

Handout 10

Introduction to Pandas data structures

Pandas library provides efficient operations on multidimensional arrays of heterogeneous data with attached row and column labels. Built on top of NumPy.

Described in <https://pandas.pydata.org/pandas-docs/stable/index.html>

```
import pandas
import pandas as pd
```

Main data structures are Series and DataFrame.

SERIES

One-dimensional array of *indexed* data. Indexing is not necessarily numeric; index may contain duplicate values.

<https://pandas.pydata.org/pandas-docs/stable/dsintro.html#series>

```
>>> s = pd.Series ([ 7, 12, 3, 45])
>>> s # will show index on the left and element on the right
0     7
1    12
2     3
3    45
dtype: int64

>>> type(s)
pandas.core.series.Series

>>> s.values
array([ 7, 12,  3, 45], dtype=int64)

>>> s.get_values()
array([ 7, 12,  3, 45], dtype=int64)

>>> type(s.values)
numpy.ndarray

>>> s.index
RangeIndex(start=0, stop=4, step=1)
```

Can assign a non-numeric index:

```
>>> si = pd.Series([210, 31, 40, 15],
                    index =['Porcupine', 'Chipmunk', 'Squirrel', 'Goat'])
>>> si

Porcupine    210
Chipmunk      31
Squirrel      40
Goat          15
dtype: int64
```

```
>>> si['Goat']
15

>>> si[0] # can also use the numeric index!
210
```

Series created from a dictionary:

```
>>> d = {'d': 45, 'b': 33, 'c':12, 'a': 34}
>>> d
{'d': 45, 'b': 33, 'c': 12, 'a': 34}
>>> sd = pd.Series(d)
>>> sd
d    45
b    33
c    12
a    34
```

DATAFRAME

A two-dimensional array with labeled rows and columns.
Each column is of type Series.

CREATE DATAFRAME

There are a number of ways to create and initialize new DataFrames, for example from

- a file
- Python dictionaries, lists or tuples
- pd.Series, pd.DataFrame, NumPy data array

Example 1

```
>>> data
{'city': ['Boston', 'Paris', 'Lagos'],
 'area': [11700, 17174, 2706],
 'population': [4628910, 12405426, 21000000]}

>>> frame = pd.DataFrame(data)
>>> frame
   city  area  population
0  Boston  11700    4628910
1   Paris  17174   12405426
2   Lagos   2706   21000000

>>> frame = pd.DataFrame(data, columns = ['city', 'country', 'area', 'population'])
>>> frame
   city  country  area  population
0  Boston     NaN  11700    4628910
1   Paris     NaN  17174   12405426
2   Lagos     NaN   2706   21000000
```

```
>>> frame.columns
Index(['city', 'country', 'area', 'population'], dtype='object')

>>> frame.index
RangeIndex(start=0, stop=3, step=1)
```

- Selection of rows.

```
>>> frame.loc[1] # there will be more on loc and iloc attributes later
city          Paris
country       NaN
area          17174
population    12405426
Name: 1, dtype: object

>>> type(frame.loc[1])
pandas.core.series.Series
```

- Selection of columns. Each column is a pd.Series:

```
>>> frame['city']
0    Boston
1    Paris
2    Lagos
Name: city, dtype: object

>>> type(frame['city'])
pandas.core.series.Series
>>> frame.city
0    Boston
1    Paris
2    Lagos
Name: city, dtype: object
```

- Assigning column values

```
>>> frame.country = 'USA'
>>> frame
   city country  area  population
0  Boston   USA  11700    4628910
1  Paris   USA  17174    12405426
2  Lagos   USA   2706    21000000

>>> frame.country = ['USA', 'France', 'Nigeria']
>>> frame
   city country  area  population
0  Boston   USA  11700    4628910
1  Paris  France  17174    12405426
2  Lagos  Nigeria   2706    21000000
```

- Assign data

```
>>> frame.area = pd.Series([30000000, 5000000], index=[2,3])
>>> frame.area
0         NaN
1         NaN
2    30000000.0
```

```
Name: area, dtype: float64
>>> frame
   city country      area  population
0  Boston   USA      NaN    4628910
1   Paris  France      NaN    12405426
2   Lagos Nigeria  30000000.0    21000000
```

- Add a column:

```
>>> frame['mega'] = (frame['population'] >= 10**7)
>>> frame
   city country      area  population  mega
0  Boston   USA      NaN    4628910  False
1   Paris  France      NaN    12405426   True
2   Lagos Nigeria  30000000.0    21000000   True
```

- Add a row : done by appending a dictionary, a DataFrame or Series object

```
>>> frame = frame.append({'city': 'Yerevan', 'country': 'Armenia', 'population': 1075000},
                          ignore_index = True)
>>> frame
   city country      area  population  mega
0  Boston   USA      NaN    4628910   0.0
1   Paris  France      NaN    12405426   1.0
2   Lagos Nigeria  30000000.0    21000000   1.0
3  Yerevan  Armenia      NaN     1075000   NaN
```

Example 2

: suppose have the following content in weather.xlsx

Day	Description	High Temp	Low Temp	Precip	Humidity
1	PM Showers	50	51	50	77
2	PM Thundershowers	52	61	90	89
3	Mostly Sunny	45	60	0	55

```
>>> exdf = pd.read_excel("weather.xlsx") # also have pd.read_csv
>>> exdf
   Day  Description  High Temp  Low Temp  Precip  Humidity
0    1    PM Showers         50         51         50         77
1    2  PM Thundershowers         52         61         90         89
2    3    Mostly Sunny         45         60          0         55

>>> exdf.values
array([[1, 'PM Showers ', 50, 51, 50, 77],
       [2, 'PM Thundershowers', 52, 61, 90, 89],
       [3, 'Mostly Sunny', 45, 60, 0, 55]], dtype=object)

>>> type(exdf.values)
numpy.ndarray

>>> exdf.values.shape
(3, 6)
```

```
>>> exdf['Precip']
0    50
1    90
2     0
Name: Precip, dtype: int64
```

```
>>> exdf['Precip'][0]
50
```

INDEX

- Pandas Index objects hold axis labels and other metadata.
- Implemented as immutable ndarray implementing an ordered, sliceable set.
- Index can contain duplicates.

```
>>> exdf.columns
Index(['Day', 'Description', 'High Temp', 'Low Temp', 'Precip', 'Humidity'],
      dtype='object')
```

```
>>> exdf.index
RangeIndex(start=0, stop=3, step=1)
```

```
>>> exdf.index = ['Mon', 'Tue', 'Wed']
>>> exdf
```

	Day	Description	Low Temp	High Temp	Precip	Humidity
Mon	1	PM Showers	50	51	50	77
Tue	2	PM Thundershowers	52	61	90	89
Wed	3	Mostly Sunny	45	60	0	55

Renaming, Adding and removing rows/columns

**** Note - function descriptions below are not complete**

```
dataframe.rename( index = index-dict,
                  columns = column-dict,
                  inplace = True )
```

index/columns param is a dictionary of form **{old:new}**. Function replaces the **old** values in the **index/columns** with the **new**. **inplace** controls whether the original dataframe is changed.

```
dataframe.reindex( index=None,
                  columns=None,
                  fill_value=NaN )
```

Returns a reindexed DataFrame. Conform DataFrame to new index with optional filling logic, placing NA/NaN in locations having no value in the previous index. A new object is produced, the original frame is unchanged

```
dataframe.drop(index=None,
               columns=None,
               inplace = False)
```

Remove rows or columns by specifying index or column names. **index, columns** : single label or list-like. A new object is produced, the original dataframe is unchanged, **unless inplace parameter is True**.

```
Index.union(index_other)
Index.intersection(index_other)
Index.difference(index_other)
```

Form the union/intersect/difference of two Index objects and sorts if possible.

```
>>> exdf.drop(columns = ['Day'])
Out[155]:
      Description  Low Temp  High Temp  Precip  Humidity
Mon      PM Showers         50         51         50         77
Tue  PM Thundershowers         52         61         90         89
Wed      Mostly Sunny         45         60          0         55
>>> exdf
      Day      Description  Low Temp  High Temp  Precip  Humidity
Mon    1      PM Showers         50         51         50         77
Tue    2  PM Thundershowers         52         61         90         89
Wed    3      Mostly Sunny         45         60          0         55
>>> exdf.drop(columns = ['Day'], inplace=True)
>>> exdf
      Description  Low Temp  High Temp  Precip  Humidity
Mon      PM Showers         50         51         50         77
Tue  PM Thundershowers         52         61         90         89
Wed      Mostly Sunny         45         60          0         55
>>> exdf.reindex(columns = exdf.columns.union(['Wind']))
      Description  Low Temp  Humidity  High Temp  Precip  Wind
Mon      PM Showers         50         77         51         50  NaN
Tue  PM Thundershowers         52         89         61         90  NaN
Wed      Mostly Sunny         45         55         60          0  NaN
>>> exdf.rename(columns = {"Description": "Descr", "Low Temp": "LowTemp"}, inplace = True)
>>> exdf
      Day      Descr  LowTemp  High Temp  Precip  Humidity
0    1      PM Showers         50         51         50         77
1    2  PM Thundershowers         52         61         90         89
2    3      Mostly Sunny         45         60          0         55
```

Practice problem

1. Create the efdx table from the xlsx file as a pandas Dataframe.
2. Change the exdf column titles to all lower case
3. Change the index (row labels) to include the rest of the week, preserving the existing data.
4. Update the Wind column with values 45.6 , 12.5, 3
5. Rearrange the columns to list the low temp, then high temp, then other fields
6. Remove column ‘Wind’
7. Add a column with the Average Temp, computed as the average of High and Low temp values
8. Create a NumPy table with low and high temperature values; compute min, max and median values in for each of the two parameters.

SELECTION, FILTERING, SORTING

'Standard' **slicing** (but loc and iloc methods are preferred, see below)

```
>>> frame
   city country  area  population
0  Boston   NaN  11700   4628910
1  Paris    NaN  17174   12405426
2  Lagos    NaN   2706   21000000
```

```
>>> frame[:2]
   city country  area  population
0  Boston   NaN  11700   4628910
1  Paris    NaN  17174   12405426
```

```
>>> frame[['city', 'area']]
   city  area
0  Boston  11700
1  Paris  17174
2  Lagos   2706
```

```
>>> frame[['city', 'area']][:2]
   city  area
0  Boston  11700
1  Paris  17174
```

Filtering by condition

```
frame[frame['area']>10000]
Out[202]:
   city country  area  population
0  Boston   NaN  11700   4628910
1  Paris    NaN  17174   12405426
```

```
frame[frame['city'] == 'Paris']
Out[206]:
   city country  area  population
1  Paris    NaN  17174   12405426
```

Sorting by index

```
>>> frame.sort_index(ascending=False, inplace = True)
>>> frame
   city  area  population
2  Lagos  2706   21000000
1  Paris  17174   12405426
0  Boston  11700   4628910
```

Sorting by value

```
>>> frame.sort_values(by=['city'])
   city  area  population
0  Boston  11700   4628910
2  Lagos  2706   21000000
1  Paris  17174   12405426
```

```
>>> frame.sort_values(by=['population'], ascending = False)
      city  area  population
2  Lagos  2706   21000000
1  Paris  17174  12405426
0  Boston  11700   4628910
```

INDEXING AND SELECTION USING LOC AND ILOC

iloc - integer based indexing

Pandas provides ways to get **purely integer based indexing**. The semantics follow closely Python and NumPy slicing. These are 0-based indexing. When slicing, the start bounds is *included*, while the upper bound is *excluded*.

The `.iiloc` attribute is the primary access method. The following are valid inputs:

- An integer e.g. 5.
- A slice object with ints 1:7.
- A list or array of integers [4, 3, 0].
- A function with one argument that returns any of the above items

```
>>> df1 = pd.DataFrame(np.arange(1,16).reshape(5,3), index=list('abcdf'),
columns=list('xyz'))
>>> df1
      x  y  z
a     1  2  3
b     4  5  6
c     7  8  9
d    10 11 12
f    13 14 15

>>> df1.iiloc[:3, :2]
Out[215]:
      x  y
a     1  2
b     4  5
c     7  8
```

loc - label based indexing

Pandas provides access to groups of rows and columns by label(s) or a boolean array.

Allowed inputs are:

- A single label, e.g. 5 or 'a', (note that 5 is interpreted as a *label* of the index, and **never** as an integer position along the index).
- A list or array of labels, e.g. ['a', 'b', 'c'].
- A slice object with labels, e.g. 'a':'f'.

Note:

Note that contrary to usual python slices, **both** the start and the stop are included

```
>>> df1.loc['a': 'c', 'x':'y']
      x  y
a     1  2
b     4  5
```



```

c 7 8
>>> df2 = pd.DataFrame(np.arange(20,35).reshape(5,3), index=[3,4, 5,6, 7 ],
columns=list('xyz'))
>>> df2
   x  y  z
3  20 21 22
4  23 24 25
5  26 27 28
6  29 30 31
7  32 33 34

>>> df2.loc[:3] # Note, 3 is evaluated as a label, not a 0-based index.
   x  y  z
3  20 21 22
>>> df2.loc[:, 'x']
Out[231]:
3    20
4    23
5    26
6    29
7    32
Name: x, dtype: int32

```

VECTORIZED OPERATIONS, UFUNCS, ALIGNMENT

The vectorized operations from NumPy are applicable to Series and DataFrame, e.g.

```

>>> df2.loc[:, 'x']*2 + 5
Out[233]:
3    45
4    51
5    57
6    63
7    69
Name: x, dtype: int32

```

Note, that in operations involving two structures:

- operations are applied to elements with matching index values (**alignment**)
- the resulting structure contains the union of all indices

```

>>> s1 = pd.Series( range(10), [chr(code) for code in range (ord('a'), ord('a')+10)] )
>>> s2 = pd.Series( [20 for i in range(5)], [chr(code) for code in range (ord('c'),
ord('c')+5)] )
>>> s3 = s1 + s2
>>> s3
Out[117]:
a    NaN
b    NaN
c    22.0
d    23.0
e    24.0
f    25.0
g    26.0

```

```

h      NaN
i      NaN
j      NaN
dtype: float64

```

In addition, there are vectorized methods for working with **text strings**.

<http://pandas.pydata.org/pandas-docs/stable/text.html#text-string-methods>

- These methods exclude missing/NA values automatically.
- They are accessed via the Series’s `str` attribute and generally have names matching the analogous built-in string methods.
- Slicing using `[::]` syntax is done with method `str.slice()` – works on strings AND lists
- Indexing - `str.get()` – works on strings AND lists

Example

```
>>> exdf = pd.read_csv("weather.csv")
```

```
>>> exdf
```

	Day	Description	High Temp	Low Temp	Precip	Humidiy
0	1	PM Showers	50	51	50	77
1	2	PM Thundershowers	52	61	90	89
2	3	Mostly Sunny	45	60	0	55

```
>>> exdf['Description'].str.split()
```

```
Out[37]:
```

```

0      [PM, Showers]
1      [PM, Thundershowers]
2      [Mostly, Sunny]

```

```
Name: Description, dtype: object
```

```
>>> exdf['Description'].str.split().str.get(0)
```

```
Out[38]:
```

```

0      PM
1      PM
2      Mostly

```

```
Name: Description, dtype: object
```

```
>>> exdf['Description'].str.lower()
```

```

0      pm showers
1      pm thundershowers
2      mostly sunny

```

```
>>> exdf['DescrAbbrev'] = exdf['Description'].str.slice(0,5)
```

```
>>> exdf
```

	Day	Description	High Temp	...	Precip	Humidiy	DescrAbbrev
0	1	PM Showers	50	...	50	77	PM Sh
1	2	PM Thundershowers	52	...	90	89	PM Th
2	3	Mostly Sunny	45	...	0	55	Mostl

AGGREGATION

Recall that `DataFrame.values` produces a NumPy table. Hence, all aggregation methods and statistical functions that work for NumPy can be applied to `DataFrame.values`.

Also, the following Pandas aggregate functions are applicable to Series and DataFrames.

Function	Description
count	Number of non-NA 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	Bessel-corrected sample standard deviation
var	Unbiased variance
quantile	Sample quantile (value at %)

```
>>> frame.sum()
city          LagosParisBoston
area                31580
population        38034336
dtype: object
```

```
>>> round(frame.mean(), 2)
area          10526.67
population    12678112.00
dtype: float64
```

```
>>> round(frame.std(), 2)
area          7305.02
population    8188950.80
dtype: float64
```

Practice problem: based on the data contained in the IMDB.csv and IMDB-dataDict.txt, compose python code using Panda's package for the following tasks.

1. Create a DataFrame containing data from the first columns up to and including Runtime.
2. Find how many movies are in the data set.
3. Select data for those movies that run for less than 2 hours
4. Find the most expensive and least expensive film, the mean and standard deviation of the budget.
5. Find the 10 top rated movies
6. Find out the range of years for the movies listed in the data set
7. List 10 other questions to answer based on this data and create solutions for these queries.

References

- Pandas documentation <http://pandas.pydata.org/pandas-docs/stable/>
- Series <https://pandas.pydata.org/pandas-docs/stable/dsintro.html#series>
- DataFrame <https://pandas.pydata.org/pandas-docs/stable/dsintro.html#dataframe>
- Complete list of Pandas functions <https://pandas.pydata.org/pandas-docs/stable/api.html>
- Unicode HOWTO <https://docs.python.org/3/howto/unicode.html>