Starbucks Capstone Challenge - Report

Definition

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.

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.

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.

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

Metrics

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).

Since the F1-Score is a weighted average of the precision and recall metrics, it works well to evaluate datasets that aren't very well balanced.

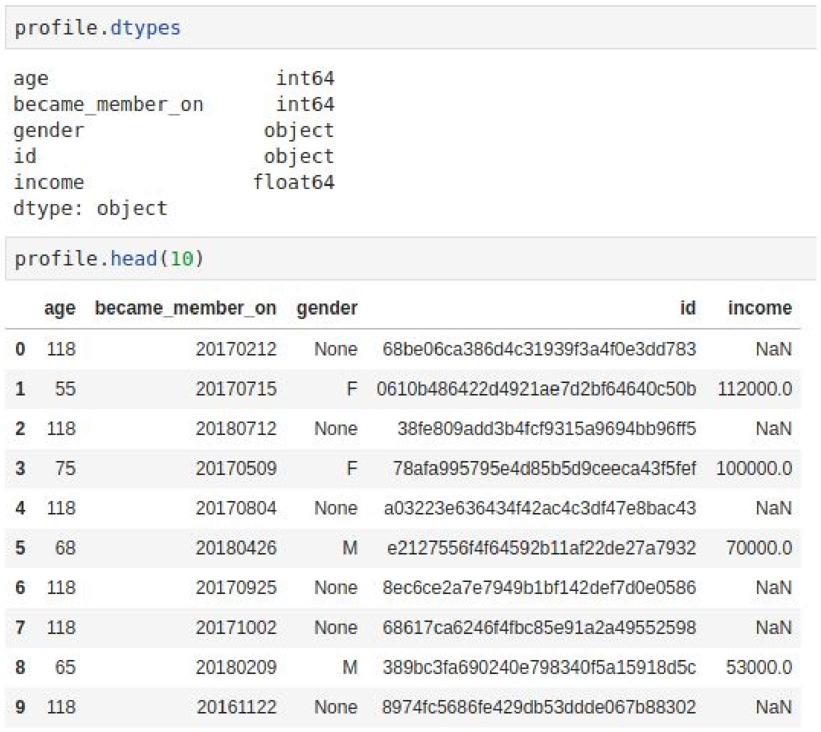
Portfolio dataset preparation

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.

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

Profile dataset preparation

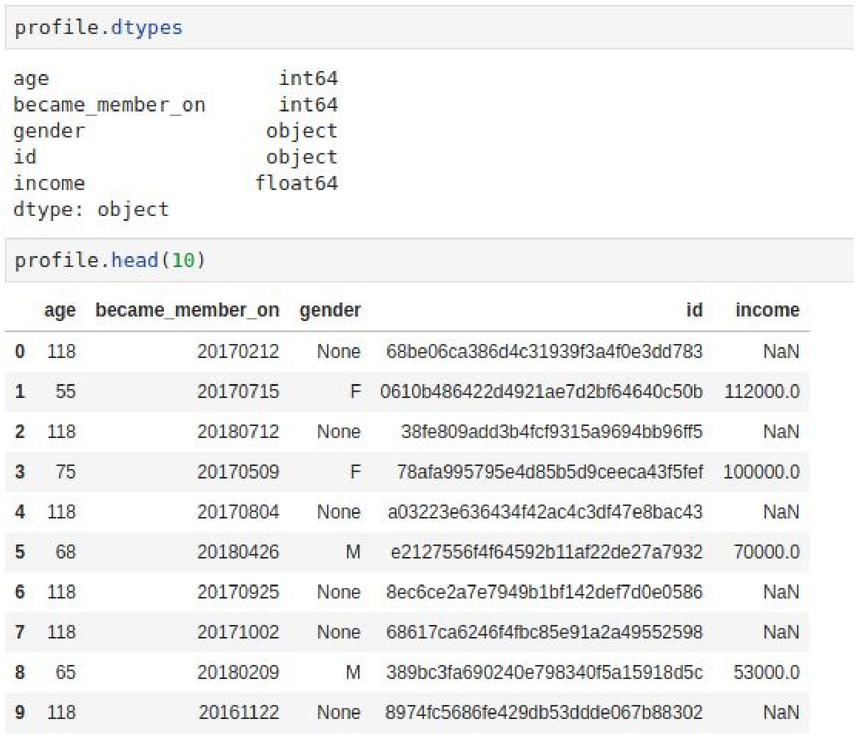


```
profile.dtypes
```

```
age                  int64
became_member_on     int64
gender              object
id                  object
income             float64
dtype: object
```

```
profile.head(10)
```

	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



```
profile.dtypes
```

```
age                  int64
became_member_on     int64
gender              object
id                  object
income             float64
dtype: object
```

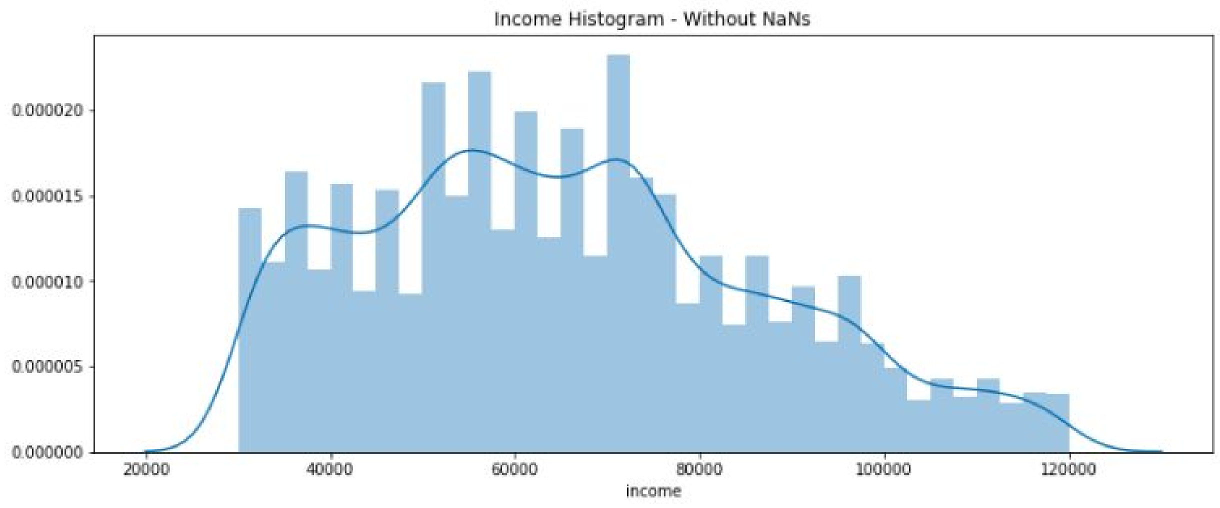
```
profile.head(10)
```

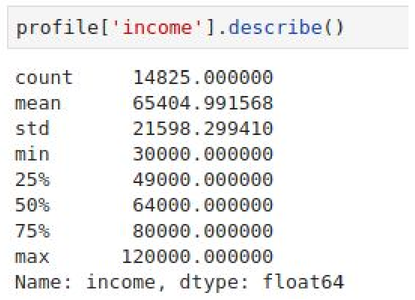
	age	became_member_on	gender	id	income
0	118	20170212	None	68be06ca386d4c31939f3a4f0e3dd783	NaN
1	55	20170715	F	0610b486422d4921ae7d2bf64640c50b	112000.0
2	118	20180712	None	38fe809add3b4fcf9315a9694bb96ff5	NaN
3	75	20170509	F	78afa995795e4d85b5d9ceeca43f5fef	100000.0
4	118	20170804	None	a03223e636434f42ac4c3df47e8bac43	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 O value - that probably means 'others'). The None value for gender is not understood as a value, so it needs treatment. We'll fill the None values with Unknown.
- 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.

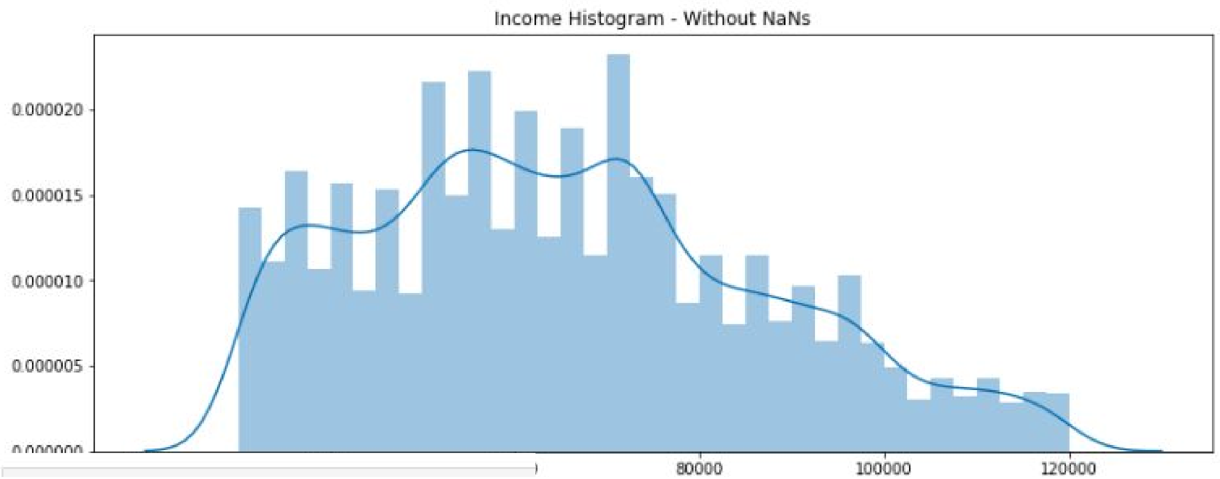
For the income column, let’s see what’s the data distribution without the NaNs:



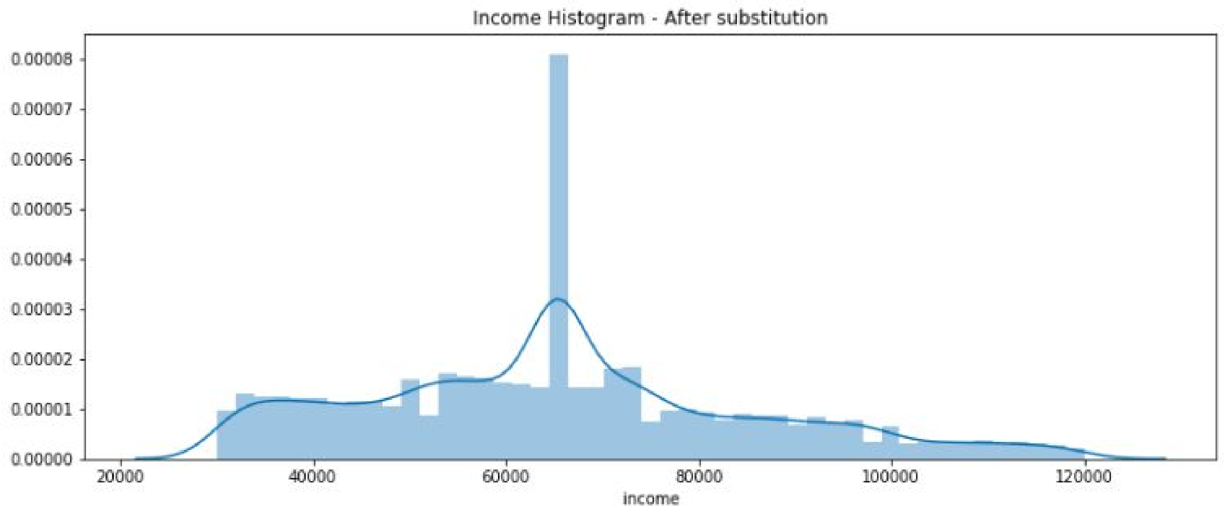


```
profile['income'].describe()
```

```
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
```



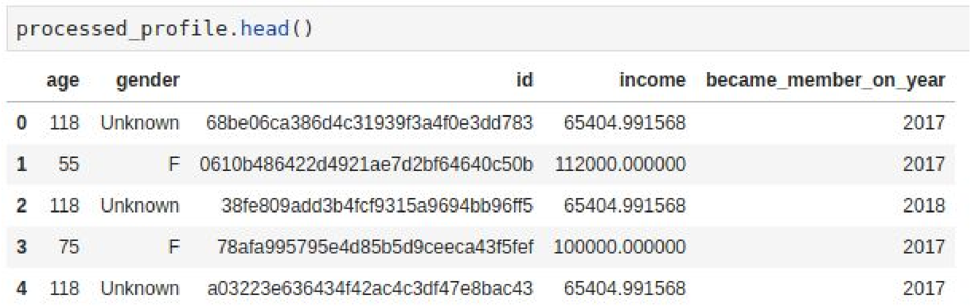
By filling the NaNs with some value, we'll change the distribution's shape. Since both mean and median are fairly close to each other, it doesn't matter much which one we'll use.
I chose the mean.



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

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

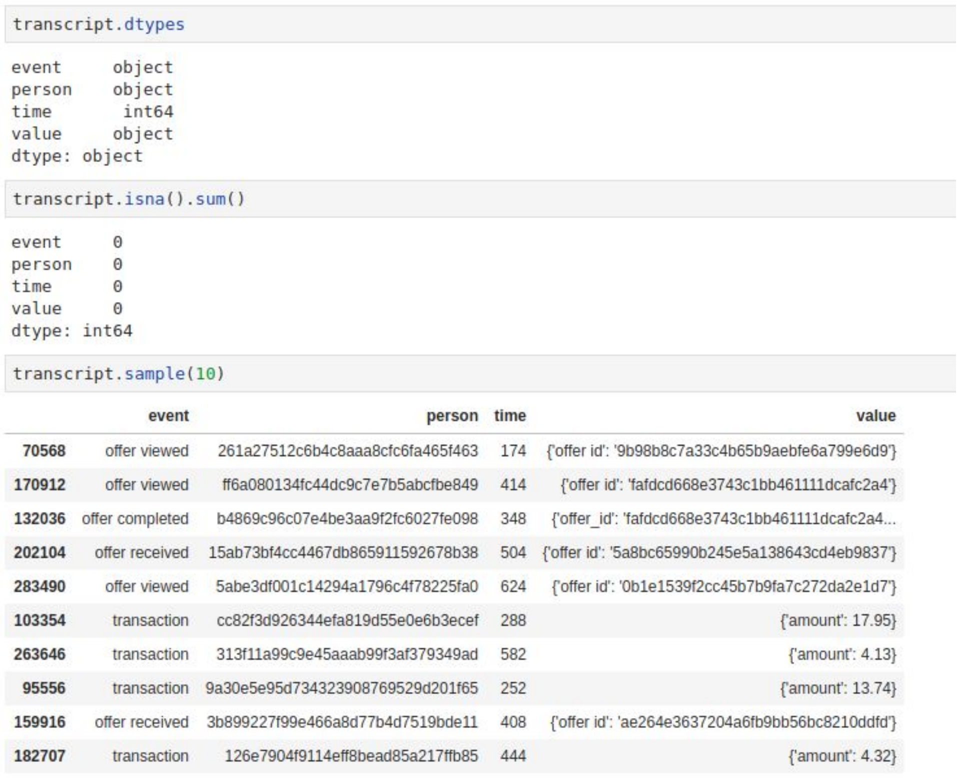
Another alteration we can work on is to transform the became_member_on column into a datetime object column, which allows us to work with time windows. This means that we can get the information for how long a customer has been a member of the program. Since we don't really know when those campaigns 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.



```
processed_profile.head()
```

	age	gender	id	income	became_member_on_year
0	118	Unknown	68be06ca386d4c31939f3a4f0e3dd783	65404.991568	2017
1	55	F	0610b486422d4921ae7d2bf64640c50b	112000.000000	2017
2	118	Unknown	38fe809add3b4fcf9315a9694bb96ff5	65404.991568	2018
3	75	F	78afa995795e4d85b5d9ceeca43f5fef	100000.000000	2017
4	118	Unknown	a03223e636434f42ac4c3df47e8bac43	65404.991568	2017

Transcript dataset preparation



```
transcript.dtypes
```

```
event     object
person    object
time       int64
value     object
dtype: object
```

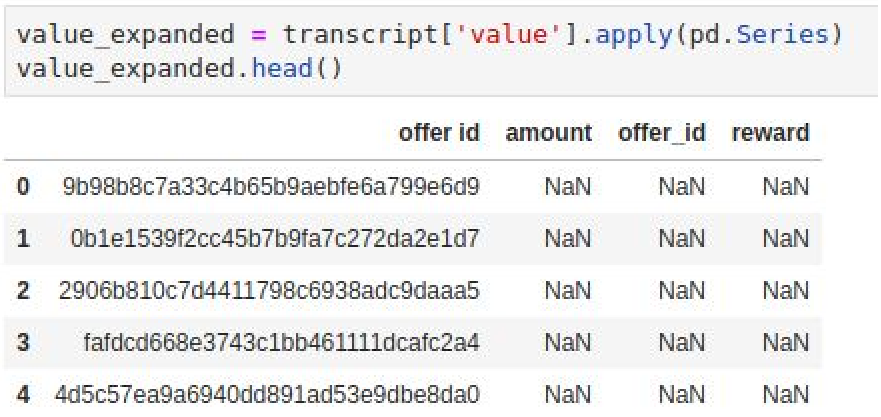
```
transcript.isna().sum()
```

```
event     0
person    0
time      0
value     0
dtype: int64
```

```
transcript.sample(10)
```

	event	person	time	value
70568	offer viewed	261a27512c6b4c8aaa8cfc6fa465f463	174	{'offer id': '9b98b8c7a33c4b65b9aebfe6a799e6d9'}
170912	offer viewed	ff6a080134fc44dc9c7e7b5abcfbe849	414	{'offer id': 'fafdcd668e3743c1bb461111dcafc2a4'}
132036	offer completed	b4869c96c07e4be3aa9f2fc6027fe098	348	{'offer_id': 'fafdcd668e3743c1bb461111dcafc2a4...
202104	offer received	15ab73bf4cc4467db865911592678b38	504	{'offer id': '5a8bc65990b245e5a138643cd4eb9837'}
283490	offer viewed	5abe3df001c14294a1796c4f78225fa0	624	{'offer id': '0b1e1539f2cc45b7b9fa7c272da2e1d7'}
103354	transaction	cc82f3d926344efa819d55e0e6b3ecef	288	{'amount': 17.95}
263646	transaction	313f11a99c9e45aaab99f3af379349ad	582	{'amount': 4.13}
95556	transaction	9a30e5e95d734323908769529d201f65	252	{'amount': 13.74}
159916	offer received	3b899227f99e466a8d77b4d7519bde11	408	{'offer id': 'ae264e3637204a6fb9bb56bc8210ddfd'}
182707	transaction	126e7904f9114eff8bead85a217ffb85	444	{'amount': 4.32}

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



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

	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

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.

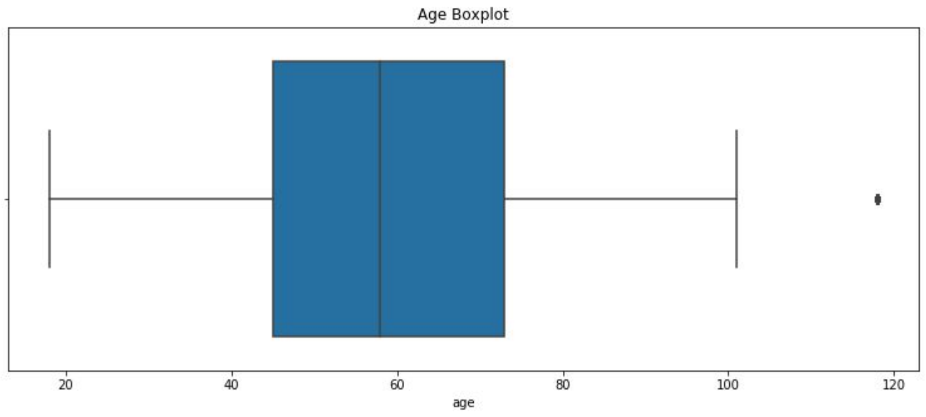
Once we do that, we can include the rest of the transcript data.

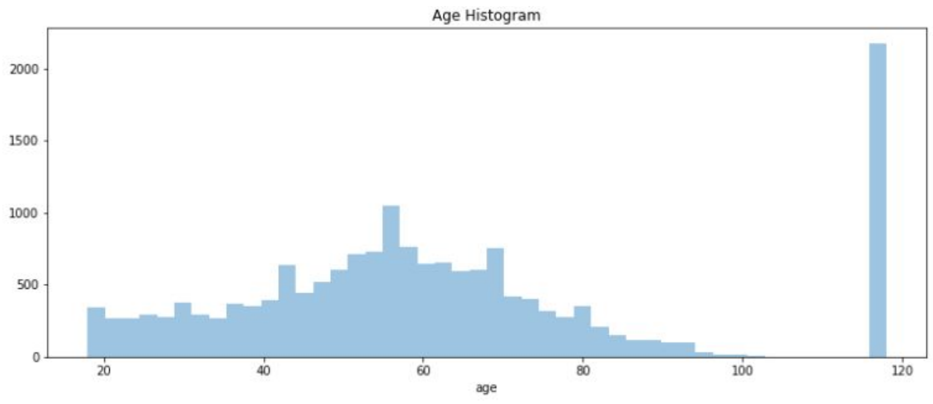
Exploratory data analysis

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

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

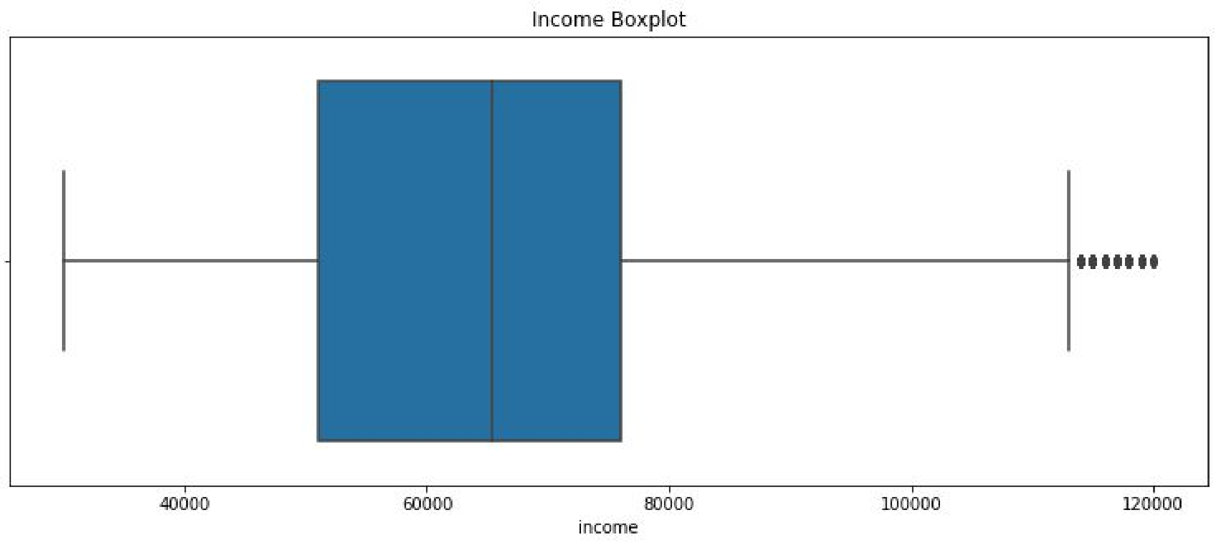
```
count    17000.000000
mean        62.531412
std         26.738580
min         18.000000
25%         45.000000
50%         58.000000
75%         73.000000
max        118.000000
Name: age, dtype: float64
```





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).

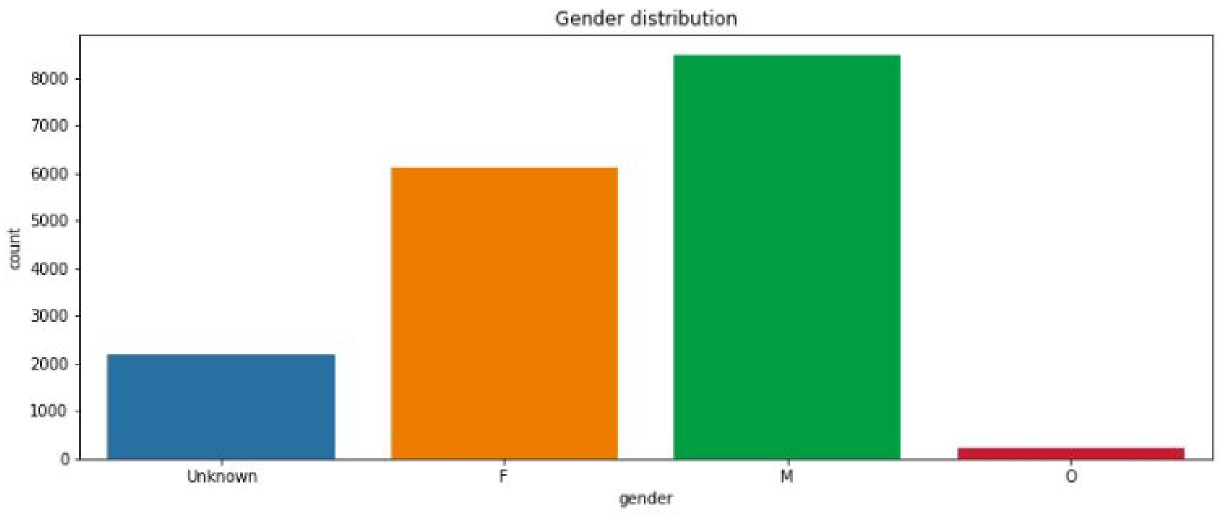
The users are mostly adults, with ages between 40 and 80 and the median in a bit lower than 60 (which is also lower than the mean, that is 62.5).



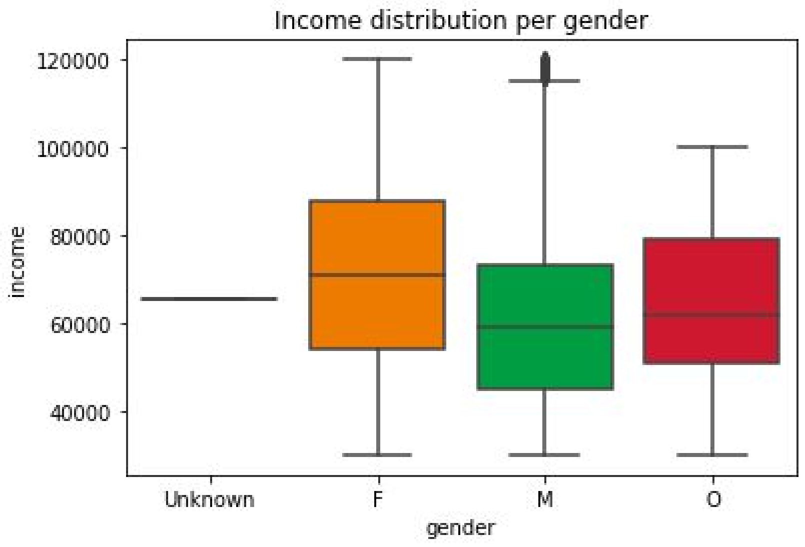
```
processed_profile['income'].describe()
```

```
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
```

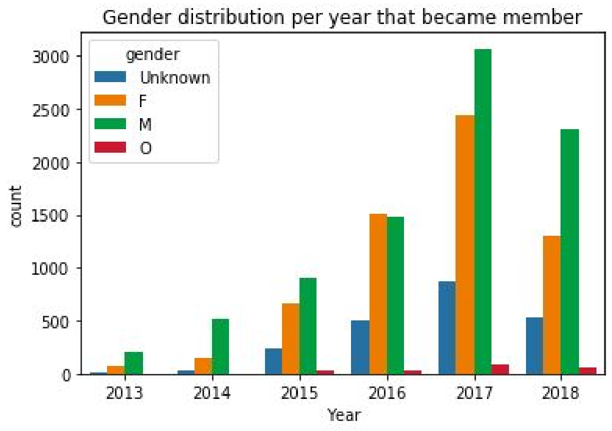
We already saw the histogram for this data, the boxplot shows the distribution is a bit skewed. Mean and median seem close (around $65,000) and most of the data is between $60,000 and $75,000.

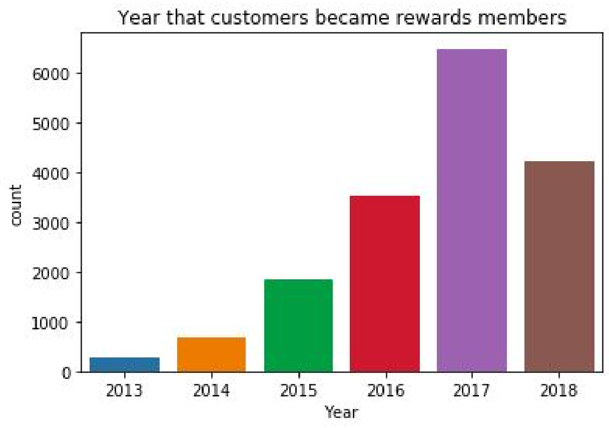


There are more male than females in the dataset. There is a large amount of Unknown and a few 'others'.

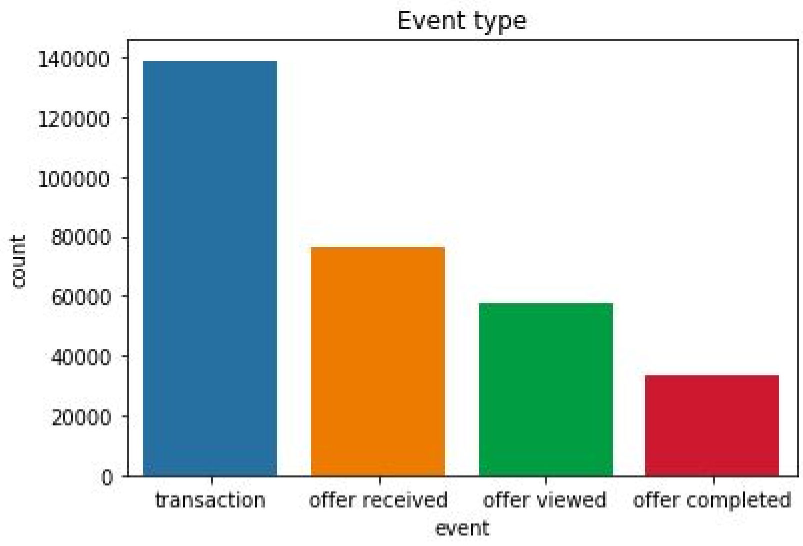


The boxplots show there is no evidence that the income differs across genders.





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).



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.

Merging datasets and data preprocessing

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

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

```
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 inly 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

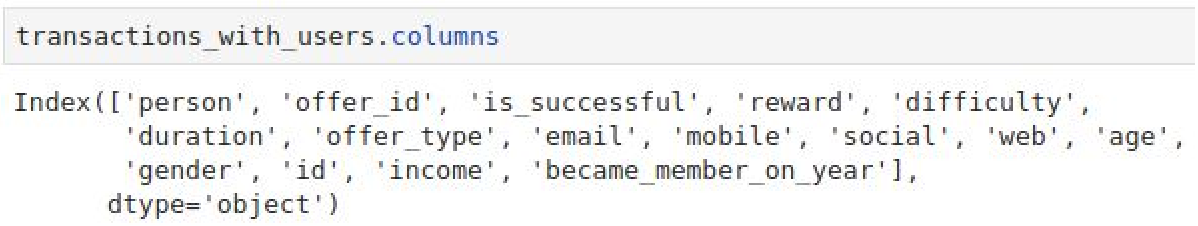
This means that we have to find a way to add the transaction data into the funnel. This can be done through the user id and the timestamp. If a user didn't get to the transaction part, it won't have an offer completed event as well. For the informational offer, the transaction data will only exist if 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.

Finding successful offers

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.

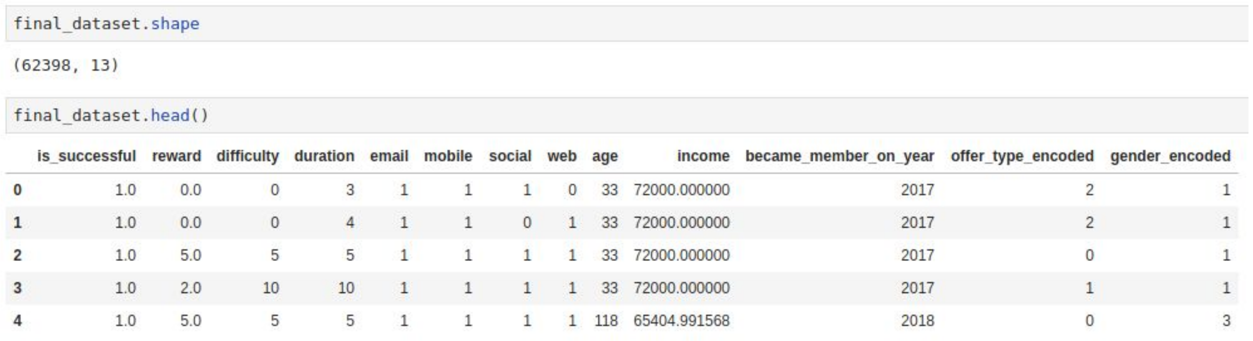
We start by finding which users got to which steps of the funnel. For that, we separate the events in it's own tables and create auxiliary columns. Then we join those datasets to come up with a single dataset that has the offer id, the person id, and three booleans for received, viewed and completed.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



```
transactions_with_users.columns
```

```
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()](#).



```
final_dataset.shape
```

```
(62398, 13)
```

```
final_dataset.head()
```

	is_successful	reward	difficulty	duration	email	mobile	social	web	age	income	became_member_on_year	offer_type_encoded	gender_encoded
0	1.0	0.0	0	3	1	1	1	0	33	72000.000000	2017	2	1
1	1.0	0.0	0	4	1	1	0	1	33	72000.000000	2017	2	1
2	1.0	5.0	5	5	1	1	1	1	33	72000.000000	2017	0	1
3	1.0	2.0	10	10	1	1	1	1	33	72000.000000	2017	1	1
4	1.0	5.0	5	5	1	1	1	1	118	65404.991568	2018	0	3

Modeling

Separating data into train and test

We used Scikit-learn's train_test_split function to separate data, with 80% as train and 20% as test.

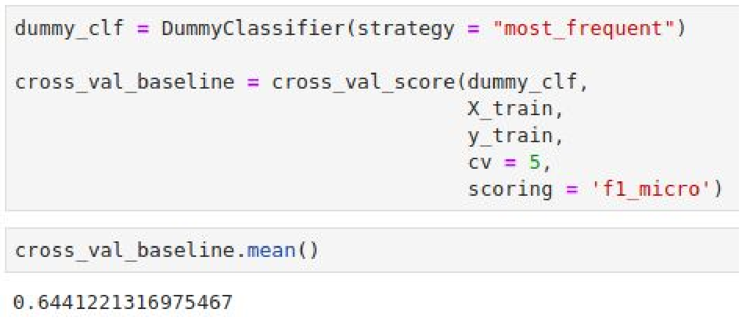
Our independent variables - or features - will be:

- reward
- difficulty
- duration
- email
- mobile
- social
- web
- amount
- age
- income
- became_member_on_year
- offer_type_encoded
- Gender_encoded

We trained supervised models for this. Our dependent variable is is_successful. We used Scikit-learn's framework for all models. We also used Scikit-learn’s cross_val_score with 5 folds.

Baseline model

As baseline model we used a naive one, given by Scikit-learn's DummyClassifier. The model predicts by most frequent class. This is a great baseline model because it's no better than just saying that an offer will be a success just because that's the majority's class (and, since this is an unbalanced problem, this means that predictions by the majority will be better than flipping a fair coin to predict the class).



```
dummy_clf = DummyClassifier(strategy = "most_frequent")

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

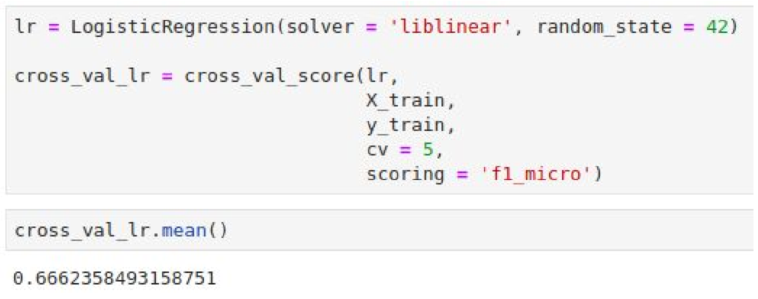
```
cross_val_baseline.mean()
```

```
0.6441221316975467
```

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

Model 1: Logistic Regression

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. The logistic regression (along with the Decision Tree) is one of the models with the easiest explanation we chose to train, which makes it a favourite to be the best model (but first let's see how the models perform).



```
lr = LogisticRegression(solver = 'liblinear', random_state = 42)

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

```
cross_val_lr.mean()
```

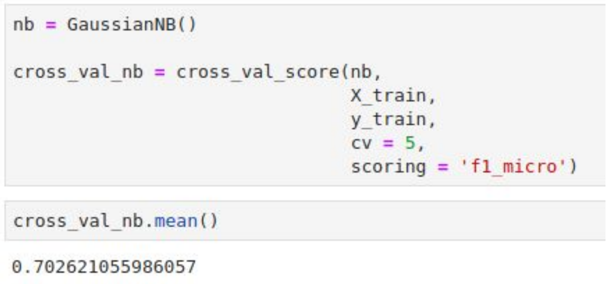
```
0.6662358493158751
```

Even though the logistic regression has analytical solution, Scikit-learn's implementation uses computational methods to solve it. Since this is a small dataset, we'll use the `liblinear` solver as suggested by the documentation.

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

Model 2: Naïve Bayes

We trained a Gaussian Näive 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).
Naïve Bayes classifiers use Bayesian statistics to find the parameter values. An advantage of Naïve Bayes is that it only requires a small number of training data to estimate the parameters necessary for classification, even though this is a naïve model, which makes it great for this problem. Also, it can perform very well for this type of problem, even when the data breaks the Gaussian assumption.



```
nb = GaussianNB()

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

```
cross_val_nb.mean()
```

```
0.702621055986057
```

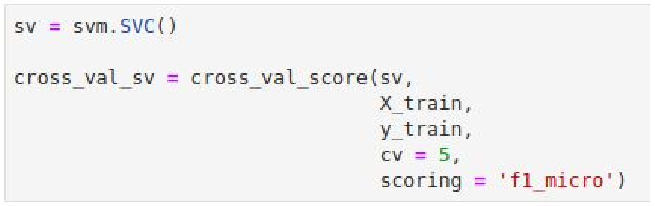
This model performed a bit better than the Logistic Regression and baseline models when it comes to F1-Score: 70.26%.

Model 3: Support Vector Machines (SVM)

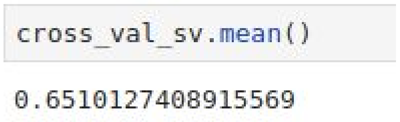
We trained support vector machines. SVMs are considered more complex than the models above. We expect it to perform better than the baseline, but it might start to overfit the model.

An SVM model builds a representation of the categories as points in space, mapped in a way that the categories are each divided by a clear gap that is as wide as possible (see more here. Also, SVMs can efficiently perform a non-linear classification using what is called the kernel trick, implicitly mapping their inputs into high-dimensional feature spaces.

For this problem, we'll use an SVC kernel, which is a linear one.



```
sv = svm.SVC()

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



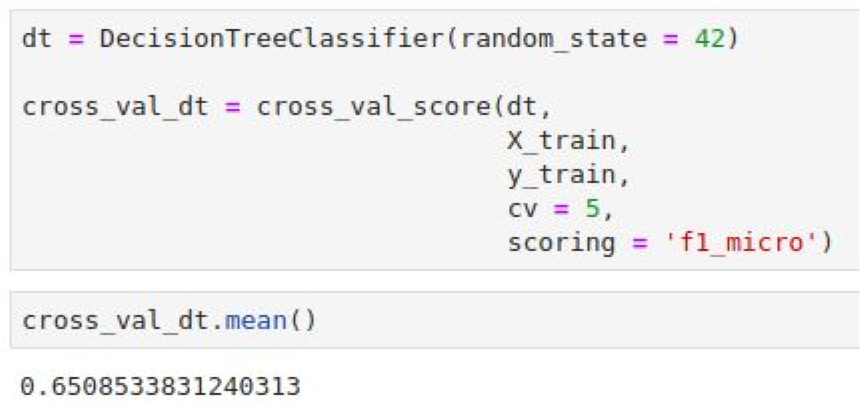
```
cross_val_sv.mean()
```

```
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.

Model 4: Decision Tree

This is the most powerful of all the chosen models and is very likely to overfit. Decision trees are highly explainable because they produce a [flow-chart like structure]() as outcome, which means that we have a linear flow of decision making in order to classify the offers. It's a favourite to best model due to it's explainability (as mentioned above).



```
dt = DecisionTreeClassifier(random_state = 42)

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

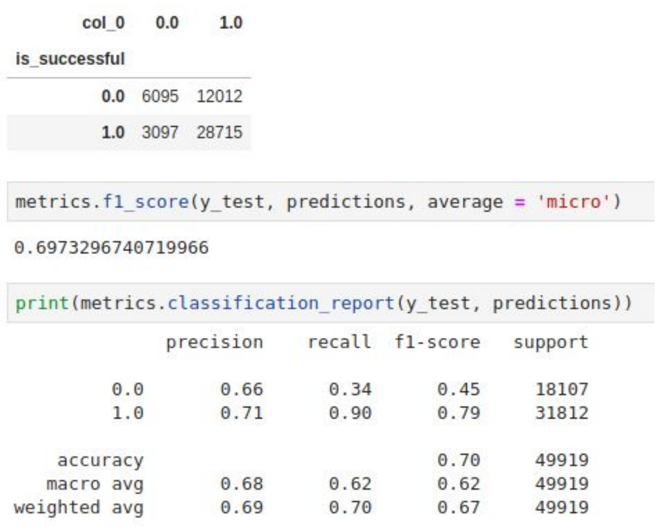
```
cross_val_dt.mean()
```

```
0.6508533831240313
```

This better than the baseline model, since the F1-Score is 65.08%. But it's not the best. The Linear Regression model has better scores and it's explainable as well.

Choosing the best model

Based on the resuls 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.



col_0	0.0	1.0
is_successful		
0.0	6095	12012
1.0	3097	28715

```
metrics.f1_score(y_test, predictions, average = 'micro')
```

```
0.6973296740719966
```

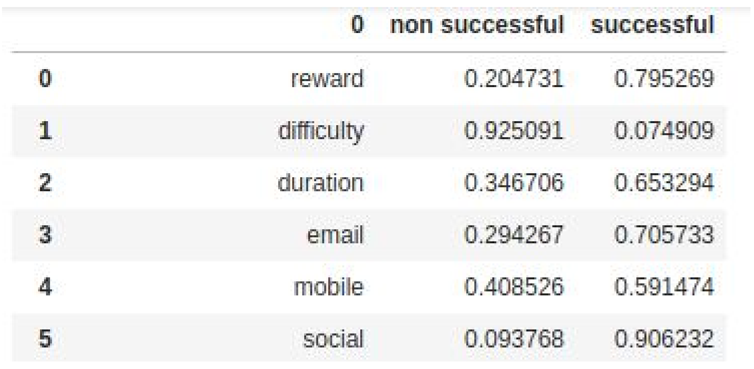
```
print(metrics.classification_report(y_test, predictions))
```

```
              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 didn't perform 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. But since this score is close to what we found during the train stage, this means that this model can be applied to other datapoints and still be trustworthy, as it'll produce similar results.

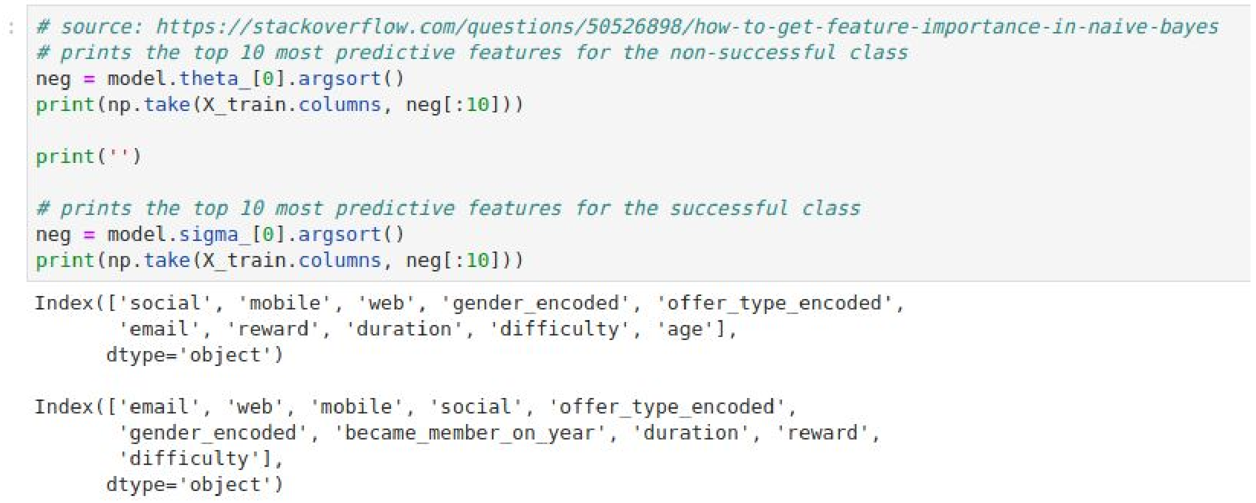
The other usual metrics performed fairly well, with exception of the recall for non-successful offers, which was 34% - meaning that it's prediction is worse than flipping a coin to predict the success of the offer.

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.



	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

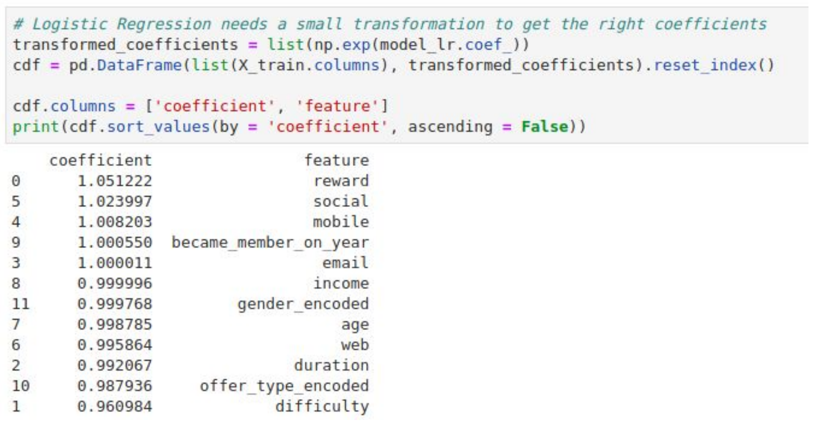
Also we can get the features that had the most predictive contribution to each class, but still no explainability.



```
# 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 Logistic Regression as the best explainable model, we can look into the coefficients to get the most important features.



```
# 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)
    - F1-Score: 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 an increase of the respective coefficient in the success of the offer.

Conclusion and refinement

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 tose 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 understainable patterns to provide offers. But this model achieved reasonable results when applied to the test dataset, meaning it can be applied to other datapoints and still produce the same level of results.

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. To make better decisions here, we could dive deeper into the post-mortem analysis of those models.