1- Definition

Project Overview

Domain Background

Starbucks Corporation is an American multinational chain of coffeehouses and roastery reserves headquartered in Seattle, Washington. As the world's largest coffeehouse chain, Starbucks is seen to be the main representation of the United States' second wave of coffee culture, As of early 2020, the company operates over 30,000 locations worldwide in more than 70 countries. Starbucks locations serve hot and cold drinks.**Starbucks** offer free application to provide best in class services to its customers which in return leverage its business , from this services Starbucks offer its clients offers to attract them to spend specific money or do specific action to redeem their price.

Due to that Data Analysis is needed to know customers preferences and provide them personalized experience and provide them with offers which they are interested in .

Problem Statement

Starbucks want to provide its customers personalized offers that they are most probably to take action to redeem them , we can do that through analysis of data recorded earlier to predict every customer own preferences

Datasets and Inputs

3 Dataset are provided , Details in table Below

Portfolio dataset :	● reward: (numeric) money awarded for the amount spent ● channels: (list) web, email, mobile, social ● difficulty: (numeric) money required to be spent to receive reward ● duration: (numeric) time for offer to be open, in days ● offer_type: (string) bogo, discount, informational ○ There are 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 ● id: (string/hash)
Profile dataset	Information about rewards program users ● gender: (categorical) M, F, O, or null ● age: (numeric) missing value encoded as 118 ● id: (string/hash) ● became_member_on: (date) format YYYYMMDD ● income: (numeric)
Transcript dataset	Customers' transactions info. ● person: (string/hash) ● event: (string) offer received, offer viewed, transaction, offer completed ● value: (dictionary) different values depending on event type ● offer id: (string/hash) not associated with any "transaction" ● amount: (numeric) money spent in "transaction"

	● reward: (numeric) money gained from "offer completed" ● time: (numeric) hours after start of test

Evaluation Metrics

To Evaluate such machine learning algorithm I am going to depend on some measures (statistical measures):

Accuracy *: ratio of correctly predicted observation to the total observations.*

Accuracy = TP+TN/TP+FP+FN+TN

Precision *- ratio of correctly predicted positive observations to the total predicted positive observations.*

Precision = TP/TP+FP

Recall *(Sensitivity) - ratio of correctly predicted positive observations to the all observations in actual class*

Recall = TP/TP+FN

F1 score *- F1 Score is the weighted average of Precision and Recall.This score takes both false positives and false negatives into account.*

F1 Score = 2(Recall * Precision) / (Recall + Precision)*

Analysis

Data Exploration

Datasets dimensions

Portfolio	(10, 6)

Profile	(17000, 5)
Transcript	(306534, 4)

Datasets statistical data

```
transcript.describe()
```

	time
count	306534.000000
mean	366.382940
std	200.326314
min	0.000000
25%	186.000000
50%	408.000000
75%	528.000000
max	714.000000

```
profile.describe()
```

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

```
portfolio.describe()
```

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

Data Null / Nan Values count : Portfolio and transcript datasets have no null values while profile data set has 2175 null values in two features

```
profile.isna().sum()
```

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

```
portfolio.isna().sum()
```

```
reward        0
channels      0
difficulty    0
duration      0
offer_type    0
id            0
dtype: int64
```

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

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

Datasets info :

```
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
```

```
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
```

```
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
```

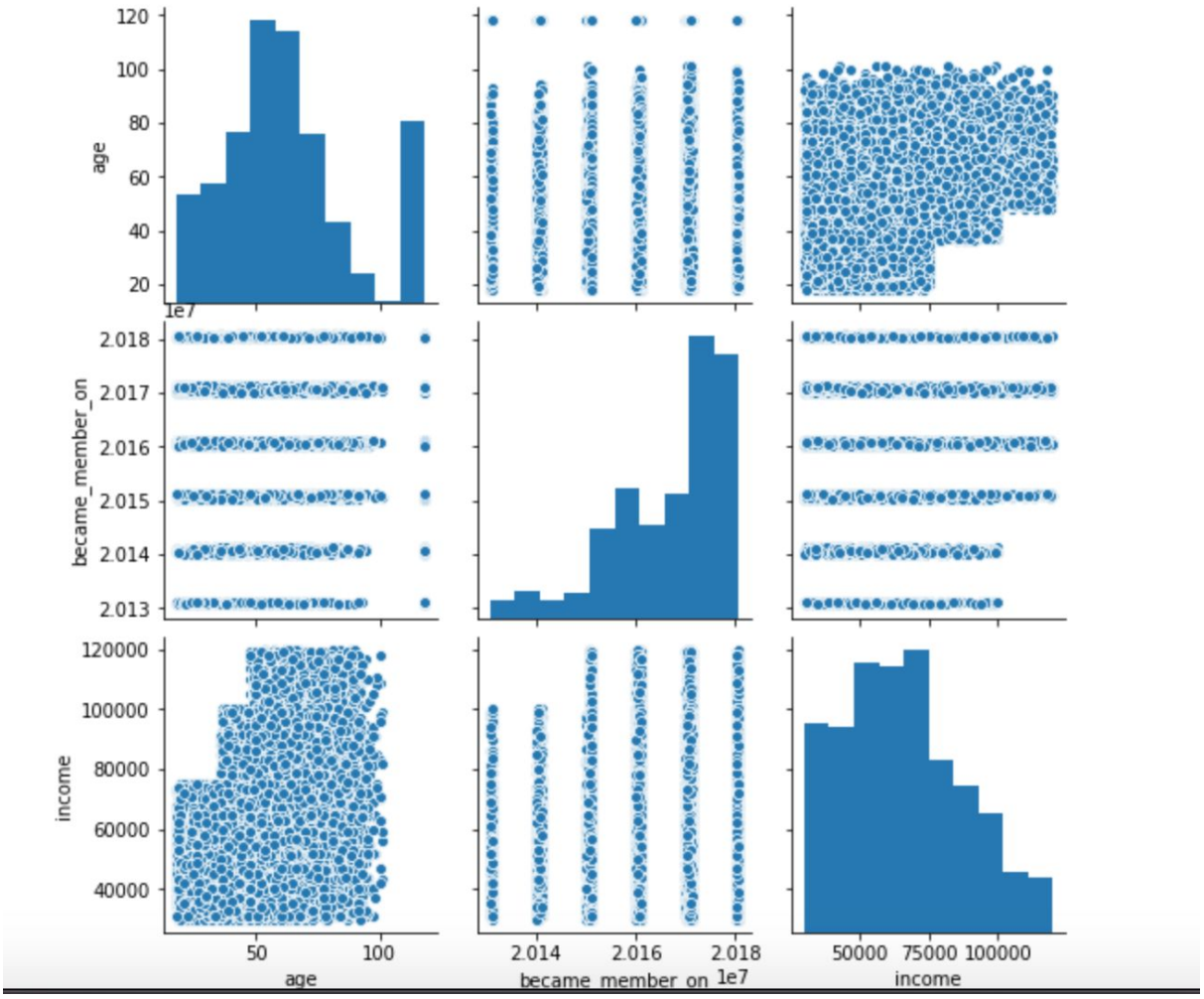
Training and testing data entered to models dimensions :

x_train	x_test	y_train	y_test
(3627, 4)	(1089, 4)	(3627,)	(1089,)

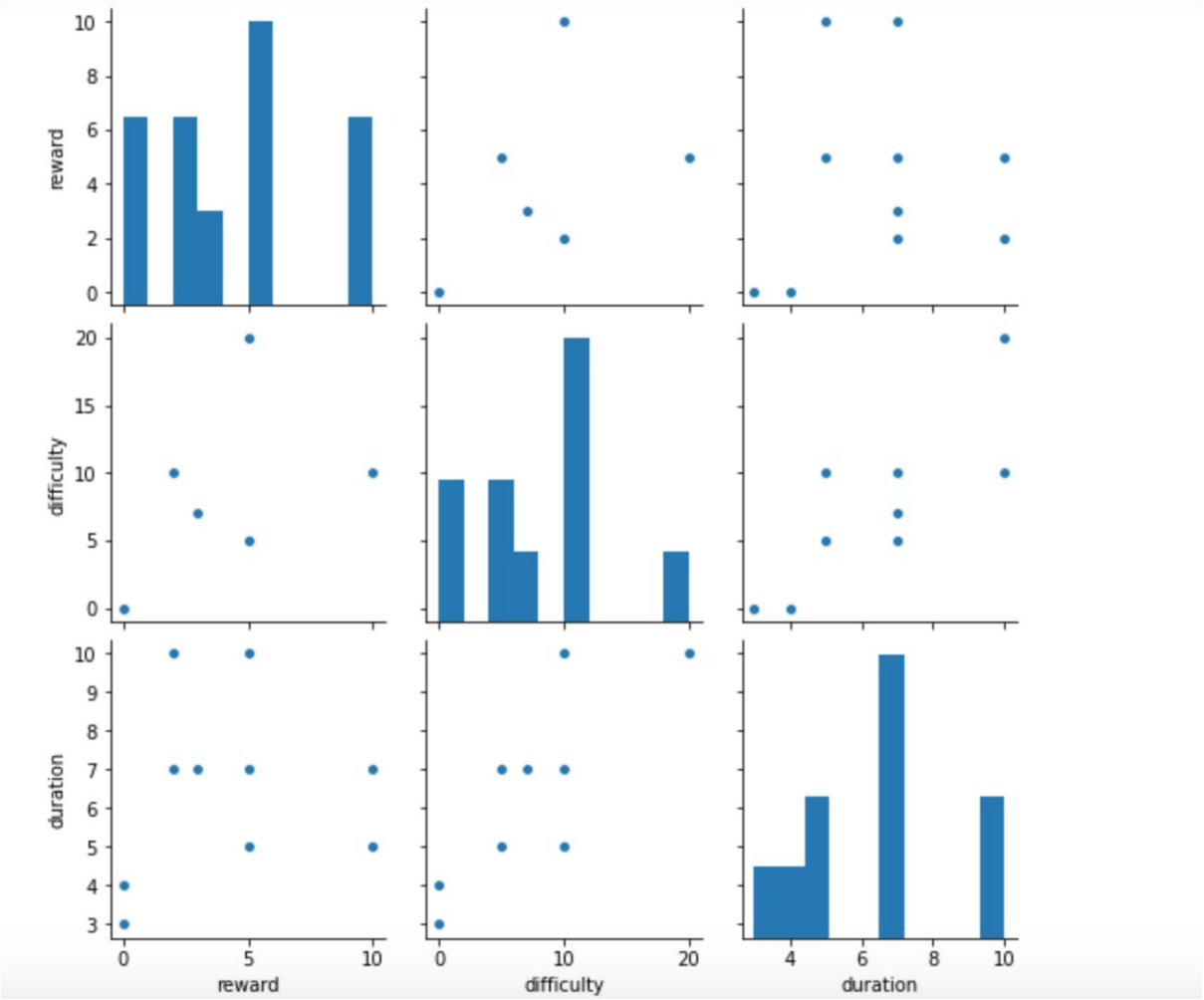
Exploratory Visualization

Pair Plots for original data sets :

1- Profile Dataset :

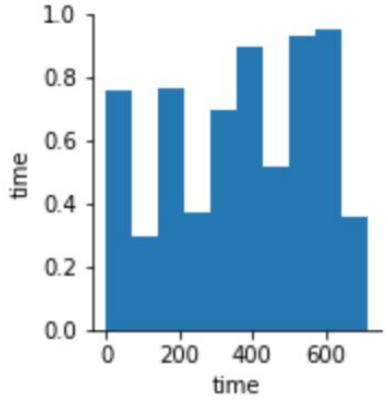


2- Portfolio Dataset

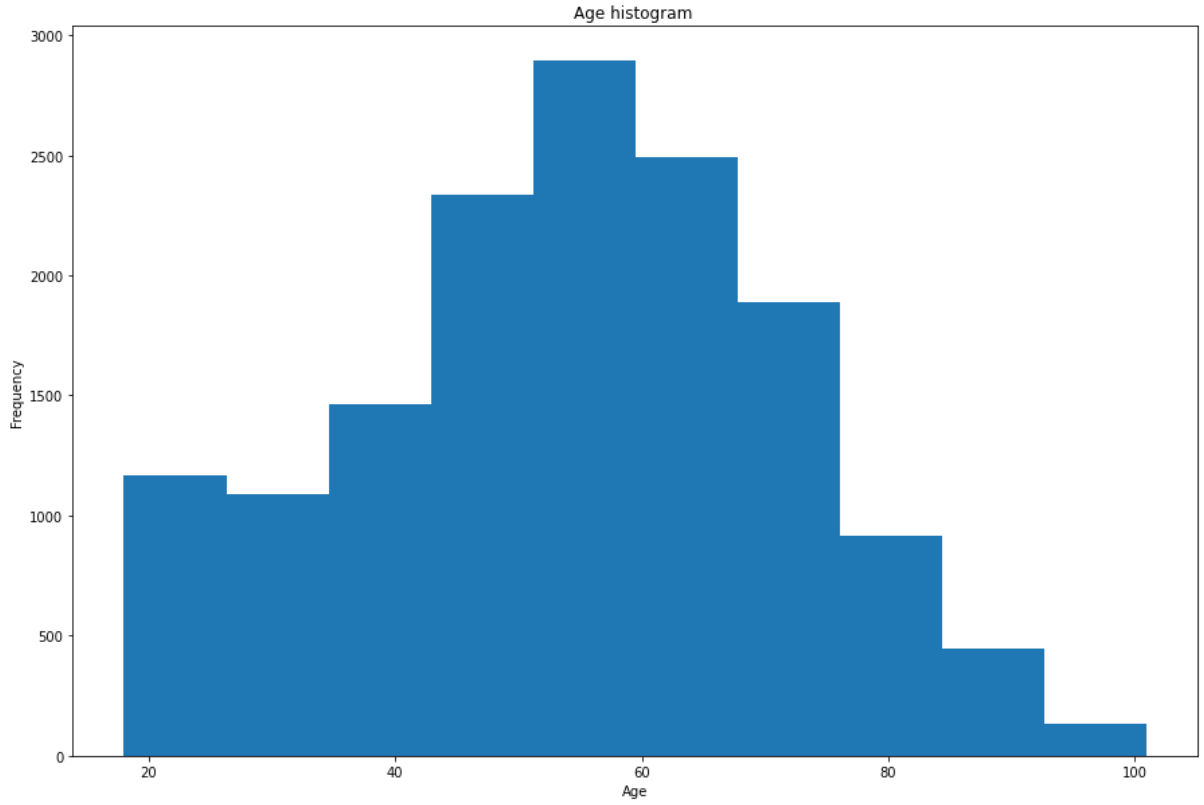


3-Transcript Dataset

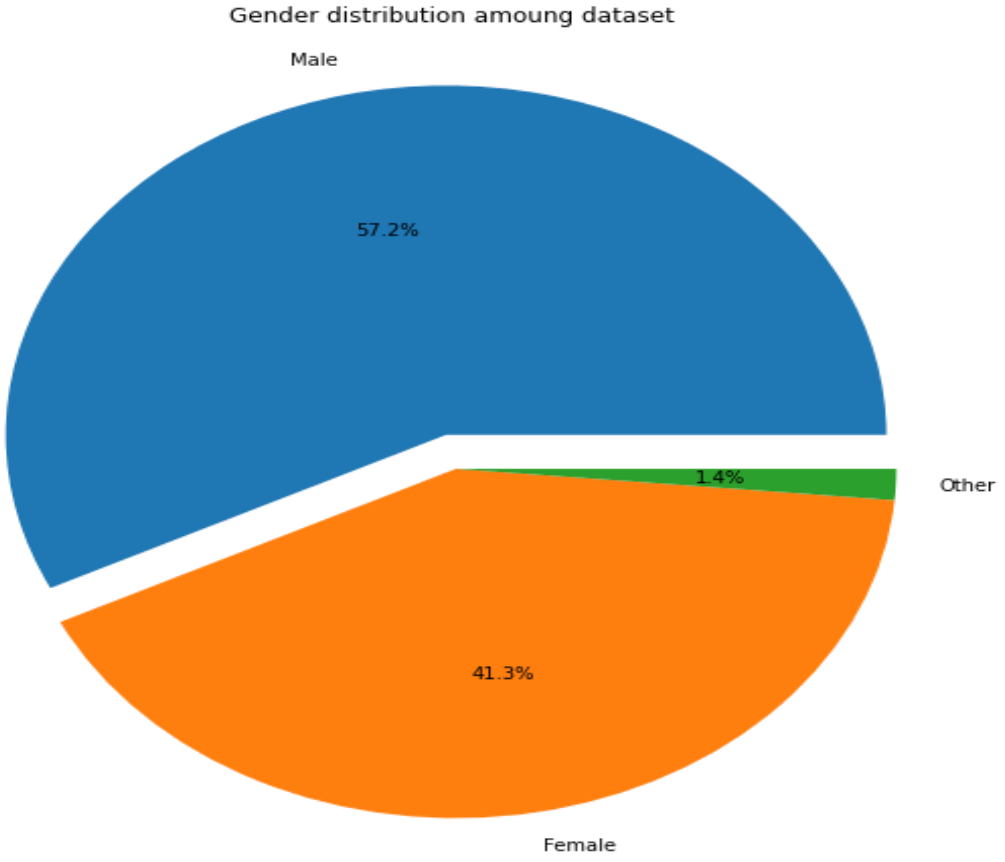
```
seaborn.pairplot(transcript) ;
# Plot pairwise relationships transcript dataset.
```



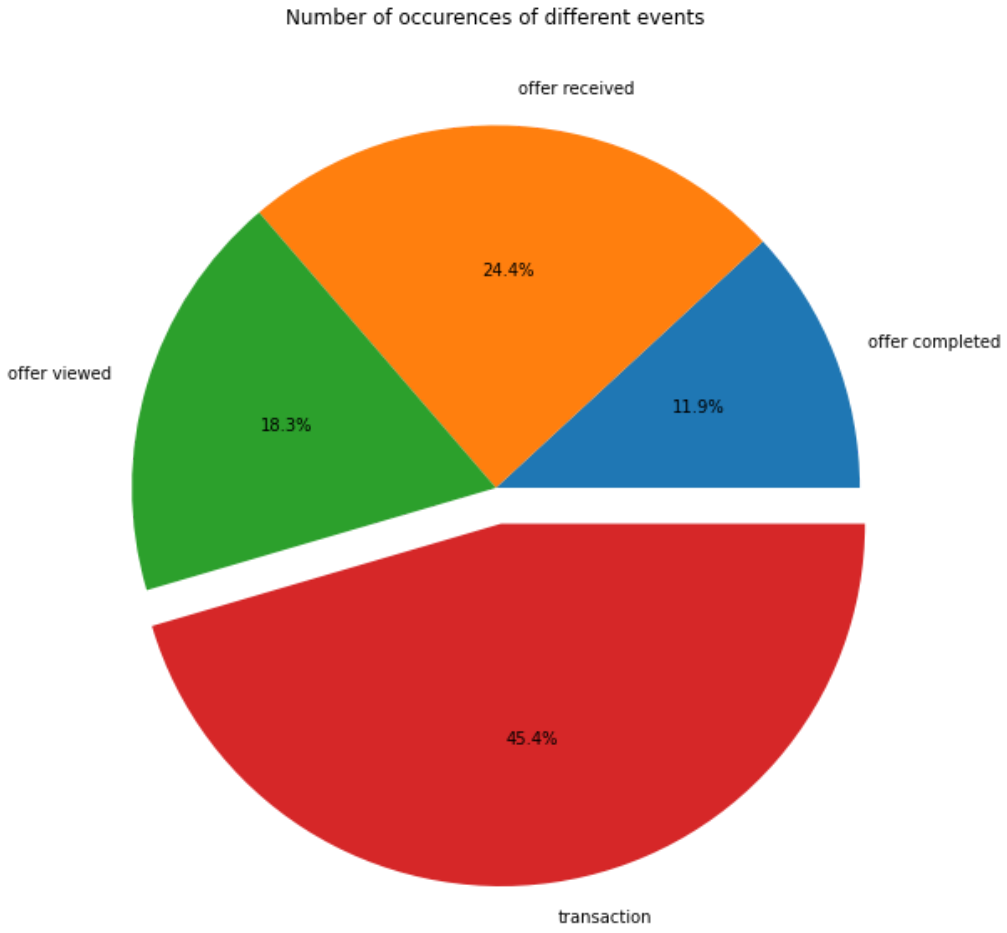
Users Age Histogram :



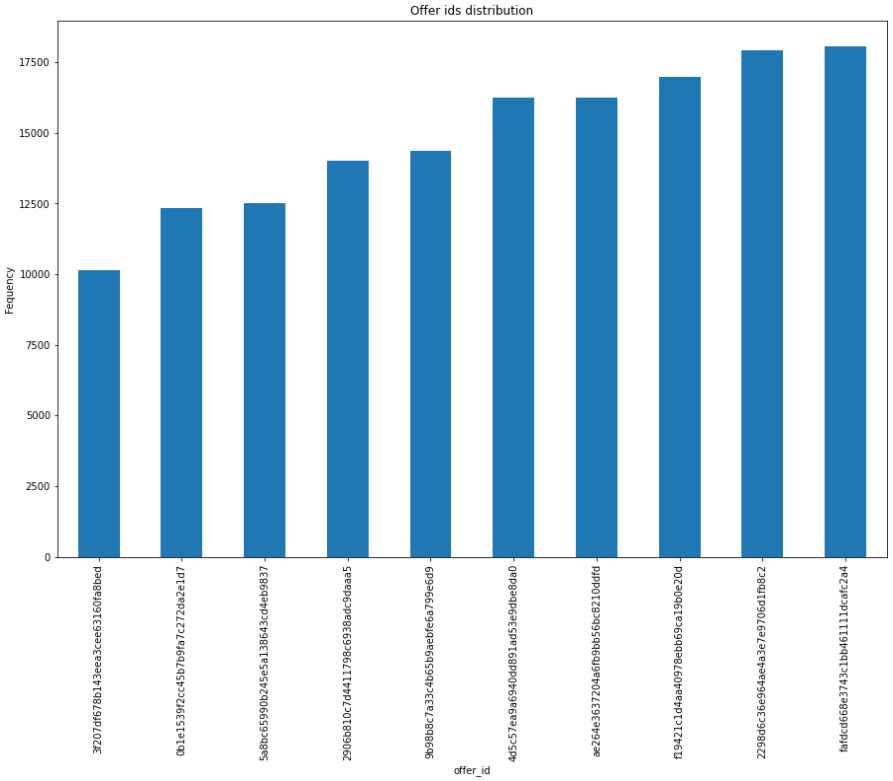
User’s gender distribution pie plot :



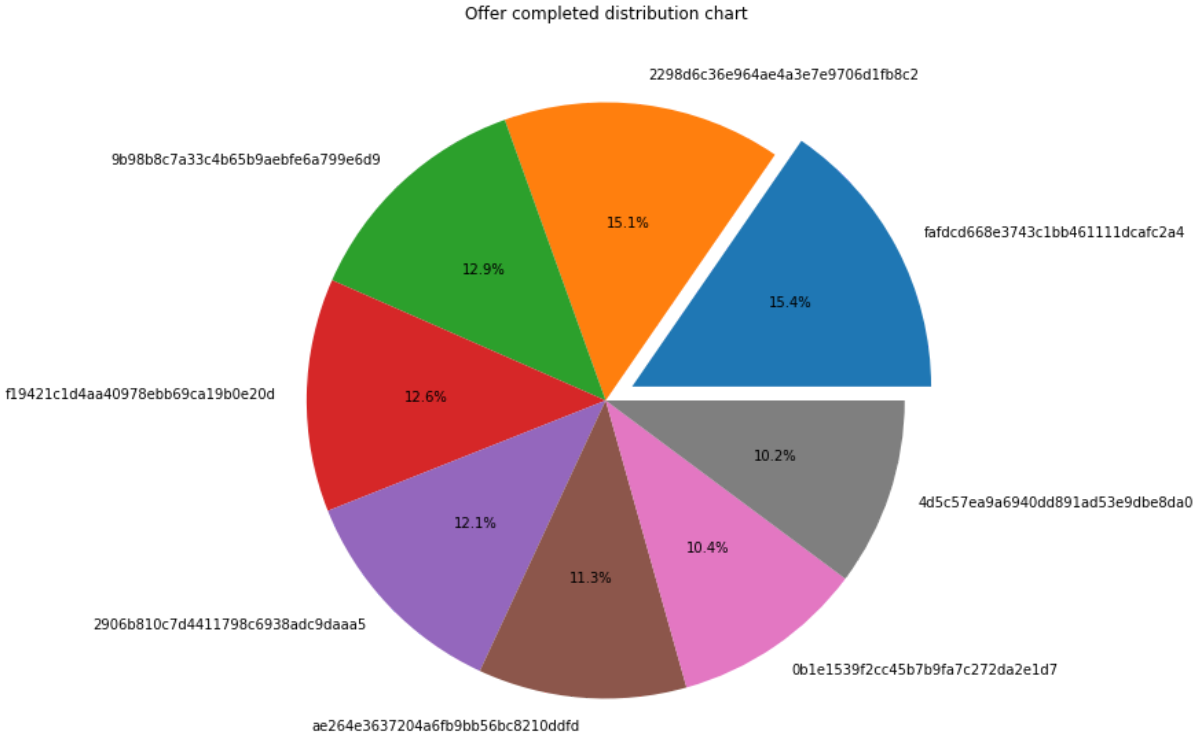
Number of occurrences of different users’ events pie chart :



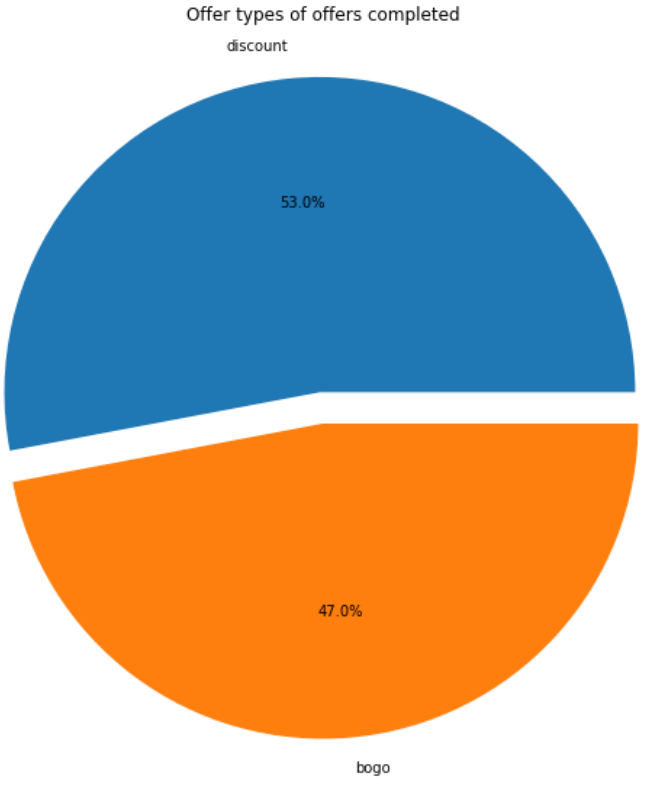
Number of Offers appearance in datasets labeled by offer ids plotted by bar plt :



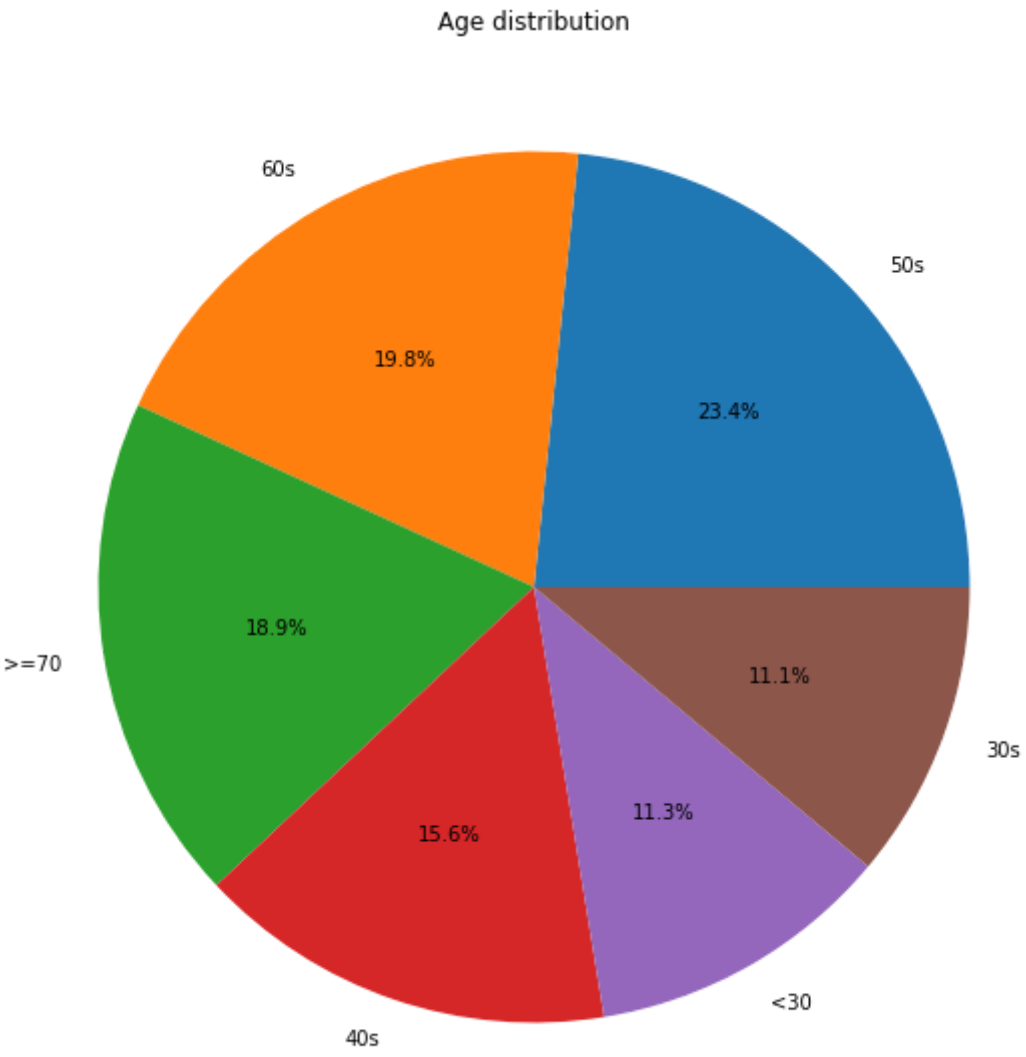
Offer completed distribution pie chart :



Offer types of offers completed pie chart :



Age Distribution in Datasets pie chart :



Algorithms and Techniques :

I will try the following models to predict customer best offer and best channel :

1- Built in Models

I will choose to go with Linaerlearner built in model to make prediction

```
# import LinearLearner
from sagemaker import LinearLearner

# specify an output path
prefix = 'starbucks_capstone'
output_path = 's3://{}/{}'.format(bucket, prefix)

# instantiate LinearLearner
linear_starbucks = LinearLearner(role=role,
                        train_instance_count=1,
                        train_instance_type='ml.c4.xlarge',
                        predictor_type='multiclass_classifier',
                        num_classes=11 ,
                        output_path=output_path,
                        sagemaker_session=sagemaker_session,
                        epochs=85)
```

I will configure it to be multiclass classifier to predict our 11 classes
Also i will choose to run it for 85 epochs
I used documentation from link below :
https://sagemaker.readthedocs.io/en/stable/algorithms/linear_learner.html

2- Sklearn Models

I also choose to go with three models from Sklearn library :
2.1 - Decision tree :
Since this is a classification problem the first solution would be to use simple decision tree these trees are simple but also effective in some model , I used it as trivial solution and benchmark for upcoming models

```
# Create sklearn estimator
from sagemaker.sklearn.estimator import SKLearn
Sklearn_tree = SKLearn(entry_point="train.py",
                    source_dir="Sklearn_tree",
                    role=role,
                    train_instance_count=1,
                    train_instance_type='ml.c4.xlarge')
```

```
This is not the latest supported version. If you would like to use version 0.23-1, please add framework_version=0.2
3-1 to your constructor.
```

I configured the model as shown below , I didn't need to configure any hyperparameters in sagemaker notebook as i went for the default setting i set in the train.py script

```
# SageMaker parameters, like the directories for training data and saving models; set automatically
# Do not need to change
parser.add_argument('--output-data-dir', type=str, default=os.environ['SM_OUTPUT_DATA_DIR'])
parser.add_argument('--model-dir', type=str, default=os.environ['SM_MODEL_DIR'])
parser.add_argument('--data-dir', type=str, default=os.environ['SM_CHANNEL_TRAIN'])
parser.add_argument('--max_depth', type=int, default=10)
# args holds all passed-in arguments
args = parser.parse_args()

# Read in csv training file
training_dir = args.data_dir
train_data = pd.read_csv(os.path.join(training_dir, "train.csv"), header=None, names=None)

# Labels are in the first column
train_y = train_data.iloc[:,0]
train_x = train_data.iloc[:,1:]

#define model
model = DecisionTreeClassifier( max_depth=args.max_depth)

# fit model
model.fit(train_x, train_y)
```

Documentation I used in decision tree classifier is in the below link :
https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html

2.2 - C-Support Vector Classification

The second model I used is C-Support Vector Classifier which is shape of support vector machines , Note that I could have used SVC with original data (before over and undersampling techniques) as SVC offer Unbalanced problems solutions but I went for oversampling and undersampling first to ensure that same data is trained by all networks to get firm understanding of which model to choose .

```
# create sklearn svm estimator
Sklearn_SVM = SKLearn(entry_point="train.py",
                      source_dir="Sklearn_SVM",
                      role=role,
                      train_instance_count=1,
                      train_instance_type='ml.c4.xlarge',
                      )
```

I also choose in SVC parameters to use rbf kernel because it has proven to me in my previous projects to be better than other kernel for example linear kernel in the scope of multi-class classification , I also made some online research about which kernel to choose I used link below in my research https://www.kdnuggets.com/2016/06/select-support-vector-machine-kernels.html

Finally I set C parameter to 0.5 , I tried 0.25 and 0.05 previously but 0.5 proved to be better. Below is my train.py script for svc.

```
# SageMaker parameters, like the directories for training data and saving models; set automatically
# Do not need to change
parser.add_argument('--output-data-dir', type=str, default=os.environ['SM_OUTPUT_DATA_DIR'])
parser.add_argument('--model-dir', type=str, default=os.environ['SM_MODEL_DIR'])
parser.add_argument('--data-dir', type=str, default=os.environ['SM_CHANNEL_TRAIN'])
parser.add_argument('--kernel', type=str, default='rbf')
# args holds all passed-in arguments
args = parser.parse_args()

# Read in csv training file
training_dir = args.data_dir
train_data = pd.read_csv(os.path.join(training_dir, "train.csv"), header=None, names=None)

# Labels are in the first column
train_y = train_data.iloc[:,0]
train_x = train_data.iloc[:,1:]

#define model
model = SVC(kernel = args.kernel , C = 0.5)

# fit model
model.fit(train_x, train_y)
```

2.3 K-Nearest Neighbour

I choose also to go with a KNN classifier (Neighbors-based classification is a type of *instance-based learning* or *non-generalizing learning*: it does not attempt to construct a general internal model, but simply stores instances of the training data. Classification is computed from a simple majority vote of the nearest neighbors of each point: a query point is assigned the data class which has the most

representatives within the nearest neighbors of the point. I used the following reference from sklearn website : https://scikit-learn.org/stable/modules/neighbors.html#neighbors)
with the number of neighbors equal to 11 which is the number of classes we have.

```
# create sklearn knn estimator
Sklearn_KNN = SKLearn(entry_point="train.py",
                      source_dir="Sklearn_KNN",
                      role=role,
                      train_instance_count=1,
                      train_instance_type='ml.c4.xlarge',
                      hyperparameters={'n_neighbors' :11 })
```

Below is code i used in train.py script

```
  # SageMaker parameters, like the directories for training data and saving models; set automatically
  # Do not need to change
  parser.add_argument('--output-data-dir', type=str, default=os.environ['SM_OUTPUT_DATA_DIR'])
  parser.add_argument('--model-dir', type=str, default=os.environ['SM_MODEL_DIR'])
  parser.add_argument('--data-dir', type=str, default=os.environ['SM_CHANNEL_TRAIN'])
  parser.add_argument('--n_neighbors', type=int, default=7)


  # args holds all passed-in arguments
  args = parser.parse_args()

  # Read in csv training file
  training_dir = args.data_dir
  train_data = pd.read_csv(os.path.join(training_dir, "train.csv"), header=None, names=None)

  # Labels are in the first column
  train_y = train_data.iloc[:,0]
  train_x = train_data.iloc[:,1:]

  #define model
  model = KNeighborsClassifier(n_neighbors = args.n_neighbors)
  # fit model
  model.fit(train_x, train_y)
```

3 - XGBoost Model

I also chose to go with XGBoost model that i was taught in the nanodegree lessons , as it has proven to be a good classifier , I use the configuration shown below in model creation

```
from sagemaker.amazon.amazon_estimator import get_image_uri
from sagemaker.predictor import csv_serializer
# As stated above, we use this utility method to construct the image name for the training container.
container = get_image_uri(sagemaker_session.boto_region_name, 'xgboost')

# Now that we know which container to use, we can construct the estimator object.
xgb = sagemaker.estimator.Estimator(container, # The image name of the training container
                                    role,      # The IAM role to use (our current role in this case)
                                    train_instance_count=1, # The number of instances to use for training
                                    train_instance_type='ml.m4.xlarge', # The type of instance to use for training
                                    output_path='s3://{}/{}/output'.format(sagemaker_session.default_bucket(), prefi
                                                                        # Where to save the output (the model artifa
                                    sagemaker_session=sagemaker_session) # The current SageMaker session
```

```
xgb.set_hyperparameters(max_depth=15,
                        eta=0.1,
                        gamma=4,
                        min_child_weight=6,
                        subsample=0.8,
                        objective='multi:softmax',
                        early_stopping_rounds=20,
                        num_round=300,
                        num_class = 11 )
```

I went for multi-class classification which has softmax as the model activation function , I went with 20 early stopping rounds to give the model some good availability of time before it stop training the model on data , I went with 300 rounds and a learning rate of 0.1 , I also set maximum depth to relatively small at 15.

Benchmark Model

As the benchmark result, we can extrapolate the current Conversion Rate of the offer received. Leaving out the informational offers, which have no real “conversion”, the CR on the viewed offers is **43% for BOGO, 56% for Discount** (37% and 42% on all the received offers).

Methodology

Data Preprocessing

Firstly I have three datasets given as json files : transcript , portfolio , profile
First : Profile Dataset Null values :

```
profile.shape #get the dimensions of the dataset
```

```
(17000, 5)
```

It has 17000 rows and 5 features , but when checking for null /Nan values I saw that there’s 2175 Nan values in gender and income columns

```
profile.isna().sum()  #Check the number of Nan values
```

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

```
Nan_gender_indexes =profile[profile.gender.isnull()]\
.index.tolist() # create a list with indexes of nan gender values
Nan_income_indexes =profile[profile.income.isnull()]\
.index.tolist()# create a list with indexes of nan income values
```

```
print (len (Nan_gender_indexes),len (Nan_income_indexes))
#print the length of Nan_gender and Nan_income lists
```

```
2175 2175
```

The next step is to get the intersection beteween the two lists to see if they have any enteries in common

Fri, Sep 4, 2020 3:27 AM

```
len (np.intersect1d(Nan_gender_indexes , Nan_income_indexes)) #print the number of indexes that appear
                                        #in both Nan_gender and Nan_income
```

```
2175
```

I checked if there’s any intersection between the rows that have loss in one feature (for example I check that if one row have gender loss does it also have income loss does they intersect) after that I found that all rows that have income loss also have gender loss which means I have 2175 rows of incomplete data.

```
profile.iloc[Nan_gender_indexes] #Show dataset with Nan gender
```

	gender	age	id	became_member_on	income
0	None	118	68be06ca386d4c31939f3a4f0e3dd783	20170212	NaN
2	None	118	38fe809add3b4fcf9315a9694bb96ff5	20180712	NaN
4	None	118	a03223e636434f42ac4c3df47e8bac43	20170804	NaN
6	None	118	8ec6ce2a7e7949b1bf142def7d0e0586	20170925	NaN
7	None	118	68617ca6246f4fbc85e91a2a49552598	20171002	NaN
...	...	...	...	...	...
16980	None	118	5c686d09ca4d475a8f750f2ba07e0440	20160901	NaN
16982	None	118	d9ca82f550ac4ee58b6299cf1e5c824a	20160415	NaN
16989	None	118	ca45ee1883624304bac1e4c8a114f045	20180305	NaN
16991	None	118	a9a20fa8b5504360beb4e7c8712f8306	20160116	NaN
16994	None	118	c02b10e8752c4d8e9b73f918558531f7	20151211	NaN

Also if found that of this rows have the same age value which is 118 which is very strange and unacceptable age , so i dropped all this rows as they don't have any valuable data .

Second Merging datasets :
At first I merged the transcript and profile datasets , to see actions done by each individual with that individual data such as his gender , age , income...etc to have more clear image
I also made a new feature (offer id) which represent the offer id value that was in the value column dictionaries

	id	event	value	time	gender	age	became_member_on	income	offer_id
0	78afa995795e4d85b5d9ceeca43f5fef	offer received	{'offer id': '9b98b8c7a33c4b65b9aebfe6a799e6d9'}	0	F	75	20170509	100000.0	9b98b8c7a33c4b65b9aebfe6a799e6d9
1	78afa995795e4d85b5d9ceeca43f5fef	offer viewed	{'offer id': '9b98b8c7a33c4b65b9aebfe6a799e6d9'}	6	F	75	20170509	100000.0	9b98b8c7a33c4b65b9aebfe6a799e6d9
2	78afa995795e4d85b5d9ceeca43f5fef	transaction	{'amount': 19.89}	132	F	75	20170509	100000.0	NaN
3	78afa995795e4d85b5d9ceeca43f5fef	offer completed	{'offer_id': '9b98b8c7a33c4b65b9aebfe6a799e6d9...	132	F	75	20170509	100000.0	9b98b8c7a33c4b65b9aebfe6a799e6d9
4	78afa995795e4d85b5d9ceeca43f5fef	transaction	{'amount': 17.78}	144	F	75	20170509	100000.0	NaN
...	...	...	...	...	...	...	...	...	...
272757	9fcbff4f8d7241faa4ab8a9d19c8a812	offer viewed	{'offer id': '3f207df678b143eea3cee63160fa8bed'}	504	M	47	20171013	94000.0	3f207df678b143eea3cee63160fa8bed
272758	9fcbff4f8d7241faa4ab8a9d19c8a812	offer received	{'offer id': '4d5c57ea9a6940dd891ad53e9dbe8da0'}	576	M	47	20171013	94000.0	4d5c57ea9a6940dd891ad53e9dbe8da0
272759	9fcbff4f8d7241faa4ab8a9d19c8a812	offer viewed	{'offer id': '4d5c57ea9a6940dd891ad53e9dbe8da0'}	576	M	47	20171013	94000.0	4d5c57ea9a6940dd891ad53e9dbe8da0
272760	3045af4e98794a04a5542d3eac939b1f	offer received	{'offer id': '4d5c57ea9a6940dd891ad53e9dbe8da0'}	576	F	58	20161020	78000.0	4d5c57ea9a6940dd891ad53e9dbe8da0
272761	3045af4e98794a04a5542d3eac939b1f	offer viewed	{'offer id': '4d5c57ea9a6940dd891ad53e9dbe8da0'}	576	F	58	20161020	78000.0	4d5c57ea9a6940dd891ad53e9dbe8da0

I then made some modifications on columns naming in portfolio dataset to make it more representative of what the dataset offer

```
# Rename columns
portfolio.rename(columns = {"id" : "offer_id" ,
                            "reward" : 'offer_reward' ,
                            "channels":"offer_channels" ,
                            "difficulty" : "offer_difficulty" ,
                            "duration" : "offer_duration"} , inplace =True)
```

```
portfolio
```

	offer_reward	offer_channels	offer_difficulty	offer_duration	offer_type	offer_id
0	10	[email, mobile, social]	10	7	bogo	ae264e3637204a6fb9bb56bc8210ddfd
1	10	[web, email, mobile, social]	10	5	bogo	4d5c57ea9a6940dd891ad53e9dbe8da0
2	0	[web, email, mobile]	0	4	informational	3f207df678b143eea3cee63160fa8bed
3	5	[web, email, mobile]	5	7	bogo	9b98b8c7a33c4b65b9aebfe6a799e6d9
4	5	[web, email]	20	10	discount	0b1e1539f2cc45b7b9fa7c272da2e1d7
5	3	[web, email, mobile, social]	7	7	discount	2298d6c36e964ae4a3e7e9706d1fb8c2
6	2	[web, email, mobile, social]	10	10	discount	fafdcd668e3743c1bb461111dcafc2a4
7	0	[email, mobile, social]	0	3	informational	5a8bc65990b245e5a138643cd4eb9837
8	5	[web, email, mobile, social]	5	5	bogo	f19421c1d4aa40978ebb69ca19b0e20d
9	2	[web, email, mobile]	10	7	discount	2906b810c7d4411798c6938adc9daaa5

The following step was merging the already two datasets merged dataframe with the portfolio dataset to have one complete dataset

```
# Create All_combined_datasets which is all three datasets
# merged together
All_combined_datasets = pd.merge(transcript_profile_df ,portfolio, on= "offer_id"  , how = "left")
```

Next I changed the became member on entries to be only the year of membership instead of the previous more details format

Old

became_member_on	income
20170509	100000.0
20170509	100000.0
20170509	100000.0
20170509	100000.0
20170509	100000.0

New

became_member_on	income
2017	100000.0
2017	100000.0
2017	100000.0
2017	100000.0
2017	100000.0

I also changed the ages column to be more general and I made six categories of ages show below

```
####
#Function take age series and return 6
#different ages categories instead of ages
####
def age (ages) :
    new_age = []
    for age in ages:
        if age < 30 :
            new_age.append("<30")
        elif 30<= age <=39 :
            new_age.append("30s")
        elif 40<= age <=49 :
            new_age.append("40s")
        elif 50<= age <=59 :
            new_age.append("50s")
        elif 60<= age <=69 :
            new_age.append("60s")
        else :
            new_age.append(">=70")
    return new_age
```

Then I made a new dataframe which represent the model dataframe where each single user has only one entry initially that dataframe had three columns which is :

Total Offers received	Total Offers viewed	Total Offers completed

Then i made more detailed columns that represent each offer status as single column

id	Total Offers recieved	Total Offers viewed	Total Offers completed	bogo received	discount received	informational received	bogo viewed	discount viewed	informational viewed	bogo completed	discount completed
0009655768c64bdeb2e877511632db8f	5	4.0	3.0	1.0	2.0	2.0	1.0	1.0	2.0	1.0	2.0
0011e0d4e6b944f998e987f904e8c1e5	5	5.0	3.0	1.0	2.0	2.0	1.0	2.0	2.0	1.0	2.0
0020c2b971eb4e9188eac86d93036a77	5	3.0	3.0	2.0	2.0	1.0	1.0	1.0	1.0	1.0	2.0
0020ccbbb6d84e358d3414a3ff76cffd	4	4.0	3.0	2.0	1.0	1.0	2.0	1.0	1.0	2.0	1.0
003d66b6608740288d6cc97a6903f4f0	5	4.0	3.0	NaN	3.0	2.0	NaN	2.0	2.0	NaN	3.0
...	...	...	...	...	...	...	...	...	...	...	...
fff3ba4757bd42088c044ca26d73817a	6	3.0	3.0	1.0	3.0	2.0	1.0	1.0	1.0	1.0	2.0
fff7576017104bcc8677a8d63322b5e1	5	4.0	3.0	3.0	2.0	NaN	2.0	2.0	NaN	1.0	2.0
fff8957ea8b240a6b5e634b6ee8eafcf	3	2.0	NaN	1.0	1.0	1.0	1.0	1.0	NaN	NaN	NaN
fffad4f4828548d1b5583907f2e9906b	4	4.0	3.0	3.0	NaN	1.0	3.0	NaN	1.0	3.0	NaN

Then i merged to the previous dataset the profile dataset to add columns of age , gender income and became member on columns .
The Next step I encoded the offer channels on the three merged datasets dataframe i created previously and created a new dataframe containing each user and his offer channels and offers completed through each offer channel

offer_channels	id	completed_channel_offer
1	0009655768c64bdeb2e877511632db8f	2
2	0009655768c64bdeb2e877511632db8f	1
1	0011e0d4e6b944f998e987f904e8c1e5	1
2	0011e0d4e6b944f998e987f904e8c1e5	1
3	0011e0d4e6b944f998e987f904e8c1e5	1
...	...	...
1	fffad4f4828548d1b5583907f2e9906b	2
2	fffad4f4828548d1b5583907f2e9906b	1
2	ffff82501cea40309d5fdd7edcca4a07	4
1	ffff82501cea40309d5fdd7edcca4a07	1
3	ffff82501cea40309d5fdd7edcca4a07	1

Then i added a new column representing best channel according to highest number of offers completed through one channel (for example if user have two channel and completed 3 offers through one channel and 5 offer through another channel , the 5 offer channel will be that user best channel)
Then I added that new best offer channel to the model dataframe.
The next step I made a new column for best offer which represent best offer for each single user through comparing each offer data and choosing best offer for each user .
The next step I made a new column best option which represent the the best offer for user and his best channel combined.

Next i dropped all columns except :

gender	age	became_member_on	income	best_option

Then i encoded categorical columns and applied normalization to numerical ones

I ended up with the following dataset

id	gender	age	became_member_on	income	best_option
0009655768c64bdeb2e877511632db8f	1	0.180723	4	0.466667	5
0011e0d4e6b944f998e987f904e8c1e5	2	0.265060	5	0.300000	5
0020c2b971eb4e9188eac86d93036a77	0	0.493976	3	0.666667	5
0020ccbbb6d84e358d3414a3ff76cffd	0	0.072289	3	0.333333	4
003d66b6608740288d6cc97a6903f4f0	0	0.096386	4	0.477778	5
...	...	...	...	...	...
fff29fb549084123bd046dbc5ceb4faa	0	0.493976	4	0.700000	4
fff3ba4757bd42088c044ca26d73817a	0	0.614458	2	0.588889	8
fff7576017104bcc8677a8d63322b5e1	1	0.638554	4	0.477778	5
fffad4f4828548d1b5583907f2e9906b	1	0.192771	4	0.044444	5
ffff82501cea40309d5fdd7edcca4a07	0	0.325301	3	0.355556	8

Implementation

Next step after data processing is implementation first I defined a function to take dataframe and split it to training and testing data :

```
# split into train/test
def train_test_split_df(transaction_df, train_frac= 0.7, seed=42):
    '''Shuffle the data and randomly split into train and test sets;
       separate the class labels (the column in transaction_df) from the features.
       :param df: Dataframe of all credit card transaction data
       :param train_frac: The decimal fraction of data that should be training data
       :param seed: Random seed for shuffling and reproducibility, default = 1
       :return: Two tuples (in order): (train_features, train_labels), (test_features, test_labels)
       '''

    # convert the df into a matrix for ease of splitting
    df_matrix = transaction_df.values

    # shuffle the data
    np.random.seed(seed)
    np.random.shuffle(df_matrix)

    # split the data
    train_size = int(df_matrix.shape[0] * train_frac)
    # features are all but last column
    train_features  = df_matrix[:train_size, :-1]
    # class labels *are* last column
    train_labels = df_matrix[:train_size, -1]
    # test data
    test_features = df_matrix[train_size:, :-1]
    test_labels = df_matrix[train_size:, -1]

    return (train_features, train_labels), (test_features, test_labels)
```

Then I used it to divide my data to 80 % training and 20 % for testing data

I faced then a problem that data is biased toward a certain class which is 5.0 , so I used oversampling and undersampling techniques through combining over- and under-sampling using SMOTE and Edited Nearest Neighbours. I used implementation from imblearn library details below : https://imbalanced-learn.readthedocs.io/en/stable/generated/imblearn.combine.SMOTEENN.html#imblearn.combine.SMOTEENN

The next step is using machine learning models on data for linearlearner i only created model illustrated in algorithms section and deployed it and used deployed model to get predictions , for Sklearn models I had to upload the data to S3 so I used the function below

```python
def make_csv(x, y, filename, data_dir,test):
    '''Merges features and labels and converts them into one csv file with labels in the first column.
       :param x: Data features
       :param y: Data labels
       :param file_name: Name of csv file, ex. 'train.csv'
       :param data_dir: The directory where files will be saved
       '''
    # make data dir, if it does not exist
    if not os.path.exists(data_dir):
        os.makedirs(data_dir)
    # if not testing data append labels else don't append labels
    if test :
        data = pd.concat([pd.DataFrame(y), pd.DataFrame(x)], axis=1)
        data.to_csv(os.path.join(data_dir, filename), index=False, header=False)
    else :
        pd.DataFrame(x).to_csv(os.path.join(data_dir, 'test.csv'), header=False, index=False)


    print('Path created: '+str(data_dir)+'/'+str(filename))
```

The above function is from my palgarism detection project

To create csv files and then uploaded them to s3 and used Sklearn models as mentioned in above sections and then deployed the models get predictions and evaluated them , I did the same process for XGBoost algorithm .

Refinement

I went through some Refinement processes first I meet low model accuracy and recall which was because data was biased so as mentioned in above section I made some modifications to data in order to get rid of this bias which increased the models accuracy greatly

Accuracy score	Linearlearner	Decision tree	SVC	KNN	XGBoost
Before	≈0.64	≈0.64	≈0.64	≈0.64	≈0.64
After	**All above 90 %**				

I also went through various model hyperparameters testing for SVC

Accuracy score	SVC
Linear Kernel	≈0.91
Rbf Kernel	≈0.93

Accuracy score	SVC
C=0.05	≈0.92
C=0.1	≈0.924
C=0.5	≈0.93

I also went through some modifications for decision tree :

Accuracy score	SVC
Max depth =50	≈0.77
Max depth =20	≈0.776
Max depth =10	≈0.78

Results

Model Evaluation and Validation

After training and testing the predictions of different classifiers mentioned above Support vector machine approach has proven to give the best performance , given table below

	SVC
accuracy	0.938476
recall	0.938476
precision_score	0.936877
f1_score	0.923689

The model has accuracy of nearly 94 % meaning that model was able to predict correctly nearly 1023 out of 1089 entries of test data which is very good indication of model performance

Also the model has recall (TP / TP +FN) of nearly 94 % which means that true positives is at its maximum along other models trained and also false negatives is at it minimum

Also the model has recall (TP / TP +FP) of nearly 93.68 % which means that true positives is at its maximum along other models trained and also false positives is at it minimum

This is ended up with f1-score of 92,3% which is a very good indicator of performance

Having all this good numbers and indicators of evaluation matrices I very confident that this model is very good at predicting what is the best offer and channel of specific user correctly however since the customers have very different preferences it is hard to have 100% prediction accuracy without overfitting model but with above performance from SVC model I can say that that model is very good for non-sensitive reason as prediction customer needs.

Justification

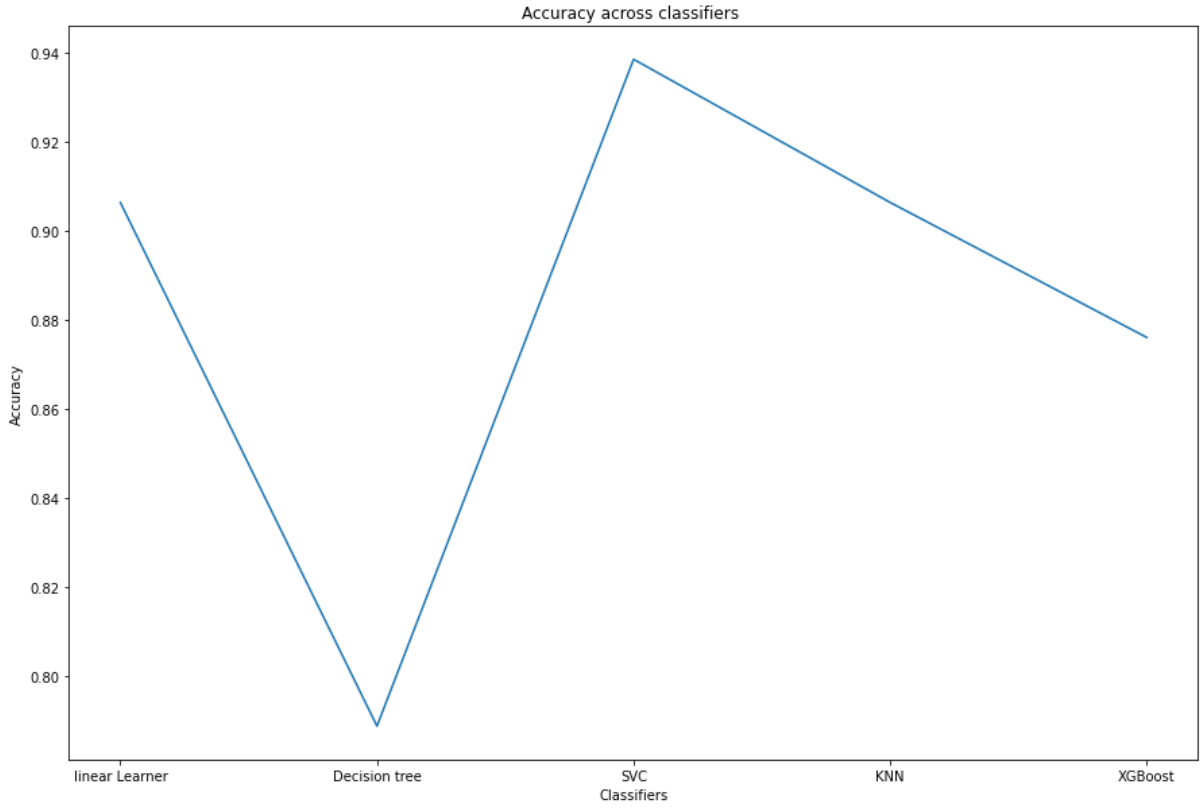
To Sum up everything we started with the benchmark indicating that the CR on the viewed offers is **43% for BOGO, 56% for Discount** (37% and 42% on all the received offers).

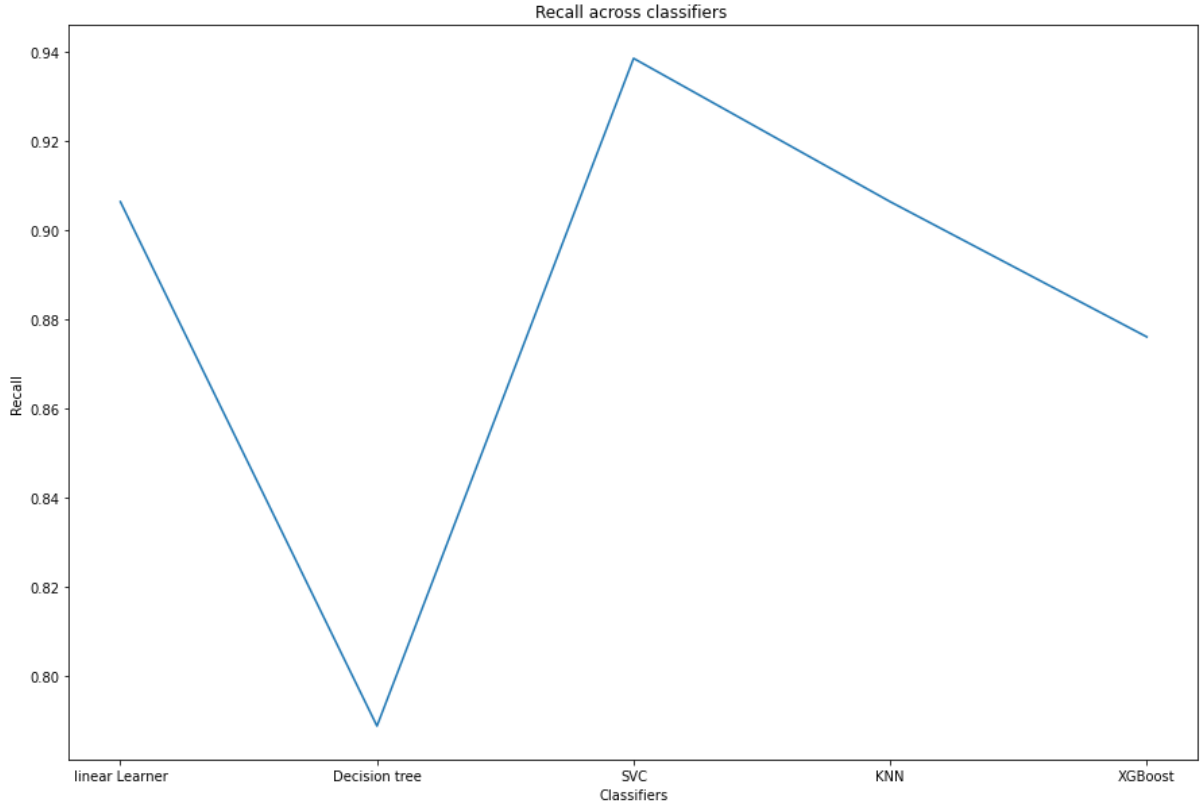
After implementing 5 different models which is :

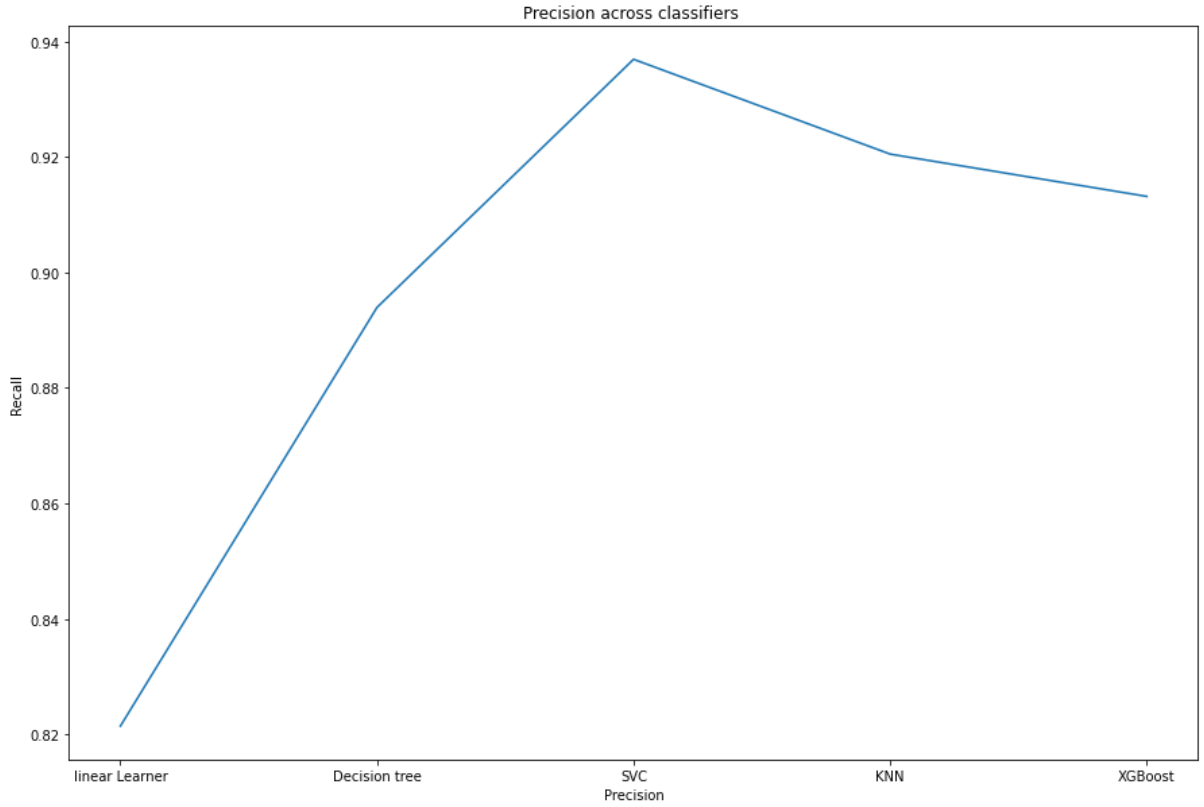
LinearLearner	Decision tree	SVC	KNN	XGBOOST

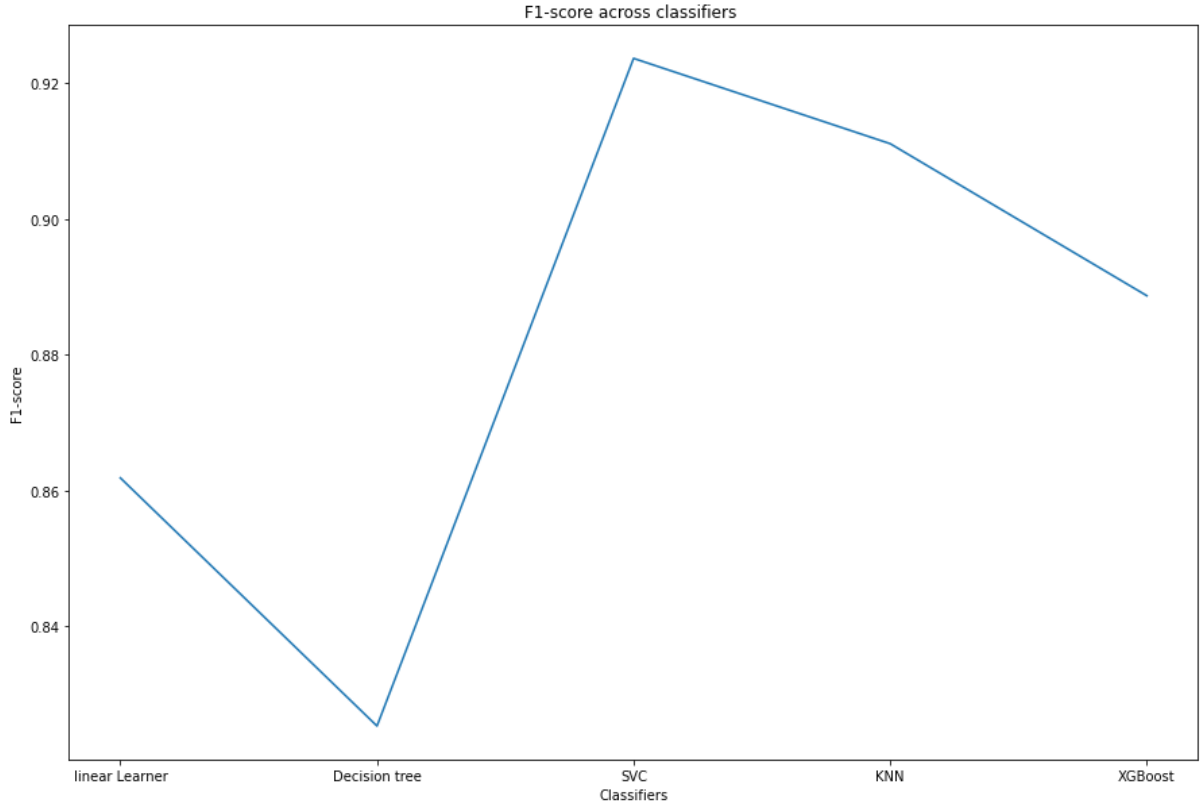
And training them and evaluating their performance I ended up with the following performance from the different models

	LinearLearner	Decision tree	SVC	KNN	XGBOOST
accuracy	0.906336	0.788797	0.938476	0.906336	0.876033
recall	0.906336	0.788797	0.938476	0.906336	0.876033
precision_score	0.821445	0.893920	0.936877	0.920463	0.913142
f1_score	0.861805	0.825209	0.923689	0.911103	0.888677


Accuracy across classifiers
0.94
0.92
0.90
0.88
0.86
0.84
0.82
0.80
Accuracy
linear Learner
Decision tree
SVC
KNN
XGBoost
Classifiers


Recall across classifiers
0.94
0.92
0.90
0.88
0.86
0.84
0.82
0.80
Recall
linear Learner
Decision tree
SVC
KNN
XGBoost
Classifiers





LinearLearner model has second highest accuracy and recall but it has low precision and f1-score

KNN model has second highest accuracy after SVC and high recall rate and also high precision rate yet lower than SVC and good SVC model

Having SVC , linear learner and KNN model which is best performance model we can say that SVC was the best to train successfully and best to learn patterns of data to predict most of data correctly.

Conclusions

Reflection

This project can be summarized as the following :
1- Cleaning all of the Datasets
2- Knowing which features needed to build the models
3-Preprocess in creating and engineering features
4-Train models on processed datasets
5-Evaluate models
6-Choose best model

Improvements

This project can be improved through getting bigger and less biased datasets from multiple sources , also if the application can gather more information about user and their backgrounds these features might be of use and increase model accuracy and increase the model understanding of data patterns.
Also Building a Neural Network might have better performance than tree and k-means and even SVC but might be performance and time costly