DATA4203 - Data Mining

Week 3 Tutorial Handbook

Section 1: Background and Jupyter Notebook Configuration

Demonstrative Python Platform
Jupyter Notebook, https://jupyter.org

Getting started with the classic Jupyter Notebook

conda

We recommend installing the classic Jupyter Notebook using the conda package manager. Either the miniconda or the miniforge conda distributions include a minimal conda installation.

Then you can install the notebook with:

```
conda install -c conda-forge notebook
```

pip

If you use `pip`, you can install it with:

```
pip install notebook
```

Congratulations, you have installed Jupyter Notebook! To run the notebook, run the following command at the Terminal (Mac/Linux) or Command Prompt (Windows):

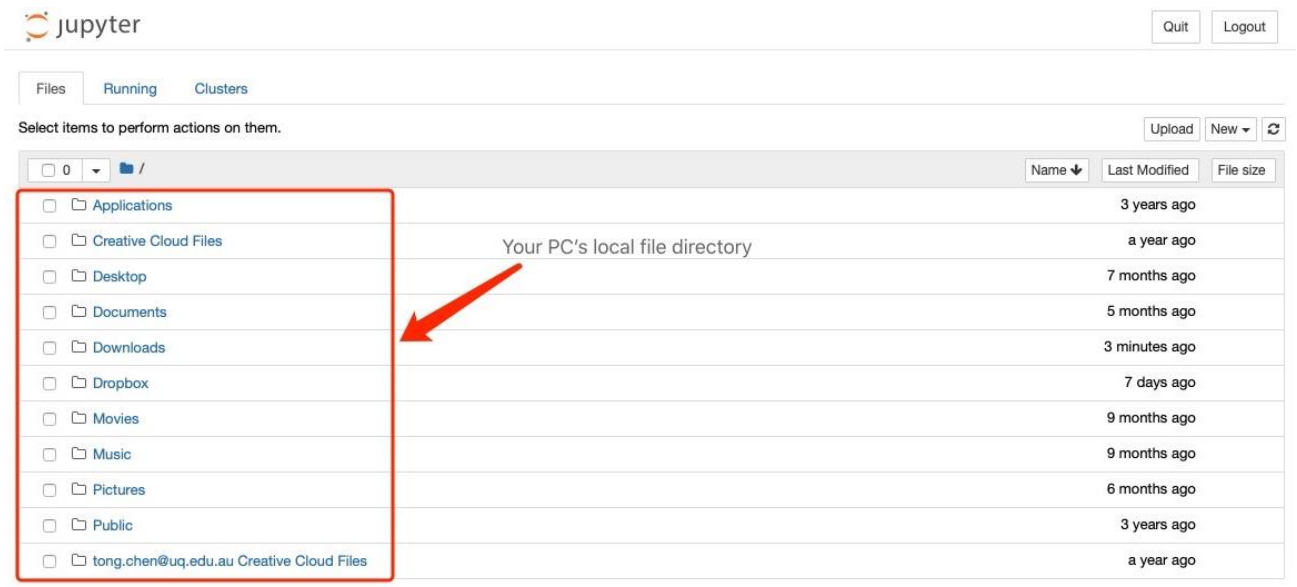
```
jupyter notebook
```

See Running the Notebook for more details.

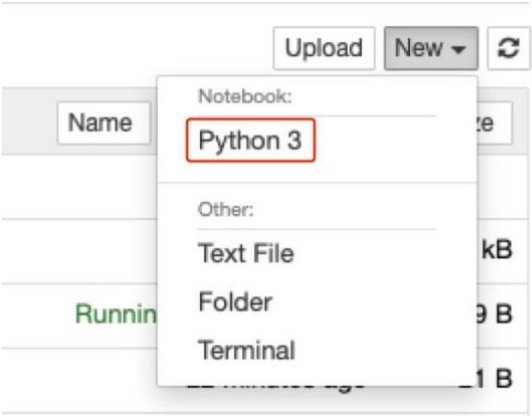
In case the above process does not work out on your MacBook:

- Install Python3
- Run “sudo python3 -m pip install jupyter”

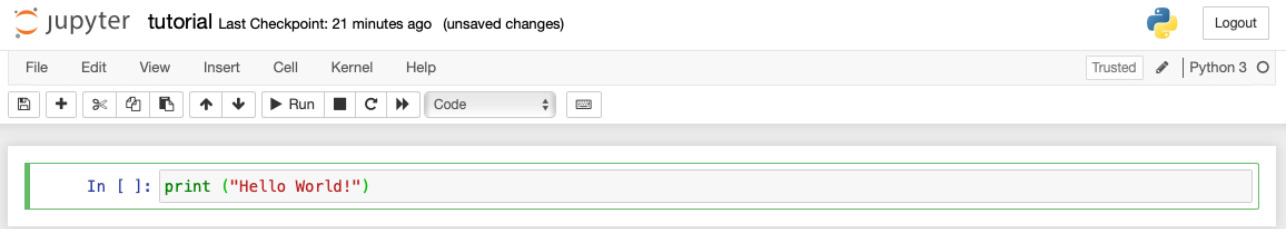
And after executing “jupyter notebook” on you PC:



Create a new .ipynb file:



Then we are in:



Now start the classic one-line trial – to execute the python code, click “Run” or press “Shift + Enter” (always remember to save your file on-the-fly!):

```
In [1]: print ("Hello World!")

        Hello World!
```

Section 2: Plotting with Python

Now, let us try a commonly-used feature of Python, which is generating plots:
After execution for the first time, we will possibly obtain:

```
# scenario: weather data visualization
day = [1, 2, 3, 4, 5, 6, 7]
temperature = [15, 19, 21, 25, 16, 14, 12]

# what if I want to plot something?
import matplotlib.pyplot as plt # import pyplot package and shorten its name
plt.plot(day, temperature) # input (x, y) values when calling plot function
```

```
---------------------------------------------------------------------------
ModuleNotFoundError                       Traceback (most recent call last)
<ipython-input-2-9762786e4a7a> in <module>
      4
      5 # what if I want to plot something?
----> 6 import matplotlib.pyplot as plt # import pyplot package and shorten its name
      7 plt.plot(day, temperature) # input (x, y) values when calling plot function

ModuleNotFoundError: No module named 'matplotlib'
```

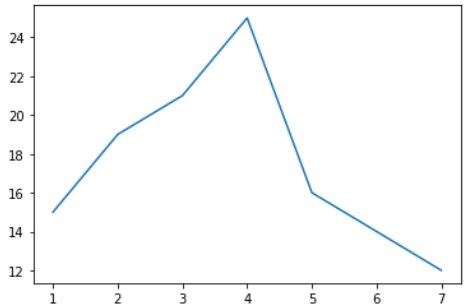
Apparently, the package “matplotlib” we want to use is not available, and this is where the Package Installation comes into play. To do it, execute “sudo python3 -m pip install matplotlib” in your command line (change “python3” to “python” if you are using Python2).

Then let us try it again:

```
# scenario: weather data visualization
day = [1, 2, 3, 4, 5, 6, 7]
temperature = [15, 19, 21, 25, 16, 14, 12]

# what if I want to plot something?
import matplotlib.pyplot as plt # import pyplot package and shorten its name
plt.plot(day, temperature) # input (x, y) values when calling plot function
```

```
[<matplotlib.lines.Line2D at 0x118ee14e0>]
```

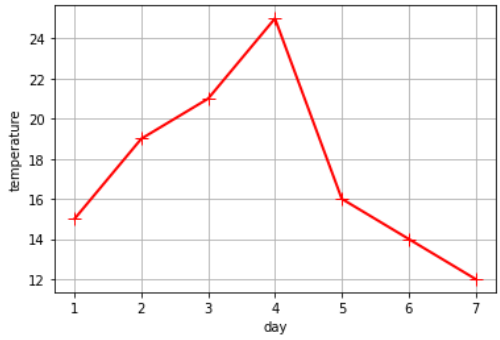


Basically, to install an unavailable Python package, use the aforementioned pip installation tool. Here is the link to an after-class reading material about pip installation tool: https://pip.pypa.io/en/stable/.

To take one step above the current plot and make it look better:

```
# scenario: weather data visualization
day = [1, 2, 3, 4, 5, 6, 7]
temperature = [15, 19, 21, 25, 16, 14, 12]

# what if I want to plot something?
import matplotlib.pyplot as plt # import pyplot package and shorten its name
plt.plot(day, temperature, color = 'red', linewidth = 2.0, marker = '+', markersize = 10)
# adding options on line color, width, and style
plt.xlabel('day')
plt.ylabel('temperature') # adding x/y labels to the plot
plt.grid(True) # adding grid for a nicer look
```



But a question here is — how can we effectively load those numbers (e.g., day and temperature) instead of manually typing them in?

Section 3: Loading Datasets with Python

Let us consider a toy dataset collected from an online video platform, stored in .csv format (file name: “tutorial_data.csv”). If opened using Microsoft Excel, it looks like:

I14

Numer of movies watched in a category

	username	gender	age	documentary movies	action movies	scifi movies	comedy movies	horror movies
2	AS	female	20	5	19	21	25	12
3	DF	female	30	4	33	9	30	5
4	GH	female	25	14	12	11	4	3
5	JK	female	19	10	50	25	3	6
6	LQ	female	19	0	9	7	12	6
7	WE	male	21	5	12	17	10	8
8	RT	male	32	5	38	5	7	15
9	YU	male	22	9	17	13	20	11
10	MJ	male	26	6	21	21	7	7
11	IP	male	18	11	18	16	9	20
12								

Then the first thing is to load this dataset into our program using Pandas. Pandas is a very versatile Data Frame Package (https://pandas.pydata.org):

```
# scenario: movie data processing
import csv # package for reading csv files
import numpy as np # package for dealing with numerical data
import pandas as pd # this is a data frame package

data_frame = pd.read_csv('/Users/rockychen/Downloads/tutorial_data.csv')
data_frame.head(11) # print all 11 lines within the loaded csv file
```

	username	gender	age	documentary movies	action movies	scifi movies	comedy movies	horror movies
0	AS	female	20	5	19	21	25	12
1	DF	female	30	4	33	9	30	5
2	GH	female	25	14	12	11	4	3
3	JK	female	19	10	50	25	3	6
4	LQ	female	19	0	9	7	12	6
5	WE	male	21	5	12	17	10	8
6	RT	male	32	5	38	5	7	15
7	YU	male	22	9	17	13	20	11
8	MJ	male	26	6	21	21	7	7
9	IP	male	18	11	18	16	9	20

Replacing “.head()” with “.info()” gives the data type of each column:

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10 entries, 0 to 9
Data columns (total 8 columns):
 #   Column               Non-Null Count  Dtype
---  ------               --------------  -----
 0   username             10 non-null     object
 1    gender              10 non-null     object
 2    age                 10 non-null     int64
 3    documentary movies  10 non-null     int64
 4    action movives      10 non-null     int64
 5    scifi movies        10 non-null     int64
 6    comedy movies       10 non-null     int64
 7    horror movies       10 non-null     int64
dtypes: int64(6), object(2)
memory usage: 768.0+ bytes
```

How can we select specific columns of a matrix (i.e., table) using pandas?
If we want to retrieve the age of each user, we need to retrieve all 10 rows with colums [‘username’, ‘age’] as follows:

```
# scenario: movie data processing
import csv # package for reading csv files
import numpy as np # package for dealing with numerical data
import pandas as pd # this is a data frame package

data_frame = pd.read_csv('/Users/rockychen/Downloads/tutorial_data.csv')
data_frame.loc[:, ['username', 'age']]
```

	username	age
0	AS	20
1	DF	30
2	GH	25
3	JK	19
4	LQ	19
5	WE	21
6	RT	32
7	YU	22
8	MJ	26
9	IP	18

Similarly, we can further use the following to select arbitrary rows from the matrix:

```
# scenario: movie data processing
import csv # package for reading csv files
import numpy as np # package for dealing with numerical data
import pandas as pd # this is a data frame package

data_frame = pd.read_csv('/Users/rockychen/Downloads/tutorial_data.csv')
data_frame.loc[[1, 8], ['username', 'age']]
```

	username	age
1	DF	30
8	MJ	26

With that, how can we calculate the average age of users DF and MJ?

```
# scenario: movie data processing
import csv # package for reading csv files
import numpy as np # package for dealing with numerical data
import pandas as pd # this is a data frame package

data_frame = pd.read_csv('/Users/rockychen/Downloads/tutorial_data.csv')
two_ages = data_frame.loc[[1, 8], ['age']]
mean_age = two_ages.values.mean()
print('The average age of users DF and MJ is: ' + str(mean_age))
```

```
The average age of users DF and MJ is: 28.0
```

*A little after-class practice: (1) How can we calculate the mean for all users? (2) How can we calculate the total number of movies watched per user?

A follow-up question: what is the most/least popular movie category?

```
# scenario: movie data processing
import csv # package for reading csv files
import numpy as np # package for dealing with numerical data
import pandas as pd # this is a data frame package

data_frame = pd.read_csv('/Users/rockychen/Downloads/tutorial_data.csv')
all_watches = data_frame.loc[:, ['documentary movies', 'action movies', 'scifi movies', 'comedy movies',\
                                 'horror movies']]
category_wise_watches = all_watches.sum() # column-wise summation
max_category = category_wise_watches.idxmax() # return the index of maximum number
min_category = category_wise_watches.idxmin() # return the index of minimum number

print('The most popular movie category is: ' + str(max_category) + '\n' + \
      'The least popular movie category is: ' + str(min_category))
```

```
The most popular movie category is: action movies
The least popular movie category is: documentary movies
```

Section 4: An introduction on vector/matrix-based calculations

In this movie dataset, we can represent each user's interest using the numbers of different movie categories she/he has interacted with. For example, DF can be represented by a 5-dimensional vector [4, 33, 9, 30, 5] extracted via the following:

	username	gender	age	documentary movies	action movies	scifi movies	comedy movies	horror movies
0	AS	female	20	5	19	21	25	12
1	DF	female	30	4	33	9	30	5
2	GH	female	25	14	12	11	4	3
3	JK	female	19	10	50	25	3	6
4	LQ	female	19	0	9	7	12	6
5	WE	male	21	5	12	17	10	8
6	RT	male	32	5	38	5	7	15
7	YU	male	22	9	17	13	20	11
8	MJ	male	26	6	21	21	7	7
9	IP	male	18	11	18	16	9	20

For all 10 users, the following will generate a 10*5 Numpy (this is a very important package! Please see: https://numpy.org) array that allows us to perform further computations:

```
# scenario: movie data processing
import csv # package for reading csv files
import numpy as np # package for dealing with numerical data
import pandas as pd # this is a data frame package

data_frame = pd.read_csv('/Users/rockychen/Downloads/tutorial_data.csv')
all_watches = data_frame.loc[:, ['documentary movies', 'action movies', 'scifi movies', 'comedy movies',\
                                 'horror movies']]
vectors = all_watches.values
print(vectors)
```

```
[[ 5 19 21 25 12]
 [ 4 33  9 30  5]
 [14 12 11  4  3]
 [10 50 25  3  6]
 [ 0  9  7 12  6]
 [ 5 12 17 10  8]
 [ 5 38  5  7 15]
 [ 9 17 13 20 11]
 [ 6 21 21  7  7]
 [11 18 16  9 20]]
```

How can we perform basic operations, like summing five vectors, with Numpy arrays?

```python
# scenario: movie data processing
import csv # package for reading csv files
import numpy as np # package for dealing with numerical data
import pandas as pd # this is a data frame package

data_frame = pd.read_csv('/Users/rockychen/Downloads/tutorial_data.csv')
all_watches = data_frame.loc[:, ['documentary movies', 'action movies', 'scifi movies', 'comedy movies',\
                                 'horror movies']]
vectors = all_watches.values
category_wise_watch = np.sum(vectors[0:5,:], axis = 0) # sum along axis 0 (column)
user_wise_watch = np.sum(vectors[0:5,:], axis = 1) # sum along axis 1 (row)
total_watch = np.sum(vectors[0:5,:]) # sum all the elements

str_list1 = [str(category_wise_watch[i]) for i in range(len(category_wise_watch))] # float to str
str_list2 = [str(user_wise_watch[i]) for i in range(len(user_wise_watch))]

print('For the first five users, they have watched ' + str_list1[0] + ' documentary, ' + \
     str_list1[1] + ' action, ' + str_list1[2] + ' scifi, ' + str_list1[3] + ' comedy, and ' + \
     str_list1[4] + ' horror movies.')
print('And each of them has watched ' + str_list2[0] + ', ' + str_list2[1] + ', ' + \
     str_list2[2] + ', ' + str_list2[3] + ', ' + str_list2[4] + ' movies, respectively, ' + \
      'leading to a total number of ' + str(total_watch) + '.')
```

```
For the first five users, they have watched 33 documentary, 123 action, 73 scifi, 74 comedy, and 32 horror movies.
And each of them has watched 82, 81, 44, 94, 34 movies, respectively, leading to a total number of 335.
```

How can we compare different users with their 5-dimensional vectors? A possible approach – calculating the L1-/L2-norm of a target vector:

```python
import csv # package for reading csv files
import numpy as np # package for dealing with numerical data
import pandas as pd # this is a data frame package

data_frame = pd.read_csv('/Users/rockychen/Downloads/tutorial_data.csv')
all_watches = data_frame.loc[:, ['documentary movies', 'action movies', 'scifi movies', 'comedy movies',\
                                 'horror movies']]
vectors = all_watches.values
DF_vector = vectors[1, :] # fetch a user's vector
l1 = np.linalg.norm(DF_vector, ord = 1) # by specifying order = 1, we calculate the L1 norm
l2 = np.linalg.norm(DF_vector) # the default setting of this function calculates the L2 norm

print('The L1-norm of DF\'s vector is: ' + str(l1) + "\n" + \
      'The L2-norm of DF\'s vector is: ' + str(l2))
```

```
The L1-norm of DF's vector is: 81.0
The L2-norm of DF's vector is: 45.94562003064057
```

Then, we can compare any two vectors using. In addition, two vectors can also be compared using metrics like Euclidean distance or dot product:

```python
import csv # package for reading csv files
import numpy as np # package for dealing with numerical data
import pandas as pd # this is a data frame package

data_frame = pd.read_csv('/Users/rockychen/Downloads/tutorial_data.csv')
all_watches = data_frame.loc[:, ['documentary movies', 'action movies', 'scifi movies', 'comedy movies',\
                                 'horror movies']]
vectors = all_watches.values
DF_vector = vectors[1, :]
MJ_vector = vectors[8, :] # fetch two users' vectors

euc_dist = np.linalg.norm(np.subtract(DF_vector, MJ_vector)) # L2-norm of vector subtraction = Euclidean distance
dot_dist = np.dot(DF_vector, MJ_vector) # dot product of two vectors

print('The Euclidean distance between DF\'s and MJ\'s vecotrs is (the lower the closer): ' + str(euc_dist) + '\n' +\
      'The dot product between DF\'s and MJ\'s vecotrs is (the higher the closer): ' + str(dot_dist))
```

```
The Euclidean distance between DF's and MJ's vecotrs is (the lower the closer): 28.722813232690143
The dot product between DF's and MJ's vecotrs is (the higher the closer): 1151
```

*Another after-class practice: how can we compute the cosine similarity between two vectors using Numpy?

Is there a more intuitive way to compare users' interest on different movie categories?

Section 5: Final Task for Today – Clustering and Plotting

Final task for today: a simple user clustering task using the 2-dimensional [‘comedy movies’, ‘horror movies’] vectors extracted from the dataset.

Step 1 – gathering vectors for all users and applying normalization:

```
import csv # package for reading csv files
import numpy as np # package for dealing with numerical data
import pandas as pd # this is a data frame package

data_frame = pd.read_csv('/Users/rockychen/Downloads/tutorial_data.csv')
all_watches = data_frame.loc[:, ['comedy movies', 'horror movies']]

vectors = all_watches.values # [comedy, horror] vector for all users
vectors_l1 = np.linalg.norm(vectors, ord = 1, axis = 1, keepdims = True) # calculate L1-norm for each vector
vectors_l1 = np.tile(vectors_l1, (1, 2))
normalized_vectors = np.divide(vectors, vectors_l1)
print(normalized_vectors)
```

```
[[0.67567568 0.32432432]
 [0.85714286 0.14285714]
 [0.57142857 0.42857143]
 [0.33333333 0.66666667]
 [0.66666667 0.33333333]
 [0.55555556 0.44444444]
 [0.31818182 0.68181818]
 [0.64516129 0.35483871]
 [0.5        0.5       ]
 [0.31034483 0.68965517]]
```

Step 2 – get your matplotlib skills ready and work towards the final results:

```
import csv # package for reading csv files
import numpy as np # package for dealing with numerical data
import pandas as pd # this is a data frame package
import matplotlib.pyplot as plt # package for plotting

data_frame = pd.read_csv('/Users/rockychen/Downloads/tutorial_data.csv')
all_watches = data_frame.loc[:, ['comedy movies', 'horror movies']]

vectors = all_watches.values # [comedy, horror] vector for all users
vectors_l1 = np.linalg.norm(vectors, ord = 1, axis = 1, keepdims = True) # calculate L1-norm for each vector
vectors_l1 = np.tile(vectors_l1, (1, 2))
normalized_vectors = np.divide(vectors, vectors_l1)

x_values, y_values = np.split(normalized_vectors, 2, axis = 1) # split the two dimensions of each vector
captions = data_frame['username'].astype(str).values # convert all usernames to strings for captioning
print(type(captions[0]))
cluster = plt.scatter(x_values, y_values, marker='x', color='red')
cluster = plt.gca()
for i, name in enumerate(captions):
    cluster.annotate(name, (x_values[i] + 0.01, y_values[i] + 0.01), size = 10) # place the captions for each point
plt.xlabel('interest on comedy movies')
plt.ylabel('interest on horror movies') # adding x/y labels to the plot
```

```
<class 'str'>

Text(0, 0.5, 'interest on horror movies')
```