

Introduction to Data Science – Week 11

NETA LIVNEH

COURSE 55793

2019-2020

Last Week Recap

Classes

List comprehension

Map, Filter and Lambda



This Week

Jupyter notebook

Introduction to Numpy

Introduction to Pandas

(SQL type of thinking)



Jupyter Notebook

The screenshot shows a Jupyter Notebook interface running in a web browser. The top navigation bar includes links for 'localhost:8888/tree' (highlighted), 'NEW', and various system icons like star, refresh, and user profile. The title bar says 'jupyter'. Below the title bar are tabs for 'Files', 'Running', and 'Clusters', with 'Running' currently selected. A message 'Select items to perform actions on them.' is displayed above a file list. The file list shows the following entries:

| | Name | Last Modified | File size |
|--------------------------|--------------------------------|---------------------|-----------|
| <input type="checkbox"/> | Danny | 14 minutes ago | |
| <input type="checkbox"/> | Example1.ipynb | Running a year ago | 2.23 kB |
| <input type="checkbox"/> | how-much-sugar-do-we-eat.ipynb | Running a month ago | 44 kB |
| <input type="checkbox"/> | Untitled.ipynb | Running a month ago | 72 B |
| <input type="checkbox"/> | visualizing-food.ipynb | 3 days ago | 21.1 kB |
| <input type="checkbox"/> | Yelp.ipynb | a month ago | 21 kB |
| <input type="checkbox"/> | average_bachdel+test.csv | a month ago | 2.53 kB |
| <input type="checkbox"/> | average_bachdel.txt | a month ago | 815 B |



How to use Jupyter Notebook

To start the notebook server from the command line:

1. Open the command line
2. By using 'cd', go to your local directory where your files are (or found in a subdirectory)
3. Enter 'jupyter notebook' in the command line
4. You should see the notebook open in your browser

Optionally, you can open Jupyter Notebook from the Anaconda navigator

```
[→ ~ cd '/Users/Neta/Documents/Teaching/Introduction to Data Science /2018-2019/]  
Examples/'  
[→ Examples] jupyter notebook  
[I 17:37:33.125 NotebookApp] Serving notebooks from local directory: /Users/Neta/  
/Documents/Teaching/Introduction to Data Science /2018-2019/Examples  
[I 17:37:33.125 NotebookApp] The Jupyter Notebook is running at:  
[I 17:37:33.126 NotebookApp] http://localhost:8888/?token=ee0b5f3761d0a5d02f495a  
f2184db6094d16d1dfc39c64d1  
[I 17:37:33.126 NotebookApp] Use Control-C to stop this server and shut down all  
kernels (twice to skip confirmation).  
[C 17:37:33.136 NotebookApp]  
  
Copy/paste this URL into your browser when you connect for the first time,  
to login with a token:  
from the http://localhost:8888/?token=ee0b5f3761d0a5d02f495af2184db6094d16d1dfc39  
c64d1  
[I 17:37:33.492 NotebookApp] Accepting one-time-token-authenticated connection f  
rom ::1  
[I 17:37:40.526 NotebookApp] Kernel started: 41a34b2b-49ad-48c2-9a96-ef418353e25  
f  
open in your browser
```



How to install

For Anaconda users, pandas is already installed so no need to do anything further.

For Python users (make sure to install all of its dependencies like numpy, scipy and matplotlib)

`pip install pandas`

`pip install jupyter-notebook`



Detour: numpy library



Numpy is the core library for scientific computing in Python.

It provides a high-performance multidimensional array object, and tools for working with these arrays.

Compared with lists, working with numpy is more compact, has faster access in reading and writing items, being more convenient and more efficient. However, there are also more constraints.

It also comes with many functions that manipulate arrays (and matrixes), and make mathematical calculations on arrays.



Numpy array

The basic object that we would work with is an array.

NumPy's array class is called ndarray. It is also known by the alias array.

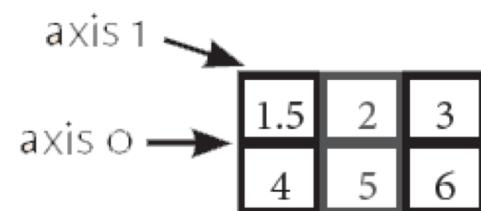
A numpy array is a grid of values, all of the same type (dtype), and is indexed by a tuple of nonnegative integers.

The number of dimensions is the *rank* of the array; the *shape* of an array is a tuple of integers giving the size of the array along each dimension

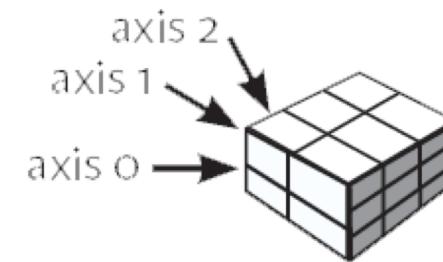
1D array

| | | |
|---|---|---|
| 1 | 2 | 3 |
|---|---|---|

2D array



3D array



Slicing and indexing an array

```
>>> a[(0,1,2,3,4), (1,2,3,4,5)]  
array([1, 12, 23, 34, 45])  
  
>>> a[3:, [0,2,5]]  
array([[30, 32, 35],  
       [40, 42, 45],  
       [50, 52, 55]])  
  
>>> mask = np.array([1,0,1,0,0,1], dtype=bool)  
>>> a[mask, 2]  
array([2, 22, 52])
```

| | | | | | |
|----|----|----|----|----|----|
| 0 | 1 | 2 | 3 | 4 | 5 |
| 10 | 11 | 12 | 13 | 14 | 15 |
| 20 | 21 | 22 | 23 | 24 | 25 |
| 30 | 31 | 32 | 33 | 34 | 35 |
| 40 | 41 | 42 | 43 | 44 | 45 |
| 50 | 51 | 52 | 53 | 54 | 55 |



Operations on arrays (to name a few)

Sum()

Count()

Mean()

Std()

Floor()

Random()

Max()

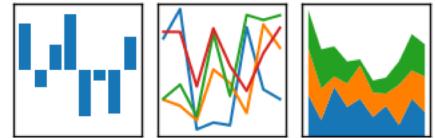
Min()

Cumsum()



Pandas library

pandas
 $y_{it} = \beta' x_{it} + \mu_i + \epsilon_{it}$



Pandas (derived from the word Panel Data) is probably the most popular library for data manipulation and analysis in Python.

Key Features:

- Fast and efficient DataFrame object with default and customized indexing, and label
- Label-based slicing, indexing, subsetting, and filtering
- Tools for loading data from different file formats like txt, csv, excel
- Integrated handling of missing data
- Integrated data manipulation like reshape and pivot, merging and joining
- Group by data for aggregation and transformations
- Time Series functionality
- Built-in plotting with matplotlib
- Based on Numpy



Pandas Series object

Series is an object which is similar to list. Unlike list, it has associated index with each element (values).

To access the index, use `my_series.index`

To access the values, use `my_series.values`

You can define a new series by using dictionary (where key is the index name and value is value)

```
>>> import pandas as pd  
>>> my_series = pd.Series([5, 6, 7, 8, 9, 10])  
>>> my_series  
0    5  
1    6  
2    7  
3    8  
4    9  
5    10  
dtype: int64
```

```
>>> my_series2 = pd.Series(range(5, 11),  
... index=['a', 'b', 'c', 'd', 'e', 'f'])  
>>> my_series2  
a    5  
b    6  
c    7  
d    8  
e    9  
f    10  
dtype: int64
```



Manipulating Series object

Can retrieve elements by their index

Make group assignment

Can filter and do math operations

```
>>> my_series2[0]
5
>>> my_series2[2:5]
c    7
d    8
e    9
dtype: int64
>>> my_series2[['a', 'f', 'c']]
a    5
f    10
c    7
dtype: int64
```

```
>>> my_series2 > 7
a    False
b    False
c    False
d    True
e    True
f    True
dtype: bool
>>> my_series2[my_series2 > 7] * 2
d    16
e    18
f    20
dtype: int64
```



Pandas DataFrame

Simply said, DataFrame is a table. It has rows and columns. Each column in a DataFrame is a Series object, rows consist of elements inside Series.

columns

| | Name | Team | Number | Position | Age | Height | Weight | College | Salary |
|---|---------------|----------------|--------|----------|------|--------|--------|-------------------|-----------|
| 0 | Avery Bradley | Boston Celtics | 0.0 | PG | 25.0 | 6-2 | 180.0 | Texas | 7730337.0 |
| 1 | Jae Crowder | Boston Celtics | 99.0 | SF | 25.0 | 6-6 | 235.0 | Marquette | 6796117.0 |
| 2 | John Holland | Boston Celtics | 30.0 | SG | 27.0 | 6-5 | 205.0 | Boston University | NaN |
| 3 | R.J. Hunter | Boston Celtics | 28.0 | SG | 22.0 | 6-5 | 185.0 | Georgia State | 1148640.0 |
| 4 | Jonas Jerebko | Boston Celtics | 8.0 | PF | 29.0 | 6-10 | 231.0 | NaN | 5000000.0 |

Index column

rows



Creating a DataFrame object

From scratch:

- from Lists
- from dict of Lists
- from list of dicts
- from dict of series
- etc.

```
>>> data = {'Name': ['Tom', 'Jack', 'Steve', 'Ricky'], 'Age': [28,34,29,42]}\n>>> df = pd.DataFrame(data, index=['rank1','rank2','rank3','rank4'])\n>>> df\n      Name  Age\nrank1   Tom   28\nrank2   Jack   34\nrank3  Steve   29\nrank4  Ricky  42
```



Reading and writing files

Reading from file:

```
pd.read_table('some_text_file.txt')
```

```
pd.read_csv('some_csv_file.csv')
```

Writing to file:

```
df.to_csv('sove_csv_file.csv')
```



Exploring a DataFrame

This is a small sample of things you can do

| module/attribute | explanation |
|------------------|--|
| df.head() | Returns the 5 first rows of df |
| df.tail() | Returns the 5 last rows of df |
| df.shape | Returns a tuple representing the dimensionality of df (row, column) |
| df.columns | Names of columns |
| df.index | Df.Index values |
| df.info() | Returns the data type and number of entries (non-null) for each column in df |
| df.describe() | Returns a df that calculates count, mean, std, etc. for every number column |
| df.sum(axis=0) | Sums the values of a row (if axis=0) or columns (axis=1) |
| df.mean(axis=0) | Return the mean of the values for the requested axis |
| df.max(axis=0) | Return the maximum value for the requested axis |



Selecting from a Dataframe

- Select a column by using a dot notation or the column label in bracket
- To select multiple columns insert a list column labels
- Use df.loc[rows filter, column labels]
- To select by index use df.iloc[]

```
>>> df.loc[df['Age'] > 30, ['Name', 'Age']]  
      Name  Age  
rank2  Jack  34  
rank4  Ricky  42
```

```
>>> df.Name  
rank1      Tom  
rank2      Jack  
rank3     Steve  
rank4    Ricky  
Name: Name, dtype: object  
>>> df['Age']  
rank1    28  
rank2    34  
rank3    29  
rank4    42  
Name: Age, dtype: int64
```



Rules for specifying multiple filter criteria in Pandas

use **&** instead of **and**

use **|** instead of **or**

add **parentheses** around each condition to specify evaluation order

