RMIT UNIVERSITY

Starbuck Offer Personalization

COSC2789 -Practical Data Science

Final Assignment

Authors: Tran Minh Quang - s3757281

Han sang yeob - s3821179

Hien Nguyen Sy - s3536650

Vu Gia Thinh - s3820373

Lecture: Dr. Nhi Vo

Created Date: 1/2021

Contents

Executive summary

The purpose of this project is to explore the relationship between the user profile and offer types in order to build a model which can suggest which offer is effective for a customer. The process of data science is conducted by the team from data preprocessing, data analysis, feature extraction and modelling to have a profound understanding about the data set and develop an efficient model that can evaluate the success of an offer. Overall, the result illustrated that the team has successfully developed a high-performance model which is LightBGMClassifier with a dashboard and APIs are deployed. However, more work can be done to improve the project such as developing models to predict the best offer type for customers or performing clustering to represent the customer demographic.

Introduction

Starbucks is being well known as the largest coffeehouse company in the world at the moment. However, it is also being famous as a world-leading data-driven company that utilizes the use of data to elevate their business. One of the most famous business data-driven business approaches of Starbucks is personalization promotion. Starbuck uses the data gathered from the Starbucks reward mobile app and performs analysis to send the most suitable offer to a customer. In the final assignment of the course Practical Data Science, a group of four students will try to simulate the analysis process to build a model that can predict which type of offer is effective for the customer.

Project Management

Project goals

As stated above, this notebook will try to answer the question of what is **the most effective offer that customer is most likely to use?** The process of data science is carried out in this notebook to have a deeper understanding of customer demographic and their behavior attributes from the given data set. Then, we will try to predict the most suitable type of promotion for each individual.

Deliverables

At the end of this project several models are trained and evaluated, the model with the highest efficiency will be selected. The chosen model will be exported and deployed using Flask. In addition, a dashboard for visualization which have interactive graphs and model evaluation is deployed using dash framework.

Program stack

For the purpose of "extract, transform, and load" (ETL) data, we use **iPython** and **Jupyter Notebook**. Jupyter is a free, open-source interactive web-based computational notebook. Computational notebooks have been around for several years; however, Jupyter has exploded in popularity over the past couple of years. This capable tool supports multi-language programming and therefore became the *de facto* choice for data scientists for practicing and sharing various codes, quick prototyping, and exploratory analysis.

Dash is an open-source Python framework for building web applications. It is built on top of Flask, Plotly.js, React and React Js. With Dash, we can easily perform data analysis, data exploration, visualization, modelling, instrument control, and reporting. It enables you to build dashboards using pure Python. Dash is open source, and its apps run on the web browser. Dash app code is declarative and reactive, which makes it easy to build complex apps that may contain many interactive factors.

Scikit-learn is the most useful and free-of-charge library for machine learning in Python. It features various algorithms like support vector machines, random forests, and k-neighbors, and it also supports Python numerical and scientific libraries like NumPy and SciPy.

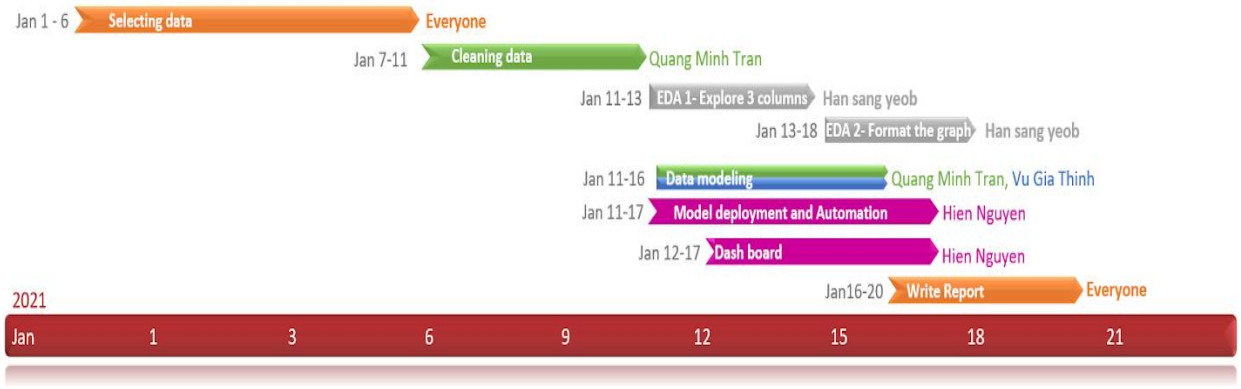
LightGBM is a gradient boosting framework that uses tree-based learning algorithms. It is designed to be distributed and efficient with the following advantages:

- Faster training speed and higher efficiency.
- Lower memory usage.
- Better accuracy.
- Support of parallel and GPU learning.
- Capable of handling large-scale data.

Flask is a lightweight Web Server Gateway Interface WSGI web application framework that was created to make getting started easy and making it easy for new beginners. With the tendency to scale up to complex applications. Flask gives the developer varieties of choice when developing web applications, it provides you with tools, libraries, and mechanics that allow you to build a web application, but it will not enforce any dependencies or tell you how the project should look like. The web application can be a blog, commercial website, or some web pages, it still allows the developers the opportunity to use some extensions provided by the community that allows you to add more functionality to the web application.

Timeline

This project timeline in the last 4 weeks of the semester. Below is the Gantt chart to overview the whole project timeline in 21 days of Jan.



Member contribution

No	Name	Contribution
1	Tran Minh Quang (s3757281)	Data cleaning, Data modelling, Report, Slide
2	Han sang yeob (s3821179)	Data cleaning, EDA, Report, Slide
3	Hien Nguyen Sy (s3536650)	Model deployment and automation, Visualization dashboard, Report
4	Vu Gia Thinh (s3820373)	Data modelling, Report

Methodology

Data understanding

In the beginning of the understanding stage, data scientists must understand the row dataset. For this project, we have used data in three json file,

- portfolio.json: offer id and its relevant data
- profile.json: customer demographic data
- transcript.json: record for transactions, offers received, offers viewed, and offers completed

To fully understand the data, we will scheme through all the columns in each data set.

- portfolio.json
  - id: offer id
  - channels: means of customer receiving offer.
  - difficulty: dollar amount needs to spend in order to complete offer.
  - duration: time length that offer valid.
  - offer_type: types of offer.
  - reward: dollar amount needs to complete offer.
- profile.json
  - age: age of customer
  - gender: gender of customer
  - id: customer id
  - income: customer's annual income
  - became_member_on: day became member.
- transcript.json
  - person: customer id
  - time: time of transaction in hour
  - value: dictionary type, can be offer id or transaction.
  - event: transaction type

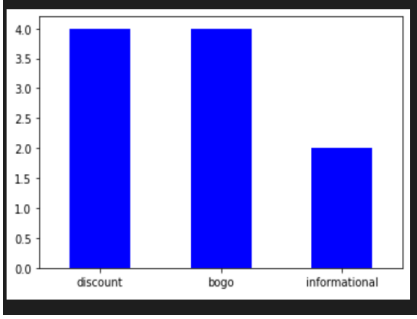
After understanding all the meaning of the columns, we need to describe the data sets. Reading the json files to the jupyter notebook, we can use pd.read_json.

To describe the each data set, for portfolio.json,

```
portfolio.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10 entries, 0 to 9
Data columns (total 6 columns):
 #   Column      Non-Null Count  Dtype
---  ------      --------------  -----
 0   reward      10 non-null     int64
 1   channels    10 non-null     object
 2   difficulty  10 non-null     int64
 3   duration    10 non-null     int64
 4   offer_type  10 non-null     object
 5   id          10 non-null     object
dtypes: int64(3), object(3)
memory usage: 608.0+ bytes
```

```
# Offer types distribution
portfolio['offer_type'].value_counts().plot(kind='bar', color='b', rot=0);
```



As we can see that the data in the portfolio does not have any null value, the categorical features of the data are correct and there do not exist any type error. The only job we need to fix in order to make the data clean is to One-hot encode the categorical feature of the data (channels, offer_type)

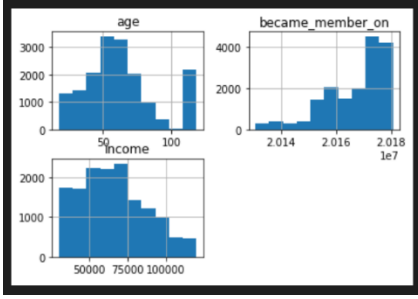
For profile.json,

```
profile.isnull().sum()
```

```
gender              2175
age                    0
id                     0
became_member_on       0
income              2175
dtype: int64
```

```
profile.hist()
```

```
array([[<AxesSubplot:title={'center':'age'}>,
        <AxesSubplot:title={'center':'became_member_on'}>],
       [<AxesSubplot:title={'center':'income'}>, <AxesSubplot:>]],
      dtype=object)
```



There exist null values in gender and income features, therefore we must handle the missing value in the data cleaning process. Also, the max age number is 118 which does not make sense, thus, all the customers who are above 100 should be changed to Nan value. We also need to transform the date time format of became_member_on feature and encoded the gender columns.

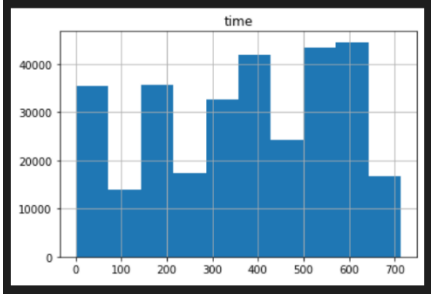
For transcript.json,

```
transcript.info()
```

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

```
transcript.hist()
```

```
array([[<AxesSubplot:title={'center':'time'}>]], dtype=object)
```



Regarding the data from transcript.json, we have to split the value col to offer_id and amount. Also, splitting event col based on offer and transaction. Finally, we have to drop the customer id value that doesn't exist in the profile.json.

Data Cleaning

After understanding the three data sets, we need to start cleaning the data sets.
For portfolio.json,

```
def id_mapper(df):

    coded_dict = dict()
    counter = 1

    for val in df['id']:
        if val not in coded_dict:
            coded_dict[val] = counter
            counter += 1

    return coded_dict
```

Mapper function is to map unique integer ids to customers. Furthermore, the data frame will hold id variables in code_dict which is a dictionary with original and encoded id values.

```
def cleanPortfolio(portfolio):

    mlb = MultiLabelBinarizer()
    portfolio_new = portfolio.copy()
    # Nomorlize the id of the offer
    offer_id_encoded = id_mapper(portfolio_new)
    portfolio_new['offer id'] = portfolio_new['id'].map(offer_id_encoded)

    # Split channel to different collums
    portfolio_new = pd.concat([portfolio_new,
                               pd.DataFrame(mlb.fit_transform(portfolio_new['channels']),columns=mlb.classes_,
index=portfolio_new.index)],
                              axis=1, sort=False
                             )

    # One-hot encode 'offer_type'
    portfolio_new = pd.get_dummies(portfolio_new, columns = ['offer_type'])
    # drop uneccessary collumns
    portfolio_new = portfolio_new.drop(['id', 'channels'], axis=1)


    return portfolio_new, offer_id_encoded

portfolio_new, offer_id_encoded = cleanPortfolio(portfolio)
portfolio_new
```

This is a code for cleaning the portfolio.json. Firstly, we will use id_mapper to normalize id. of the offer. Then we will split the channel into different columns. To compare the information of the portfolio.json into portfolio_new.json

```
portfolio.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10 entries, 0 to 9
Data columns (total 6 columns):
 #   Column      Non-Null Count  Dtype
---  ------      --------------  -----
 0   reward      10 non-null     int64
 1   channels    10 non-null     object
 2   difficulty  10 non-null     int64
 3   duration    10 non-null     int64
 4   offer_type  10 non-null     object
 5   id          10 non-null     object
dtypes: int64(3), object(3)
memory usage: 608.0+ bytes
```

Before cleaning

```
portfolio_new.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10 entries, 0 to 9
Data columns (total 11 columns):
 #   Column                    Non-Null Count  Dtype
---  ------                    --------------  -----
 0   reward                    10 non-null     int64
 1   difficulty                10 non-null     int64
 2   duration                  10 non-null     int64
 3   offer id                  10 non-null     int64
 4   email                     10 non-null     int64
 5   mobile                    10 non-null     int64
 6   social                    10 non-null     int64
 7   web                       10 non-null     int64
 8   offer_type_bogo           10 non-null     uint8
 9   offer_type_discount       10 non-null     uint8
 10  offer_type_informational  10 non-null     uint8
dtypes: int64(8), uint8(3)
memory usage: 798.0 bytes
```

After cleaning

As we can see, the data is represented as we expected, offer id is normalized to integer. The categorical columns channel and offer_type is one hot encoded.

For profile.json,

```
def cleanProfile(profile):

    profile_new = profile.copy()

    # Drop collumn with missing gender and age
    profile_new = profile_new.dropna()

    cust_id_encoded = id_mapper(profile_new)
    profile_new['customer_id'] = profile_new['id'].map(cust_id_encoded)
    profile_new = profile_new.drop('id', axis=1)

    # Format date in became_member_on
    profile_new['became_member_on'] = profile_new['became_member_on'].apply(lambda x: pd.to_datetime(str(x), format='%Y%m%d'))
    profile_new['membership_duration'] = profile_new.became_member_on.max() - \
        profile_new['became_member_on']
    profile_new['membership_duration'] = profile_new['membership_duration'].apply(
        lambda x: x.days if x is not np.nan else x)

    # One hot encoded gender
    profile_new = pd.get_dummies(profile_new, columns = ['gender'])
    # change the age number over 100 to NaN value
    profile_new.loc[profile_new['age'] > 100, 'age'] = np.nan

    return profile_new, cust_id_encoded

profile_new, cust_id_encoded = cleanProfile(profile)
profile_new
```

First of all, we will drop all the columns that have missing values of gender and age. Then we will change became_member_on into date value using lambda function. After that we will change the customer's age which is over 100 into NaN value. membership duration is a newly added column to calculate the duration. After using the class above, the data info will look like,

```
profile.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 17000 entries, 0 to 16999
Data columns (total 5 columns):
 #   Column            Non-Null Count  Dtype
---  ------            --------------  -----
 0   gender            14825 non-null  object
 1   age               17000 non-null  int64
 2   id                17000 non-null  object
 3   became_member_on  17000 non-null  int64
 4   income            14825 non-null  float64
dtypes: float64(1), int64(2), object(2)
memory usage: 664.2+ KB
```

Before cleaning

```
profile_new.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 14825 entries, 1 to 16999
Data columns (total 8 columns):
 #   Column               Non-Null Count  Dtype
---  ------               --------------  -----
 0   age                  14820 non-null  float64
 1   became_member_on     14825 non-null  datetime64[ns]
 2   income               14825 non-null  float64
 3   customer_id          14825 non-null  int64
 4   membership_duration  14825 non-null  int64
 5   gender_F             14825 non-null  uint8
 6   gender_M             14825 non-null  uint8
 7   gender_O             14825 non-null  uint8
dtypes: datetime64[ns](1), float64(2), int64(2), uint8(3)
memory usage: 738.4 KB
```

After cleaning

As we can see, gender columns have been modified to three different columns as well as the id. After dropping the null values, the total number of rows have been reduced from 17,000 to 14825.

For transcript.json,

```
def cleanTranscript(transcript, offer_id_encoded, cust_id_encoded):

    transcript_new = transcript.copy()

    # map customer id
    transcript_new['customer_id'] = transcript_new['person'].map(cust_id_encoded)

    # sort data
    transcript_new = transcript_new.set_index(['customer_id', 'time']).sort_values(by=['customer_id', 'time'])
.reset_index()

    # split 'value' collumn
    transcript_new = pd.concat([transcript_new.drop(['value'], axis=1), transcript_new['value'].apply(pd.Series)]
, axis=1)

    # fill Na value
    transcript_new['amount'] = transcript_new['amount'].fillna(0)
    transcript_new['reward'] = transcript_new['reward'].fillna(0)

    # split 'event' collumn
    transcript_new = pd.get_dummies(transcript_new, columns = ['event'])

    # Move the non NaN values in 'offer_id' to 'offer id'
    transcript_new['offer id'] = transcript_new['offer id'].fillna(transcript_new['offer_id'])
    transcript_new = transcript_new.drop(columns='offer_id', axis=1)
   # split 'value' collumn
   transcript_new = pd.concat([transcript_new.drop(['value'], axis=1), transcript_new['value'].apply(pd.Series)]
, axis=1)

   # fill Na value
   transcript_new['amount'] = transcript_new['amount'].fillna(0)
   transcript_new['reward'] = transcript_new['reward'].fillna(0)

   # split 'event' collumn
   transcript_new = pd.get_dummies(transcript_new, columns = ['event'])

   # Move the non NaN values in 'offer_id' to 'offer id'
   transcript_new['offer id'] = transcript_new['offer id'].fillna(transcript_new['offer_id'])
   transcript_new = transcript_new.drop(columns='offer_id', axis=1)

   # map the offer id to the mapped value
   transcript_new['offer id'] = transcript_new['offer id'].map(offer_id_encoded)

   transcript_new = transcript_new.drop('person', axis=1)

   # Change the time from hours to days
   transcript_new['time'] = transcript_new['time'] / 24

   return transcript_new

transcript_new = cleanTranscript(transcript, offer_id_encoded, cust_id_encoded)
transcript_new
```

This is the function to clean to transcript.json, input will be the ‘transcriptmember’ which is a row dataframe read from the original data set. Second input will be cust_id_encoded and offer_id_encoded which is an encoded map of the custer_id and offer_id. We will fill out all the Na values in amount and reward to 0 and map the offer_id to mapped value. Finally change the time from hours to days since the value is too huge. After all these processes, the info will look like

```
transcript.info()

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

Before cleaning

```
transcript_new.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 306534 entries, 0 to 306533
Data columns (total 9 columns):
 #   Column                 Non-Null Count   Dtype
---  ------                 --------------   -----
 0   customer_id            272762 non-null  float64
 1   time                   306534 non-null  float64
 2   amount                 306534 non-null  float64
 3   offer id               167581 non-null  float64
 4   reward                 306534 non-null  float64
 5   event_offer completed  306534 non-null  uint8
 6   event_offer received   306534 non-null  uint8
 7   event_offer viewed     306534 non-null  uint8
 8   event_transaction      306534 non-null  uint8
dtypes: float64(5), uint8(4)
memory usage: 12.9 MB
```

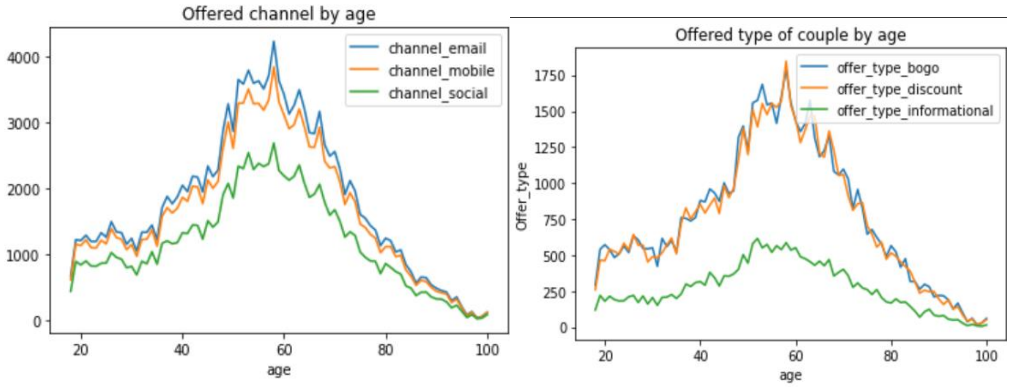
After cleaning

Exploratory Data Analysis

Spending trend(age) by offered type of coupon & channel.

```
spending_trend_age_channel = df[['age','channel_email','channel_mobile','channel_social']].groupby('age', as_index=False).sum()
spending_trend_age_channel.plot(x="age",title="Offered channel by age")
plt.legend(facecolor='white', fontsize=10)
```

```
spending_trend_age = df[['age','offer_type_bogo','offer_type_discount','offer_type_informational']].groupby('age', as_index=False).sum
()
spending_trend_age.plot(x="age",title="Offered type of couple by age")
plt.ylabel('Offer_type',fontsize=10)
plt.legend(facecolor='white', fontsize=10)
```



By using. plot function,

1. We can say that most of the members get their coupons through the email and mobile. However, the number of the members who got notice via email is slightly higher then the members who got notification through the mobile.

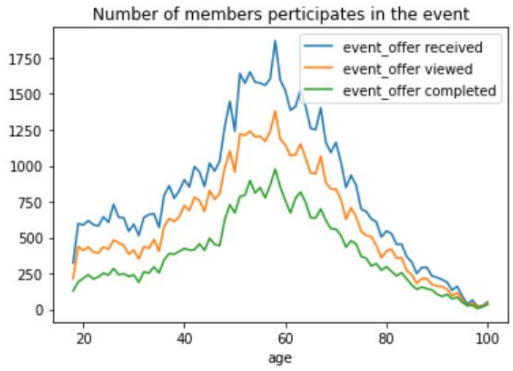
2. The type of the coupons are usually bogo or discount. Members in their early 20s, early 30 get buy one get one coupon.

3. Starbucks is not promoting enough coupons by informational offer and channel through sns.

Participant rate of event

```
received = df[['age','event_offer received','event_offer viewed','event_offer completed']].groupby('age', as_index=False).sum()
received.plot(x="age",title="Number of members perticipates in the event")
```

```
AxesSubplot:title={'center':'Number of members perticipates in the event'}, xlabel='age'>
```



Figure

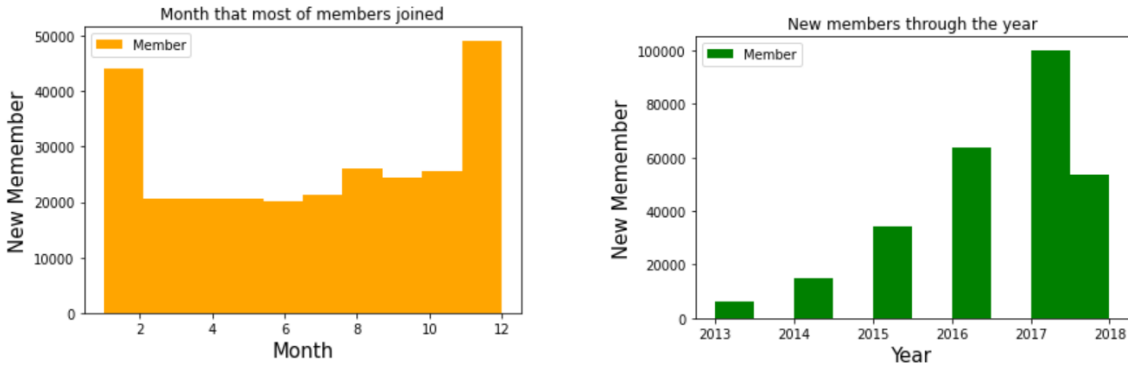
Even though Starbucks offers various promotions, only half of the users actually use the offer to get the drinks.

Finding out which year and month have joined to become a member.

```
#using lambda function to sort the 'become_member_on' into month and year
plt.clf()
df['became_member_on'].map(lambda d: d.month).plot(kind='hist',color='orange',title='Month that most of members joined',
label="Member")
plt.xlabel('Month',fontsize=15)
plt.ylabel('New Mememberr',fontsize=15)
plt.legend(facecolor='white', fontsize=10)
plt.show()
```

```
plt.clf()
df['became_member_on'].map(lambda d: d.year).plot(kind='hist',color='g',title='New members through the year',label="Member")
plt.xlabel('Year',fontsize=15)
plt.ylabel('New Mememberr',fontsize=15)
plt.legend(facecolor='white', fontsize=10)
plt.show()
```

First, we used lambda to get month from became_member_on and displayed in the plot by month and the number of new members. Same method was used when it comes to the year graph. The result is,



Figure

1. We can say that December and January gained almost double the number of members from 2013 - 2018.
2. Starbucks is getting more and more members through the year.
3. Regarding this dataset being realized in the middle of 2018, we can assume that Starbucsk will get more new members at the end of 2018.
4. Nevertheless, the number of new members started to decrease from 2016. It seems like starbucks needs to improve their promotions to get more new members.

Feature Engineering

1) After merging three datasets into one, there appears to have existed many NaN values in the data set. The first thing we have to do is to handle all NaN values that exist in the data set. We can see that all of the attributes related to the customer and offer are null. Since we only care about the transcripts with offer and customer information, we can drop all the rows without offer id and age values using dropna() function.

```
# check null
df.isnull().sum()
```

```
customer_id               33772
time                          0
amount                        0
offer id                 138953
amount_rewarded               0
event_offer completed         0
event_offer received          0
event_offer viewed            0
event_transaction             0
offer_reward             138953
difficulty               138953
duration                 138953
channel_email            138953
channel_mobile           138953
channel_social           138953
channel_web              138953
offer_type_bogo          138953
offer_type_discount      138953
offer_type_informational 138953
age                       33870
became_member_on          33772
income                    33772
membership_duration       33772
gender_F                  33772
gender_M                  33772
gender_O                  33772
dtype: int64
```

Before dropping NaN value

```
# remove null value
new_df = df.dropna(subset=['offer id'])
new_df = new_df.dropna(subset=['age'])
new_df.isnull().sum()
```

```
customer_id               0
time                      0
amount                    0
offer id                  0
amount_rewarded           0
event_offer completed     0
event_offer received      0
event_offer viewed        0
event_transaction         0
offer_reward              0
difficulty                0
duration                  0
channel_email             0
channel_mobile            0
channel_social            0
channel_web               0
offer_type_bogo           0
offer_type_discount       0
offer_type_informational  0
age                       0
became_member_on          0
income                    0
membership_duration       0
gender_F                  0
gender_M                  0
gender_O                  0
dtype: int64
```

After dropping NaN values.

2) Next, using drop_duplicates(), we drop duplicate rows that exist in the data.

```
new_df = new_df.drop('became_member_on', axis=1)

# drop duplicate rows
new_df.drop_duplicates()
```

3) From the target feature, as stated at the beginning of the assignment: **All of the offers which are viewed and completed are considered as success offers**. So we have to create a binary column offer_succeed from event_offers column in order to retrieve the results whether the customer.

```
# make new collumns which is the target for train model.
for i, item in new_df.iterrows():
    offer_succeed = 0
    if item['event_offer completed'] == 1 or item['event_offer viewed'] == 1:
        offer_succeed = 1
    else:
        offer_succeed = 0
    new_df.loc[i, 'offer_succeed'] = offer_succeed
new_df['offer_succeed'].value_counts()
```

```
1.0    82276
0.0    66478
Name: offer_succeed, dtype: int64
```

4) After modifying the data, we can see that there are 25 columns with the same amount of rows and fully converted to number type attributes. The final step we do is to select the list of features needed to train the model.

```
# select features
X = new_df.drop(['customer_id', 'amount', 'event_offer completed', 'event_offer received',
                 'event_offer viewed', 'event_transaction', 'offer_succeed'], axis=1)
X
# Export the data to csv for later use
X.to_csv("data/features.csv")
```

```
# select target
Y = new_df['offer_succeed']
Y
# Export the data to csv for later use
X.to_csv("data/target.csv")
```

Data Modelling

As stated from the beginning, we have to build a binary classification model to predict whether the offer is effective or not. Our team has selected four models for our project. The first two models are quite basic one which are LogisticRegression and RandomForest. The two remainings, AdaBoostClassifier and LGBMClassifier, are more advanced ones which are considered more effective. Before diving in the modelling and training process, first we import all the libraries needed.

```
# Modeling
from sklearn.metrics import classification_report, f1_score, accuracy_score
from lightgbm import LGBMClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import AdaBoostClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import KFold, StratifiedKFold, cross_val_score, train_test_split, RandomizedSearchCV
from sklearn.preprocessing import RobustScaler
from sklearn.pipeline import make_pipeline
from sklearn.model_selection import GridSearchCV

# Export model
import pickle

#List to store the model point
model_score = []
```

1) LogisticRegression

The LogisticRegression will construct a linear decision boundary to decide effective and non-effective offers. When having more than one explanatory variable, LogisticRegression is often used to calculate odds ratio. Excluding the response variable is binomial, the procedure is quite like multiple LinearRegression calculations. The result is the impact of each variable on the odds ratio of the observed event of interest. The main advantage is to avoid confounding effects by analyzing the association of all variables together. In this part we will use LogisticRegression method and perform parameter tuning with GridSearchCV to fit the training dataset to the model

```
%%time

model =  LogisticRegression()

parameters = {
    'C': [ 1, 10, 20, 30],
    'max_iter': [1000, 4000, 10000]
}

log_reg = GridSearchCV(model, parameters, refit=True)
log_reg.fit(X_train, y_train)

print('Best Score: ', log_reg.best_score_*100, '\nBest Parameters: ', log_reg.best_params_)
```

```
Best Score:  63.888261668265464
Best Parameters:  {'C': 10, 'max_iter': 1000}
CPU times: user 12min 34s, sys: 6min 43s, total: 19min 18s
Wall time: 2min 33s
```

The C value is Inverse of regularization strength, given how Scikit cites it as beings: $C = 1 / \lambda$. The relationship would be the smaller value denotes stronger regularization, and the C value that is best fit for this model is 1. Also, the maximum iteration that the model needs to fit the train data is 1000.

2) RandomForestClassifier

Random forest is a supervised learning algorithm. The "forest" it builds, is an ensemble of decision trees, commonly trained with the “bagging” method - a combination of learning models

increases the overall result. The RandomForestClassifier uses several decision trees and trains each tree individually with a random fraction of the data. One big advantage of random forest is that it can be used for both classification and regression problems, which form the majority of current machine learning systems. We will also perform parameter tuning in max_feature an n_estimator parameter and fit the model with train data.

```
%%time

model =  RandomForestClassifier()

parameters = {
    'max_features': ['sqrt', 'log2'],
    'n_estimators': [10, 100, 1000]
}

rf_clf= GridSearchCV(model, parameters, refit=True)
rf_clf.fit(X_train, y_train)

print('Best Score: ', rf_clf.best_score_*100, '\nBest Parameters: ', rf_clf.best_params_)
```

```
Best Score:  78.43163969171552
Best Parameters:  {'max_features': 'sqrt', 'n_estimators': 1000}
CPU times: user 18min 42s, sys: 5.72 s, total: 18min 48s
Wall time: 18min 48s
```

The max_features parameter representing the maximum number of features Random Forest is allowed to try in an individual tree. The *sqrt* option will take square root of the total number of features in the individual run. For example, if the total number of variables is 100, we can only take 10 of them in an individual tree. *log2* is another similar type of option for max_features. The best fit for this model is *sqrt*. n_estimator is the number of trees you want to build before taking the maximum voting or averages of predictions. Higher number of trees give you better performance but makes your code slower, and 1000 is best fitted for this model.

3) AdaBoostClassifier

The AdaBoostClassifier is a combination of multiple classifiers in order to achieve higher accuracy score. The methodology of AdaboostClassifier is to set the weight of the classifier and train it iteratively. The most important parameter of AdaBoostClassifier is n_estimators - number of weak learner and learning_rate - weight of weak learner. We will use GridSearchCV to perform parameter tuning on these parameters and evaluate the model performance.

```
%%time

model = AdaBoostClassifier()

parameters = {
    'learning_rate': [0.001, 0.01, 0.02, 0.1, 0.2, 1.0],
    'n_estimators': [10, 50, 100, 200]
}

ada_clf = GridSearchCV(model, parameters, refit=True)
ada_clf.fit(X_train, y_train)

print('Best Score: ', ada_clf.best_score_*100, '\nBest Parameters: ', ada_clf.best_params_)
```

```
Best Score:  92.68253872533427
Best Parameters:  {'learning_rate': 1.0, 'n_estimators': 100}
CPU times: user 10min 36s, sys: 312 ms, total: 10min 37s
Wall time: 10min 37s
```

The parameter learning_rate is the contribution of each model to the weights and defaults to 1. Reducing the learning rate will mean the weights will be increased or decreased to a small degree, forcing the model train slower (but sometimes resulting in better performance scores). n_estimators is the number of models to iteratively train. The best result recorded in this model consists of learning_rate 1.0 and n_estimators 100.

4) LightGBMClassifier

LightGBMClassifier is a gradient boosting framework that uses tree-based learning algorithms. It is designed to be distributed and efficient as compared to other boosting algorithms. We will train the model to fit with our training data.

Figure

num_leaves is one of the most important parameters that controls the **complexity** of the model. With it, you set the maximum number of leaves each weak learner has. Large num_leaves increase accuracy on the training set and the chance of getting hurt by overfitting. max_depth parameter controls max depth of each trained tree and will have an impact on the best value for the num_leaves parameter, model performance, and training time. Pay attention if you use a large value of max_depth, your model will likely be **overfit** to the train set. boosting_type parameter depends on the processing unit type, the number of objects in the training dataset and the selected learning mode. min_data_in_leaf is a very important parameter to prevent over-fitting in a leaf-wise tree. Its optimal value depends on the number of training samples and num_leaves. Setting it to a large value can avoid growing too deep a tree but may cause under-fitting. The best parameter sets for this model is boosting_type: gbdt, max_depth: 5, min_data_in_leaf: 20, num_leaves: 6.

Result

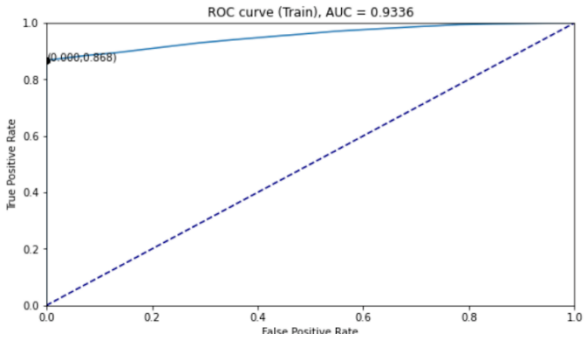
After we got information from EDA (Exploratory data analysis), we conducted to learn and evaluate many types of methods to fit the solving solution problem. We decided to use 4 types of model. Finally, the Model gives the following results:

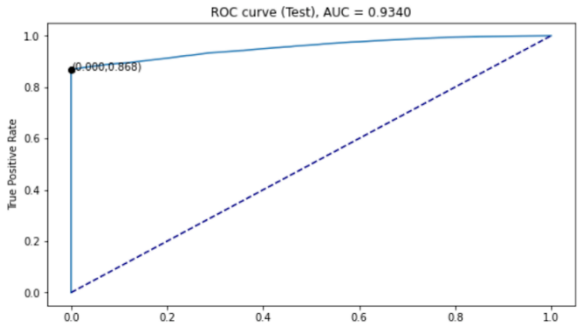
	Model	Accuracy Score	F1 Score
0	LogisticRegression	0.662230	0.643438
1	RandomForestClassifier	0.772109	0.788178
2	AdaBoostClassifier	0.927364	0.929598
3	LGBMClassifier	0.927364	0.929598

From the table, we can see that AdaBoostClassifier and LGBMClassifier have approximately the same accuracy score and F1 score. However, the LGBMClassifier has a faster training time, so we can conclude that the best model is the LGBMClassifier

LightGBM Model AUC - ROC Curve

AUC - ROC is a method of calculating the performance of a classification model according to different classification thresholds. Assuming that with a binary classification problem (2 classes) using logistic regression, choosing classification threshold [0..1] will affect the classification ability of the model and it is necessary to calculate the impact level of the threshold. AUC stands for Area Under The Curve and ROC stands for Receiver Operating Characteristic. The ROC is a curve representing probability and the AUC represents the classification of the model. AUC-ROC is also known as AUROC (Area Under The Receiver Operating Characteristics). The meaning of AUROC can be interpreted as follows: It is the probability that a randomly sampled positive sample will be ranked higher than a randomly drawn negative sample. The higher the AUC, the more accurate the model is in classifying the classes.





As noted above, the closer the AUC number is to 1, the more accurate the model will classify. The closer the AUC is to 0.5, the worse classification performance will be. If it is closed to 0, the model will reverse classify the result (classify positive as negative and vice versa). From the graph, the AUC number is approximately 0.93 of both Train data and Test data, which means our model has a high performance.

API

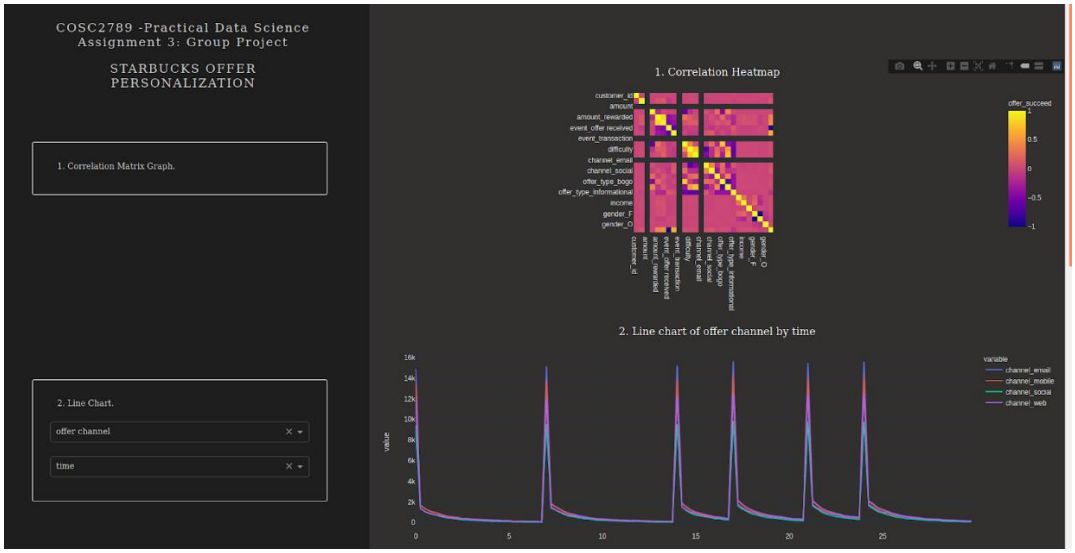
There are 2 type of API:

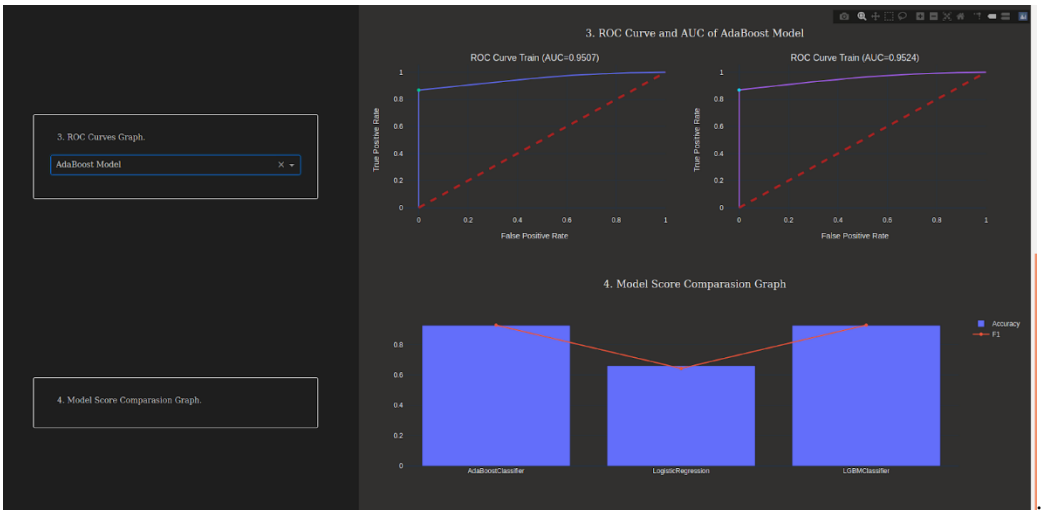
HTTP methods	Route	Description
GET	/api/evaluate/<model_name>	Return a model score
GET	/api/predict/<model_name>	Return a list of result predicted by model
POST	/api/predict_offer_effective	Return the result predict whether offer effective or not against specific customer

Visualization Dashboard

We use Dash library to build our dashboard with 4 plots which are:

- Correlation Matrix
- Using line chart to statistically the columns for each different offer group
- AUC and ROC curve
- Model score comparison graph





Discussion

In general, we have built a model with high accuracy and this project has illustrated how Starbucks use their data science to improve their business. However, several think we can improve our model. Firstly, in the feature selection process, we only use features from the data set. We can make some features from the given dataset to increase the accuracy of the model. Secondly, the model we use is a binary classification which is not a complicated one. We can divide the offer by its offer types and make predictions about the impact of the offer type to each customer which uses a multi classification model or clustering to have a deeper understanding about the customer demographic and gain more insight from the data.

Conclusion

In conclusion, from the data of customer behavior of Starbucks application, we have applied the process of data science to draw some insights and build a predictive model to evaluate the effectiveness of an offer based on the customer profile. The first part of this process is data cleaning which is one of the most challenging parts in this project, because the raw data is in JSON type with dictionary structure. Thus, a tremendous amount of work has to be done when dealing with data such as one hot encoding categorical features, drop null value, merge the data

sets. After finished cleaning, in the EDA part we have analyzed the data to acquire some insights about the customer spending trend with different offer types and customer membership duration over time. Finally, we have performed parameter tuning with 4 different binary classification models and found out that the LightGBMClassifier is the most effective model with the accuracy and f1 score over 0.9. In the future, some improvement can be done with this project. From the information, we can make more features to increase the accuracy of the model, other high-performance models such as XGBoost or CatBoost could be considered and the project can be taken to a further step which is to predict the best offer type for a specific customer.

References

1. Albon, C. (2017, December 20). *Adaboost Classifier*. Chris Albon. https://chrisalbon.com/machine_learning/trees_and_forests/adaboost_classifier/
2. Bahmani, M. J. (2020, December 14). *Understanding LightGBM Parameters (and How to Tune Them)*. Neptune.Ai. https://neptune.ai/blog/lightgbm-parameters-guide
3. Donges, N. (2020, September 3). *A complete guide to the random forest algorithm*. Built In. https://builtin.com/data-science/random-forest-algorithm
4. Nature Editorial. (2018, October 30). *Why Jupyter is data scientists' computational notebook of choice*. Nature. https://www.nature.com/articles/d41586-018-07196-1
5. Pal, S. (2020, August 14). *Scikit-learn Tutorial: Machine Learning in Python*. Dataquest. https://www.dataquest.io/blog/sci-kit-learn-tutorial/
6. Plotly. (2019, August 7). *Introducing Dash - Plotly*. Medium. https://medium.com/plotly/introducing-dash-5ecf7191b503
7. Sperandei, S. (2014). Understanding logistic regression analysis. *Biochemia Medica*, 12–18. https://doi.org/10.11613/bm.2014.003
8. Srivastava, T. (2020, June 26). *Tuning the parameters of your Random Forest model*. Analytics Vidhya. https://www.analyticsvidhya.com/blog/2015/06/tuning-random-forest-model/

9. *Welcome to LightGBM's documentation! — LightGBM 3.1.1.99 documentation*. (n.d.). LightGBM. https://lightgbm.readthedocs.io/en/latest/
10. Business Insider. 2021. *Starbucks Is Rolling Out A New System To Convince You To Buy Exactly Want It Wants You To Buy*. https://www.businessinsider.com/starbucks-develops-personalization-system-2016-12
11. Google Developers. 2021. *Classification: ROC Curve And AUC | Machine Learning Crash Course.* https://developers.google.com/machine-learning/crash-course/classification/roc-and-auc
12. Acutecaretesting.org. 2021. *ROC Curves – What Are They And How Are They Used?.*
13. https://acutecaretesting.org/en/articles/roc-curves-what-are-they-and-how-are-they-used

14. Medium. 2021. *Understanding AUC - ROC Curve.* *https://towardsdatascience.com/understanding-auc-roc-curve-68b2303cc9c5*