11_Pandas

October 31, 2019

1 1. What is Pandas

Pandas is a Python package providing fast, exible, and expressive data structures. It aims to be the fundamental high-level building block for doing practical, real world data analysis in Python (http://pandas.pydata.org/)

Pandas builds on top of Numpy to ease managing heterogeneous data sets.

1.1 1.1 Data Handled by Pandas

Pandas is well suited for many different kinds of data:

- Tabular data with heterogeneously-typed columns (comparable to EXCEL, R or relational Databases)
- Time series data
- Matrix data(homogeneously typed or heterogeneous) with row and column labels
- Any other form of observational / statistical data sets.

1.2 1.2 Feature Overview

- Easy handling of missing data (represented as NaN)
- Size mutability: columns can be inserted and deleted from DataFrame and higher dimensional objects
- Automatic and explicit data alignment
- Powerful, flexible group by functionality to perform split-apply-combine operations on data sets, for both ag- gregating and transforming data
- Intelligent label-based slicing, fancy indexing, and subsetting of large data sets
- Intuitive merging and joining data sets
- Flexible reshaping and pivoting of data sets
- Hierarchical labeling of axes (possible to have multiple labels per tick)
- Robust IO tools for loading and storing data
- Time series-specific functionality

2 2. Pandas Data Structures

Pandas is build around two data structures

- `Series` represent 1 dimensional datasets as subclass of Numpy's ndarray

- `DataFrame` represent 2 dimensional data sets as list of `Series`

For all data structures, labels/indices can be defined per row and column.
Data alignment is intrinsict, i.e. the link between labels and data will not be broken.
Series: * Homogeneous data * Size Immutable * Values of Data Mutable
Data Frames: * Heterogeneous data * Size Mutable * Data Mutable

2.1 2.1. Series

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:

```
Series(data, index=index)
```

`data` may be a dict, a `numpy.ndarray` or a sclar value
A series can be created using various inputs like

- Array
- Dict
- Scalar value or constant

2.1.1 2.1.1 Creating a series from ndarray

[1]:
```
#import the pandas library and aliasing as pd
import pandas as pd
import numpy as np
data = np.array(['a','b','c','d'])
s = pd.Series(data)
print (s)
```

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

[2]:
```
data = np.array(['a','b','c','d'])
s = pd.Series(data, index=[100,101,102,103])
print (s)
```

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

2.1.2 2.1.2 Creating a Series from dict

A dict can be passed as input and if no index is specified, then the dictionary keys are taken in a sorted order to construct index. If index is passed, the values in data corresponding to the labels in the index will be pulled out.

[3]:

```
data = {'a' : 0, 'b' : 1, 'c' : 2}
s = pd.Series(data)
print (s)
```

```
a    0
b    1
c    2
dtype: int64
```

Dictionary keys are used to construct index.

[4]:

```
data1 = {'a' : 0., 'b' : 1., 'c' : 2.}
s1 = pd.Series(data1,index=['b','c','d','a'])
print (s1)
```

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

Index order is persisted and the missing element is filled with NaN (Not a Number).

2.1.3 2.1.3 Creating a Series from Scalar

If data is a scalar value, an index must be provided. The value will be repeated to match the length of index

[5]:

```
s = pd.Series(5, index=[0, 1, 2, 3])
print (s)
```

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

[6]:

```
#show the index
s.index
```

[6]:

```
Int64Index([0, 1, 2, 3], dtype='int64')
```

[7]:

```
#show the value
s.values
```

[7]:

```
array([5, 5, 5, 5])
```

2.1.4 2.1.4 Series Indexing

Accessing elements in a series can be either done via the number or the index

[8]:
```
s = pd.Series([1,2,3,4,5], index = ['a','b','c','d','e'])

#retrieve a single element
s['c']
```

[8]:
```
3
```

[9]:
```
#retrieve multiple elements
s[['a','c','d']]
```

[9]:
```
a    1
c    3
d    4
dtype: int64
```

[10]:
```
s.at['a'] # value at index 'a'
```

[10]:
```
1
```

2.2 2.2. DataFrame: a Series of Series

The pandas DataFrame is a 2 dimensional labeled data structure with columns of potentially different types. Similar to * a spreadsheet * relational database table * a dictionary of series

Creating DataFrame's

A pandas DataFrame can be created using various inputs like

- Lists
- Dict
- Series
- Numpy ndarrays
- Another DataFrame

2.2.1 2.2.1 Create a DataFrame from Lists

[11]:
```
import pandas as pd
data = [1,2,3,4,5]
df = pd.DataFrame(data)
df
```

[11]:
```
   0
0  1
1  2
2  3
3  4
4  5
```

[12]:
```
data = [['Ramesh',10],['Himesh',12],['Suresh',13]]
df = pd.DataFrame(data,columns=['Name','Age'])
```

```
df
```

[12]:

```
     Name  Age
0  Ramesh   10
1  Himesh   12
2  Suresh   13
```

[13]:

```
data = [['Ramesh',10],['Himesh',12],['Suesh',13]]
df = pd.DataFrame(data,columns=['Name','Age'],
                  dtype=float)
df
```

[13]:

```
     Name   Age
0  Ramesh  10.0
1  Himesh  12.0
2   Suesh  13.0
```

2.2.2 2.2.2 Create a DataFrame from Dict of ndarrays / Lists

All the ndarrays must be of same length. If index is passed, then the length of the index should equal to the length of the arrays.

If no index is passed, then by default, index will be range(n), where n is the array length.

[14]:

```
data = {'Name':['Nitesh','Ramesh','Rajesh','Nilesh'],
        'Age':[28,34,29,45]}
df = pd.DataFrame(data)
df
```

[14]:

```
     Name  Age
0  Nitesh   28
1  Ramesh   34
2  Rajesh   29
3  Nilesh   45
```

[15]:

```
data = {'Name':['Ramesh', 'Rajesh', 'Nitesh', 'Nilesh'],
        'Age':[28,34,29,42]}
df = pd.DataFrame(data, index=['rank1','rank2','rank3','rank4'])
df
```

[15]:

```
         Name  Age
rank1  Ramesh   28
rank2  Rajesh   34
rank3  Nitesh   29
rank4  Nilesh   42
```

2.2.3 2.2.3 Create a DataFrame from List of Dicts

List of Dictionaries can be passed as input data to create a DataFrame. The dictionary keys are by default taken as column names.

[16]:
```
data = [{'a': 1, 'b': 2},{'a': 5, 'b': 10, 'c': 20}]
df = pd.DataFrame(data)
df
```

[16]:
```
   a   b     c
0  1   2   NaN
1  5  10  20.0
```

[17]:
```
data = [{'a':1, 'b':2},{'a':5, 'b':10, 'c':20}]
df = pd.DataFrame(data, index=['first','second']) # passing row indices
df
```

[17]:
```
        a   b     c
first   1   2   NaN
second  5  10  20.0
```

[18]:
```
data = [{'a':1, 'b':2},{'a':5, 'b':10, 'c':20}]

#With two column indices, values same as dictionary keys
df1 = pd.DataFrame(data, index=['first', 'second'],
                   columns=['a', 'b'])

#With two column indices with one index with other name
df2 = pd.DataFrame(data, index=['first', 'second'],
                   columns=['a', 'b1'])

print (df1)
print()
print (df2)
```

```
        a   b
first   1   2
second  5  10

        a  b1
first   1 NaN
second  5 NaN
```

2.2.4 2.2.4 Create a DataFrame from Dict of Series

Dictionary of Series can be passed to form a DataFrame. The resultant index is the union of all the series indexes passed.

[19]:
```
d = {'one' : pd.Series([1, 2, 3], index=['a','b','c']),
      'two' : pd.Series([1, 2, 3, 4], index=['a','b','c','d'])}
df = pd.DataFrame(d)
df
```

[19]:
```
   one  two
a  1.0    1
b  2.0    2
```

```
c  3.0    3
d  NaN    4
```

2.2.5 2.2.5 Column selection, addition, deletion

[20]:

```
d = {'one' : pd.Series([1, 2, 3],
                        index=['a', 'b', 'c']),
      'two' : pd.Series([1, 2, 3, 4],
                        index=['a', 'b', 'c', 'd'])}
df = pd.DataFrame(d)
df['one']
```

[20]:

```
a    1.0
b    2.0
c    3.0
d    NaN
Name: one, dtype: float64
```

[21]:

```
# Adding a new column to an existing DF object
# with column label by passing new series
print ("Adding a new column by passing as Series:")
df['three']=pd.Series([10,20,30],
                      index=['a','b','c'])
print (df)
```

```
Adding a new column by passing as Series:
   one  two  three
a  1.0    1   10.0
b  2.0    2   20.0
c  3.0    3   30.0
d  NaN    4    NaN
```

[22]:

```
# Adding a new column using the existing columns
df['four']=df['one']+df['three']
print (df)
```

```
   one  two  three  four
a  1.0    1   10.0  11.0
b  2.0    2   20.0  22.0
c  3.0    3   30.0  33.0
d  NaN    4    NaN   NaN
```

[23]:

```
# deleting a column using del function

del df['one']
df
```

[23]:

```
   two  three  four
a    1   10.0  11.0
b    2   20.0  22.0
c    3   30.0  33.0
d    4    NaN   NaN
```

[24]:

```
# Deleting another column using POP function
df.pop('two')
df
```

[24]:

```
   three  four
a   10.0  11.0
b   20.0  22.0
c   30.0  33.0
d    NaN   NaN
```

2.2.6 2.2.5 Row Selection, Addition, and Deletion

Selection by Row Label

Rows can be selected by passing row label to a loc function.

[25]:

```
#import pandas as pd
d = {'one' : pd.Series([1, 2, 3],
                      index=['a', 'b', 'c']),
     'two' : pd.Series([1, 2, 3, 4],
                    index=['a', 'b', 'c', 'd'])}

df = pd.DataFrame(d)
print(df)
print("\n Accessing row having label 'b':")
print (df.loc['b'])
```

```
   one  two
a  1.0    1
b  2.0    2
c  3.0    3
d  NaN    4

 Accessing row having label 'b':
one    2.0
two    2.0
Name: b, dtype: float64
```

Selection by integer location

Rows can be selected by passing integer location to an iloc function.

[26]:

```
print (df.iloc[1])
```

```
one    2.0
```

```
two    2.0
Name: b, dtype: float64
```

Slice Rows

Multiple rows can be selected using ' : ' operator.

[27]:

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

```
   one  two
c  3.0    3
d  NaN    4
```

Addition of Rows

Add new rows to a DataFrame using the append function. This function will append the rows at the end.

[28]:

```
df1 = pd.DataFrame([[1, 2], [3, 4]], columns=['a','b'])
df2 = pd.DataFrame([[5, 6], [7, 8]], columns=['a','b'])

df3 = df1.append(df2)
print (df3)
```

```
   a  b
0  1  2
1  3  4
0  5  6
1  7  8
```

Deletion of Rows

Use index label to delete or drop rows from a DataFrame. If label is duplicated, then multiple rows will be dropped.

[29]:

```
# Drop rows with label 0
df = df3.drop(0)

print (df)
```

```
   a  b
1  3  4
1  7  8
```

2.3 3 Basic Functionality

[30]:

```
#import pandas as pd

#Create a Dictionary of series
d={'Name':pd.Series(['Ramesh','Suresh','Rajesh']),
   'Age':pd.Series([25,26,25]),
   'Rating':pd.Series([4.23,3.24,3.98])}
```

[31]:

```
#Create a DataFrame
df = pd.DataFrame(d)
df
```

[31]:

```
      Name  Age  Rating
0  Ramesh   25    4.23
1  Suresh   26    3.24
2  Rajesh   25    3.98
```

T (Transpose)

Returns the transpose of the DataFrame. The rows and columns will interchange.

[32]:

```
print ("The transpose of the data series is:", )
print (df.T)
```

```
The transpose of the data series is:
             0       1       2
Name    Ramesh  Suresh  Rajesh
Age         25      26      25
Rating    4.23    3.24    3.98
```

axes

Returns the list of row axis labels and column axis labels.

[33]:

```
print ("Row axis labels and column axis labels are:")
print (df.axes)
```

```
Row axis labels and column axis labels are:
[RangeIndex(start=0, stop=3, step=1), Index(['Name', 'Age', 'Rating'],
dtype='object')]
```

dtypes

Returns the data type of each column.

[34]:

```
print ("The data types of each column are:")
print (df.dtypes)
```

```
The data types of each column are:
Name       object
Age         int64
Rating    float64
dtype: object
```

ndim

Returns the number of dimensions of the object. By definition, DataFrame is a 2D object.

[35]:

```
print ("The dimension of the object is:", df.ndim)
```

```
The dimension of the object is: 2
```

shape

Returns a tuple (a,b), where a represents the number of rows and b represents the number of columns.

[36]:

```
print ("The shape of the object is:",df.shape)
```

```
The shape of the object is: (3, 3)
```

size

Returns the number of elements in the DataFrame.

[37]:

```
print ("The total no. of elements:",df.size )
```

```
The total no. of elements: 9
```

values

Returns the actual data in the DataFrame as an NDarray.

[38]:

```
print ("The actual data in our data frame is:")
print (df.values)
```

```
The actual data in our data frame is:
[['Ramesh' 25 4.23]
 ['Suresh' 26 3.24]
 ['Rajesh' 25 3.98]]
```

Head & Tail

To view a small sample of a DataFrame object, use the head() and tail() methods. - head() returns the first n rows (observe the index values). - tail() returns the last few rows

The default number of elements to display is 5, but you may pass a custom number.

[39]:

```
# first few rows of the data frame
print (df.head())
```

```
     Name  Age  Rating
0  Ramesh   25    4.23
1  Suresh   26    3.24
2  Rajesh   25    3.98
```

[40]:

```
# first 2 rows of the data frame
print (df.head(2))
```

```
     Name  Age  Rating
0  Ramesh   25    4.23
1  Suresh   26    3.24
```

[41]:

```
# last few rows of the data frame
print (df.tail())
```

```
     Name  Age  Rating
0  Ramesh   25    4.23
1  Suresh   26    3.24
2  Rajesh   25    3.98
```

3 4. Descriptive Statistics

Descriptive Statistics sumarizes the underlying distribution of data values through statistical values like mean, variance etc.

3.0.1 Basic Functions

Function	Description
count	Number of non-null observations
sum	Sum of values
mean	Mean of values
mad	Mean absolute deviation
median	Arithmetic median of values
min	Minimum
max	Maximum
mode	Mode
abs	Absolute Value
prod	Product of values
std	Unbiased standard deviation
var	Unbiased variance
skew	Unbiased skewness (3rd moment)
kurt	Unbiased kurtosis (4th moment)
quantile	Sample quantile (value at %)
cumsum	Cumulative sum
cumprod	Cumulative product
cummax	Cumulative maximum
cummin	Cumulative minimum

3.0.2 4.1 sum()

Returns the sum of the values for the requested axis. By default, axis is index (axis=0).

[42]:
```
df
```

[42]:
```
      Name  Age  Rating
0   Ramesh   25    4.23
1   Suresh   26    3.24
2   Rajesh   25    3.98
```

[43]:
```
print (df.sum()) # axis = 0
```

```
Name       RameshSureshRajesh
Age                        76
Rating                  11.45
dtype: object
```

Each individual column is added individually (Strings are appended).

[44]:
```
print (df.sum(1)) # axis = 1, adds columns
```

```
0    29.23
1    29.24
2    28.98
dtype: float64
```

3.0.3 4.2 mean()

Returns the average value

[45]:
```
print (df.mean())
```

```
Age       25.333333
Rating     3.816667
dtype: float64
```

3.0.4 4.3 std()

Returns the Bressel standard deviation of the numerical columns.

[46]:
```
print (df.std())
```

```
Age       0.577350
Rating    0.514814
dtype: float64
```

[47]:
```
df.min()
```

[47]:
```
Name       Rajesh
Age            25
Rating       3.24
```

```
dtype: object
```

[48]:
```
df.max()
```

[48]:
```
Name       Suresh
Age            26
Rating       4.23
dtype: object
```

3.0.5 4.4 Summarizing Data

The describe() function computes a summary of statistics pertaining to the DataFrame columns.

[49]:
```
print(df)
print (df.describe())
```

```
     Name  Age  Rating
0  Ramesh   25    4.23
1  Suresh   26    3.24
2  Rajesh   25    3.98

             Age    Rating
count   3.000000  3.000000
mean   25.333333  3.816667
std     0.577350  0.514814
min    25.000000  3.240000
25%    25.000000  3.610000
50%    25.000000  3.980000
75%    25.500000  4.105000
max    26.000000  4.230000
```

This function gives the mean, std and IQR values. And, function excludes the character columns and given summary about numeric columns.

'include' is the argument which is used to pass necessary information regarding what columns need to be considered for summarizing. Takes the list of values; by default, 'number'.

- object Summarizes String columns
- number Summarizes Numeric columns
- all Summarizes all columns together (Should not pass it as a list value)

[50]:
```
print (df.describe(include=['object']))
```

```
          Name
count        3
unique       3
top     Ramesh
freq         1
```

[73]:
```
print (df.describe(include='all'))
```

3.1 5. Input/Output 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. * read_csv * read_excel * read_hdf * read_sql * read_json * read_msgpack (experimental) * read_html * read_gbq (experimental) * read_stata * read_clipboard * read_pickle

The corresponding writer functions are object methods that are accessed like df.to_csv(). * to_csv * to_excel * to_hdf * to_sql * to_json * to_msgpack (experimental) * to_html * to_gbq (experimental) * to_stata * to_clipboard * to_pickle

3.1.1 5.1 Loading the Weather Data from the CSV

In this example we load the weather datafrom the data directory ("data_data.csv")

[54]:
```
#! executes a shell command
#!ls data
```

[55]:
```
df = pd.read_csv("data/weather_data.csv")
print (df)
```

```
   Day   outlook  temperature  humidity  windy play
0    1     sunny           85        85  False   no
1    2     sunny           80        90   True   no
2    3  overcast           83        86  False  yes
3    4     rainy           70        96  False  yes
4    5     rainy           68        80  False  yes
5    6     rainy           65        70   True   no
6    7  overcast           64        65   True  yes
```

[56]:
```
pd.read_csv?
```

[57]:
```
df.to_csv('temp1.csv')
```

3.1.2 5.2 Excel data

[ ]:
```
#use help to see the parameters
#pd.read_excel?
```

[58]:
```
import pandas as pd
df_out = pd.DataFrame([('Ramesh', 25), ('Rajesh', 20), ('Kamesh', 35)],␣
 ↪columns=['Name', 'Age'])
```

[59]:
```
df_out
```

[59]:
```
     Name  Age
0  Ramesh   25
1  Rajesh   20
2  Kamesh   35
```

[60]:
```
df_out.to_excel('tmp.xlsx', index=False)
```

[61]:
```
pd.read_excel('tmp.xlsx')
```

[61]:

```
     Name  Age
0  Ramesh   25
1  Rajesh   20
2  Kamesh   35
```

3.1.3 5.3 Sqlite data

[62]:

```
import sqlite3 as lite
import sys

con = lite.connect('employee.db')
```

[63]:

```
with con:
    cur = con.cursor()
    cur.execute("CREATE TABLE IF NOT EXISTS csdept(eid INTEGER PRIMARY KEY,␣
 ↪ename TEXT, esalary INT)")
    cur.execute("INSERT INTO csdept VALUES(101,'Ramesh',25000)")
    cur.execute("INSERT INTO csdept VALUES(102,'Suresh', 6500)")
    cur.execute("INSERT INTO csdept VALUES(103,'Naresh', 45000)")
    cur.execute("INSERT INTO csdept VALUES(104,'Mahesh', 60000)")
```

[64]:

```
q="select * from csdept"
df = pd.read_sql_query(q,con)
print(df)
```

```
   eid   ename  esalary
0  101  Ramesh    25000
1  102  Suresh     6500
2  103  Naresh    45000
3  104  Mahesh    60000
```

[65]:

```
query = "SELECT ename FROM csdept WHERE esalary > 20000;"

df = pd.read_sql_query(query,con)

for i in df['ename']:
    print(i)
```

```
Ramesh
Naresh
Mahesh
```

[66]:

```
con.close()
```

3.1.4 5.4 JSON Data

JSON (JavaScript Object Notation) is a popular data format used for representing structured data. It's common to transmit and receive data between a server and web application in JSON format.

JSON is built on two structures:

- A collection of name/value pairs. This is realized as an object, record, dictionary, hash table, keyed list, or associative array.
- An ordered list of values. This is realized as an array, vector, list, or sequence.

```
[67]: # creating a data frame
      df = pd.DataFrame([['a', 'b'], ['c', 'd']],
                        index=['row 1', 'row 2'],
                        columns=['col 1', 'col 2'])
```

```
[68]: df
```

```
[68]:       col 1 col 2
      row 1     a     b
      row 2     c     d
```

```
[69]: df.to_json?
```

Convert the object to a JSON string:

```
[70]: df.to_json(orient='split')
      '{"columns":["col 1","col 2"], "index":["row 1","row 2"], "data":
       ↪[["a","b"],["c","d"]]}'
```

```
[70]: '{"columns":["col 1","col 2"], "index":["row 1","row 2"],
      "data":[["a","b"],["c","d"]]}'
```

Encoding/decoding a Dataframe using `'split'` formatted JSON

```
[71]: pd.read_json?
```

```
[72]: pd.read_json(_,orient='split') #input function
```

```
[72]:       col 1 col 2
      row 1     a     b
      row 2     c     d
```

3.2 Resources:

- Book : Python for Data Analysis
- SQLite Tutorial :
  - http://www.sqlitetutorial.net,
  - https://www.sqlite.org/lang.html