Credit Card Approval Prediction

Group 4
Min Joo / Hoon /Shino / Fang Wei

CRISP-DM for **problem-solving**

1. Business Understanding	Credit card approval prediction can not only benefit for bank and applicants
2. Data Understanding	Collated the historical loan data and background info from previous applicants
2. Data Preparation	Clean data, classify the good/bad applicants and do feature engineering
3. Modeling Building	Modeling Data with logistic regression and Random Forest.
3. Testing Evaluation	Compare 2 models
4.Development	Finding and overcoming the limitations of our project

01 Business Understanding

Problem & Topic
Predictive Analytics

Credit card approval prediction by analyzing the credit card applicants' data.

1. Help the bank, by minimizing the potential risks of issuing credit cards for unqualified applicants.

2. The credit card applicants know in advance whether they can pass the application to avoid credit score decline due to failure.

02

Data Understanding & Preparation

Prepare data and libraries

Import required libraries

```
import warnings
warnings.filterwarnings('ignore')
#Import the warnings library and suppress warnings that might occur in the code.

import numpy as np
#Support numerical operations
import pandas as pd
#Data structures and data analysis.
import matplotlib.pyplot as plt
import seaborn as sns
#Matplotlib and Seaborn are visualization libraries used to create charts and graphs.

from imblearn.over_sampling import SMOTE
#imports the Synthetic Minority Over-sampling Technique (SMOTE) algorithm used to address class imbalance in machine learning problems.
import itertools
#produce complex iterators.

from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, confusion_matrix
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
#working with data and building machine learning models.

from xgboost import XGBClassifier
from lightgbm import LGBMClassifier
from sklearn import svm
from sklearn.ensemble import RandomForestClassifier
#import decision tree model
```

Import data from files

```
from google.colab import files
f=files.upload()
```

```
選擇檔案 未選擇任何檔案        Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable.
Saving application_record.csv to application_record.csv
```

```
f_list=files.upload()
```

```
選擇檔案 未選擇任何檔案        Upload widget is only available when the cell has been executed in the current browser sessio
Saving credit_record.csv to credit_record.csv
```

Two dataframe

1. Personal background
2. Personal monthly credit status

Data Understanding

Step 2. Data Understanding

to know colum and variables.

```
[34] data_application = pd.read_csv("application_record.csv")
     print(data_application.head())
     print(data_application.tail())
     data_application.shape

        ID CODE_GENDER FLAG_OWN_CAR FLAG_OWN_REALTY  CNT_CHILDREN  \
0  5008804           M            Y               Y             0
1  5008805           M            Y               Y             0
2  5008806           M            Y               Y             0
3  5008808           F            N               Y             0
4  5008809           F            N               Y             0

   AMT_INCOME_TOTAL      NAME_INCOME_TYPE            NAME_EDUCATION_TYPE  \
0          427500.0               Working               Higher education
1          427500.0               Working               Higher education
2          112500.0               Working  Secondary / secondary special
3          270000.0  Commercial associate  Secondary / secondary special
4          270000.0  Commercial associate  Secondary / secondary special

  NAME_FAMILY_STATUS  NAME_HOUSING_TYPE  DAYS_BIRTH  DAYS_EMPLOYED  \
```

Understand the Application_Record Dataframe

```
[35] data_credit = pd.read_csv("credit_record.csv")
     print(data_credit.head())
     print(data_credit.tail())
     data_credit.shape

        ID  MONTHS_BALANCE STATUS
0  5001711               0      X
1  5001711              -1      0
2  5001711              -2      0
3  5001711              -3      0
4  5001712               0      C
              ID  MONTHS_BALANCE STATUS
1048570  5150487             -25      C
1048571  5150487             -26      C
1048572  5150487             -27      C
1048573  5150487             -28      C
1048574  5150487             -29      C
(1048575, 3)
```

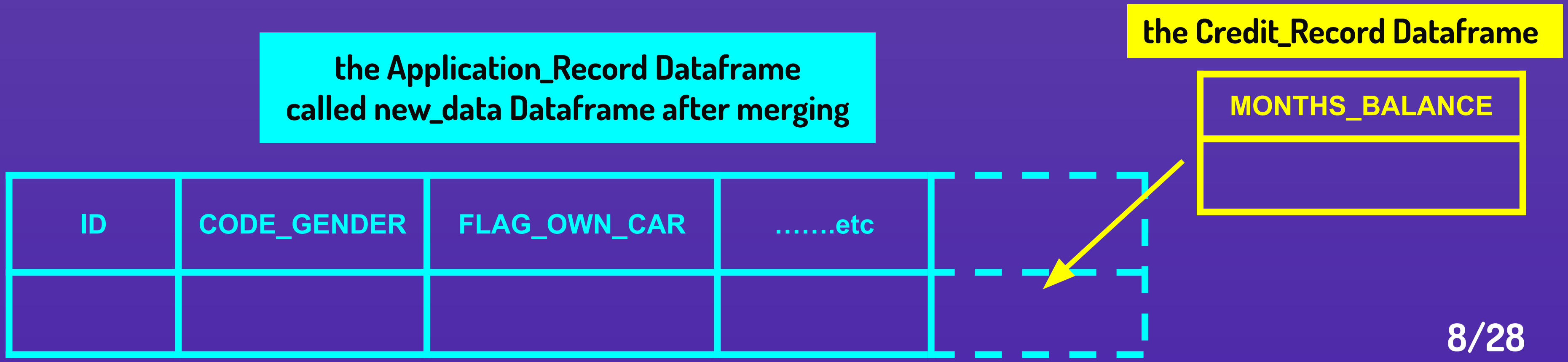
Understand the Credit_Record Dataframe

Data Preparation (New Dataframe)

Step 3-1. Data Preparation (New Dataframe)

1. Merge the colum "MONTHS_BALANCE" of credit_record.csv into a new colum of application_record.csv called **new_data**

```
[36] begin_month=pd.DataFrame(data_credit.groupby(["ID"])["MONTHS_BALANCE"].agg(min))
     begin_month=begin_month.rename(columns={'MONTHS_BALANCE':'begin_month'})
     new_data=pd.merge(data_application,begin_month,how="left",on="ID")
     new_data.shape

(438557, 19)
```



2. Define which ID with good credit or not in overall in credit_record.csv and make a new colum to show the result in application_record.csv

```
[37] #A. if the STATUS shows the specific value from 1-5,then add a colum ("dep_value")in Yes ("Bad record") value in credit_record.csv
     data_credit['dep_value'] = None
     data_credit['dep_value'][data_credit['STATUS'] =='1']='Yes'
     data_credit['dep_value'][data_credit['STATUS'] =='2']='Yes'
     data_credit['dep_value'][data_credit['STATUS'] =='3']='Yes'
     data_credit['dep_value'][data_credit['STATUS'] =='4']='Yes'
     data_credit['dep_value'][data_credit['STATUS'] =='5']='Yes'
```

```
[38] #B. if the same ID more than one Yes, and identify is the bad ID and show in the new colum of "New Data"
     cpunt=data_credit.groupby('ID').count()
     cpunt['dep_value'][cpunt['dep_value'] > 1]='Yes'
     cpunt['dep_value'][cpunt['dep_value'] == 1]='No'
     cpunt['dep_value'][cpunt['dep_value'] == 0]='No'
     cpunt = cpunt[['dep_value']]

     new_data=pd.merge(new_data,cpunt,how='inner',on='ID')
     new_data['target']=new_data['dep_value']
     new_data.loc[new_data['target']=='Yes','target']=1
     new_data.loc[new_data['target']=='No','target']=0
```

```
[39] #C. To count and percentage the Good and Bad ID
     print(cpunt['dep_value'].value_counts())
     cpunt['dep_value'].value_counts(normalize=True)

     No     43401
     Yes     2584
     Name: dep_value, dtype: int64
     No     0.943808
     Yes    0.056192
     Name: dep_value, dtype: float64
```

0: 1-29 days past due
1: 30-59 days past due mark as YES
2: 60-89 days overdue mark as YES
3: 90-119 days overdue mark as YES
4: 120-149 days overdue mark as YES
5: Overdue or bad debts, write-offs for more than 150 days mark as YES
C: paid off that month

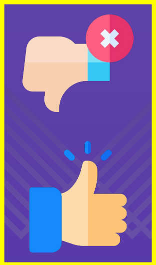
ID	MONTHS_BALANCE	STATUS
12345	-3	3

Feature Engineering

Identify **Good ID or not &** Merge to new dataframe

each **ID, Client number** more than 1 time **YES** **STATUS**

each **ID, Client number** less than or equal to 1 time **YES** **STATUS**



New column variable into the new_data Dataframe

ID	CODE_GENDER	FLAG_OWN_CAR	.......etc	MONTH_BALANCE	

Check New_Data Dataframe is well prepared

3. Make sure new dataframe "new_data" is well prepared.

```
[41] new_data.tail()
```

	ID	CODE_GENDER	FLAG_OWN_CAR	FLAG_OWN_REALTY	CNT_CHILDREN	AMT_INCOME_TOTAL	NAME_INCOME_TYPE	NAME_EDUCATION_TYPE	NAME_FAMILY_S
36452	5149828	M	Y	Y	0	315000.0	Working	Secondary / secondary special	M
36453	5149834	F	N	Y	0	157500.0	Commercial associate	Higher education	M
36454	5149838	F	N	Y	0	157500.0	Pensioner	Higher education	M
36455	5150049	F	N	Y	0	283500.0	Working	Secondary / secondary special	M
36456	5150337	M	N	Y	0	112500.0	Working	Secondary / secondary special	Single / not r

5 rows × 21 columns

Data Preparation (feature-cleaning)

Step 3-2. Data prepation (Preproceess and featuresand-cleaning)

Delete “NaN” & “NULL”

```
[45] new_data.dropna()
     new_data = new_data.mask(new_data == 'NULL').dropna()
```

2. Only rename useful colums

- FLAG_OWN_REALTY:House
- CNT_FAM_MEMBERS: famsize
- FLAG_OWN_CAR: Car
- AMT_INCOME_TOTAL: inc
- OCCUPATION_TYPE: occyp

Rename

```
new_data.rename(columns={'FLAG_OWN_REALTY':' House ','CNT_FAM_MEMBERS':'famsize',
                         'FLAG_OWN_CAR':'Car','AMT_INCOME_TOTAL':'inc','OCCUPATION_TYPE':'occyp'
                         },inplace=True)
```

3. Make the binary features from Y/N to 1/0 and remove the bias colums (EX: gender)

- Car
- House

Change the binary feature from string Y/N to integer 1/0

```
new_data['Car'] = new_data['Car'].replace(['N','Y'],[0,1])
new_data[ " House " '] = new_data['Reality'].replace(['N','Y'],[0,1])
new_data=new_data[["ID","Car"  " House "  ,"famsize","inc","occyp","begin_month","target"]]
new_data.tail()
```

Remove the bias or we believed is not relative columns

						occyp	begin_month	target
36452	5149828	1	1	2.0	315000.0	Managers	-11.0	1
36453	5149834	0	1	2.0	157500.0	Medicine staff	-23.0	1
36454	5149838	0	1	2.0	157500.0	Medicine staff	-32.0	1
36455	5150049	0	1	2.0	283500.0	Sales staff	-9.0	1
36456	5150337	0	1	1.0	112500.0	Laborers	-13.0	1

Continuous variable - dummy

Understand the data type & rename

(1) The number of family members

```
new_data['famsize']=new_data['famsize'].astype(int)
new_data['famsizegp']=new_data['famsize']
new_data['famsizegp']=new_data['famsizegp'].astype(object)
new_data.loc[new_data['famsizegp']>=3,'famsizegp']='3more'
new_data.head()
```

	ID	Car	House	famsize	inc	occyp	begin_month	target	famsizegp
2	5008806	1	1	2	112500.0	Security staff	-29.0	0	2
3	5008808	0	1	1	270000.0	Sales staff	-4.0	0	1
4	5008809	0	1	1	270000.0	Sales staff	-26.0	0	1
5	5008810	0	1	1	270000.0	Sales staff	-26.0	0	1
6	5008811	0	1	1	270000.0	Sales staff	-38.0	0	1

```
new_data["famsizegp"].unique()
```

```
array([2, 1, '3more'], dtype=object)
```

Manipulation operation, called 'new_data' to create the new columns

New column famsizegp_1, famsizegp_3

Divide into 3 groups

```
[ ] new_data["famsizegp"].unique()

array([2, 1, '3more'], dtype=object)
```

```
def convert_dummy(df, feature,rank=0):
    pos = pd.get_dummies(df[feature], prefix=feature)
    mode = df[feature].value_counts().index[rank]
    biggest = feature + '_' + str(mode)
    pos.drop([biggest],axis=1,inplace=True)
    df.drop([feature],axis=1,inplace=True)
    df=df.join(pos)
    return df
```

```
[ ] new_data = convert_dummy(new_data,'famsizegp')
```

famsizegp
2
1
1
1
1

famsizegp_1	famsizegp_3more
0	0
1	0
1	0
1	0
1	0

the family member is described as followings

1… 1 0
2… 0 0
3… 0 1

Visualize the data - dummy

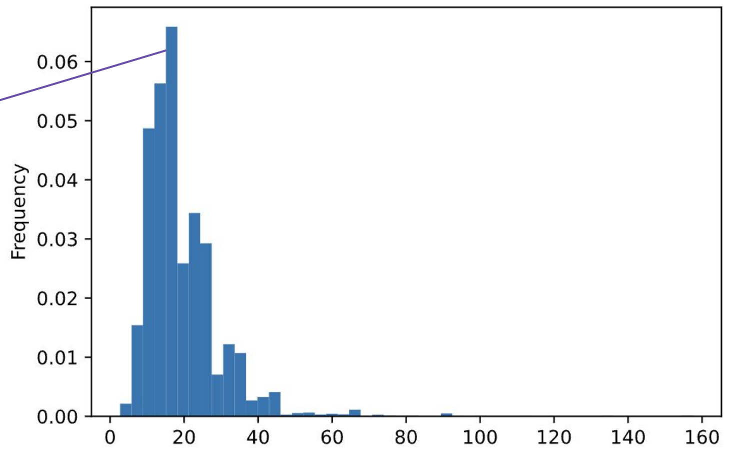
- to know which range of income is the most frequent number

(2) Income range

```
new_data['inc']=new_data['inc'].astype(object)
new_data['inc'] = new_data['inc']/10000
print(new_data['inc'].value_counts(bins=10,sort=False))
new_data['inc'].plot(kind='hist',bins=50,density=True)
```

```
(2.544, 18.18]     14663
(18.18, 33.66]      8464
(33.66, 49.14]      1637
(49.14, 64.62]       175
(64.62, 80.1]        124
(80.1, 95.58]         50
(95.58, 111.06]        4
(111.06, 126.54]       3
(126.54, 142.02]       6
(142.02, 157.5]        8
Name: inc, dtype: int64
&lt;Axes: ylabel='Frequency'&gt;
```

The most frequent range
25K ~ 181K



Visualize the data - dummy

- to know which range of income is the most frequent number

```
[ ] def get_category(df, col, binsnum, labels, qcut = False):
        if qcut:
            localdf = pd.qcut(df[col], q = binsnum, labels = labels) # quantile cut
        else:
            localdf = pd.cut(df[col], bins = binsnum, labels = labels) # equal-length cut

        localdf = pd.DataFrame(localdf)
        name = 'gp' + '_' + col
        localdf[name] = localdf[col]
        df = df.join(localdf[name])
        df[name] = df[name].astype(object)
        return df
```

divided the income level into 3 groups

quantile cut
Using qcut() function,
the data is divided into 3
equal parts that we decide

#equal - length cut
Used so that the width
of bins are equally
divided

```
new_data = get_category(new_data,'inc', 3, ["low","medium", "high"], qcut = True)
new_data.head()
```

	ID	Car	House	famsize	inc	occyp	begin_month	target	famsizegp_1	famsizegp_3more	gp_inc
2	5008806	1	1	2	11.25	Security staff	-29.0	0	0	0	low
3	5008808	0	1	1	27.00	Sales staff	-4.0	0	1	0	high
4	5008809	0	1	1	27.00	Sales staff	-26.0	0	1	0	high
5	5008810	0	1	1	27.00	Sales staff	-26.0	0	1	0	high
6	5008811	0	1	1	27.00	Sales staff	-38.0	0	1	0	high

```
[ ] new_data = convert_dummy(new_data,'gp_inc')
```

Split into three categories based on the values of the 'inc' (income)

'Low', 'medium', 'high'

This process helps us ensure that each category has a roughly equal number of observations.

Categorical variable

(3)Occupation

```
new_data.loc[(new_data['occyp']=='Cleaning staff') | (new_data['occyp']=='Cooking staff') | (new_data['occyp']=='Drivers') | (new_data['occyp']=='Laborers')
new_data.loc[(new_data['occyp']=='Accountants') | (new_data['occyp']=='Core staff') | (new_data['occyp']=='HR staff') | (new_data['occyp']=='Medicine staff'
new_data.loc[(new_data['occyp']=='Managers') | (new_data['occyp']=='High skill tech staff') | (new_data['occyp']=='IT staff'),'occyp']='hightecwk'
print(new_data['occyp'].value_counts())
```

```
Laborwk      10496
officewk     10183
hightecwk     4455
Name: occyp, dtype: int64
```

```
(new_data['occyp']=='Low-skill Laborers') | (new_data['occyp']=='Security staff') | (new_data['occyp']=='Waiters/barmen staff'),'occyp']='Laborwk'
 (new_data['occyp']=='Private service staff') | (new_data['occyp']=='Realty agents') | (new_data['occyp']=='Sales staff') | (new_data['occyp']=='Secretaries
```

```
Laborwk      10496
officewk     10183
hightecwk     4455
Name: occyp, dtype: int64
```

Categorize types of occupation

Because of the various types of occupations, narrowed down to 3

03

Model Building / Test & Evaluation

Address Imbalance Dataset - SMOTE

New dataset :
Define Y and X variables

```
new_data.columns
```

```
Index(['ID', 'Car', 'Reality', 'famsize', 'inc', 'begin_month', 'target',
       'famsizegp_1', 'famsizegp_3more', 'gp_inc_high', 'gp_inc_medium',
       'occyp_hightecwk', 'occyp_officewk'],
      dtype='object')
```

```
Y = new_data['target']
X = new_data[['Car','Reality',
              'famsizegp_1', 'famsizegp_3more',
              'gp_inc_high', 'gp_inc_medium',
              'occyp_hightecwk', 'occyp_officewk']]
```

Addressing imbalanced datasets by oversampling the minority class.

Synthetic Minority Oversampling Technique (SMOTE)

```
from imblearn.over_sampling import SMOTE
import itertools
```

```
Y = Y.astype('int')
oversample = SMOTE()
X_balance,Y_balance = oversample.fit_resample(X, Y)
X_balance = pd.DataFrame(X_balance, columns = X.columns)
```

Logistic Regression - Confusion Matrix

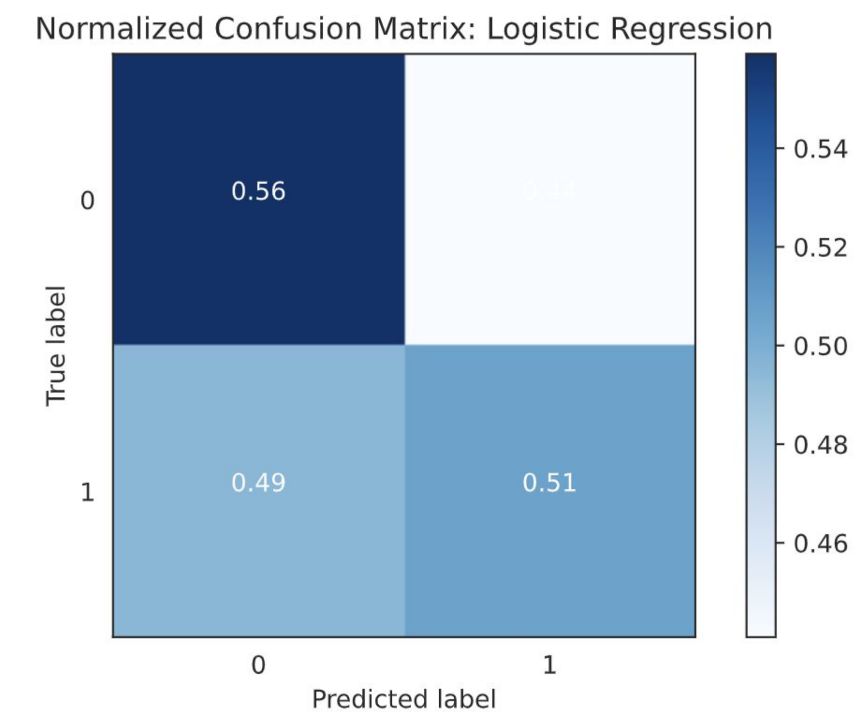
Split the data into train & test

```
[ ]  X_train, X_test, y_train, y_test = train_test_split(X_balance,Y_balance,
                                                          stratify=Y_balance, test_size=0.3,
                                                          random_state = 10000)
```

Logistic Regression

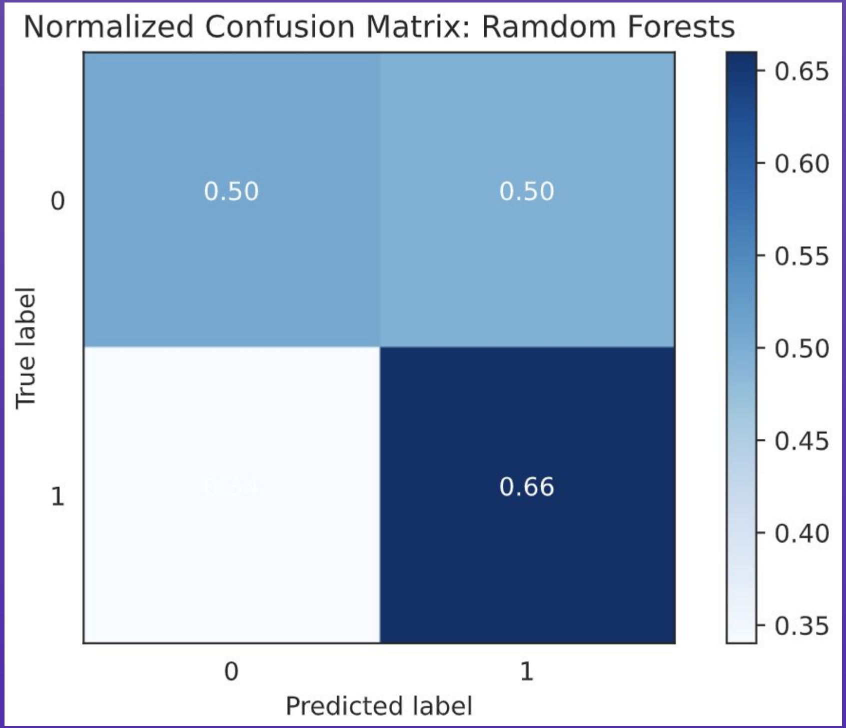
```
model = LogisticRegression(C=0.8,
                           random_state=0,
                           solver='lbfgs')
model.fit(X_train, y_train)
y_predict = model.predict(X_test)

print('Accuracy Score is {:.5}'.format(accuracy_score(y_test, y_predict)))
print(pd.DataFrame(confusion_matrix(y_test,y_predict)))

sns.set_style('white')
class_names = ['0','1']
plot_confusion_matrix(confusion_matrix(y_test,y_predict),
                      classes= class_names, normalize = True,
                      title='Normalized Confusion Matrix: Logistic Regression')
```

Logistic Regression - Confusion Matrix




```
Accuracy Score is 0.53253
        0       1
0    3958    3121
1    3497    3581
[[0.55911852 0.44088148]
 [0.49406612 0.50593388]]
```

Random Forest Classifier

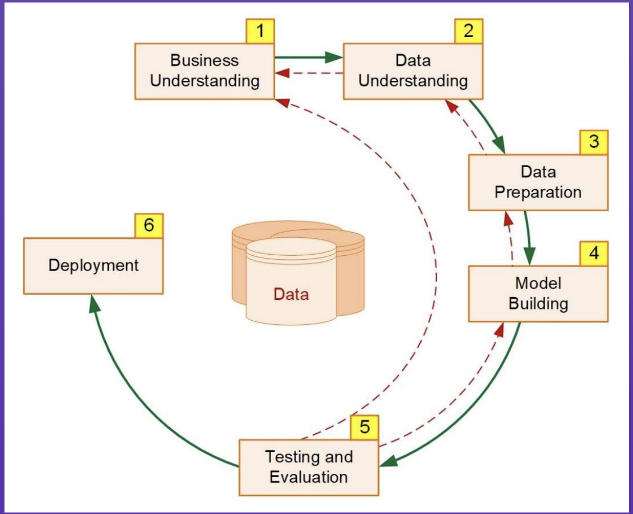



```
Accuracy Score is 0.58226
      0     1
0  3573  3506
1  2408  4670
[[0.50473231 0.49526769]
 [0.3402091  0.6597909 ]]
```

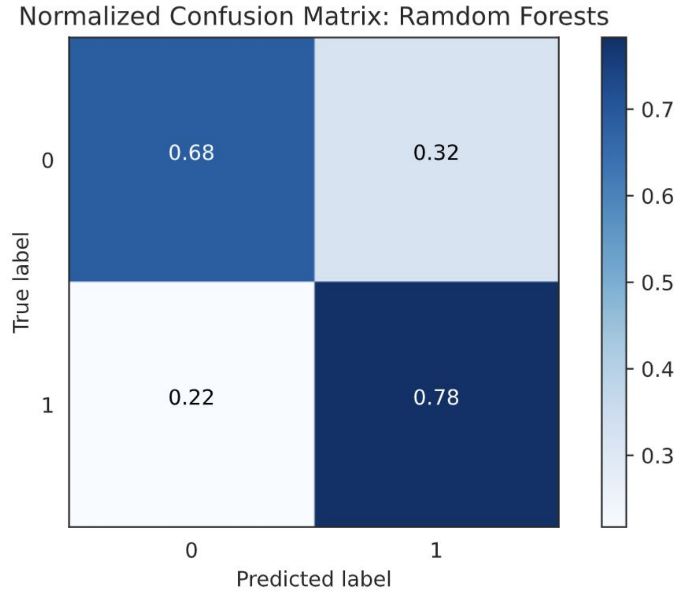
04

Development

Development - Improvements




```
Accuracy Score is 0.73356
       0     1
0   4844  2235
1   1537  5541
[[0.68427744 0.31572256]
 [0.21715174 0.78284826]]
```



After evaluating the result, we concluded that the results are invalid.

In order to improve our accuracy score,
we can revisit the steps from 1 to 4 on CRISP-DM before deployment.

Summary

Credit Card Approval Prediction	By analyzing the credit card applicants' data
CRISP - DM	Cross Industry Standard Process for Data Mining Methodology
Data Preparation	Cleaning / New Dataframe / feature engineering
Model Building	Logistic Regression & Random Forest
Evaluation	Improving accuracy score Constraints : Time limitation

Thank You
Q & A