Data Science using Python

Dr. Kamini Solanki (Associate Professor)
Parul Institute of Computer Application - BCA

Parul®
University

DIGITAL LEARNING CONTENT

CHAPTER-5

Data Manipulation with Pandas

Before starting Data Science in python lets brief about pandas library

- It is open source library which is use to manipulate numerical and time series data.
- The name Pandas is derived from the word Panel Data – an Econometrics from Multidimensional data.
- It is built on the Numpy package and its key data structure is called the DataFrame.
- DataFrame allow you to store and manipulated data in tabular format.
- Where rows consider as numbers of observation and column consider as variable.
- Pandas officially stands for **'Python Data Analysis Library'**
- **Pandas is officially supported by Python 3.6.1 and above, 3.7, and 3.8.**

Installation of Pandas

- IF you using Anacoda or Jupyter than there is no need to install pandas, it available by defualt in this IDLE.
- But if you using law version of python than have to install is explicitly by using PIP command like,

```
pip install pandas
```

Note : at the time installation internet connection is required.

- after installing the pandas have to import it in our program than and than one can use it else not so, to import it use below line of code.

```
import pandas as pd
```

Note : here “pd” is nothing but allies name so that now we use pandas as pd through out the program.

Continue about Pandas

- We can say pandas id Excel of python. Display data in the tabular format while we performing any data manipulation operation on data using library pandas.
- Pandas read data like csv file, Excel, sql or even web pages data.

Why pandas for data science?

- It is easy to read and learn
- It is extremely fast and powerful
- It integrates well with other visualization libraries
- It is black, white and Asian at the same time

Data Structure of Pandas

Pandas generally provide two types of data structure i.e.

- Series
- Data Frame

What is series?

- Series in pandas is one dimensional array with label which allow to store any type of data. (integer, float, any python object etc).
- Series is nothing but column in excel sheet.
- Labels need not be unique but must be a hashable type.

What is DataFrame?

- DataFrame is two dimensional heterogeneous tabular data structure which have labeled rows and columns.
- Pandas DataFrame consists of three principal components, the data, rows, and columns.

How to create DataFrame and Series?

Creating Series using list

```
#creating series using list
#importing pandas
import pandas as pd
#list declaration
mylist = ['jonny','Frai','Stifen',1001,1234,4532]
#creating series
series1 = pd.Series(mylist)
print(series1)
```

O/P

```
0      jonny
1       Frai
2     Stifen
3       1001
4       1234
5       4532
dtype: object
```

Another way to create Series using list, see it on next slide.

```
'''
Series can be created with different data types such as
interger, string, Python Object, Floating Point
Series can be used data inputs of 4 types - ndarray, dict, scalar, list'''
# Create a series from a list
first_series = pd.Series(list('abcde'))
first_series
# indexing (data alignment) is created automatically
# datatype of series is Object
```

O/P

```
0    a
1    b
2    c
3    d
4    e
dtype: object
```

Using ndarray

Creating Series using ndarray (numpy array)

```
#create series using ndarray
# for that have to import numpy package and pandas
import pandas as pd
import numpy as np
state = np.array(["Gujarat","Rajsthat","J&K","Maharashtra"])
#here state is ndarray, so convert series from ndarray
series2=pd.Series(state)
print(series2)
```

```
0        Gujarat
1       Rajsthat
2            J&K
3    Maharashtra
dtype: object
```

O/P

- Importing numpy package is mandatory to create series using ndarray.
- Here 0,1,2,3 is index value

Continue

```
#you can also pass you own index value
#while create series using ndarray or list
#import pandas and numpy is necessory
state=np.array(["Gujarat","Rajshthan","J&k","Maharashtra"])
#create series by passing index value using index attribute
series2=pd.Series(state,index=[10,20,30,40])
print(series2)
```

```
10        Gujarat
20      Rajshthan
30            J&k
40    Maharashtra
dtype: object
```

O/P

- In this code you can see index value is change by using attribute index of series function.
- In output index value is change with given value.

You can also give index value like,
series2=pd.Series(state,index=['Gj','Rj','Jk','Mh'])

Using Dictionary

Creating Series using Dictionary

```
#creating series using dictionary
mydict={1001:["donald",30,"USA"],1002:["ronald",28,"US"],1003:["Maryam",20,"India"]}
#print(mydict)
series3=pd.Series(mydict)
print(series3)
```

```
1001      [donald, 30, USA]
1002       [ronald, 28, US]
1003    [Maryam, 20, India]
dtype: object
```

O/P

In dictionary key value taken as index value of series and here you can see value of it is assign using list.

Creating series using scalar

- By providing scalar value one can also create series using pandas.
- Scalar value mean same value or data is repeated up to the given length of series.
- In below example 10 is repeated up to the given length and according to that index value is given.

```
# Create a series with scalar input
# here datatype is integer
#as first argunent is 10 which is of integer type
scalar_series = pd.Series(10,index=['a','b','c','d','e'])
scalar_series
```

```
a    10
b    10
c    10
d    10
e    10
dtype: int64
```

Operations on series

Accessing element from series

- For accessing element from the series we have to pass index value.
- data can be assessed through different functions like loc, iloc by passing data element position or index range.

```
dict_country_gdp[0]
```

```
15
```

```
# Acess first 3 countries from the series
dict_country_gdp[0:4]
```

```
India     15
USA       40
Canada    30
China     20
dtype: int64
```

O/P

Here in this example accessing data by using slice operation. And also by passing index value for specific value.

Continue

Accessing last three element from the series and by passing label

- For accessing element right hand side (start from last) have to pass negative index value.
- One can also pass negative index using slice operation. Even one can also fetch value by passing label.

```
#accessing element at the end (last)
#pass -ve index value
print(series2[-3:])
#featching value by passing label
#here 'jk' is lable value
#like this one can pass more lable also
print(series2['Jk'])
```

```
Rj      Rajshthan
Jk            J&k
Mh    Maharashtra
dtype: object
J&k
```

Continue

ioc and iloc command

- By using ioc and iloc command we can also access data for that have to pass lable.

```
# Look up a country by name or index
dict_country_gdp.loc['USA']
```

```
40
```

```
# look up by position
dict_country_gdp.iloc[1]
```

```
40
```

Here in this example ‘USA’ and 1 is index or label value.

Vectorized Operation on Series

```
# Vectorized operations in Series
first_vector_series = pd.Series([1,2,3,4],index=['a','b','c','d'])
second_vector_series = pd.Series([10,20,30,40],index=['a','b','c','d'])
first_vector_series
second_vector_series
```

```
a    10
b    20
c    30
d    40
dtype: int64
```

Vectorization is the process of executing operations on entire arrays.

Continue

addition using verctorize operation

- It perform addition based on the index value. In previous example you see that index value of both the vector is same.

```
# It performs addition based on index of both the series
first_vector_series+second_vector_series
```

```
a    11
b    22
c    33
d    44
dtype: int64
```

Continue

```
third_vector_series = pd.Series([10,20,30,40],index=['a','d','b','c'])

# it perfroms additon based on index value
first_vector_series+third_vector_series
```

```
a    11
b    32
c    43
d    24
dtype: int64
```

- In this example you can see here we declare third vector with same index value.
- Than perform addition operation for first and third vector so it will add value based on index.

Continue

```
fourth_vector_series = pd.Series([10,20,30,40],index=['a','b','e','f'])
```

```
first_vector_series+fourth_vector_series
```

```
a    11.0
b    22.0
c     NaN
d     NaN
e     NaN
f     NaN
dtype: float64
```

- Here in this example you can see we declare one more series with few different index value.
- And it will do addition those value only which is common between first and forth vector for other it gives output as NaN.

Series object attributes

Attributes	Description
Series.index	Defines the index of the Series.
Series.shape	It returns a tuple of shape of the data.
Series.dtype	It returns the data type of the data.
Series.size	It returns the size of the data.
Series.empty	It returns True if Series object is empty, otherwise returns false.
Series.hasnans	It returns True if there are any NaN values, otherwise returns false.
Series.nbytes	It returns the number of bytes in the data.
Series.ndim	It returns the number of dimensions in the data.
Series.itemsize	It returns the size of the datatype of item.

Pandas Dataframe

Dataframe

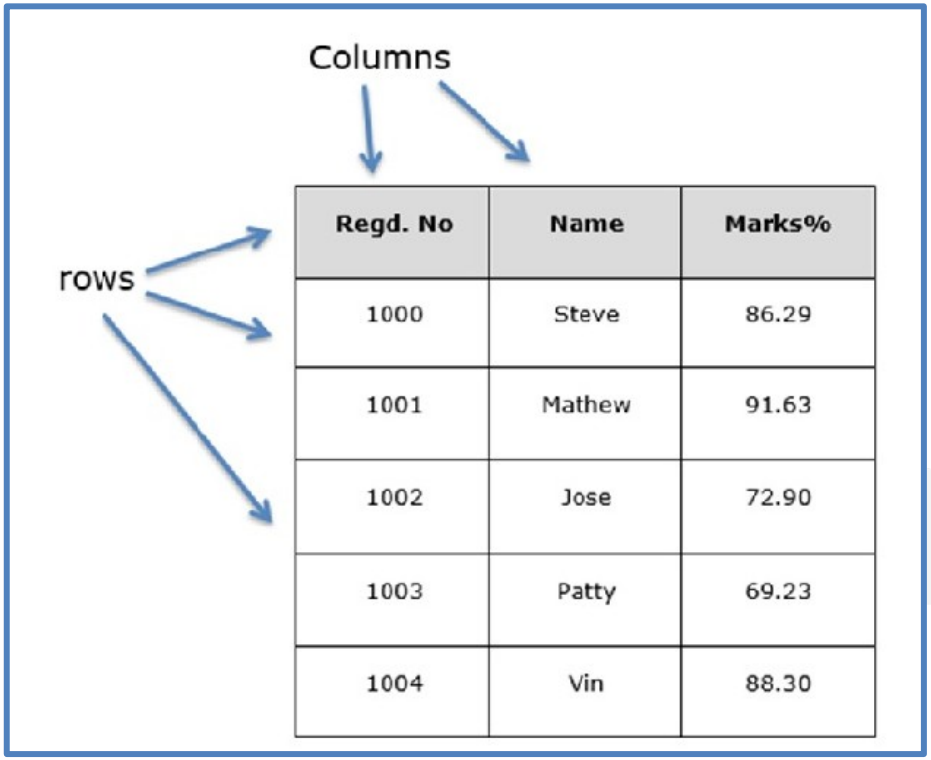
- Dataframe is combination of rows and column's.
- It a represent data in the form of two dimension we can say in tabular manner.
- DataFrame is defined as a standard way to store data that has two different indexes, i.e., row index and column index.
- Dataframe has labeled axis for row value and column value.

Features of Dataframe

- Potentially columns are of different types
- Size – Mutable
- Labeled axes (rows and columns)
- Can Perform Arithmetic operations on rows and columns

Example of dataframe.

here dataframe for student data



Regd. No	Name	Marks%
1000	Steve	86.29
1001	Mathew	91.63
1002	Jose	72.90
1003	Patty	69.23
1004	Vin	88.30

https://www.tutorialspoint.com/

Continue

Pandas dataframe is created using below constructor

pandas. Dataframe(data, index, column, dtype, copy)

Data	It may be any ndarray, dictionary, series, map or constant
Index	Default is np.arrange(n) if no index passed.
Column	Default is np.arrange(n). It show true if no column is passed.
Dtype	Refer data type for each column
Copy	Is use to copy the dataframe

We can create dataframe using dictionary, list, numpy array and series.

Creating dataframe

creating empty dataframe

- For create empty dataframe “DataFrame()” is use with no parameter like below,

```
#creating empty dataframe
import pandas as pd
df = pd.DataFrame()
print(df)
```

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

In this example you can see total numbers of rows and columns is empty. As it store data in tabular Format.

Continue...

creating dataframe from the list

```
#creating dataframe from list
import pandas as pd
country = ['india','USA','Belgioum', 'Oman','Nepal']
#calling dataframe constructor
df = pd.DataFrame(country)
print(df)
```

```
          0
0     india
1       USA
2  Belgioum
3      Oman
4     Nepal
```

- In above example you can see column number is indicate as 0 and raw numbers start with 0 to n.

Creating dataframe

creating dataframe using dictionary

```
# Create DataFrame from dict of equal length lists
# last five olympics data: place,year and number of countries participated
olympic_data_list = {'place':['london','Beijing','Athens','Sydeny','Atlanta'],
                     'year':[2012,2008,2004,2000,1996],
                     'No. of paricipating Countries':[205,204,201,200,197]}
df_olympic_data = pd.DataFrame(olympic_data_list)
df_olympic_data
```

	place	year	No. of paricipating Countries
0	london	2012	205
1	Beijing	2008	204
2	Athens	2004	201
3	Sydeny	2000	200
4	Atlanta	1996	197

Continue...

creating dataframe using dictionary (by passing another dictionary as value)

```
# Create dataframe from dict
#Note: here Name of country is first dictionary and year is 2nd dictionary
olympic_data_dict = {'London':{2012:205},'Beljium':{2008:204}}
df_olympic_data_dict = pd.DataFrame(olympic_data_dict)
df_olympic_data_dict
```

	London	Beljium
2012	205.0	NaN
2008	NaN	204.0

- In this example you can see here we pass two dictionary as value on one dictionary. It give output NaN if value is not present for that year.

Continue...

creating dataframe by passing index value

```
#creating dataframe using dictionary
#here we are passing index value as a,b,c
dict1={'1':'Priya','2':'Pankaj','3':'Ravi'}
df_dict1 = pd.DataFrame(dict1,index=['a','b','c'])
df_dict1
```

	1	2	3
a	Priya	Pankaj	Ravi
b	Priya	Pankaj	Ravi
c	Priya	Pankaj	Ravi

Here a,b,c is index value which is given by passing it with index

Parul®
University

DIGITAL LEARNING CONTENT

Creating dataframe

creating dataframe using dictionary series

```
import pandas as pd

info = {'one' : pd.Series([34,20,11,67,80,43], index=['a', 'b', 'c', 'd', 'e', 'f']),
   'two' : pd.Series([11, 21, 33, 45, 53, 67, 77, 81], index=['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h'])}

d1 = pd.DataFrame(info)
print (d1)
```

```
    one  two
a  34.0   11
b  20.0   21
c  11.0   33
d  67.0   45
e  80.0   53
f  43.0   67
g   NaN   77
h   NaN   81
```

Parul®
University

DIGITAL LEARNING CONTENT

Creating dataframe

creating dataframe using dictionary series

```
# Create DataFrame from dict of Series
olympic_series_participantion = pd.Series([205,204,201,200,197],index=[2012,2008,2004,2000,1996])
olympic_series_country = pd.Series(['London','Beijing','Athens','Sydney','Atlanta'],
                                   index=[2012,2008,2004,2000,1996])
df_olympic_series = pd.DataFrame({'No. of Participating Countries':olympic_series_participantion,
                                  'Host Cities':olympic_series_country})
df_olympic_series
```

	No. of Participating Countries	Host Cities
2012	205	London
2008	204	Beijing
2004	201	Athens
2000	200	Sydney
1996	197	Atlanta

Dataframe using ndarray

creating dataframe using numpy (ndarray) array

```
# Create DataFrame from ndarray
import numpy as np
np_array = np.array([2012,2008,2004,2006])
dict_ndarray = {'year':np_array}
df_ndarray = pd.DataFrame(dict_ndarray)
df_ndarray
```

	year
0	2012
1	2008
2	2004
3	2006

Creating dataframe

creating dataframe using another daraframe

```
# Create DataFrame from DataFrame
df_from_df = pd.DataFrame(df_olympic_series)
```

```
df_from_df
```

	Host Cities	No. of Participating Countries
2012	London	205
2008	Beijing	204
2004	Athens	201
2000	Sydney	200
1996	Atlanta	197

In this other dataframe is passed in constructor to create datafram.

Operation on dataframe

printing (selected) column

	No. of Participating Countries	Host Cities
2012	205	London
2008	204	Beijing
2004	201	Athens
2000	200	Sydney
1996	197	Atlanta

```
df_from_df['Host Cities']
```

```
2012      London
2008     Beijing
2004      Athens
2000      Sydney
1996     Atlanta
Name: Host Cities, dtype: object
```

In this example ‘Host Cities’ column name and year is index value

Continue...

Printing selected column

```
# importing the pandas library
import pandas as pd

info = {'one' : pd.Series([1, 2, 3, 4, 5, 6], index=['a', 'b', 'c', 'd', 'e', 'f']),
   'two' : pd.Series([1, 2, 3, 4, 5, 6, 7, 8], index=['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h'])}

d1 = pd.DataFrame(info)
print (d1 ['one'])
```

```
a    1.0
b    2.0
c    3.0
d    4.0
e    5.0
f    6.0
g    NaN
h    NaN
Name: one, dtype: float64
```

Continue...

column addition

```
info = {'one' : pd.Series([1, 2, 3, 4, 5], index=['a', 'b', 'c', 'd', 'e']),
   'two' : pd.Series([1, 2, 3, 4, 5, 6], index=['a', 'b', 'c', 'd', 'e', 'f'])}

df = pd.DataFrame(info)
# Add a new column to an existing DataFrame object
print ("Add new column by passing series")
df['three']=pd.Series([20,40,60],index=['a','b','c'])
print (df)
```

```
Add new column by passing series
   one  two  three
a  1.0    1   20.0
b  2.0    2   40.0
c  3.0    3   60.0
d  4.0    4    NaN
e  5.0    5    NaN
f  NaN    6    NaN
```

Here you can see third column is added with column name three

Continue...

add one more column by performing addition of two column

```
print ("Add new column using existing DataFrame columns")
df['four']=df['one']+df['three']

print (df)
```

```
Add new column using existing DataFrame columns
   one  two  three  four
a  1.0    1   20.0  21.0
b  2.0    2   40.0  42.0
c  3.0    3   60.0  63.0
d  4.0    4    NaN   NaN
e  5.0    5    NaN   NaN
f  NaN    6    NaN   NaN
```

Here you can see Forth column is added by performing addition of two column one and three

Continue...

read (access) data from top and bottom using head() and tail() method

```
#reading raws from head or from end
#for read data from top head() is use
#if not pass no's of raw than it read 5 raws by default
#same for tail() function
print(df_olympic_series.head(2))
print("reading data from end")
print(df_olympic_series.tail())
```

```
      No. of Participating Countries Host Cities
2012                             205      London
2008                             204     Beijing
reading data from end
      No. of Participating Countries Host Cities
2012                             205      London
2008                             204     Beijing
2004                             201      Athens
2000                             200      Sydney
1996                             197     Atlanta
```

Here you can see in head() method n=2 is pass so it read first 2 rows and in tail() nothing is pass so it read last 5 rows

Continue...

View only title of columns and index value of dataframe

```
# To view coulumns of the dataframe
df_olympic_series.columns
```

```
Index(['Host Cities', 'No. of Participating Countries'], dtype='object')
```

```
# To view indexes of dataset
df_olympic_series.index
```

```
Int64Index([2012, 2008, 2004, 2000, 1996], dtype='int64')
```

To view only columns df.columns is use and for index df.index is use.

Continue...

Delete column from dataframe

To delete column from dataframe del or pop command is use.

```
# using del function
print ("Delete the first column:")
del df['one']
print (df)
# using pop function
print ("Delete the another column:")
df.pop('two')
print (df)
```

O/P

```
Delete the first column:
   two  three  four
a    1   20.0  21.0
b    2   40.0  42.0
c    3   60.0  63.0
d    4    NaN   NaN
e    5    NaN   NaN
f    6    NaN   NaN
Delete the another column:
   three  four
a   20.0  21.0
b   40.0  42.0
c   60.0  63.0
d    NaN   NaN
e    NaN   NaN
f    NaN   NaN
```

Some inbuilt function for pandas dataframe

Method	Description
DataFrame.append()	Add the rows of other dataframe to the end of the given dataframe.
DataFrame.apply()	Allows the user to pass a function and apply it to every single value of the Pandas series.
DataFrame.assign()	Add new column into a dataframe.
DataFrame.astype()	Cast the Pandas object to a specified dtype.astype() function.
DataFrame.concat()	Perform concatenation operation along an axis in the DataFrame.
DataFrame.count()	Count the number of non-NA cells for each column or row.
For more please visit : https://www.javatpoint.com/python-pandas-dataframe	

Parul®
University

DIGITAL LEARNING CONTENT

Reading CSV and perform operation on CSV using pandas

- CSV stand as comma separated Value.
- It store data in tabular format like rows and column wise.
- It can be open in Excel also as well as in notepad or word but the data is separated by comma delimiter.
- To read CSV file using pandas there is no need to write bunch of code one can read by using code of few lines.
- To read CSV file first one have to load CSV file in pandas dataframe.
- In next slide you can see how CSV file look when it open in notepad and in an Excel.

Example of CSV file

Data opened in Microsoft Excel Spreadsheet

	A	B	C	D
1	Name	Age	Country	Goals
2	James Rodriguez	26	Colombia	6
3	Thomas Muller	28	Germany	5
4	Lionel Messi	31	Argentina	4
5	Neymar	26	Brazil	4
6	Robin van Persie	34	Netherlands	4

Column names are first row in text file

Data in CSV format opened in Text Editor

```
1  Name,Age,Country,Goals
2  James Rodriguez,26,Colombia,6
3  Thomas Muller,28,Germany,5
4  Lionel Messi,31,Argentina,4
5  Neymar,26,Brazil,4
6  Robin van Persie,34,Netherlands,4
7
```

Commas are used to separate columns

Load CSV file

load the CSV file using pandas

1) Import pandas library
2) Read CSV file using read_csv() command of pandas.
3) If file is located at the same folder than there is no need to give absolute path.

Continue...

```
#Reading CSV File
import pandas as pd
csv_df = pd.read_csv('Machine_readable_file.csv')
csv_df
```

	Series_reference	Period	Data_value	Suppressed	STATUS	UNITS	Magnitude	Subject	Group
0	ECTA.S19A1	2001.03	2462.5	NaN	F	Dollars	6	Electronic Card Transactions (ANZSIC06) - ECT	Total values - Electronic card transactions A/...
1	ECTA.S19A1	2002.03	17177.2	NaN	F	Dollars	6	Electronic Card Transactions (ANZSIC06) - ECT	Total values - Electronic card transactions A/...
2	ECTA.S19A1	2003.03	22530.5	NaN	F	Dollars	6	Electronic Card Transactions (ANZSIC06) - ECT	Total values - Electronic card transactions A/...

Store dataframe data into CSV

```
#storing data of dataframe in CSV
df.to_csv("new_csv")
```

```
df2=pd.read_csv('new_csv')
```

```
df2
```

	Unnamed: 0	one	two	three
0	a	1.0	1	20.0
1	b	2.0	2	40.0
2	c	3.0	3	60.0
3	d	4.0	4	NaN
4	e	5.0	5	NaN
5	f	NaN	6	NaN

To convert data of dataframe into CSV format to_csv() is use.
You can also see index value is printed two times while it created and while it's loaded.
In next slide using index=false one can stop passing index value while coverted dataframe data into CSV.

Continue...

```
df.to_csv("new_csv", index=False)
df2=pd.read_csv('new_csv')
df2
```

	one	two	three
0	1.0	1	20.0
1	2.0	2	40.0
2	3.0	3	60.0
3	4.0	4	NaN
4	5.0	5	NaN
5	NaN	6	NaN

using index=false one can stop passing index value while coverted dataframe data into CSV.

in this example you can see index value a, b, c... is skipped.

Customize column name

```
#Reading CSV File with cutomized column name
#column name passd using list as a value of names
import pandas as pd
csv_df = pd.read_csv('Machine_readable_file.csv', names=['a1','a2','a3','a4','a5','a6','a7',
                                                         'a8','a9','a10','a11','a12','a13','a14'])
csv_df
```

	a1	a2	a3	a4	a5	a6	a7	a8	a9	a10
0	Series_reference	Period	Data_value	Suppressed	STATUS	UNITS	Magnitude	Subject	Group	Series_title_1
1	ECTA.S19A1	2001.03	2462.5	NaN	F	Dollars	6	Electronic Card Transactions (ANZSIC06) - ECT	Total values - Electronic card transactions A/...	Actual
2	ECTA.S19A1	2002.03	17177.2	NaN	F	Dollars	6	Electronic Card Transactions (ANZSIC06) - ECT	Total values - Electronic card transactions A/...	Actual
3	ECTA.S19A1	2003.03	22530.5	NaN	F	Dollars	6	Electronic Card Transactions (ANZSIC06) - ECT	Total values - Electronic card transactions A/...	Actual

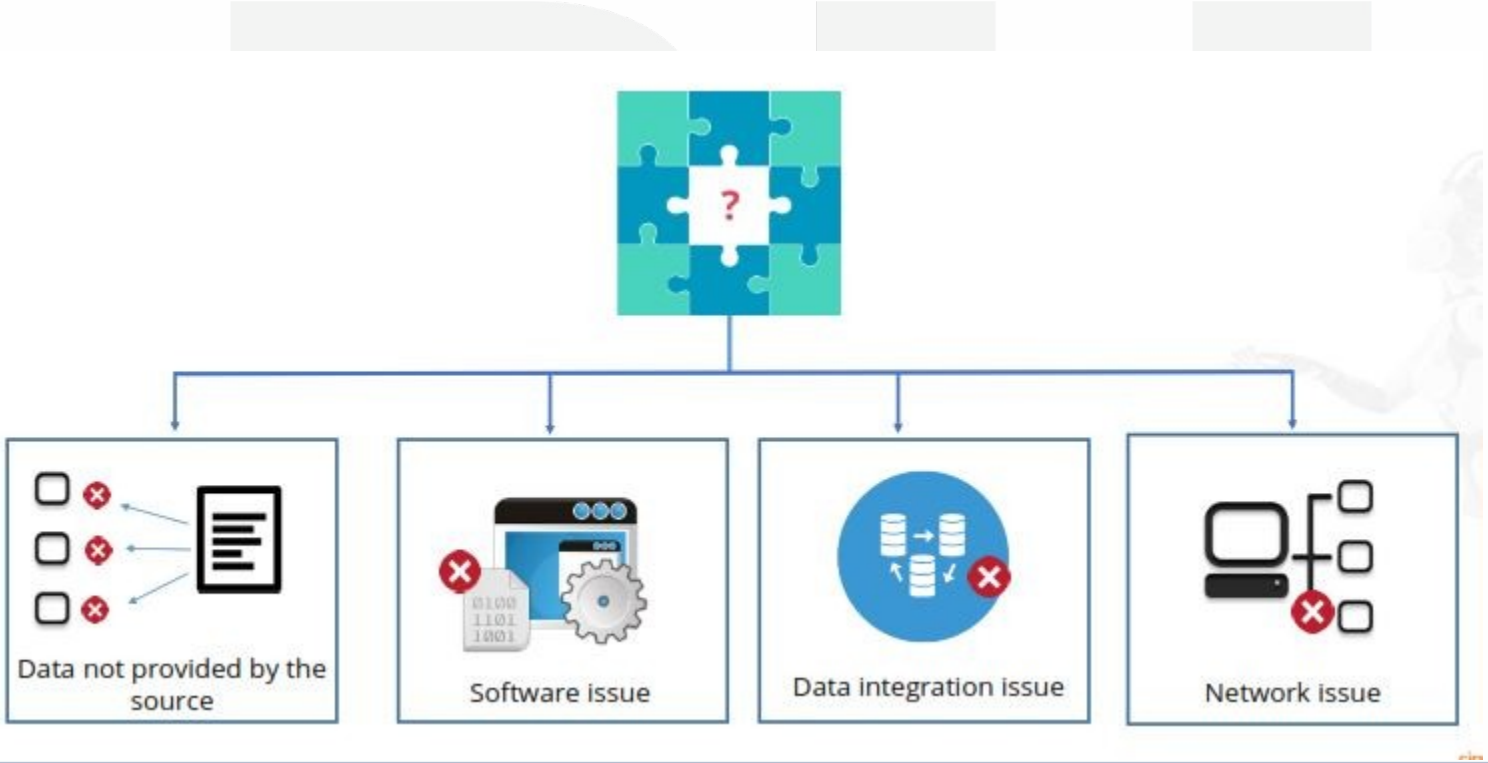
Parul® University

Installation of Pandas

- https://www.geeksforgeeks.org/introduction-to-pandas-in-python/#getting
- https://www.geeksforgeeks.org/pandas-tutorial
- https://www.tutorialspoint.com/python_pandas/python_pandas_introduction_to_data_structures.htm
- https://www.learndatasci.com/tutorials/python-pandas-tutorial-complete-introduction-for-beginners/
- https://pandas.pydata.org/pandas-docs/stable/getting_started/tutorials.html
- https://www.learnpython.org/en/Pandas_Basics
- https://pythonprogramming.net/data-analysis-python-pandas-tutorial-introduction/

Missing Values

- Various factors may lead to missing data values:



Handling Missing Values

- It’s difficult to operate a dataset when it has missing values or uncommon indices.

```
In [3]: import pandas as pd

In [4]: #declare first series
        first_series = pd.Series([1,2,3,4,5],index=['a','b','c','d','e'])

In [5]: #declare second series
        second_series=pd.Series([10,20,30,40,50],index=['c','e','f','g','h'])

In [6]: sum_of_series = first_series+second_series

In [7]: sum_of_series

Out[7]: a    NaN
        b    NaN
        c     13
        d    NaN
        e     25
        f    NaN
        g    NaN
        h    NaN
        dtype: float64
```

Handling Missing Values with Functions

- The dropna function drops all the values with uncommon indices.

```
In [5]: sum_of_series
Out[5]: a     NaN
        b     NaN
        c    13.0
        d     NaN
        e    25.0
        f     NaN
        g     NaN
        h     NaN
        dtype: float64

In [6]: # drop NaN( Not a Number) values from dataset
        dropna_s = sum_of_series.dropna()

In [7]: dropna_s
Out[7]: c    13.0
        e    25.0
        dtype: float64
```

Handling Missing Values with Functions

- The fillna function fills all the uncommon indices with a number instead of dropping them.

```
In [8]: dropna_s.fillna(0)   <-- Fill the missing values with zero
Out[8]: c    13.0
        e    25.0
        dtype: float64

In [9]: # Fill NaN( Not a Number) values with Zeroes (0)
        fillna_s = sum_of_series.fillna(0)

In [10]: fillna_s
Out[10]: a     0.0
         b     0.0
         c    13.0
         d     0.0
         e    25.0
         f     0.0
         g     0.0
         h     0.0
         dtype: float64
```

Handling Missing Values with Functions: Example

```
In [10]: #fill values with zeroes before performing addition operation for missing indices
         fill_NaN_with_zeros_before_sum =first_series.add(second_series,fill_value=0)

In [11]: fill_NaN_with_zeros_before_sum

Out[11]: a     1
         b     2
         c    13
         d     4
         e    25
         f    30
         g    40
         h    50
         dtype: float64
```

Data Operation

- Data operation can be performed through various built-in methods for faster data processing.

```
In [1]: import pandas as pd

In [2]: #declare movie rating dataframe: ratings from 1 to 5 (star * rating)
        df_movie_rating = pd.DataFrame(
                          {'movie 1': [5,4,3,3,2,1],
                           'movie 2': [4,5,2,3,4,2]},
                          index=['Tom','Jeff','Peter','Ram','Ted','Paul']
        )

In [3]: df_movie_rating
```

Out[3]:

	movie 1	movie 2
Tom	5	4
Jeff	4	5
Peter	3	2
Ram	3	3
Ted	2	4
Paul	1	2

Data Operation with Functions

- While performing data operation, custom functions can be applied using the applymap method.

```
In [4]: def movie_grade(rating):
            if rating==5:
                return 'A'
            if rating==4:
                return 'B'
            if rating==3:
                return 'C'
            else:
                return 'F'
```

← Declare a custom function

```
In [5]: print (movie_grade(5))
A
```

← Test the function

```
In [6]: df_movie_rating.applymap(movie_grade)
```

← Apply the function to the DataFrame

Out[6]:

	movie 1	movie 2
Tom	A	B
Jeff	B	A
Peter	C	F
Ram	C	C
Ted	F	B
Paul	F	F

Data Operation with Statistical Functions

```
In [7]: df_test_scores = pd.DataFrame(
                    {'Test1': [95,84,73,88,82,61],
                    'Test2': [74,85,82,73,77,79]},
                    index=['Jack','Lewis','Patrick','Rich','Kelly','Paula']
        )
```

Create a DataFrame with two test

```
In [8]: df_test_scores.max()
```

Apply the max function to find the maximum score

```
Out[8]: Test1    95
        Test2    85
        dtype: int64
```

```
In [9]: df_test_scores.mean()
```

Apply the mean function to find the average score

```
Out[9]: Test1    80.500000
        Test2    78.333333
        dtype: float64
```

```
In [10]: df_test_scores.std()
```

Apply the std function to find the standard deviation for both the tests

```
Out[10]: Test1    11.979149
         Test2     4.633213
         dtype: float64
```

Data Operation Using Groupby

```
In [16]: df_president_name = pd.DataFrame({'first':['George','Bill', 'Ronald','Jimmy','George'],
                                           'last':['Bush','Clinton', 'Regan', 'Carter', 'Washington']})
```

Create a DataFrame with first and last name as former presidents

```
In [17]: df_president_name
```

Out[17]:

	first	last
0	George	Bush
1	Bill	Clinton
2	Ronald	Regan
3	Jimmy	Carter
4	George	Washington

```
In [18]: grouped = df_president_name.groupby('first')
```

Group the DataFrame with the first name

```
In [19]: grp_data = grouped.get_group('George')
         grp_data
```

Group the DataFrame with the first name

Out[19]:

	first	last
0	George	Bush
4	George	Washington

Data Operation Using Sorting

```
In [20]: df_president_name.sort_values('first')
```

Sort values by first name

Out[20]:

	first	last
1	Bill	Clinton
0	George	Bush
4	George	Washington
3	Jimmy	Carter
2	Ronald	Regan

Data Standardization

```
In [11]: def standardize_tests(test):
             return (test-test.mean())/ test.std()
```

Create a function to return the standardize value

```
In [12]: standardize_tests(df_test_scores['Test1'])

Out[12]: Jack       1.210437
         Lewis      0.292174
         Patrick   -0.626088
         Rich       0.626088
         Kelly      0.125218
         Paula     -1.627829
         Name: Test1, dtype: float64
```

```
In [13]: def standardize_test_scores(datafrm):
             return datafrm.apply(standardize_tests)
```

Apply the function to the entire dataset

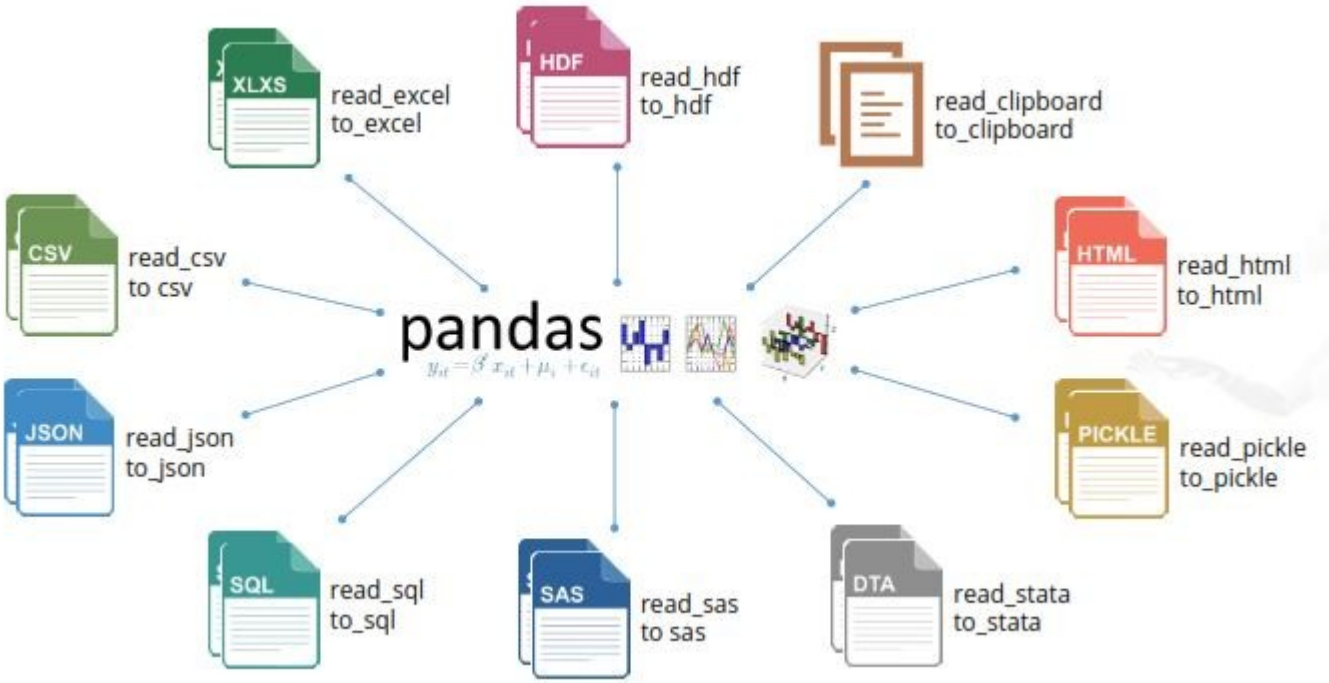
```
In [14]: standardize_test_scores(df_test_scores)
```

Out[14]:

	Test1	Test2
Jack	1.210437	-0.935276
Lewis	0.292174	1.438886
Patrick	-0.626088	0.791387
Rich	0.626088	-1.151109
Kelly	0.125218	-0.287777
Paula	-1.627829	0.143889

Standardized test data is applied for the entire DataFrame

File Read and Write Support



Parul® University

DIGITAL LEARNING CONTENT

Activity: Sequence it Right!

- The code here is buggy. You have to correct its sequence to debug it. To do that, click any two code snippets, which you feel are out of place, to swap their places.

1
```
df_movie_rating = pd.DataFrame(
                  {'movie 1': [5,4,3,3,2,1],
                  'movie 2': [4,5,2,3,4,2]},
                  index=['Tom','Jeff','Peter','Ram','Ted','Paul']
```

2
```
print (movie_grade(5))
```
```
A
```

3
```
def movie_grade(rating):
    if rating==5:
        return 'A'
    if rating==4:
        return 'B'
    if rating==3:
        return 'C'
    else:
        return 'F'
```

4
```
df_movie_rating.applymap(movie_grade)
```

Click any two code snippets to swap them

Pandas SQL Operations

In [1]:
```
#import pandas library
import pandas as pd
```

In [2]:
```
#import sqllite
import sqlite3
```

In [3]:
```
#Create SQL table
create_table = """
CREATE TABLE student_score
(Id INTEGER, Name VARCHAR(20), Math REAL,
Science REAL
);"""
```

In [4]:
```
#execute the SQL statement
executeSQL = sqlite3.connect(':memory:')
executeSQL.execute(create_table)
executeSQL.commit()
```

In [5]:
```
#prepare a SQL query
SQL_query = executeSQL.execute('select * from student_score')
```

In [7]:
```
#fetch result from the SQLlite database
resulset = SQL_query.fetchall()
```

In [8]:
```
#view result (empty data)
resulset
```

Out[8]:
```
[]
```

Parul®
University

DIGITAL LEARNING CONTENT

Pandas SQL Operations

In [9]:
```
#prepare records to be inserted into SQL table through SQL statement
insertSQL = [(10,'Jack',85,92),
                        (29,'Tom',73,89),
                         (65,'Ram',65.5,77),
                         (5,'Steve',55,91)
                        ]
```

In [10]:
```
#insert records into SQL table through SQL statement
insert_statement = "Insert into student_score values(?,?,?,?)"
executeSQL.executemany(insert_statement,insertSQL)
executeSQL.commit()
```

In [11]:
```
#prepare SQL query
SQL_query = executeSQL.execute("select * from student_score")
```

In [12]:
```
#fetch the resultset for the query
resulset = SQL_query.fetchall()
```

In [13]:
```
#view the resultset
resulset
```

Out[13]:
```
[(10, u'Jack', 85.0, 92.0),
 (29, u'Tom', 73.0, 89.0),
 (65, u'Ram', 65.5, 77.0),
 (5, u'Steve', 55.0, 91.0)]
```

Parul®
University

DIGITAL LEARNING CONTENT

Pandas SQL Operations

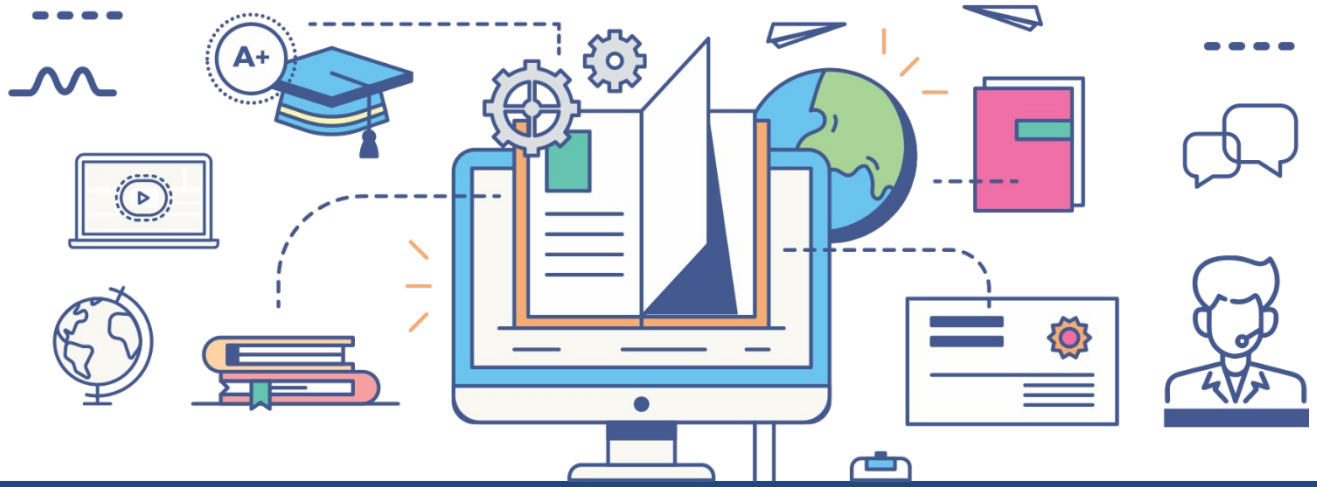
```
In [14]: #put the records together in dataframe
         df_student_recors = pd.DataFrame(resulset,columns=zip(*SQL_query.description)[0])

In [15]: #view the records in pandas dataframe
         df_student_recors
```

Out[15]:

	Id	Name	Math	Science
0	10	Jack	85.0	92.0
1	29	Tom	73.0	89.0
2	65	Ram	65.5	77.0
3	5	Steve	55.0	91.0

DIGITAL LEARNING CONTENT



Parul® University

www.paruluniversity.ac.in