Algorithms and Data Structures with Python

Pandas - intro

Pandas Series – how to

Assign it all together into a pandas Series

(don’t forget to define data type)

Original data (ndarray)

```
s = pd.Series(data, \
              index=index,\
              name = col_name,\
              dtype = 'int32')
```

index labels (row names)

column labels (name)

Data types

More about Pandas Series

Pandas series are data structures that hold retrievable **index** and **values** of certain **type**

s - Series

Index	val
0	5
1	1
2	3
3	8
4	0
5	4
6	2
7	9
8	6
9	7

```
s.values
array([5, 1, 3, 8, 0, 4, 2, 9, 6, 7])
```

```
s.index
RangeIndex(start=0, stop=10, step=1)
```

```
s.dtype
dtype('int32')
```

Pandas Series from Dictionaries

- Data series can be directly assigned from dictionaries
- Index labels are defined by dictionary keys
- Column names and type can be re-assigned

dict - Dictionary (3 elements)

Key	Type	Size	Value
Kanchenjunga	int	1	8586
everest	int	1	8848
k2	int	1	8611

```
s = pd.Series(dict,\
              name = col_name,\
              dtype ='float')
```

s - Series

Index	val
everest	8848
k2	8611
Kanche...	8586

Pandas Series from Dictionaries

- What if there are more rows than dictionary data inserts?
- pd.Series leaves ‘nan’ vacancies

dict - Dictionary (3 elements)

Key	Type	Size	Value
Kanchenjunga	int	1	8586
everest	int	1	8848
k2	int	1	8611

```
index_name = ['everest','k2', 'Kanchenjunga','mount Elbrus']
s = pd.Series(dict,\
              index = index_name,\
              name = col_name,\
              dtype ='float')
```

s - Series

Index	val
everest	8848
k2	8611
Kanchenjunga	8586
mount Elbrus	nan

Basic Pandas Series Operations

pd.Series slicing is similar to values retrieval from dictionary

```
In [15]: s[:3]
Out[15]:
a    1
b    0
c    9
Name: val, dtype: int32
```

```
In [31]: s[[3,5,9]]
Out[31]:
d    2
f    8
j    5
Name: val, dtype: int32
```

s - Series

Index	val
a	1
b	0
c	9
d	2
e	3
f	8
g	6
h	4
i	7
j	5

pd.Series enables dictionary-like slicing

```
In [18]: s['j']
Out[18]: 5
```

```
In [19]: 'e' in s
Out[19]: True
```

Basic Pandas Series Operations

pd.Series enables basic math operations

Series Mean

```
In [22]: s.mean()
Out[22]: 4.5
```

Series standard deviation

```
In [33]: s.std()
Out[33]: 3.0276503540974917
```

Scalar / cross product multiplication

```
In [39]: s*5
Out[39]:
a     5
b     0
c    45
d    10
e    15
f    40
g    30
```

```
In [43]: s*s
Out[43]:
a     1
b     0
c    81
d     4
e     9
f    64
g    36
```

s - Series

Index	val
a	1
b	0
c	9
d	2
e	3
f	8
g	6
h	4
i	7
j	5

Basic Pandas Series Operations

data series filtration
enables slicing by criterion (condition) using
a boolean vector as series index “slicer”

```
In [28]: s[s>s.mean()]
Out[28]:
c    9
f    8
g    6
i    7
j    5
Name: val, dtype: int32
```

s - Series

Index	val
a	1
b	0
c	9
d	2
e	3
f	8
g	6
h	4
i	7
j	5

Pandas Dataframes

DataFrame is a **2-dimensional labeled data structure** with columns of potentially different types

- **Most commonly used** data structure in pandas
- Has **index** and **column** labels
- Accepts the following **inputs**:
- Dict of 1D ndarrays, lists, dicts, or Series
- 2-D numpy.ndarray
- Series
- Another DataFrame

Pandas Dataframes – how to

1. Create dictionary with 2D list entries + column labels

```
d = {'c1' : [1., 2., 3., 4.],\
     'c2' : [4., 3., 2., 1.]}
```

d - Dictionary (2 elements)

Key	Type	Size	Value
c1	list	4	[1.0, 2.0, 3.0, 4.0]
c2	list	4	[4.0, 3.0, 2.0, 1.0]

2. Plug it into **pd.DataFrame()** function

```
df = pd.DataFrame(d,index = ['a','b','c','d'])
```

Don't forget row labels!

df - DataFrame

Index	c1	c2
a	1	4
b	2	3
c	3	2
d	4	1

Pandas Dataframes – how to

1. Create 2D list

```
ls = [[1, 2, 3, 4],\
      [4, 3, 2, 1]]
```

ls - List (2 elements)

Index	Type	Size	Value
0	list	4	[1, 2, 3, 4]
1	list	4	[4, 3, 2, 1]

2. Plug it into **pd.DataFrame()** function

```
df_ls = pd.DataFrame(ls,\
                     index = ['c1','c2'],\
                     columns = ['a','b','c','d'])
```

df_ls - DataFrame

Index	a	b	c	d
c1	1	2	3	4
c2	4	3	2	1

Basic operations with Pandas Dataframes

df_ls - DataFrame

Index	a	b	c	d
c1	1	2	3	4
c2	4	3	2	1

```
df_ls = df_ls.transpose()
```

df_ls - DataFrame

Index	c1	c2
a	1	4
b	2	3
c	3	2
d	4	1

Column Assignment with Pandas Dataframes

df_ls - DataFrame

Index	c1	c2
a	1	4
b	2	3
c	3	2
d	4	1

```
df_ls['c3'] = df_ls['c1']+df_ls['c2']
```

df_ls - DataFrame

Index	c1	c2	c3
a	1	4	5
b	2	3	5
c	3	2	5
d	4	1	5

Pandas Dataframes Selection by label

df_ls - DataFrame

Index	c1	c2	c3
a	1	4	5
b	2	3	5
c	3	2	5
d	4	1	5

df.loc[ ] enables **row selection by index label**
(index is a **list** of strings)

```
df_ls.loc[['a','d']]
```

⇩

	c1	c2	c3
a	1	4	5
d	4	1	5

Pandas Dataframes Selection by label

Index	c1	c2	c3
a	1	4	5
b	2	3	5
c	3	2	5
d	4	1	5

df.loc[:,[ ]] enables **column slicing by label**
(similar to list slicing, column label is list)

```
df_ls.loc[:,['c1','c3']]
```

⇩

	c1	c3
a	1	5
b	2	5
c	3	5
d	4	5

Pandas Dataframes selection by position

Index	c1	c2	c3
a	1	4	5
b	2	3	5
c	3	2	5
d	4	1	5

df.iloc[ ] enables **row selection by position**
(index is a **list** of integers)

```
df_ls.iloc[[0,2]]
```

⇩

```
   c1  c2  c3
a   1   4   5
c   3   2   5
```

Pandas Dataframes selection by position

Index	c1	c2	c3
a	1	4	5
b	2	3	5
c	3	2	5
d	4	1	5

df.iloc[: , [ ]] enables **column selection by position**
(column index is a **list** of integers)

```
df_ls.iloc[:,[0,2]]
```

⇩

```
   c1  c3
a   1   5
b   2   5
c   3   5
d   4   5
```