Project Overview :

This data set contains simulated data that mimics customer behavior on the Starbucks rewards mobile app.

Once every few days, Starbucks sends out an offer to users of the mobile app.

An offer can be merely an advertisement for a drink or an actual offer such as a discount or BOGO (buy one get one free).

Some users might not receive any offer during certain weeks.

Not all users receive the same offer, and that was the challenge to solve with this data set.

Our task is to combine transaction, demographic and offer data to determine which demographic groups respond best to which offer type. This data set is a simplified version of the real Starbucks app because the underlying simulator only has one product whereas Starbucks actually sells dozens of products.

Every offer has a validity period before the offer expires.

As an example, a BOGO offer might be valid for only 5 days. Informational offers also have a validity period even though these ads are merely providing information about a product; for example,

if an 1 informational offer has 7 days of validity, we can assume the customer is feeling the influence of the offer for 7 days after receiving the advertisement.

Data Dictionary :

profile.json Rewards program users (17000 users x 5 fields)

- gender: (categorical) M, F, O, or null
- age: (numeric) missing value encoded as 118
- id: (string/hash)

portfolio.json Offers sent during 30-day test period (10 offers x 6 fields)

- 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
- id: (string/hash)

transcript.json Event log (306648 events x 4 fields)

- person: (string/hash) 2
- 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

Offer Types :

- 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 spent.
- In an informational offer, there is no reward, but neither is there a requisite amount that the user is expected to spend.

Offers can be delivered via multiple channels.

Problem Statement :

4 Predicting the purchase offer to which a possible higher level of response or user actions like ‘offer received’, ‘offer viewed’, ‘transaction’ and ‘offer completed’ can be achieved based on the demographic attributes of the customer and other attributes of the companies purchase offers.

Evaluation Metrics :

Accuracy is the quintessential classification metric. ... And easily suited for binary as well as a multiclass classification problem and I choose it cause I want to increase the true positive and decrease the false negative equally so I will not chose the pression or the recall metrics

Data collecting, Accessing and Cleaning :

- Read portfolio, profile, transcript from their json file and convert them to Dataframe

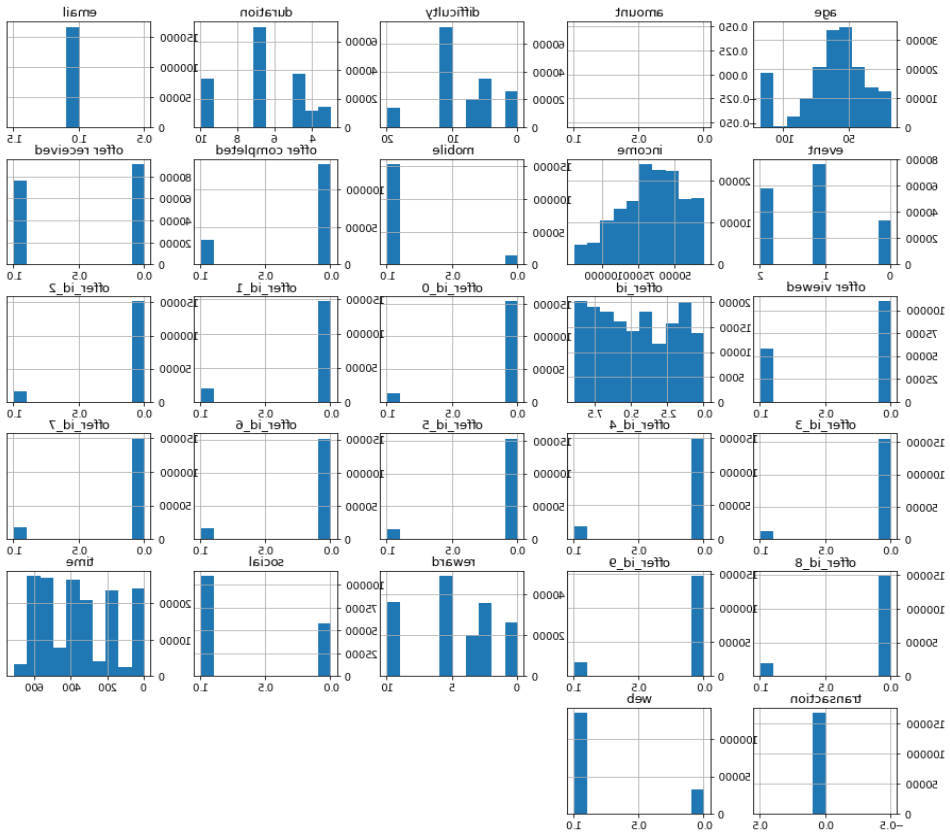
```
In [1]: import pandas as pd
        import numpy as np
        import math
        import json
        % matplotlib inline

        # read in the json files
        portfolio = pd.read_json('data/portfolio.json', orient='records', lines=True)
        profile = pd.read_json('data/profile.json', orient='records', lines=True)
        transcript = pd.read_json('data/transcript.json', orient='records', lines=True)
```

- Explore the data
- Change the Value column in transcript dataframe to 2 columns (offer_id , amount)
- Merge between ‘transcript’ and ‘profile’ on (‘customer_id’) as df
- Then merge the output (df) with ‘portfolio’ on (‘offer_id’)

Data Exploration and Data Visualization :

- Explore the data distributions on histograms

```
data.hist(figsize=(15,15));
```



- Get the info and the description of the numerical data

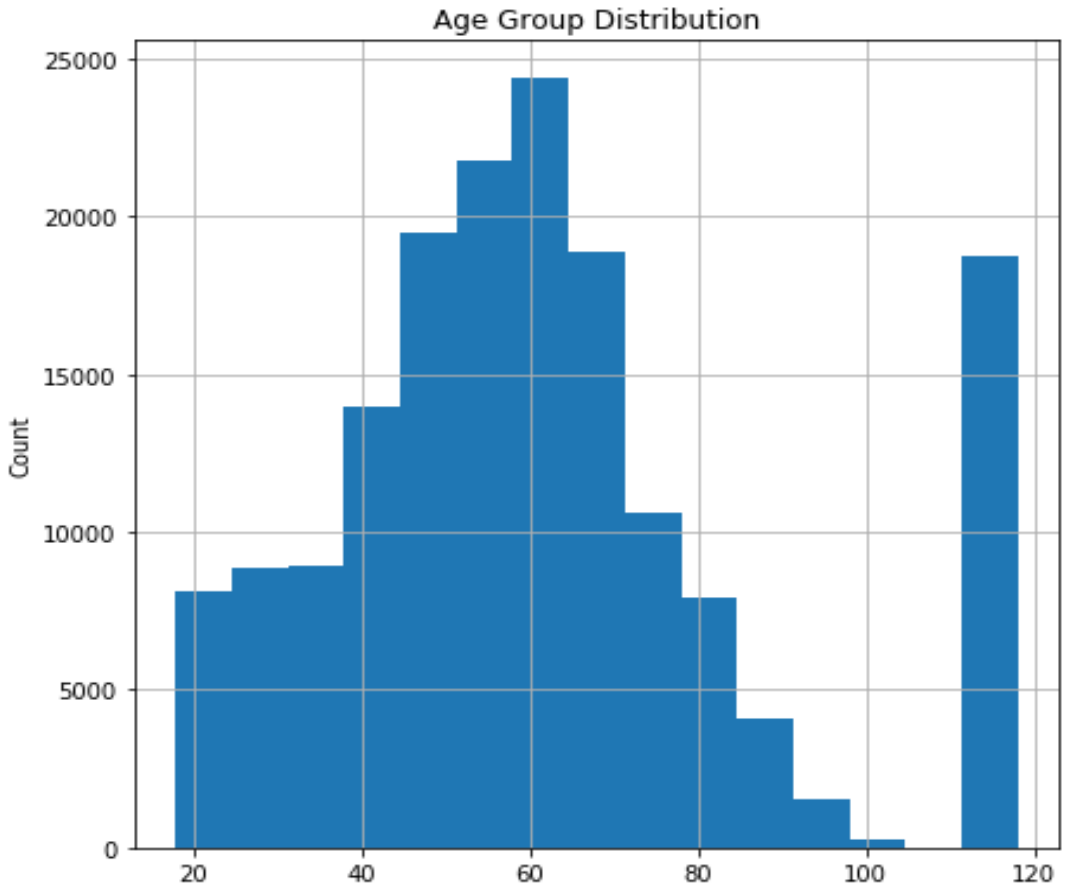
```
In [38]: data[['time','amount','age','became_member_on','gender','income','difficulty','duration','reward']].describe()
```

Out[38]:

	time	amount	age	income	difficulty	duration	reward
count	167581.000000	0.0	167581.000000	148805.000000	167581.000000	167581.000000	167581.00000
mean	353.778412	NaN	61.862616	66414.119149	7.850401	6.610737	4.41991
std	198.301287	NaN	25.693155	21496.947967	5.048944	2.136130	3.37336
min	0.000000	NaN	18.000000	30000.000000	0.000000	3.000000	0.00000
25%	168.000000	NaN	45.000000	51000.000000	5.000000	5.000000	2.00000
50%	408.000000	NaN	58.000000	65000.000000	10.000000	7.000000	5.00000
75%	510.000000	NaN	72.000000	81000.000000	10.000000	7.000000	5.00000
max	714.000000	NaN	118.000000	120000.000000	20.000000	10.000000	10.00000

- Investigate more in the age data to find out if there is any outliers

```
data.age.hist(figsize =(7,7), bins =15)
plt.xlabel('Age Group')
plt.ylabel('Count')
plt.title('Age Group Distribution');
```

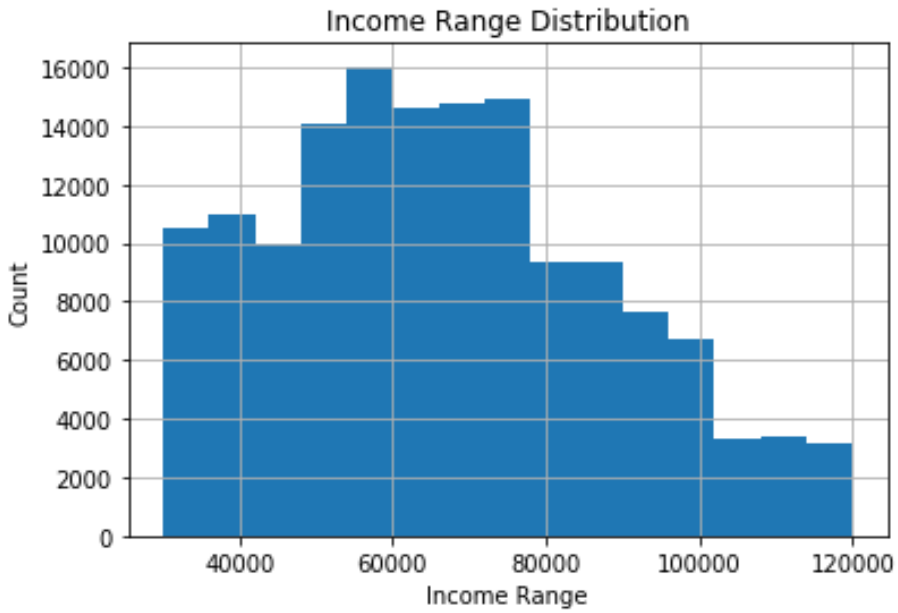


- Outliers observed so I will drop data of age more than 100

```
indexAge = df[(df['age'] >= 100)].index
df.drop(indexAge , inplace=True)
df.head(15)
```

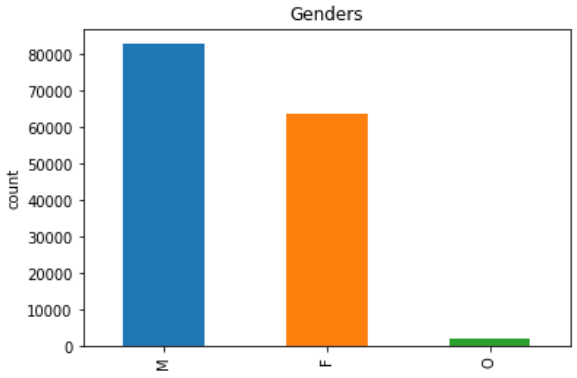
- Investigate more in ‘Income’ data to find out if there is any outliers but no outliers detected

```
data.income.hist(bins = 15);
plt.xlabel('Income Range')
plt.ylabel('Count')
plt.title('Income Range Distribution');
```



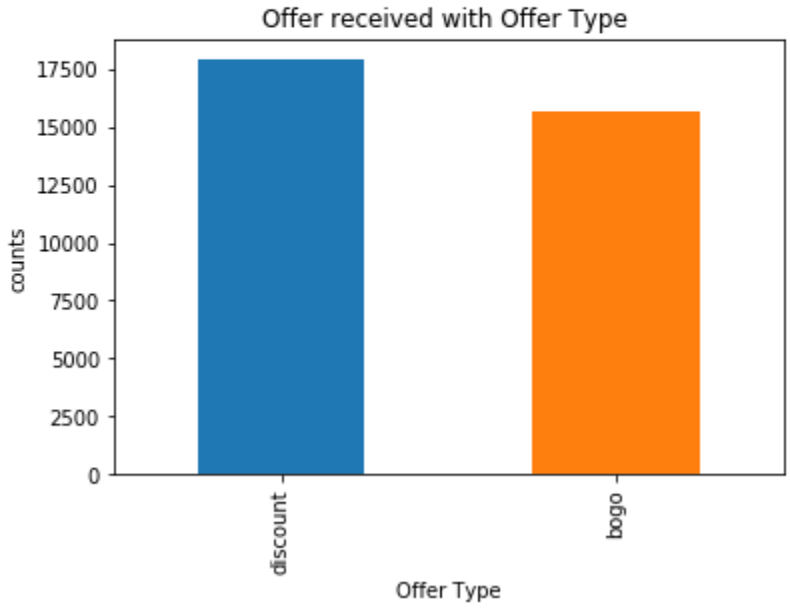
- Check the numbers of ‘gender’

```
ax = data.gender.value_counts()
ax.plot(kind='bar')
plt.ylabel('count')
plt.title('Genders');
```



- Check the count difference between different of types where **offer completed**

```
offer_received = data[data['offer completed'] == 1].offer_type.value_counts()
offer_received.plot(kind='bar')
plt.ylabel('counts')
plt.xlabel('Offer Type')
plt.title('Offer received with Offer Type ');
```



Data preprocessing:

- Convert the “Became_member_on” column into Year, Month, Day columns

```
data['year'] = df['became_member_on'].apply(lambda mytime: int(str(mytime)[0:4]))
data['day'] = df['became_member_on'].apply(lambda mytime: int(str(mytime)[4:6]))
data['month'] = df['became_member_on'].apply(lambda mytime: int(str(mytime)[6:8]))

data[['became_member_on','year','month','day']].head()
```

	became_member_on	year	month	day
0	20170509	2017.0	9.0	5.0
1	20170509	2017.0	9.0	5.0
2	20170509	2017.0	9.0	5.0
3	20180426	2018.0	26.0	4.0
4	20180426	2018.0	26.0	4.0

- One hot encode the “offer type” column into Year, Month, Day columns

```
dummy = pd.get_dummies(data['offer_type'],prefix='offer_type')
data = pd.concat([df, dummy], axis=1 )
data[['offer_type', 'offer_type_bogo', 'offer_type_discount', 'offer_type_informational']].head()
```

	offer_type	offer_type_bogo	offer_type_discount	offer_type_informational
0	bogo	1	0	0
1	bogo	1	0	0
2	bogo	1	0	0
3	bogo	1	0	0
4	bogo	1	0	0

- One hot encode the “offer id” column into Year, Month, Day columns

```
dummy = pd.get_dummies(data['offer_id'],prefix='offer_id')
data = pd.concat([data, dummy], axis=1 )
data.head()
```

- One hot encode ‘event’ column

```
dummy = pd.get_dummies(data['event'])
data = pd.concat([data, dummy], axis=1 )
data[['event', 'offer completed','offer received', 'offer viewed']].head()
```

	event	offer completed	offer received	offer viewed
0	offer received	0	1	0
1	offer viewed	0	0	1
2	offer completed	1	0	0
3	offer received	0	1	0
4	offer viewed	0	0	1

- Convert the channels column into 4 column for each channel and put one if it exist in the channel array

```
dummy = pd.get_dummies(data.channels.apply(pd.Series).stack()).sum(level=0)
data = pd.concat([data, dummy], axis=1)
data = data.drop(columns='channels')
data[['email', 'mobile', 'social', 'web']].head()
```

```
/tmp/ipykernel_27887/3169838132.py:1: FutureWarning: Using the level keyword i
and will be removed in a future version. Use groupby instead. df.sum(level=1)
  dummy = pd.get_dummies(data.channels.apply(pd.Series).stack()).sum(level=0)
```

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

Handling Null values :

- There are 18,953 nulls in some columns so I dropped them because it is a very small proportion of the data so I dropped them to avoid missing with the data

Data Modeling :

- choose the features of training

```
]: X = data.drop(['customer_id' , 'event' ,'offer completed', 'offer received',
       'offer viewed', 'transaction', 'offer_id'], axis=1)
   Y = data['offer completed']
```

- split data to Training and testing data

```
from sklearn.model_selection import train_test_split


X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=0.125, random_state=42)
print(X_train.shape, X_test.shape, y_train.shape, y_test.shape)
```

- save them in different CSV files to be more accessible for and future need of for the deployment process

```
train_data.to_csv('data/train.csv', index=False)
test_data.to_csv('data/test.csv', index=False)
```

- read the data from the saved file to use them in the classification process

```
X_train = pd.read_csv('data/train.csv').drop(columns=['event'])
X_test = pd.read_csv('data/test.csv').drop(columns=['event'])
y_train = pd.read_csv('data/train.csv')['event']
y_test = pd.read_csv('data/test.csv')['event']
print(X_train.shape, X_test.shape, y_train.shape, y_test.shape)
```

- scale the data using standard scaler

```
std = StandardScaler()
X_train = std.fit_transform(X_train)
X_test = std.transform(X_test)
```

how to refine and tune the models and how to get the best hyperparametes:

to get the best hyperparameter I decided to use a method in GridSearchCV

```
from sklearn.model_selection import GridSearchCV
```

which will takes the algorithm (estimator) and all the parameters you want to try and it returns the parameters the gives the highest accuracy

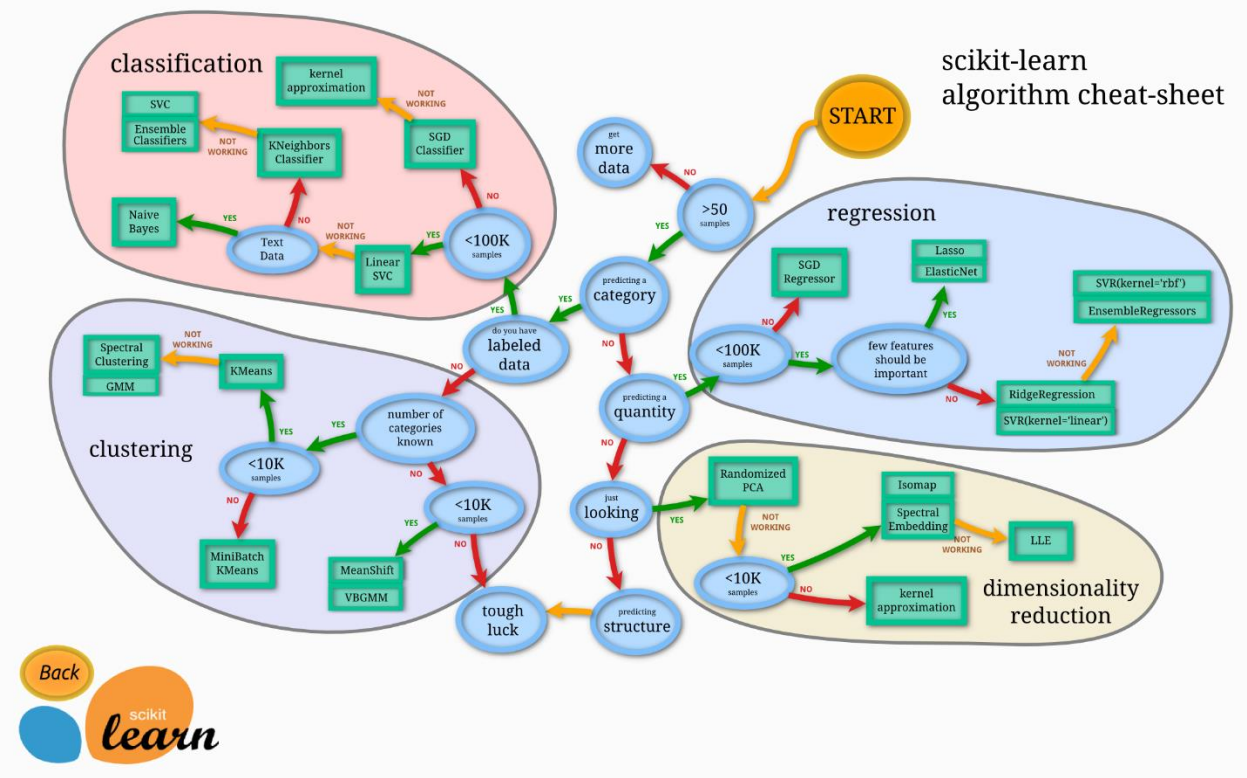
so I did that for the 3 model I choose (SGD, naïve bayes and Random forest Classifiers)

after getting the best of each algorithm I compared between them to choose the one I will deploy

Benchmark model:

We use SGD classifier model as a Benchmark in which to compare our models' performance to, because it is theoretically the best choice for our problem because we have >100K records

Ref: https://scikit-learn.org/stable/tutorial/machine_learning_map/index.html



Model and their performance:

SGD Classifier:

- With best parameters ({'alpha': 0.001,loss: ‘squared_error’, 'penalty': 'L1'})

First model : SGD Classifier

```
param_grid = dict(penalty=['l2', 'l1','elasticnet'],
                          alpha=[0.1, 0.01, 0.001, 0.0001],
                         loss = ['squared_error','hinge'])

clf_SGD = GridSearchCV(estimator=SGDClassifier(), param_grid= param_grid)
clf_SGD.fit(X_train, y_train)

print("training Accuracy: ", clf_SGD.score(X_train,y_train))
print("testing Accuracy: ", clf_SGD.score(X_test,y_test))
print(clf_SGD.best_params_)
```

```
training Accuracy:  0.7819360394928065
testing Accuracy:  0.7825501910759459
{'alpha': 0.001, 'loss': 'squared_error', 'penalty': 'l1'}
```

Naïve bayes Classifier (GaussianNB):

- With best parameters: ('var_smoothing': 1.0)

2nd model : Naive Bayes classifier (gaussian Naive bayes)

```
param_grid= {
    'var_smoothing': np.logspace(0,-9, num=100)
}

clf_NB = GridSearchCV(estimator=GaussianNB(),param_grid= param_grid)
clf_NB.fit(X_train, y_train)

print("traning Accuracy: ", clf_NB.score(X_train,y_train))
print("testing Accuracy: ", clf_NB.score(X_test,y_test))
print("best parameters: ", clf_NB.best_params_)
```

```
traning Accuracy:  0.5612653692069912
testing Accuracy:  0.559395015878142
best parameters:  {'var_smoothing': 1.0}
```

Random forest Classifier: (not in proposal)

- With best parameters: ('max_depth': 6, 'min_samples_split': 5)

3rd model: Random forest Classifier

```
param_grid = {"max_depth": [3,6, None], "min_samples_split": [3, 5, 10, 15]}
clf_RF = GridSearchCV(estimator=RandomForestClassifier(),param_grid= param_grid)
clf_RF.fit(X_train, y_train)

print("traning Accuracy: ", clf_RF.score(X_train,y_train))
print("testing Accuracy: ", clf_RF.score(X_test,y_test))
print("best parameters: ", clf_RF.best_params_)
```

```
traning Accuracy:  0.7957692869610685
testing Accuracy:  0.7950374078260402
best parameters:  {'max_depth': 6, 'min_samples_split': 5}
```

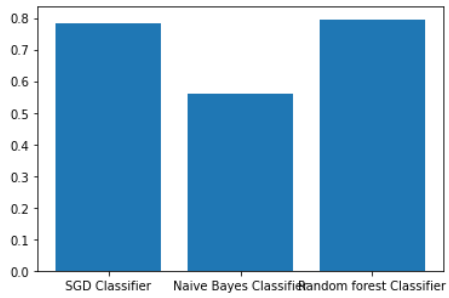
<u>check the accuracy of those models to choose the best of them which will be deployed based on our evaluation metrics the training and testing accuracy</u>

- Comparison between the 3 models in accuracy on the training data:

compare between their accuracy on the training data

```
acc_on_training = {}
acc_on_training['SGD Classifier'] = clf_SGD.score(X_train,y_train)
acc_on_training['Naive Bayes Classifier'] = clf_NB.score(X_train,y_train)
acc_on_training['Random forest Classifier'] = clf_RF.score(X_train,y_train)

print(acc_on_training)

plt.bar(range(len(acc_on_training)), list(acc_on_training.values()), tick_label=list(acc_on_training.keys()))
plt.show()
```

```
{'SGD Classifier': 0.7819360394928065, 'Naive Bayes Classifier': 0.5612653692069912, 'Random forest Classifier': 0.795769286961
0685}
```

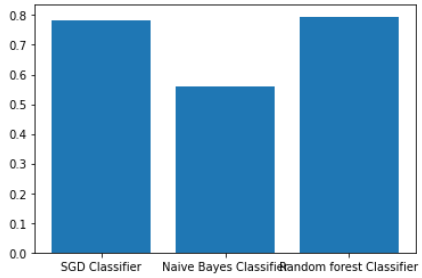


- Comparison between the 3 models in accuracy on the testing data:

compare between their accuracy on the testing data

```
acc_on_testing = {}
acc_on_testing['SGD Classifier'] = clf_SGD.score(X_test,y_test)
acc_on_testing['Naive Bayes Classifier'] = clf_NB.score(X_test,y_test)
acc_on_testing['Random forest Classifier'] = clf_RF.score(X_test,y_test)

print(acc_on_testing)

plt.bar(range(len(acc_on_testing)), list(acc_on_testing.values()), tick_label=list(acc_on_testing.keys()))
plt.show()
```

```
{'SGD Classifier': 0.7825501910759459, 'Naive Bayes Classifier': 0.559395015878142, 'Random forest Classifier': 0.7950374078260
402}
```



Observation:

Both Random forest and SGD classifiers gets high accuracies while the Random Forest Classifier gives the highest accuracy in both training and testing so it is the one will get deployed with hyperparameters (max_depth= 6, min_samples_split= 5)

Deployment on AWS :

- upload the data to S3 bucket

```
sm_boto3 = boto3.client("sagemaker")

sess = sagemaker.Session()

region = sess.boto_session.region_name

bucket = sess.default_bucket()  # this could also be a hard-coded bucket name

print("Using bucket " + bucket)
```

```
Using bucket sagemaker-us-east-1-408035773647
```

```
# send data to S3. SageMaker will take training data from s3
trainpath = sess.upload_data(
    path="data/train.csv", bucket=bucket, key_prefix="sagemaker/sklearncontainer"
)

testpath = sess.upload_data(
    path="data/test.csv", bucket=bucket, key_prefix="sagemaker/sklearncontainer"
)
```

- create the training script with the classifier which give the higher accuracy

```
%%writefile script.py

import argparse
import joblib
import os

import numpy as np
import pandas as pd


# inference functions ---------------
def model_fn(model_dir):
    clf = joblib.load(os.path.join(model_dir, "model.joblib"))
    return clf


if __name__ == "__main__":

    print("extracting arguments")
    parser = argparse.ArgumentParser()

    # hyperparameters sent by the client are passed as command-line arguments to the script.
    # to simplify the demo we don't use all sklearn RandomForest hyperparameters
    parser.add_argument("--n-estimators", type=int, default=10)
    parser.add_argument("--min-samples-leaf", type=int, default=3)

    # Data, model, and output directories
    parser.add_argument("--model-dir", type=str, default=os.environ.get("SM_MODEL_DIR"))
    parser.add_argument("--train", type=str, default=os.environ.get("SM_CHANNEL_TRAIN"))
    parser.add_argument("--test", type=str, default=os.environ.get("SM_CHANNEL_TEST"))
    parser.add_argument("--train-file", type=str, default="train.csv")
```

```
    X_test = test_df.drop(columns=[args.target])
    y_train = train_df[args.target]
    y_test = test_df[args.target]

    # train
    print("training model")
    from sklearn.preprocessing import StandardScaler
    from sklearn.linear_model import SGDClassifier, LogisticRegression
    from sklearn.naive_bayes import GaussianNB
    from sklearn.tree import DecisionTreeClassifier
    from sklearn.ensemble import RandomForestClassifier

    from sklearn.model_selection import GridSearchCV
    from sklearn.pipeline import Pipeline
    std = StandardScaler()

    X_train = std.fit_transform(X_train)
    X_test = std.transform(X_test)


    model = RandomForestClassifier(max_depth= 6, min_samples_split= 5)
    model.fit(X_train, y_train)

    print("traning Accuracy: ", model.score(X_train,y_train))
    print("testing Accuracy: ", model.score(X_test,y_test))
```

```
! python script.py --model-dir ./ \
                   --train ./data \
                   --test ./data \
                   --features 'CRIM ZN INDUS CHAS NOX RM AGE DIS RAD TAX PTRATIO B LSTAT' \
                   --target offer_completed
```

- create the estimator for to start a training job

```
from sagemaker.sklearn.estimator import SKLearn

FRAMEWORK_VERSION = "0.23-1"

sklearn_estimator = SKLearn(
    entry_point="script.py",
    role=get_execution_role(),
    instance_count=1,
    instance_type="ml.m4.xlarge",#"ml.c5.xlarge",#"ml.m4.xlarge",
    framework_version=FRAMEWORK_VERSION
)

import time
# tic = time.clock()

# launch training job, with asynchronous call
sklearn_estimator.fit({"train": trainpath, "test": testpath}, wait=True)
```

- get the trained model artifact

```
artifact = sm_boto3.describe_training_job(
    TrainingJobName=sklearn_estimator.latest_training_job.name
)["ModelArtifacts"]["S3ModelArtifacts"]

print("Model artifact persisted at " + artifact)
```

- deploy the model to endpoint

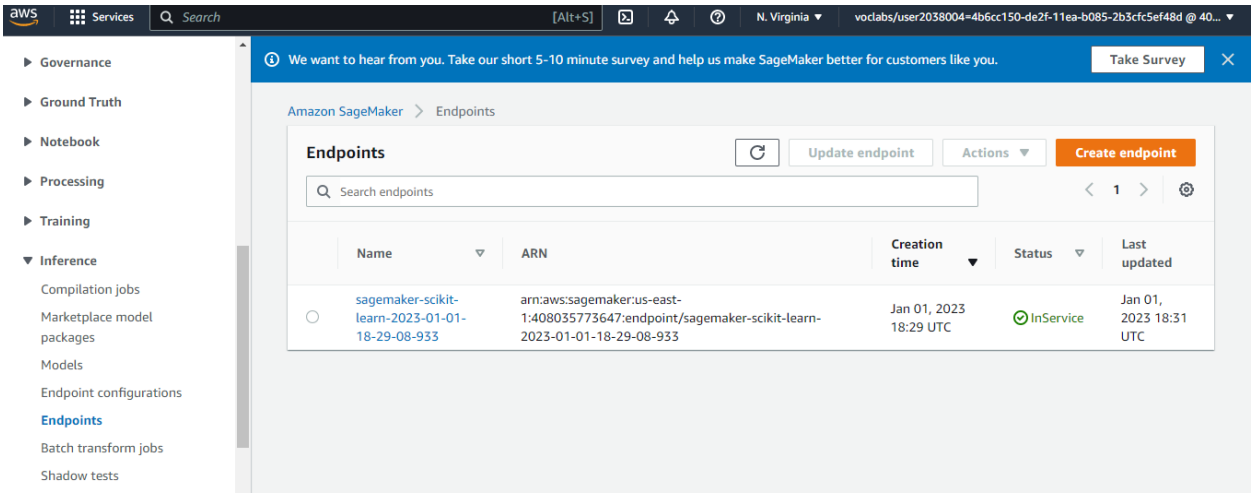
```
from sagemaker.sklearn.model import SKLearnModel

model = SKLearnModel(
    model_data=artifact,
    role=get_execution_role(),
    entry_point="script.py",
    framework_version=FRAMEWORK_VERSION,
)

predictor = model.deploy(instance_type="ml.c5.large", initial_instance_count=1)
```

```
INFO:sagemaker:Creating model with name: sagemaker-scikit-learn-2022-12-31-00-09-24-291
INFO:sagemaker:Creating endpoint-config with name sagemaker-scikit-learn-2022-12-31-00-09-24-725
INFO:sagemaker:Creating endpoint with name sagemaker-scikit-learn-2022-12-31-00-09-24-725
```

```
-----!
```

```
sm_boto3.delete_endpoint(EndpointName=predictor.endpoint_name)
```

- the deployed endpoint:



References :

- https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.replace.html

- https://pandas.pydata.org/pandas-docs/version/0.23.4/generated/pandas.merge.html

- https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.SGDClassifier.html

- https://scikit-learn.org/stable/modules/generated/sklearn.naive_bayes.GaussianNB.html

- https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html#sklearn.ensemble.RandomForestClassifier

Thank you