index

ser = pd.Series(data, index =['North', 'East',

'South', 'West'])

print(ser)
```

OUTPUT

```
North     5400

East      2500

South     7634

West      8725


dtype: int32
```

Creating a series from Lists

In order to create a series from list, we have to first create a list after that we can create a series from list.

Example:

```
import pandas as pd

list = ['W','e','l','c','o','m','e']

series2 = pd.Series(list)

print(series2)
```

OUTPUT

```
0   W

1    e

2    l

3    c

4    o

5    m

6    e

dtype: object
```

Creating a series from Dictionary

In order to create a series from dictionary, we have to first create a dictionary after that we can make a series using dictionary. Dictionary key are used to construct index.

Example:

```
import pandas as pd

# a simple dictionary

dict = {'Ankit' : 1, 'Ramit' : 2, 'Geetika' : 3}

# create series from dictionary

ser = pd.Series(dict)

print(ser)
```

OUTPUT

```
Ankit     1
Ramit      2
Geetika    3
```

Here, keys of the dictionary become the indexes of the series.

Creating a series from Scalar value

In order to create a series from scalar value, an index must be provided. The scalar value will be repeated to match the length of index.

Example:

```
import pandas as pd

# giving a scalar value with index

ser = pd.Series(10, index =[0, 1, 2, 3, 4, 5])

print(ser)
```

OUTPUT

```
0   10
1   10
2   10
3   10
4   10
5   10
```

Here, all the elements of a series are filled with a scalar value 10

Create a series using Mathematical expression or Mathematical functions

Example:

```
import pandas as pd

import numpy as np

num=np.arange(10,60,10)

s1=pd.Series(index=num, data=num*2)

print(s1)
```

OUTPUT

```
10    20
20    40
30    60
40    80
50   100
dtype: int32
```

Important Attributes of Series

Attribute	**Meaning**
shape	Returns a tuple of shape of the data.
index	The index of series.
dtype	Returns the data type of the data.
ndim	Returns the number of dimensions in the data.
hasnans	Returns True if there are any NaN values, otherwise returns false.
is_unique	Returns boolean if values in the object are unique.
size	Returns the number of elements in the underlying data.
values	Returns Series as ndarray.
iloc	Purely integer-location based indexing for selection by position.
loc	Returns value based on the given label.
	Rajesh Verma

Series

Examples:

```
wages = pd.Series
([500,450,400,550,600])

wages
Output:
0     500
1     450
2     400
3     550
4     600
dtype: int64
```

```
wages =
pd.Series([500,450,400,550,600],
index=['Mon','Tue','Wed','Thu','Fri’])

wages
Output
Mon     500
Tue     450
Wed     400
Thu     550
Fri     600
dtype: int64
```

Attributes Examples

```
wages.index
Index(['Mon', 'Tue', 'Wed', 'Thu', 'Fri'], dtype='object’)

wages.is_unique
True

wages.size
5

wages.values
array([500, 450, 400, 550, 600], dtype=int64)

wages.iloc[1]
450

wages.loc['Tue']
450
```

Accessing elements of a series

Elements of a series can be easily accessed using the following methods:

- Indexing
- Slicing

Let use learn about these methods in detail.

Indexing Method

There are two types of indexes:

1. Positional indexes
2. Label indexes.

Positional indexes

Positional indexes are used to extract a data element present at a particular index location from a series. The index operator [ ] along with the index number can be used to access an element in a series. Remember the indexes starts from 0.

Example:

```
import pandas as pd
import numpy as np
data = np.array([5400,2500,7634,8725])
# providing an index
ser = pd.Series(data, index =['North', 'East',
'South', 'West'])
print(ser[1])
```

OUTPUT

```
2500
```

Here, the data element present at index location 1 i.e. 2500 is accessed and displayed

Label indexes

Label indexes are used to extract a data element present at a particular index label from a series. The index operator [ ] along with the label index can be used to access an element in a series.

Example:

```
import pandas as pd

import numpy as np

data = np.array([5400,2500,7634,8725])

# providing an index

ser = pd.Series(data, index =['North', 'East',

'South', 'West'])

print(ser['South'])
```

OUTPUT

```
7634
```

Here, the data element present at label index ‘South’ i.e. 7634 is accessed and displayed.

Positional or Label indexes

Multiple values can also be accessed by giving a list of positional or label indexes.

Example:

```
import pandas as pd
import numpy as np
data = np.array([5400,2500,7634,8725])
# providing an index
ser = pd.Series(data, index =['North', 'East',
'South', 'West'])
print(ser[[1,3]])
```

OUTPUT

```
East    2500
West    8725
dtype: int32
```

Here, the data corresponding to 1st and 3rd index is displayed.

Boolean Indexing

Boolean indexing is a type of indexing which uses actual values of the data in the Series. Using Boolean indexing we can filter data by applying certain condition on data using relational operators like ==, >, <, <=, >= and logical operators like ~(not), &(and) and |(or).

Example:

Consider the following series:

```
s1=[2,5,9,12,34,56]

import pandas as pd

ser1=pd.Series(s1)
```

Examples

Example:

print(ser1>10)

Here, entire series is displayed with False value at places where value<=10 and True value at places where value>10

OUTPUT

```
0   False
1    False
2    False
3    True
4    True
5    True
dtype: bool
```

print(ser1[ser1>10])

Here, elements with value>10 are displayed.

OUTPUT

```
3    12
4    34
5    56
dtype: int64
```

Examples

Example:

```
print(ser1[~(ser1>10)])
```

Here, elements with value>10 are not displayed.

OUTPUT

```
0    2
1    5
2    9
dtype: int64
```

```
print(ser1[(ser1>10) & (ser1<30)])
```

Here, elements with value>10 and value<30 are displayed.

OUTPUT

```
3    12
dtype: int64
```

Slicing Method

Slicing is used to extract a subset of a series. You can specify beginning parameter (beg) and end parameter (end) to indicate the size of the slice to be extracted from the series.

SYNTAX

Series_name[beg:end]

When positional indices are used for slicing, the value of end index position is excluded and when label indexing is used for slicing, the value of end index label is included.

Slicing Method

Example:

```
import pandas as pd

subject=['French', 'English', 'Maths', 'Geography','Science']     # list of subjects

Girls=[20,30,36,38,45]          #No. Of girls

series1 = pd.Series(Girls,index=subject)

print(series1[1:3])
```

OUTPUT

```
English    30
Maths      36
dtype: int64
```

Here, the data corresponding to 1st and 2nd index elements is displayed and the last index element of the slice ie 3 is not shown in the output.

Slicing Method

Example:

```
import pandas as pd

subject=['French', 'English', 'Maths', 'Geography',

'Science']     # list of subjects

Girls=[20,30,36,38,45]

series1 = pd.Series(Girls,index=subject)

print(series1['English':'Maths'])
```

OUTPUT

```
English    30
Maths      36
dtype: int64
```

Here, values corresponding to the both the labels i.e. 'English' and 'Maths' are included in the output.

Slicing Method

The data of the series can be easily modified using indexing and slicing methods.

Example:

```
import pandas as pd
subject=['French', 'English', 'Maths',
'Geography', 'Science']     # list of subjects
Girls=[20,30,36,38,45]


series1 = pd.Series(Girls,index=subject)
series1['English':'Maths']=90
series1[4]=100
print(series1)
```

OUTPUT

```
French       20
English      90
Maths        90
Geography    38
Science      100
dtype: int64
```

Boolean Expression

It is also possible to filter values on the basis of **Boolean expression**

Example:

```
import pandas as pd

ser1=pd.Series([20,30,36,38,45],

index=['A','B','C','D','E'])

print(ser1[ser1>25])
```

OUTPUT

```
B   30
C   36
D    38
E   45
dtype: int64
```

Example:

```
import pandas as pd

ser1=pd.Series([20,30,36,38,45],

index=['A','B','C','D','E'])

print([ser1>25])
```

```
A    False
B     True
C     True
D      True
E     True
dtype: bool
```

Methods of Series

Head and Tail functions

head(): The ***head function*** is used to return a specified number of rows from the beginning of a Series. The function returns a new Series.

Example:

```
import pandas as pd

subject=['French', 'English', 'Maths', 'Geography',

'Science']     # list of subjects

Girls=[20,30,36,38,45]

series1 = pd.Series(Girls,index=subject)

series1['English':'Maths']=90

data=series1.head(3)

print(data)
```

OUTPUT

```
French     20
English    90
Maths      90
dtype: int64
```

Here, three rows from the beginning of the series are displayed.

Methods of Series

Tail functions

tail(): The ***tail function*** is used to return a specified number of rows from the end of a Series. The function returns a new Series.

Example:

```
import pandas as pd
subject=['French', 'English', 'Maths', 'Geography',
'Science']     # list of subjects
Girls=[20,30,36,38,45]
series1 = pd.Series(Girls,index=subject)
series1['English':'Maths']=90
data=series1.tail(3)
print(data)
```

OUTPUT

```
Maths        90
Geography    38
Science      45
dtype: int64
```

Here, three rows from the end of the series are displayed.

Methods of Series

Count functions

count(): To count the number of values, present in a series

Example:

```
import pandas as pd
val=[10,30,27,68,85,45]
s1=pd.Series(val)
print("Count of values present in series s1:",s1.count())
```

OUTPUT

```
Count of values present in series s1: 6
```

Methods of Series

Sum functions

sum() To add all values present in a series

Example:

```
import pandas as pd
val=[10,30,27,68,85,45]
s1=pd.Series(val)
print("Sum of values present in series s1:",s1.sum())
```

OUTPUT

```
Sum of values present in series s1: 265
```

Methods of Series

prod functions

prod() To multiply all values present in a series

Example:

```
import pandas as pd
val=[10,30,27,68,85,45]
s1=pd.Series(val)
print("Product of values present in series s1:",s1.prod())
```

OUTPUT

```
Sum of values present in series s1: 2106810000
```

Methods of Series

mean functions

mean(): To find the mean of all values present in a series

Example:

```
import pandas as pd
val=[10,30,27,68,85,45]
s1=pd.Series(val)
print("Mean of values present in series s1:",s1.mean())
```

OUTPUT

```
Mean of values present in series s1: 44.166666666666664
```

Methods of Series

min functions

min(): To find the minimum of all values present in a series.

Example:

```
import pandas as pd
val=[10,30,27,68,85,45]
s1=pd.Series(val)
print("Minimum of values present in series s1:",s1.min())
```

OUTPUT

```
Minimum of values present in series s1: 10
```

Methods of Series

max functions

max(): To find the maximum of all values present in a series.

Example:

```
import pandas as pd
val=[10,30,27,68,85,45]
s1=pd.Series(val)
print("Maximum of values present in series s1:",s1.max())
```

OUTPUT

```
Maximum of values present in series s1: 85
```

Methods of Series

sort_values functions

sort_values(): To return the sorted series(ascending=True/False)

The argument ascending=True will arrange the values in ascending order and ascending =False will arrange the values in descending order. By default if nothing is mentioned the values are arranged in ascending order.

Example:

```
import pandas as pd
val=[10,30,27,68,85,45]
s1=pd.Series(val)
print("Values arranged in ascending order in series s1:",s1.sort_values())
```

OUTPUT

```
Values arranged in ascending order in series s1: 0    10
2    27
1    30
5    45
3    68
4    85
dtype: int64
```

Methods of Series

sort_values functions

sort_values(): To return the sorted series(ascending=True/False)

The argument ascending=True will arrange the values in ascending order and ascending =False will arrange the values in descending order. By default if nothing is mentioned the values are arranged in ascending order.

Example:

```
import pandas as pd
val=[10,30,27,68,85,45]
s1=pd.Series(val)
print("Values arranged in descending order in series s1:",s1.sort_values(ascending=False))
```

OUTPUT

```
Values arranged in descending order in series s1:
4    85
3    68
5    45
1    30
2    27
0    10
dtype: int64
```

Methods of Series

isnull functions

isnull() : checks for missing data on a Series object. It evaluates each object in the Series and provide a boolean value (True or False) indicating if the data is missing or not.

Example:

```
import pandas as pd
dict={"1":89,"3":75,"4":98}
s1=pd.Series(dict,index=["1","2","3","4"])
print(s1.isnull())
```

OUTPUT

```
1    False
2     True
3    False
4    False
dtype: bool
```

MATHEMATICAL OPERATIONS

We can perform binary operation on series like addition, subtraction etc. All these operations are done by index matching and missing values are filled in with NaN by default.

Operation	Operator	Function
Addition	+	add()
subtraction	-	sub()/subtract()
Multiplication	*	mul()/multiply()
Division	/	div()/divide()

ADDITION OPERATION

The add() function is used to perform addition operation on series. It can also be done using + operator.

Example:

```
import pandas as pd

ser1=pd.Series([20,30,36,38,45], index=['A','B','C','D','E'])

ser2=pd.Series([75,34,56,12,87], index=['A','F','D','G','E'])

data=ser1.add(ser2) # data=ser1+ser2

print(data)
```

OUTPUT

```
A     95.0
B      NaN
C      NaN
D     94.0
E    132.0
F      NaN
G      NaN
dtype: float64
```

ADDITION OPERATION

The parameter fill_value can be used to fill the specified value instead of NaN value for any elements in ser1 or ser2 that might be missing.

Example:

```
import pandas as pd

ser1=pd.Series([20,30,36,38,45], index=['A','B','C','D','E'])

ser2=pd.Series([75,34,56,12,87], index=['A','F','D','G','E'])

data=ser1.add(ser2, fill_value=0)

print(data)
```

OUTPUT

```
A     95.0
B     30.0
C     36.0
D     94.0
E    132.0
F     34.0
G     12.0
dtype: float64
```

Here, you can see now that as index ‘B’ in ser1 does not have corresponding matching index in ser2 thus the corresponding position in ser2 is filled with 0. Similarly, the fill_value parameter fills all missing indexes with value 0.

Rajesh Verma

SUBTRACTION OPERATION

The sub() function is used to perform subtraction operation on series. It can also be done using - operator

Example:

```
import pandas as pd

ser1=pd.Series([20,30,36,38,45], index=['A','B','C','D','E'])

ser2=pd.Series([75,34,56,12,87], index=['A','F','D','G','E'])

data=ser1.sub(ser2) # data=ser1-ser2

print(data)
```

OUTPUT

```
A   -55.0
B     NaN
C     NaN
D   -18.0
E   -42.0
F     NaN
G     NaN
dtype: float64
```

MULTIPLICATION OPERATION

The mul() function is used to perform multiplication operation on series. It can also be done using * operator.

Example:

```
import pandas as pd

ser1=pd.Series([20,30,36,38,45], index=['A','B','C','D','E'])

ser2=pd.Series([75,34,56,12,87], index=['A','F','D','G','E'])

data=ser2.mul(ser1, fill_value=100)

print(data)
```

OUTPUT

```
A    1500.0
B    3000.0
C    3600.0
D    2128.0
E    3915.0
F    3400.0
G    1200.0
dtype: float64
```

DIVISION OPERATION

The div() function is used to perform division operation on series. It can also be done using / operator.

Example:

```
import pandas as pd

ser1=pd.Series([20,30,36,38,45], index=['A','B','C','D','E'])

ser2=pd.Series([75,34,56,12,87], index=['A','F','D','G','E'])

data=ser2.div(ser1, fill_value=100)

print(data)
```

OUTPUT

```
A    3.750000
B    3.333333
C    2.777778
D    1.473684
E    1.933333
F    0.340000
G    0.120000
dtype: float64
```

Thank You