Numpy and Pandas

Agenda

In this session, you will learn about:

- Basics of Numpy
- Introduction to Pandas
- Data Structure in Python
- Descriptive Analysis
- Function Applications
- Reindexing
- I/O Tools

Basics of NumPy

Basics of NumPy

NumPy is the fundamental package for scientific computing with Python.

'Numerical Python'

- **Provides powerful N-dimensional array object**
- **sophisticated (broadcasting) functions**
- **tools for integrating C/C++ and Fortran code**
- **useful linear algebra, Fourier transform, and random number capabilities**

NumPy can also be used as an efficient multi-dimensional container of generic data. Arbitrary data-types can be defined. This allows NumPy to seamlessly and speedily integrate with a wide variety of databases.

Basics of NumPy

In 2006

Created NumPy package by incorporating the features of Numarray into Numeric package

Operations using NumPy

Using NumPy, a developer can perform the following operations:

01 Mathematical and logical operations on arrays

02 Fourier transforms and routines for shape manipulation

03 Operations related to linear algebra

NumPy – A Replacement for MatLab

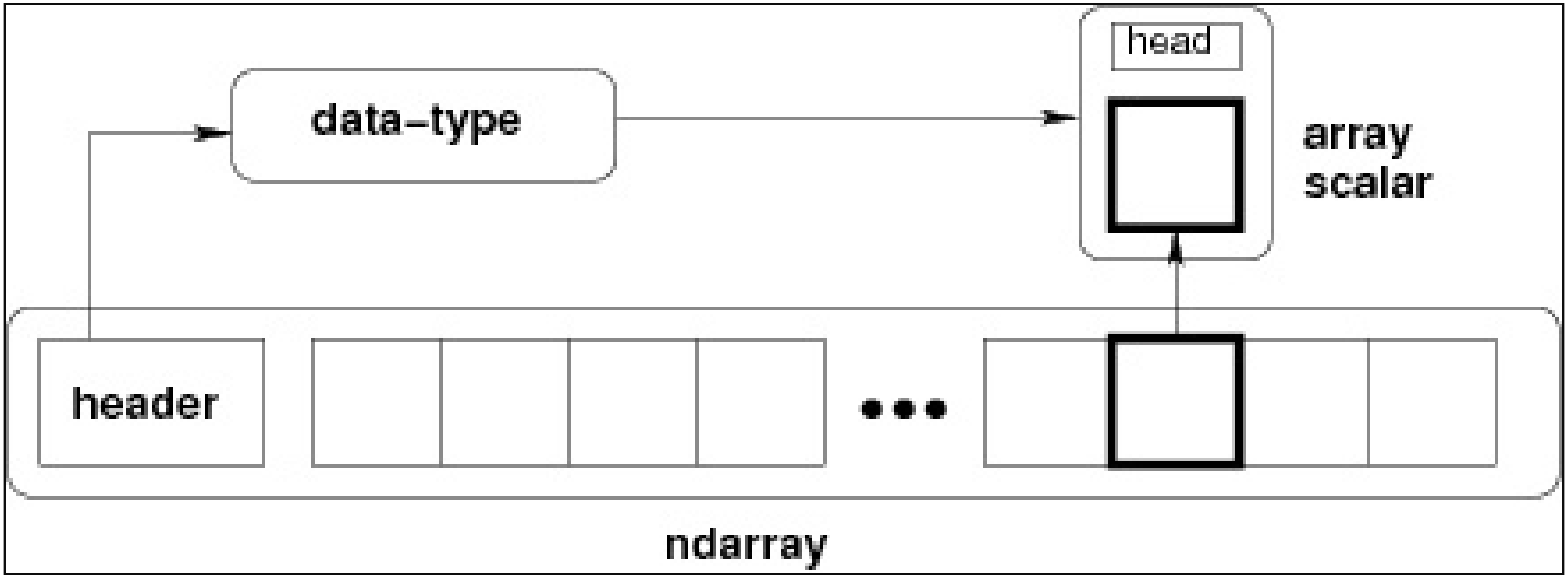
1. NumPy is often used along with packages like SciPy (Scientific Python) and Matplotlib (plotting library)

2. This combination is widely used as a replacement for MatLab, a popular platform for technical computing

3. Python alternative to MatLab is now seen as a more modern and complete programming language

4. It is open source, which is an added advantage of NumPy

NumPy – Ndarray Object



- The most important object defined in NumPy is an N-dimensional array type called ndarray
- It describes the collection of items of the same type. Items in the collection can be accessed using a zero-based index
- Every item in an ndarray takes the same size of block in the memory
- Each element in ndarray is an object of data-type object

Ndarray Object – Array Function

The basic ndarray is created using an array function in NumPy

```
numpy.array
```

It creates an ndarray from any object exposing array interface

```
import numpy

numpy.array(object, dtype = None, copy = True, order = None,
           ndmin= 0 )
```

Ndarray Object – Parameter

Sr. No.	Parameters & Description
1	**object** Any object exposing the array interface method returns an array, or any (nested) sequence
2	**dtype** Desired data type of array, optional
3	**copy** Optional. By default (true), the object is copied
4	**order** C (row major) or F (column major) or A (any) (default)
5	**ndmin** Specifies minimum dimension of resultant array

Ndarray Object – Parameter

```
>>> import numpy as np
>>> a = np.arange(15).reshape(3, 5)
>>> a
array([[ 0,  1,  2,  3,  4],
       [ 5,  6,  7,  8,  9],
       [10, 11, 12, 13, 14]])
```

```
>>> a.shape
(3, 5)
```

```
>>> a.ndim
2
```

```
>>> a.size
15
```

Ndarray Object – Parameter

```
>>> import numpy as np
>>> a = np.arange(15).reshape(3, 5)
>>> a
array([[ 0,  1,  2,  3,  4],
       [ 5,  6,  7,  8,  9],
       [10, 11, 12, 13, 14]])
```

```
>>> type(a)
<type 'numpy.ndarray'>
```

```
>>> a.dtype.name
 'int64'
```

```
>>> b = np.array([6, 7, 8])
>>> b
array([6, 7, 8])
```

```
>>> type(b)
<type 'numpy.ndarray'>
```

Data Type Objects

A data type object describes interpretation of fixed block of memory corresponding to an array

Depending on the following aspects:

1. Type of data (integer or float)
2. Size of data
3. Byte order
4. If structured type
5. If Data type is a subarray

The byte order is decided by prefixing '<' or '>' to data type.

'<' means that encoding is little-endian

'>' means that encoding is big-endian

Data Type Objects – Syntax and Parameter

A dtype object is constructed using the following syntax:

```
numpy.dtype(object, align, copy)
```

Parameters

1	2	3
Object	**Align**	**Copy**
To be converted to data type object	If true, adds padding to the field to make it similar to C-struct	Makes a new copy of dtype object

I/O with Numpy

The ndarray objects can be saved to and loaded from the disk files.

NumPy introduces a simple file format for ndarray objects.

The IO functions available are –

load() and **save()** functions handle /numPy binary files

loadtxt() and **savetxt()** functions handle normal text fill

numpy.save()

The numpy.save() file stores the input array in a disk file with npy extension

```
import numpy as np
a = np.array ( [1,2,3,4,5] )
np.save('outfile',a)
```

To reconstruct array from outfile.npy, use load() function

```
import numpy as np
b = np.load('outfile.npy')
print (b)
```

It will produce the following output :

```
arrry( [1, 2, 3, 4, 5] )
```

numpy.save()

The save() and load() functions accept an additional Boolean parameter “allow_pickles”

A pickle in Python is used to serialize and de-serialize objects before saving to or reading from a disk file

savetxt()

The storage and retrieval of array data in simple text file format is done with **savetxt()** and **loadtxt()** functions.

```
import numpy as np
a = np.array ( [1,2,3,4,5] )
np.save('out.txt',a)
b = np.loadtxt('out.txt')
print (b)
```

Output

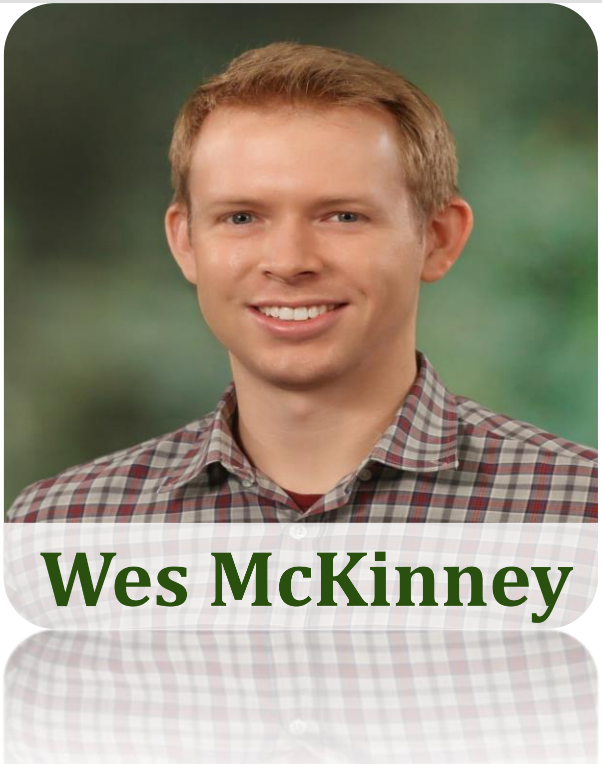
```
[ 1.  2.  3.  4.  5.]
```

The savetxt() and loadtxt() functions accept additional optional parameters such as:

- Header
- Footer
- Delimiter

Pandas

The word pandas is derived from "Python and data analysis" and "panel data"



The most powerful and flexible open source data analysis / manipulation tool available in any language

pandas is a Python package providing fast, flexible, and expressive data structures designed to make working with “relational” or “labeled” data both easy and intuitive.

Pandas in Python

Tabular data with heterogeneously-typed columns, as in an SQL table or Excel spreadsheet

Ordered and unordered (not necessarily fixed-frequency) time series data.

Arbitrary matrix data (homogeneously typed or heterogeneous) with row and column labels

Any other form of observational / statistical data sets. The data actually need not be labeled at all to be placed into a pandas data structure

Pandas in Python (Contd.)

Python was majorly used for data munging and preparation.

Python had very little contribution towards data analysis and Pandas solved this problem.

Five typical steps in the processing and analysis of data

01 Load

02 Prepare

03 Manipulate

04 Model

05 Analyze

Python with Pandas is used in a wide range of fields including academic and commercial domains including:

- Finance
- Economics
- Statistics
- Analytics

Key features of Pandas (1/3)

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

2. Tools for loading data into in-memory data objects from different file formats

3. Data alignment and integrated handling of missing data

Key features of Pandas (2/3)

4

Reshaping and pivoting of date sets

5

Label-based slicing, indexing and subsetting of large data sets

6

Columns from a data structure can be deleted or inserted

Key features of Pandas (3/3)

7

Group by data for aggregation and transformations

8

High performance merging and joining of data

9

Time Series functionality

Environment Setup

Standard Python distribution doesn't come bundled with Pandas module

A lightweight alternative is to install NumPy using popular Python package installer, **pip**

```
pip  install pandas
```

For LINUX (Ubuntu Users)

Package managers of respective Linux distributions are used to install one or more packages in SciPy stack.

```
sudo apt-get install python-numpy python-scipy python-matplotlibipythonipythonnotet
python-pandas python-sympy python-nose
```

Data Structures in Pandas

Data Structures

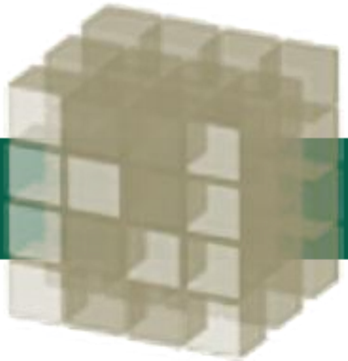
Pandas deals with the following three data structures:

Series

DataFrame

Panel

These data structures are built on top of Numpy array.



Dimension and Description

The best way to think of these data structures is that the higher dimensional data structure is a container of its lower dimensional data structure.

Example

DataFrame is a container of Series

Panel is a container of DataFrame

Dimension and Description (Contd.)

Data Structure	Dimensions	Description
Series	1	1D labeled homogeneous array, size-immutable
Data Frames	2	General 2D labeled, size-mutable tabular structure with potentially heterogeneously typed columns
Panel	3	General 3D labeled, size-mutable array

Mutability

- **All Pandas data structures are value mutable**
- **Except Series all are size mutable.**

Series	Size immutable
DataFrame	Widely used and is most important data structures
Panel	Used much less

Series

Series is a one-dimensional labeled array capable of holding data of any type

Integer | **String** | **Float** | **Python Objects**

The axis labels are collectively called index.

Collection of integers

10	23	56	17	52	61	73	90	26	72

Key Points

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

pandas.Series

A pandas Series can be created using the following constructor:

```
pandas.Series( data, index, dtype, copy)
```

Parameters of constructor

S.No	Parameter & Description
1	**data** data takes various forms like ndarray, list, constants
2	**index** Index values must be unique and hashable, same length as data. Default **np.arrange(n)** if no index is passed.
3	**dtype** dtype is for data type. If None, data type will be inferred
4	**copy** Copy data. Default False

pandas.Series (Contd.)

A basic Series that can be created is an Empty Series

Example

```
# import the pandas library and aliasing as pd
import pandas as pd
s = pd.Series()
print(s)
```

Output

```
Series([], dtype: float64)
```

A series can be created using the following:

Array	Dict	Scalar Value

DataFrame

DataFrame is a two-dimensional array with heterogeneous data.

Name	Age	Gender	Rating
Steve	32	Male	3.45
Lia	28	Female	4.6
Vin	45	Male	3.9
Katie	38	Female	2.78

- The above table represents the data of a sales team of an organization with their overall performance rating
- The data is represented in rows and columns
- Each column represents an attribute and each row represents a person

Features of DataFrame

Four key features of Dataframe:

Potentially columns are of different types

Size – Mutable

Features of Dataframe

Can perform arithmetic operations on numeric rows and columns

Labeled axes

Structure of DataFrame

Creating a data frame with student's data

Columns

Rows

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

pandas.DataFrame

A pandas DataFrame can be created using the following constructor:

```
pandas.DataFrame( data, index, columns, dtype, copy)
```

Parameters of constructor

S.No	Parameter & Description
1	**data** data takes various forms like ndarray, series, map, lists, dict, constants and also another DataFrame.
2	**index** For the row labels, the Index to be used for the resulting frame is Optional Default np.arrange(n) if no index is passed.
3	**columns** For column labels, the optional default syntax is - np.arrange(n). This is only true if no index is passed.
4	**dtype** Data type of each column.
4	**copy** This command (or whatever it is) is used for copying of data, if the default is False.

Create DataFrame

A basic DataFrame can be created is an Empty Series

Example

```
# import the pandas library and aliasing as pd
import pandas as pd
df= pd.DataFrame( )
print (df)
```

Output

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

A pandas DataFrame can be created using various inputs like:

- Lists
- Dict
- Series
- Another DataFrame

Data Types of Column

The data types of the four columns are as follows:

Column	Type
Name	String
Age	Integer
Gender	String
Rating	Float

Key Points

- **Heterogeneous data**
- **Size Mutable**
- **Values of Data Mutable**

Panel

- A panel is a 3D container of data
- The term Panel data is derived from econometrics and is partially responsible for the name pandas – pan(el)-da(ta)-s

The names for the 3 axes are intended to give some semantic meaning to describing operations involving panel data

They are:

items – axis 0	Each item corresponds to a DataFrame contained inside
major_axis – axis 1	It is the index (rows) of each of the DataFrames
minor_axis – axis 2	It is the columns of each of the DataFrames

pandas.Panel( )

A Panel can be created using the following constructor:

```
pandas.Panel(data, items, major_axis, minor_axis, dtype, copy)
```

Parameters of constructor

Parameter	Description
data	Data takes various forms like ndarray, series, map, lists, dict, constants and also another DataFrame
items	axis=0
major_axis	axis=1
minor_axis	axis=2
dtype	Data type of each column
copy	Copy data. Default, **false**

Descriptive Statistics

Descriptive Statistics

Descriptive statistics provide simple summaries about the sample and about the observations that have been made

The *describe()* function on the Pandas DataFrame lists 8 statistical properties of each attribute:

- Count
- Mean
- Standard Deviation
- Minimum Value
- 25th Percentile
- 50th Percentile (Median)
- 75th Percentile
- Maximum Value

Functions & Description

Sr. No.	Function	Description
1	count()	Number of non-null observation
2	sum()	Sum of values
3	mean()	Mean of values
4	median()	Median of values
5	mode()	Mode of values
6	std()	Standard deviation of the values
7	min()	Minimum value
8	max()	Maximum value
9	abs()	Absolute value
10	prod()	Product of values
11	cumsum()	Cumulative Sum
12	cumprod	Cumulative Product

Functions & Description

Since, DataFrame is a Heterogeneous data structure

Generic operations don't work with all functions

Functions like sum(), cumsum() work with both numeric and character (or) string data elements without any error

Functions like abs(), cumprod() throw exception when the DataFrame contains character or string data

Descriptive Statistics

Example

```
import pandas as pd
import numpy as np
# Create a Dictionary of series
d =
{'Name':pd.Series(['Tom','James','Ricky','Vin','Steve','Smith','Jack','Lee','David',
'Gasper','Betina','Andres']),'Age':pd.Series([25,26,25,23,30,29,23,34,40,30,51,46]),
'Rating':pd.Series([4.23,3.24,3.98,2.56,3.20,4.6,3.8,3.78,2.98,4.80,4.10,3.65])}
```

```
# Crete a DataFrame
df = pd.DataFrame(d)
print(df)
```

Output

	Age	Name	Rating
0	25	Tom	4.23
1	26	James	3.24
2	25	Ricky	3.98
3	23	Vin	2.56
4	30	Steve	3.20
5	29	Smith	4.60
6	23	Jack	3.80
7	34	Lee	3.78
8	40	David	2.98
9	30	Gasper	4.80
10	51	Betina	4.10
11	46	Andres	3.65

sum( )

Returns the sum of the values for the requested axis

```
import pandas as pd
import numpy as np
# Create a Dictionary of series
d =
{'Name':pd.Series(['Tom','James','Ricky','Vin','Steve','Smith','Jack','Lee','David',
'Gasper','Betina','Andres']),'Age':pd.Series([25,26,25,23,30,29,23,34,40,30,51,46]),
'Rating':pd.Series([4.23,3.24,3.98,2.56,3.20,4.6,3.8,3.78,2.98,4.80,4.10,3.65])}
# Crete a DataFrame
df = pd.DataFrame(d)
print(df.sum())
```

Output

```
Age                                                382
Name      TomJamesRickyVinSteveSmithJackLeeDavidGasperBe...
Rating                                           44.92
dtype: object
```

mean( )

Returns the average value

```
import pandas as pd
import numpy as np
# Create a Dictionary of series
d =
{'Name':pd.Series(['Tom','James','Ricky','Vin','Steve','Smith','Jack','Lee','David',
'Gasper','Betina','Andres']),'Age':pd.Series([25,26,25,23,30,29,23,34,40,30,51,46]),
'Rating':pd.Series([4.23,3.24,3.98,2.56,3.20,4.6,3.8,3.78,2.98,4.80,4.10,3.65])}
# Crete a DataFrame
df = pd.DataFrame(d)
print(df.mean())
```

Output

```
Age       31.833333
Rating     3.743333
dtype: float64
```

std( )

Returns the Bressel standard deviation of the numerical columns

```
import pandas as pd
import numpy as np
# Create a Dictionary of series
d =
{'Name':pd.Series(['Tom','James','Ricky','Vin','Steve','Smith','Jack','Lee','David',
'Gasper','Betina','Andres']),'Age':pd.Series([25,26,25,23,30,29,23,34,40,30,51,46]),
'Rating':pd.Series([4.23,3.24,3.98,2.56,3.20,4.6,3.8,3.78,2.98,4.80,4.10,3.65])}
# Crete a DataFrame
df = pd.DataFrame(d)
print(df.std())
```

Output

```
Age        9.232682
Rating     0.661628
dtype: float64
```

describe( )

Generates descriptive statistics that summarize the central tendency, dispersion and shape of a dataset's distribution, excluding NaN values.

```
import pandas as pd
import numpy as np
# Create a Dictionary of series
d =
{'Name':pd.Series(['Tom','James','Ricky','Vin','Steve','Smith','Jack','Lee','David',
'Gasper','Betina','Andres']),'Age':pd.Series([25,26,25,23,30,29,23,34,40,30,51,46]),
'Rating':pd.Series([4.23,3.24,3.98,2.56,3.20,4.6,3.8,3.78,2.98,4.80,4.10,3.65])}
# Crete a DataFrame
df = pd.DataFrame(d)
df.describe
```

describe( )

Output

	Age	Rating
count	12.000000	12.000000
mean	31.833333	3.743333
std	9.232682	0.661628
min	23.000000	2.560000
25%	25.000000	3.230000
50%	29.500000	3.790000
75%	35.500000	4.132500
max	51.000000	4.800000

- For **numeric data**, the result's index will include count, mean, std, min, max as well as lower, 50 and upper percentiles.
- By default the lower percentile is 25 and the upper percentile is 75.
- The 50 percentile is the same as the median.

- For **object data** (e.g. strings or timestamps), the result's index will include count, unique, top, and freq.

Function Application

Function Application

To apply own or another library's functions to Pandas objects, one should be aware of the three important methods:

The appropriate method to use depends on whether your function expects to operate on an entire DataFrame, row- or column-wise, or element wise.

Table wise Function Application	pipe()
Row or Column Wise Function Application	apply()
Element wise Function Application	applymap()

Table-wise Function Application

Custom operations can be performed by passing the function and the appropriate number of parameters as pipe arguments.

Operation is performed on the whole DataFrame.

Example

Add a value 2 to all the elements in the DataFrame, then adder function.

The adder function adds two numeric values as parameters and returns the sum.

```
def adder(ele1,ele2):
       return (ele1 + ele2);
```

Use the custom function to conduct operation on the DataFrame.

```
df = pd.DataFrame(np.random.randn(5,3),columns=['col1','col2','col3'])
df.pipe(adder,2)
```

Table-wise Function Application

Example

```
import pandas as pd
import numpy as np
def adder(ele1,ele2):
       return (ele1 + ele2);
df = pd.DataFrame(np.random.randn(5,3),columns=['col1','col2','col3'])
df.pipe(adder,2)
```

Output

	col1	col2	col3
0	2.315234	2.921155	2.573712
1	2.920473	1.242196	0.391140
2	2.646938	1.317068	2.713296
3	2.471674	3.337855	1.690517
4	2.021106	1.098972	2.168525

Row or Column Wise Function Application

Arbitrary functions can be applied along the axes of a DataFrame or Panel using the apply() method.

Example

```
import pandas as pd
import numpy as np
df = pd.DataFrame(np.random.randn(5,3),columns=['col1','col2','col3'])
df.apply(np.mean)
```

Output

```
col1   -0.019656
col2    0.556761
col3    0.078555
dtype: float64
```

Element Wise Function Application

The methods applymap() on DataFrame and analogously map() on Series accept any Python function taking a single value and returning a single value.

Example

```
import pandas as pd
import numpy as np
df = pd.DataFrame(np.random.randn(5,3),columns=['col1','col2','col3'])
# My custom function
df['col1'].map(lambda x:x*100)
print(df.apply(np.mean))
```

Output

```
col1    0.133485
col2   -0.083308
col3    0.433805
dtype: float64
```


Reindexing

Reindexing

Reindexing changes the row labels and column labels of a DataFrame.

To reindex means to conform the data to match a given set of labels along a particular axis.

Multiple operations can be accomplished through indexing:

Reorder the existing data to match a new set of labels

Insert missing value (NA) markers in label locations where no data for the label existed

Reindexing

Example

```
import pandas as pd
import numpy as np
N = 20
df = pd.DataFrame({
    'A': pd.date_range(start='2016-01-01',periods=N,freq='D'),
    'x': np.linspace(0,stop=N-1,num=N),
    'y': np.random.rand(N),
    'C': np.random.choice(['Low','Medium','High'],N).tolist(),
    'D': np.random.normal(100,10,size=(N)).tolist()})
#reindex the DataFrame
df_reindexed = df.reindex(index=[0,2,5], columns=['A','C','B'])
print(df_reindexed)
```

Output

```
           A     C     B
0 2016-01-01   Low   NaN
2 2016-01-03  High   NaN
5 2016-01-06   Low   NaN
```

Reindex to Align with Other Objects

Example

```
import pandas as pd
import numpy as np
df1 = pd.DataFrame( np.random.randn (10,3), columns=['col1', 'col2', 'col3'])
df2 = pd.DataFrame( np.random.randn (7,3), columns=['col1', 'col2', 'col3'])
df1 = df1.reindex_like(df2)
print (df1)
```

Output

	col1	col2	col3
0	-2.467652	-1.211687	-0.391761
1	-0.287396	0.522350	0.562512
2	-0.255409	-0.483250	1.866258
3	-1.150467	-0.646493	-0.222462
4	0.152768	-2.056643	1.877233
5	-1.155997	1.528719	-1.343719
6	-1.015606	-1.245936	-0.295275

Filling while Reindexing

Reindex() takes an optional parameter method which is a filling method with values as follows:

pad/ffill	Fill values forward
bfill/backfill	Fill values backward
nearest	Fill from the nearest index values

Filling while Reindexing – Example

Example

```
import pandas as pd
import numpy as np
df1 = pd.DataFrame( np.random.randn (10,3), columns=['col1', 'col2', 'col3’])
df2 = pd.DataFrame( np.random.randn (7,3), columns=['col1', 'col2', 'col3’])
# Padding NAN’s
print (df2.reindex_like(df1))
#Now fill the NAN’s with preceding value
print("Data Frame with Forward Fill:")
print(df2.reindex_like(df1,method='ffill'))
```

Output

```
       col1      col2      col3
0  1.267556 -0.437309 -0.303115
1  2.079728 -0.085903  0.380246
2  0.458929 -0.197438 -0.665476
3  1.330227 -0.856160 -0.699978
4  0.047595 -1.566915  0.700972
5  0.452587 -1.351010 -0.920064
6 -0.138627 -1.360132 -0.471695
7       NaN       NaN       NaN
8       NaN       NaN       NaN
9       NaN       NaN       NaN
```

```
Data Frame with Forward Fill:
       col1      col2      col3
0  1.267556 -0.437309 -0.303115
1  2.079728 -0.085903  0.380246
2  0.458929 -0.197438 -0.665476
3  1.330227 -0.856160 -0.699978
4  0.047595 -1.566915  0.700972
5  0.452587 -1.351010 -0.920064
6 -0.138627 -1.360132 -0.471695
7 -0.138627 -1.360132 -0.471695
8 -0.138627 -1.360132 -0.471695
9 -0.138627 -1.360132 -0.471695
```

Limits on Filling while Reindexing

- The limit argument provides additional control over filling while reindexing
- Limit specifies the maximum count of consecutive matches

Example

```
import pandas as pd
import numpy as np
df1 = pd.DataFrame( np.random.randn (10,3), columns=['col1', 'col2', 'col3’])
df2 = pd.DataFrame( np.random.randn (7,3), columns=['col1', 'col2', 'col3’])
# Padding NAN’s
print (df2.reindex_like(df1))
#Now fill the NAN’s with preceding value
print("Data Frame with Forward Fill:")
print(df2.reindex_like(df1,method='ffill’, limit=1))
```

Limits on Filling while Reindexing (Contd.)

Output

```
       col1      col2      col3
0  1.222683  2.293787 -0.102225
1 -1.542144 -0.774326 -0.452945
2 -2.024487  0.287117 -0.894737
3 -1.219979  1.112291 -0.081755
4 -0.758945  0.213186  0.150298
5 -0.773376  0.476354 -0.784072
6 -0.969162 -0.290719  1.935481
7       NaN       NaN       NaN
8       NaN       NaN       NaN
9       NaN       NaN       NaN
Data Frame with Forward Fill:
       col1      col2      col3
0  1.222683  2.293787 -0.102225
1 -1.542144 -0.774326 -0.452945
2 -2.024487  0.287117 -0.894737
3 -1.219979  1.112291 -0.081755
4 -0.758945  0.213186  0.150298
5 -0.773376  0.476354 -0.784072
6 -0.969162 -0.290719  1.935481
7 -0.969162 -0.290719  1.935481
8       NaN       NaN       NaN
9       NaN       NaN       NaN
```

Renaming

The rename() method allows you to relabel an axis based on some mapping or an arbitrary function.

Example

```
import pandas as pd
import numpy as np
df = pd.DataFrame( np.random.randn (6,3), columns=['col1', 'col2', 'col3’])
print (df)
print ("After renaming the rows and columns:")
print (df.rename(columns={'col1' : 'c1', 'col2' : 'c2' },
                    index = {0 : 'apple', 1 : 'banana', 2 : 'durian' }))
```

Renaming (Contd.)

Output

```
        col1      col2      col3
0  2.245667  0.445119 -1.350078
1 -0.967365 -1.641705 -0.404663
2  0.223049 -1.189413  1.057567
3 -1.199742  0.806714  0.559279
4 -0.177894  0.154729  0.935000
5 -0.507553 -1.672507  1.742449
After renaming the rows and columns:
              c1        c2      col3
apple   2.245667  0.445119 -1.350078
banana -0.967365 -1.641705 -0.404663
durian  0.223049 -1.189413  1.057567
3      -1.199742  0.806714  0.559279
4      -0.177894  0.154729  0.935000
5      -0.507553 -1.672507  1.742449
```

The rename() method provides an inplace named parameter, which by default is False and copies the underlying data

Pass inplace = True to rename the data in place

I/O Tools

I/O Tools

The Pandas I/O API is a set of top level reader functions accessed like pd.read_csv() that generally return a Pandas object

The two workhorse functions for reading text files (or the flat files) are read_csv() and read_table()

They both use the same parsing code to intelligently convert tabular data into a DataFrame object.

```
pandas.read_csv(filepath_or_buffer, sep=‘ , ‘, delimiter=None,
header=‘infer’, names=None, index_col=None, usecols=None,
encoding=None, nrows=Nonena_values=None, keep_default_na=True,
na_filter=True)
```

I/O Tools (Contd.)

Here is how the **CSV** file data looks like -

```
S.No,Name,Age,City,Salary
1,Tom,28,Toronto,20000
2,Lee,32,HongKong,3000
3,Steven,43,Bay Area,8300
4,Ram,38,Hyderabad,3900
```

Save this data as **temp.csv** and conduct operations on it.


```
S.No,Name,Age,City,Salary
1,Tom,28,Toronto,20000
2,Lee,32,HongKong,3000
3,Steven,43,Bay Area,8300
4,Ram,38,Hyderabad,3900
```

Read a .csv File

Read.csv reads data from the csv files and creates a DataFrame object.

```
import pandas as pd
df = pd.read_csv(“temp.csv”)
print (df)
```

Output

	S.No	Name	Age	City	Salary
0	1	Tom	28	Toronto	20000
1	2	Lee	32	HongKong	3000
2	3	Steven	43	Bay Area	8300
3	4	Ram	38	Hyderabad	3900

Mark Column as Index Column

This specifies a column in the csv file to customize the index using index_col.

```
import pandas as pd
df = pd.read_csv("temp.csv", index_col=['S.No'])
print (df)
```

Output

S.No	Name	Age	City	Salary
1	Tom	28	Toronto	20000
2	Lee	32	HongKong	3000
3	Steven	43	Bay Area	8300
4	Ram	38	Hyderabad	3900

Column Type Conversion

dtype of the columns can be passed as a dict.

```
import pandas as pd
df = pd.read_csv("temp.csv", dtype={'Salary' : np.float64})
print (df.dtypes)
```

Output

```
S.No         int64
Name        object
Age          int64
City        object
Salary     float64
dtype: object
```

By default, the dtype of the Salary column is int, but the result shows it as float because we have explicitly casted the type.

	S.No	Name	Age	City	Salary
0	1	Tom	28	Toronto	20000.0
1	2	Lee	32	HongKong	3000.0
2	3	Steven	43	Bay Area	8300.0
3	4	Ram	38	Hyderabad	3900.0

Assign Header Names to Columns

Specify the names of the header using the names argument.

```
import pandas as pd
df = pd.read_csv("temp.csv", names=[ 'a', 'b', 'c', 'd', 'e' ])
print (df)
```

Output

	a	b	c	d	e
0	S.No	Name	Age	City	Salary
1	1	Tom	28	Toronto	20000
2	2	Lee	32	HongKong	3000
3	3	Steven	43	Bay Area	8300
4	4	Ram	38	Hyderabad	3900

Skip Rows in a DataFrame

Skiprows skips the number of rows specified.

```
import pandas as pd
df = pd.read_csv("temp.csv", skiprows=2)
print (df)
```

Output

	2	Lee	32	HongKong	3000
0	3	Steven	43	Bay Area	8300
1	4	Ram	38	Hyderabad	3900