Coverage for **recommendation_engine.py** : 85%

111 statements | 94 run | 17 missing | 0 excluded

```
1  import pytest
2  import unittest
3  import pandas as pd
4  import sys
5  import os
6
7  import numpy as np
8  import pandas as pd
9  from collections import defaultdict
10 from numpy import dot
11 from numpy.linalg import norm
12
13 #dummy data frame for testing
14 dummy_df = {'event_time':  ['test','test','test','test','test'],
15         'event_type': ['view','cart','purchase','view','cart'],
16         'product_id': [1,2,3,2,1],
17         'category_id': [101,102,103,102,101],
18         'category_code': ['electronics','kids','kitchen','kids','electronics'],
19         'brand': ['brand1','brand2','brand3','brand2','brand1'],
20         'price': [100,101,102,101,100],
21         'user_id': [1,2,3,1,1],
22         'user_session': ['test','test','test','test','test']
23         }
24
25 dummy_df = pd.DataFrame (dummy_df, columns = list(dummy_df.keys()))
26
27 #Pre-processing part
28 #Unique user and item
29 unique_user = set()
30 unique_item = set()
31 user_to_idx = {}
32
33 user_to_item = defaultdict(set)
34 item_to_user = defaultdict(set)
35
36 itemid_to_category_code = {}
37 itemid_to_brand = {}
38 item_history = defaultdict(dict)
39
40 #Loop through all the rows and create a unique user and item set
41 idx = 0
42 for i, row in dummy_df.iterrows():
43     userid,itemid,categorycode,brand,event_type,price = row[7],row[2],row[4],row[5],row[1],row[6]
44     unique_user.add(userid)
45     unique_item.add(itemid)
46
47     if userid not in user_to_idx:
48         user_to_idx[userid] = idx
49         idx+=1
50
51     if itemid not in item_history:
52         item_history[itemid]['view']=0
53         item_history[itemid]['purchase']=0
54         item_history[itemid]['cart']=0
55         item_history[itemid]['price']=price
56
57     user_to_item[userid].add(itemid)
58     item_to_user[itemid].add(userid)
59     itemid_to_category_code[itemid] = categorycode
60     itemid_to_brand[itemid] = brand
61     item_history[itemid][event_type]+=1
62
63 def OneHotEncode(item):
64   '''
65   Returns one-hot encoded vector for a given item ID indicating the purchase history of a user
66   :param item: item ID of the item
67   :type item: int
68   :return: one hot encoded vector
69   :rtype: list[int]
```

```
    '''
    vector = [0 for _ in range(len(unique_user))]

    for user in item_to_user[item]:
        vector[user_to_idx[user]]=1

    return vector


def CalcScore_cosine(item1, item2):
    '''
    Calculates the cosine similirity between two item IDs. Uses OneHotEncode helper function to retrieve one hot encodings for itemIDs
    :param item1: item ID of item1
    :type item1: int
    :param item2: item ID of item2
    :type item2: int
    :return: Cosine Similarity Score
    :rtype: float
    '''
    a,b = OneHotEncode(item1), OneHotEncode(item2)

    cos_sim = dot(a, b)/(norm(a)*norm(b))
    return cos_sim

def jaccard(s1, s2):
    '''
    Calculates the cosine similirity between two input vectors.
    :param s1: first input vector
    :type s1: list[int]
    :param s2: second input vector
    :type s2: list[int]
    :return: Jaccard Similarity Score
    :rtype: float
    '''
    numer = len(s1.intersection(s2))
    denom = len(s1.union(s2))
    return numer/denom

def findNearestItem(itemid):
    '''
    Returns a list of itemIDs of the most similiar 50 items to the given input item.
    :param itemid: item ID of the item
    :type itemid: int
    :return: list of itemIDs of closest similar items
    :rtype: list[int]
    '''

    maxSimilarityScore = float('-inf')
    ClosestItem = None
    candidateItems = set()
    users = item_to_user[itemid]
    similarities = []

    #reduce the search space of the candidate items
    for u in users:
        candidateItems = candidateItems.union(user_to_item[u])

    for item in candidateItems:
        if item==itemid:
            continue

        # score = CalcScore_cosine(item,itemid)
        score = jaccard(users, item_to_user[item])
        if score==float('nan'):
            continue

        similarities.append((score, item))
    similarities.sort(reverse=True)

    return similarities[:50]

def find_Nearest_User(userid, item_to_user, user_to_item, top=10):
    '''
    Returns a list of UserIDs of the most similiar users to the given input user.
    :param userid: item ID of the item
```

```
    :type itemid: int
    :param item_to_user: dict which stores item to user list information
    :type item_to_user: dict
    :param user_to_item: dict which stores user to item list information
    :type user_to_item: dict
    :param top: parameter to control number of returned similar users
    :type top: int
    :return: list of userIDs of closest similar users
    :rtype: list[int]
    '''

    maxSimilarityScore = float('-inf')
    candidateUsers = set()
    items = user_to_item[userid]
    similarities = []

    #reduce the search space of the candidate items
    for i in items:
        candidateUsers = candidateUsers.union(item_to_user[i])

    for user in candidateUsers:
        if user==userid:
            continue

        # score = CalcScore_cosine(item,itemid)
        score = jaccard(items, user_to_item[user])
        if score==float('nan'):
            continue

        similarities.append((score, user))
    similarities.sort(reverse=True)

    return similarities[:top]



userhistory_price = dummy_df[['price', 'user_id']]
userhistory_event = dummy_df[['event_type', 'user_id']]

one_hot_event = pd.get_dummies(userhistory_event['event_type'])
userhistory_event = userhistory_event.drop('event_type',axis = 1)
userhistory_event = userhistory_event.join(one_hot_event)

price_df = userhistory_price.groupby(by='user_id').sum()
event_df = userhistory_event.groupby(by='user_id').sum()
userHistory = price_df.join(event_df)

def findNearestUsersfromItem(target_item, item_to_user, user_to_item):
    '''
    Returns a list of potential buyer list for the target item.

    :param target_item:  itemID of the item
    :type target_item: int
    :param item_to_user: dict which stores item to user list information
    :type item_to_user: dict
    :param user_to_item: dict which stores user to item list information
    :type user_to_item: dict
    :return: list of potential buyer userIDs
    :rtype: list[int]
    '''
    target_users = set()
    purchased_users = item_to_user[target_item]

    for user in purchased_users:
        set_users = set(list(zip(*find_Nearest_User(user, item_to_user, user_to_item)))[1])
        new_users = set_users.difference(purchased_users)
        target_users = target_users.union(new_users)

    target_users = list(target_users)
    views = []
    purchases = []
    carts = []
    amount = []
    for userid in target_users:
        views.append(userHistory.loc[userid, 'view'])
```

```
220         purchases.append(userHistory.loc[userid, 'purchase'])
221         carts.append(userHistory.loc[userid, 'cart'])
222         amount.append(userHistory.loc[userid, 'price'])
223 
224     target_df = pd.DataFrame(data={ 'Target users': target_users,
225                                     '#Views': views,
226                                     '#Purchases': purchases,
227                                     '#AddedToCart': carts,
228                                     'Amount Spent': amount})
229 
230 
231     return target_df
```