20 DATA HANDLING USING PANDAS (NCERT CLASS 12)

PYTHON LIBRARY – PANDAS

It is a most famous Python package for data science, which offers powerful and flexible data structures that make data analysis and manipulation easy.Pandas makes data importing and data analyzing much easier. Pandas builds on packages like NumPy and matplotlib to give us a single & convenient place for data analysis and visualization work.

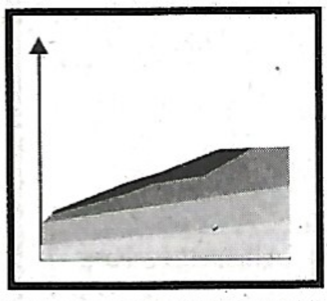
Python Library – Matplotlib

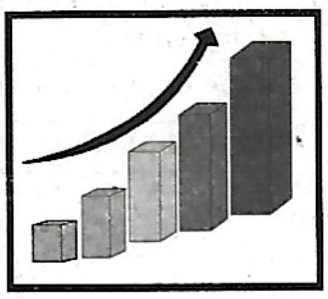
Matplotlib is a comprehensive library for creating static, animated, and interactive visualizations in Python.It is used to:-

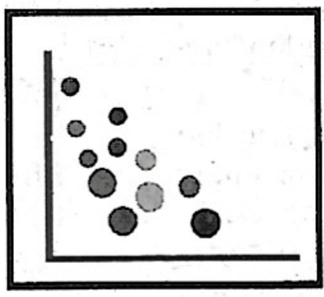
(I) Develop publication quality plots with just a few lines of code.

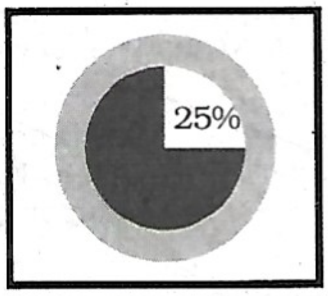
(II) Use interactive figures that can zoom, pan, update.

We can customize and Take full control of line styles, font properties, axes properties as well as export and embed to a number of file formats and interactive environments.

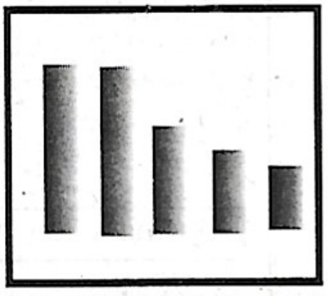

Area Plot


Histogram


Scatter Plot



Pie Plot


Bar Graph

Basic Features of Pandas :

- It help a lot in keeping track of our data.
- With a pandas dataframe, we can have different data types (float, int, string, datetime, etc) all in one place.
- Pandas has built in functionality for like easy grouping & easy joins of data, rolling windows.
- Good IO capabilities; Easily pull data from a MySQL database directly into a data frame.
- With pandas, you can use patsy for R-style syntax in doing regressions.
- Tools for loading data into in-memory data objects from different file formats.
- Data alignment and integrated handling of missing data.
- Reshaping and pivoting of data sets.
- Label-based slicing, indexing and subsetting of large data sets.

Pandas – Installation/Environment Setup

Pandas module doesn't come bundled with Standard Python. If we install Anaconda Python package Pandas will be installed by default.

Steps for Anaconda installation & Use

(I) visit the site https://www.anaconda.com/download/

(II) Download appropriate anaconda installer.

(III) After download install it.

(IV) During installation check for set path and all user.

(V) After installation start spyder utility of anaconda from start menu.

(VI) Type import pandas as pd in left pane(temp.py) .

(VII) Then run it.

(VIII) If no error is show then it shows pandas is installed.

(IX) Like default temp.py we can create another .py file from new window option of file menu for new program.

Pandas installation can be done in Standard Python distribution,using following steps :

(I) There must be service pack installed on our computer if we are using windows.If it is not installed then we will not be able to install pandas in existing Standard Python(which is already installed).So install it first(google it).

(II) We can check it through properties option of my computer icon.

(III) Now install latest version(any one above 3.4) of python.

(IV) Now move to script folder of python distribution in command prompt (through cmd command of windows).

(V) Execute following commands in command prompt serially.

```
>pip install numpy
>pip install six
>pip install pandas
```

Wait after each command for installation Now we will be able to use pandas in standard python distribution.

(VI) Type import pandas as pd in python (IDLE) shell.

(VII) If it executed without error(it means pandas is installed on your system).

Data Structures in Pandas

Two important data structures of pandas are– Series, DataFrame.

(I) **Series :** Series is like a one-dimensional array like structure with homogeneous data. For example, the following series is a collection of integers.

78	45	12	89	56

Basic feature of series are :

- Homogeneous data
- Size Immutable
- Values of Data Mutable.

(II) **DataFrame :** DataFrame is like a two-dimensional array with heterogeneous data.

SR. No.	Admn No	Student Name	Class	Section	Gender	Date of Birth
1	001284	Nidhi Mandal	I	A	Girl	07/08/2010
2	001285	Soumyadip Bhattacharya	I	A	Boy	24/02/2011
3	001286	Shreyaang Shandilya	I	A	Boy	29/12/2010

Basic feature of DataFrame are :

- Heterogeneous data
- Size Mutable
- Data Mutable.

Pandas Series

It is like one-dimensional array capable of holding data of any type (integer, string, float, python objects, etc.). Series can be created using constructor.

Syntax :- pandas.Series(data, index, dtype, copy)

Creation of Series is also possible from – ndarray, dictionary, scalar value.

Series can be created using:-

(I) Array

(II) Dict

(III) Scalar value or constant

Pandas Series :

Create an Empty Series

e.g.

```
import pandas as pseries
s = pseries.Series()
print(s)
Output
Series([], dtype: float64)
```

Create a Series from ndarray :

Without index
e.g.

```
import pandas as pd1
import numpy as np1
data = np1.array(['a','b','c','d'])
s = pd1.Series(data)
print(s)
```

Output

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

Note: default index is starting from 0

Without index position
e.g.

```
import pandas as p1
import numpy as np1
data = np1.array(['a','b','c','d'])
s = pd1.Series(data,index=[100, 101, 102, 103])
print(s)
```

Output

```
100    a
101    b
102    c
103d   dtype:
object
```

Note: index is starting from 100

Create a Series from dict :

Eg. 1(without index)

```
import pandas as pd1
import numpy as np1
data = {'a' : 0., 'b' : 1., 'c' : 2.}
s = pd1.Series(data)
print(s)
```

Output

```
a       0.0
b       1.0
c       2.0
dtype: float64
```

Eg.2 (with index)

```
import pandas as pd1
import numpy as np1
data = {'a' : 0., 'b' : 1., 'c' : 2.}
s = pd1.Series(data,index=['b','c','d','a'])
print(s)
```

Output

```
b       1.0
c       2.0
d       NaN
a       0.0
dtype: float64
```

Create a Series from Scalar :

e.g

```
import pandas as pd1
import numpy as np1
s = pd1.Series(5, index=[0, 1, 2, 3])
print(s)
```

Output

```
0 5
1 5
2 5
3 5
dtype: int64
```

Note : here 5 is repeated for 4 times (as per no of index).

Maths operations with Series :

e.g.

```
import pandas as pd1
s = pd1.Series([1,2,3])
t = pd1.Series([1,2,4])
u=s+t #addition operation print (u)
u=s*t # multiplication operation
print (u)
```

OUTPUT

```
0 2
1 4
2 7
dtype: int64
0 1
1 4
2 12
dtype: int64
```

Head function

e.g

```
import pandas as pd1
s  =  pd1.Series([1,2,3,4,5],index  =
['a','b','c','d','e'])
print (s.head(3))
```

Output

```
a.1
b. 2
c. 3
dtype: int64
```

Return first 3 elements

Tail function :

e.g

```
import pandas as pd1
s  =  pd1.Series([1,2,3,4,5],index  =
['a','b','c','d','e'])
print (s.tail(3))
```

Output

```
c.  3
d.  4
e.  5
dtype: int64
```

Return last 3 elements

Accessing Data from Series with indexing and slicing

e.g.

e.g.

```
import pandas as pd1
s = pd1.Series([1,2,3,4,5],index =
['a','b','c','d','d'])
print (s[0]) # for 0 index position
print (s[:3]) # for first 3 index values
print (s[-3:]) # slicing for last 3 index values
```

Output

```
1
a.    1
b.    2
c.    3
dtype: int64   c      3
d.    4
e.    5
dtype: int64
```

Retrieve Data Using Label as (Index):

e.g.

```
import pandas as pd1
s  =  pd1.Series([1,2,3,4,5],index  =
['a','b','c','d','e'])
print (s[['c','d']])
```

Output

```
c 3
d 4
dtype: int64
```

Retrieve Data from selection :

There are three methods for data selection:

(I) loc gets rows (or columns) with particular labels from the index.

(II) iloc gets rows (or columns) at particular positions in the index (so it only takes integers).

(III) ix usually tries to behave like loc but falls back to behaving like iloc if a label is not present in the index. ix is deprecated and the use of loc and iloc is encouraged instead.

Retrieve Data from selection :

```
e.g.
>>>s = pd.Series (np.nan,
index = [49, 48, 47, 46, 45, 1, 2, 3, 4, 5]
>>>s.iloc[:3]# slice the first three rows
49  NaN
48  NaN
47  NaN
>>>s.loc[:3] # slice up to and including label 3
49  NaN
48  NaN
47  NaN
46  NaN
45  NaN
1   NaN
2   NaN
3   NaN
```

```
>>>s.ix[:3]# the integer is in the index so
s.ix[:3] works like loc
49  NaN
48  NaN
47  NaN
46  NaN
45  NaN
1   NaN
2   NaN
3   NaN
```

Pandas DataFrame :

It is a two-dimensional data structure, just like any table (with rows & columns).

Basic Features of Data Frame

- Columns may be of different types
- Size can be changed (Mutable)
- Labeled axes (rows/columns)
- Arithmetic operations on rows and columns

Structure

Columns

SR. No.	Admn No	Student Name	Class	Section	Gender	Date of Birth
1	001284	Nidhi Mandal	I	A	Girl	07/08/2010
2	001285	Soumyadip Bhattacharya	I	A	Boy	24/02/2011
3	001286	Shreyaang Shandilya	I	A	Boy	29/12/2010

Rows

It can be created using constructor
pandas.DataFrame(data, index, columns, dtype, copy).

Create an Empty DataFrame :

```
e.g.
import pandas as pd1
df1 = pd1.DataFrame()
print(df1)
OUTPUT:
Empty
DataFrame
Columns: [ ]
Index: [ ]
Create a DataFrame from Lists :-
e.g.1
```

```
import pandas as pd1
data1 = [1,2,3,4,5]
df1 = pd1.DataFrame(data1)
print (df1)
OUTPUT:
0
0 1
1 2
2 3
3 4
4 5
e.g.2.
```

```
import pandas as pd1
data1                                   =
[['Freya',10],['Mohak',12],['Dwivedi',13]]
d              f                        1
=pd1.DataFrame(data1,columns=['Name','Age'])
print (df1)
OUTPUT:
   NAME      AGE
1  Freya     10
2  Mohak     12
2  Dwivedi   13
```

Write below for numeric value as float

```
df1 = pd1.DataFrame(data,columns
   = ['Name','Age'],dtype=float)
```

Create a DataFrame from Dict of ndarrays / Lists :

```
e.g.1
import pandas as pd1
data1        =         {'Name':['Freya',
'Mohak'],'Age':[9,10]}
df1 = pd1.DataFrame(data1)
print (df1)
Output
Name Age
1 Freya 9
2 Mohak 10
```

Write below as 3rd statement in above prog for indexing

```
df1  =  pd1.DataFrame(data1,  index=
['rank1','rank2', 'rank3','rank4'])
```

Create a DataFrame from List of Dicts :

```
e.g.1
import pandas as pd1
data1 = [{'x': 1, 'y': 2},{'x': 5, 'y': 4,
'z': 5}]
df1 = pd1.DataFrame(data1)
print (df1)
Output
x y z
0 1 2 NaN
1 5 4 5.0
```

Write below as 3rd stmnt in above program for indexing

```
df  =  pd.DataFrame(data,  index=['first',
'second'])
```

Create a DataFrame from Dict of Series :

```
e.g.1
import pandas as pd1
d1  =  {'one'  :  pd1.Series([1,  2,  3],
index=['a', 'b', 'c']),
'two' : pd1.Series([1, 2, 3, 4], index=['a',
'b', 'c', 'd'])}
df1 = pd1.DataFrame(d1)
print (df1)
Output
one two
a 1.0 1
b 2.0 2
c 3.0 3
d NaN 4
```

Column Selection -> print (df ['one'])

Adding a new column by passing as Series: -> df1['three']=pd1.Series([10,20,30],index=['a','b','c'])

Adding a new column using the existing columns values df1['four']=df1['one']+df1['three']

Create a DataFrame from .txt file:

Having a text file './inputs/dist.txt' as:

```
1 1 12.92
1 2 90.75
1 3 60.90
2 1 71.34
```

Pandas is shipped with built-in reader methods. For example the pandas.read_table method seems to be a good way to read (also in chunks) a tabular data file.

```
import pandas
df = pandas.read_table('./input/dists.txt',
delim_whitespace=True,
names=('A', 'B', 'C'))
```

will create a DataFrame objects with column named A made of data of type int64, B of int64 and C of float64.

Create a DataFrame from csv(comma separated value) file / import data from cvs file :

e.g.

Suppose filename.csv file contains following data

```
Date,"price","factor_1","factor_2"
2012-06-11,1600.20,1.255,1.548
2012-06-12,1610.02,1.258,1.554
```

```
import pandas as pd
# Read data from file 'filename.csv'
# (in the same directory that your python program
is based)
# Control delimiters, rows, column names with
read_csv data = pd.read_csv("filename.csv")
# Preview the first 1 line of the loaded data
data.head(1)
```

Column addition :

```
df = pd.DataFrame({"A": [1, 2, 3], "B": [4, 5, 6]})
c = [7,8,9]
df['C'] = c
```

Column Deletion :

```
del df1['one'] # Deleting the first column using DEL
function
df.pop('two') #Deleting another column using POP
function
```

Rename columns :

```
df = pd.DataFrame({"A": [1, 2, 3], "B": [4, 5, 6]})
>>> df.rename(columns={"A": "a", "B": "c"})
a c
0 1 4
1 2 5
2 3 6
```

Row Selection, Addition, and Deletion:

```
#Selection by Label
import pandas as pd1
d1 = {'one' : pd1.Series([1, 2, 3], index=['a', 'b', 'c']),
'two' : pd1.Series([1, 2, 3, 4], index=['a', 'b', 'c', 'd'])}
df1
= pd1.DataFrame(d1)
print (df1.loc['b'])
Output
one 2.0
two 2.0
Name: b, dtype: float64
#Selection by integer location
import pandas as pd1
d1 = {'one' : pd1.Series([1, 2, 3], index=['a', 'b', 'c']),
'two' : pd1.Series([1, 2, 3, 4], index=['a', 'b', 'c','d'])}
df1 = pd1.DataFrame(d1)
print (df1.iloc[2])
Output
one 3.0
two 3.0
Name: c, dtype: float64
```

Slice Rows :

Multiple rows can be selected using ' : 'operator.

```
print (df1[2:4])
```

Addition of Rows :

```
import pandas as pd1
df1 = pd1.DataFrame([[1, 2], [3, 4]], columns =
['a','b'])
df2 = pd1.DataFrame([[5, 6], [7, 8]], columns =
['a','b'])
 df1 = df1.append(df2)
print (df1)
Deletion of Rows
# Drop rows with label 0
df1 = df1.drop(0)
```

Iterate over rows in a dataframe :

e.g.

```
import pandas as pd1
import numpy as np1
raw_data1 = {'name': ['freya', 'mohak'],
'age': [10, 1],
'favorite_color': ['pink', 'blue'],
'grade': [88, 92]}
df1 = pd1.DataFrame(raw_data1, columns =
['name', 'age', 'favorite_color', 'grade'])
for index, row in df1.iterrows():
print (row["name"], row["age"])
Output
freya 10
mohak 1
```

Head & Tail

head() returns the first n rows (observe the index values). The default number of elements to display is five, but you may pass a custom number. tail() returns the last n rows .e.g.

```
import pandas as pd
import numpy as np
#Create a Dictionary of series
d = {'Name':pd.Series(['Tom','James','Ricky',
'Vin','Steve', 'Smith','Jack']), 'Age':pd.Series
([25,26,25, 23,30,29,23]), 'Rating':pd.Series
([4.23,3.24,3.98, 2.56,3.20,4.6,3.8])}
#Create a DataFrame
df = pd.DataFrame(d)
print ("Our data frame is:")
 print df
print ("The first two rows of the data frame is:")
print df.head(2)
```

Indexing a DataFrame using .loc[ ] :

This function selects data by the label of the rows and columns.

```
#import the pandas library and aliasing as pd
import pandas as pd
import numpy as np
df = pd.DataFrame(np.random.randn(8, 4),
index = ['a','b','c','d','e','f','g','h'],
columns = ['A', 'B', 'C', 'D'])
#select all rows for a specific column
print df.loc[:,'A'].
```

Accessing a DataFrame with a boolean index :

In order to access a dataframe with a boolean index, we have to create a dataframe in which index of dataframe contains a boolean value that is "True" or "False".

```
# importing pandas as pd
import pandas as pd
# dictionary of lists
dict = {'name':["Mohak", "Freya", "Roshni"],
'degree': ["MBA", "BCA", "M.Tech"],
 'score':[90, 40, 80]}
# creating a dataframe with boolean index
df = pd.DataFrame(dict, index = [True, False, True])
# accessing a dataframe using .loc[] function
print(df.loc[True])
#it will return rows of Mohak and Roshni
only(matching true only)
```

import csv file in Pandas DataFrame :

e.g.

```
import pandas as pd
# Takes the file's folder
filepath = r"csv file path"
 # read the CSV file
df = pd.read_csv(filepath)
# print the first five rows
print(df.head())
```

Export Pandas DataFrame to a CSV File

e.g.

```
import pandas as pd
cars = {'Brand': ['Honda Civic','Toyota
Corolla','Ford Focus','Audi A4'],
'Price':[22000,25000,27000,35000]
 }
df = pd.DataFrame(cars, columns= ['Brand',
'Price'])
df.to_csv (r'C:\export_dataframe.csv',
index = False, header=True)
print (df)
```