Starbucks_Capstone_notebook

May 29, 2021

1 Starbucks Capstone Challenge

1.0.1 Introduction

This data set contains simulated data that mimics customer behavior on the Starbucks rewards mobile app. Once every few days, Starbucks sends out an offer to users of the mobile app. An offer can be merely an advertisement for a drink or an actual offer such as a discount or BOGO (buy one get one free). Some users might not receive any offer during certain weeks.

Not all users receive the same offer, and that is the challenge to solve with this data set.

Your task is to combine transaction, demographic and offer data to determine which demographic groups respond best to which offer type. This data set is a simplified version of the real Starbucks app because the underlying simulator only has one product whereas Starbucks actually sells dozens of products.

Every offer has a validity period before the offer expires. As an example, a BOGO offer might be valid for only 5 days. You'll see in the data set that informational offers have a validity period even though these ads are merely providing information about a product; for example, if an informational offer has 7 days of validity, you can assume the customer is feeling the influence of the offer for 7 days after receiving the advertisement.

You'll be given transactional data showing user purchases made on the app including the timestamp of purchase and the amount of money spent on a purchase. This transactional data also has a record for each offer that a user receives as well as a record for when a user actually views the offer. There are also records for when a user completes an offer.

Keep in mind as well that someone using the app might make a purchase through the app without having received an offer or seen an offer.

1.0.2 Example

To give an example, a user could receive a discount offer buy 10 dollars get 2 off on Monday. The offer is valid for 10 days from receipt. If the customer accumulates at least 10 dollars in purchases during the validity period, the customer completes the offer.

However, there are a few things to watch out for in this data set. Customers do not opt into the offers that they receive; in other words, a user can receive an offer, never actually view the offer, and still complete the offer. For example, a user might receive the "buy 10 dollars get 2 dollars off offer", but the user never opens the offer during the 10 day validity period. The customer spends 15 dollars during those ten days. There will be an offer completion record in the data set; however, the customer was not influenced by the offer because the customer never viewed the offer.

1.0.3 Cleaning

This makes data cleaning especially important and tricky.

You'll also want to take into account that some demographic groups will make purchases even if they don't receive an offer. From a business perspective, if a customer is going to make a 10 dollar purchase without an offer anyway, you wouldn't want to send a buy 10 dollars get 2 dollars off offer. You'll want to try to assess what a certain demographic group will buy when not receiving any offers.

1.0.4 Final Advice

Because this is a capstone project, you are free to analyze the data any way you see fit. For example, you could build a machine learning model that predicts how much someone will spend based on demographics and offer type. Or you could build a model that predicts whether or not someone will respond to an offer. Or, you don't need to build a machine learning model at all. You could develop a set of heuristics that determine what offer you should send to each customer (i.e., 75 percent of women customers who were 35 years old responded to offer A vs 40 percent from the same demographic to offer B, so send offer A).

2 Data Sets

The data is contained in three files:

- portfolio.json - containing offer ids and meta data about each offer (duration, type, etc.)
- profile.json - demographic data for each customer
- transcript.json - records for transactions, offers received, offers viewed, and offers completed

Here is the schema and explanation of each variable in the files:

portfolio.json * id (string) - offer id * offer_type (string) - type of offer ie BOGO, discount, informational * difficulty (int) - minimum required spend to complete an offer * reward (int) - reward given for completing an offer * duration (int) - time for offer to be open, in days * channels (list of strings)

profile.json * age (int) - age of the customer * became_member_on (int) - date when customer created an app account * gender (str) - gender of the customer (note some entries contain 'O' for other rather than M or F) * id (str) - customer id * income (float) - customer's income

transcript.json * event (str) - record description (ie transaction, offer received, offer viewed, etc.) * person (str) - customer id * time (int) - time in hours since start of test. The data begins at time t=0 * value - (dict of strings) - either an offer id or transaction amount depending on the record

2.1 Problem

The problem that I chose to solve in this project focused on building a model that predicts the optimal offer should be sent to a customer and which offer got the best sales.

```
In [1]: #Import libs
        import pandas as pd
        import numpy as np
        import math
```

```
import json
from sklearn.preprocessing import MultiLabelBinarizer
import datetime
import os
import seaborn as sns
import pickle
%matplotlib inline
from matplotlib import pyplot as plt
import warnings
warnings.filterwarnings('ignore')
```

2.1.1 1. Read Dataset

In [2]:
```
# read in the json files
portfolio = pd.read_json('data/portfolio.json', orient='records', lines=True)
profile = pd.read_json('data/profile.json', orient='records', lines=True)
transcript = pd.read_json('data/transcript.json', orient='records', lines=True)
```

2.1.2 2. Data understanding and cleaning

A. Portfolio

In [3]:
```
portfolio
```

Out[3]:

	channels	difficulty	duration	\
0	[email, mobile, social]	10	7	
1	[web, email, mobile, social]	10	5	
2	[web, email, mobile]	0	4	
3	[web, email, mobile]	5	7	
4	[web, email]	20	10	
5	[web, email, mobile, social]	7	7	
6	[web, email, mobile, social]	10	10	
7	[email, mobile, social]	0	3	
8	[web, email, mobile, social]	5	5	
9	[web, email, mobile]	10	7	

	id	offer_type	reward
0	ae264e3637204a6fb9bb56bc8210ddfd	bogo	10
1	4d5c57ea9a6940dd891ad53e9dbe8da0	bogo	10
2	3f207df678b143eea3cee63160fa8bed	informational	0
3	9b98b8c7a33c4b65b9aebfe6a799e6d9	bogo	5
4	0b1e1539f2cc45b7b9fa7c272da2e1d7	discount	5
5	2298d6c36e964ae4a3e7e9706d1fb8c2	discount	3
6	fafdcd668e3743c1bb461111dcafc2a4	discount	2
7	5a8bc65990b245e5a138643cd4eb9837	informational	0
8	f19421c1d4aa40978ebb69ca19b0e20d	bogo	5
9	2906b810c7d4411798c6938adc9daaa5	discount	2

In [4]:
```
portfolio.shape
```

```
Out[4]: (10, 6)

In [5]: portfolio.describe()

Out[5]:        difficulty   duration     reward
        count   10.000000  10.000000  10.000000
        mean     7.700000   6.500000   4.200000
        std      5.831905   2.321398   3.583915
        min      0.000000   3.000000   0.000000
        25%      5.000000   5.000000   2.000000
        50%      8.500000   7.000000   4.000000
        75%     10.000000   7.000000   5.000000
        max     20.000000  10.000000  10.000000

In [6]: #Check missing Values
        print(portfolio.isnull().sum().sum())

0


In [7]: #check type of offers
        portfolio.offer_type.unique()

Out[7]: array(['bogo', 'informational', 'discount'], dtype=object)

In [8]: #check duplicate rows
        portfolio[portfolio.duplicated(['id'], keep=False)]

Out[8]: Empty DataFrame
        Columns: [channels, difficulty, duration, id, offer_type, reward]
        Index: []

In [9]: #Rename col "id" to "offer_id"
        portfolio.rename(columns={'id': 'offer_id'}, inplace=True)

In [10]: def encod_portfolio(portfolio):
             '''
             Encoding portfolio data to required format for applying analyses

             Input: portfolio data

             Output: df_portfolio - encoded data

             '''
             df_portfolio = portfolio.copy()
             #encoding channels
             mlb = MultiLabelBinarizer()
             df_portfolio = df_portfolio.join(pd.DataFrame(mlb.fit_transform(df_portfolio.pop('c
                                    columns = mlb.classes_,
```

```
                                index = df_portfolio.index))
        #encoding offer_type
        offer_type = pd.get_dummies(df_portfolio['offer_type'])
        # drop the offer_type column
        df_portfolio.drop(['offer_type'], axis=1, inplace=True)
        #Transform duration from day to hour
        df_portfolio['duration'] = df_portfolio['duration'] * 24
        # Merge portfolio and offer_type
        df_portfolio = pd.concat([df_portfolio, offer_type], axis=1, sort=False)

        return df_portfolio
```

In [11]:
```
df_portfolio = encod_portfolio(portfolio)
df_portfolio.head()
```

Out[11]:

	difficulty	duration	offer_id	reward	email	mobile	social	web	bogo	discount	informational
0	10	168	ae264e3637204a6fb9bb56bc8210ddfd	10	1	1	1	0	1	0	0
1	10	120	4d5c57ea9a6940dd891ad53e9dbe8da0	10	1	1	1	1	1	0	0
2	0	96	3f207df678b143eea3cee63160fa8bed	0	1	1	0	1	0	0	1
3	5	168	9b98b8c7a33c4b65b9aebfe6a799e6d9	5	1	1	0	1	1	0	0
4	20	240	0b1e1539f2cc45b7b9fa7c272da2e1d7	5	1	0	0	1	0	1	0

B. Profile

In [12]:
```
profile.head(10)
```

Out[12]:

	age	became_member_on	gender	id	income
0	118	20170212	None	68be06ca386d4c31939f3a4f0e3dd783	NaN
1	55	20170715	F	0610b486422d4921ae7d2bf64640c50b	112000.0
2	118	20180712	None	38fe809add3b4fcf9315a9694bb96ff5	NaN
3	75	20170509	F	78afa995795e4d85b5d9ceeca43f5fef	100000.0
4	118	20170804	None	a03223e636434f42ac4c3df47e8bac43	NaN
5	68	20180426	M	e2127556f4f64592b11af22de27a7932	70000.0
6	118	20170925	None	8ec6ce2a7e7949b1bf142def7d0e0586	NaN
7	118	20171002	None	68617ca6246f4fbc85e91a2a49552598	NaN
8	65	20180209	M	389bc3fa690240e798340f5a15918d5c	53000.0
9	118	20161122	None	8974fc5686fe429db53ddde067b88302	NaN

In [13]:
```
profile.shape
```

Out[13]:
```
(17000, 5)
```

```
In [14]: #Rename col "id" to "customer_id"
         profile.rename(columns={'id': 'customer_id'}, inplace=True)

In [15]: profile.describe()
```

Out[15]:

	age	became_member_on	income
count	17000.000000	1.700000e+04	14825.000000
mean	62.531412	2.016703e+07	65404.991568
std	26.738580	1.167750e+04	21598.299410
min	18.000000	2.013073e+07	30000.000000
25%	45.000000	2.016053e+07	49000.000000
50%	58.000000	2.017080e+07	64000.000000
75%	73.000000	2.017123e+07	80000.000000
max	118.000000	2.018073e+07	120000.000000

```
In [16]: #check duplicate clients
         profile['customer_id'].nunique()
```

Out[16]: 17000

```
In [17]: profile.head()
```

Out[17]:

	age	became_member_on	gender	customer_id	income
0	118	20170212	None	68be06ca386d4c31939f3a4f0e3dd783	NaN
1	55	20170715	F	0610b486422d4921ae7d2bf64640c50b	112000.0
2	118	20180712	None	38fe809add3b4fcf9315a9694bb96ff5	NaN
3	75	20170509	F	78afa995795e4d85b5d9ceeca43f5fef	100000.0
4	118	20170804	None	a03223e636434f42ac4c3df47e8bac43	NaN

No duplicates are found among customrs clounm

```
In [18]: #Check missing values by col
         profile.isnull().sum()

Out[18]: age                 0
         became_member_on    0
         gender           2175
         customer_id         0
         income           2175
         dtype: int64
```

The columns gender and income include missing values. Thus we will proceed by cleaning the data

```
In [19]: def profile_clean(profile):
             '''
             Cleaning the profile data

             Iuput: profile - data to proceed cleaning
```

```
        Output: profile - cleaned profile data

        '''
        #clean age missing values
        profile['age'] = profile['age'].apply(lambda x: np.nan if x == 118 else x)

        #drop all missing values
        profile.dropna(inplace=True)

        #change date format in col became_member_on
        profile.became_member_on = profile['became_member_on'].astype(str).astype('datetime
        # add col days_member
        profile['days_member'] = datetime.datetime.today().date() - pd.to_datetime(profile[
        profile['days_member'] = profile['days_member'].dt.days


        return profile
```

```
In [20]: df_profile = profile_clean(profile)
```

```
In [21]: df_profile.head()
```

Out[21]:

	age	became_member_on	gender	customer_id	income	days_member
1	55.0	2017-07-15	F	0610b486422d4921ae7d2bf64640c50b	112000.0	1414
3	75.0	2017-05-09	F	78afa995795e4d85b5d9ceeca43f5fef	100000.0	1481
5	68.0	2018-04-26	M	e2127556f4f64592b11af22de27a7932	70000.0	1129
8	65.0	2018-02-09	M	389bc3fa690240e798340f5a15918d5c	53000.0	1205
12	58.0	2017-11-11	M	2eeac8d8feae4a8cad5a6af0499a211d	51000.0	1295

```
In [ ]:
```

C. Transcript

```
In [22]: transcript.head(10)
```

Out[22]:

	event	person	time
0	offer received	78afa995795e4d85b5d9ceeca43f5fef	0
1	offer received	a03223e636434f42ac4c3df47e8bac43	0
2	offer received	e2127556f4f64592b11af22de27a7932	0
3	offer received	8ec6ce2a7e7949b1bf142def7d0e0586	0
4	offer received	68617ca6246f4fbc85e91a2a49552598	0
5	offer received	389bc3fa690240e798340f5a15918d5c	0

```
6  offer received  c4863c7985cf408faee930f111475da3     0
7  offer received  2eeac8d8feae4a8cad5a6af0499a211d     0
8  offer received  aa4862eba776480b8bb9c68455b8c2e1     0
9  offer received  31dda685af34476cad5bc968bdb01c53     0

                                                  value
0  {'offer id': '9b98b8c7a33c4b65b9aebfe6a799e6d9'}
1  {'offer id': '0b1e1539f2cc45b7b9fa7c272da2e1d7'}
2  {'offer id': '2906b810c7d4411798c6938adc9daaa5'}
3  {'offer id': 'fafdcd668e3743c1bb461111dcafc2a4'}
4  {'offer id': '4d5c57ea9a6940dd891ad53e9dbe8da0'}
5  {'offer id': 'f19421c1d4aa40978ebb69ca19b0e20d'}
6  {'offer id': '2298d6c36e964ae4a3e7e9706d1fb8c2'}
7  {'offer id': '3f207df678b143eea3cee63160fa8bed'}
8  {'offer id': '0b1e1539f2cc45b7b9fa7c272da2e1d7'}
9  {'offer id': '0b1e1539f2cc45b7b9fa7c272da2e1d7'}
```

```
In [23]: transcript.event.unique()
```

```
Out[23]: array(['offer received', 'offer viewed', 'transaction', 'offer completed'], dtype=objec
```

```
In [24]: transcript.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 306534 entries, 0 to 306533
Data columns (total 4 columns):
event     306534 non-null object
person    306534 non-null object
time      306534 non-null int64
value     306534 non-null object
dtypes: int64(1), object(3)
memory usage: 9.4+ MB
```

```
In [25]: #Rename col "person" to "customer_id"
         transcript.rename(columns={'person': 'customer_id'}, inplace=True)
```

```
In [26]: def transcript_clean(transcript):
             '''
             Cleaning the transcript data
             Iuput: transcript - data to proceed cleaning
             Output: df_offers - cleaned transcript data
             '''
             #extract offer type
             df_offer = transcript[transcript['value'].apply(lambda x: True if ('offer id' in x)
             # extract the offer id from value column
             df_offer['offer_id'] = df_offer['value'].apply(lambda x: x['offer id'] if ('offer i

             return df_offer
```

```
In [27]: df_offer = transcript_clean(transcript)

In [28]: df_offer.head()
```

Out[28]:

	event	customer_id	time	value	offer_id
0	offer received	78afa995795e4d85b5d9ceeca43f5fef	0	{'offer id': '9b98b8c7a33c4b65b9aebfe6a799e6d9'}	9b98b8c7a33c4b65b9aebfe6a799e6d9
1	offer received	a03223e636434f42ac4c3df47e8bac43	0	{'offer id': '0b1e1539f2cc45b7b9fa7c272da2e1d7'}	0b1e1539f2cc45b7b9fa7c272da2e1d7
2	offer received	e2127556f4f64592b11af22de27a7932	0	{'offer id': '2906b810c7d4411798c6938adc9daaa5'}	2906b810c7d4411798c6938adc9daaa5
3	offer received	8ec6ce2a7e7949b1bf142def7d0e0586	0	{'offer id': 'fafdcd668e3743c1bb461111dcafc2a4'}	fafdcd668e3743c1bb461111dcafc2a4
4	offer received	68617ca6246f4fbc85e91a2a49552598	0	{'offer id': '4d5c57ea9a6940dd891ad53e9dbe8da0'}	4d5c57ea9a6940dd891ad53e9dbe8da0

2.1.3 3. Exploratory Data Analysis

A. Profile

```
In [29]: df_profile.head()
```

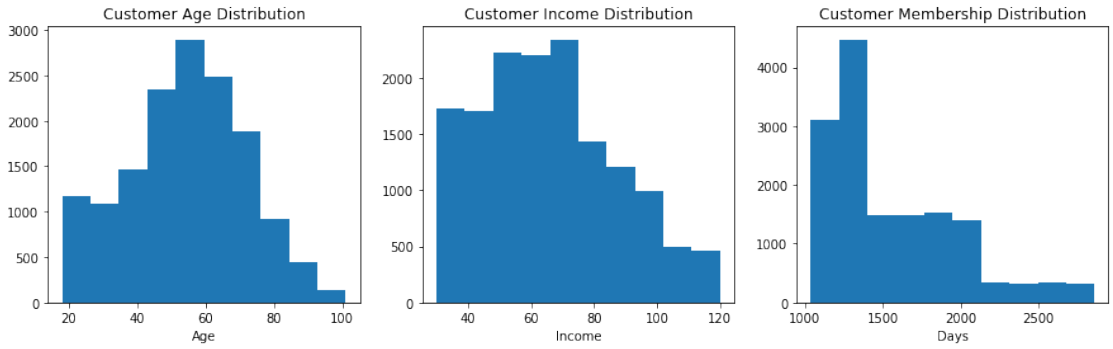
Out[29]:

	age	became_member_on	gender	customer_id	income	days_member
1	55.0	2017-07-15	F	0610b486422d4921ae7d2bf64640c50b	112000.0	1414
3	75.0	2017-05-09	F	78afa995795e4d85b5d9ceeca43f5fef	100000.0	1481
5	68.0	2018-04-26	M	e2127556f4f64592b11af22de27a7932	70000.0	1129
8	65.0	2018-02-09	M	389bc3fa690240e798340f5a15918d5c	53000.0	1205
12	58.0	2017-11-11	M	2eeac8d8feae4a8cad5a6af0499a211d	51000.0	1295

General Distribution of data in Profile

```
In [30]: # draw 3 subplots in same row
         fig, ax = plt.subplots(figsize=(15, 4), nrows=1, ncols=3)
```

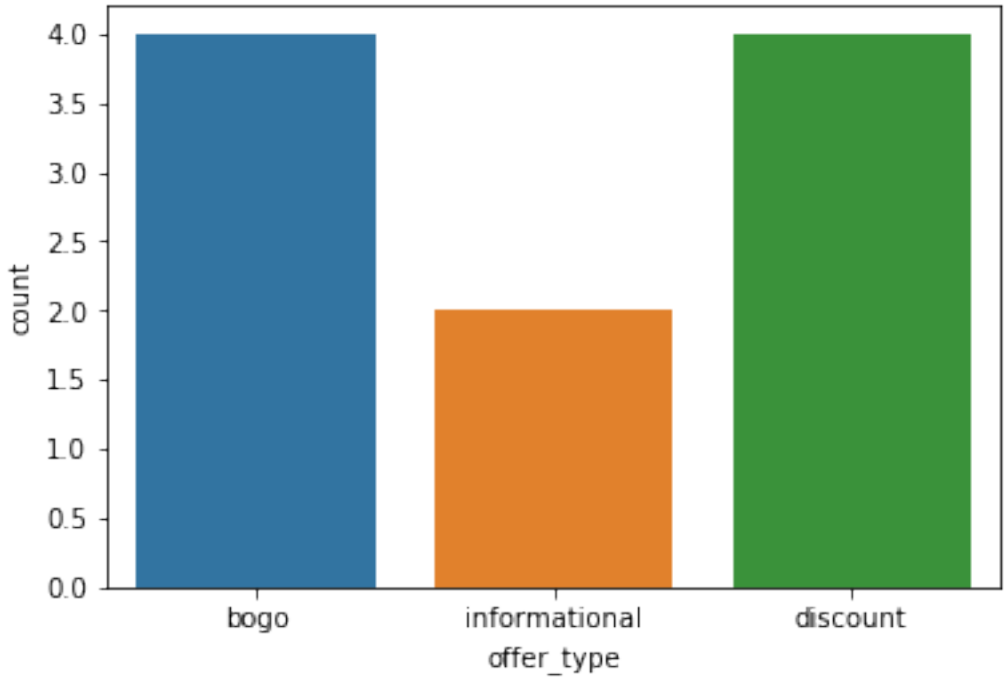
```
# plot client age distribution
plt.sca(ax[0])
plt.hist(df_profile['age'])
plt.xlabel('Age')

plt.title('Customer Age Distribution')

# plot client distribution
plt.sca(ax[1])
plt.hist(df_profile['income'] * 1E-3 )
plt.xlabel('Income')
plt.title('Customer Income Distribution');

# plot client membership distribution
plt.sca(ax[2])
plt.hist(df_profile['days_member'])
plt.xlabel('Days')
plt.title('Customer Membership Distribution');
```



The plot of customer age distribution highlights that the median age of a customer is 60. Furthermore, it can be revealed that customers age is ranging between 40 to 70 years old. Regarding the customer income distribution, it is highlighted that most of the costumers are with an average salary less than 70K. In addition, the customer membership distribution plot pointed out that the membership of customers range between 1000 and 2100 days.

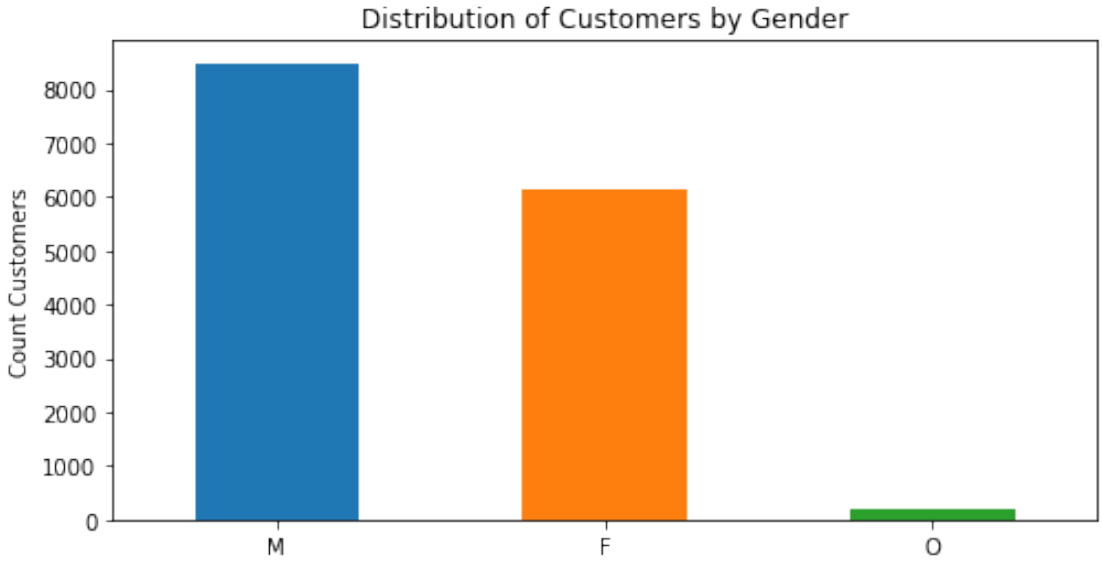
B. Portfolio

```
In [40]: #checking distribution of offers in the dataframe
         sns.countplot('offer_type', data= portfolio)

Out[40]: <matplotlib.axes._subplots.AxesSubplot at 0x7f5c93733748>
```



The plot highlights that the offers bogo and discount are mainly dominating.

```
In [39]: fig, ax = plt.subplots(figsize = (8,4))
         df_profile.gender.value_counts().plot(kind='bar', ax=ax)
         ax.set_xticklabels(ax.get_xticklabels(), rotation=0)
         ax.set_title('Distribution of Customers by Gender')
         ax.set_ylabel('Count Customers')
         plt.show ()
```



The plot highlights that in the dataset the distribution of Male customers is higher than female and other genders.

2.1.4 4. Modeling: Build Recommendation Engine

A. Build User Matrix

```
In [ ]: def create_user_item_matrix(df_offer, filename):
            '''
            Return the user item matrix that indicate the number of offer complete of a particul

            INPUT:
            df_offer - a cleaned transcript dataframe
            filename(string) - the file name that save the user item matrix

            OUTPUT:
            user_item_matrix - the user item matrix which
                - row is user
                - column is offer
                - value is the number of offer complete by the user (NaN means no offer given)

            '''
            # create an empty user item matrix
            user_item_matrix = df_offer.groupby(['customer_id', 'offer_id'])['event'].agg(lambda
            # we just focus on bogo and discount first
            user_item_matrix.drop(list(portfolio[portfolio['offer_type']=='informational']['id']

            for offer_id in user_item_matrix.columns:
                print("Now processing: ", offer_id)
                num = 0
                for person in user_item_matrix.index:
                    num += 1
                    if num % 1000 == 0:
                        print("finished ", num/16994*100, '%')
                    events = []
                    for event in df_offer[(df_offer['offer_id']==offer_id) & (df_offer['customer
                        events.append(event)
                    if len(events) >= 3:
                        user_item_matrix.loc[person, offer_id] = 0
                        for i in range(len(events)-2):
                            # check if the transaction sequence is offer received -> offer viewe
                            # if yes, we assume the user reacted positively with the offer we pr
                            if (events[i] == 'offer received') & (events[i+1] == 'offer viewed')
                                user_item_matrix.loc[person, offer_id] += 1
                    elif len(events) > 0:
                        user_item_matrix.loc[person, offer_id] = 0
```

```
            # store the large martix into file
            fh = open(filename, 'wb')
            pickle.dump(user_item_matrix,fh)
            fh.close()

            return user_item_matrix
```

```
In [74]: df = create_user_item_matrix(df_offer, 'user_item_matrix.p')
```

```
Now processing:  0b1e1539f2cc45b7b9fa7c272da2e1d7
finished  5.884429798752501 %
finished  11.768859597505003 %
finished  17.653289396257502 %
finished  23.537719195010006 %
finished  29.422148993762505 %
finished  35.306578792515005 %
finished  41.19100859126751 %
finished  47.07543839002001 %
finished  52.95986818877251 %
finished  58.84429798752501 %
finished  64.7287277862775 %
finished  70.61315758503001 %
finished  76.4975873837825 %
finished  82.38201718253502 %
finished  88.26644698128752 %
finished  94.15087678004002 %
Now processing:  2298d6c36e964ae4a3e7e9706d1fb8c2
finished  5.884429798752501 %
finished  11.768859597505003 %
finished  17.653289396257502 %
finished  23.537719195010006 %
finished  29.422148993762505 %
finished  35.306578792515005 %
finished  41.19100859126751 %
finished  47.07543839002001 %
finished  52.95986818877251 %
finished  58.84429798752501 %
finished  64.7287277862775 %
finished  70.61315758503001 %
finished  76.4975873837825 %
finished  82.38201718253502 %
finished  88.26644698128752 %
finished  94.15087678004002 %
Now processing:  2906b810c7d4411798c6938adc9daaa5
finished  5.884429798752501 %
finished  11.768859597505003 %
finished  17.653289396257502 %
```

```
finished  23.537719195010006 %
finished  29.422148993762505 %
finished  35.306578792515005 %
finished  41.19100859126751 %
finished  47.07543839002001 %
finished  52.95986818877251 %
finished  58.84429798752501 %
finished  64.7287277862775 %
finished  70.61315758503001 %
finished  76.4975873837825 %
finished  82.38201718253502 %
finished  88.26644698128752 %
finished  94.15087678004002 %
Now processing:  4d5c57ea9a6940dd891ad53e9dbe8da0
finished  5.884429798752501 %
finished  11.768859597505003 %
finished  17.653289396257502 %
finished  23.537719195010006 %
finished  29.422148993762505 %
finished  35.306578792515005 %
finished  41.19100859126751 %
finished  47.07543839002001 %
finished  52.95986818877251 %
finished  58.84429798752501 %
finished  64.7287277862775 %
finished  70.61315758503001 %
finished  76.4975873837825 %
finished  82.38201718253502 %
finished  88.26644698128752 %
finished  94.15087678004002 %
Now processing:  9b98b8c7a33c4b65b9aebfe6a799e6d9
finished  5.884429798752501 %
finished  11.768859597505003 %
finished  17.653289396257502 %
finished  23.537719195010006 %
finished  29.422148993762505 %
finished  35.306578792515005 %
finished  41.19100859126751 %
finished  47.07543839002001 %
finished  52.95986818877251 %
finished  58.84429798752501 %
finished  64.7287277862775 %
finished  70.61315758503001 %
finished  76.4975873837825 %
finished  82.38201718253502 %
finished  88.26644698128752 %
finished  94.15087678004002 %
Now processing:  ae264e3637204a6fb9bb56bc8210ddfd
```

```
finished  5.884429798752501 %
finished  11.768859597505003 %
finished  17.653289396257502 %
finished  23.537719195010006 %
finished  29.422148993762505 %
finished  35.306578792515005 %
finished  41.19100859126751 %
finished  47.07543839002001 %
finished  52.95986818877251 %
finished  58.84429798752501 %
finished  64.7287277862775 %
finished  70.61315758503001 %
finished  76.4975873837825 %
finished  82.38201718253502 %
finished  88.26644698128752 %
finished  94.15087678004002 %
Now processing:  f19421c1d4aa40978ebb69ca19b0e20d
finished  5.884429798752501 %
finished  11.768859597505003 %
finished  17.653289396257502 %
finished  23.537719195010006 %
finished  29.422148993762505 %
finished  35.306578792515005 %
finished  41.19100859126751 %
finished  47.07543839002001 %
finished  52.95986818877251 %
finished  58.84429798752501 %
finished  64.7287277862775 %
finished  70.61315758503001 %
finished  76.4975873837825 %
finished  82.38201718253502 %
finished  88.26644698128752 %
finished  94.15087678004002 %
Now processing:  fafdcd668e3743c1bb461111dcafc2a4
finished  5.884429798752501 %
finished  11.768859597505003 %
finished  17.653289396257502 %
finished  23.537719195010006 %
finished  29.422148993762505 %
finished  35.306578792515005 %
finished  41.19100859126751 %
finished  47.07543839002001 %
finished  52.95986818877251 %
finished  58.84429798752501 %
finished  64.7287277862775 %
finished  70.61315758503001 %
finished  76.4975873837825 %
finished  82.38201718253502 %
```

```
finished  88.26644698128752 %
finished  94.15087678004002 %
```

```
In [75]: df.head()
```

```
Out[75]: offer_id                          0b1e1539f2cc45b7b9fa7c272da2e1d7  \
         customer_id
         0009655768c64bdeb2e877511632db8f                               NaN
         00116118485d4dfda04fdbaba9a87b5c                               NaN
         0011e0d4e6b944f998e987f904e8c1e5                               1.0
         0020c2b971eb4e9188eac86d93036a77                               NaN
         0020ccbbb6d84e358d3414a3ff76cffd                               NaN

         offer_id                          2298d6c36e964ae4a3e7e9706d1fb8c2  \
         customer_id
         0009655768c64bdeb2e877511632db8f                               NaN
         00116118485d4dfda04fdbaba9a87b5c                               NaN
         0011e0d4e6b944f998e987f904e8c1e5                               1.0
         0020c2b971eb4e9188eac86d93036a77                               NaN
         0020ccbbb6d84e358d3414a3ff76cffd                               1.0

         offer_id                          2906b810c7d4411798c6938adc9daaa5  \
         customer_id
         0009655768c64bdeb2e877511632db8f                               0.0
         00116118485d4dfda04fdbaba9a87b5c                               NaN
         0011e0d4e6b944f998e987f904e8c1e5                               NaN
         0020c2b971eb4e9188eac86d93036a77                               NaN
         0020ccbbb6d84e358d3414a3ff76cffd                               NaN

         offer_id                          4d5c57ea9a6940dd891ad53e9dbe8da0  \
         customer_id
         0009655768c64bdeb2e877511632db8f                               NaN
         00116118485d4dfda04fdbaba9a87b5c                               NaN
         0011e0d4e6b944f998e987f904e8c1e5                               NaN
         0020c2b971eb4e9188eac86d93036a77                               1.0
         0020ccbbb6d84e358d3414a3ff76cffd                               NaN

         offer_id                          9b98b8c7a33c4b65b9aebfe6a799e6d9  \
         customer_id
         0009655768c64bdeb2e877511632db8f                               NaN
         00116118485d4dfda04fdbaba9a87b5c                               NaN
         0011e0d4e6b944f998e987f904e8c1e5                               1.0
         0020c2b971eb4e9188eac86d93036a77                               NaN
         0020ccbbb6d84e358d3414a3ff76cffd                               1.0

         offer_id                          ae264e3637204a6fb9bb56bc8210ddfd  \
         customer_id
```

```
0009655768c64bdeb2e877511632db8f                                NaN
00116118485d4dfda04fdbaba9a87b5c                                NaN
0011e0d4e6b944f998e987f904e8c1e5                                NaN
0020c2b971eb4e9188eac86d93036a77                                0.0
0020ccbbb6d84e358d3414a3ff76cffd                                NaN

offer_id                        f19421c1d4aa40978ebb69ca19b0e20d  \
customer_id
0009655768c64bdeb2e877511632db8f                                0.0
00116118485d4dfda04fdbaba9a87b5c                                0.0
0011e0d4e6b944f998e987f904e8c1e5                                NaN
0020c2b971eb4e9188eac86d93036a77                                NaN
0020ccbbb6d84e358d3414a3ff76cffd                                1.0

offer_id                        fafdcd668e3743c1bb461111dcafc2a4
customer_id
0009655768c64bdeb2e877511632db8f                                0.0
00116118485d4dfda04fdbaba9a87b5c                                NaN
0011e0d4e6b944f998e987f904e8c1e5                                NaN
0020c2b971eb4e9188eac86d93036a77                                1.0
0020ccbbb6d84e358d3414a3ff76cffd                                NaN
```

```
In [76]: size_train = int(df_offer.shape[0]*0.7)
         size_test = df_offer.shape[0] - size_train
         size_train, size_test
```

```
Out[76]: (117306, 50275)
```

```
In [77]: offer_train = df_offer[:size_train]
```

```
In [78]: offer_test = df_offer[size_train:]
```

```
In [79]: df_train = create_user_item_matrix(offer_train, 'df_train.p')
```

```
Now processing:  0b1e1539f2cc45b7b9fa7c272da2e1d7
finished  5.884429798752501 %
finished  11.768859597505003 %
finished  17.653289396257502 %
finished  23.537719195010006 %
finished  29.422148993762505 %
finished  35.306578792515005 %
finished  41.19100859126751 %
finished  47.07543839002001 %
finished  52.95986818877251 %
finished  58.84429798752501 %
finished  64.7287277862775 %
finished  70.61315758503001 %
finished  76.4975873837825 %
finished  82.38201718253502 %
```

```
finished  88.26644698128752 %
finished  94.15087678004002 %
Now processing:  2298d6c36e964ae4a3e7e9706d1fb8c2
finished  5.884429798752501 %
finished  11.768859597505003 %
finished  17.653289396257502 %
finished  23.537719195010006 %
finished  29.422148993762505 %
finished  35.306578792515005 %
finished  41.19100859126751 %
finished  47.07543839002001 %
finished  52.95986818877251 %
finished  58.84429798752501 %
finished  64.7287277862775 %
finished  70.61315758503001 %
finished  76.4975873837825 %
finished  82.38201718253502 %
finished  88.26644698128752 %
finished  94.15087678004002 %
Now processing:  2906b810c7d4411798c6938adc9daaa5
finished  5.884429798752501 %
finished  11.768859597505003 %
finished  17.653289396257502 %
finished  23.537719195010006 %
finished  29.422148993762505 %
finished  35.306578792515005 %
finished  41.19100859126751 %
finished  47.07543839002001 %
finished  52.95986818877251 %
finished  58.84429798752501 %
finished  64.7287277862775 %
finished  70.61315758503001 %
finished  76.4975873837825 %
finished  82.38201718253502 %
finished  88.26644698128752 %
finished  94.15087678004002 %
Now processing:  4d5c57ea9a6940dd891ad53e9dbe8da0
finished  5.884429798752501 %
finished  11.768859597505003 %
finished  17.653289396257502 %
finished  23.537719195010006 %
finished  29.422148993762505 %
finished  35.306578792515005 %
finished  41.19100859126751 %
finished  47.07543839002001 %
finished  52.95986818877251 %
finished  58.84429798752501 %
finished  64.7287277862775 %
```

```
finished  70.61315758503001 %
finished  76.4975873837825 %
finished  82.38201718253502 %
finished  88.26644698128752 %
finished  94.15087678004002 %
Now processing:  9b98b8c7a33c4b65b9aebfe6a799e6d9
finished  5.884429798752501 %
finished  11.768859597505003 %
finished  17.653289396257502 %
finished  23.537719195010006 %
finished  29.422148993762505 %
finished  35.306578792515005 %
finished  41.19100859126751 %
finished  47.07543839002001 %
finished  52.95986818877251 %
finished  58.84429798752501 %
finished  64.7287277862775 %
finished  70.61315758503001 %
finished  76.4975873837825 %
finished  82.38201718253502 %
finished  88.26644698128752 %
finished  94.15087678004002 %
Now processing:  ae264e3637204a6fb9bb56bc8210ddfd
finished  5.884429798752501 %
finished  11.768859597505003 %
finished  17.653289396257502 %
finished  23.537719195010006 %
finished  29.422148993762505 %
finished  35.306578792515005 %
finished  41.19100859126751 %
finished  47.07543839002001 %
finished  52.95986818877251 %
finished  58.84429798752501 %
finished  64.7287277862775 %
finished  70.61315758503001 %
finished  76.4975873837825 %
finished  82.38201718253502 %
finished  88.26644698128752 %
finished  94.15087678004002 %
Now processing:  f19421c1d4aa40978ebb69ca19b0e20d
finished  5.884429798752501 %
finished  11.768859597505003 %
finished  17.653289396257502 %
finished  23.537719195010006 %
finished  29.422148993762505 %
finished  35.306578792515005 %
finished  41.19100859126751 %
finished  47.07543839002001 %
```

```
finished  52.95986818877251 %
finished  58.84429798752501 %
finished  64.7287277862775 %
finished  70.61315758503001 %
finished  76.4975873837825 %
finished  82.38201718253502 %
finished  88.26644698128752 %
finished  94.15087678004002 %
Now processing:  fafdcd668e3743c1bb461111dcafc2a4
finished  5.884429798752501 %
finished  11.768859597505003 %
finished  17.653289396257502 %
finished  23.537719195010006 %
finished  29.422148993762505 %
finished  35.306578792515005 %
finished  41.19100859126751 %
finished  47.07543839002001 %
finished  52.95986818877251 %
finished  58.84429798752501 %
finished  64.7287277862775 %
finished  70.61315758503001 %
finished  76.4975873837825 %
finished  82.38201718253502 %
finished  88.26644698128752 %
finished  94.15087678004002 %
```

```
In [80]: df_train.head()
```

```
Out[80]: offer_id                          0b1e1539f2cc45b7b9fa7c272da2e1d7  \
         customer_id
         0009655768c64bdeb2e877511632db8f                               NaN
         00116118485d4dfda04fdbaba9a87b5c                               NaN
         0011e0d4e6b944f998e987f904e8c1e5                               0.0
         0020c2b971eb4e9188eac86d93036a77                               NaN
         0020ccbbb6d84e358d3414a3ff76cffd                               NaN

         offer_id                          2298d6c36e964ae4a3e7e9706d1fb8c2  \
         customer_id
         0009655768c64bdeb2e877511632db8f                               NaN
         00116118485d4dfda04fdbaba9a87b5c                               NaN
         0011e0d4e6b944f998e987f904e8c1e5                               1.0
         0020c2b971eb4e9188eac86d93036a77                               NaN
         0020ccbbb6d84e358d3414a3ff76cffd                               1.0

         offer_id                          2906b810c7d4411798c6938adc9daaa5  \
         customer_id
         0009655768c64bdeb2e877511632db8f                               NaN
```

```
00116118485d4dfda04fdbaba9a87b5c                                NaN
0011e0d4e6b944f998e987f904e8c1e5                                NaN
0020c2b971eb4e9188eac86d93036a77                                NaN
0020ccbbb6d84e358d3414a3ff76cffd                                NaN

offer_id                        4d5c57ea9a6940dd891ad53e9dbe8da0  \
customer_id
0009655768c64bdeb2e877511632db8f                                NaN
00116118485d4dfda04fdbaba9a87b5c                                NaN
0011e0d4e6b944f998e987f904e8c1e5                                NaN
0020c2b971eb4e9188eac86d93036a77                                0.0
0020ccbbb6d84e358d3414a3ff76cffd                                NaN

offer_id                        9b98b8c7a33c4b65b9aebfe6a799e6d9  \
customer_id
0009655768c64bdeb2e877511632db8f                                NaN
00116118485d4dfda04fdbaba9a87b5c                                NaN
0011e0d4e6b944f998e987f904e8c1e5                                0.0
0020c2b971eb4e9188eac86d93036a77                                NaN
0020ccbbb6d84e358d3414a3ff76cffd                                0.0

offer_id                        ae264e3637204a6fb9bb56bc8210ddfd  \
customer_id
0009655768c64bdeb2e877511632db8f                                NaN
00116118485d4dfda04fdbaba9a87b5c                                NaN
0011e0d4e6b944f998e987f904e8c1e5                                NaN
0020c2b971eb4e9188eac86d93036a77                                0.0
0020ccbbb6d84e358d3414a3ff76cffd                                NaN

offer_id                        f19421c1d4aa40978ebb69ca19b0e20d  \
customer_id
0009655768c64bdeb2e877511632db8f                                0.0
00116118485d4dfda04fdbaba9a87b5c                                0.0
0011e0d4e6b944f998e987f904e8c1e5                                NaN
0020c2b971eb4e9188eac86d93036a77                                NaN
0020ccbbb6d84e358d3414a3ff76cffd                                1.0

offer_id                        fafdcd668e3743c1bb461111dcafc2a4
customer_id
0009655768c64bdeb2e877511632db8f                                0.0
00116118485d4dfda04fdbaba9a87b5c                                NaN
0011e0d4e6b944f998e987f904e8c1e5                                NaN
0020c2b971eb4e9188eac86d93036a77                                1.0
0020ccbbb6d84e358d3414a3ff76cffd                                NaN
```

```
In [81]: df_test = create_user_item_matrix(offer_test, 'df_test.p')
```

```
Now processing:  0b1e1539f2cc45b7b9fa7c272da2e1d7
finished  5.884429798752501 %
```

```
finished  11.768859597505003 %
finished  17.653289396257502 %
finished  23.537719195010006 %
finished  29.422148993762505 %
finished  35.306578792515005 %
finished  41.19100859126751 %
finished  47.07543839002001 %
finished  52.95986818877251 %
finished  58.84429798752501 %
finished  64.7287277862775 %
finished  70.61315758503001 %
finished  76.4975873837825 %
finished  82.38201718253502 %
finished  88.26644698128752 %
Now processing:  2298d6c36e964ae4a3e7e9706d1fb8c2
finished  5.884429798752501 %
finished  11.768859597505003 %
finished  17.653289396257502 %
finished  23.537719195010006 %
finished  29.422148993762505 %
finished  35.306578792515005 %
finished  41.19100859126751 %
finished  47.07543839002001 %
finished  52.95986818877251 %
finished  58.84429798752501 %
finished  64.7287277862775 %
finished  70.61315758503001 %
finished  76.4975873837825 %
finished  82.38201718253502 %
finished  88.26644698128752 %
Now processing:  2906b810c7d4411798c6938adc9daaa5
finished  5.884429798752501 %
finished  11.768859597505003 %
finished  17.653289396257502 %
finished  23.537719195010006 %
finished  29.422148993762505 %
finished  35.306578792515005 %
finished  41.19100859126751 %
finished  47.07543839002001 %
finished  52.95986818877251 %
finished  58.84429798752501 %
finished  64.7287277862775 %
finished  70.61315758503001 %
finished  76.4975873837825 %
finished  82.38201718253502 %
finished  88.26644698128752 %
Now processing:  4d5c57ea9a6940dd891ad53e9dbe8da0
finished  5.884429798752501 %
```

```
finished  11.768859597505003 %
finished  17.653289396257502 %
finished  23.537719195010006 %
finished  29.422148993762505 %
finished  35.306578792515005 %
finished  41.19100859126751 %
finished  47.07543839002001 %
finished  52.95986818877251 %
finished  58.84429798752501 %
finished  64.7287277862775 %
finished  70.61315758503001 %
finished  76.4975873837825 %
finished  82.38201718253502 %
finished  88.26644698128752 %
Now processing:  9b98b8c7a33c4b65b9aebfe6a799e6d9
finished  5.884429798752501 %
finished  11.768859597505003 %
finished  17.653289396257502 %
finished  23.537719195010006 %
finished  29.422148993762505 %
finished  35.306578792515005 %
finished  41.19100859126751 %
finished  47.07543839002001 %
finished  52.95986818877251 %
finished  58.84429798752501 %
finished  64.7287277862775 %
finished  70.61315758503001 %
finished  76.4975873837825 %
finished  82.38201718253502 %
finished  88.26644698128752 %
Now processing:  ae264e3637204a6fb9bb56bc8210ddfd
finished  5.884429798752501 %
finished  11.768859597505003 %
finished  17.653289396257502 %
finished  23.537719195010006 %
finished  29.422148993762505 %
finished  35.306578792515005 %
finished  41.19100859126751 %
finished  47.07543839002001 %
finished  52.95986818877251 %
finished  58.84429798752501 %
finished  64.7287277862775 %
finished  70.61315758503001 %
finished  76.4975873837825 %
finished  82.38201718253502 %
finished  88.26644698128752 %
Now processing:  f19421c1d4aa40978ebb69ca19b0e20d
finished  5.884429798752501 %
```

```
finished  11.768859597505003 %
finished  17.653289396257502 %
finished  23.537719195010006 %
finished  29.422148993762505 %
finished  35.306578792515005 %
finished  41.19100859126751 %
finished  47.07543839002001 %
finished  52.95986818877251 %
finished  58.84429798752501 %
finished  64.7287277862775 %
finished  70.61315758503001 %
finished  76.4975873837825 %
finished  82.38201718253502 %
finished  88.26644698128752 %
Now processing:  fafdcd668e3743c1bb461111dcafc2a4
finished  5.884429798752501 %
finished  11.768859597505003 %
finished  17.653289396257502 %
finished  23.537719195010006 %
finished  29.422148993762505 %
finished  35.306578792515005 %
finished  41.19100859126751 %
finished  47.07543839002001 %
finished  52.95986818877251 %
finished  58.84429798752501 %
finished  64.7287277862775 %
finished  70.61315758503001 %
finished  76.4975873837825 %
finished  82.38201718253502 %
finished  88.26644698128752 %


In [82]: df_test.head()

Out[82]: offer_id                          0b1e1539f2cc45b7b9fa7c272da2e1d7  \
         customer_id
         0009655768c64bdeb2e877511632db8f                               NaN
         00116118485d4dfda04fdbaba9a87b5c                               NaN
         0011e0d4e6b944f998e987f904e8c1e5                               0.0
         0020c2b971eb4e9188eac86d93036a77                               NaN
         0020ccbbb6d84e358d3414a3ff76cffd                               NaN

         offer_id                          2298d6c36e964ae4a3e7e9706d1fb8c2  \
         customer_id
         0009655768c64bdeb2e877511632db8f                               NaN
         00116118485d4dfda04fdbaba9a87b5c                               NaN
         0011e0d4e6b944f998e987f904e8c1e5                               NaN
         0020c2b971eb4e9188eac86d93036a77                               NaN
```

```
0020ccbbb6d84e358d3414a3ff76cffd                                NaN

offer_id                        2906b810c7d4411798c6938adc9daaa5  \
customer_id
0009655768c64bdeb2e877511632db8f                                0.0
00116118485d4dfda04fdbaba9a87b5c                                NaN
0011e0d4e6b944f998e987f904e8c1e5                                NaN
0020c2b971eb4e9188eac86d93036a77                                NaN
0020ccbbb6d84e358d3414a3ff76cffd                                NaN

offer_id                        4d5c57ea9a6940dd891ad53e9dbe8da0  \
customer_id
0009655768c64bdeb2e877511632db8f                                NaN
00116118485d4dfda04fdbaba9a87b5c                                NaN
0011e0d4e6b944f998e987f904e8c1e5                                NaN
0020c2b971eb4e9188eac86d93036a77                                0.0
0020ccbbb6d84e358d3414a3ff76cffd                                NaN

offer_id                        9b98b8c7a33c4b65b9aebfe6a799e6d9  \
customer_id
0009655768c64bdeb2e877511632db8f                                NaN
00116118485d4dfda04fdbaba9a87b5c                                NaN
0011e0d4e6b944f998e987f904e8c1e5                                0.0
0020c2b971eb4e9188eac86d93036a77                                NaN
0020ccbbb6d84e358d3414a3ff76cffd                                0.0

offer_id                        ae264e3637204a6fb9bb56bc8210ddfd  \
customer_id
0009655768c64bdeb2e877511632db8f                                NaN
00116118485d4dfda04fdbaba9a87b5c                                NaN
0011e0d4e6b944f998e987f904e8c1e5                                NaN
0020c2b971eb4e9188eac86d93036a77                                NaN
0020ccbbb6d84e358d3414a3ff76cffd                                NaN

offer_id                        f19421c1d4aa40978ebb69ca19b0e20d  \
customer_id
0009655768c64bdeb2e877511632db8f                                NaN
00116118485d4dfda04fdbaba9a87b5c                                0.0
0011e0d4e6b944f998e987f904e8c1e5                                NaN
0020c2b971eb4e9188eac86d93036a77                                NaN
0020ccbbb6d84e358d3414a3ff76cffd                                NaN

offer_id                        fafdcd668e3743c1bb461111dcafc2a4
customer_id
0009655768c64bdeb2e877511632db8f                                0.0
00116118485d4dfda04fdbaba9a87b5c                                NaN
0011e0d4e6b944f998e987f904e8c1e5                                NaN
0020c2b971eb4e9188eac86d93036a77                                0.0
```

```
          0020ccbbb6d84e358d3414a3ff76cffd                                NaN
```

```
In [83]: df = pd.read_pickle('user_item_matrix.p')
```

```
In [85]: df_train = pd.read_pickle('df_train.p')
```

```
In [86]: df_test = pd.read_pickle('df_test.p')
```

B. Implement FunkSVD

```
In [93]: #https://medium.datadriveninvestor.com/how-funk-singular-value-decomposition-algorithm-
         #https://towardsdatascience.com/personalised-restaurant-recommendations-using-funksvd-3


         def FunkSVD(user_item_matrix, latent_features=12, learning_rate=0.0001, iters=100):
             '''
             This function performs matrix factorization using a basic form of FunkSVD with no r

             Input:
             user_item_matrix - (numpy array) a matrix with users as rows, offers as columns, an
             latent_features - (int) the number of latent features used
             learning_rate - (float) the learning rate
             iters - (int) the number of iterations

             Output:
             user_matrix - (numpy array) a user by latent feature matrix
             offer_matrix - (numpy array) a latent feature by offer matrix
             '''

             # Set up useful values to be used through the rest of the function
             n_users = user_item_matrix.shape[0]
             n_offers = user_item_matrix.shape[1]
             num_offers = np.count_nonzero(~np.isnan(user_item_matrix))

             # initialize the user and movie matrices with random values
             user_matrix = np.random.rand(n_users, latent_features)
             offer_matrix = np.random.rand(latent_features, n_offers)

             # initialize sse at 0 for first iteration
             sse_accum = 0

             # header for running results
             print("Optimizaiton Statistics")
             print("Iterations | Mean Squared Error ")

             # for each iteration
             for iteration in range(iters):

                 # update our sse
```

```
                old_sse = sse_accum
                sse_accum = 0

                # For each user-offer pair
                for i in range(n_users):
                    for j in range(n_offers):

                        # if the offer was completed
                        if user_item_matrix[i, j] > 0:

                            # compute the error as the actual minus the dot product of the user
                            diff = user_item_matrix[i, j] - np.dot(user_matrix[i, :], offer_mat

                            # Keep track of the sum of squared errors for the matrix
                            sse_accum += diff**2

                            # update the values in each matrix in the direction of the gradient
                            for k in range(latent_features):
                                user_matrix[i, k] += learning_rate * (2*diff*offer_matrix[k, j]
                                offer_matrix[k, j] += learning_rate * (2*diff*user_matrix[i, k]

                # print results for iteration
                print("%d \t\t %f" % (iteration+1, sse_accum / num_offers))

            return user_matrix, offer_matrix
```

```
In [94]: # Create user-by-item matrix - nothing to do here
         np_train_data = np.array(df_train)
```

```
In [95]: # Fit FunkSVD with the specified hyper parameters to the training data
         user_matrix_20, offer_matrix_20 = FunkSVD(np_train_data, latent_features=20, learning_r
```

```
Optimizaiton Statistics
Iterations | Mean Squared Error
1                  0.087035
2                  0.030220
3                  0.028875
4                  0.028548
5                  0.028360
6                  0.028200
7                  0.028048
8                  0.027898
9                  0.027750
10                 0.027603
11                 0.027457
12                 0.027311
13                 0.027167
14                 0.027023
```

15	0.026881
16	0.026739
17	0.026598
18	0.026458
19	0.026319
20	0.026181
21	0.026044
22	0.025907
23	0.025772
24	0.025637
25	0.025503
26	0.025370
27	0.025238
28	0.025106
29	0.024975
30	0.024846
31	0.024716
32	0.024588
33	0.024461
34	0.024334
35	0.024208
36	0.024082
37	0.023958
38	0.023834
39	0.023711
40	0.023589
41	0.023467
42	0.023346
43	0.023226
44	0.023106
45	0.022988
46	0.022870
47	0.022752
48	0.022635
49	0.022519
50	0.022404
51	0.022289
52	0.022175
53	0.022062
54	0.021949
55	0.021837
56	0.021725
57	0.021614
58	0.021504
59	0.021394
60	0.021285
61	0.021176
62	0.021069

63	0.020961
64	0.020855
65	0.020748
66	0.020643
67	0.020538
68	0.020434
69	0.020330
70	0.020226
71	0.020124
72	0.020021
73	0.019920
74	0.019819
75	0.019718
76	0.019618
77	0.019519
78	0.019420
79	0.019321
80	0.019223
81	0.019126
82	0.019029
83	0.018932
84	0.018836
85	0.018741
86	0.018646
87	0.018551
88	0.018457
89	0.018364
90	0.018271
91	0.018178
92	0.018086
93	0.017994
94	0.017903
95	0.017812
96	0.017722
97	0.017632
98	0.017542
99	0.017453
100	0.017364
101	0.017276
102	0.017189
103	0.017101
104	0.017014
105	0.016928
106	0.016842
107	0.016756
108	0.016671
109	0.016586
110	0.016501

111	0.016417
112	0.016333
113	0.016250
114	0.016167
115	0.016085
116	0.016002
117	0.015921
118	0.015839
119	0.015758
120	0.015678
121	0.015597
122	0.015517
123	0.015438
124	0.015358
125	0.015280
126	0.015201
127	0.015123
128	0.015045
129	0.014968
130	0.014890
131	0.014814
132	0.014737
133	0.014661
134	0.014585
135	0.014510
136	0.014435
137	0.014360
138	0.014285
139	0.014211
140	0.014137
141	0.014064
142	0.013991
143	0.013918
144	0.013845
145	0.013773
146	0.013701
147	0.013629
148	0.013558
149	0.013487
150	0.013416
151	0.013346
152	0.013276
153	0.013206
154	0.013136
155	0.013067
156	0.012998
157	0.012929
158	0.012861

159	0.012793
160	0.012725
161	0.012657
162	0.012590
163	0.012523
164	0.012456
165	0.012390
166	0.012324
167	0.012258
168	0.012192
169	0.012127
170	0.012062
171	0.011997
172	0.011932
173	0.011868
174	0.011804
175	0.011740
176	0.011677
177	0.011613
178	0.011550
179	0.011488
180	0.011425
181	0.011363
182	0.011301
183	0.011239
184	0.011178
185	0.011117
186	0.011056
187	0.010995
188	0.010934
189	0.010874
190	0.010814
191	0.010755
192	0.010695
193	0.010636
194	0.010577
195	0.010518
196	0.010460
197	0.010401
198	0.010343
199	0.010286
200	0.010228
201	0.010171
202	0.010114
203	0.010057
204	0.010000
205	0.009944
206	0.009888

```
207                 0.009832
208                 0.009776
209                 0.009721
210                 0.009665
211                 0.009611
212                 0.009556
213                 0.009501
214                 0.009447
215                 0.009393
216                 0.009339
217                 0.009285
218                 0.009232
219                 0.009179
220                 0.009126
221                 0.009073
222                 0.009021
223                 0.008969
224                 0.008916
225                 0.008865
226                 0.008813
227                 0.008762
228                 0.008711
229                 0.008660
230                 0.008609
231                 0.008558
232                 0.008508
233                 0.008458
234                 0.008408
235                 0.008359
236                 0.008309
237                 0.008260
238                 0.008211
239                 0.008162
240                 0.008114
241                 0.008065
242                 0.008017
243                 0.007969
244                 0.007922
245                 0.007874
246                 0.007827
247                 0.007780
248                 0.007733
249                 0.007686
250                 0.007640
```

```
In [96]: # Fit FunkSVD with the specified hyper parameters to the training data
         user_matrix_15, offer_matrix_15 = FunkSVD(np_train_data, latent_features=15, learning_r
```

```
Optimizaiton Statistics
Iterations | Mean Squared Error
1                   0.068025
2                   0.031165
3                   0.030202
4                   0.029888
5                   0.029660
6                   0.029449
7                   0.029243
8                   0.029040
9                   0.028838
10                   0.028639
11                   0.028442
12                   0.028246
13                   0.028053
14                   0.027860
15                   0.027670
16                   0.027481
17                   0.027294
18                   0.027109
19                   0.026925
20                   0.026743
21                   0.026563
22                   0.026384
23                   0.026207
24                   0.026031
25                   0.025857
26                   0.025684
27                   0.025513
28                   0.025343
29                   0.025175
30                   0.025008
31                   0.024843
32                   0.024679
33                   0.024516
34                   0.024355
35                   0.024196
36                   0.024038
37                   0.023881
38                   0.023725
39                   0.023571
40                   0.023418
41                   0.023266
42                   0.023116
43                   0.022967
44                   0.022819
45                   0.022673
46                   0.022528
```

47	0.022384
48	0.022241
49	0.022099
50	0.021959
51	0.021820
52	0.021682
53	0.021545
54	0.021409
55	0.021274
56	0.021141
57	0.021009
58	0.020877
59	0.020747
60	0.020618
61	0.020490
62	0.020363
63	0.020237
64	0.020112
65	0.019988
66	0.019865
67	0.019743
68	0.019622
69	0.019502
70	0.019383
71	0.019265
72	0.019148
73	0.019032
74	0.018917
75	0.018803
76	0.018689
77	0.018577
78	0.018465
79	0.018355
80	0.018245
81	0.018136
82	0.018028
83	0.017921
84	0.017814
85	0.017709
86	0.017604
87	0.017500
88	0.017397
89	0.017295
90	0.017193
91	0.017093
92	0.016993
93	0.016893
94	0.016795

95	0.016697
96	0.016600
97	0.016504
98	0.016409
99	0.016314
100	0.016220
101	0.016127
102	0.016034
103	0.015942
104	0.015851
105	0.015761
106	0.015671
107	0.015581
108	0.015493
109	0.015405
110	0.015318
111	0.015231
112	0.015145
113	0.015060
114	0.014975
115	0.014891
116	0.014808
117	0.014725
118	0.014642
119	0.014561
120	0.014480
121	0.014399
122	0.014319
123	0.014240
124	0.014161
125	0.014083
126	0.014005
127	0.013928
128	0.013851
129	0.013775
130	0.013699
131	0.013624
132	0.013550
133	0.013476
134	0.013402
135	0.013329
136	0.013257
137	0.013185
138	0.013113
139	0.013042
140	0.012972
141	0.012902
142	0.012832

143	0.012763
144	0.012694
145	0.012626
146	0.012558
147	0.012491
148	0.012424
149	0.012358
150	0.012292
151	0.012226
152	0.012161
153	0.012097
154	0.012032
155	0.011969
156	0.011905
157	0.011842
158	0.011779
159	0.011717
160	0.011655
161	0.011594
162	0.011533
163	0.011472
164	0.011412
165	0.011352
166	0.011293
167	0.011234
168	0.011175
169	0.011116
170	0.011058
171	0.011001
172	0.010943
173	0.010886
174	0.010830
175	0.010773
176	0.010717
177	0.010662
178	0.010607
179	0.010552
180	0.010497
181	0.010443
182	0.010389
183	0.010335
184	0.010282
185	0.010229
186	0.010176
187	0.010124
188	0.010072
189	0.010020
190	0.009968

191	0.009917
192	0.009866
193	0.009816
194	0.009766
195	0.009716
196	0.009666
197	0.009616
198	0.009567
199	0.009518
200	0.009470
201	0.009421
202	0.009373
203	0.009326
204	0.009278
205	0.009231
206	0.009184
207	0.009137
208	0.009091
209	0.009044
210	0.008998
211	0.008953
212	0.008907
213	0.008862
214	0.008817
215	0.008772
216	0.008728
217	0.008683
218	0.008639
219	0.008596
220	0.008552
221	0.008509
222	0.008466
223	0.008423
224	0.008380
225	0.008338
226	0.008295
227	0.008253
228	0.008212
229	0.008170
230	0.008129
231	0.008088
232	0.008047
233	0.008006
234	0.007966
235	0.007925
236	0.007885
237	0.007845
238	0.007806

```
239                  0.007766
240                  0.007727
241                  0.007688
242                  0.007649
243                  0.007610
244                  0.007572
245                  0.007533
246                  0.007495
247                  0.007457
248                  0.007420
249                  0.007382
250                  0.007345
```

In [97]:
```
# Fit FunkSVD with the specified hyper parameters to the training data
user_matrix_10, offer_matrix_10 = FunkSVD(np_train_data, latent_features=10, learning_r
```

```
Optimizaiton Statistics
Iterations | Mean Squared Error
1                   0.049506
2                   0.034150
3                   0.033404
4                   0.033004
5                   0.032647
6                   0.032301
7                   0.031962
8                   0.031627
9                   0.031297
10                   0.030972
11                   0.030652
12                   0.030335
13                   0.030023
14                   0.029716
15                   0.029412
16                   0.029113
17                   0.028818
18                   0.028527
19                   0.028240
20                   0.027957
21                   0.027678
22                   0.027403
23                   0.027131
24                   0.026864
25                   0.026600
26                   0.026339
27                   0.026082
28                   0.025828
29                   0.025578
```

30	0.025332
31	0.025088
32	0.024848
33	0.024612
34	0.024378
35	0.024148
36	0.023920
37	0.023696
38	0.023475
39	0.023257
40	0.023041
41	0.022829
42	0.022619
43	0.022412
44	0.022208
45	0.022007
46	0.021808
47	0.021612
48	0.021419
49	0.021228
50	0.021039
51	0.020853
52	0.020670
53	0.020489
54	0.020310
55	0.020134
56	0.019960
57	0.019788
58	0.019618
59	0.019451
60	0.019285
61	0.019122
62	0.018961
63	0.018803
64	0.018646
65	0.018491
66	0.018338
67	0.018187
68	0.018038
69	0.017891
70	0.017746
71	0.017602
72	0.017461
73	0.017321
74	0.017183
75	0.017047
76	0.016912
77	0.016779

78	0.016648
79	0.016518
80	0.016390
81	0.016264
82	0.016139
83	0.016016
84	0.015894
85	0.015774
86	0.015655
87	0.015537
88	0.015422
89	0.015307
90	0.015194
91	0.015082
92	0.014972
93	0.014863
94	0.014755
95	0.014648
96	0.014543
97	0.014439
98	0.014337
99	0.014235
100	0.014135
101	0.014036
102	0.013938
103	0.013841
104	0.013746
105	0.013651
106	0.013558
107	0.013466
108	0.013374
109	0.013284
110	0.013195
111	0.013107
112	0.013020
113	0.012934
114	0.012849
115	0.012765
116	0.012682
117	0.012599
118	0.012518
119	0.012438
120	0.012358
121	0.012280
122	0.012202
123	0.012125
124	0.012049
125	0.011974

126	0.011900
127	0.011827
128	0.011754
129	0.011682
130	0.011611
131	0.011541
132	0.011471
133	0.011403
134	0.011335
135	0.011267
136	0.011201
137	0.011135
138	0.011070
139	0.011005
140	0.010941
141	0.010878
142	0.010816
143	0.010754
144	0.010693
145	0.010633
146	0.010573
147	0.010514
148	0.010455
149	0.010397
150	0.010340
151	0.010283
152	0.010226
153	0.010171
154	0.010116
155	0.010061
156	0.010007
157	0.009954
158	0.009901
159	0.009848
160	0.009796
161	0.009745
162	0.009694
163	0.009644
164	0.009594
165	0.009545
166	0.009496
167	0.009447
168	0.009399
169	0.009352
170	0.009305
171	0.009258
172	0.009212
173	0.009166

174	0.009121
175	0.009076
176	0.009031
177	0.008987
178	0.008944
179	0.008900
180	0.008858
181	0.008815
182	0.008773
183	0.008731
184	0.008690
185	0.008649
186	0.008608
187	0.008568
188	0.008528
189	0.008489
190	0.008450
191	0.008411
192	0.008372
193	0.008334
194	0.008296
195	0.008259
196	0.008221
197	0.008184
198	0.008148
199	0.008112
200	0.008076
201	0.008040
202	0.008004
203	0.007969
204	0.007935
205	0.007900
206	0.007866
207	0.007832
208	0.007798
209	0.007765
210	0.007731
211	0.007698
212	0.007666
213	0.007633
214	0.007601
215	0.007569
216	0.007537
217	0.007506
218	0.007475
219	0.007444
220	0.007413
221	0.007382

```
222                 0.007352
223                 0.007322
224                 0.007292
225                 0.007262
226                 0.007233
227                 0.007204
228                 0.007175
229                 0.007146
230                 0.007117
231                 0.007089
232                 0.007060
233                 0.007032
234                 0.007005
235                 0.006977
236                 0.006949
237                 0.006922
238                 0.006895
239                 0.006868
240                 0.006841
241                 0.006814
242                 0.006788
243                 0.006762
244                 0.006736
245                 0.006710
246                 0.006684
247                 0.006658
248                 0.006633
249                 0.006607
250                 0.006582
```

```
In [98]: # Fit FunkSVD with the specified hyper parameters to the training data
         user_matrix_5, offer_matrix_5 = FunkSVD(np_train_data, latent_features=5, learning_rate
```

```
Optimizaiton Statistics
Iterations | Mean Squared Error
1                   0.049851
2                   0.045655
3                   0.044428
4                   0.043423
5                   0.042477
6                   0.041563
7                   0.040677
8                   0.039814
9                   0.038975
10                   0.038159
11                   0.037364
12                   0.036591
```

13	0.035838
14	0.035106
15	0.034393
16	0.033699
17	0.033024
18	0.032367
19	0.031727
20	0.031105
21	0.030498
22	0.029908
23	0.029334
24	0.028775
25	0.028230
26	0.027700
27	0.027184
28	0.026681
29	0.026192
30	0.025715
31	0.025251
32	0.024798
33	0.024358
34	0.023929
35	0.023510
36	0.023103
37	0.022706
38	0.022319
39	0.021942
40	0.021575
41	0.021217
42	0.020868
43	0.020528
44	0.020196
45	0.019873
46	0.019558
47	0.019251
48	0.018951
49	0.018659
50	0.018374
51	0.018095
52	0.017824
53	0.017560
54	0.017302
55	0.017050
56	0.016804
57	0.016564
58	0.016330
59	0.016102
60	0.015879

61	0.015661
62	0.015449
63	0.015241
64	0.015039
65	0.014841
66	0.014648
67	0.014460
68	0.014276
69	0.014096
70	0.013920
71	0.013749
72	0.013581
73	0.013417
74	0.013257
75	0.013101
76	0.012948
77	0.012799
78	0.012653
79	0.012510
80	0.012371
81	0.012235
82	0.012101
83	0.011971
84	0.011844
85	0.011719
86	0.011597
87	0.011478
88	0.011362
89	0.011248
90	0.011137
91	0.011028
92	0.010921
93	0.010817
94	0.010715
95	0.010615
96	0.010517
97	0.010421
98	0.010328
99	0.010236
100	0.010147
101	0.010059
102	0.009973
103	0.009889
104	0.009807
105	0.009726
106	0.009647
107	0.009570
108	0.009495

109	0.009421
110	0.009348
111	0.009277
112	0.009207
113	0.009139
114	0.009073
115	0.009007
116	0.008943
117	0.008880
118	0.008819
119	0.008759
120	0.008700
121	0.008642
122	0.008585
123	0.008530
124	0.008475
125	0.008422
126	0.008370
127	0.008318
128	0.008268
129	0.008219
130	0.008171
131	0.008123
132	0.008077
133	0.008031
134	0.007987
135	0.007943
136	0.007900
137	0.007858
138	0.007817
139	0.007776
140	0.007737
141	0.007698
142	0.007659
143	0.007622
144	0.007585
145	0.007549
146	0.007514
147	0.007479
148	0.007445
149	0.007411
150	0.007378
151	0.007346
152	0.007314
153	0.007283
154	0.007253
155	0.007223
156	0.007194

157	0.007165
158	0.007136
159	0.007109
160	0.007081
161	0.007054
162	0.007028
163	0.007002
164	0.006977
165	0.006952
166	0.006927
167	0.006903
168	0.006879
169	0.006856
170	0.006833
171	0.006811
172	0.006789
173	0.006767
174	0.006745
175	0.006724
176	0.006704
177	0.006683
178	0.006664
179	0.006644
180	0.006625
181	0.006606
182	0.006587
183	0.006569
184	0.006551
185	0.006533
186	0.006515
187	0.006498
188	0.006481
189	0.006464
190	0.006448
191	0.006432
192	0.006416
193	0.006400
194	0.006385
195	0.006370
196	0.006355
197	0.006340
198	0.006326
199	0.006311
200	0.006297
201	0.006283
202	0.006270
203	0.006256
204	0.006243

205	0.006230
206	0.006217
207	0.006204
208	0.006192
209	0.006179
210	0.006167
211	0.006155
212	0.006143
213	0.006131
214	0.006120
215	0.006108
216	0.006097
217	0.006086
218	0.006075
219	0.006064
220	0.006054
221	0.006043
222	0.006033
223	0.006022
224	0.006012
225	0.006002
226	0.005992
227	0.005982
228	0.005972
229	0.005963
230	0.005953
231	0.005944
232	0.005934
233	0.005925
234	0.005916
235	0.005907
236	0.005898
237	0.005889
238	0.005880
239	0.005872
240	0.005863
241	0.005855
242	0.005846
243	0.005838
244	0.005830
245	0.005821
246	0.005813
247	0.005805
248	0.005797
249	0.005789
250	0.005781

According to the MSE values, the model with 5 latent featues performs better than 20, 15, and 10 features. This can be ascribed to overfitting. Thus, as a next step, it is required to perform prediction of the test data to validate the results.

C. Check FUNKSVD models with test data

```
In [119]: def predict_reaction(user_matrix, offer_matrix, user_id, offer_id):
              '''
              INPUT:
              user_matrix - user by latent factor matrix
              offer_matrix - latent factor by offer matrix
              user_id - the user_id from the df reviews
              offer_id - the offer_id according the df offer

              OUTPUT:
              prediction - the predicted reaction for user_id-offer_id according to FunkSVD
              '''
              try:
                  # Use the training data to create a series of users and movies that matches th
                  user_ids_series = np.array(df_train.index)
                  offer_ids_series = np.array(df_train.columns)

                  # User row and Movie Column
                  user_row = np.where(user_ids_series == user_id)[0][0]
                  offer_col = np.where(offer_ids_series == offer_id)[0][0]

                  # Take dot product of that row and column in U and V to make prediction
                  prediction = np.dot(user_matrix[user_row, :], offer_matrix[:, offer_col])

                  return prediction

              except:
                  return None

In [120]: def validation(df_test, user_matrix, offer_matrix):
              '''Measure the squared errors for the prediction'''
              num_complete = np.count_nonzero(~np.isnan(df_test))

              sse_accum = 0

              for user_id in df_test.index:
                  for offer_id in df_test.columns:
                      if ~np.isnan(df_test.loc[user_id, offer_id]):
                          predict_value = predict_reaction(user_matrix, offer_matrix, user_id, o
                          if predict_value != None:
                              # compute the error as the actual minus the dot product of the use
                              diff = df_test.loc[user_id, offer_id] - predict_reaction(user_matr
```

```
                                # Keep track of the sum of squared errors for the matrix
                                sse_accum += diff**2

            print(sse_accum / num_complete)
```

```
In [122]: # Evaluation for latent features of 20
          validation(df_test, user_matrix_20, offer_matrix_20)
```

```
0.907767556582
```

```
In [124]: # Evaluation for latent features of 15
          validation(df_test, user_matrix_15, offer_matrix_15)
```

```
0.907527939812
```

```
In [125]: # Evaluation for latent features of 10
          validation(df_test, user_matrix_10, offer_matrix_10)
```

```
0.917476810067
```

```
In [126]: # Evaluation for latent features of 5
          validation(df_test, user_matrix_5, offer_matrix_5)
```

```
0.906086489741
```

The scores obtained after performing the model validation highlights that using latent featues of 5 may lead to better performance.

D. Recommendation The main target is to build a recommendation engine supporting new customers. The engine will recommend offers based on the best sales of offers dedicated to existing customers.

```
In [148]: #https://towardsdatascience.com/starbucks-offer-personalization-sending-the-right-offe
          # extract best sales offers
          def best_sales(user_item_matrix):

              gain_offer = []
              for offer_id in user_item_matrix.columns:
                  gain_offer.append([offer_id, transcript[(transcript['customer_id'].isin(list(u

              offer_value = pd.DataFrame(gain_offer, columns=['offer_id', 'gain'])
              offer_value['gain'] = pd.to_numeric(offer_value['gain'])
              offer_value.sort_values(by='gain', ascending=False, inplace=True)
              print(offer_value)
```

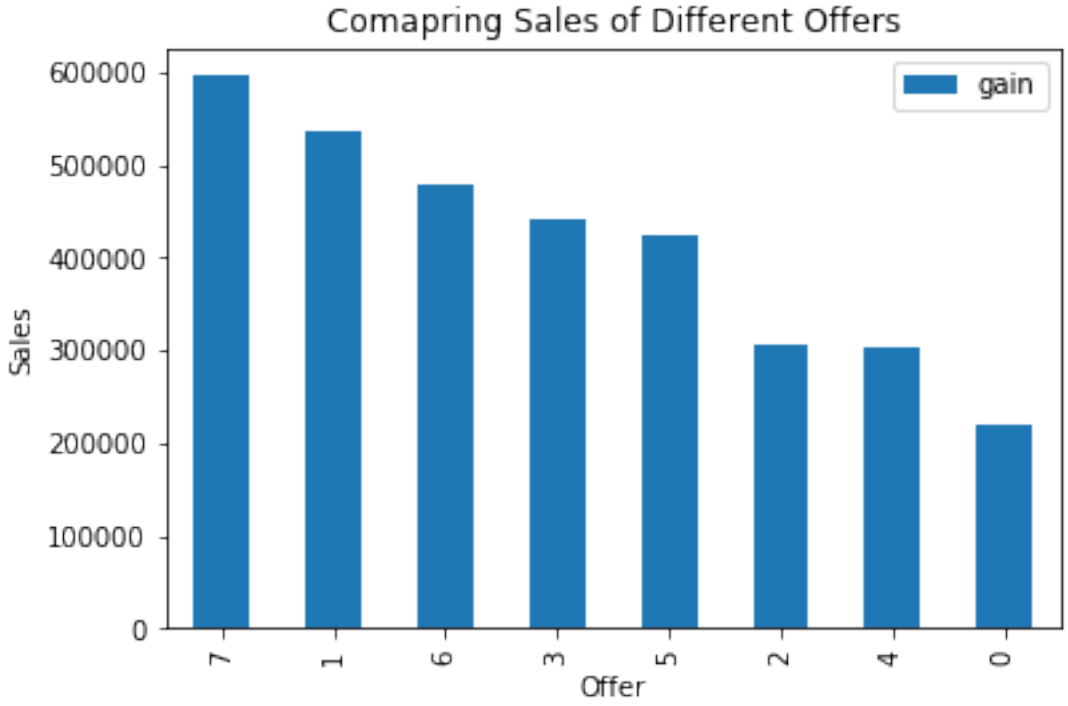
```
    best_sale = offer_value.plot(kind='bar', title='Comapring Sales of Different Offer
    best_sale.set_xlabel('Offer')
    best_sale.set_ylabel('Sales')

    return offer_value
```

```
In [149]: best_sales(df)
```

```
                          offer_id       gain
7  fafdcd668e3743c1bb461111dcafc2a4  596220.84
1  2298d6c36e964ae4a3e7e9706d1fb8c2  538027.98
6  f19421c1d4aa40978ebb69ca19b0e20d  480555.03
3  4d5c57ea9a6940dd891ad53e9dbe8da0  441918.93
5  ae264e3637204a6fb9bb56bc8210ddfd  423683.27
2  2906b810c7d4411798c6938adc9daaa5  305568.42
4  9b98b8c7a33c4b65b9aebfe6a799e6d9  304361.77
0  0b1e1539f2cc45b7b9fa7c272da2e1d7  218905.94
```

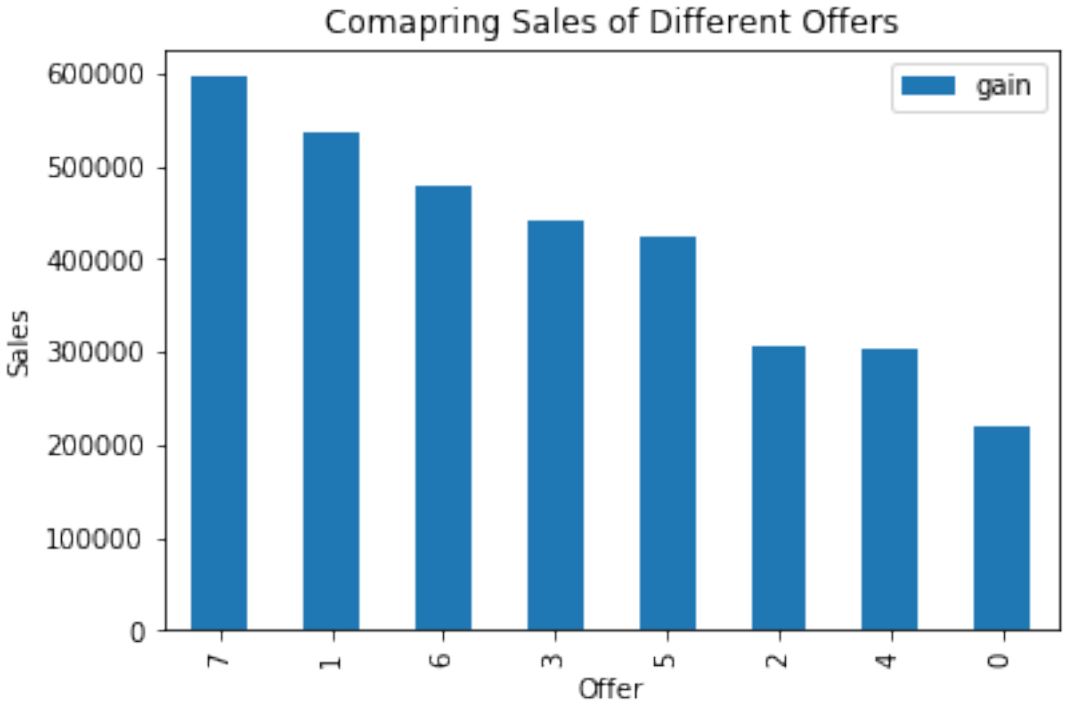
Out[149]:

	offer_id	gain
7	fafdcd668e3743c1bb461111dcafc2a4	596220.84
1	2298d6c36e964ae4a3e7e9706d1fb8c2	538027.98
6	f19421c1d4aa40978ebb69ca19b0e20d	480555.03
3	4d5c57ea9a6940dd891ad53e9dbe8da0	441918.93
5	ae264e3637204a6fb9bb56bc8210ddfd	423683.27
2	2906b810c7d4411798c6938adc9daaa5	305568.42
4	9b98b8c7a33c4b65b9aebfe6a799e6d9	304361.77
0	0b1e1539f2cc45b7b9fa7c272da2e1d7	218905.94



```
In [153]: def offer_recommendation(user_id, user_matrix, offer_matrix):
              recommendation = {}
              for offer_id in df_train.columns:
                  predict_value = predict_reaction(user_matrix, offer_matrix, user_id, offer_id)
                  if predict_value != None:
                      recommendation[offer_id] = predict_value
                  else:
                      break
              if predict_value == None:
                  print("New Customer, generating top best sales offer for existing customers.")
                  best_offer = best_sales(df)
                  for offer_id in best_offer['offer_id']:
                      print("offer id: ", offer_id)
              else:
                  print("Recommended offer for customer number: ", user_id)
                  for offer_id, predict_value in sorted(recommendation.items(), key=lambda kv:(k
                      print("offer id: ", offer_id, " predicted value: ", round(predict_value,2)

In [154]: # recommended offers for an existing customer
          offer_recommendation('78afa995795e4d85b5d9ceeca43f5fef', user_matrix_5, offer_matrix_5

Recommended offer for customer number:  78afa995795e4d85b5d9ceeca43f5fef
offer id:  ae264e3637204a6fb9bb56bc8210ddfd  predicted value:  1.16
offer id:  2906b810c7d4411798c6938adc9daaa5  predicted value:  1.15
offer id:  f19421c1d4aa40978ebb69ca19b0e20d  predicted value:  1.14
offer id:  2298d6c36e964ae4a3e7e9706d1fb8c2  predicted value:  1.07
offer id:  4d5c57ea9a6940dd891ad53e9dbe8da0  predicted value:  1.03
offer id:  fafdcd668e3743c1bb461111dcafc2a4  predicted value:  1.02
offer id:  0b1e1539f2cc45b7b9fa7c272da2e1d7  predicted value:  0.96
offer id:  9b98b8c7a33c4b65b9aebfe6a799e6d9  predicted value:  0.95


In [155]: # recommended offers to new customer
          offer_recommendation('new_user', user_matrix_5, offer_matrix_5)

New Customer, generating top best sales offer for existing customers.
                          offer_id       gain
7  fafdcd668e3743c1bb461111dcafc2a4  596220.84
1  2298d6c36e964ae4a3e7e9706d1fb8c2  538027.98
6  f19421c1d4aa40978ebb69ca19b0e20d  480555.03
3  4d5c57ea9a6940dd891ad53e9dbe8da0  441918.93
5  ae264e3637204a6fb9bb56bc8210ddfd  423683.27
2  2906b810c7d4411798c6938adc9daaa5  305568.42
4  9b98b8c7a33c4b65b9aebfe6a799e6d9  304361.77
0  0b1e1539f2cc45b7b9fa7c272da2e1d7  218905.94
offer id:  fafdcd668e3743c1bb461111dcafc2a4
```

```
offer id:  2298d6c36e964ae4a3e7e9706d1fb8c2
offer id:  f19421c1d4aa40978ebb69ca19b0e20d
offer id:  4d5c57ea9a6940dd891ad53e9dbe8da0
offer id:  ae264e3637204a6fb9bb56bc8210ddfd
offer id:  2906b810c7d4411798c6938adc9daaa5
offer id:  9b98b8c7a33c4b65b9aebfe6a799e6d9
offer id:  0b1e1539f2cc45b7b9fa7c272da2e1d7
```



2.1.5 5. Improve the Recommendation Engine

Further steps can be applied to enhance the implemented recommendation engine including for example: - Taking into consideration the customer profile such as age, gender, incomes - Applying further algorithms

2.1.6 6. References and Acknowledgement

- Ref1. The dataset is provided by Starbucks
- Ref2. Thank you Udacity for guiding with knowledge building.
- Ref3. Implementing Recommendation Engine for Sturbucks Andrea Xue
- Ref4. Implementing FunkSVD Medhy Vinceslas and Paul Stubley
- Ref5. The Medium Blog of Joshua Yeung guided me a lot with understanding steps to implement the recommendation engine using Starbucks data. Thank you for sharing your valuable knowledge.