

Starbucks_Capstone_Project

December 22, 2019

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

Note: If you are using the workspace, you will need to go to the terminal and run the command `conda update pandas` before reading in the files. This is because the version of pandas in the workspace cannot read in the `transcript.json` file correctly, but the newest version of pandas can. You can access the terminal from the orange icon in the top left of this notebook.

2.1 Problem Statement

The problem is simple: we want to make better purchasing offers to Starbucks' customers. For this, we can use customer's past behaviour to find patterns and try to be more assertive. As given

by the Udacity's Starbucks Project Overview, the basic task is to use the data to identify which groups of people are most responsive to each type of offer, and how best to present each type of offer. In other words, this is a classification problem where the model takes user behaviour data as input and produces a group as output (either previously defined or not).

This has been one of the [most used](#) applications of machine learning in the industry, since it provides you with means to save money spent on marketing campaigns by directing content to users who are more likely to convert based on a multitude of characteristics.

To evaluate the trained models, we'll compare the models based on it's F1-Score. This is a widely used metric to evaluate classification problems. [Aditya Mishra defines](#) it as follows: > F1 Score is the Harmonic Mean between precision and recall. The range for F1 Score is [0, 1]. It tells you how precise your classifier is (how many instances it classifies correctly), as well as how robust it is (it does not miss a significant number of instances).

```
[1]: import pandas as pd
import numpy as np
import math
import json
import seaborn as sns
import matplotlib.pyplot as plt

from sklearn import metrics
from sklearn.preprocessing import MultiLabelBinarizer, LabelEncoder
from sklearn.model_selection import train_test_split, cross_val_score
from sklearn.dummy import DummyClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.naive_bayes import GaussianNB
from sklearn import svm
from sklearn.tree import DecisionTreeClassifier
from datetime import datetime

%matplotlib inline
```

```
[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)
```

2.1.1 portfolio dataset preparation

```
[3]: portfolio.shape
```

```
[3]: (10, 6)
```

```
[4]: portfolio.dtypes
```

```
[4]: channels      object
     difficulty    int64
     duration      int64
     id            object
     offer_type    object
     reward        int64
     dtype: object
```

```
[5]: portfolio.head(10)
```

```
[5]:
```

	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	fafdc668e3743c1bb461111dcafc2a4	discount	2
7	5a8bc65990b245e5a138643cd4eb9837	informational	0
8	f19421c1d4aa40978ebb69ca19b0e20d	bogo	5
9	2906b810c7d4411798c6938adc9daaa5	discount	2

We can see some variables that could use some preprocessing, such as `channels` which is a list of values. We'll use [Scikit-learn's MultiLabelBinarizer](#) to transform it.

Then we'll create a copy of the original `portfolio` dataset with the new encoded columns.

```
[6]: mlb = MultiLabelBinarizer()
     encoded_channels = mlb.fit_transform(portfolio['channels'])
```

```
[7]: encoded_channels_df = pd.DataFrame(encoded_channels,
                                         columns = mlb.classes_,
                                         index = portfolio['channels'].index)
     encoded_channels_df.head()
```

```
[7]:      email  mobile  social  web
0         1        1        1    0
1         1        1        1    1
2         1        1        0    1
3         1        1        0    1
4         1        0        0    1
```

```
[8]: processed_portfolio = portfolio.copy()

processed_portfolio = processed_portfolio.join(encoded_channels_df)
processed_portfolio.head()
```

```
[8]:      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

      id      offer_type  reward  email  mobile  \
0  ae264e3637204a6fb9bb56bc8210ddfd      bogo      10      1      1
1  4d5c57ea9a6940dd891ad53e9dbe8da0      bogo      10      1      1
2  3f207df678b143eea3cee63160fa8bed  informational      0      1      1
3  9b98b8c7a33c4b65b9aebfe6a799e6d9      bogo      5      1      1
4  0b1e1539f2cc45b7b9fa7c272da2e1d7      discount      5      1      0

      social  web
0         1    0
1         1    1
2         0    1
3         0    1
4         0    1
```

```
[9]: processed_portfolio = processed_portfolio.drop('channels', axis = 1)
print(processed_portfolio)
```

```
      difficulty  duration      id      offer_type  \
0         10      7  ae264e3637204a6fb9bb56bc8210ddfd      bogo
1         10      5  4d5c57ea9a6940dd891ad53e9dbe8da0      bogo
2          0      4  3f207df678b143eea3cee63160fa8bed  informational
3          5      7  9b98b8c7a33c4b65b9aebfe6a799e6d9      bogo
4         20     10  0b1e1539f2cc45b7b9fa7c272da2e1d7      discount
5          7      7  2298d6c36e964ae4a3e7e9706d1fb8c2      discount
6         10     10  fafdcd668e3743c1bb461111dcafc2a4      discount
7          0      3  5a8bc65990b245e5a138643cd4eb9837  informational
8          5      5  f19421c1d4aa40978ebb69ca19b0e20d      bogo
9         10      7  2906b810c7d4411798c6938adc9daaa5      discount
```

	reward	email	mobile	social	web
0	10	1	1	1	0
1	10	1	1	1	1
2	0	1	1	0	1
3	5	1	1	0	1
4	5	1	0	0	1
5	3	1	1	1	1
6	2	1	1	1	1
7	0	1	1	1	0
8	5	1	1	1	1
9	2	1	1	0	1

```
[10]: processed_portfolio.isna().sum()
```

```
[10]: difficulty    0
      duration      0
      id            0
      offer_type    0
      reward        0
      email         0
      mobile        0
      social        0
      web           0
      dtype: int64
```

No empty cells, looks like this dataset is good enough for now.

2.1.2 profile dataset preparation

```
[11]: profile.shape
```

```
[11]: (17000, 5)
```

```
[12]: profile.dtypes
```

```
[12]: age                int64
      became_member_on  int64
      gender            object
      id                object
      income            float64
      dtype: object
```

```
[13]: profile.head(10)
```

```
[13]:   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

The `profile` dataset has some interesting characteristics, such as: * `NaN` values for the `income` column. * `None` value for the `gender` column (which is ok, since there are more genders than M, F, but this might need to be treated since we have 0 value - that probably means ‘others’). * `became_member_on` has dates that are treated as numbers. * `age` has the value 118 repeated several times. Since it’s fairly unlikely for the customers to be 118 years-old, I’m assuming this is also some kind of previous data preprocessing and won’t change it.

```
[14]: profile['gender'].unique()
```

```
[14]: array([None, 'F', 'M', '0'], dtype=object)
```

```
[15]: profile.isna().head(10)
```

```
[15]:
```

	age	became_member_on	gender	id	income
0	False	False	True	False	True
1	False	False	False	False	False
2	False	False	True	False	True
3	False	False	False	False	False
4	False	False	True	False	True
5	False	False	False	False	False
6	False	False	True	False	True
7	False	False	True	False	True
8	False	False	False	False	False
9	False	False	True	False	True

```
[16]: profile.isna().sum()
```

```
[16]: age                0
became_member_on      0
gender               2175
id                   0
income              2175
dtype: int64
```

The `None` value for `gender` is not understood as a value, so it needs treatment. We’ll fill the `None` values with `Unknown`.

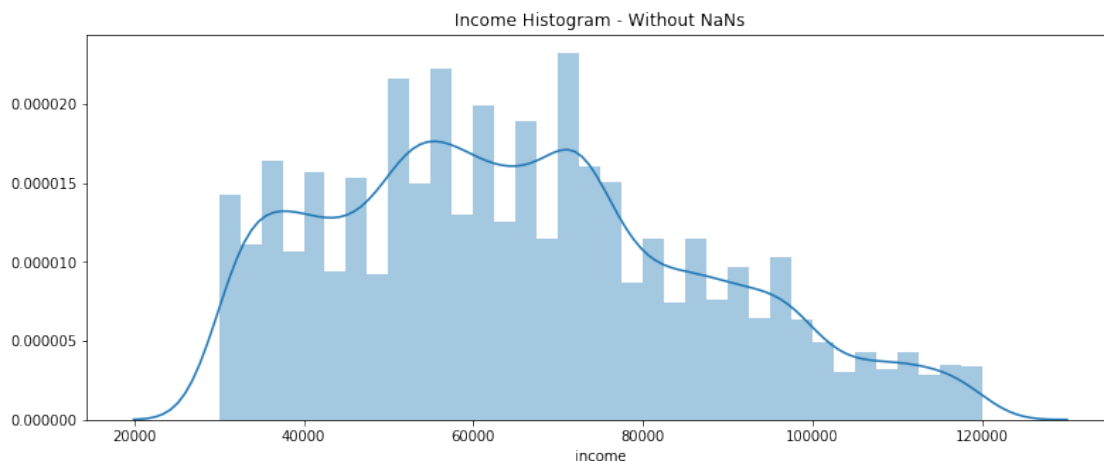
```
[17]: processed_profile = profile.copy()
```

```
processed_profile['gender'].fillna('Unknown', inplace = True)
processed_profile.head(10)
```

```
[17]:   age  became_member_on  gender  id  income
0   118      20170212  Unknown  68be06ca386d4c31939f3a4f0e3dd783    NaN
1    55      20170715      F    0610b486422d4921ae7d2bf64640c50b  112000.0
2   118      20180712  Unknown  38fe809add3b4fcf9315a9694bb96ff5    NaN
3    75      20170509      F    78afa995795e4d85b5d9ceeca43f5fef  100000.0
4   118      20170804  Unknown  a03223e636434f42ac4c3df47e8bac43    NaN
5    68      20180426      M    e2127556f4f64592b11af22de27a7932   70000.0
6   118      20170925  Unknown  8ec6ce2a7e7949b1bf142def7d0e0586    NaN
7   118      20171002  Unknown  68617ca6246f4fbc85e91a2a49552598    NaN
8    65      20180209      M    389bc3fa690240e798340f5a15918d5c   53000.0
9   118      20161122  Unknown  8974fc5686fe429db53ddde067b88302    NaN
```

For the income column, let's see what's the data distribution without the NaNs:

```
[18]: plt.figure(figsize = (13, 5))
sns.distplot(profile['income'].dropna());
plt.title('Income Histogram - Without NaNs');
```



```
[19]: profile['income'].describe()
```

```
[19]: count      14825.000000
mean       65404.991568
std       21598.299410
min       30000.000000
25%      49000.000000
50%      64000.000000
75%      80000.000000
max      120000.000000
```


Name: income, dtype: float64

By filling the NaNs with some value, we'll change the distribution's shape. Since both mean and median are fairly close to each other, it doesn't matter much which one we'll use. I chose the mean.

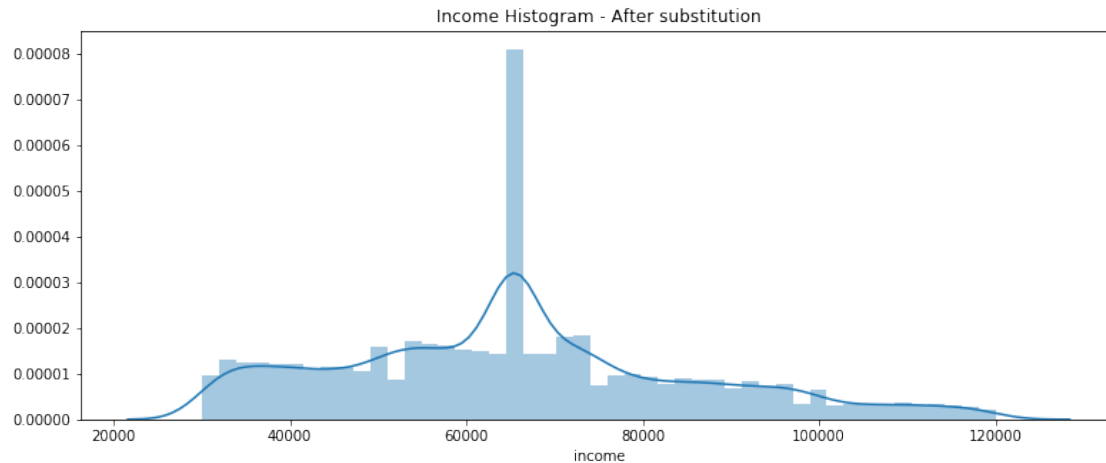
```
[20]: processed_profile['income'].fillna(profile['income'].mean(), inplace = True)
      processed_profile.head(10)
```

```
[20]:
```

	age	became_member_on	gender	id \
0	118	20170212	Unknown	68be06ca386d4c31939f3a4f0e3dd783
1	55	20170715	F	0610b486422d4921ae7d2bf64640c50b
2	118	20180712	Unknown	38fe809add3b4fcf9315a9694bb96ff5
3	75	20170509	F	78afa995795e4d85b5d9ceeca43f5fef
4	118	20170804	Unknown	a03223e636434f42ac4c3df47e8bac43
5	68	20180426	M	e2127556f4f64592b11af22de27a7932
6	118	20170925	Unknown	8ec6ce2a7e7949b1bf142def7d0e0586
7	118	20171002	Unknown	68617ca6246f4fbc85e91a2a49552598
8	65	20180209	M	389bc3fa690240e798340f5a15918d5c
9	118	20161122	Unknown	8974fc5686fe429db53ddde067b88302

	income
0	65404.991568
1	112000.000000
2	65404.991568
3	100000.000000
4	65404.991568
5	70000.000000
6	65404.991568
7	65404.991568
8	53000.000000
9	65404.991568

```
[21]: plt.figure(figsize = (13, 5))
      sns.distplot(processed_profile['income']);
      plt.title('Income Histogram - After substitution');
```



We can see it changed a bit the distribution of the data, but we were already expecting that.

```
[22]: processed_profile.isna().sum()
```

```
[22]: age           0
      became_member_on  0
      gender         0
      id             0
      income         0
      dtype: int64
```

No empty cells, looks like this dataset is good enough for now.

Another alteration we can work on is to transform the `became_member_on` column into a `datetime` object column, which allows us to work with time windows. This means that we can get the information for how long a customer has been a member of the program. Since we don't really know when those campaigns were build, we don't have a specific point in time to measure a customer's "age" as a customer. On the other hand, we can simply use the year the user signed up for the membership.

```
[23]: def convert_to_date(date_element):
      return datetime.strptime(str(date_element), '%Y%m%d')
```

```
[24]: became_member_on_date = processed_profile['became_member_on'].
      ↪ apply(convert_to_date)
```

```
[25]: became_member_on_date.head()
```

```
[25]: 0    2017-02-12
      1    2017-07-15
      2    2018-07-12
      3    2017-05-09
```

```
4    2017-08-04
Name: became_member_on, dtype: datetime64[ns]
```

```
[26]: became_member_on_year = became_member_on_date.apply(lambda date_point:
    ↪date_point.year)
```

```
[27]: became_member_on_year.head()
```

```
[27]: 0    2017
      1    2017
      2    2018
      3    2017
      4    2017
Name: became_member_on, dtype: int64
```

```
[28]: processed_profile['became_member_on_year'] = became_member_on_year
```

```
[29]: processed_profile = processed_profile.drop(['became_member_on'], axis = 1)
```

```
[30]: processed_profile.head()
```

```
[30]:   age  gender          id      income \
0  118  Unknown  68be06ca386d4c31939f3a4f0e3dd783    65404.991568
1   55         F  0610b486422d4921ae7d2bf64640c50b   112000.000000
2  118  Unknown  38fe809add3b4fcf9315a9694bb96ff5    65404.991568
3   75         F  78afa995795e4d85b5d9ceeca43f5fef   100000.000000
4  118  Unknown  a03223e636434f42ac4c3df47e8bac43    65404.991568

      became_member_on_year
0                2017
1                2017
2                2018
3                2017
4                2017
```

2.1.3 transcript dataset preparation

```
[31]: transcript.shape
```

```
[31]: (306534, 4)
```

```
[32]: transcript.dtypes
```

```
[32]: event      object
      person    object
      time      int64
      value     object
```

dtype: object

```
[33]: transcript.head(10)
```

```
[33]:
```

	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': 'fafdc668e3743c1bb461111dcafc2a4'}
4	{'offer id': '4d5c57ea9a6940dd891ad53e9dbe8da0'}
5	{'offer id': 'f19421c1d4aa40978ebb69ca19b0e20d'}
6	{'offer id': '2298d6c36e964ae4a3e7e9706d1fb8c2'}
7	{'offer id': '3f207df678b143eea3cee63160fa8bed'}
8	{'offer id': '0b1e1539f2cc45b7b9fa7c272da2e1d7'}
9	{'offer id': '0b1e1539f2cc45b7b9fa7c272da2e1d7'}

```
[34]: transcript.isna().sum()
```

```
[34]: event      0
      person    0
      time      0
      value     0
      dtype: int64
```

```
[35]: transcript.sample(10)
```

```
[35]:
```

	event	person	time	\
76736	offer completed	0999506e6713408b96827816f547f0ff	186	
8070	offer received	4972685b8b39473782a14f7709afe8b3	0	
305497	transaction	ad18ad432f5a40c7b8ccce4be4fb530c	714	
147920	offer viewed	a14d4f2ec359464f8d4aeac17b755903	396	
95745	transaction	31943ca096f1439596b2dff3da2448e0	252	
143173	transaction	c100c7f877fd45d7ab09717bf812dc47	378	
172874	offer viewed	8c20d3c8e4ef4ad997923937a2637851	420	
166318	offer viewed	b4869c96c07e4be3aa9f2fc6027fe098	408	

```

230843    offer viewed    3f0ee4aacad648a2b800641dc09003ad    534
284810    offer completed 760fe67c2f7f4a67b6e10fe5c8d129e3    630

```

```

                                value
76736    {'offer_id': 'ae264e3637204a6fb9bb56bc8210ddfd...
8070     {'offer_id': '0b1e1539f2cc45b7b9fa7c272da2e1d7'}
305497                                {'amount': 20.49}
147920    {'offer_id': 'f19421c1d4aa40978ebb69ca19b0e20d'}
95745                                {'amount': 13.9}
143173                                {'amount': 2.92}
172874    {'offer_id': '5a8bc65990b245e5a138643cd4eb9837'}
166318    {'offer_id': '2298d6c36e964ae4a3e7e9706d1fb8c2'}
230843    {'offer_id': 'fafdcd668e3743c1bb461111dcafc2a4'}
284810    {'offer_id': '2298d6c36e964ae4a3e7e9706d1fb8c2...

```

This dataset has no empty values. The `value` column has a dictionary that has different keys and values. It will be easier to have this column expanded into multiple columns to work with the events.

```
[36]: value_expanded = transcript['value'].apply(pd.Series)
      value_expanded.head()
```

```
[36]:
```

	offer_id	amount	offer_id	reward
0	9b98b8c7a33c4b65b9aebfe6a799e6d9	NaN	NaN	NaN
1	0b1e1539f2cc45b7b9fa7c272da2e1d7	NaN	NaN	NaN
2	2906b810c7d4411798c6938adc9daaa5	NaN	NaN	NaN
3	fafdc668e3743c1bb461111dcafc2a4	NaN	NaN	NaN
4	4d5c57ea9a6940dd891ad53e9dbe8da0	NaN	NaN	NaN

```
[37]: value_expanded.sample(10)
```

```
[37]:
```

	offer_id	amount	\
91816	f19421c1d4aa40978ebb69ca19b0e20d	NaN	
83549		NaN	
182786		NaN	
152065	f19421c1d4aa40978ebb69ca19b0e20d	NaN	
125504	4d5c57ea9a6940dd891ad53e9dbe8da0	NaN	
212858	4d5c57ea9a6940dd891ad53e9dbe8da0	NaN	
240218		NaN	3.36
206141	f19421c1d4aa40978ebb69ca19b0e20d	NaN	
104215		NaN	14.85
76467		NaN	10.31

	offer_id	reward
91816	NaN	NaN
83549	2298d6c36e964ae4a3e7e9706d1fb8c2	3.0
182786	ae264e3637204a6fb9bb56bc8210ddfd	10.0

152065	NaN	NaN
125504	NaN	NaN
212858	NaN	NaN
240218	NaN	NaN
206141	NaN	NaN
104215	NaN	NaN
76467	NaN	NaN

Now that we have this matrix, we can include it to the rest of the transcript data.

```
[38]: transcript_expanded = pd.concat([transcript, value_expanded], axis = 1)
transcript_expanded.head()
```

```
[38]:
```

	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	

	value	\
0	{'offer id': '9b98b8c7a33c4b65b9aebfe6a799e6d9'}	
1	{'offer id': '0b1e1539f2cc45b7b9fa7c272da2e1d7'}	
2	{'offer id': '2906b810c7d4411798c6938adc9daaa5'}	
3	{'offer id': 'fafdcd668e3743c1bb461111dcafc2a4'}	
4	{'offer id': '4d5c57ea9a6940dd891ad53e9dbe8da0'}	

	offer id	amount	offer_id	reward
0	9b98b8c7a33c4b65b9aebfe6a799e6d9	NaN	NaN	NaN
1	0b1e1539f2cc45b7b9fa7c272da2e1d7	NaN	NaN	NaN
2	2906b810c7d4411798c6938adc9daaa5	NaN	NaN	NaN
3	fafdcd668e3743c1bb461111dcafc2a4	NaN	NaN	NaN
4	4d5c57ea9a6940dd891ad53e9dbe8da0	NaN	NaN	NaN

```
[39]: transcript_expanded.sample(10)
```

```
[39]:
```

	event	person	time	\
2748	offer received	bd91b42ebde945ea81af9a64685b4b79	0	
301287	transaction	0922ba5634c748439e91d5c6ed793d8e	690	
128108	transaction	dd2f15d3224349b3a54cbb4a91e89e0c	342	
59268	offer received	8b9d3f738e7e499b9e820de07c16bc62	168	
45484	transaction	3bcc51fdde354eb1949c813dbc905182	120	
36160	transaction	0b3878783eed4009993f706e51f19d9e	72	
187911	offer viewed	8e7a22dd150f4ecea1a6be17be1d2381	456	
38931	offer viewed	79e0e0838fb4443e8fcd50efd81c0648	84	
212318	offer received	e44d3bde10b147568f9c11fb0414abf5	504	
208158	offer received	63bead43dc4a40f9a8da24856f98367b	504	

	value \
2748	{'offer id': '2906b810c7d4411798c6938adc9daaa5'}
301287	{'amount': 23.49}
128108	{'amount': 16.13}
59268	{'offer id': '5a8bc65990b245e5a138643cd4eb9837'}
45484	{'amount': 5.05}
36160	{'amount': 14.98}
187911	{'offer id': 'f19421c1d4aa40978ebb69ca19b0e20d'}
38931	{'offer id': '0b1e1539f2cc45b7b9fa7c272da2e1d7'}
212318	{'offer id': '2906b810c7d4411798c6938adc9daaa5'}
208158	{'offer id': 'f19421c1d4aa40978ebb69ca19b0e20d'}

	offer id	amount	offer_id	reward
2748	2906b810c7d4411798c6938adc9daaa5	NaN	NaN	NaN
301287	NaN	23.49	NaN	NaN
128108	NaN	16.13	NaN	NaN
59268	5a8bc65990b245e5a138643cd4eb9837	NaN	NaN	NaN
45484	NaN	5.05	NaN	NaN
36160	NaN	14.98	NaN	NaN
187911	f19421c1d4aa40978ebb69ca19b0e20d	NaN	NaN	NaN
38931	0b1e1539f2cc45b7b9fa7c272da2e1d7	NaN	NaN	NaN
212318	2906b810c7d4411798c6938adc9daaa5	NaN	NaN	NaN
208158	f19421c1d4aa40978ebb69ca19b0e20d	NaN	NaN	NaN

Now we can manipulate this dataset in order to clean it. We will deal with the duplicate `offer id/offer_id` column and remove unnecessary columns.

The `offer id/offer_id` duplicates seem to relate to the same information, but for different events, which means that each line can never have two values for `offer id/offer_id` (only one per line). To treat that, we can create a single column that receives the presented `offer id/offer_id` value when one exists.

```
[40]: transcript_expanded['offer_id_redux'] = np.where(transcript_expanded['offer_id']
↳ id'].isnull() & transcript_expanded['offer_id'].notnull(),
↳ transcript_expanded['offer_id'],
↳ transcript_expanded['offer_id'])
transcript_expanded.head()
```

```
[40]:      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
```

```

                                value \
0  {'offer id': '9b98b8c7a33c4b65b9aebfe6a799e6d9'}
1  {'offer id': '0b1e1539f2cc45b7b9fa7c272da2e1d7'}
2  {'offer id': '2906b810c7d4411798c6938adc9daaa5'}
3  {'offer id': 'fafdcd668e3743c1bb461111dcafc2a4'}
4  {'offer id': '4d5c57ea9a6940dd891ad53e9dbe8da0'}

                                offer id  amount offer_id  reward \
0  9b98b8c7a33c4b65b9aebfe6a799e6d9      NaN      NaN      NaN
1  0b1e1539f2cc45b7b9fa7c272da2e1d7      NaN      NaN      NaN
2  2906b810c7d4411798c6938adc9daaa5      NaN      NaN      NaN
3  fafdcd668e3743c1bb461111dcafc2a4      NaN      NaN      NaN
4  4d5c57ea9a6940dd891ad53e9dbe8da0      NaN      NaN      NaN

                                offer_id_redux
0  9b98b8c7a33c4b65b9aebfe6a799e6d9
1  0b1e1539f2cc45b7b9fa7c272da2e1d7
2  2906b810c7d4411798c6938adc9daaa5
3  fafdcd668e3743c1bb461111dcafc2a4
4  4d5c57ea9a6940dd891ad53e9dbe8da0

```

```
[41]: transcript_expanded.sample(10)
```

```

[41]:
                                event                person  time \
49025      transaction  1c79e428cf124617aad6f62c6cf496ba   138
151574  offer received  1679b7af5a294c6d9ae961232318ad55   408
25759      transaction  0063def0f9c14bc4805322a488839b32    30
12170  offer received  f0ea4fff9485045ffb647a09f3b9c3063     0
172029      transaction  7b262b5d16eb491e8681f2e44106d70e   420
10484  offer received  37155a5f4ff248529399c95472186879     0
193664      transaction  552d9d2c23f34ce8bf2f352378b4ca46   474
239390      transaction  c4a3c86c7b5541cf83d443c2a89e5b5a   558
156872  offer received  187bc1f183174e6799994488d7b2bf2b   408
89338      transaction  a349c2f0f90c430aae70eed8937c2fa   228

                                value \
49025                                {'amount': 4.06}
151574  {'offer id': '0b1e1539f2cc45b7b9fa7c272da2e1d7'}
25759                                {'amount': 2.76}
12170  {'offer id': 'ae264e3637204a6fb9bb56bc8210ddfd'}
172029                                {'amount': 1.12}
10484  {'offer id': '2298d6c36e964ae4a3e7e9706d1fb8c2'}
193664                                {'amount': 20.31}
239390                                {'amount': 27.24}
156872  {'offer id': 'f19421c1d4aa40978ebb69ca19b0e20d'}
89338                                {'amount': 5.25}

```


	offer_id	amount	offer_id	reward	\
49025	NaN	4.06	NaN	NaN	
151574	0b1e1539f2cc45b7b9fa7c272da2e1d7	NaN	NaN	NaN	
25759	NaN	2.76	NaN	NaN	
12170	ae264e3637204a6fb9bb56bc8210ddfd	NaN	NaN	NaN	
172029	NaN	1.12	NaN	NaN	
10484	2298d6c36e964ae4a3e7e9706d1fb8c2	NaN	NaN	NaN	
193664	NaN	20.31	NaN	NaN	
239390	NaN	27.24	NaN	NaN	
156872	f19421c1d4aa40978ebb69ca19b0e20d	NaN	NaN	NaN	
89338	NaN	5.25	NaN	NaN	

	offer_id_redux
49025	NaN
151574	0b1e1539f2cc45b7b9fa7c272da2e1d7
25759	NaN
12170	ae264e3637204a6fb9bb56bc8210ddfd
172029	NaN
10484	2298d6c36e964ae4a3e7e9706d1fb8c2
193664	NaN
239390	NaN
156872	f19421c1d4aa40978ebb69ca19b0e20d
89338	NaN

Now we can drop unnecessary columns.

```
[42]: transcript_expanded = transcript_expanded.drop(['value',
                                                    'offer_id',
                                                    'offer_id'],
                                                    axis = 1)
```

```
[43]: transcript_expanded.head()
```

	event	person	time	amount	reward	\
0	offer received	78afa995795e4d85b5d9ceeca43f5fef	0	NaN	NaN	
1	offer received	a03223e636434f42ac4c3df47e8bac43	0	NaN	NaN	
2	offer received	e2127556f4f64592b11af22de27a7932	0	NaN	NaN	
3	offer received	8ec6ce2a7e7949b1bf142def7d0e0586	0	NaN	NaN	
4	offer received	68617ca6246f4fbc85e91a2a49552598	0	NaN	NaN	

	offer_id_redux
0	9b98b8c7a33c4b65b9aebfe6a799e6d9
1	0b1e1539f2cc45b7b9fa7c272da2e1d7
2	2906b810c7d4411798c6938adc9daaa5
3	fafdc668e3743c1bb461111dcafc2a4
4	4d5c57ea9a6940dd891ad53e9dbe8da0

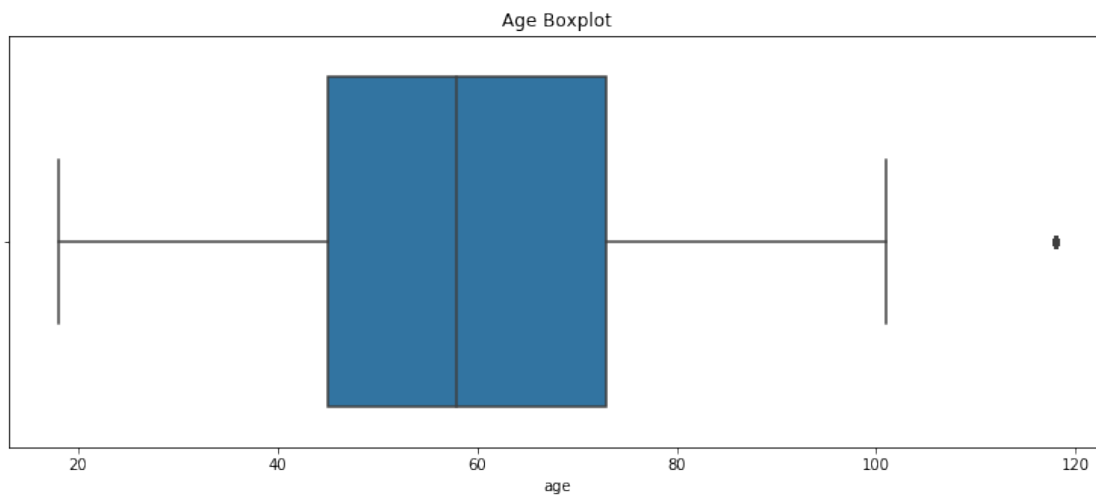
2.2 Exploratory Data Analysis

Starting with the `profile` dataset, we can check a bit of the age and income of the customers.

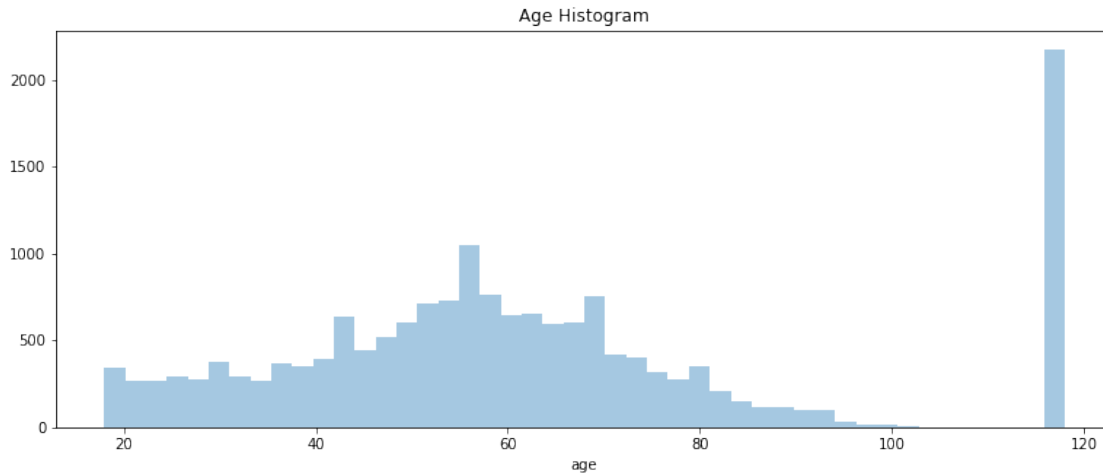
```
[44]: processed_profile['age'].describe()
```

```
[44]: count      17000.000000  
      mean        62.531412  
      std         26.738580  
      min         18.000000  
      25%         45.000000  
      50%         58.000000  
      75%         73.000000  
      max        118.000000  
      Name: age, dtype: float64
```

```
[45]: plt.figure(figsize = (13, 5))  
      sns.boxplot(processed_profile['age']);  
      plt.title('Age Boxplot');
```



```
[46]: plt.figure(figsize = (13, 5))  
      sns.distplot(processed_profile['age'], kde = False);  
      plt.title('Age Histogram');
```

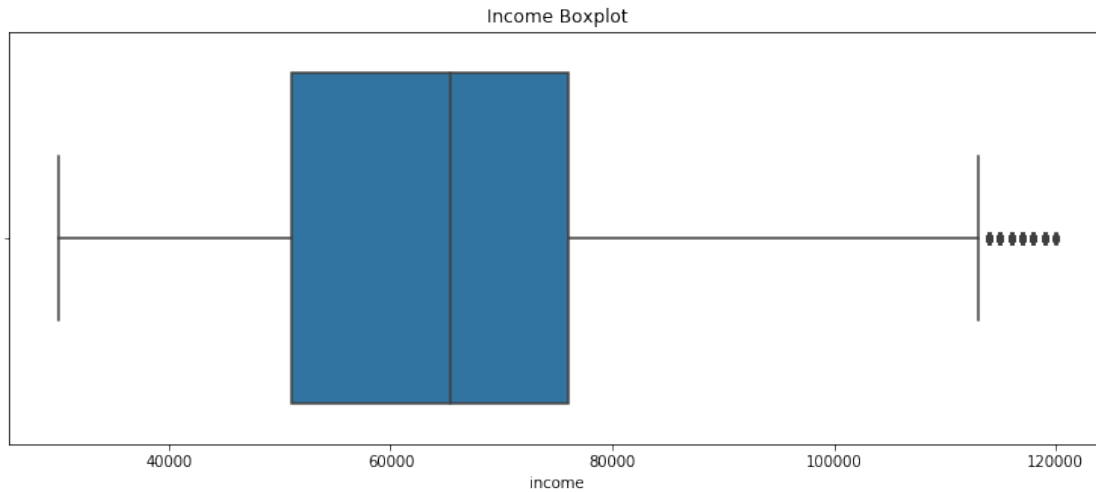


This histogram and the boxplot show us a great amount of outliers near 120. That was expected, since we decided not to treat the 118 values (as discussed earlier). The users are mostly adults, with ages between 40 and 80 and the median in a bit lower than 60 (which is also lower than the mean, that is 62.5).

```
[47]: processed_profile['income'].describe()
```

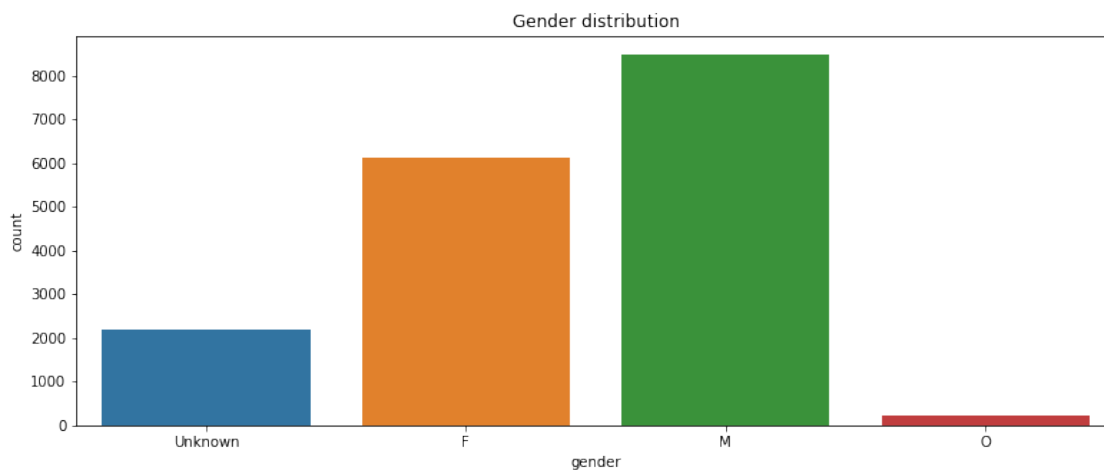
```
[47]: count      17000.000000
      mean       65404.991568
      std        20169.288288
      min        30000.000000
      25%        51000.000000
      50%        65404.991568
      75%        76000.000000
      max        120000.000000
      Name: income, dtype: float64
```

```
[48]: plt.figure(figsize = (13, 5))
      sns.boxplot(processed_profile['income']);
      plt.title('Income Boxplot');
```



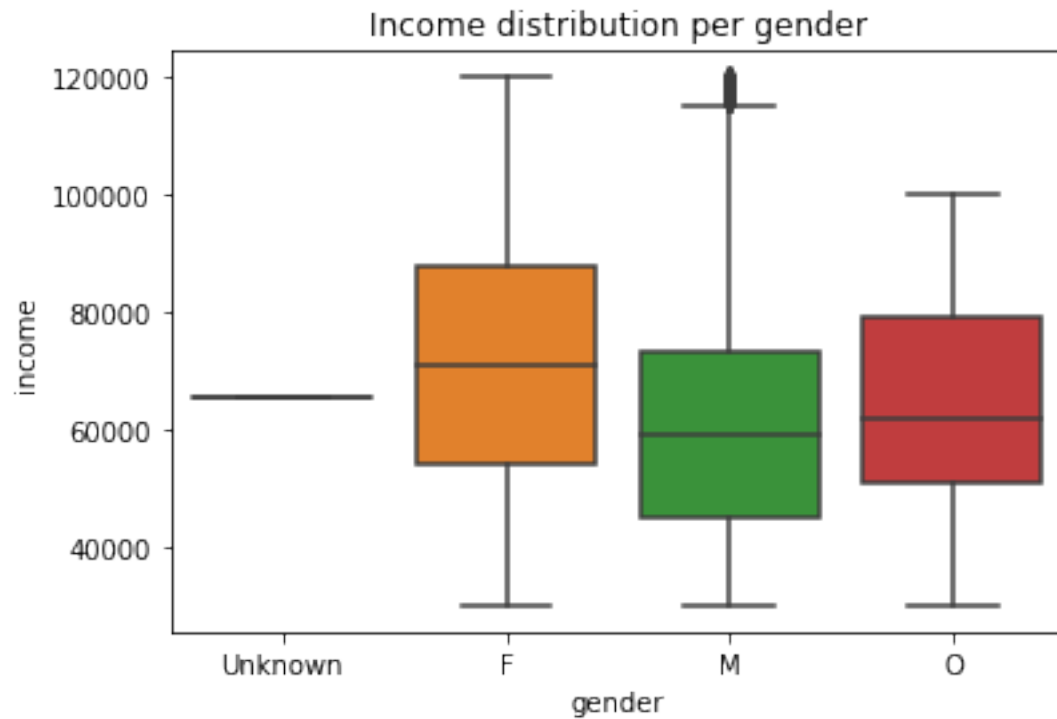
We already saw the histogram for this data, the boxplot shows the distribution is a bit skewed. Mean and median seem close (around \$65,000) and most of the data is between \$60,000 and \$75,000.

```
[49]: plt.figure(figsize = (13, 5))
sns.countplot(processed_profile['gender']);
plt.title('Gender distribution');
```



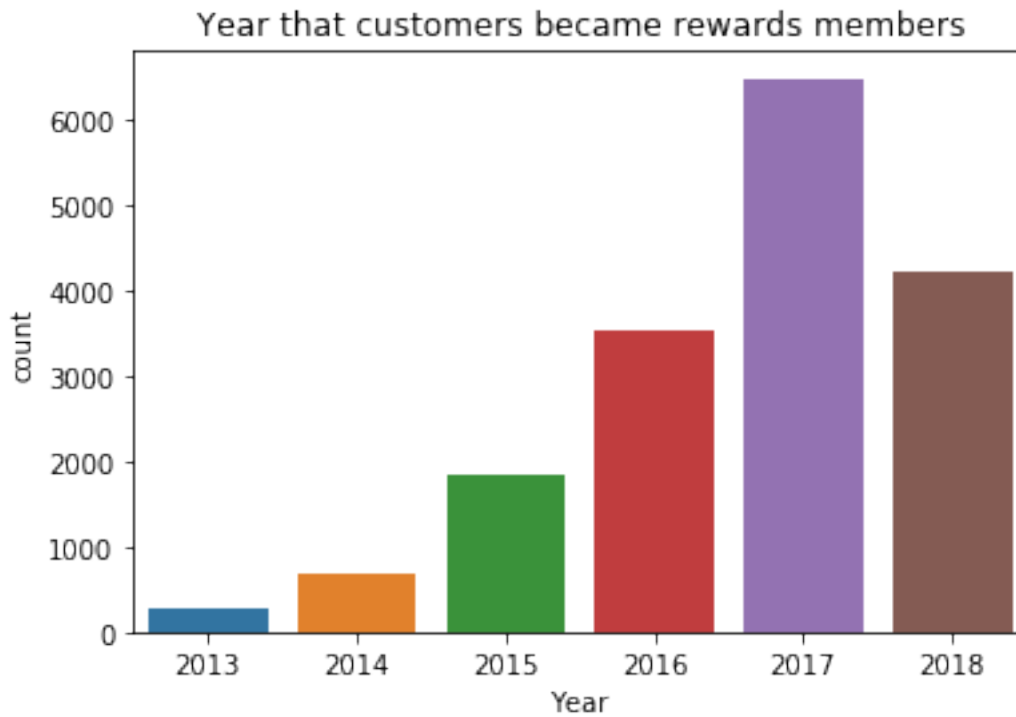
There are more male than females in the dataset. There is a large amount of `Unknown` and a few 'others'.

```
[50]: sns.boxplot(x = 'gender',
                  y = 'income',
                  data = processed_profile);
plt.title('Income distribution per gender');
```



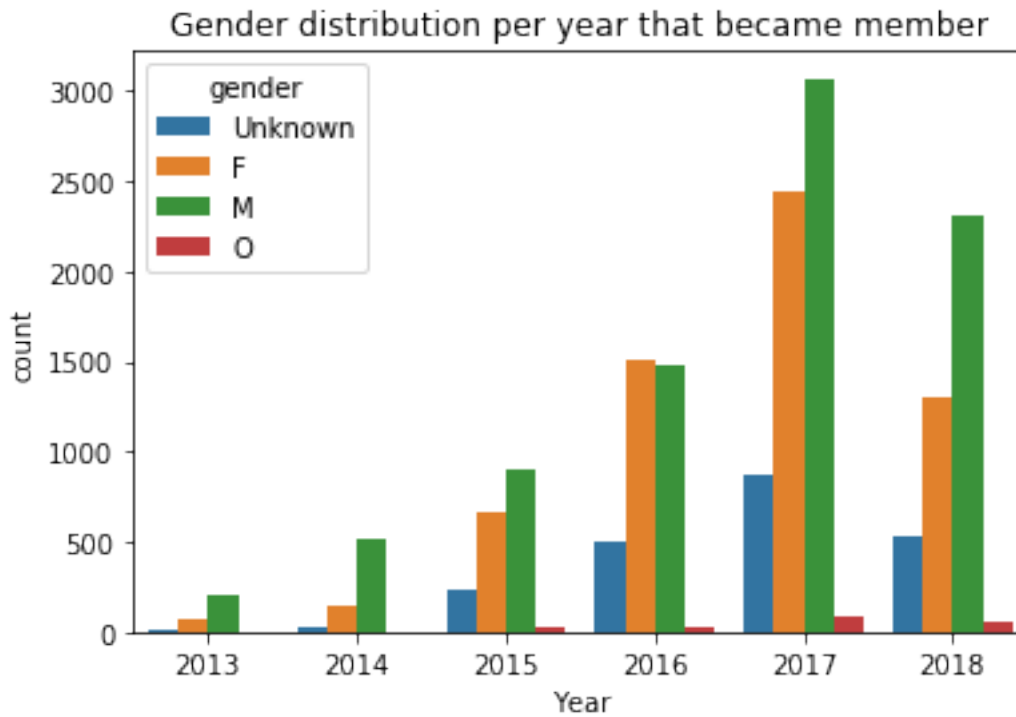
The boxplots show there is no evidence that the income differs accross genders.

```
[51]: sns.countplot('became_member_on_year',  
                  data = processed_profile);  
plt.title('Year that customers became rewards members')  
plt.xlabel('Year');
```



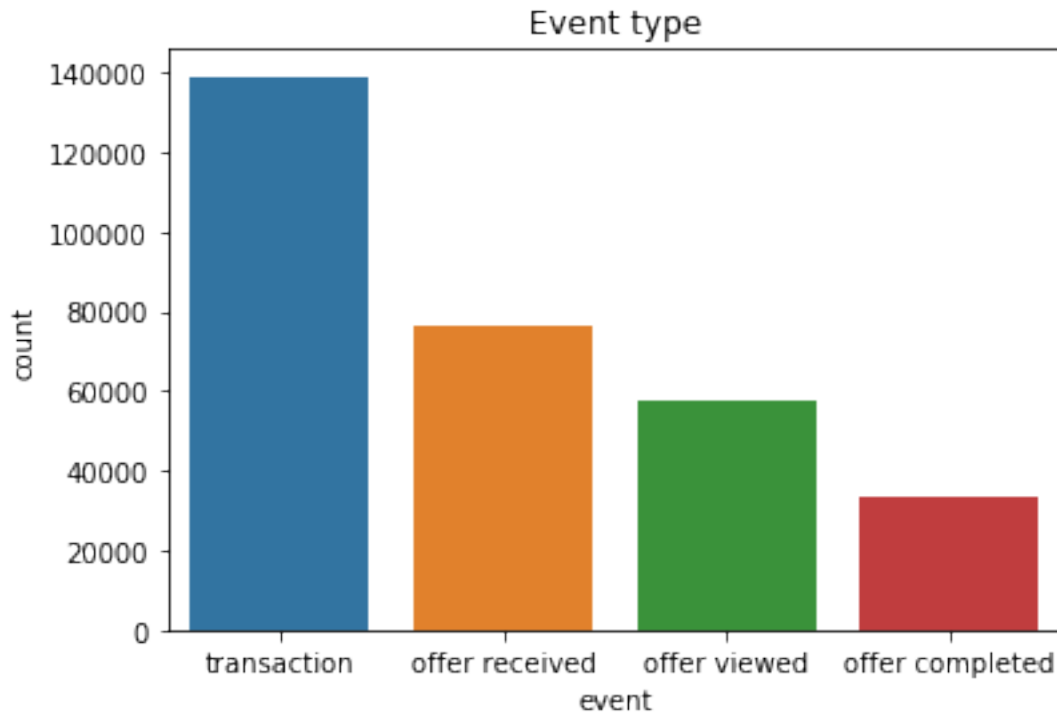
More customers became members in 2017, followed by 2018 then 2016. The first year the with records is 2013 and it's also the year with fewer members signing for the program.

```
[52]: sns.countplot(x = 'became_member_on_year',  
                    hue = 'gender',  
                    data = processed_profile);  
plt.xlabel('Year');  
plt.title('Gender distribution per year that became member');
```



In all years, there are either more male members signing up for the program (with exception of 2016, in which there were slight more female members registrations).

```
[53]: sns.countplot(transcript_expanded['event'],  
                    order = transcript_expanded['event'].value_counts().index;  
plt.title('Event type');
```



The event type that is most frequent is `transaction`, followed by `offer received`, `offer viewed` and `offer completed`. This makes sense, since it looks like a usual customer funnel for marketing.

2.3 Merging datasets and data preprocessing

We can get add more information to the `transcript_expanded` dataset by joining it to the `processed_portfolio` dataset, that contains information on the offers.

```
[54]: transcript_with_portfolio = transcript_expanded.merge(processed_portfolio,
                                                         how = 'left',
                                                         left_on = 'offer_id',
                                                         right_on = 'id')
```

```
[55]: transcript_with_portfolio.head()
```

```
[55]:
```

	event	person	time	amount	reward_x	\
0	offer received	78afa995795e4d85b5d9ceeca43f5fef	0	NaN	NaN	
1	offer received	a03223e636434f42ac4c3df47e8bac43	0	NaN	NaN	
2	offer received	e2127556f4f64592b11af22de27a7932	0	NaN	NaN	
3	offer received	8ec6ce2a7e7949b1bf142def7d0e0586	0	NaN	NaN	
4	offer received	68617ca6246f4fbc85e91a2a49552598	0	NaN	NaN	

```

offer_id_redux  difficulty  duration  \

```


0	9b98b8c7a33c4b65b9aebfe6a799e6d9	5.0	7.0
1	0b1e1539f2cc45b7b9fa7c272da2e1d7	20.0	10.0
2	2906b810c7d4411798c6938adc9daaa5	10.0	7.0
3	fafdc668e3743c1bb461111dcafc2a4	10.0	10.0
4	4d5c57ea9a6940dd891ad53e9dbe8da0	10.0	5.0

	id	offer_type	reward_y	email	mobile	\
0	9b98b8c7a33c4b65b9aebfe6a799e6d9	bogo	5.0	1.0	1.0	
1	0b1e1539f2cc45b7b9fa7c272da2e1d7	discount	5.0	1.0	0.0	
2	2906b810c7d4411798c6938adc9daaa5	discount	2.0	1.0	1.0	
3	fafdc668e3743c1bb461111dcafc2a4	discount	2.0	1.0	1.0	
4	4d5c57ea9a6940dd891ad53e9dbe8da0	bogo	10.0	1.0	1.0	

	social	web
0	0.0	1.0
1	0.0	1.0
2	0.0	1.0
3	1.0	1.0
4	1.0	1.0

```
[56]: # sort dataset by user and time, to make it easier for humans to see data
transcript_with_portfolio = transcript_with_portfolio.sort_values(['person',
↪ 'time'])
```

```
[57]: transcript_with_portfolio.head(50)
```

```
[57]:
```

	event	person	time	amount	\
55972	offer received	0009655768c64bdeb2e877511632db8f	168	NaN	
77705	offer viewed	0009655768c64bdeb2e877511632db8f	192	NaN	
89291	transaction	0009655768c64bdeb2e877511632db8f	228	22.16	
113605	offer received	0009655768c64bdeb2e877511632db8f	336	NaN	
139992	offer viewed	0009655768c64bdeb2e877511632db8f	372	NaN	
153401	offer received	0009655768c64bdeb2e877511632db8f	408	NaN	
168412	transaction	0009655768c64bdeb2e877511632db8f	414	8.57	
168413	offer completed	0009655768c64bdeb2e877511632db8f	414	NaN	
187554	offer viewed	0009655768c64bdeb2e877511632db8f	456	NaN	
204340	offer received	0009655768c64bdeb2e877511632db8f	504	NaN	
228422	transaction	0009655768c64bdeb2e877511632db8f	528	14.11	
228423	offer completed	0009655768c64bdeb2e877511632db8f	528	NaN	
233413	offer viewed	0009655768c64bdeb2e877511632db8f	540	NaN	
237784	transaction	0009655768c64bdeb2e877511632db8f	552	13.56	
247879	offer received	0009655768c64bdeb2e877511632db8f	576	NaN	
258883	transaction	0009655768c64bdeb2e877511632db8f	576	10.27	
258884	offer completed	0009655768c64bdeb2e877511632db8f	576	NaN	
293497	transaction	0009655768c64bdeb2e877511632db8f	660	12.36	
300930	transaction	0009655768c64bdeb2e877511632db8f	690	28.16	
302205	transaction	0009655768c64bdeb2e877511632db8f	696	18.41	

56475	offer received	00116118485d4dfda04fdbaba9a87b5c	168	NaN
85769	offer viewed	00116118485d4dfda04fdbaba9a87b5c	216	NaN
104088	transaction	00116118485d4dfda04fdbaba9a87b5c	294	0.70
187632	transaction	00116118485d4dfda04fdbaba9a87b5c	456	0.20
193680	transaction	00116118485d4dfda04fdbaba9a87b5c	474	3.19
248359	offer received	00116118485d4dfda04fdbaba9a87b5c	576	NaN
284472	offer viewed	00116118485d4dfda04fdbaba9a87b5c	630	NaN
3066	offer received	0011e0d4e6b944f998e987f904e8c1e5	0	NaN
16179	offer viewed	0011e0d4e6b944f998e987f904e8c1e5	6	NaN
47805	transaction	0011e0d4e6b944f998e987f904e8c1e5	132	13.49
56298	offer received	0011e0d4e6b944f998e987f904e8c1e5	168	NaN
75427	offer viewed	0011e0d4e6b944f998e987f904e8c1e5	186	NaN
95421	transaction	0011e0d4e6b944f998e987f904e8c1e5	252	11.93
95422	offer completed	0011e0d4e6b944f998e987f904e8c1e5	252	NaN
113919	offer received	0011e0d4e6b944f998e987f904e8c1e5	336	NaN
133370	offer viewed	0011e0d4e6b944f998e987f904e8c1e5	354	NaN
153697	offer received	0011e0d4e6b944f998e987f904e8c1e5	408	NaN
177937	offer viewed	0011e0d4e6b944f998e987f904e8c1e5	432	NaN
204643	offer received	0011e0d4e6b944f998e987f904e8c1e5	504	NaN
222679	offer viewed	0011e0d4e6b944f998e987f904e8c1e5	516	NaN
258979	transaction	0011e0d4e6b944f998e987f904e8c1e5	576	22.05
258980	offer completed	0011e0d4e6b944f998e987f904e8c1e5	576	NaN
258981	offer completed	0011e0d4e6b944f998e987f904e8c1e5	576	NaN
288294	transaction	0011e0d4e6b944f998e987f904e8c1e5	642	23.03
291925	transaction	0011e0d4e6b944f998e987f904e8c1e5	654	8.96
1889	offer received	0020c2b971eb4e9188eac86d93036a77	0	NaN
18431	offer viewed	0020c2b971eb4e9188eac86d93036a77	12	NaN
31326	transaction	0020c2b971eb4e9188eac86d93036a77	54	17.63
31327	offer completed	0020c2b971eb4e9188eac86d93036a77	54	NaN
34287	transaction	0020c2b971eb4e9188eac86d93036a77	66	32.00

	reward_x	offer_id_redux	difficulty	duration	\
55972	NaN	5a8bc65990b245e5a138643cd4eb9837	0.0	3.0	
77705	NaN	5a8bc65990b245e5a138643cd4eb9837	0.0	3.0	
89291	NaN	NaN	NaN	NaN	
113605	NaN	3f207df678b143eea3cee63160fa8bed	0.0	4.0	
139992	NaN	3f207df678b143eea3cee63160fa8bed	0.0	4.0	
153401	NaN	f19421c1d4aa40978ebb69ca19b0e20d	5.0	5.0	
168412	NaN	NaN	NaN	NaN	
168413	5.0	f19421c1d4aa40978ebb69ca19b0e20d	5.0	5.0	
187554	NaN	f19421c1d4aa40978ebb69ca19b0e20d	5.0	5.0	
204340	NaN	fafdc668e3743c1bb461111dcafc2a4	10.0	10.0	
228422	NaN	NaN	NaN	NaN	
228423	2.0	fafdc668e3743c1bb461111dcafc2a4	10.0	10.0	
233413	NaN	fafdc668e3743c1bb461111dcafc2a4	10.0	10.0	
237784	NaN	NaN	NaN	NaN	
247879	NaN	2906b810c7d4411798c6938adc9daaa5	10.0	7.0	

258883	NaN	NaN	NaN	NaN
258884	2.0	2906b810c7d4411798c6938adc9daaa5	10.0	7.0
293497	NaN	NaN	NaN	NaN
300930	NaN	NaN	NaN	NaN
302205	NaN	NaN	NaN	NaN
56475	NaN	f19421c1d4aa40978ebb69ca19b0e20d	5.0	5.0
85769	NaN	f19421c1d4aa40978ebb69ca19b0e20d	5.0	5.0
104088	NaN	NaN	NaN	NaN
187632	NaN	NaN	NaN	NaN
193680	NaN	NaN	NaN	NaN
248359	NaN	f19421c1d4aa40978ebb69ca19b0e20d	5.0	5.0
284472	NaN	f19421c1d4aa40978ebb69ca19b0e20d	5.0	5.0
3066	NaN	3f207df678b143eea3cee63160fa8bed	0.0	4.0
16179	NaN	3f207df678b143eea3cee63160fa8bed	0.0	4.0
47805	NaN	NaN	NaN	NaN
56298	NaN	2298d6c36e964ae4a3e7e9706d1fb8c2	7.0	7.0
75427	NaN	2298d6c36e964ae4a3e7e9706d1fb8c2	7.0	7.0
95421	NaN	NaN	NaN	NaN
95422	3.0	2298d6c36e964ae4a3e7e9706d1fb8c2	7.0	7.0
113919	NaN	5a8bc65990b245e5a138643cd4eb9837	0.0	3.0
133370	NaN	5a8bc65990b245e5a138643cd4eb9837	0.0	3.0
153697	NaN	0b1e1539f2cc45b7b9fa7c272da2e1d7	20.0	10.0
177937	NaN	0b1e1539f2cc45b7b9fa7c272da2e1d7	20.0	10.0
204643	NaN	9b98b8c7a33c4b65b9aebfe6a799e6d9	5.0	7.0
222679	NaN	9b98b8c7a33c4b65b9aebfe6a799e6d9	5.0	7.0
258979	NaN	NaN	NaN	NaN
258980	5.0	0b1e1539f2cc45b7b9fa7c272da2e1d7	20.0	10.0
258981	5.0	9b98b8c7a33c4b65b9aebfe6a799e6d9	5.0	7.0
288294	NaN	NaN	NaN	NaN
291925	NaN	NaN	NaN	NaN
1889	NaN	fafdc668e3743c1bb461111dcafc2a4	10.0	10.0
18431	NaN	fafdc668e3743c1bb461111dcafc2a4	10.0	10.0
31326	NaN	NaN	NaN	NaN
31327	2.0	fafdc668e3743c1bb461111dcafc2a4	10.0	10.0
34287	NaN	NaN	NaN	NaN

	id	offer_type	reward_y	email \
55972	5a8bc65990b245e5a138643cd4eb9837	informational	0.0	1.0
77705	5a8bc65990b245e5a138643cd4eb9837	informational	0.0	1.0
89291	NaN	NaN	NaN	NaN
113605	3f207df678b143eea3cee63160fa8bed	informational	0.0	1.0
139992	3f207df678b143eea3cee63160fa8bed	informational	0.0	1.0
153401	f19421c1d4aa40978ebb69ca19b0e20d	bogo	5.0	1.0
168412	NaN	NaN	NaN	NaN
168413	f19421c1d4aa40978ebb69ca19b0e20d	bogo	5.0	1.0
187554	f19421c1d4aa40978ebb69ca19b0e20d	bogo	5.0	1.0
204340	fafdc668e3743c1bb461111dcafc2a4	discount	2.0	1.0

228422		NaN	NaN	NaN	NaN
228423	fafdc668e3743c1bb461111dcafc2a4	discount	2.0	1.0	
233413	fafdc668e3743c1bb461111dcafc2a4	discount	2.0	1.0	
237784		NaN	NaN	NaN	NaN
247879	2906b810c7d4411798c6938adc9daaa5	discount	2.0	1.0	
258883		NaN	NaN	NaN	NaN
258884	2906b810c7d4411798c6938adc9daaa5	discount	2.0	1.0	
293497		NaN	NaN	NaN	NaN
300930		NaN	NaN	NaN	NaN
302205		NaN	NaN	NaN	NaN
56475	f19421c1d4aa40978ebb69ca19b0e20d	bogo	5.0	1.0	
85769	f19421c1d4aa40978ebb69ca19b0e20d	bogo	5.0	1.0	
104088		NaN	NaN	NaN	NaN
187632		NaN	NaN	NaN	NaN
193680		NaN	NaN	NaN	NaN
248359	f19421c1d4aa40978ebb69ca19b0e20d	bogo	5.0	1.0	
284472	f19421c1d4aa40978ebb69ca19b0e20d	bogo	5.0	1.0	
3066	3f207df678b143eea3cee63160fa8bed	informational	0.0	1.0	
16179	3f207df678b143eea3cee63160fa8bed	informational	0.0	1.0	
47805		NaN	NaN	NaN	NaN
56298	2298d6c36e964ae4a3e7e9706d1fb8c2	discount	3.0	1.0	
75427	2298d6c36e964ae4a3e7e9706d1fb8c2	discount	3.0	1.0	
95421		NaN	NaN	NaN	NaN
95422	2298d6c36e964ae4a3e7e9706d1fb8c2	discount	3.0	1.0	
113919	5a8bc65990b245e5a138643cd4eb9837	informational	0.0	1.0	
133370	5a8bc65990b245e5a138643cd4eb9837	informational	0.0	1.0	
153697	0b1e1539f2cc45b7b9fa7c272da2e1d7	discount	5.0	1.0	
177937	0b1e1539f2cc45b7b9fa7c272da2e1d7	discount	5.0	1.0	
204643	9b98b8c7a33c4b65b9aebfe6a799e6d9	bogo	5.0	1.0	
222679	9b98b8c7a33c4b65b9aebfe6a799e6d9	bogo	5.0	1.0	
258979		NaN	NaN	NaN	NaN
258980	0b1e1539f2cc45b7b9fa7c272da2e1d7	discount	5.0	1.0	
258981	9b98b8c7a33c4b65b9aebfe6a799e6d9	bogo	5.0	1.0	
288294		NaN	NaN	NaN	NaN
291925		NaN	NaN	NaN	NaN
1889	fafdc668e3743c1bb461111dcafc2a4	discount	2.0	1.0	
18431	fafdc668e3743c1bb461111dcafc2a4	discount	2.0	1.0	
31326		NaN	NaN	NaN	NaN
31327	fafdc668e3743c1bb461111dcafc2a4	discount	2.0	1.0	
34287		NaN	NaN	NaN	NaN

	mobile	social	web
55972	1.0	1.0	0.0
77705	1.0	1.0	0.0
89291	NaN	NaN	NaN
113605	1.0	0.0	1.0
139992	1.0	0.0	1.0

153401	1.0	1.0	1.0
168412	NaN	NaN	NaN
168413	1.0	1.0	1.0
187554	1.0	1.0	1.0
204340	1.0	1.0	1.0
228422	NaN	NaN	NaN
228423	1.0	1.0	1.0
233413	1.0	1.0	1.0
237784	NaN	NaN	NaN
247879	1.0	0.0	1.0
258883	NaN	NaN	NaN
258884	1.0	0.0	1.0
293497	NaN	NaN	NaN
300930	NaN	NaN	NaN
302205	NaN	NaN	NaN
56475	1.0	1.0	1.0
85769	1.0	1.0	1.0
104088	NaN	NaN	NaN
187632	NaN	NaN	NaN
193680	NaN	NaN	NaN
248359	1.0	1.0	1.0
284472	1.0	1.0	1.0
3066	1.0	0.0	1.0
16179	1.0	0.0	1.0
47805	NaN	NaN	NaN
56298	1.0	1.0	1.0
75427	1.0	1.0	1.0
95421	NaN	NaN	NaN
95422	1.0	1.0	1.0
113919	1.0	1.0	0.0
133370	1.0	1.0	0.0
153697	0.0	0.0	1.0
177937	0.0	0.0	1.0
204643	1.0	0.0	1.0
222679	1.0	0.0	1.0
258979	NaN	NaN	NaN
258980	0.0	0.0	1.0
258981	1.0	0.0	1.0
288294	NaN	NaN	NaN
291925	NaN	NaN	NaN
1889	1.0	1.0	1.0
18431	1.0	1.0	1.0
31326	NaN	NaN	NaN
31327	1.0	1.0	1.0
34287	NaN	NaN	NaN

```
[58]: transcript_with_portfolio.groupby(['event', 'offer_type'])['offer_type'].count()
```

```
[58]: event          offer_type
      offer completed  bogo          15669
              discount  17910
      offer received  bogo          30499
              discount  30543
              informational  15235
      offer viewed    bogo          25449
              discount  21445
              informational  10831
      Name: offer_type, dtype: int64
```

BOGO and discount offers are the only ones with an associated `offer completed` event. Also, the `transaction` event doesn't have an associated id, which means that the join won't add data to those.

Basically, we can look into the BOGO and discount offers user funnels in this order: * offer received * offer viewed * transaction * offer completed

For the informational offer, the funnel has this order: * offer received * offer viewed * transaction

This means that we have to find a way to add the transaction data into the funnel. This can be done through the user id and the timestamp. If a user didn't get to the transaction part, it won't have an `offer completed` event as well. For the informational offer, the transaction data will only exist if it came after the `offer viewed` event. The timestamp will help us find the "organic" users: users that became members regardless of the campaigns. All those scenarios can be found by ordering data by time, offer id, person and event.

```
[59]: view_to_complete = transcript_with_portfolio[['time', 'offer_id_redux',
→ 'person', 'event']] [(transcript_with_portfolio['event']=='transaction') |
→ (transcript_with_portfolio['event'] == 'offer viewed')].
→ groupby(['person', 'offer_id_redux'])
```

```
[60]: view_to_complete.head()
```

```
[60]:      time          offer_id_redux  \
77705    192  5a8bc65990b245e5a138643cd4eb9837
89291    228                                     NaN
139992   372  3f207df678b143eea3cee63160fa8bed
168412   414                                     NaN
187554   456  f19421c1d4aa40978ebb69ca19b0e20d
228422   528                                     NaN
233413   540  fafdcd668e3743c1bb461111dcafc2a4
237784   552                                     NaN
258883   576                                     NaN
85769    216  f19421c1d4aa40978ebb69ca19b0e20d
284472   630  f19421c1d4aa40978ebb69ca19b0e20d
16179     6  3f207df678b143eea3cee63160fa8bed
75427   186  2298d6c36e964ae4a3e7e9706d1fb8c2
133370   354  5a8bc65990b245e5a138643cd4eb9837
```

177937	432	0b1e1539f2cc45b7b9fa7c272da2e1d7
222679	516	9b98b8c7a33c4b65b9aebfe6a799e6d9
18431	12	fafdc668e3743c1bb461111dcafc2a4
174774	426	4d5c57ea9a6940dd891ad53e9dbe8da0
293372	660	5a8bc65990b245e5a138643cd4eb9837
67584	168	2298d6c36e964ae4a3e7e9706d1fb8c2
131476	348	f19421c1d4aa40978ebb69ca19b0e20d
165442	408	5a8bc65990b245e5a138643cd4eb9837
263808	582	9b98b8c7a33c4b65b9aebfe6a799e6d9
27148	36	5a8bc65990b245e5a138643cd4eb9837
105508	300	fafdc668e3743c1bb461111dcafc2a4
140716	372	3f207df678b143eea3cee63160fa8bed
173080	420	fafdc668e3743c1bb461111dcafc2a4
27263	36	5a8bc65990b245e5a138643cd4eb9837
76441	186	fafdc668e3743c1bb461111dcafc2a4
219332	510	f19421c1d4aa40978ebb69ca19b0e20d
...
26549	36	fafdc668e3743c1bb461111dcafc2a4
72825	180	4d5c57ea9a6940dd891ad53e9dbe8da0
269576	594	f19421c1d4aa40978ebb69ca19b0e20d
103669	288	3f207df678b143eea3cee63160fa8bed
148670	396	4d5c57ea9a6940dd891ad53e9dbe8da0
287565	636	3f207df678b143eea3cee63160fa8bed
68851	168	fafdc668e3743c1bb461111dcafc2a4
132444	348	4d5c57ea9a6940dd891ad53e9dbe8da0
182159	438	f19421c1d4aa40978ebb69ca19b0e20d
221408	510	4d5c57ea9a6940dd891ad53e9dbe8da0
289457	642	ae264e3637204a6fb9bb56bc8210ddfd
16325	6	fafdc668e3743c1bb461111dcafc2a4
83982	210	9b98b8c7a33c4b65b9aebfe6a799e6d9
233587	540	5a8bc65990b245e5a138643cd4eb9837
23028	24	fafdc668e3743c1bb461111dcafc2a4
73167	180	4d5c57ea9a6940dd891ad53e9dbe8da0
168973	414	ae264e3637204a6fb9bb56bc8210ddfd
226096	522	fafdc668e3743c1bb461111dcafc2a4
177513	432	fafdc668e3743c1bb461111dcafc2a4
293268	660	4d5c57ea9a6940dd891ad53e9dbe8da0
15591	6	f19421c1d4aa40978ebb69ca19b0e20d
65899	168	5a8bc65990b245e5a138643cd4eb9837
218451	510	f19421c1d4aa40978ebb69ca19b0e20d
294735	666	9b98b8c7a33c4b65b9aebfe6a799e6d9
15836	6	fafdc668e3743c1bb461111dcafc2a4
69626	174	0b1e1539f2cc45b7b9fa7c272da2e1d7
133074	354	2906b810c7d4411798c6938adc9daaa5
168022	414	2906b810c7d4411798c6938adc9daaa5
230690	534	9b98b8c7a33c4b65b9aebfe6a799e6d9
262475	582	2906b810c7d4411798c6938adc9daaa5

	person	event
77705	0009655768c64bdeb2e877511632db8f	offer viewed
89291	0009655768c64bdeb2e877511632db8f	transaction
139992	0009655768c64bdeb2e877511632db8f	offer viewed
168412	0009655768c64bdeb2e877511632db8f	transaction
187554	0009655768c64bdeb2e877511632db8f	offer viewed
228422	0009655768c64bdeb2e877511632db8f	transaction
233413	0009655768c64bdeb2e877511632db8f	offer viewed
237784	0009655768c64bdeb2e877511632db8f	transaction
258883	0009655768c64bdeb2e877511632db8f	transaction
85769	00116118485d4dfda04fdbaba9a87b5c	offer viewed
284472	00116118485d4dfda04fdbaba9a87b5c	offer viewed
16179	0011e0d4e6b944f998e987f904e8c1e5	offer viewed
75427	0011e0d4e6b944f998e987f904e8c1e5	offer viewed
133370	0011e0d4e6b944f998e987f904e8c1e5	offer viewed
177937	0011e0d4e6b944f998e987f904e8c1e5	offer viewed
222679	0011e0d4e6b944f998e987f904e8c1e5	offer viewed
18431	0020c2b971eb4e9188eac86d93036a77	offer viewed
174774	0020c2b971eb4e9188eac86d93036a77	offer viewed
293372	0020c2b971eb4e9188eac86d93036a77	offer viewed
67584	0020ccbbb6d84e358d3414a3ff76cffd	offer viewed
131476	0020ccbbb6d84e358d3414a3ff76cffd	offer viewed
165442	0020ccbbb6d84e358d3414a3ff76cffd	offer viewed
263808	0020ccbbb6d84e358d3414a3ff76cffd	offer viewed
27148	003d66b6608740288d6cc97a6903f4f0	offer viewed
105508	003d66b6608740288d6cc97a6903f4f0	offer viewed
140716	003d66b6608740288d6cc97a6903f4f0	offer viewed
173080	003d66b6608740288d6cc97a6903f4f0	offer viewed
27263	00426fe3ffde4c6b9cb9ad6d077a13ea	offer viewed
76441	00426fe3ffde4c6b9cb9ad6d077a13ea	offer viewed
219332	004b041fbfe44859945daa2c7f79ee64	offer viewed
...
26549	ffede3b700ac41d6a266fa1ba74b4f16	offer viewed
72825	ffede3b700ac41d6a266fa1ba74b4f16	offer viewed
269576	ffede3b700ac41d6a266fa1ba74b4f16	offer viewed
103669	fff0f0aac6c547b9b263080f09a5586a	offer viewed
148670	fff0f0aac6c547b9b263080f09a5586a	offer viewed
287565	fff0f0aac6c547b9b263080f09a5586a	offer viewed
68851	fff29fb549084123bd046dbc5ceb4faa	offer viewed
132444	fff29fb549084123bd046dbc5ceb4faa	offer viewed
182159	fff29fb549084123bd046dbc5ceb4faa	offer viewed
221408	fff29fb549084123bd046dbc5ceb4faa	offer viewed
289457	fff29fb549084123bd046dbc5ceb4faa	offer viewed
16325	fff3ba4757bd42088c044ca26d73817a	offer viewed
83982	fff3ba4757bd42088c044ca26d73817a	offer viewed
233587	fff3ba4757bd42088c044ca26d73817a	offer viewed


```

23028    fff7576017104bcc8677a8d63322b5e1    offer viewed
73167    fff7576017104bcc8677a8d63322b5e1    offer viewed
168973   fff7576017104bcc8677a8d63322b5e1    offer viewed
226096   fff7576017104bcc8677a8d63322b5e1    offer viewed
177513   fff8957ea8b240a6b5e634b6ee8eafcf    offer viewed
293268   fff8957ea8b240a6b5e634b6ee8eafcf    offer viewed
15591    fffad4f4828548d1b5583907f2e9906b    offer viewed
65899    fffad4f4828548d1b5583907f2e9906b    offer viewed
218451   fffad4f4828548d1b5583907f2e9906b    offer viewed
294735   fffad4f4828548d1b5583907f2e9906b    offer viewed
15836    ffff82501cea40309d5fdd7edcca4a07    offer viewed
69626    ffff82501cea40309d5fdd7edcca4a07    offer viewed
133074   ffff82501cea40309d5fdd7edcca4a07    offer viewed
168022   ffff82501cea40309d5fdd7edcca4a07    offer viewed
230690   ffff82501cea40309d5fdd7edcca4a07    offer viewed
262475   ffff82501cea40309d5fdd7edcca4a07    offer viewed

```

```
[57730 rows x 4 columns]
```

Since the dataset is grouped and ordered, we can simply fill the offer id gaps with the previous value.

```
[61]: fill_offer_id = view_to_complete['offer_id_redux'].
      ↪ fillna(view_to_complete['offer_id_redux'].ffill()).ffill()
```

```
[62]: fill_offer_id.head()
```

```

[62]: 77705      5a8bc65990b245e5a138643cd4eb9837
      89291      5a8bc65990b245e5a138643cd4eb9837
      139992     3f207df678b143eea3cee63160fa8bed
      168412     3f207df678b143eea3cee63160fa8bed
      187554     f19421c1d4aa40978ebb69ca19b0e20d
      Name: offer_id_redux, dtype: object

```

Since this column is indexed, we can join it with the original dataset and add the missing values to either one of the redundant columns (we will actually make a new column with the values for sanity check). Then drop the unnecessary columns.

```
[63]: with_full_offer_id = transcript_with_portfolio.join(fill_offer_id,
                                                         rsuffix = '_right')
```

```
[64]: with_full_offer_id.head()
```

```

[64]:          event          person  time  amount  \
55972  offer received  0009655768c64bdeb2e877511632db8f  168    NaN
77705    offer viewed  0009655768c64bdeb2e877511632db8f  192    NaN
89291    transaction  0009655768c64bdeb2e877511632db8f  228  22.16
113605  offer received  0009655768c64bdeb2e877511632db8f  336    NaN

```

139992	offer viewed	0009655768c64bdeb2e877511632db8f	372	NaN
--------	--------------	----------------------------------	-----	-----

	reward_x	offer_id_redux	difficulty	duration	\
55972	NaN	5a8bc65990b245e5a138643cd4eb9837	0.0	3.0	
77705	NaN	5a8bc65990b245e5a138643cd4eb9837	0.0	3.0	
89291	NaN	NaN	NaN	NaN	
113605	NaN	3f207df678b143eea3cee63160fa8bed	0.0	4.0	
139992	NaN	3f207df678b143eea3cee63160fa8bed	0.0	4.0	

	id	offer_type	reward_y	email	\
55972	5a8bc65990b245e5a138643cd4eb9837	informational	0.0	1.0	
77705	5a8bc65990b245e5a138643cd4eb9837	informational	0.0	1.0	
89291	NaN	NaN	NaN	NaN	
113605	3f207df678b143eea3cee63160fa8bed	informational	0.0	1.0	
139992	3f207df678b143eea3cee63160fa8bed	informational	0.0	1.0	

	mobile	social	web	offer_id_redux_right
55972	1.0	1.0	0.0	NaN
77705	1.0	1.0	0.0	5a8bc65990b245e5a138643cd4eb9837
89291	NaN	NaN	NaN	5a8bc65990b245e5a138643cd4eb9837
113605	1.0	0.0	1.0	NaN
139992	1.0	0.0	1.0	3f207df678b143eea3cee63160fa8bed

```
[65]: with_full_offer_id['offer_id'] = np.where(with_full_offer_id['offer_id_redux'].
        ↳ isnull(),
        ↳
        ↳ with_full_offer_id['offer_id_redux_right'],
        ↳ with_full_offer_id['offer_id_redux'])
```

```
[66]: with_full_offer_id.head()
```

```
[66]:
```

	event	person	time	amount	\
55972	offer received	0009655768c64bdeb2e877511632db8f	168	NaN	
77705	offer viewed	0009655768c64bdeb2e877511632db8f	192	NaN	
89291	transaction	0009655768c64bdeb2e877511632db8f	228	22.16	
113605	offer received	0009655768c64bdeb2e877511632db8f	336	NaN	
139992	offer viewed	0009655768c64bdeb2e877511632db8f	372	NaN	

	reward_x	offer_id_redux	difficulty	duration	\
55972	NaN	5a8bc65990b245e5a138643cd4eb9837	0.0	3.0	
77705	NaN	5a8bc65990b245e5a138643cd4eb9837	0.0	3.0	
89291	NaN	NaN	NaN	NaN	
113605	NaN	3f207df678b143eea3cee63160fa8bed	0.0	4.0	
139992	NaN	3f207df678b143eea3cee63160fa8bed	0.0	4.0	

	id	offer_type	reward_y	email	\
55972	5a8bc65990b245e5a138643cd4eb9837	informational	0.0	1.0	

77705	5a8bc65990b245e5a138643cd4eb9837	informational	0.0	1.0
89291	NaN	NaN	NaN	NaN
113605	3f207df678b143eea3cee63160fa8bed	informational	0.0	1.0
139992	3f207df678b143eea3cee63160fa8bed	informational	0.0	1.0

	mobile	social	web	offer_id_redux_right \
55972	1.0	1.0	0.0	NaN
77705	1.0	1.0	0.0	5a8bc65990b245e5a138643cd4eb9837
89291	NaN	NaN	NaN	5a8bc65990b245e5a138643cd4eb9837
113605	1.0	0.0	1.0	NaN
139992	1.0	0.0	1.0	3f207df678b143eea3cee63160fa8bed

	offer_id
55972	5a8bc65990b245e5a138643cd4eb9837
77705	5a8bc65990b245e5a138643cd4eb9837
89291	5a8bc65990b245e5a138643cd4eb9837
113605	3f207df678b143eea3cee63160fa8bed
139992	3f207df678b143eea3cee63160fa8bed

```
[67]: with_full_offer_id = with_full_offer_id.drop(['offer_id_redux',
                                                    'offer_id_redux_right'],
                                                    axis = 1)
```

```
[68]: with_full_offer_id.columns
```

```
[68]: Index(['event', 'person', 'time', 'amount', 'reward_x', 'difficulty',
            'duration', 'id', 'offer_type', 'reward_y', 'email', 'mobile', 'social',
            'web', 'offer_id'],
            dtype='object')
```

Now we can merge it again to get the transaction events data. This means we'll duplicate many columns, but all we have to do is drop them.

```
[69]: with_transaction = with_full_offer_id.merge(processed_portfolio,
                                                    how = 'left',
                                                    left_on = 'offer_id',
                                                    right_on = 'id')
```

```
[70]: with_transaction.head(10)
```

	event	person	time	amount	reward_x \
0	offer received	0009655768c64bdeb2e877511632db8f	168	NaN	NaN
1	offer viewed	0009655768c64bdeb2e877511632db8f	192	NaN	NaN
2	transaction	0009655768c64bdeb2e877511632db8f	228	22.16	NaN
3	offer received	0009655768c64bdeb2e877511632db8f	336	NaN	NaN
4	offer viewed	0009655768c64bdeb2e877511632db8f	372	NaN	NaN
5	offer received	0009655768c64bdeb2e877511632db8f	408	NaN	NaN

6	transaction	0009655768c64bdeb2e877511632db8f	414	8.57	NaN
7	offer completed	0009655768c64bdeb2e877511632db8f	414	NaN	5.0
8	offer viewed	0009655768c64bdeb2e877511632db8f	456	NaN	NaN
9	offer received	0009655768c64bdeb2e877511632db8f	504	NaN	NaN

	difficulty_x	duration_x	id_x	offer_type_x	\
0	0.0	3.0	5a8bc65990b245e5a138643cd4eb9837	informational	
1	0.0	3.0	5a8bc65990b245e5a138643cd4eb9837	informational	
2	NaN	NaN	NaN	NaN	
3	0.0	4.0	3f207df678b143eea3cee63160fa8bed	informational	
4	0.0	4.0	3f207df678b143eea3cee63160fa8bed	informational	
5	5.0	5.0	f19421c1d4aa40978ebb69ca19b0e20d	bogo	
6	NaN	NaN	NaN	NaN	
7	5.0	5.0	f19421c1d4aa40978ebb69ca19b0e20d	bogo	
8	5.0	5.0	f19421c1d4aa40978ebb69ca19b0e20d	bogo	
9	10.0	10.0	fafdc668e3743c1bb461111dcafc2a4	discount	

	reward_y	...	offer_id	difficulty_y	duration_y	\
0	0.0	...	5a8bc65990b245e5a138643cd4eb9837	0	3	
1	0.0	...	5a8bc65990b245e5a138643cd4eb9837	0	3	
2	NaN	...	5a8bc65990b245e5a138643cd4eb9837	0	3	
3	0.0	...	3f207df678b143eea3cee63160fa8bed	0	4	
4	0.0	...	3f207df678b143eea3cee63160fa8bed	0	4	
5	5.0	...	f19421c1d4aa40978ebb69ca19b0e20d	5	5	
6	NaN	...	3f207df678b143eea3cee63160fa8bed	0	4	
7	5.0	...	f19421c1d4aa40978ebb69ca19b0e20d	5	5	
8	5.0	...	f19421c1d4aa40978ebb69ca19b0e20d	5	5	
9	2.0	...	fafdc668e3743c1bb461111dcafc2a4	10	10	

	id_y	offer_type_y	reward	email_y	mobile_y	\
0	5a8bc65990b245e5a138643cd4eb9837	informational	0	1	1	
1	5a8bc65990b245e5a138643cd4eb9837	informational	0	1	1	
2	5a8bc65990b245e5a138643cd4eb9837	informational	0	1	1	
3	3f207df678b143eea3cee63160fa8bed	informational	0	1	1	
4	3f207df678b143eea3cee63160fa8bed	informational	0	1	1	
5	f19421c1d4aa40978ebb69ca19b0e20d	bogo	5	1	1	
6	3f207df678b143eea3cee63160fa8bed	informational	0	1	1	
7	f19421c1d4aa40978ebb69ca19b0e20d	bogo	5	1	1	
8	f19421c1d4aa40978ebb69ca19b0e20d	bogo	5	1	1	
9	fafdc668e3743c1bb461111dcafc2a4	discount	2	1	1	

	social_y	web_y
0	1	0
1	1	0
2	1	0
3	0	1
4	0	1

```

5      1      1
6      0      1
7      1      1
8      1      1
9      1      1

```

[10 rows x 24 columns]

```
[71]: with_transaction.columns
```

```
[71]: Index(['event', 'person', 'time', 'amount', 'reward_x', 'difficulty_x',
          'duration_x', 'id_x', 'offer_type_x', 'reward_y', 'email_x', 'mobile_x',
          'social_x', 'web_x', 'offer_id', 'difficulty_y', 'duration_y', 'id_y',
          'offer_type_y', 'reward', 'email_y', 'mobile_y', 'social_y', 'web_y'],
          dtype='object')
```

```
[72]: with_transaction[['reward',
                        'reward_x',
                        'reward_y',
                        'difficulty_x',
                        'difficulty_y',
                        'duration_x',
                        'duration_y']]
```

```
[72]:
```

	reward	reward_x	reward_y	difficulty_x	difficulty_y	duration_x	\
0	0	NaN	0.0	0.0	0	3.0	
1	0	NaN	0.0	0.0	0	3.0	
2	0	NaN	NaN	NaN	0	NaN	
3	0	NaN	0.0	0.0	0	4.0	
4	0	NaN	0.0	0.0	0	4.0	
5	5	NaN	5.0	5.0	5	5.0	
6	0	NaN	NaN	NaN	0	NaN	
7	5	5.0	5.0	5.0	5	5.0	
8	5	NaN	5.0	5.0	5	5.0	
9	2	NaN	2.0	10.0	10	10.0	
10	5	NaN	NaN	NaN	5	NaN	
11	2	2.0	2.0	10.0	10	10.0	
12	2	NaN	2.0	10.0	10	10.0	
13	2	NaN	NaN	NaN	10	NaN	
14	2	NaN	2.0	10.0	10	7.0	
15	2	NaN	NaN	NaN	10	NaN	
16	2	2.0	2.0	10.0	10	7.0	
17	2	NaN	NaN	NaN	10	NaN	
18	2	NaN	NaN	NaN	10	NaN	
19	2	NaN	NaN	NaN	10	NaN	
20	5	NaN	5.0	5.0	5	5.0	
21	5	NaN	5.0	5.0	5	5.0	

22	5	NaN	NaN	NaN	5	NaN
23	5	NaN	NaN	NaN	5	NaN
24	5	NaN	NaN	NaN	5	NaN
25	5	NaN	5.0	5.0	5	5.0
26	5	NaN	5.0	5.0	5	5.0
27	0	NaN	0.0	0.0	0	4.0
28	0	NaN	0.0	0.0	0	4.0
29	0	NaN	NaN	NaN	0	NaN
...
306504	2	2.0	2.0	10.0	10	10.0
306505	2	NaN	NaN	NaN	10	NaN
306506	2	NaN	NaN	NaN	10	NaN
306507	5	NaN	5.0	20.0	20	10.0
306508	5	NaN	5.0	20.0	20	10.0
306509	5	NaN	NaN	NaN	20	NaN
306510	5	5.0	5.0	20.0	20	10.0
306511	5	NaN	NaN	NaN	20	NaN
306512	5	NaN	NaN	NaN	20	NaN
306513	5	NaN	NaN	NaN	20	NaN
306514	5	NaN	NaN	NaN	20	NaN
306515	2	NaN	2.0	10.0	10	7.0
306516	2	NaN	2.0	10.0	10	7.0
306517	2	NaN	NaN	NaN	10	NaN
306518	2	2.0	2.0	10.0	10	7.0
306519	2	NaN	2.0	10.0	10	7.0
306520	2	NaN	2.0	10.0	10	7.0
306521	2	NaN	NaN	NaN	10	NaN
306522	2	2.0	2.0	10.0	10	7.0
306523	2	NaN	NaN	NaN	10	NaN
306524	5	NaN	5.0	5.0	5	7.0
306525	2	NaN	NaN	NaN	10	NaN
306526	5	5.0	5.0	5.0	5	7.0
306527	5	NaN	5.0	5.0	5	7.0
306528	2	NaN	2.0	10.0	10	7.0
306529	5	NaN	NaN	NaN	5	NaN
306530	2	2.0	2.0	10.0	10	7.0
306531	2	NaN	2.0	10.0	10	7.0
306532	2	NaN	NaN	NaN	10	NaN
306533	2	NaN	NaN	NaN	10	NaN

	duration_y
0	3
1	3
2	3
3	4
4	4
5	5

6	4
7	5
8	5
9	10
10	5
11	10
12	10
13	10
14	7
15	10
16	7
17	10
18	10
19	10
20	5
21	5
22	5
23	5
24	5
25	5
26	5
27	4
28	4
29	4
...	...
306504	10
306505	10
306506	10
306507	10
306508	10
306509	10
306510	10
306511	10
306512	10
306513	10
306514	10
306515	7
306516	7
306517	7
306518	7
306519	7
306520	7
306521	7
306522	7
306523	7
306524	7
306525	7

306526	7
306527	7
306528	7
306529	7
306530	7
306531	7
306532	7
306533	7

[306534 rows x 7 columns]

```
[73]: with_transaction[['reward',
                        'reward_x',
                        'reward_y',
                        'difficulty_x',
                        'difficulty_y',
                        'duration_x',
                        'duration_y']].isnull().sum()
```

```
[73]: reward          0
reward_x      272955
reward_y      138953
difficulty_x   138953
difficulty_y         0
duration_x     138953
duration_y         0
dtype: int64
```

We'll keep the `reward_y` column instead of the other ones because the lack of information there is related to the event type.

```
[74]: with_transaction.columns
```

```
[74]: Index(['event', 'person', 'time', 'amount', 'reward_x', 'difficulty_x',
          'duration_x', 'id_x', 'offer_type_x', 'reward_y', 'email_x', 'mobile_x',
          'social_x', 'web_x', 'offer_id', 'difficulty_y', 'duration_y', 'id_y',
          'offer_type_y', 'reward', 'email_y', 'mobile_y', 'social_y', 'web_y'],
          dtype='object')
```

```
[75]: with_transaction = with_transaction.drop(['reward',
                                                'reward_x',
                                                'difficulty_x',
                                                'duration_x',
                                                'id_x',
                                                'id_y',
                                                'offer_type_x',
                                                'email_x',
```



```

        'mobile_x',
        'social_x',
        'web_x'
    ],
    axis = 1)

with_transaction.columns

```

```

[75]: Index(['event', 'person', 'time', 'amount', 'reward_y', 'offer_id',
        'difficulty_y', 'duration_y', 'offer_type_y', 'email_y', 'mobile_y',
        'social_y', 'web_y'],
        dtype='object')

```

```

[76]: with_transaction = with_transaction.rename(columns = {'reward_y': 'reward',
        'difficulty_y':
        ↪ 'difficulty',
        'duration_y': 'duration',
        'offer_type_y':
        ↪ 'offer_type',
        'email_y': 'email',
        'mobile_y': 'mobile',
        'social_y': 'social',
        'web_y': 'web'
        })

```

```

[77]: with_transaction.head()

```

```

[77]:
      event      person  time  amount  reward \
0  offer received  0009655768c64bdeb2e877511632db8f  168    NaN    0.0
1    offer viewed  0009655768c64bdeb2e877511632db8f  192    NaN    0.0
2    transaction  0009655768c64bdeb2e877511632db8f  228   22.16    NaN
3  offer received  0009655768c64bdeb2e877511632db8f  336    NaN    0.0
4    offer viewed  0009655768c64bdeb2e877511632db8f  372    NaN    0.0

      offer_id  difficulty  duration  offer_type \
0  5a8bc65990b245e5a138643cd4eb9837      0      3  informational
1  5a8bc65990b245e5a138643cd4eb9837      0      3  informational
2  5a8bc65990b245e5a138643cd4eb9837      0      3  informational
3  3f207df678b143eea3cee63160fa8bed      0      4  informational
4  3f207df678b143eea3cee63160fa8bed      0      4  informational

      email  mobile  social  web
0         1         1         1    0
1         1         1         1    0
2         1         1         1    0
3         1         1          0    1
4         1         1          0    1

```

2.4 Finding successful offers

2.4.1 Defining success

To train the proposed models, we need to define success. For this problem, it makes sense that success is when a user has completed the funnels mentioned above successfully.

Let's start by finding which users got to which steps of the funnel. For that, we'll separate the events in its own tables and create auxiliary columns. Then we'll join those datasets to come up with a single dataset that has the offer id, the person id, and three booleans for received, viewed and completed. We'll also include the considerations for the order of the performed events to be considered valid in this step.

```
[78]: def creating_event_df(original_df, event):  
    '''  
    this function takes in a dataframe that contains the 'event',  
    'person' and 'time' columns and creates a new dataframe for only  
    one selected event sorted by people and time.  
  
    INPUT  
    - original_df (df): dataframe that contains the 'event',  
      'person' and 'time' columns  
    - event (str): the target event  
  
    OUTPUT  
    - target_df (df): sorted dataframe with the desired filters  
    '''  
    target_df = original_df[original_df['event'] == event].copy()  
    target_df.sort_values(['person', 'time'])  
  
    return target_df  
  
[79]: offer_received = creating_event_df(with_transaction, 'offer received')  
  
offer_received.rename(columns = {'time': 'time_received'},  
                      inplace = True)  
  
offer_received = offer_received.drop(['event',  
                                     'amount',  
                                     'difficulty',  
                                     'offer_type',  
                                     'email',  
                                     'mobile',  
                                     'social',  
                                     'web'],  
                                axis = 1)  
  
offer_received['received'] = np.ones(len(offer_received))
```

```
offer_received.head()
```

```
[79]:
```

	person	time_received	reward	\
0	0009655768c64bdeb2e877511632db8f	168	0.0	
3	0009655768c64bdeb2e877511632db8f	336	0.0	
5	0009655768c64bdeb2e877511632db8f	408	5.0	
9	0009655768c64bdeb2e877511632db8f	504	2.0	
14	0009655768c64bdeb2e877511632db8f	576	2.0	

	offer_id	duration	received
0	5a8bc65990b245e5a138643cd4eb9837	3	1.0
3	3f207df678b143eea3cee63160fa8bed	4	1.0
5	f19421c1d4aa40978ebb69ca19b0e20d	5	1.0
9	fafdc668e3743c1bb461111dcafc2a4	10	1.0
14	2906b810c7d4411798c6938adc9daaa5	7	1.0

```
[80]: offer_viewed = creating_event_df(with_transaction, 'offer viewed')

offer_viewed.rename(columns = {'time': 'time_viewed'},
                    inplace = True)

offer_viewed = offer_viewed.drop(['event',
                                   'amount',
                                   'reward',
                                   'duration',
                                   'difficulty',
                                   'offer_type',
                                   'email',
                                   'mobile',
                                   'social',
                                   'web'],
                                   axis = 1)

offer_viewed['viewed'] = np.ones(len(offer_viewed))

offer_viewed.head()
```

```
[80]:
```

	person	time_viewed	\
1	0009655768c64bdeb2e877511632db8f	192	
4	0009655768c64bdeb2e877511632db8f	372	
8	0009655768c64bdeb2e877511632db8f	456	
12	0009655768c64bdeb2e877511632db8f	540	
21	00116118485d4dfda04fdbaba9a87b5c	216	

	offer_id	viewed
1	5a8bc65990b245e5a138643cd4eb9837	1.0
4	3f207df678b143eea3cee63160fa8bed	1.0

8	f19421c1d4aa40978ebb69ca19b0e20d	1.0
12	fafdc668e3743c1bb461111dcafc2a4	1.0
21	f19421c1d4aa40978ebb69ca19b0e20d	1.0

```
[81]: offer_transaction = creating_event_df(with_transaction, 'transaction')

offer_transaction.rename(columns = {'time': 'time_transaction'},
                           inplace = True)

offer_transaction = offer_transaction.drop(['event',
                                             'reward',
                                             'duration',
                                             'difficulty',
                                             'offer_type',
                                             'email',
                                             'mobile',
                                             'social',
                                             'web'],
                                             axis = 1)

offer_transaction['has_transaction'] = np.ones(len(offer_transaction))

offer_transaction.head()
```

```
[81]:
```

	person	time_transaction	amount	\
2	0009655768c64bdeb2e877511632db8f	228	22.16	
6	0009655768c64bdeb2e877511632db8f	414	8.57	
10	0009655768c64bdeb2e877511632db8f	528	14.11	
13	0009655768c64bdeb2e877511632db8f	552	13.56	
15	0009655768c64bdeb2e877511632db8f	576	10.27	

	offer_id	has_transaction
2	5a8bc65990b245e5a138643cd4eb9837	1.0
6	3f207df678b143eea3cee63160fa8bed	1.0
10	f19421c1d4aa40978ebb69ca19b0e20d	1.0
13	fafdc668e3743c1bb461111dcafc2a4	1.0
15	fafdc668e3743c1bb461111dcafc2a4	1.0

```
[82]: offer_completed = creating_event_df(with_transaction, 'offer completed')

offer_completed.rename(columns = {'time': 'time_completed'},
                        inplace = True)

offer_completed = offer_completed.drop(['event',
                                         'amount',
                                         'reward',
                                         'duration'],
```

```

        'difficulty',
        'offer_type',
        'email',
        'mobile',
        'social',
        'web'],
        axis = 1)

offer_completed['completed'] = np.ones(len(offer_completed))

offer_completed.head()

```

```

[82]:
           person  time_completed  \
7    0009655768c64bdeb2e877511632db8f    414
11   0009655768c64bdeb2e877511632db8f    528
16   0009655768c64bdeb2e877511632db8f    576
33   0011e0d4e6b944f998e987f904e8c1e5    252
41   0011e0d4e6b944f998e987f904e8c1e5    576

           offer_id  completed
7    f19421c1d4aa40978ebb69ca19b0e20d    1.0
11   fafdcd668e3743c1bb461111dcafc2a4    1.0
16   2906b810c7d4411798c6938adc9daaa5    1.0
33   2298d6c36e964ae4a3e7e9706d1fb8c2    1.0
41   0b1e1539f2cc45b7b9fa7c272da2e1d7    1.0

```

```

[83]: # merge event dataframes into one
offers_received_viewed = offer_received.merge(offer_viewed,
                                              how = 'left',
                                              on = ['person', 'offer_id'])

offers_received_viewed_transaction = offers_received_viewed.
    ↳ merge(offer_transaction,
                                              how = 'left',
                                              on =
    ↳ ['person', 'offer_id'])

offers = offers_received_viewed_transaction.merge(offer_completed,
                                              how = 'left',
                                              on = ['person', 'offer_id'])

# reorder columns
offers = offers[['person',
                'offer_id',
                'received',
                'viewed',
                'has_transaction',

```

```

        'completed',
        'duration',
        'time_received',
        'time_viewed',
        'time_transaction',
        'time_completed',
        'amount',
        'reward']]

# add expiration time for the offer
offers['time_expiry'] = offers['duration'] + offers['time_received']

```

[84]: offers.head()

```

[84]:
      person                                     offer_id \
0  0009655768c64bdeb2e877511632db8f  5a8bc65990b245e5a138643cd4eb9837
1  0009655768c64bdeb2e877511632db8f  3f207df678b143eea3cee63160fa8bed
2  0009655768c64bdeb2e877511632db8f  f19421c1d4aa40978ebb69ca19b0e20d
3  0009655768c64bdeb2e877511632db8f  fafdcd668e3743c1bb461111dcafc2a4
4  0009655768c64bdeb2e877511632db8f  fafdcd668e3743c1bb461111dcafc2a4

      received  viewed  has_transaction  completed  duration  time_received \
0          1.0    1.0             1.0         NaN         3            168
1          1.0    1.0             1.0         NaN         4            336
2          1.0    1.0             1.0         1.0         5            408
3          1.0    1.0             1.0         1.0        10            504
4          1.0    1.0             1.0         1.0        10            504

      time_viewed  time_transaction  time_completed  amount  reward  time_expiry
0          192.0           228.0           NaN    22.16     0.0           171
1          372.0           414.0           NaN     8.57     0.0           340
2          456.0           528.0         414.0    14.11     5.0           413
3          540.0           552.0         528.0    13.56     2.0           514
4          540.0           576.0         528.0    10.27     2.0           514

```

```

[85]: # filter only pairs of events and person where the
      # funnel order was followed correctly and the offer
      # was successful
successful_offers = offers[(offers['has_transaction'] == 1) &
    →      # the offer must have had a transaction
    (offers['time_transaction'] >=
    →offers['time_viewed']) &      # the moment the transaction happened must be
    →after the offer was viewed
    (offers['time_viewed'] >= offers['time_received']) &
    →      # the moment the offer was viewed must be after the offer was received

```

```

                                ((offers['time_expiry'] < offers['time_completed']))
→|      # the offers must have been completed - either by transaction in the
→case
                                (offers['time_expiry'] <
→offers['time_transaction']))]      ## informational offers or by event -
→before the expiration time

# add a column of ones signaling those are successful offers
successful_offers['is_successful'] = np.ones(len(successful_offers))

```

/home/julia.tessler/anaconda3/lib/python3.7/site-packages/ipykernel_launcher.py:11: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: <http://pandas.pydata.org/pandas-docs/stable/indexing.html#indexing-view-versus-copy>
This is added back by InteractiveShellApp.init_path()

```
[86]: successful_offers.shape
```

```
[86]: (217659, 15)
```

```

[87]: # filter only pairs of events and person where the
      # funnel order was followed correctly but the offer
      # was not successful
      non_successful_offers = offers[offers['has_transaction'].isna()]      # if the
      →offer has no transaction, it wasn't successful

      # add a column of ones signaling those are successful offers
      non_successful_offers['is_successful'] = np.zeros(len(non_successful_offers))

```

/home/julia.tessler/anaconda3/lib/python3.7/site-packages/ipykernel_launcher.py:7: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: <http://pandas.pydata.org/pandas-docs/stable/indexing.html#indexing-view-versus-copy>
import sys

```
[88]: non_successful_offers.shape
```

```
[88]: (27573, 15)
```

Now we can merge it back into one dataset and then merge this data with the rest of the profile data.

```
[89]: offers = successful_offers.merge(non_successful_offers,
                                     how = 'outer')
```

```
[90]: offers.shape
```

```
[90]: (245232, 15)
```

We also need to refine user data. Let's start by joining the `profile` data with the transactions.

```
[91]: len(offers['person'].unique())
```

```
[91]: 16993
```

```
[92]: len(processed_profile['id'].unique())
```

```
[92]: 17000
```

```
[93]: len(with_transaction['person'].unique())
```

```
[93]: 17000
```

The amount of distinct users in our `offers` data isn't the same as it is in the `profile` data, which means there is a chance we have most of our offers covered by the `profile` data.

The `offers` dataset has 245,232 rows.

```
[94]: selected_users_profile = offers['person'].isin(processed_profile['id'])
      selected_users_offers = offers[selected_users_profile]
```

```
[95]: selected_users = with_transaction['person'].
      ↪isin(selected_users_offers['person'])
      selected_transactions = with_transaction[selected_users]
```

```
[96]: selected_transactions = selected_transactions.drop(['event',
                                                         'time'], axis = 1)
      selected_transactions.head()
```

```
[96]:
```

	person	amount	reward	\
0	0009655768c64bdeb2e877511632db8f	NaN	0.0	
1	0009655768c64bdeb2e877511632db8f	NaN	0.0	
2	0009655768c64bdeb2e877511632db8f	22.16	NaN	
3	0009655768c64bdeb2e877511632db8f	NaN	0.0	
4	0009655768c64bdeb2e877511632db8f	NaN	0.0	

	offer_id	difficulty	duration	offer_type	\
0	5a8bc65990b245e5a138643cd4eb9837	0	3	informational	
1	5a8bc65990b245e5a138643cd4eb9837	0	3	informational	
2	5a8bc65990b245e5a138643cd4eb9837	0	3	informational	

3	3f207df678b143eea3cee63160fa8bed	0	4	informational
4	3f207df678b143eea3cee63160fa8bed	0	4	informational

	email	mobile	social	web
0	1	1	1	0
1	1	1	1	0
2	1	1	1	0
3	1	1	0	1
4	1	1	0	1

```
[97]: selected_transactions.shape
```

```
[97]: (306508, 11)
```

Now we have to filter this dataset only for the successful or non-successful offers we found above.

```
[98]: successful_transactions = offers.merge(selected_transactions,
                                             how = 'inner',
                                             right_on = ['person', 'offer_id'],
                                             left_on = ['person', 'offer_id'])
```

```
[99]: successful_transactions.columns
```

```
[99]: Index(['person', 'offer_id', 'received', 'viewed', 'has_transaction',
           'completed', 'duration_x', 'time_received', 'time_viewed',
           'time_transaction', 'time_completed', 'amount_x', 'reward_x',
           'time_expiry', 'is_successful', 'amount_y', 'reward_y', 'difficulty',
           'duration_y', 'offer_type', 'email', 'mobile', 'social', 'web'],
          dtype='object')
```

```
[100]: successful_transactions = successful_transactions.drop(['reward_x',
                                                             'received',
                                                             'viewed',
                                                             'has_transaction',
                                                             'completed',
                                                             'duration_x',
                                                             'time_received',
                                                             'time_viewed',
                                                             'time_transaction',
                                                             'time_completed',
                                                             'amount_x',
                                                             'time_expiry',
                                                             ],
                                                             axis = 1)

successful_transactions.rename(columns = {'reward_y': 'reward',
                                         'duration_y': 'duration',
```

```

                                'amount_y': 'amount'}},
    inplace = True)

successful_transactions.head(20)

```

```

[100]:
      person                                     offer_id \
0  0009655768c64bdeb2e877511632db8f  5a8bc65990b245e5a138643cd4eb9837
1  0009655768c64bdeb2e877511632db8f  5a8bc65990b245e5a138643cd4eb9837
2  0009655768c64bdeb2e877511632db8f  5a8bc65990b245e5a138643cd4eb9837
3  0009655768c64bdeb2e877511632db8f  3f207df678b143eea3cee63160fa8bed
4  0009655768c64bdeb2e877511632db8f  3f207df678b143eea3cee63160fa8bed
5  0009655768c64bdeb2e877511632db8f  3f207df678b143eea3cee63160fa8bed
6  0009655768c64bdeb2e877511632db8f  f19421c1d4aa40978ebb69ca19b0e20d
7  0009655768c64bdeb2e877511632db8f  f19421c1d4aa40978ebb69ca19b0e20d
8  0009655768c64bdeb2e877511632db8f  f19421c1d4aa40978ebb69ca19b0e20d
9  0009655768c64bdeb2e877511632db8f  f19421c1d4aa40978ebb69ca19b0e20d
10 0009655768c64bdeb2e877511632db8f  fafdcd668e3743c1bb461111dcafc2a4
11 0009655768c64bdeb2e877511632db8f  fafdcd668e3743c1bb461111dcafc2a4
12 0009655768c64bdeb2e877511632db8f  fafdcd668e3743c1bb461111dcafc2a4
13 0009655768c64bdeb2e877511632db8f  fafdcd668e3743c1bb461111dcafc2a4
14 0009655768c64bdeb2e877511632db8f  fafdcd668e3743c1bb461111dcafc2a4
15 0009655768c64bdeb2e877511632db8f  fafdcd668e3743c1bb461111dcafc2a4
16 0009655768c64bdeb2e877511632db8f  fafdcd668e3743c1bb461111dcafc2a4
17 0009655768c64bdeb2e877511632db8f  fafdcd668e3743c1bb461111dcafc2a4
18 0009655768c64bdeb2e877511632db8f  fafdcd668e3743c1bb461111dcafc2a4
19 0009655768c64bdeb2e877511632db8f  fafdcd668e3743c1bb461111dcafc2a4

      is_successful  amount  reward  difficulty  duration  offer_type  email \
0                1.0    NaN    0.0           0         3  informational    1
1                1.0    NaN    0.0           0         3  informational    1
2                1.0  22.16    NaN           0         3  informational    1
3                1.0    NaN    0.0           0         4  informational    1
4                1.0    NaN    0.0           0         4  informational    1
5                1.0   8.57    NaN           0         4  informational    1
6                1.0    NaN    5.0           5         5          bogo    1
7                1.0    NaN    5.0           5         5          bogo    1
8                1.0    NaN    5.0           5         5          bogo    1
9                1.0  14.11    NaN           5         5          bogo    1
10               1.0    NaN    2.0          10        10    discount    1
11               1.0    NaN    2.0          10        10    discount    1
12               1.0    NaN    2.0          10        10    discount    1
13               1.0  13.56    NaN          10        10    discount    1
14               1.0  10.27    NaN          10        10    discount    1
15               1.0  12.36    NaN          10        10    discount    1
16               1.0  28.16    NaN          10        10    discount    1
17               1.0  18.41    NaN          10        10    discount    1
18               1.0    NaN    2.0          10        10    discount    1

```

19	1.0	NaN	2.0	10	10	discount	1
----	-----	-----	-----	----	----	----------	---

	mobile	social	web
0	1	1	0
1	1	1	0
2	1	1	0
3	1	0	1
4	1	0	1
5	1	0	1
6	1	1	1
7	1	1	1
8	1	1	1
9	1	1	1
10	1	1	1
11	1	1	1
12	1	1	1
13	1	1	1
14	1	1	1
15	1	1	1
16	1	1	1
17	1	1	1
18	1	1	1
19	1	1	1

```
[101]: successful_transactions.shape
```

```
[101]: (2472021, 12)
```

Now we have multiple lines for the same successful transaction. Let's leave only one line of informations for each person and offer pair.

```
[102]: successful_transactions = successful_transactions.drop_duplicates()
successful_transactions.head()
```

```
[102]:
```

	person	offer_id	\
0	0009655768c64bdeb2e877511632db8f	5a8bc65990b245e5a138643cd4eb9837	
2	0009655768c64bdeb2e877511632db8f	5a8bc65990b245e5a138643cd4eb9837	
3	0009655768c64bdeb2e877511632db8f	3f207df678b143eea3cee63160fa8bed	
5	0009655768c64bdeb2e877511632db8f	3f207df678b143eea3cee63160fa8bed	
6	0009655768c64bdeb2e877511632db8f	f19421c1d4aa40978ebb69ca19b0e20d	

	is_successful	amount	reward	difficulty	duration	offer_type	email	\
0	1.0	NaN	0.0	0	3	informational	1	
2	1.0	22.16	NaN	0	3	informational	1	
3	1.0	NaN	0.0	0	4	informational	1	
5	1.0	8.57	NaN	0	4	informational	1	
6	1.0	NaN	5.0	5	5	bogo	1	

	mobile	social	web
0	1	1	0
2	1	1	0
3	1	0	1
5	1	0	1
6	1	1	1

We still have some duplicates due to the multiple lines related to events... Since the `amount` column only happens when a user makes a transaction, this feature is highly important to predict the success because it's part of the definition of success. Therefore, it can't be used in the model and should be dropped.

```
[103]: transactions_logs = successful_transactions.drop(['amount'], axis = 1)

transactions_logs = transactions_logs.dropna().drop_duplicates()
```

```
[104]: transactions_logs.shape
```

```
[104]: (62398, 11)
```

```
[105]: transactions_logs.head(20)
```

```
[105]:
```

	person	offer_id \
0	0009655768c64bdeb2e877511632db8f	5a8bc65990b245e5a138643cd4eb9837
3	0009655768c64bdeb2e877511632db8f	3f207df678b143eea3cee63160fa8bed
6	0009655768c64bdeb2e877511632db8f	f19421c1d4aa40978ebb69ca19b0e20d
10	0009655768c64bdeb2e877511632db8f	fafdc668e3743c1bb461111dcafc2a4
50	00116118485d4dfda04fdbaba9a87b5c	f19421c1d4aa40978ebb69ca19b0e20d
71	0011e0d4e6b944f998e987f904e8c1e5	3f207df678b143eea3cee63160fa8bed
74	0011e0d4e6b944f998e987f904e8c1e5	2298d6c36e964ae4a3e7e9706d1fb8c2
78	0011e0d4e6b944f998e987f904e8c1e5	9b98b8c7a33c4b65b9aebfe6a799e6d9
96	0020c2b971eb4e9188eac86d93036a77	fafdc668e3743c1bb461111dcafc2a4
168	0020c2b971eb4e9188eac86d93036a77	4d5c57ea9a6940dd891ad53e9d8da0
178	0020c2b971eb4e9188eac86d93036a77	5a8bc65990b245e5a138643cd4eb9837
186	0020ccbbb6d84e358d3414a3ff76cffd	2298d6c36e964ae4a3e7e9706d1fb8c2
214	0020ccbbb6d84e358d3414a3ff76cffd	f19421c1d4aa40978ebb69ca19b0e20d
219	0020ccbbb6d84e358d3414a3ff76cffd	5a8bc65990b245e5a138643cd4eb9837
246	0020ccbbb6d84e358d3414a3ff76cffd	9b98b8c7a33c4b65b9aebfe6a799e6d9
256	003d66b6608740288d6cc97a6903f4f0	5a8bc65990b245e5a138643cd4eb9837
271	003d66b6608740288d6cc97a6903f4f0	fafdc668e3743c1bb461111dcafc2a4
1525	003d66b6608740288d6cc97a6903f4f0	3f207df678b143eea3cee63160fa8bed
1533	00426fe3ffde4c6b9cb9ad6d077a13ea	5a8bc65990b245e5a138643cd4eb9837
1596	00426fe3ffde4c6b9cb9ad6d077a13ea	fafdc668e3743c1bb461111dcafc2a4

	is_successful	reward	difficulty	duration	offer_type	email \
0	1.0	0.0	0	3	informational	1

3	1.0	0.0	0	4	informational	1
6	1.0	5.0	5	5	bogo	1
10	1.0	2.0	10	10	discount	1
50	1.0	5.0	5	5	bogo	1
71	1.0	0.0	0	4	informational	1
74	1.0	3.0	7	7	discount	1
78	1.0	5.0	5	7	bogo	1
96	1.0	2.0	10	10	discount	1
168	1.0	10.0	10	5	bogo	1
178	1.0	0.0	0	3	informational	1
186	1.0	3.0	7	7	discount	1
214	1.0	5.0	5	5	bogo	1
219	1.0	0.0	0	3	informational	1
246	1.0	5.0	5	7	bogo	1
256	1.0	0.0	0	3	informational	1
271	1.0	2.0	10	10	discount	1
1525	1.0	0.0	0	4	informational	1
1533	1.0	0.0	0	3	informational	1
1596	1.0	2.0	10	10	discount	1

	mobile	social	web
0	1	1	0
3	1	0	1
6	1	1	1
10	1	1	1
50	1	1	1
71	1	0	1
74	1	1	1
78	1	0	1
96	1	1	1
168	1	1	1
178	1	1	0
186	1	1	1
214	1	1	1
219	1	1	0
246	1	0	1
256	1	1	0
271	1	1	1
1525	1	0	1
1533	1	1	0
1596	1	1	1

Now we can join the user data with the transaction logs.

```
[106]: transactions_with_users = transactions_logs.merge(processed_profile,
                                                         how = 'left',
                                                         left_on = 'person',
```

```
right_on = 'id')
```

```
[107]: transactions_with_users.columns
```

```
[107]: Index(['person', 'offer_id', 'is_successful', 'reward', 'difficulty',  
        'duration', 'offer_type', 'email', 'mobile', 'social', 'web', 'age',  
        'gender', 'id', 'income', 'became_member_on_year'],  
        dtype='object')
```

Cool! Now we're almost ready to start modelling. To model data, we need to remove the `person/id` and `offer_id` columns as well as encode the `offer_type` and `gender` (we'll use [Scikit-learn's LabelEncoder\(\)](#) for this). The amount coly

```
[108]: final_dataset = transactions_with_users.drop(['person',  
        'offer_id',  
        'id'],  
        axis = 1)
```

```
[109]: le = LabelEncoder()
```

```
[110]: offer_types_list = list(final_dataset['offer_type'].unique())  
types_encoder = le.fit(offer_types_list)  
offer_type_encoded = le.transform(final_dataset['offer_type'])  
  
final_dataset['offer_type_encoded'] = offer_type_encoded
```

```
[111]: gender_list = list(final_dataset['gender'].unique())  
gender_encoder = le.fit(gender_list)  
gender_encoded = le.transform(final_dataset['gender'])  
  
final_dataset['gender_encoded'] = gender_encoded
```

```
[112]: final_dataset.head(20)
```

```
[112]:
```

	is_successful	reward	difficulty	duration	offer_type	email	mobile	\
0	1.0	0.0	0	3	informational	1	1	
1	1.0	0.0	0	4	informational	1	1	
2	1.0	5.0	5	5	bogo	1	1	
3	1.0	2.0	10	10	discount	1	1	
4	1.0	5.0	5	5	bogo	1	1	
5	1.0	0.0	0	4	informational	1	1	
6	1.0	3.0	7	7	discount	1	1	
7	1.0	5.0	5	7	bogo	1	1	
8	1.0	2.0	10	10	discount	1	1	
9	1.0	10.0	10	5	bogo	1	1	
10	1.0	0.0	0	3	informational	1	1	
11	1.0	3.0	7	7	discount	1	1	

12	1.0	5.0	5	5	bogo	1	1
13	1.0	0.0	0	3	informational	1	1
14	1.0	5.0	5	7	bogo	1	1
15	1.0	0.0	0	3	informational	1	1
16	1.0	2.0	10	10	discount	1	1
17	1.0	0.0	0	4	informational	1	1
18	1.0	0.0	0	3	informational	1	1
19	1.0	2.0	10	10	discount	1	1

	social	web	age	gender	income	became_member_on_year	\
0	1	0	33	M	72000.000000	2017	
1	0	1	33	M	72000.000000	2017	
2	1	1	33	M	72000.000000	2017	
3	1	1	33	M	72000.000000	2017	
4	1	1	118	Unknown	65404.991568	2018	
5	0	1	40	O	57000.000000	2018	
6	1	1	40	O	57000.000000	2018	
7	0	1	40	O	57000.000000	2018	
8	1	1	59	F	90000.000000	2016	
9	1	1	59	F	90000.000000	2016	
10	1	0	59	F	90000.000000	2016	
11	1	1	24	F	60000.000000	2016	
12	1	1	24	F	60000.000000	2016	
13	1	0	24	F	60000.000000	2016	
14	0	1	24	F	60000.000000	2016	
15	1	0	26	F	73000.000000	2017	
16	1	1	26	F	73000.000000	2017	
17	0	1	26	F	73000.000000	2017	
18	1	0	19	F	65000.000000	2016	
19	1	1	19	F	65000.000000	2016	

	offer_type_encoded	gender_encoded
0	2	1
1	2	1
2	0	1
3	1	1
4	0	3
5	2	2
6	1	2
7	0	2
8	1	0
9	0	0
10	2	0
11	1	0
12	0	0
13	2	0
14	0	0

15	2	0
16	1	0
17	2	0
18	2	0
19	1	0

```
[113]: final_dataset = final_dataset.drop(['offer_type',
                                         'gender'], axis = 1)
```

```
[114]: final_dataset.columns
```

```
[114]: Index(['is_successful', 'reward', 'difficulty', 'duration', 'email', 'mobile',
            'social', 'web', 'age', 'income', 'became_member_on_year',
            'offer_type_encoded', 'gender_encoded'],
            dtype='object')
```

```
[115]: final_dataset.isna().sum()
```

```
[115]: is_successful      0
reward                  0
difficulty              0
duration               0
email                  0
mobile                 0
social                 0
web                   0
age                   0
income                0
became_member_on_year  0
offer_type_encoded     0
gender_encoded         0
dtype: int64
```

```
[116]: final_dataset.shape
```

```
[116]: (62398, 13)
```

```
[141]: final_dataset.head()
```

```
[141]:   is_successful  reward  difficulty  duration  email  mobile  social  web  \
0             1.0    0.0           0         3      1      1      1    0
1             1.0    0.0           0         4      1      1      0    1
2             1.0    5.0           5         5      1      1      1    1
3             1.0    2.0          10        10      1      1      1    1
4             1.0    5.0           5         5      1      1      1    1

   age      income  became_member_on_year  offer_type_encoded  \
```


0	33	72000.000000	2017	2
1	33	72000.000000	2017	2
2	33	72000.000000	2017	0
3	33	72000.000000	2017	1
4	118	65404.991568	2018	0

	gender_encoded
0	1
1	1
2	1
3	1
4	3

2.5 Modeling

2.5.1 Separating data into train and test

We'll use [Scikit-learn's train_test_split function](#) to separate data, with 80% as train and 20% as test.

Our independent variables - or features - will be: * reward * difficulty * duration * email * mobile * social * web * amount * age * income * became_member_on_year * offer_type_encoded * gender_encoded

We'll train supervised models for this. Our dependent variable will be `is_successful`.

The supervised models that will be trained are: * Model 1: Logistic Regression * Model 2: Naïve Bayes * Model 3: Support Vector Machines (SVM) * Model 4: Decision Tree

We'll use Scikit-learn's framework for all models.

```
[117]: features = final_dataset[['reward',
                                'difficulty',
                                'duration',
                                'email',
                                'mobile',
                                'social',
                                'web',
                                'age',
                                'income',
                                'became_member_on_year',
                                'offer_type_encoded',
                                'gender_encoded']]

y = final_dataset['is_successful']
```

```
[118]: X_train, X_test, y_train, y_test = train_test_split(features,
                                                            y,
                                                            test_size = 0.8,
                                                            random_state = 42)
```

To train all models, we'll use [Scikit-learn's cross_val_score](#) with 5 folds.

2.5.2 Baseline model

As baseline model we'll use a naive one, given by [Scikit-learn's DummyClassifier](#). The model shall predict by most frequent class.

```
[119]: dummy_clf = DummyClassifier(strategy = "most_frequent")

cross_val_baseline = cross_val_score(dummy_clf,
                                     X_train,
                                     y_train,
                                     cv = 5,
                                     scoring = 'f1_micro')
```

```
[120]: cross_val_baseline.mean()
```

```
[120]: 0.6441221316975467
```

The baseline model has 64.41% of F1-Score. We're looking for models that can perform better than this one.

2.5.3 Model 1: Logistic Regression

Since this is a small dataset, we'll use the `liblinear` solver as suggested by [the documentation](#).

The logistic regression is a fairly simple model that presents good results in robust datasets, such as this one. We expect it to be better than the baseline and, since this is one of the most explainable models there is, we might favor it depending on the results.

```
[121]: lr = LogisticRegression(solver = 'liblinear', random_state = 42)

cross_val_lr = cross_val_score(lr,
                                X_train,
                                y_train,
                                cv = 5,
                                scoring = 'f1_micro')
```

```
[122]: cross_val_lr.mean()
```

```
[122]: 0.6662358493158751
```

We can see some marginal gain in the F1-Score: this model reached 66.62%.

2.5.4 Model 2: Naïve Bayes

We'll train a Gaussian Naïve Bayes model. This model usually performs well in well defined classification models. The chosen classifier, Gaussian, assumes that the likelihood of the features comes from a Gaussian distribution (see [documentation](#) for more details).

Even though this is a naïve model, it can perform very well for this type of problem, even when the data breaks the Gaussian assumption.

```
[123]: nb = GaussianNB()

cross_val_nb = cross_val_score(nb,
                                X_train,
                                y_train,
                                cv = 5,
                                scoring = 'f1_micro')
```

```
[124]: cross_val_nb.mean()
```

```
[124]: 0.702621055986057
```

This model performed a bit better than the Logistic Regression and baseline models when it comes to F1-Score: 70.26%.

2.5.5 Model 3: Support Vector Machines (SVM)

We'll train support vector machines. SVMs are considered more complex than the models above. We expect it to perform better than the baseline, but it might start to overfit the model.

```
[125]: sv = svm.SVC()

cross_val_sv = cross_val_score(sv,
                                X_train,
                                y_train,
                                cv = 5,
                                scoring = 'f1_micro')
```

```
/home/julia.tessler/anaconda3/lib/python3.7/site-
packages/sklearn/svm/base.py:193: FutureWarning: The default value of gamma will
change from 'auto' to 'scale' in version 0.22 to account better for unscaled
features. Set gamma explicitly to 'auto' or 'scale' to avoid this warning.
```

```
"avoid this warning.", FutureWarning)
```

```
/home/julia.tessler/anaconda3/lib/python3.7/site-
packages/sklearn/svm/base.py:193: FutureWarning: The default value of gamma will
change from 'auto' to 'scale' in version 0.22 to account better for unscaled
features. Set gamma explicitly to 'auto' or 'scale' to avoid this warning.
```

```
"avoid this warning.", FutureWarning)
```

```
/home/julia.tessler/anaconda3/lib/python3.7/site-
packages/sklearn/svm/base.py:193: FutureWarning: The default value of gamma will
change from 'auto' to 'scale' in version 0.22 to account better for unscaled
features. Set gamma explicitly to 'auto' or 'scale' to avoid this warning.
```

```
"avoid this warning.", FutureWarning)
```

```
/home/julia.tessler/anaconda3/lib/python3.7/site-
packages/sklearn/svm/base.py:193: FutureWarning: The default value of gamma will
change from 'auto' to 'scale' in version 0.22 to account better for unscaled
```

```
features. Set gamma explicitly to 'auto' or 'scale' to avoid this warning.
    "avoid this warning.", FutureWarning)
/home/julia.tessler/anaconda3/lib/python3.7/site-
packages/sklearn/svm/base.py:193: FutureWarning: The default value of gamma will
change from 'auto' to 'scale' in version 0.22 to account better for unscaled
features. Set gamma explicitly to 'auto' or 'scale' to avoid this warning.
    "avoid this warning.", FutureWarning)
```

```
[126]: cross_val_sv.mean()
```

```
[126]: 0.6510127408915569
```

This model has 65.10% of F1-Score. It is better than the baseline model, but not better than the Naïve Bayes model.

2.5.6 Model 4: Decision Tree

We'll train a decision tree classifier. This is the most powerful of all the chosen models and is very likely to overfit.

```
[127]: dt = DecisionTreeClassifier(random_state = 42)

cross_val_dt = cross_val_score(dt,
                                X_train,
                                y_train,
                                cv = 5,
                                scoring = 'f1_micro')
```

```
[128]: cross_val_dt.mean()
```

```
[128]: 0.6508533831240313
```

This better than the baseline model, since the F1-Score is 65.08%. But it's not the best.

2.6 Choosing the best model

Based on the results we found above, the best model is the Naïve Bayes. Let's retrain it so we can keep it's coefficients and apply it to the test features.

```
[129]: model = nb.fit(X_train, y_train)
```

```
[130]: predictions = model.predict(X_test)
```

```
[131]: # get the confusion matrix
pd.crosstab(y_test, predictions)
```

```
[131]: col_0      0.0    1.0
is_successful
0.0         6095   12012
```

1.0 3097 28715

```
[132]: metrics.f1_score(y_test, predictions, average = 'micro')
```

```
[132]: 0.6973296740719966
```

```
[133]: print(metrics.classification_report(y_test, predictions))
```

	precision	recall	f1-score	support
0.0	0.66	0.34	0.45	18107
1.0	0.71	0.90	0.79	31812
accuracy			0.70	49919
macro avg	0.68	0.62	0.62	49919
weighted avg	0.69	0.70	0.67	49919

The model performed not very well in the test dataset, reaching a 69.73% F1-Score. The decrease in the score was already expected, since this data is new to the model.

The other usual metrics performed fairly well, with exception of the recall for non-successful offers, which was 34% - meaning that it's prediction is worse than flipping a coin to predict the success of the offer.

Unfortunately, Naïve Bayes model do not have easy explanations for the coefficients, which means that we can't easily explain what variables are related to the success of the offer. What we can do is look into each variable and the associated probability to look into the relations.

```
[134]: result_df = pd.DataFrame(X_train.columns)
result_df[['non successful', 'successful']] = pd.DataFrame(model.
    ↳predict_proba(X_train).tolist())

result_df
```

```
[134]:
```

	0	non successful	successful
0	reward	0.204731	0.795269
1	difficulty	0.925091	0.074909
2	duration	0.346706	0.653294
3	email	0.294267	0.705733
4	mobile	0.408526	0.591474
5	social	0.093768	0.906232
6	web	0.924745	0.075255
7	age	0.337013	0.662987
8	income	0.398634	0.601366
9	became_member_on_year	0.288445	0.711555
10	offer_type_encoded	0.118418	0.881582
11	gender_encoded	0.086060	0.913940

Also we can get the features that had the most predictive contribution to each class, but still no explainability.

```
[135]: # source: https://stackoverflow.com/questions/50526898/
      ↪how-to-get-feature-importance-in-naive-bayes
      # prints the top 10 most predictive features for the non-successful class
      neg = model.theta_[0].argsort()
      print(np.take(X_train.columns, neg[:10]))

      print('')

      # prints the top 10 most predictive features for the successful class
      neg = model.sigma_[0].argsort()
      print(np.take(X_train.columns, neg[:10]))
```

```
Index(['social', 'mobile', 'web', 'gender_encoded', 'offer_type_encoded',
      'email', 'reward', 'duration', 'difficulty', 'age'],
      dtype='object')
```

```
Index(['email', 'web', 'mobile', 'social', 'offer_type_encoded',
      'gender_encoded', 'became_member_on_year', 'duration', 'reward',
      'difficulty'],
      dtype='object')
```

If we consider the Logist Regression as the best explainable model, we can look into the coefficients to get the most important features.

```
[136]: model_lr = lr.fit(X_train, y_train)
```

```
[137]: predictions_lr = model_lr.predict(X_test)
```

```
[138]: metrics.f1_score(y_test, predictions_lr, average = 'micro')
```

```
[138]: 0.6438029607964904
```

```
[139]: # get the confusion matrix
      pd.crosstab(y_test, predictions_lr)
```

```
[139]: col_0      0.0    1.0
      is_successful
      0.0         523  17584
      1.0         197  31615
```

```
[140]: # Logistic Regression needs a small transformation to get the right coefficients
      transformed_coefficients = list(np.exp(model_lr.coef_))
      cdf = pd.DataFrame(list(X_train.columns), transformed_coefficients).
      ↪reset_index()
```

```
cdf.columns = ['coefficient', 'feature']
print(cdf.sort_values(by = 'coefficient', ascending = False))
```

	coefficient	feature
0	1.051222	reward
5	1.023997	social
4	1.008203	mobile
9	1.000550	became_member_on_year
3	1.000011	email
8	0.999996	income
11	0.999768	gender_encoded
7	0.998785	age
6	0.995864	web
2	0.992067	duration
10	0.987936	offer_type_encoded
1	0.960984	difficulty

The supervised models trained, with respective F1-Scores, were: * Baseline Model: Dummy Classifier for most frequent class * F1-Score: 64.41% * Model 1: Logistic Regression * F1-Score: 66.62% * Model 2: Naïve Bayes * F1-Score: 70.26% * Model 3: Support Vector Machines (SVM) * 65.10% * Model 4: Decision Tree * F1-Score: 65.08%

The best model was Naïve Bayes, but it has not explainability. The second best and explainable model was the Logistic Regression.

In order to have explainability, we loose predictive power: the F1-Score of the Logistic Regression model is 64.38%. But the value of the reward, followed by the offer sent by social networks. For the increase of one unit in each feature, we expect a increase of the respective coefficient in the success of the offer.

2.7 Conclusion

When we started, we wanted to make better purchasing offers to Starbucks' customers. For this, we used customer's past behaviour to find patterns and try to be more assertive. As given by the Udacity's Starbucks Project Overview, the basic task was to use the data to identify which groups of people are most responsive to each type of offer, and how best to present each type of offer. In other words, this is a classification problem where the model takes user behaviour data as input and produces a group as output (either previously defined or not).

For this project, we spent quite some time dealing with the features, manipulating those to fit into the models. For that to happen, we found a way to define an offer success based on the user funnel performed from the transcript dataset.

Once we had the dataset, we trained 4 supervised learning models and a baseline one. The baseline was an incredibly naïve model that classified the items based on the most frequent class. The model with best performance was a Naïve Bayes. This model doesn't have easy explainability, which means we fail to find understandable patterns to provide offers.

In order to get explainability, we chose the Logistic Regression model, that has lower predictive power. But with this model, we identified the top three most important features: reward, social and mobile.

Next steps would include better feature engineering and selection (we just used all features we could) and other classification models. The model selection should probably account for model explainability, which failed in this case.

[]: