Capstone Project

Project Goal

Since it's important to provide customized experience from ordering to offer on loyalty app, the problem becomes to how correctly and precisely locate the target customer group. The objective is to find What are the main features influencing the effectiveness of an offer on the Starbucks app and in a second time to find if the data provided can predict whether a user would take up an offer.The solution aims to analyze how people make purchasing decisions and how those decisions are influenced by promotional offers. In fact, every individual in the dataset has some hidden attributes that impact their buying patterns and are related to their discernible characteristics. Individuals produce different events, including accepting offers, opening offers, and making buys. We notice only three types of offers that can be sent, buy-one-get-one (BOGO), discount, and informational. In a BOGO offer, a user needs to spend a certain amount to get a reward equal to that threshold amount. In a discount, a user gains a reward equal to a fraction of the amount spent. In an informational offer, there is no reward, but neither is there a required amount that the user is expected to spend.

The project will be divided in several steps:

- Prepare and clean data: combine transaction, demographic and offer data. Understand the connection between columns and dataset. Try to get the useful information from data.

- Data exploration -- In order to analyze the problem better in next sections, we first need to explore the datasets which includes checking the missing value and visualizing the data distribution.

- Data preprocessing -- In order to find out what mainly affect the finish of the transaction by sending the offer, in the data processing process, also need to process the data to merge the events of each specific offer sent so as to find out which offer were received, viewed and finally completed with a transaction.

- Feature engineering -- After basic processing, the next step will look if there are any columns that can be used to create new features. For example, generating a new column for length of customer's membership, the count of offer received for each user, calculate the time lap between offers.

- Building model – after pre-processing and feature engineering, next step is to build the model using response flag generated in previous steps to predict whether the customer will respond to the offer or not

- Model tuning – Compare the model using metrics selected above and tune the parameters of initial model using GridSearch method to get higher performance.

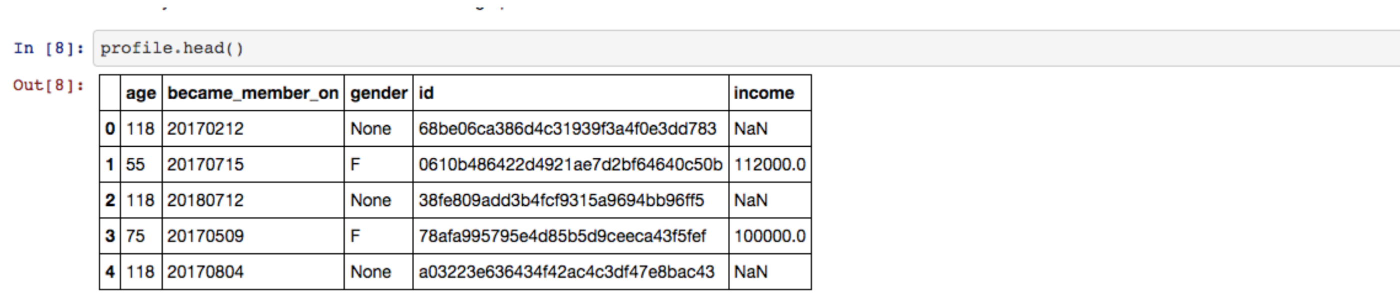
- Conclusion and further improvement – compare the final selected model to benchmark to see if the solution provide a better personalized offer. Also, review the built process and see if there's any opportunities to enhance the model in the future.

Metrics

Since the project is building classification model is based on both accuracy and F1 score as the model evaluation metric. The reason of choosing both metrics is for when the dataset is imbalanced, the accuracy only couldn't objectively show how the model is performing on the dataset, while F1 score provides a better sense of model performance compared to purely accuracy as takes both false positives and false negatives in the calculation. With an imbalanced class distribution, F1 may be more useful than accuracy. Also, since the F1 score is based on the harmonic mean of precision and recall and focuses on positive cases. For the Starbucks app here, it would be fine as we would prioritize more on whether offers are effective, and less focus on why offers are ineffective.

Data Exploration

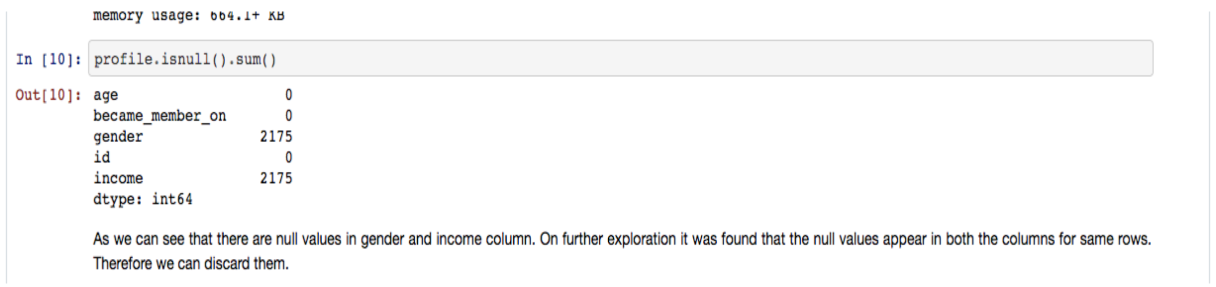
In order to analyze the problem in an appealing format in the next sections, we first need to explore the datasets which includes checking the missing value, visualizing the data distribution, etc. In that way, we can have a better understanding of how the dataset looks like and how we can feature the data to make it ready for modelling.



```
In [8]: profile.head()
```

Out[8]:

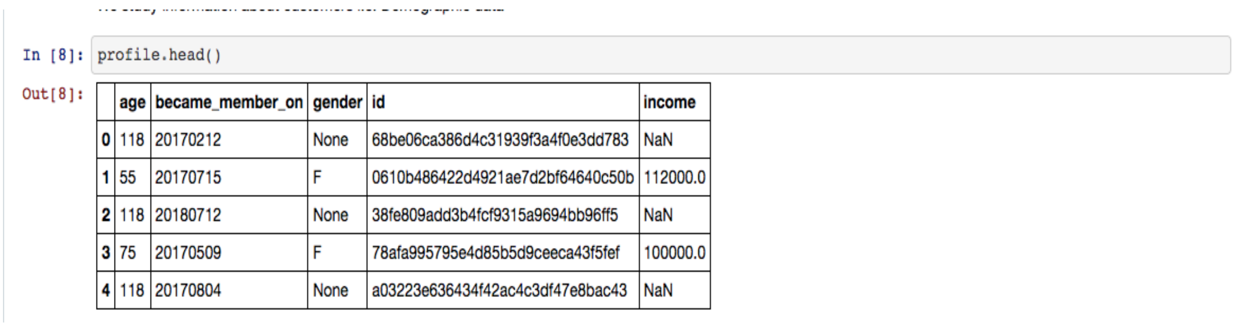
	age	became_member_on	gender	id	income
0	118	20170212	None	68be06ca386d4c31939f3a4f0e3dd783	NaN
1	55	20170715	F	0610b486422d4921ae7d2bf64640c50b	112000.0
2	118	20180712	None	38fe809add3b4fcf9315a9694bb96ff5	NaN
3	75	20170509	F	78afa995795e4d85b5d9ceeca43f5fef	100000.0
4	118	20170804	None	a03223e636434f42ac4c3df47e8bac43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



```
memory usage: 664.1+ KB

In [10]: profile.isnull().sum()

Out[10]: age                    0
         became_member_on       0
         gender              2175
         id                     0
         income              2175
         dtype: int64
```

As we can see that there are null values in gender and income column. On further exploration it was found that the null values appear in both the columns for same rows. Therefore we can discard them.

As shown above, there are no missing values in the portfolio dataset. The channels columns require to be one-hot encoded. we need to rename id column name to offer_id.



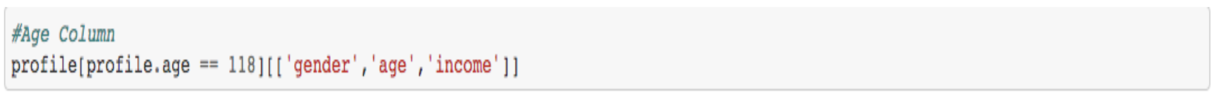
```
In [8]: profile.head()
```

Out[8]:

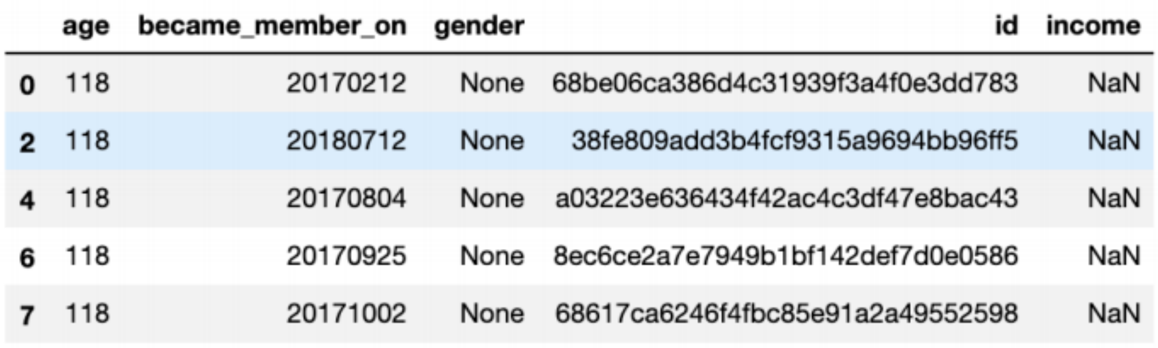
	age	became_member_on	gender	id	income
0	118	20170212	None	68be06ca386d4c31939f3a4f0e3dd783	NaN
1	55	20170715	F	0610b486422d4921ae7d2bf64640c50b	112000.0
2	118	20180712	None	38fe809add3b4fcf9315a9694bb96ff5	NaN
3	75	20170509	F	78afa995795e4d85b5d9ceeca43f5fef	100000.0
4	118	20170804	None	a03223e636434f42ac4c3df47e8bac43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

We notice that values in the age column, which is encoded as 118 are mossing, we therefore need to investigate further:

→ First, we drop rows with no gender and income data:

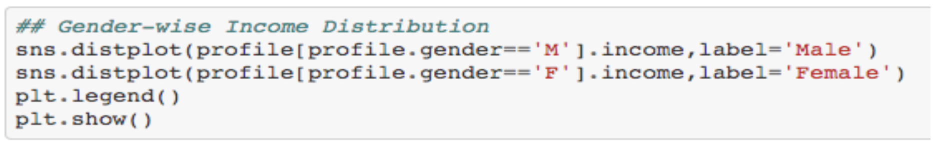


```
#Age Column
profile[profile.age == 118][['gender','age','income']]
```

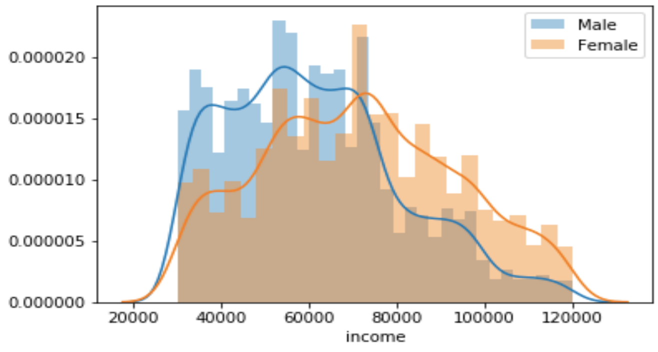


	age	became_member_on	gender	id	income
0	118	20170212	None	68be06ca386d4c31939f3a4f0e3dd783	NaN
2	118	20180712	None	38fe809add3b4fcf9315a9694bb96ff5	NaN
4	118	20170804	None	a03223e636434f42ac4c3df47e8bac43	NaN
6	118	20170925	None	8ec6ce2a7e7949b1bf142def7d0e0586	NaN
7	118	20171002	None	68617ca6246f4fbc85e91a2a49552598	NaN

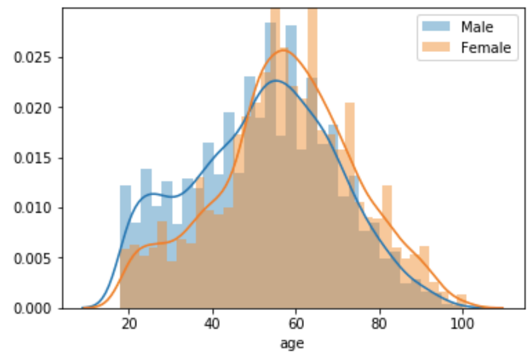
There are 17000 unique customer information within dataset, and there are a number of missing values in age column which is encoded as 118. By filtering out the entries which have 118 in age columns, there are 2175 missing ages, which also have missing values in both gender and income columns. Since it doesn't account big proportion of the whole dataset. These entries can be removed in later steps. And quickly look at the statistics summary of age and income features both of which have normal distribution.



```
## Gender-wise Income Distribution
sns.distplot(profile[profile.gender=='M'].income,label='Male')
sns.distplot(profile[profile.gender=='F'].income,label='Female')
plt.legend()
plt.show()
```



```
## Gender-wise age distribution
sns.distplot(profile[profile.gender=='M'].age,label='Male')
sns.distplot(profile[profile.gender=='F'].age,label='Female')
plt.legend()
plt.show()
```



Age distribution plot depicts that the median age of a customer is 60 and most of the customers belong to the age range between 40 to 70. Income distribution plot shows that the number of customers whose average salary is less than 70,000 is high than the other side considering 70,000 to be median of the income distribution. Plots also conclude that minimum and maximum income for both male and female are approximately the same but the count of male customers in low-income level is slightly higher than that of female customers

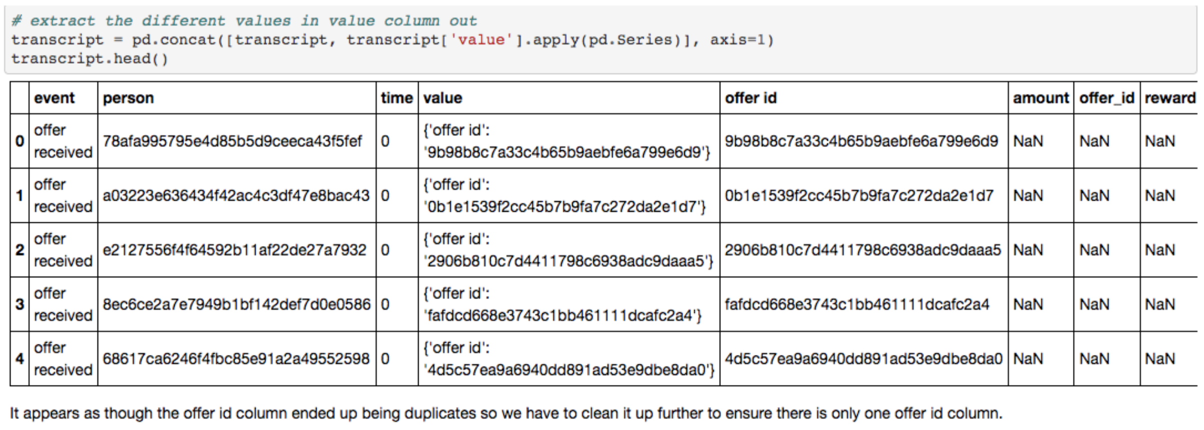
The last dataset is transcript which contains records for transactions, offers received, offers viewed, and offers completed. The dataset has 306534 records with 4 columns. Except transaction entries which accounts a big amount of data, the other three types of event have similar proportion within the dataset.

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

Cleaning and preprocessing of the transcript is done as follows:

1/Convert time in hours to time in days
2/Process the value column
3/Segregate offer and transaction data

Since the value columns include multiple information which should be removed for clarity, it is essential to do some prior basic arrangement first:



```
# extract the different values in value column out
transcript = pd.concat([transcript, transcript['value'].apply(pd.Series)], axis=1)
transcript.head()
```

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

It appears as though the offer id column ended up being duplicates so we have to clean it up further to ensure there is only one offer id column.

Algorithm and Techniques

To solve the problem stated previously, the project will first do some data preprocessing which aims to abstract the information within transcript dataset, then combine the transcript dataset to portfolio and customer information.

Then, feature engineering will be applied to get some more useful features to help building more accurate machine learning model such as time since the user becomes a member, number of offers received, number of transactions completed…

After getting the dataset prepared, the next step is building the model. Here will try simple decision tree classifier and random forest. Since we have 3 offer types, there are thus 3 different models to be built. This is effectively a binary classification supervised learning model. We decided to compare the performance of a simple decision tree classifier model as a baseline model, with an ensemble random forest classifier model. Reason for which we selected tree-based models because we also want interpretability of the model. And this project also attempts to predict whether the customer will respond to the different types of offers or not. Meanwhile, we also selected a random forest as an alternate model to compare the baseline model. Random Forest is an ensemble bagging of decision trees, which aim towards a high accuracy in training the model.

Methodology

Data Processing

In order to identify the main drivers of an effective offer, we first define what an 'effective' offer is within the Starbucks app. We also need to process the data to merge the events of each specific offer sent so as to find out which offer was received, viewed and finally completed with a transaction.

Since offer_id is not associated with any ‘transaction’ event, in order to flag whether the offer has been finally completed with a transaction, here we need to link the offer id back to all transaction events. For BOGO and discount offer, both of them will have the consequence of offers received, viewed, transaction and offer completed which will apparently show that the offer is redeemed and should definitely be sent out. For the information offer, though there’s no reward step there should still be a transaction that is linked to the usage of the offer.

```
# clean up the dataframe (removing the dummy columns)
transcript_processed['offer_id'] = np.where(transcript_processed['offer_id_x'].isnull(),\
                                            transcript_processed['offer_id_y'], transcript_processed['offer_id_x'])
transcript_processed.drop(columns=['offer_id_x','offer_id_y'], axis=1, inplace=True)
```

```
#merge portfolio dataset to get offer data
transcript_processed = transcript_processed.merge(portfolio, how = 'left',on='offer_id')
transcript_processed['duration'] = np.where(transcript_processed['duration_x'].isnull(), \
                                            transcript_processed['duration_y'], transcript_processed['duration_x'])
transcript_processed.drop(columns=['duration_x','offer_type_x','difficulty_x','channels_x','duration_y'],\
                          axis=1, inplace=True)
transcript_processed.rename(columns={'channels_y':'channels','reward_y':'reward','difficulty_y':'difficulty','offer_type_y':'o
ffer_type'},inplace=True)
```

```
# quick check on processed dataset
transcript_processed.head()
```

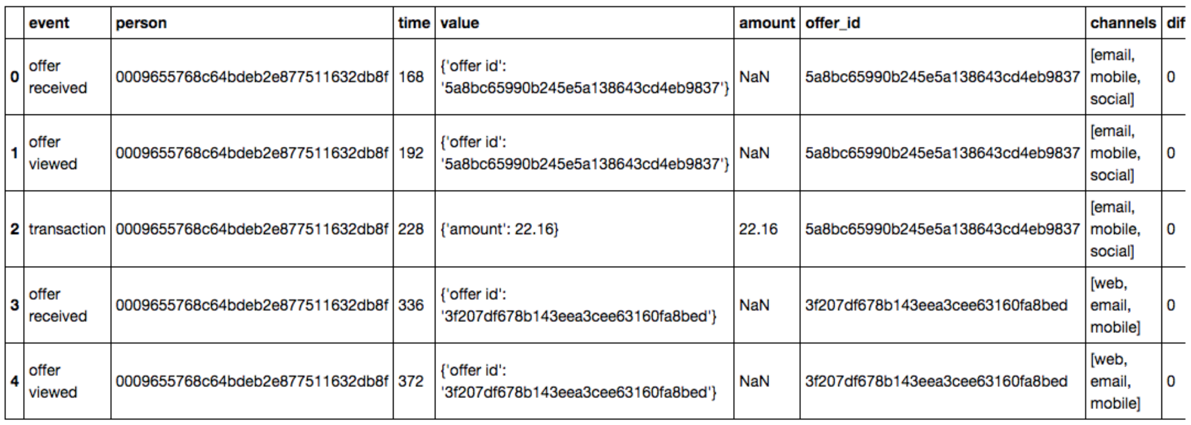
	event	person	time	value	amount	offer_id	channels	dif
0	offer received	0009655768c64bdeb2e877511632db8f	168	{'offer id': '5a8bc65990b245e5a138643cd4eb9837'}	NaN	5a8bc65990b245e5a138643cd4eb9837	[email, mobile, social]	0
1	offer viewed	0009655768c64bdeb2e877511632db8f	192	{'offer id': '5a8bc65990b245e5a138643cd4eb9837'}	NaN	5a8bc65990b245e5a138643cd4eb9837	[email, mobile, social]	0
2	transaction	0009655768c64bdeb2e877511632db8f	228	{'amount': 22.16}	22.16	5a8bc65990b245e5a138643cd4eb9837	[email, mobile, social]	0
3	offer received	0009655768c64bdeb2e877511632db8f	336	{'offer id': '3f207df678b143eea3cee63160fa8bed'}	NaN	3f207df678b143eea3cee63160fa8bed	[web, email, mobile]	0
4	offer viewed	0009655768c64bdeb2e877511632db8f	372	{'offer id': '3f207df678b143eea3cee63160fa8bed'}	NaN	3f207df678b143eea3cee63160fa8bed	[web, email, mobile]	0

Next, after we get the data together, we need to extract the transactions which were completed after the offer was received and viewed. Since we’ve already filled all transaction’s offer id, we can extract the transactions converted from offers by checking if the offer id before the transaction is the same as the transaction’s offer id.

```
# subset the dataset with only offer viewed, transaction, and offer completed events
transactions_after_viewed = transcript_processed[(transcript_processed['event']=='offer viewed')|\
                                                 (transcript_processed['event']=='transaction')|\
                                                 (transcript_processed['event']=='offer completed')].copy()

# generate the previous offer id
transactions_after_viewed['pre_offer_id'] = transactions_after_viewed.groupby(['person', 'offer_id'])['offer_id'].shift()

# create flag for responsed offer which competed after customer viewing the offer
transactions_after_viewed['completed_offer'] = np.where(transactions_after_viewed['pre_offer_id']==\
                                                        transactions_after_viewed['offer_id'],1,0)
```

```
# join back the 'offer received' events which was filtered out in the previous step
offer_received = transcript_processed[transcript_processed['event']=='offer received']

offer_received['pre_offer_id']=np.nan
offer_received['completed_offer']=np.nan

transcript_processed = offer_received.append(transactions_after_viewed).sort_values(['person','time'])
transcript_processed.head()
```

```
/opt/conda/lib/python3.6/site-packages/ipykernel_launcher.py:4: 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
  after removing the cwd from sys.path.
/opt/conda/lib/python3.6/site-packages/ipykernel_launcher.py:5: 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
  """
```



	event	person	time	value	amount	offer_id	channels	dif
0	offer received	0009655768c64bdeb2e877511632db8f	168	{'offer id': '5a8bc65990b245e5a138643cd4eb9837'}	NaN	5a8bc65990b245e5a138643cd4eb9837	[email, mobile, social]	0
1	offer viewed	0009655768c64bdeb2e877511632db8f	192	{'offer id': '5a8bc65990b245e5a138643cd4eb9837'}	NaN	5a8bc65990b245e5a138643cd4eb9837	[email, mobile, social]	0
2	transaction	0009655768c64bdeb2e877511632db8f	228	{'amount': 22.16}	22.16	5a8bc65990b245e5a138643cd4eb9837	[email, mobile, social]	0
3	offer received	0009655768c64bdeb2e877511632db8f	336	{'offer id': '3f207df678b143eea3cee63160fa8bed'}	NaN	3f207df678b143eea3cee63160fa8bed	[web, email, mobile]	0
4	offer viewed	0009655768c64bdeb2e877511632db8f	372	{'offer id': '3f207df678b143eea3cee63160fa8bed'}	NaN	3f207df678b143eea3cee63160fa8bed	[web, email, mobile]	0

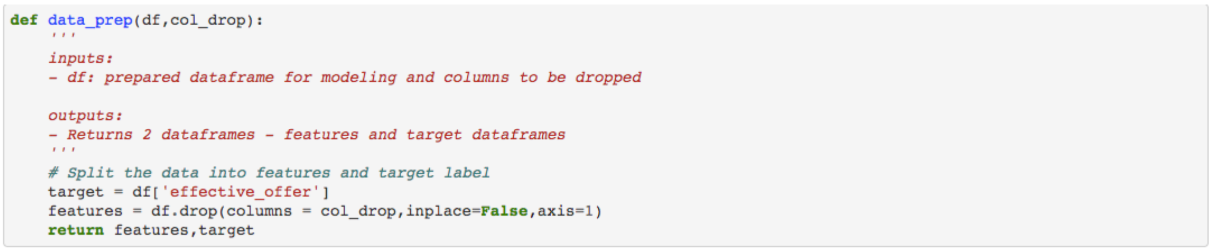
Since the different offer has difference consequence of completion, for example, for the informational offer, there'll not be rewards. Therefore, here separate the transcript data by offer type (BOGO, discount, informational) for easier processing.
Within each offer type, use responded_offer flagged in previous steps we can filter out the offers which were successfully viewed and completed by users. For BOGO and discount offer, the responsed offer should be the one that with 'offer complete' events, and for the informational offer, just 'transaction' can be seen as a successful offer.
Next, will separate out customers who only viewed the offers without transaction and completion at the end and the customers who only received the offer without viewing it. Then, based on merged dataset, we can separate out customers who only viewed the offer after they received the offer and customers who didn't even open the offer after they receive the offer. Do the same steps for both BOGO and discount offer. After separating the different cases of customers, the following steps will firstly focus on customers who finish the transaction after receiving the offer and customers who only view the offer without any transaction. As for the informational offer, the offer could only be counted as responded under the effect of the offer when the transaction is finished within the duration of the offer.
Except basic data processing, basic feature engineering is also included here. There are several simple extended features which may help defining the model later. These features are the length of customer's membership, the count of the offer received for each user, the time lap between offers. After steps above, merge the temporary data created above together, then drop the missing values in gender column, and split the channel column to the categorical variable using dummy variable.

Model Implementation and Preparation

After pre-processing the data, the next step we'll start to implement models to figure out which factors affect most whether the customer will respond to the offer or not. And this project also attempts to predict whether the customer will respond to the different types of offers or not.

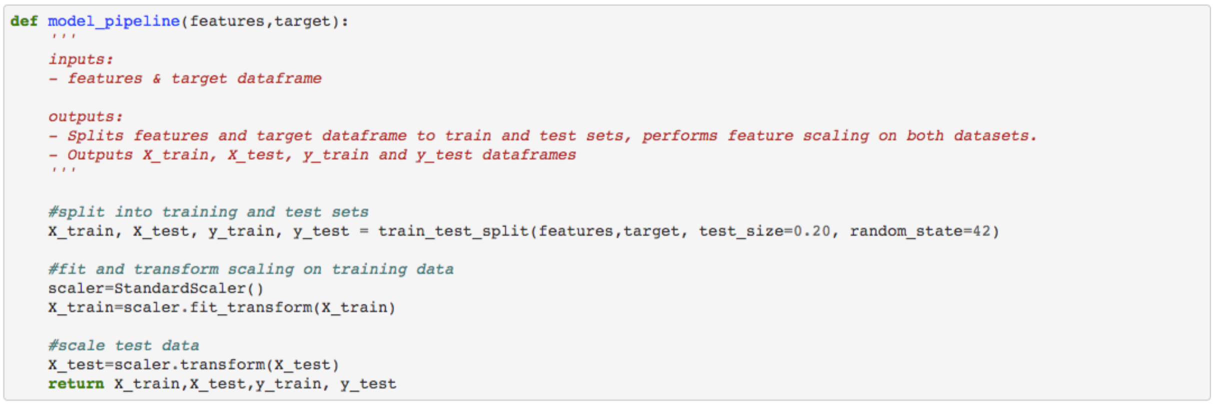
Since the project designing is to separate the dataset by different type of offers, it'll be much convenient to create function to make these steps into module.

First, we prepare the dataset and process features and target columns.



```python
def data_prep(df,col_drop):
    '''
    inputs:
    - df: prepared dataframe for modeling and columns to be dropped

    outputs:
    - Returns 2 dataframes - features and target dataframes
    '''
    # Split the data into features and target label
    target = df['effective_offer']
    features = df.drop(columns = col_drop,inplace=False,axis=1)
    return features,target
```

Then, we split data into training and test sets



```python
def model_pipeline(features,target):
    '''
    inputs:
    - features & target dataframe

    outputs:
    - Splits features and target dataframe to train and test sets, performs feature scaling on both datasets.
    - Outputs X_train, X_test, y_train and y_test dataframes
    '''

    #split into training and test sets
    X_train, X_test, y_train, y_test = train_test_split(features,target, test_size=0.20, random_state=42)

    #fit and transform scaling on training data
    scaler=StandardScaler()
    X_train=scaler.fit_transform(X_train)

    #scale test data
    X_test=scaler.transform(X_test)
    return X_train,X_test,y_train, y_test
```

Last, we create a function to execute the model for different offer types

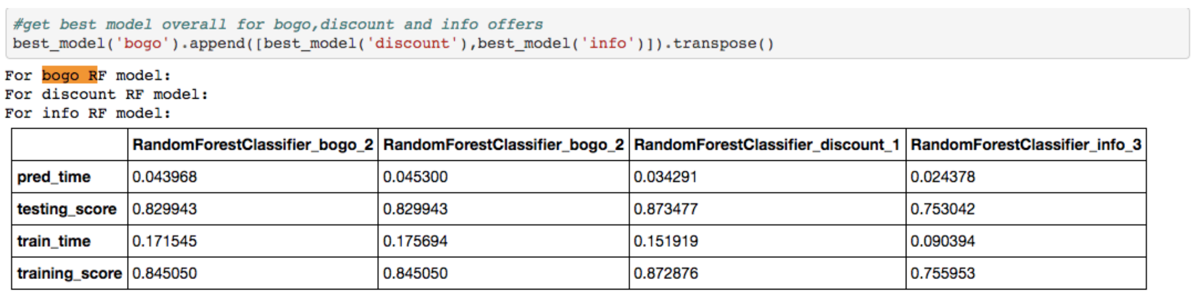
```python
def train_predict(model, X_train, y_train, X_test, y_test):
    '''
    inputs:
       - model: the learning algorithm to be trained and predicted on
       - X_train: features training set
       - y_train: review_scores_rating training set
       - X_test: features testing set
       - y_test: review_scores_rating testing set
    '''
    results = {}

    #Fit the learner to the training data and get training time
    start = time()
    model = model.fit(X_train, y_train)
    end = time()
    results['train_time'] = end-start

    # Get predictions on the test set(X_test)
    start = time()
    predictions_test = model.predict(X_test)
    predictions_train = model.predict(X_train)
    end = time()

    # Calculate the total prediction time
    results['pred_time'] = end-start

    #add training accuracy to results
    results['training_score']=model.score(X_train,y_train)

    #add testing accuracy to results
    results['testing_score']=model.score(X_test,y_test)

    print("{} trained on {} samples.".format(model.__class__.__name__, len(y_train)))
    print("MSE_train: %.4f" % mean_squared_error(y_train,predictions_train))
    print("MSE_test: %.4f" % mean_squared_error(y_test,predictions_test))
    print("Training accuracy:%.4f" % results['training_score'])
    print("Test accuracy:%.4f" % results['testing_score'])
    print(classification_report(y_test, predictions_test,digits=4))
    return results
```

```python
def run_model(clf1,clf2,name):
    '''
    inputs:
    - clf1: first classifier model
    - clf2: 2nd classifier model for comparison
    - name: name of models for comparison

    outputs:
    - Dataframe of results from model training and prediction
    '''

    # Collect results on the learners
    results = {}
    for clf in [clf1, clf2]:
        clf_name = clf.__class__.__name__ + '_' +name
        results[clf_name] = {}
        results[clf_name]= train_predict(clf, X_train, y_train, X_test, y_test)
    return pd.DataFrame(results)
```

Results

Model Evaluation and Validation

Next, we'll look at the model's result and see if there's any insight into main factors which decide whether customers will respond to offers we could get by investigating feature importance.

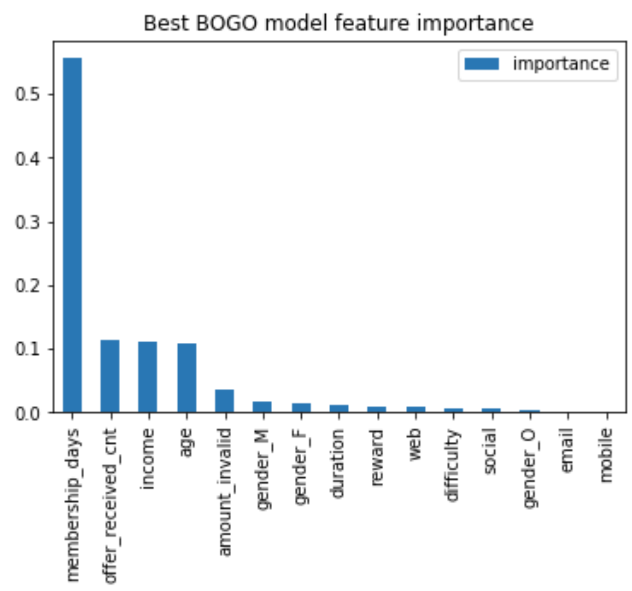


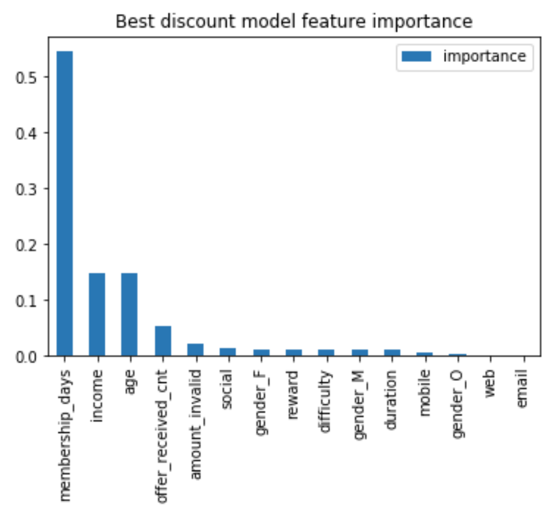
```
#get best model overall for bogo,discount and info offers
best_model('bogo').append([best_model('discount'),best_model('info')]).transpose()
```

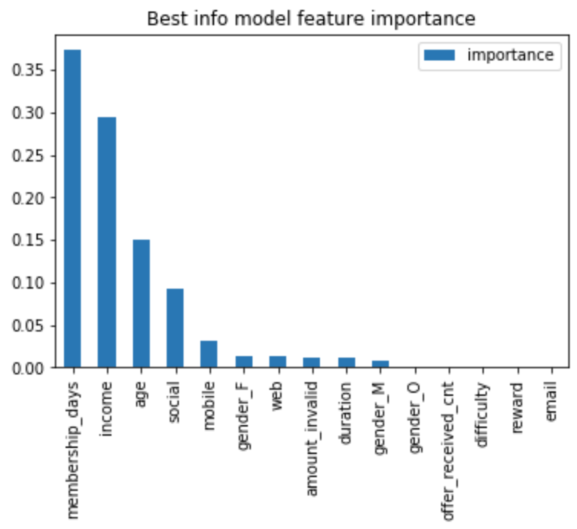
```
For bogo RF model:
For discount RF model:
For info RF model:
```

	RandomForestClassifier_bogo_2	RandomForestClassifier_bogo_2	RandomForestClassifier_discount_1	RandomForestClassifier_info_3
pred_time	0.043968	0.045300	0.034291	0.024378
testing_score	0.829943	0.829943	0.873477	0.753042
train_time	0.171545	0.175694	0.151919	0.090394
training_score	0.845050	0.845050	0.872876	0.755953

Based on our model, let's display the importance of the feature:







Conclusion

This project is trying to figure out:

- What factors mainly affect the usage of the offer from the customer? Should the company send out the offer or not?
- How possible will a customer open and use the offer sent to them? Are there any common characteristics of the customers who take the offer?

By training and tuning the model on three type of offers' dataset, the final tuned models get quite good performance with BOGO 0.838, discount 0.873 and informational 0.753. With the machine learning model, we can know how possible the customer will open, view and finally complete the offer after the offer sent out.

As shown in the previous graphs, we can see that for all three types of offer, the most important factor that largely affects if the offer will be responded to eventually is the length of membership. That is, the longer the customer as a member of Starbucks, the more likely he will respond to the offer they receive. Then the second and third important factors which affect the possibility of customer's response are age and income which very make sense. Also, the number of offers they received will also affect the response a lot.

Improvement

It's worthwhile to try some other enhancement in the step of model tuning. For example, probably, we can do some more experiment on feature engineering step to see if any other new features can improve the model, also I could also try to reduce some feature to see how it will affect the model performance. Also, so far, the analysis is focused more on customer's who successfully finish the transaction after they received the offer, there should be more insight for the other cases where the customer finishes the transactions regardless of the offer. If we could get any insight into those cases, maybe we can send out more offers to those customers. In addition, maybe we could do some unsupervised learning on clustering the customers based on information we are given, to see if there are any specific characteristics on a group of customers who will be more likely to respond to the offer.