**INFORMATICS PRACTICES (NEW)**

**SUBJECT CODE-065**

**CH-02(PYTHAN PANDAS)**

**What is Pythan Pandas**

Pandas is an open source, BSD-licensed library providing high- performance, easy-to-use data structures and data analysis tools for the Python programming language.

Python with pandas is in use in a wide variety of academic and commercial domains, including Finance, Neuroscience, Economics, Statistics, Advertising, Web Analytics, and more.

**What problem does pandas solve?**

It enables us to carry out our entire data analysis workflow in Python. Combined with the excellent IPython toolkit and other libraries, the environment for doing data analysis in Python excels in performance, productivity, and the ability to collaborate.

**Some of the Highlights of Python pandas**

1. A fast and efficient DataFrame object for data manipulation with integratedindexing.
2. Tools for reading and writing data between in-memory data structures and different formats: CSV and text files, Microsoft Excel, SQL databases etc.
3. Flexible reshaping and pivoting of datasets

**Installing pandas**

The simplest way to install not only pandas, but Python and the most popular packages that is with [Anaconda](http://docs.continuum.io/anaconda/), a cross-platform (Linux, Mac OS X, Windows) Python distribution for data analytics and scientific computing. After running the installer, the user will have access to pandas and the rest of the stack without needing to install anything else, and without needing to wait for any software to be compiled.

Installation instructions for [Anaconda](https://www.anaconda.com/distribution/) [can be found here](http://docs.continuum.io/anaconda/install.html).

Another advantage to installing Anaconda is that you don’t need admin rights to install it. Anaconda can install in the user’s home directory, which makes it trivial to delete Anaconda if you decide (just delete that folder).

**Note:** Each time we need to use pandas in our python program we need to write a line of code at the top of the program:

import pandas as <identifier\_name>

Above statement will import the pandas library to our program. We will use two different pandas libraries in in our programs

1. Series
2. DataFrames

**pandas Series**

[Series](https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.Series.html#pandas.Series) is a one-dimensional labeled array capable of holding any data type (integers, strings, floating point numbers, Python objects, etc.). The axis labels are collectively referred to as the **index**. The basic method to create a Series is to call:

import pandas as <identifier name>

<Series\_name> = <identifier name>.Series(data, index=index) Data can be many different things:

* a Python dict
* a Python list
* a Python tuple

The passed index is a list of axis labels.

**Step by Step method to create a pandas Series**

Step 1

Suppose we have a list of games created with following python codes: games\_list = ['Cricket', 'Volleyball', 'Judo', 'Hockey']

Step 2

Now we create a pandas Series with above list

# Python script to generate a Series object from List import pandas as ps

games\_list = ['Cricket', 'Volleyball', 'Judo', 'Hockey'] s= ps.Series(games\_list)

print(s)

**OUTPUT**

1. Cricket
2. Volleyball 2 Judo

3 Hockey dtype: object

In the above output 0,1,2,3 are the indexes of list values. We can also create our own index for each value. Let us create another series with the same values with our own index values:

# Python script to generate a Series object from List using custom Index import pandas as pd

games\_list = ['Cricket', 'Volleyball', 'Judo', 'Hockey'] s= pd.Series(games\_list, index =['G1','G2','G3','G4']) print(s)

**OUTPUT**

G1 CRICKET

G2 VOLLEYBALL G3 JUDO

G4 HOCKEY

dtype: object

In the above output Game\_1, Game\_2, Game\_3, Game\_4 are our own created indexes of list values.

In the similar manner we can create pandas Series with different data (tuple, dictionary, Object) etc.

**Now we will create a Series with a Dictionary**

Suppose we have a dictionary of games created with the following Python codes:

d = {'Cricket': 1, 'Volleyball': 2, 'Judo': 3 , ‘Hockey’:4} Now we create a pandas Series with above dictionary # Python script to generate a Dictionary Object import pandas as pd

games\_dict = {'Cricket': 1, 'Volleyball': 2, 'Judo': 3 , 'Hockey':4} s= pd.Series(games\_dict)

print(s)

**OUTPUT**

Cricket 1 Volleyball 2 Judo 3 Hockey 4 Dtype : int64

**The Python Pandas DataFrame**

DataFrame is a Two-dimensional size-mutable, potentially heterogeneous tabular data structure. Tabular data structure has rows and columns. DataFrame is a way to represent and work with tabular data.

Pandas DataFrame is similar to excel sheet and looks like this

**How to create a Pandas DataFrame?**

In the real world, a Panda DataFrame will be created by loading the datasets from the permanent storage, including but not limited to excel, csv and MySQL database.

First we will use Python Data Structures (Dictionary and list) to create DataFrame.

Using Python Dictionary to create a DataFrame object name\_dict = { 'name' : ["Anita", "Sajal", "Ayaan", "Abhey"], 'age' : [14,32, 3, 6] }

If we print this dictionary using print(name\_dict) command, it will show us the output like this:

{'name': ['Anita', 'Sajal', 'Ayaan', 'Abhey'], 'age': [14, 32, 3, 6]}

We can create a Pandas DataFrame out of this dictionary

# Python script to generate a Dictionary Object and print using variable import pandas as pd

name\_dict = {

 'Name' : ["Anita", "Sajal", "Ayaan", "Abhey"],

 'Age' : [14,32, 4, 6]

 }

df = pd.DataFrame(name\_dict) print(df)

**Output**

Name Age

1. Anita 14 1 Sajal 15 2 Ayaan 4 3 Abhey 6

As you can see the output generated for the DataFrame object is look similar to what we have seen in the excel sheet as. Only difference is that the default index value for the first row is 0 in DataFrame whereas in excel sheet this value is 1. We can also customize this index value as per our need.

**Note:** A side effect of dictionary is that when accessing the same dictionary at two separate times, the order in which the information is returned by the does not remained constant.

One more example of DataFrame with customize index value

# Python script to generate a Dictionary Object with custom index import pandas as pd

name\_dict = {

 'Name' : ["Anita", "Sajal", "Ayaan", "Abhey"],

'Age' : [14,32, 4, 6] }

df = pd.DataFrame(name\_dict , index=[1,2,3,4]) print(df)

**Output**

Name Age

1. Anita 14 2 Sajal 15 3 Ayaan 4 4 Abhey 6

In the preceding output the index values start from 1 instead of 0

**Viewing the Data of a DataFrame**

To selectively view the rows, we can use head(…) and tail(…) functions, which by default give first or last five rows (if no input is provided), otherwise shows specific number of rows from top or bottom

Here is how it displays the contents df.head() # Displays first Five Rows df.tails() # Displays last Five Rows print(df.head(2)) # Displays first Two Rows print(df.tail(1)) #Displays last One Row

print(df.head(-2)) #Displays all rows except last two rows print(df.tail(-1)) #Displays all rows except first row **Advance operations on Data Frames:**

**Pivoting:**

Sample Pivot chart created in Excel

A Pivot Table is an interactive way to quickly summarize large amounts of data. We can use a Pivot Table to analyse numerical data in detail, and answer unanticipated questions about our data. A PivotTable is especially designed for:

* 1. Querying large amounts of data in many user-friendly ways.
	2. Expanding and collapsing levels of data to focus your results.
	3. Filtering, sorting, grouping, and conditionally formatting the most useful and interesting subset of data enabling you to focus on just the information you want.

**Creating Pivoting Tables with pandas’ DataFrame**

**Pivot Tables in pandas**

With pandas’ pivot tables we can create a spreadsheet-style pivot table using DataFrame.

Steps to create a pandas’ pivot table Step 1

Create a DataFrame using Dictionary or any other sequence Step 2

Use previously created DataFrame to generate a Pivot Table Step 3

Print the Pivot Table

**Example 1:**

# Pyhton script demonstrating the use of pivot\_table() method import pandas as pd

name\_dict = {

 'INVIGILATOR' : ["Rajesh", "Naveen","Anil","Naveen","Rajesh"],

 'AMOUNT' : [550,550,550,550,550],

 }

df = pd.DataFrame(name\_dict ) print(df)

pd.pivot\_table(df, index = ['INVIGILATOR'],aggfunc=’sum’)

**Output**

 INVIGILATOR AMOUNT

0 Rajesh 550

|  |  |  |
| --- | --- | --- |
| 1 | Naveen | 550 |
|  |  |  |
| 2 | Anil | 550 |
|  |  |  |
| 3 | Naveen | 550 |
|  |  |  |
| 4 | Rajesh | 550 |

**Output in pivot table form**

INVIGILATOR AMOUNT

|  |  |
| --- | --- |
| Anil | 550 |
|  |  |
| Naveen | 1100 |
|  |  |
| Rajesh | 1100 |

**Example 2:**

# Pyhton script demonstrating the use of pivot\_table() method import pandas as pd

sale\_dict = {

 'ITEM\_NAME' : ["NOTEBOOK", "PEN","INKPEN","NOTEBOOK","PEN"],

 'AMOUNT' : [100,50,30,100,50],

 'QUANTITY' :[2,5,3,3,5]

 }

df = pd.DataFrame(sale\_dict ) print(df)

pd.pivot\_table(df, index = ['ITEM\_NAME','AMOUNT','QUANTITY'],

aggfunc='sum')

**Output:**

ITEM\_NAME AMOUNT QUANTITY

0 NOTEBOOK 100 2

|  |  |  |  |
| --- | --- | --- | --- |
| 1 | PEN | 50 | 5 |
|  |  |  |  |
| 2 | INKPEN | 30 | 3 |

3 NOTEBOOK 100 3

4 PEN 50 5

**Output in pivot table form**

ITEM\_NAME AMOUNT QUANTITY

|  |  |  |
| --- | --- | --- |
| INKPEN | 30 | 3 |
|  |  |  |
| NOTEBOOK | 100 | 2 |

 3

PEN 50 5

**Descriptive Statistics**

After data collection, we generally use different ways to summarise the data. Python pandas provide different methods to generate descriptive statistics. Some of the common methods are:

min, max, mode, mean, count, sum, median Example 1:

#Total sales per employee import pandas as pd

monthlysale = { 'Salesman' : ["Akshit", "Jaswant","Karan","Akshit",

"Jaswant","Karan","Akshit", "Jaswant","Karan","Akshit",

"Jaswant","Karan"],

 'Sales' : [1000,300,800,1000,500,60,1000,900,300,1000,900,50],

 'Quarter' :[1,1,1,2,2,2,3,3,3,4,4,4],

'District':

['Kangra','Hamirpur','Kangra','Mandi','Hamirpur','Kangra','Kangra','Hami rpur','Mandi','Hamirpur','Hamirpur','Kangra']

 }

df = pd.DataFrame(monthlysale ) # Employee wise total sale:

pd.pivot\_table(df, index = ['Salesman'], values = ['Sales'],aggfunc='sum')

**Output:**

Salesman Sales

|  |  |
| --- | --- |
| Akshit | 4000 |
|  |  |
| Jaswant | 2600 |
|  |  |
| Karan | 1210 |

Example 2:

#Total sales Per District import pandas as pd

monthlysale = { 'Salesman' : ["Akshit", "Jaswant","Karan","Akshit",

"Jaswant","Karan","Akshit", "Jaswant","Karan","Akshit",

"Jaswant","Karan"],

 'Sales' : [1000,300,800,1000,500,60,1000,900,300,1000,900,50],

 'Quarter' :[1,1,1,2,2,2,3,3,3,4,4,4],

'District':

['Kangra','Hamirpur','Kangra','Mandi','Hamirpur','Kangra','Kangra','Hami rpur','Mandi','Hamirpur','Hamirpur','Kangra']

 }

df = pd.DataFrame(monthlysale ) # District wise total sale:

pd.pivot\_table(df, index = ['District'], values = ['Sales'],aggfunc='sum')

**Output:**

District Sales

Hamirpur 3600

Kangra 2910

Mandi 1300

Example 3:

#Total sales per employee and per district

import pandas as pd

monthlysale = { 'Salesman' : ["Akshit", "Jaswant","Karan","Akshit",

"Jaswant","Karan","Akshit", "Jaswant","Karan","Akshit",

"Jaswant","Karan"],

 'Sales' : [1000,300,800,1000,500,60,1000,900,300,1000,900,50],

 'Quarter' :[1,1,1,2,2,2,3,3,3,4,4,4],

'District':

['Kangra','Hamirpur','Kangra','Mandi','Hamirpur','Kangra','Kangra','Hami rpur','Mandi','Hamirpur','Hamirpur','Kangra']

 }

df = pd.DataFrame(monthlysale )

# Employee and district wise total sale:

pd.pivot\_table(df, index = ['Salesman','District'], values =

['Sales'],aggfunc='sum')

**Output:**

Salesman District Sales Akshit Hamirpur 1000 Kangra 2000

Mandi 1000

Jaswant Hamirpur 2600

Karan Kangra 910

Mandi 300