# MT#1 Problem 0 Simulating Monty Hall Problem

## Exercise 0 Initialize the game

**Exercise 0** (2 points) Complete the function initialize\_game() which has 1 input argument:

def initialize\_game(n):

d={'Door ' + str(i): 'Goat' for i in range(1,n+1)}

d['Door ' + str(random.randint(1,n))]='Car'

return d

## Exercise 1 First choice (use random.randint)

**Exercise 1** (1 point) Complete the function first\_choice() which has 1 input argument:

* door\_dict: A dictionary representing every numbered door and what that door is hiding (a dictionary similar to that returned by function intitialize\_game(n))

def first\_choice(door\_dict):

import random

n=len(door\_dict)

return random.randint(1,n)

## Exercise 2 Monty opens door

**Exercise 2** (3 points) Complete the function monty\_opens() which has 3 input arguments.

* door\_dict: A dictionary similar to that returned by function initialize\_game()
* player\_door: An integer value denoting the player's first choice of door
* k: The number of doors to be opened by Monty

import random

def monty\_opens(door\_dict, player\_door, k):

assert type(door\_dict) is dict, "input variable door\_dict should be a dictionary"

assert k <= len(door\_dict) - 2, "Incorrect number of doors to be opened in dictionary"

pd='Door ' + str(player\_door)

car\_door=[k for k,val in door\_dict.items() if val=='Car']

d2=[k for k,val in door\_dict.items() if val!='Car' and k!=pd]

n=len(d2)

import numpy as np

keep\_doors=list(np.random.choice(d2,n-k,replace=False))

keep\_doors.append(pd)

keep\_doors.append(car\_door[0])

keep\_doors=list(set(keep\_doors))

d={}

for i in keep\_doors:

d[i]=door\_dict[i]

return d

## Exercise 3 Make final descision

**Exercise 3** (2 points) Complete the function final\_decision() which has 3 input arguments:

* door\_dict: A dictionary after Monty has opened k doors using monty\_opens() function
* player\_door: An integer value denoting player's first choice
* decision: A string value, either 'Stay' or 'Switch', depending on whether player decides to stay with first choice or switch to another door

def final\_decision(door\_dict, player\_door, decision):

import numpy as np

import random

pd='Door ' + str(player\_door)

if decision=='Stay':

if door\_dict[pd]=='Car':

return 'Won'

else:

return 'Lost'

if pd in door\_dict:

del door\_dict[pd]

choice=np.random.choice(list(door\_dict.keys()),1)[0]

if door\_dict[choice]=='Car':

return 'Won'

else:

return 'Lost'

## Exercise 4 Run the game

**Exercise 4** (2 points) Now, let's bring the pieces together and simulate a game. In this exercise, you will complete the given simulate\_game function that simulates Monty Hall game 10000 times. The function has 2 input arguments:

* n: Number of doors
* k: Number of doors to be opened be Monty
* decision : Either Stay or Switch

def simulate\_game(n, k, decision):

num\_won = 0 #Variable to keep a track of the number of times the player won

assert decision == 'Switch' or decision == 'Stay' , "Check the parameter input for decision variable"

num\_won=0

for i in range(10000):

#

door\_dict=initialize\_game(n)

player\_door=first\_choice(door\_dict)

door\_dict=monty\_opens(door\_dict,player\_door,k)

if final\_decision(door\_dict,player\_door,decision)=='Won':

num\_won+=1

#

return(num\_won)

# MT#1 Problem 1 Linear Optimization using Regex

## Exercise 0 Get regular

**Exercise 0** (3 points) Complete the getRegEx() function which returns a regular expression string to match any linear equations or inequations, such as the ones below. You may have any number of terms on the left hand side or the right hand side of the equation.

def getRegEx():

regex = ""

#

regex = '(?=.)([-+]\*)([\d]\*)([a-z]\*)([=|<=|>=]\*)'

return regex

## Exercise 1

## Exercise 2

## Exercise 3

# MT#1 Problem 2 Exponential smoothing

## Exercise 0

## Exercise 1

## Exercise 2

## Exercise 3

## Exercise 4

# MT#1 Problem 3 Floating-point arithmetic reloaded

## Exercise 0

## Exercise 1

## Exercise 2

## Exercise 3

## Exercise 4

# MT#1 Problem 4 Change Detection

## Exercise 0

## Exercise 1

## Exercise 2

## Exercise 3

## Exercise 4

# MT#2 Problem 0 Using pandas

## Exercise 1 Import tab-delimited file to panda dataframe

Exercise 1 (1 point) Given the filename, write a function to import the "data dictionary" tab-delimited file into a pandas dataframe.

def tab\_import(fname):

df=pd.read\_csv(fname, sep='\t')

return df

## Exercise 2 Add columns

Exercise 2 (1 point) Write a function that

1. Adds a column to the df called 'Coded' that contains Boolean values: False if the columns labeled '-2' through '107' are all NaN, and True otherwise
2. Splits the df into two new dataframes, one containing the rows that are 'Coded' and those that are not
3. Removes the 'Coded' column from each new dataframe, since it is no longer required

def split\_by\_coded(full\_df):

full\_df["Coded"]=~dd\_df.loc[:, '-2':'107' ].isna().all(1)

coded\_df=full\_df[full\_df["Coded"]==True]

not\_coded\_df=full\_df[full\_df["Coded"]==False]

coded\_df.drop("Coded",axis=1,inplace=True)

not\_coded\_df.drop("Coded",axis=1,inplace=True)

return coded\_df, not\_coded\_df

## Exercise 3 Melt and other transformations

Exercise 3 (2 points) Write a function that

1. Melts the coded dataframe columns '-2' to '107' into a column 'Code' containing the labels and a column 'Description' containing the code description
2. Removes all the rows in which 'Description' is NaN from the melted coded dataframe, since these rows add no information
3. Deletes the not\_coded dataframe columns '-2' to '107' and adds two columns, labeled 'Code' and 'Description', to the not\_coded dataframe, populating them with NaN's (use np.nan), since this dataframe had no codes to describe. It is recommended to use the pandas.DataFrame.assign() method. The .assign() method is useful for cleanly creating a new dataframe with new columns, optimized for "chaining" methods. <https://stackoverflow.com/questions/48177914/why-use-pandas-assign-rather-than-simply-initialize-new-column>
4. Concatenates the two new dataframes into a single dataframe
5. The columns should be ordered as **NAME OF DATA ELEMENT, dev-category, developer-friendly name, API data type, VARIABLE NAME, Code, Description**.

import numpy as np

def melt(df, col\_vals, key, value):

assert type(df) is pd.DataFrame

keep\_vars=df.columns.difference(col\_vals) #

melted\_sections=[]

for c in col\_vals: # go through all the column values.

melted\_c=df[keep\_vars].copy() # first time in loop: 1999

melted\_c[key]=c

melted\_c[value]=df[c]

melted\_sections.append(melted\_c)

melted = pd.concat(melted\_sections)

return melted

def melt\_together(coded\_df, not\_coded\_df):

cdf=coded\_df.copy()

ncdf=not\_coded\_df.copy()

cv=cdf.loc[0:0, '-2':'107' ].columns

mcdf=melt(cdf,cv,"Code","Description")

mask=mcdf.Description.notna()

mcdf=mcdf[mask]

ncdf.drop(cv,axis=1,inplace=True)

ncdf["Code"]=np.nan

ncdf["Description"]=np.nan

cols=['NAME OF DATA ELEMENT','dev-category','developer-friendly name','API data type','VARIABLE NAME','Code', 'Description']

frames=[mcdf,ncdf]

melted\_df=pd.concat(frames,sort=False)[cols]

return melted\_df

## Exercise 4 Return five element tuple

Exercise 4 (2 points) Now that you have the "data dictionary" in a nicer form, use it to write a function that returns a 5-element tuple containing:

1. The string value for the '**VARIABLE NAME**' that corresponds to the '**developer-friendly name**', 'name'. This is the name of the column that contains college names.
2. The string value for the '**VARIABLE NAME**' that corresponds to the '**developer-friendly name**', 'tuition.in\_state'. This is the name of the column that contains the in-state tuition amount for each college.
3. The string value for the '**VARIABLE NAME**' that corresponds to the '**developer-friendly name**', 'tuition.out\_of\_state'. This is the name of the column that contains the out-of-state tuition amount for each college.
4. The string value for the '**VARIABLE NAME**' that corresponds to the '**developer-friendly name**', 'program.bachelors.engineering'. This is the name of the column that contains all the colleges that offer a bachelor's degree in engineering.
5. The float value of the **Code** for a college that offers a bachelor's degree in engineering. Hence, the 'Description' corresponding to this data point must be 'Program offered'.

For example, suppose the given developer-friendly name is 'carnegie\_basic', the corresponding VARIABLE NAME is 'CCBASIC'.

def get\_dd\_info(dd):

name=dd[dd['developer-friendly name']=='name']["VARIABLE NAME"].astype('str').iloc[0]

in\_state=dd[dd['developer-friendly name']=='tuition.in\_state']["VARIABLE NAME"].iloc[0]

out\_of\_state=dd[dd['developer-friendly name']=='tuition.out\_of\_state']["VARIABLE NAME"].iloc[0]

a=dd[dd['developer-friendly name']=='program.bachelors.engineering']

b=a[a['Description'].str.contains("Program offered",regex=False)].iloc[0]

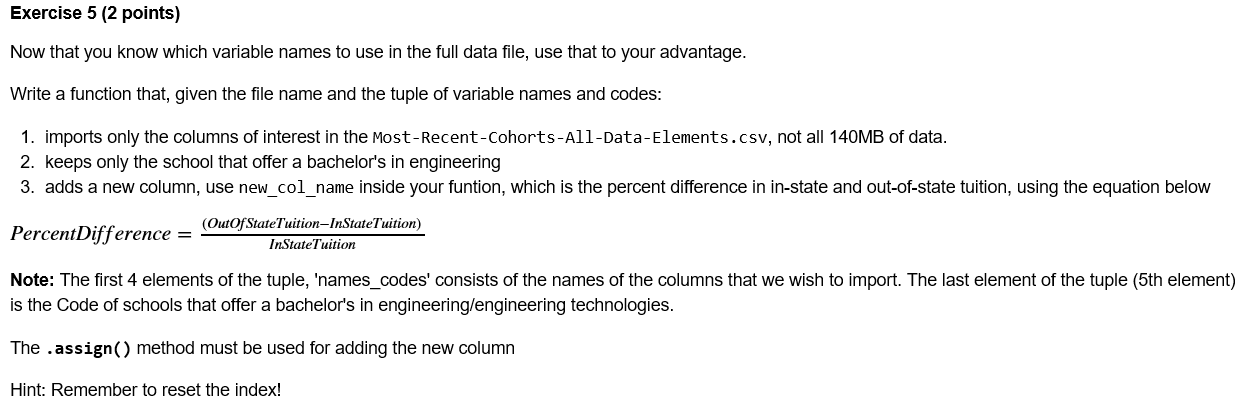
bach\_eng=b["VARIABLE NAME"]

offered=float(b["Code"])

#(name,in\_state,out\_of\_state,bach\_eng,offered)

return (name, in\_state, out\_of\_state, bach\_eng, offered)

## Exercise 5 Compute percent difference



def percent\_diff(fname, names\_codes, new\_col\_name):

cols=list(names\_codes)[0:4]

Code=list(names\_codes)[4]

data=pd.read\_csv(fname)[cols]

data=data[data[cols[3]]==Code]

perc\_diff\_df=data.assign(mydiff=lambda x: (x['TUITIONFEE\_OUT']-x['TUITIONFEE\_IN'])/x['TUITIONFEE\_IN'])

perc\_diff\_df.rename(columns={"mydiff":new\_col\_name},inplace=True)

perc\_diff\_df.reset\_index(drop=True,inplace=True)

return perc\_diff\_df

## Exercise 6 Clear up dataframe

Exercise 6 ( 2 points) For the final product, let's make this dataframe more user-friendly.

Write a function that

1. sorts descending by the tuition difference, tuition.difference
2. resets the index to provide a clear "ranking" without the old index being included in the dataframe
3. renames the columns using the data dictionary 'developer-friendly name' instead of the 'VARIABLE NAME'

def friendlify(unsorted\_df, sort\_column\_name, data\_dictionary):

friendly\_df=unsorted\_df.copy()

friendly\_df.sort\_values(by=sort\_column\_name,ascending=False,inplace=True)

friendly\_df.reset\_index(drop=True,inplace=True)

clu=dict(zip(data\_dictionary['VARIABLE NAME'],data\_dictionary['developer-friendly name']))

cols=list(friendly\_df)

for old\_col in cols[0:-1]:

new\_col=clu[old\_col]

friendly\_df.rename(columns={old\_col:new\_col},inplace=True)

cols=list(friendly\_df)

return friendly\_df

# MT#2 Problem 1 Visualization application on Clustering

## Exercise 0 Distance Cluster assignment

**Exercise 0:** First, we will implement the assign function, which assigns data points to its nearest center. For example, suppose data = [[101, 0], [99,1], [-100,2]], mu = [[100,0], [-100, 0]], then you would expect to return membership = [0, 0, 1]. If there are multiple centers that are equidistant to a point, pick the center with the lowest index, i.e., the first center in mu that is nearest to the point. (3 points)

def distance(dd,mm):

import math

d=np.array(dd)-np.array(mm)

return(math.sqrt(d.dot(d)))

def assign(data, mu):

# data: (N, 2) numpy.array

# mu: (K, 2) numpy.array

# return: (N,) numpy.array indicating class membership of each data point

membership = np.zeros(len(data))

for i,vi in enumerate(data):

for j,vj in enumerate(mu):

d=distance(vi,vj)

if j==0:

dmin=d

idx=0

elif d<dmin:

dmin=d

idx=j

membership[i]=idx

return membership

## Exercise 1: Get coordinates for the center of each class

def re\_estimate(data, membership):

# data: (N, 2) numpy.array

# membership: (N,) numpy.array indicating class membership of each data point

# mu: (K, 2) numpy.array

n=np.zeros(int(membership.max()+1))

sv=np.zeros(shape=(len(n),2))

for i,c in zip(data,membership):

idx=c.astype(int)

sv[idx:idx+1,0:1]+=i[0]

sv[idx:idx+1,1:2]+=i[1]

n[idx]+=1

for i in range(len(sv)):

sv[i,0]/=n[i]

sv[i,1]/=n[i]

mu=sv

return mu

## Exercise 2 Kernel

**Exercise 2** Implement K(z) based on the above kernel function. Return a numpy array with shape (500,)（2 points)

def kernel(z):

import math

from numpy import linalg as la

# z: (N, 2) numpy.array

# returns (N, 1) numpy.array

zz=np.zeros(shape=(len(z)))

for i,v in enumerate(z):

zz[i]=math.exp(-v.dot(v)/2)/math.sqrt(2\*np.pi)

return zz

## Exercise 3 KDE Function

**Exercise 3** Based on the KDE function:

*f*(*x*)=∑*i*=1*N*1*NK*(*x*−*xih*)f(x)=∑i=1N1NK(x−xih)

Compute the probablity density at [0,0], i.e., *f*((0,0))f((0,0)). This should return a **scalar** value. (2 points）

def compute(data, h):

n=len(data)

return sum(kernel(data1/h))/n

# MT#2 Problem 2 SQL NYPD Motor Vehicle Collisions

## Exercise 0 Sum injuries over a nested query

**Exercise 0** (2 points) Using the table **nypd**, here are some tasks to complete in this exercise.

1. Using the column, DATE, which is in the form mm/dd/yyyy, add a new column to the dataset called Month.
2. Then return the total number of PERSONS INJURED (total\_num\_of\_injured) in each month, by the descending order of Num\_of\_injured.

Save your result in a table named **new\_dataset**.

query='''

select month, sum(Num\_of\_Persons\_injured) as total\_num\_of\_injured

from (

select substr(DATE,1,2) as month, Num\_of\_Persons\_injured

from nypd)

group by month

order by total\_num\_of\_injured desc

'''

new\_dataset=pd.read\_sql\_query(query,conn)

## Exercise 1 Group by with a where clause, order by, and limit

**Exercise 1 (2 point):** Using the table **nypd**, here are some tasks to complete in this exercise. There are many contributing factors for the collisions. please list top 10 contributing factors (unspecified is not a factor). Save your result in a table named **top10\_factors**, by the descending order of cnt.

query='''

select "CONTRIBUTING FACTOR VEHICLE 1", count(\*) as cnt

from nypd

where "CONTRIBUTING FACTOR VEHICLE 1" <> "Unspecified"

group by "CONTRIBUTING FACTOR VEHICLE 1"

order by cnt desc

limit 10

'''

top10\_factors=pd.read\_sql\_query(query,conn)

## Exercise 2 Nested from clauses with Union all, multiple limits, order by

**Exercise 2 (3 points):** Using the table **nypd**, here are some tasks to complete in this exercise.

There are five boroughs in NYC, please list ONE location in QUEENS and ONE location in Bronx that has the highest number of collisions.((latitude, longitude) = (0,0) should not be chosen).

query='''

select \*

from (

select borough as borough, latitude, longitude, count(\*) as collisions

from nypd

where latitude<>0

and longitude<>0

and borough="BRONX"

group by borough, latitude, longitude

order by collisions desc

limit 1)

union all

select \*

from (

select borough as borough, latitude, longitude, count(\*) as collisions

from nypd

where latitude<>0

and longitude<>0

and borough="QUEENS"

group by borough, latitude, longitude

order by collisions desc

limit 1)

order by collisions desc

'''

top\_collisions=pd.read\_sql\_query(query,conn).drop(["collisions"],axis=1)

## Exercise 3 Complex Query

query='''

select LATITUDE,LONGITUDE,Zip\_code, num\_of\_collisions from (

select LATITUDE,LONGITUDE,Zip\_code, count(\*) as num\_of\_collisions, sum(injured) as injured

from (

select unique\_key, latitude, longitude, zip\_code

from nypd

where latitude<>-10

and longitude<>-10) as A

inner join

(select unique\_key, sum(injured) as injured

from (select unique\_key, Num\_of\_persons\_injured as injured

from Detail\_injured

where Num\_of\_persons\_injured is not NULL

union all

select unique\_key, Num\_of\_pedestrians\_injured as injured

from Detail\_injured

where Num\_of\_pedestrians\_injured is not NULL

)

group by unique\_key) as B

on A.unique\_key=B.unique\_key

group by A.LATITUDE,A.LONGITUDE,A.Zip\_code

order by injured desc

limit 10

)

order by num\_of\_collisions desc

'''

top\_ten\_locations=pd.read\_sql\_query(query,conn)

# MT#2 Problem 3 Geometric Matrix Operation

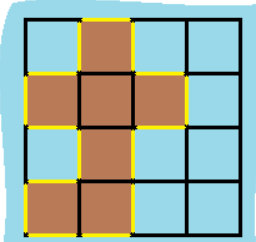
## Exercise 0 Find the perimeter

**Exercise 0** (3 points) Find the perimeter

Let's start off with a relatively simpler example. You are given a map in form of a two-dimensional integer grid where 1 represents land and 0 represents water. Grid cells are connected horizontally/vertically (not diagonally). The grid is completely surrounded by water, and there is exactly one island (i.e., one or more connected land cells).

The island doesn't have "lakes" (water inside that isn't connected to the water around the island). One cell is a square with side length 1. The grid is rectangular, width and height don't exceed 100. Determine the perimeter of the island.

Example: Input: [[0,1,0,0], [1,1,1,0], [0,1,0,0], [1,1,0,0]]



import numpy as np

def islandPerimeter(grid):

A=np.pad(grid,pad\_width=1, mode='constant',constant\_values=0)

shore=0

for (r,c), v in np.ndenumerate(A):

if v==1:

if A[r,c-1]==0:

shore+=1

if A[r,c+1]==0:

shore+=1

if A[r-1,c]==0:

shore+=1

if A[r+1,c]==0:

shore+=1

return shore

## Exercise 1 Mimimum volume of ellipsoid

**Exercise 1:** This problem is concerned with computing the minimum volume covering ellipsoid (MVCE) in 2-dimensional space. Suppose we have a set of points stored in matrix P, we want to find the “smallest” ellipsoid that contains all data points. An example of the ellipse is shown below:

random.seed(10)

P = np.reshape([random.random()\*10 for i in range(20)],(10,2)) #Let's start off by generating an matrix with 20 entries

(N, d) = np.shape(P) #We use N and d to extract the row and column of the matrix

d = float(d)

tolerance = 0.001

# Q will be our working array

Q = np.vstack([np.copy(P.T), np.ones(N)])

QT = Q.T

# initializations

err = 1.0 + tolerance

u = (1.0 / N) \* np.ones(N)

# Khachiyan Algorithm

while err > tolerance:

V = np.dot(Q, np.dot(np.diag(u), QT))

M = np.diag(np.dot(QT , np.dot(linalg.inv(V), Q))) # M the diagonal vector of an NxN matrix

j = np.argmax(M)

maximum = M[j]

step\_size = (maximum - d - 1.0) / ((d + 1.0) \* (maximum - 1.0))

new\_u = (1.0 - step\_size) \* u

new\_u[j] += step\_size

err = np.linalg.norm(new\_u - u)

u = new\_u

**Exercise 1** (2 points) In order to find the center of the ellipse, you need to find the dot product between **P\_transpose** and u. (Hint: use the built-in function in numpy for computation)

def find\_center(P\_transpose, u,):

return P\_transpose.dot(u)

## Exercise 2 Find Radius

# the A matrix for the ellipse

center = find\_center(P.T, u)

A = linalg.inv(

np.dot(P.T, np.dot(np.diag(u), P)) -

np.array([[a \* b for b in center] for a in center])

) / d

# Get the values we'd like to return

U, s, rotation = linalg.svd(A)

def find\_radius(s):

return 1/np.sqrt(s)

# MT#2 Problem 4 Recommender system – Movies

## Exercise 1 Create summary measures. Use panda group by.

Excercise 1 (1 points) In this exercise, you will compute the mean rating and adjusted rating for each movie by a particular user. Then final result will be a table named "adjusted\_ratings" with these columns: userId, movieId, rating, title, genres, rating\_mean, rating\_adjusted.

# Remember to replace 0 adjusted rating values to 1\*e-6 in order to avoid 0 denominator

rating\_mean=full\_ratings[['movieId', 'rating']].groupby(['movieId'],as\_index=False).mean()

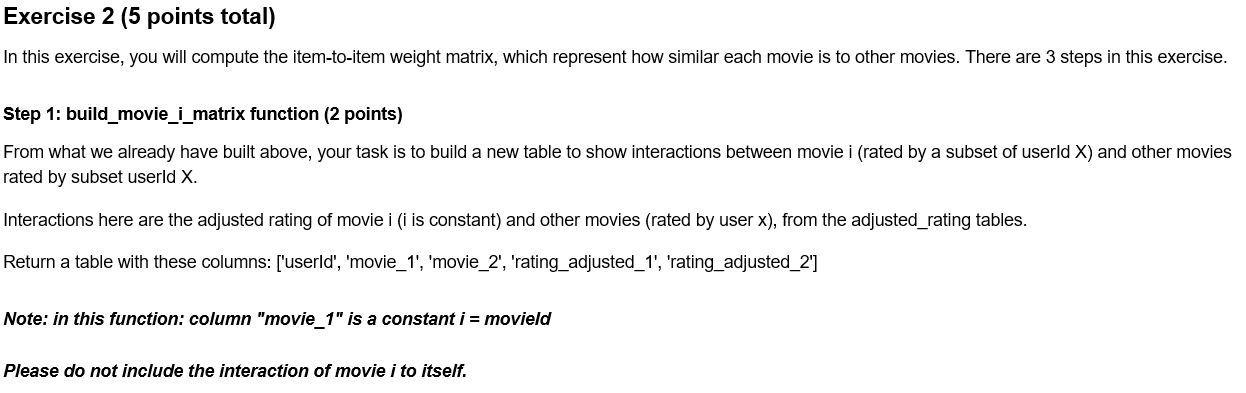
rating\_mean.rename(columns={"rating":"rating\_mean"},inplace=True)

adjusted\_ratings=full\_ratings.merge(rating\_mean,on="movieId")

adjusted\_ratings["rating\_adjusted"]=adjusted\_ratings.rating-adjusted\_ratings.rating\_mean

adjusted\_ratings.rating\_adjusted[adjusted\_ratings["rating\_adjusted"]==0]=1e-6

## Exercise 2: item-to-item weight matrix



def build\_movie\_i\_matrix(movie\_i):

# Columns used in the output format

used\_cols = ['userId', 'movie\_1', 'movie\_2', 'rating\_adjusted\_1', 'rating\_adjusted\_2']

# Subset userId A rated movie i

userids=pd.DataFrame(adjusted\_ratings[adjusted\_ratings['movieId']==movie\_i]['userId'].unique(),columns=["userId"])

cols=['userId', 'movieId', 'rating\_adjusted']

usermovies=adjusted\_ratings.merge(userids,on="userId")[cols]

same=usermovies[usermovies['movieId']==movie\_i].rename(columns={"movieId": "movie\_1","rating\_adjusted":"rating\_adjusted\_1"})

diff=usermovies[usermovies['movieId']!=movie\_i].rename(columns={"movieId": "movie\_2","rating\_adjusted":"rating\_adjusted\_2"})

used\_cols = ['userId', 'movie\_1', 'movie\_2', 'rating\_adjusted\_1', 'rating\_adjusted\_2']

movie\_i\_table=same.merge(diff,on="userId")

return movie\_i\_table[used\_cols]

def similarity\_score(m\_i\_matrix, movie\_2):

from numpy.linalg import norm

m1\_m2\_matrix = m\_i\_matrix[m\_i\_matrix['movie\_2'] == movie\_2]

data=m\_i\_matrix[m\_i\_matrix.movie\_2==movie\_2][['rating\_adjusted\_1','rating\_adjusted\_2']].values

A=data[:,0]

B=data[:,1]

similar\_score=A.dot(B)/(norm(A)\*norm(B))

return similar\_score

## Exercise 3

Step 3: (2 points)[¶](https://proxy.vocareum.com/hostip/172.31.20.94:6000/user/ccc_v1_w_MGQ1N_87933_207423_207423_46124_0/notebooks/problem4.ipynb#Step-3:-(2-points))

Your Task: Build a weight matrix table that shows the similarity score between movies[¶](https://proxy.vocareum.com/hostip/172.31.20.94:6000/user/ccc_v1_w_MGQ1N_87933_207423_207423_46124_0/notebooks/problem4.ipynb#Your-Task:-Build-a-weight-matrix-table-that-shows-the-similarity-score-between-movies)

Notes: do not include the score of that movie to itself

Have a look at our sample table below.

def create\_weight\_matrix(adjusted\_ratings):

# define weight matrix

matrix\_cols = ['movie\_1', 'movie\_2', 'weight']

#w\_matrix = pd.DataFrame(columns = matrix\_cols)

# list all distinct movies in dataset

distinct\_mv = np.unique(adjusted\_ratings.movieId)

distinct\_mv1 = np.unique(adjusted\_ratings.movieId)

data = list()

for mid1 in distinct\_mv1:

movies=build\_movie\_i\_matrix(mid1)

distinct\_mv2 = np.unique(movies.movie\_2)

for mid2 in distinct\_mv2:

sim\_score=similarity\_score(movies, mid2)

data.append([mid1,mid2,sim\_score])

w\_matrix=pd.DataFrame(data,columns = matrix\_cols)

return w\_matrix

def predict\_rating(userId, movieId ,w\_matrix):

# rating\_mean for each movie

rating\_mean = full\_ratings.groupby(['movieId'], as\_index = False).mean().rename(columns = {'rating': 'rating\_mean'})[['movieId','rating\_mean']]

# set mean\_rating = 2.5 if that movie has not been rated by any user.

if rating\_mean[rating\_mean['movieId'] == movieId].shape[0] > 0:

mean\_rating = rating\_mean[rating\_mean['movieId'] == movieId]['rating\_mean'].iloc[0]

else:

mean\_rating = 2.5

Iu=pd.DataFrame(np.unique(full\_ratings[full\_ratings["userId"]==userId][["movieId"]]),columns=["movieId"])

userid\_ratings=full\_ratings[full\_ratings["userId"]==userId]

userid\_ratings=userid\_ratings.groupby(['movieId'], as\_index = False).mean().rename(columns = {'rating': 'ruj'})[['movieId','ruj']]

rating\_mean = full\_ratings.groupby(['movieId'], as\_index = False).mean().rename(columns = {'rating': 'rating\_mean'})[['movieId','rating\_mean']]

userid\_ratings=userid\_ratings.merge(rating\_mean,on="movieId").rename(columns={"rating\_mean":"uj"})

wgt=w\_matrix[w\_matrix["movie\_1"]==movieId][["movie\_2","weight"]].rename(columns={"movie\_2":"movieId","weight":"wij"})

userid\_ratings=userid\_ratings.merge(wgt,on="movieId")

userid\_ratings=userid\_ratings.assign(awj=lambda x: np.abs(x.wij))

userid\_ratings=userid\_ratings.assign(product=lambda x: (x.ruj-x.uj)\*x.wij)

svars=userid\_ratings[["awj","product"]].sum(axis=0)

denominator=svars[0]

numerator=svars[1]

if denominator==0:

predicted\_rating=mean\_rating

else:

predicted\_rating=mean\_rating+numerator/denominator

# Calculate predicted\_rating for movieId by userId (based on w\_matrix and formula given above)

# Note: In case the denominator == 0 ( sum of absolute weight\_ij == 0 ):

# => s(i;u) = Mean\_i ~ predicted\_rating = mean\_rating

return predicted\_rating

Exercise 4: Make Recommendation (2 point)

def recommend(userId, w\_matrix, adjusted\_ratings, top\_n=10):

distinct\_mv = adjusted\_ratings['movieId'].unique()

user\_rating\_full = pd.DataFrame(columns=['movieId', 'rating'])

# Loop through all distinct\_mv and calculate the ratings for all movies that the user hasn't rated

# Store these predicted ratings to user\_rating\_full dataframe (only not-yet-rated movies from user "userId")

# Because we only want to recommend top\_n movies with highest predicted ratings, that the user "userId" has not watched/rated.

# We will show top\_n items from your user\_rating\_full table to user as our recommendation

###

### YOUR CODE HERE

###

# select top 10 movies rated by the user

usermovies=adjusted\_ratings[adjusted\_ratings['userId']==userId][['movieId']]

watched=set(usermovies.movieId.unique())

user\_rating=list()

for mvid in distinct\_mv:

if mvid not in watched:

rating=predict\_rating(userId, mvid ,w\_matrix)

user\_rating.append([mvid,rating])

user\_rating\_full = pd.DataFrame(user\_rating,columns=['movieId', 'rating'])

recommendations = user\_rating\_full.sort\_values(by=['rating'], ascending=False).head(10)

recommendations["movieId"] = recommendations["movieId"].astype("int64")

recommendations = recommendations.merge(movies, on="movieId",how="left")

return recommendations