Pandas: Data Structures, Reading and Writing

Basic introduction

- pandas (Python Data Analysis Library) is a library specialized for data analysis and used a series of I/O API functions to read and write data as dataframe objects.

- Python IDE (Integrated Development Environment):Jupyter Notebook or Spyder.

- Easiest way to get Pandas set up is to install it through a package like the Anaconda distribution.

- Primary data structures on which all transactions are centred (generally made during the analysis):
  - `Series:` Object of the library designed to represent one-dimensional data structure.
  - `Dataframes:` More complex data structure designed to contain cases with several dimensions.

The Series

```
import pandas as pd
import numpy as np
```

```
s = pd.Series([12,-4,7,9])
s
```

Declaring a series

```
0    12
1    -4
2     7
3     9
dtype: int64
```

```
s[2]
```

Selecting one internal element

```
7
```

```
s[0:3]
```

Selecting multiple elements

```
0    12
1    -4
2     7
dtype: int64
```

```
s[1] = 0
s
```

Assigning a value to an item using its index.

```
0    12
1     0
2     7
3     9
dtype: int64
```

```
s = pd.Series([12,-4,7,9],
              index=['a','b','c','d'])
s
```

Declaring a series, assigning an index

```
a    12
b    -4
c     7
d     9
dtype: int64
```

```
s['b'] = 100
s
```

Assigning a value to an item using its label

```
a     12
b    100
c      7
d      9
dtype: int64
```

```
s[s > 8]
```

Filtering values

```
a     12
b    100
d      9
dtype: int64
```

```
s / 2
```

Operators (+,-,* and /) and mathematical function that are applicable to NumPy array can be extended to Series

```
a     6.0
b   -2.0
c     3.5
d     4.5
```

The Series – Evaluating Values

```
serd = pd.Series([1,0,2,1,2,3],
                 index=['white','white','blue',
                        'green','green','yellow'])
serd
```

```
white     1
white     0
blue      2
green     1
green     2
yellow    3
dtype: int64
```

```
serd.unique()
```

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

```
serd.value_counts()
```

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

```
serd.isin([0,3])
```

```
white     False
white      True
blue      False
green     False
green     False
yellow     True
dtype: bool
```

- The `unique ()` function will tell us all the values contained in a series, excluding duplicates

- The `value_counts()` will return the unique values but also calculate the occurrences within a series.

- The `isin()` function tells us if the values are contained in the data structure. Boolean values returned can be can be very useful when filtering data in a series or in a column of a dataframe.

The Series – NaN Values

- `NaN` (Not a Number) is used in pandas data structures to indicate the presence of an empty field or a non-numeric element. To define a missing value, we enter `np.NaN`.

- The `isnull()` and `notnull()` functions are useful to identify the indexes without a value.

- The `isnull()` returns `True` at NaN values in the series.

- The `notnull()` returns `True` if they are not NaN.

- These functions are often put inside filters to make a condition

```
s2 = pd.Series([5,-3,np.NaN, 20])
s2
```

```
0     5.0
1    -3.0
2     NaN
3    20.0
dtype: float64
```

```
s2.isnull()
```

```
0    False
1    False
2     True
3    False
dtype: bool
```

```
s2.notnull()
```

```
0     True
1     True
2    False
3     True
dtype: bool
```

```
s2[s2.notnull()]
```

```
0     5.0
1    -3.0
3    20.0
dtype: float64
```

```
s2[s2.isnull()]
```

```
2   NaN
dtype: float64
```

The Series

We can create a series from a previously defined dictionary. The array of the index is filled with the keys while the data are filled with their values.

```
mydict = {'red':250,'blue': 560,
         'green':700,'white':1456}
mydict
```

```
{'red': 250, 'blue': 560, 'green': 700, 'white': 1456}
```

```
myseries = pd.Series(mydict)
myseries
```

```
red       250
blue      560
green     700
white    1456
dtype: int64
```

We can also define the array indexes separately. As seen, if there is a mismatch, pandas will add the NaN value.

```
colours = ['red','blue','green','white', 'purple']
myseries = pd.Series(mydict, index = colours)
myseries
```

```
red        250.0
blue       560.0
green      700.0
white     1456.0
purple       NaN
dtype: float64
```

We can perform operations between two series. Series can align data addressed differently between them by identifying their corresponding labels.

```
mydict2 = {'red':900,'black':800,'white':500}
myseries2 = pd.Series(mydict2)
myseries + myseries2
```

```
black        NaN
blue         NaN
green        NaN
purple       NaN
red       1150.0
white     1956.0
dtype: float64
```

The DataFrame

- Tabular structure very similar to a spreadsheet.
- Designed to extend series to multiple dimensions.
- Consists of an ordered collection of columns, each of which can contain a value of a different type (numeric, string, Boolean, etc.)
- Unlike series which have an index array containing labels associated with each elements, the dataframe has two index arrays.
- It can be understood as a dictionary of series where the keys are the column names and the values are the series that will form the columns of the dataframe

```
import numpy as np
import pandas as pd
```

```
data = {'color': ['white','red','black','green','purple'],
        'items':['ball','pen','pencil','paper',
                 'eraser'],
        'price':[2.5,1.5,0.5,0.6,0.15]}

frame = pd.DataFrame(data)
frame
```

Define a dataframe

	color	items	price
0	white	ball	2.50
1	red	pen	1.50
2	black	pencil	0.50
3	green	paper	0.60
4	purple	eraser	0.15

```
frame2 = pd.DataFrame(data,columns=['items','price'])
frame2
```

	items	price
0	ball	2.50
1	pen	1.50
2	pencil	0.50
3	paper	0.60
4	eraser	0.15

We can select the data we want to display. Use column option to specify the sequence of columns. dataframe

The DataFrame

- A common data structure used in Python is a `nested dict`. When it is passed directly as an argument to the DataFrame() constructor, pandas will treat external keys as column names and internal keys as labels for indexes.
- Fields with no match are assigned the `NaN` value.

```
nesteddict = {'red':{2012:22, 2014:45},
'green':{2008: 23, 2012:22,2014:17},
    'blue':{2008: 18,2012:28,2014: 19}}
frame3 = pd.DataFrame(nesteddict)
frame3
```

	red	green	blue
2008	NaN	23	18
2012	22.0	22	28
2014	45.0	17	19

- `Transposition`: columns become row and rows become columns. It is achieved by adding the T attribute to its operation

```
frame3.T
```

	2008	2012	2014
red	NaN	22.0	45.0
green	23.0	22.0	17.0
blue	18.0	28.0	19.0

Reading Data in CSV or Text files

- Most common operation for data analysis is to read the data contained in a .CSV file of even a text file.
- To achieve the, we must import the following libraries `numpy` and `pandas` in our Jupyter Notebook

- The `read_csv()` function will read the content of the `.csv` file and convert it to a `dataframe object.`

In [6]:
```python
import numpy as np
import pandas as pd
csvframe = pd.read_csv('Documents/texting.csv')
csvframe
```

Out[6]:

	white	red	blue	green	animal
0	1	5	2	3	car
1	2	7	8	5	dog
2	3	3	6	7	horse
3	2	2	8	3	duck
4	4	4	2	1	mouse

In [13]:
```python
csvframe1 = pd.read_csv('Documents/texting.txt')
csvframe1
```

Out[13]:

	white	red	blue	green	animal
0	1	5	2	3	cat
1	2	7	8	5	dog
2	3	3	6	7	horse
3	2	2	8	3	duck
4	4	4	2	1	mouse

`Original texting.csv and texting.txt files as seen in the spreadsheet and notepad`

white	red	blue	green	animal
1	5	2	3	car
2	7	8	5	dog
3	3	6	7	horse
2	2	8	3	duck
4	4	2	1	mouse

```
white,red,blue,green,animal
1,5,2,3,cat
2,7,8,5,dog
3,3,6,7,horse
2,2,8,3,duck
4,4,2,1,mouse
```

Reading Data in CSV or Text files

- When using `read_table()` function to read a csv or txt file, specify the delimiter otherwise, the data will not be in a tabulated format.

Output of `read_table()` function without specifying the delimiter

```
In [15]: pd.read_table('Documents/texting.csv')
Out[15]:
```

	white,red,blue,green,animal
0	1,5,2,3,car
1	2,7,8,5,dog
2	3,3,6,7,horse
3	2,2,8,3,duck
4	4,4,2,1,mouse

Output of `read_table()` function with specified delimiter

```
In [16]: pd.read_table('Documents/texting.csv', sep=',')
Out[16]:
```

	white	red	blue	green	animal
0	1	5	2	3	car
1	2	7	8	5	dog
2	3	3	6	7	horse
3	2	2	8	3	duck
4	4	4	2	1	mouse

Reading Data in CSV or Text files

- `pd.read_csv('Documents/texting.csv', header=None):` This will tell pandas to assign the default name to the columns.

- You can specify the names directly by assigning a list of labels to the name options `pd.read_csv('Documents/texting.csv', names=[‘white’,’red’,’blue’,’green’,’animal])`

- Create a dataframe with a hierarchical structure by extending the functionality of the `read_csv()` function by adding the `index_col` option.

Original Hierarchical data

```
In [22]: pd.read_csv('Documents/texting1.txt', index_col=['color','status'])
Out[22]:
```

color	status	item1	item2	item3
black	up	3	4	6
	down	2	6	7
white	up	5	5	5
	down	3	3	2
	left	1	2	1
red	up	2	2	2
	down	1	1	4

texting1 - Notepad

File Edit Format View Help

```
color,status,item1,item2,item3
black,up,3,4,6
black,down,2,6,7
white,up,5,5,5
white,down,3,3,2
white,left,1,2,1
red,up,2,2,2
red,down,1,1,4
```

Using RegExp to Parse TXT files

- Sometimes the files on which to parse the data do not show separators such as comma or a semicolon.

- Regular expressions can be used as criteria for value separation.

.	Single character, except newline
\d	Digit
\D	Non-digit character
\s	Whitespace character
\S	Non-whitespace character
\n	New line character
\t	Tab character
\uxxxx	Unicode character specified by the hexadecimal number xxxx

Using RegExp to Parse TXT files - Examples

```
In [23]: pd.read_table('Documents/texting2.txt', sep='\s+', engine='python')
```

Out[23]:

	white	red	blue	green
0	1	5	2	3
1	2	7	8	5
2	3	3	6	7
3	2	2	8	3
4	4	4	2	1

texting2 - Notepad

File Edit Format View Help

```
white   red     blue    green
1       5       2       3
2       7       8       5
3       3       6       7
2       2       8       3
4       4       2       1
```

```
In [28]: pd.read_table('Documents/texting3.txt', sep='\D+', header=None, engine='python')
```

Out[28]:

	0	1	2
0	0	123	122
1	1	124	321
2	2	125	333

texting3 - Notepad

File Edit Format View

```
00END123AAA122
001END124BBB321
002END125CCC333
```

This example extract the numeric part from a texting3.txt file. Header option = None because the column heading is not in the texting3.txt file.

Using RegExp to Parse TXT files - Examples

With the `skiprows` option, you can exclude the lines you want.

```
pd.read_table('Documents/texting2.txt', sep='\D+', header=None, engine='python', skiprows=[0,1,2])
```

	0	1	2	3
0	3	3	6	7
1	2	2	8	3
2	4	4	2	1

texting2.txt file dataframe after skipping the first three rows

	0	1	2	3
0	white	red	blue	green
1	1	5	2	3
2	2	7	8	5
3	3	3	6	7
4	2	2	8	3
5	4	4	2	1

Original texting2.txt file dataframe

Reading TXT Files into Parts

- To read only a portion of the file, you can specify the numbers of lines on which to parse. You can use the `nrows` and `skiprows` options.

```
pd.read_csv('Documents/texting.csv', skiprows=[2], nrows=3, header=None)
```

	0	1	2	3	4
0	white	red	blue	green	animal
1	1	5	2	3	car
2	3	3	6	7	horse

Dataframe after using nrows and skiprows

Original dataframe

	0	1	2	3	4
0	white	red	blue	green	animal
1	1	5	2	3	car
2	2	7	8	5	dog
3	3	3	6	7	horse
4	2	2	8	3	duck
5	4	4	2	1	mouse

Writing Data in CSV

- `to_csv()` : Function used to write the data contained in a dataframe to a csv file.

```
frame = pd.DataFrame(np.arange(16).reshape((4,4)),
        index = ['red','blue','yellow','white'],
        columns = ['ball','pen','pencil','paper'])
```

```
frame.to_csv('writin.txt')
```

```
pd.read_csv('writin.txt')
```

	Unnamed: 0	ball	pen	pencil	paper
0	red	0	1	2	3
1	blue	4	5	6	7
2	yellow	8	9	10	11
3	white	12	13	14	15

The `index` and `header` to `False` options are used to remove the default indexes and columns that are marked on the file by default.

```
frame.to_csv('writin.txt', index=False, header=False)
```

	0	1	2	3
0	4	5	6	7
1	8	9	10	11
2	12	13	14	15

Reading and Writing HTML & Excel Files

- `to_html()` function will convert a dataframe into an HTML table.
- `read_html()` function returns a list of dataframes even if there is only one table.
- `to_excel()` function to convert a dataframe into a spreadsheet on Excel.
- `read_excel()` read the data contained in the excel file and convert it into a dataframe

```
frame = pd.DataFrame(np.arange(4).reshape(2,2))
```

```
print(frame.to_html())
```

```
<table border="1" class="dataframe">
  <thead>
    <tr style="text-align: right;">
      <th></th>
      <th>0</th>
      <th>1</th>
    </tr>
  </thead>
  <tbody>
    <tr>
      <th>0</th>
      <td>0</td>
      <td>1</td>
    </tr>
    <tr>
      <th>1</th>
      <td>2</td>
      <td>3</td>
    </tr>
  </tbody>
</table>
```

Interacting with Databases

- `pandas.io.sql` module provides a unified interface independent of the DB called `sqlalchemy`. This interface simplifies the connection mode, regardless of the commands will be always be the same.

- The `create_engine()` function is used to make a connection.

- Example of code for connecting different databases.

```
For PostgreSQL
engine = create_engine('postgresql://mireilla:text@localhost:5432/mydatabase')

For MySQL
engine = create_engine('mysql+mysqldb://mireilla:text@localhost/foo')

For Oracle
engine = create_engine('oracle://mireilla:text@127.0.0.1:1521/sidname')

For MSSQL
engine = create_engine('mssql+pyodbc://mydsn')

For SQLite
engine = create_engine('sqlite:///foo.db')
```

Interacting with Databases – SQLite3

```
from sqlalchemy import create_engine
```

```
frame = pd.DataFrame(np.arange(20).reshape(4,5),
                     columns=['white','red','blue','black','green'])
```

```
frame
```

	white	red	blue	black	green
0	0	1	2	3	4
1	5	6	7	8	9
2	10	11	12	13	14
3	15	16	17	18	19

Create a dataframe that will be used to create a new table on the SQLite3 database

```
engine = create_engine('sqlite:///foo.db')
```

Implement the connection to the SQLite3 database

```
frame.to_sql('colors',engine)
```

Convert the dataframe

```
pd.read_sql('colors',engine)
```

Read the database with the read_sql() function with the name of the table and the engine

	index	white	red	blue	black	green
0	0	0	1	2	3	4
1	1	5	6	7	8	9
2	2	10	11	12	13	14
3	3	15	16	17	18	19