NumPy Limitations

1. Supports only homogeneous data types. Some of the elements' types are changed to end up with a homogeneous list. This is known as type coercion.

2. The typical arithmetic operators, such as +, -, * and / have a different meaning for regular Python lists and numpy arrays.

3. Difficult to handle certain data format like Excel, SqL Database etc

Data Analytics Tools : Pandas

Introduction

- pandas is an open-source, BSD-licensed Python library providing high-performance, easy-to-use data structures and data analysis tools for the Python programming language.
- Full form: Panel Data
- pandas is built on top of NumPy library.
- pandas is well suited for many different kinds of data:
  - Tabular data: Ex-SQL, Excel spreadsheet
  - Ordered and Unordered data: (Quantification)
    - Eg: Marital Status (U)
    - Eg: Mark of Students(O)
  - Arbitrary matrix data with row and column labels
    - Ex- Temperature of City (Row-WeekDays, Col-Time)
  - Any other form of observational / statistical data sets.

Name	Age	Sex	Year
A	21	M	3rd
B	23	F	3rd
C	22	M	4th

Data 1

Age
21
23
22

Data 2

Name
A
B
C

Data 3

Can you guess what kind of data they are ?

Pandas Data Structure

1. **Series :** Series is a one-dimensional labeled array capable of holding data of any type (integer, string, float, python object etc)

2. **DataFrame:** A Data frame is a two-dimensional data structure, i.e., data is aligned in a tabular fashion in rows and columns.

3. **Panel:** A panel is a 3D container of data (Now removed from the current version)

Convert list into series of elements

```
import pandas as pd
my_data=[10,20,30,40,50]
s1 = pd.Series(data=my_data,index=['a', 'b', 'c', 'd', 'e'])
print(s1)
# convert element lists into series of elements, which have indexes are from ‘a’ to ‘e’
```

Convert dictionary into series of elements

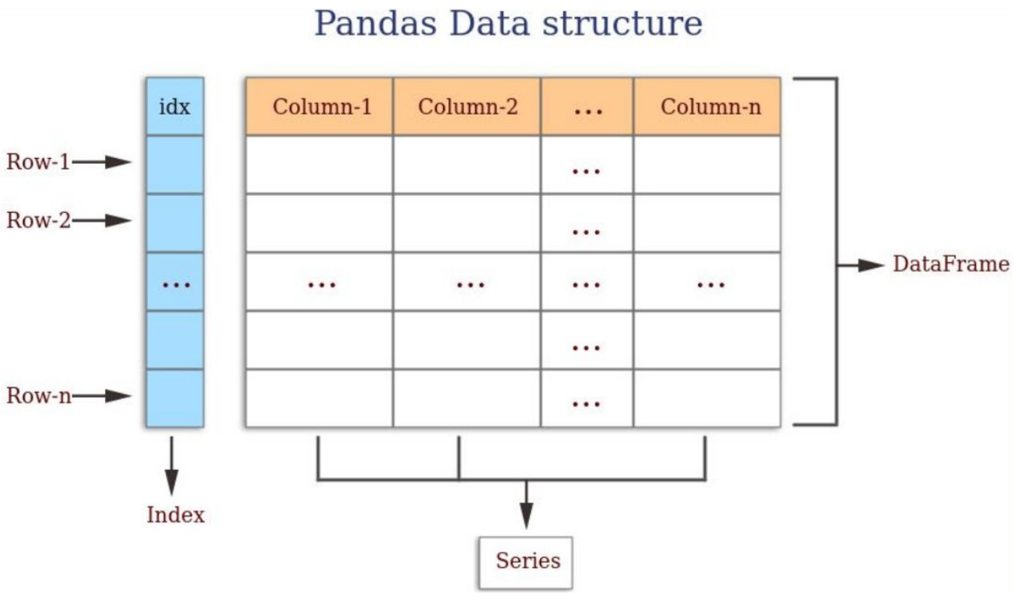
```
import pandas as pd
d={'a':10,'b':20,'c':30,'d':40}
#dictionary keys act as index and values with every key act as series values
s2=pd.Series(d)
print(s2)
```

Addition of two series

```
# Addition of two series
import pandas as pd
ser1=pd.Series([1,2,3,4],['India','Srilanka', 'Bangladesh', 'Russia'])
ser2=pd.Series([1,2,5,9],['India','Srilanka', 'Bangladesh', 'Russia'])
print(ser1+ser2)
```

Pandas DataFrames

Pandas DataFrame consists of three main components: the data, the index, and the columns.



Create Dataframe & Select columns

```
from numpy.random import randn
import pandas as pd
np.random.seed(101)
df1=pd.DataFrame(randn(5,4),['A','B','C','D','E'],['W','X','Y','Z'])
#generate random number for 5 rows and 4 columns
print(df1['W'])
print(df1[['W','Z']])
```

Data Manipulation: Data selection

DataFrame.loc() will select rows by index values
DataFrame.iloc() will select rows by rows numbers

```
df1.loc['A']                    # fetch particular row from dataset having index ‘A’
df1.iloc[3]                     # fetch 3rd row from dataset
df1.loc[['A','C'],['X','Z’]]    # fetch a subset of data from given dataset
df1 > 0
df1[df1>0]            # fetch original values
# df1.drop('A',axis=0,inplace=False)
# df1.drop('W',axis=1,inplace=False)
# df1.drop('W',axis=1,inplace=True)
```

Data Manipulation: Data selection

```
df2=pd.DataFrame(randn(5,4),['A','B','C','D','E'],['W','X','Y','Z’])
df2[df2['W']>0]
df2[df2['W']>0][['X','Y']]
# fetch out desired frame of X & Y from dataset, for those rows where value is more than 0 in ‘W’ column

df3=df2.reset_index() #assign natural index
#df3=df2.set_index('Z') #set ‘Z’ column as index value
df3
```

Data Manipulation: Drop missing elements

```
import pandas as pd
d={'A':[1,2,np.NaN], 'B':[1,np.NaN,np.NaN],'C':[1,2,3]}
# np.NaN is the missing element in DataFrame
df4=pd.DataFrame(d)
df4.dropna()            #pandas would drop any row with missing value
df4.dropna(axis=1)              #drop column with NULL value
df4.dropna(thresh=2)    #Require <2 non-NA values to drop row.
```

Data Manipulation: Filling suitable value

```
df4.fillna(value='FILL VALUE')          #NaN is replaced by value=FILL VALUE

df4['A'].fillna(value=df4['A'].mean())

#Select column "A" and fill the missing value with mean value of the column A
```

OR

```
df['A’].fillna(value=df['A’].std())

#Select column "A" and fill the missing value with standard deviation value of the column A
```

Replacing

- Many times, we have to replace a generic value with some specific value.
- We can achieve this by applying the replace method.
- Replacing NA with a scalar value is equivalent behavior of the fillna() function.

```
import numpy as np
df5 = pd.DataFrame({'one':[10,20,30,40,50,2000], 'two':[1000,0,30,40,50,60]})
print df5.replace({1000:10,2000:60})
```

Groupby() function

```
data = {'Company': [ 'CompA', 'CompA', 'CompB', 'CompB', 'CompC', 'CompC'],
        'Person': ['Rajesh', 'Pradeep', 'Amit', 'Rakesh', 'Suresh', 'Raj'],
        'Sales': [200, 120, 340, 124, 243, 350]}
df6=pd.DataFrame(data)
print(df6)
comp=df6.groupby("Company")                #grouping done using label name “Company”
print(comp.mean())                         #mean appliead on grouped data
comp_std=df6.groupby("Company").std()      #grouping done + standard deviation applied”
comp_std
list(comp)[1]
```

Groupby() function

```
df6.groupby("Company").sum().loc[“CompB"]

# group data by ‘company’ label, apply sum function such that all
data of same company gets added and then fetch Company
“CompB” value after summation
```

Finding unique value & number of occurrence from Dataframe

```
df =
pd.DataFrame({'col1':[1,2,3,4],'col2':[444,555,666,444],'col3':['abc','def','ghi','xyz']}
)
# col1, col2 & col3 are column labels, each column have their own values


df['col2'].unique()          #fetches the unique values available in column
df['col2'].value_counts() # count number of occurance of every value
```

File Handling

A comma-separated values (CSV) file is a plaintext file with a .csv extension that holds tabular data.

Each row of the CSV file represents a single table row.

Read a CSV File

```
df1 = pd.read_csv(‘filename_with_path’)
```

Write a CSV File

```
df2.to_csv(‘filename_with_path’)
```

File Handling

The **Excel spreadsheet** app lets you create, view, edit, and share your files with others quickly and easily. Create **spreadsheets**, data analyses, charts, budgets and more while you view and edit workbooks

It also holds tabular data.

Read a Excel File

```
df = pd.read_excel('filename_with_path',sheet_name='SheetName')
```

Write a Excel File with Single sheet

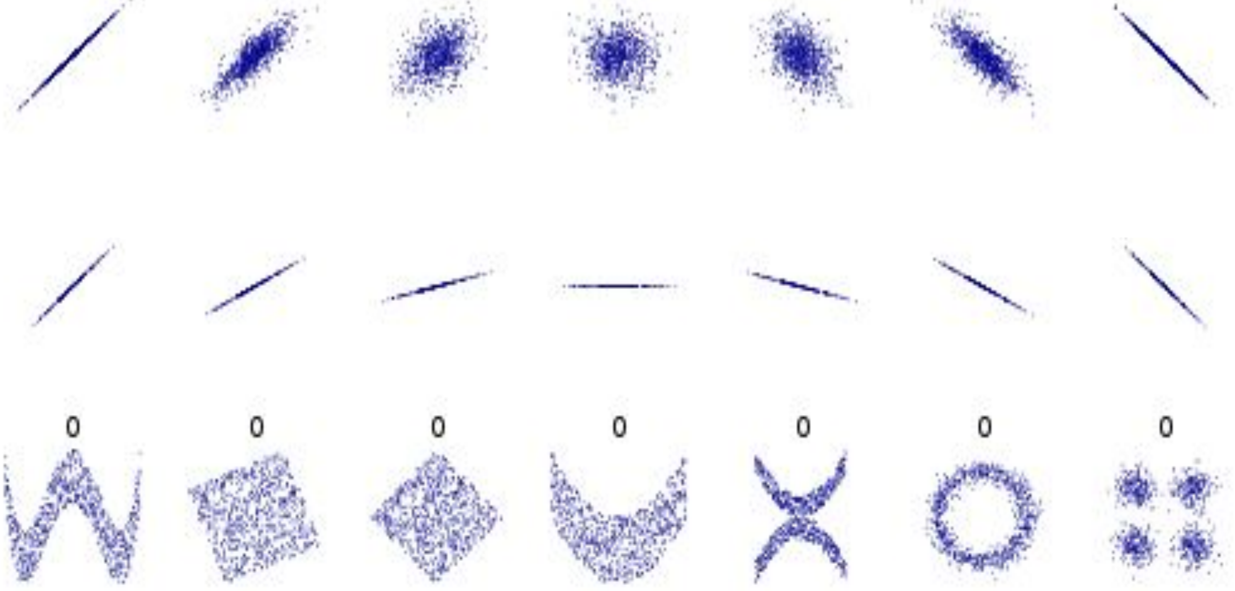
```
df.to_excel('filename_with_path',sheet_name='SheetName')
```

Statistical Functions

- **Covariance** is applied on series data. The Series object has a method cov() to compute covariance between series objects.
- “**Covariance**” indicates the direction of the linear relationship between variables.

+Ve, Value depends on no.of Data Points

-Ve, Value depends on no.of Data Points



Statistical Functions

Covariance

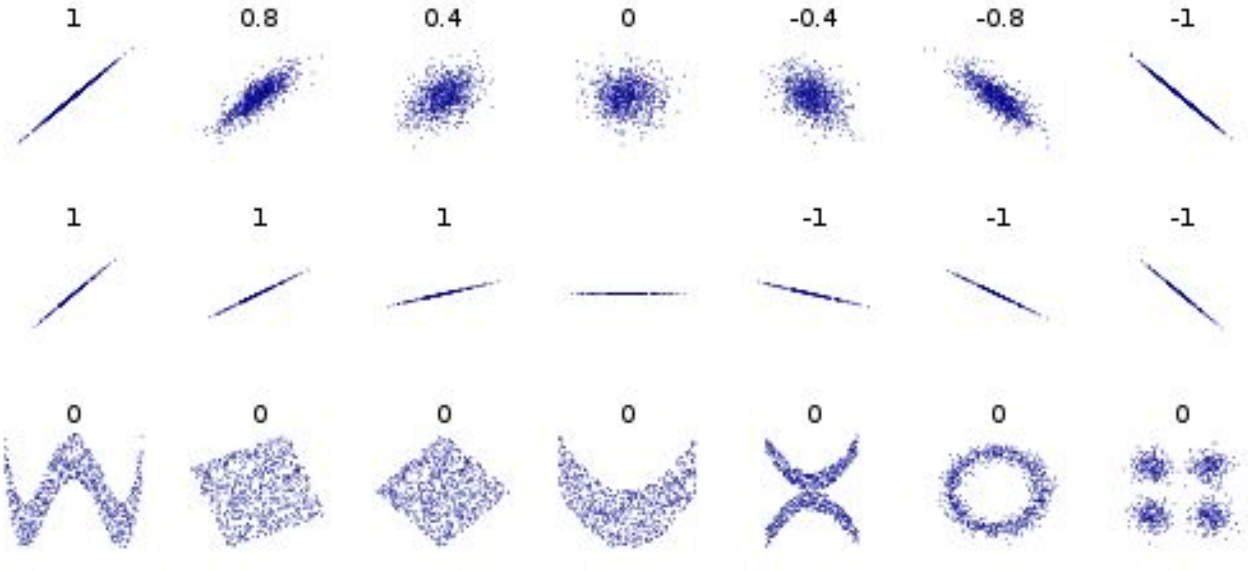
```
import numpy as np
s1 = pd.Series(np.random.randn(10))
s2 = pd.Series(np.random.randn(10))
print s1.cov(s2)
```

Statistical Functions

Correlation(-1 to 1) measures both the strength **and** direction of the linear relationship between two variables.

There are multiple methods to compute the correlation like

→ pearson(default),
→ Spearman
→ kendall.



https://commons.wikimedia.org/wiki/File:Correlation_examples2.svg

Statistical Functions

Correlation

```
from numpy.random import randn
s1 = pd.Series(randn(10))
s2 = pd.Series(randn(10))
print(s1.corr(s2))
```

Statistical Functions

“**Covariance**” indicates the direction of the linear relationship between variables. “**Correlation**” on the other hand measures both the strength **and** direction of the linear relationship between two variables

Statistical Functions

Data Ranking produces ranking for each element in the array of elements
Large value assigned higher rank.

In case of ties, assigns the mean rank.

```
import pandas as pd
s = pd.Series([6,8,7,6,5], index=list('abcde'))
# s['a'] = s['d'] # so there's a tie
for i in range(5):
    print(s[i],'\t', s.rank()[i])
```