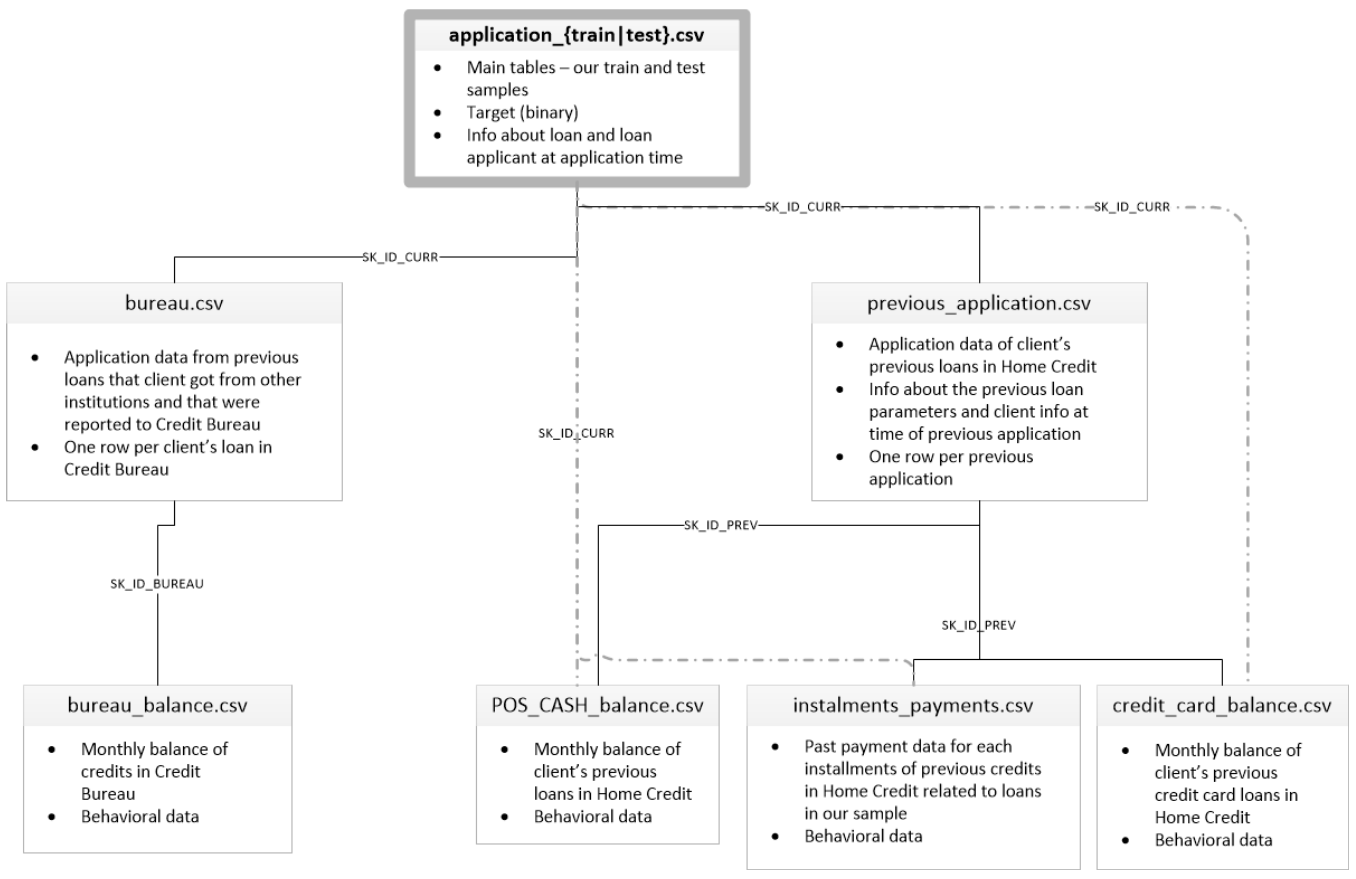
Project 1 : House Loan Data

The Core Data Set files is used as well as has been broken into Small Files for various operation.
Breakup of the files are and its details are listed below for better understanding.

- **HomeCredit_columns_description.csv**
    1. This file contains descriptions for the columns in the various data files.

- **application_{train|test}.csv**
    1. This is the main table, broken into two files for Train (with TARGET) and Test (without TARGET).
    2. Static data for all applications. One row represents one loan in our data sample.
- **bureau.csv**
    1. All client's previous credits provided by other financial institutions that were reported to Credit Bureau (for clients who have a loan in our sample).
    2. For every loan in our sample, there are as many rows as number of credits the client had in Credit Bureau before the application date.
- **bureau_balance.csv**
    1. Monthly balances of previous credits in Credit Bureau.
    2. This table has one row for each month of history of every previous credit reported to Credit Bureau – i.e the table has (#loans in sample * # of relative previous credits * # of months where we have some history observable for the previous credits) rows.
- **POS_CASH_balance.csv**
    1. Monthly balance snapshots of previous POS (point of sales) and cash loans that the applicant had with Home Credit.
    2. This table has one row for each month of history of every previous credit in Home Credit (consumer credit and cash loans) related to loans in our sample – i.e. the table has (#loans in sample * # of relative previous credits * # of months in which we have some history observable for the previous credits) rows.
- **credit_card_balance.csv**
    1. Monthly balance snapshots of previous credit cards that the applicant has with Home Credit.
    2. This table has one row for each month of history of every previous credit in Home Credit (consumer credit and cash loans) related to loans in our sample – i.e. the table has (#loans in sample * # of relative previous credit cards * # of months where we have some history observable for the previous credit card) rows.
- **previous_application.csv**
    1. All previous applications for Home Credit loans of clients who have loans in our sample.
    2. There is one row for each previous application related to loans in our data sample.
- **installments_payments.csv**
    - Repayment history for the previously disbursed credits in Home Credit related to the loans in our sample.
    - There is a) one row for every payment that was made plus b) one row each for missed payment.
    - One row is equivalent to one payment of one installment OR one installment corresponding to one payment of one previous Home Credit credit related to loans in our sample.



application_{train|test}.csv

- Main tables – our train and test samples
- Target (binary)
- Info about loan and loan applicant at application time

bureau.csv

- Application data from previous loans that client got from other institutions and that were reported to Credit Bureau
- One row per client's loan in Credit Bureau

previous_application.csv

- Application data of client's previous loans in Home Credit
- Info about the previous loan parameters and client info at time of previous application
- One row per previous application

bureau_balance.csv

- Monthly balance of credits in Credit Bureau
- Behavioral data

POS_CASH_balance.csv

- Monthly balance of client's previous loans in Home Credit
- Behavioral data

instalments_payments.csv

- Past payment data for each installments of previous credits in Home Credit related to loans in our sample
- Behavioral data

credit_card_balance.csv

- Monthly balance of client's previous credit card loans in Home Credit
- Behavioral data

Real World / Business Objectives and Constraints:

Before starting any problem, it is better to lay the constraints. So that we can change the modelling process based on the constraints.

1. No strict latency constraints.
Given the loan application data, we don't have to predict whether the applicant is going to repay loan in milli seconds or seconds. We can take couple of minutes to predict. Even 1 hour should be fine. Since we don't have strict latency constraint, we can use Ensemble models like Random forest, Xgboost etc.

2. Predict the probability of capability of each applicant of repaying a loan.
Suppose there are two applicants A, B whose probability values of repaying loan is 0.6 and 0.9, since the general threshold each machine learning model considers is 0.5. Both the applicants are labeled as 'Repaying the loan'. But applicant B is more likely to repay the loan when compared to applicant A. If we predict the probabilities we can set the threshold like 0.8, 0.9 based on the business requirements.

3. The cost of a mis-classification is very high.
Let's say, for an applicant A our model labelled as 'Repaying the loan'. So the organization sanctioned the loan for that applicant. But the applicant for some reason is not able to pay the loan. This type of scenarios is loss to the organization. Hence we should come up with a model which can reduce the mis-classifications as much as possible.

4. Interpretability is partially important.
As long as our model predicts well on test data, we don't need to care much about interpretability. Since our problem is not related to medical domain(Interpretability is important- cancer detection), it is fine if we can give some form of interpretability like feature importances.

Performance Metric:
In this problem, the data is imbalanced. So we can't use accuracy as a error metric. When data is imbalanced we can use Log loss, F1-score and AUC. Here we are sticking to AUC which can handle imbalanced datasets.

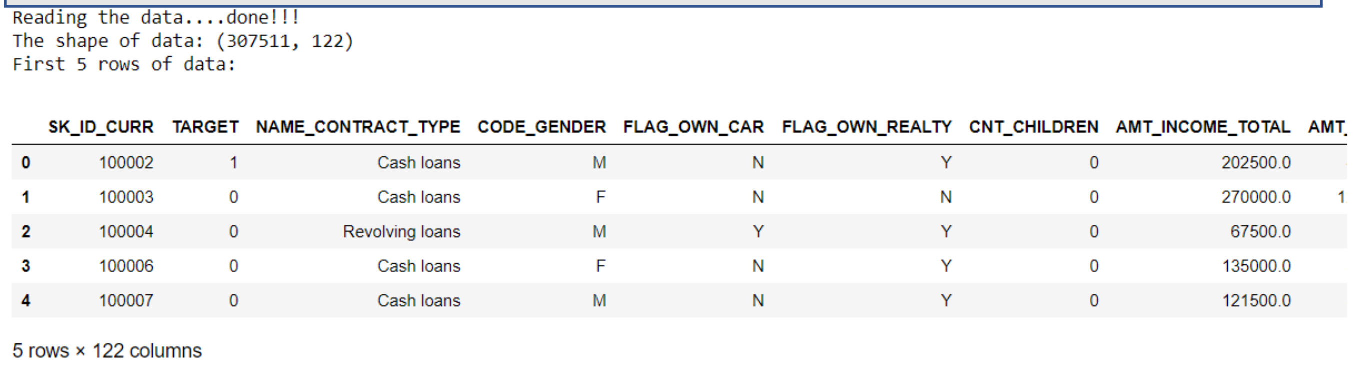
Area Under Curve (AUC)
An ROC curve is the most commonly used way to visualize the performance of a binary classifier, and AUC is (arguably) the best way to summarize its performance in a single number.
Confusion Matrix (To get an overview of complete predictions)

Exploratory Data Analysis:
At first, import the necessary packages.

```
import pandas as pd
import sklearn
import numpy as np
import matplotlib.pyplot as plt
import os
import warnings
import seaborn as snsfrom sklearn.preprocessing import OneHotEncoder
from sklearn.impute import SimpleImputer
from sklearn.pipeline import Pipeline
from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import StandardScaler
from sklearn.svm import LinearSVC
from sklearn.metrics import roc_auc_score
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import roc_auc_score
from sklearn.calibration import CalibratedClassifierCV
from sklearn.metrics import confusion_matrix
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score
from sklearn.linear_model import SGDClassifierimport plotly.offline as py
import plotly.graph_objs as gofrom plotly.offline import init_notebook_mode, iplot
from sklearn.model_selection import train_test_split
init_notebook_mode(connected=True)import cufflinks as cf
cf.go_offline()import pickle
import gc
import lightgbm as lgbwarnings.filterwarnings('ignore')
%matplotlib inline
```

Load the data from given csv file into a pandas dataframe.

```
print('Reading the data....', end='')
application = pd.read_csv('application_train.csv')
print('done!!!')
print('The shape of data:',application.shape)
print('First 5 rows of data:')
application.head()
```



```
Reading the data....done!!!
The shape of data: (307511, 122)
First 5 rows of data:
```

	SK_ID_CURR	TARGET	NAME_CONTRACT_TYPE	CODE_GENDER	FLAG_OWN_CAR	FLAG_OWN_REALTY	CNT_CHILDREN	AMT_INCOME_TOTAL	AMT_
0	100002	1	Cash loans	M	N	Y	0	202500.0	
1	100003	0	Cash loans	F	N	N	0	270000.0	1
2	100004	0	Revolving loans	M	Y	Y	0	67500.0	
3	100006	0	Cash loans	F	N	Y	0	135000.0	
4	100007	0	Cash loans	M	N	Y	0	121500.0	

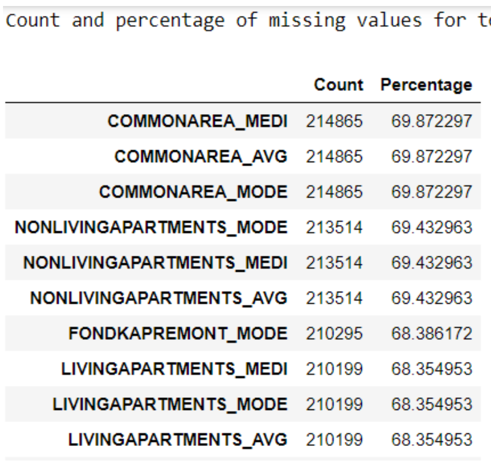
5 rows × 122 columns

We are using ‘application_train.csv’ file :

1. This dataset consists of 307511 rows and 122 columns.
2. Each row has unique id ‘SK_ID_CURR’ and the output label is in the ‘TARGET’ column.
3. TARGET indicating 0: the loan was repaid or 1: the loan was not repaid.
4. The description of each column can be found in the file ‘**HomeCredit_columns_description.csv’**

Let us check for missing values in each column.

```
count = application.isnull().sum().sort_values(ascending=False)
percentage =
((application.isnull().sum()/len(application)*100)).sort_values(ascending=False)missing_application =
pd.concat([count, percentage], axis=1, keys=['Count','Percentage'])
print('Count and percentage of missing values for top 20 columns:')
missing_application.head(20)
```



```
Count and percentage of missing values for top 20 columns:
```

	Count	Percentage
COMMONAREA_MEDI	214865	69.872297
COMMONAREA_AVG	214865	69.872297
COMMONAREA_MODE	214865	69.872297
NONLIVINGAPARTMENTS_MODE	213514	69.432963
NONLIVINGAPARTMENTS_MEDI	213514	69.432963
NONLIVINGAPARTMENTS_AVG	213514	69.432963
FONDKAPREMONT_MODE	210295	68.386172
LIVINGAPARTMENTS_MEDI	210199	68.354953
LIVINGAPARTMENTS_MODE	210199	68.354953
LIVINGAPARTMENTS_AVG	210199	68.354953

Observations:
1. There are lot of missing values in each column.
2. We need to somehow handle these missing values, we will see how to handle later in the case study.

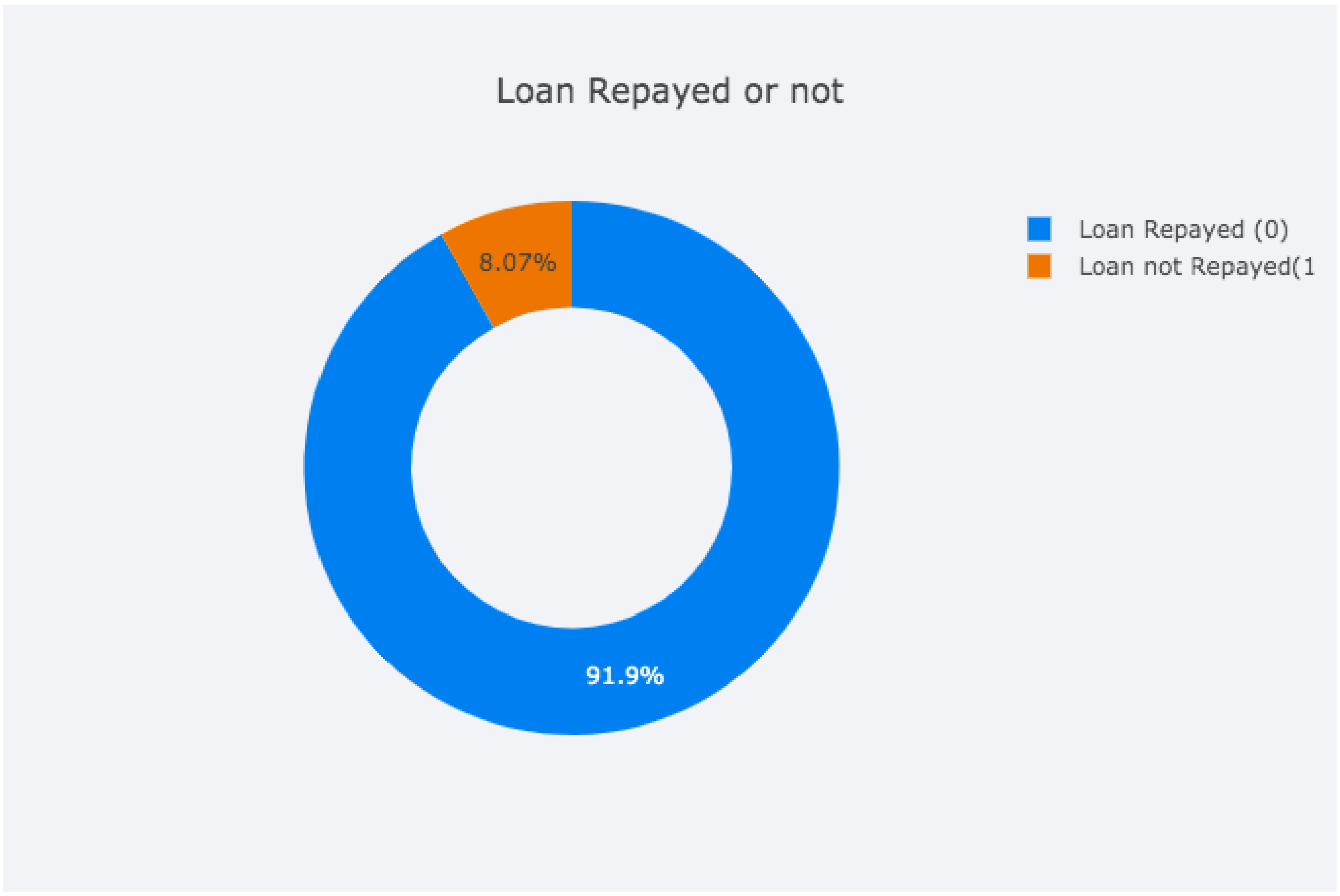
Let’s check for duplicate data:

```
columns_without_id = [col for col in application.columns if col!='SK_ID_CURR']#Checking for
duplicates in the data.
application[application.duplicated(subset = columns_without_id, keep=False)]
print('The no of duplicates in the data:',application[application.duplicated(subset = columns_without_id,
keep=False)].shape[0])
```

```
The no of duplicates in the data: 0
```

Let’s check the distribution of data points among output class.

```
cf.set_config_file(theme='polar')

contract_val = application['NAME_CONTRACT_TYPE'].value_counts()
contract_df = pd.DataFrame({'labels': contract_val.index,
'values': contract_val.values
})

contract_df.iplot(kind='pie',labels='labels',values='values', title='Types of Loan', hole = 0.6)
```

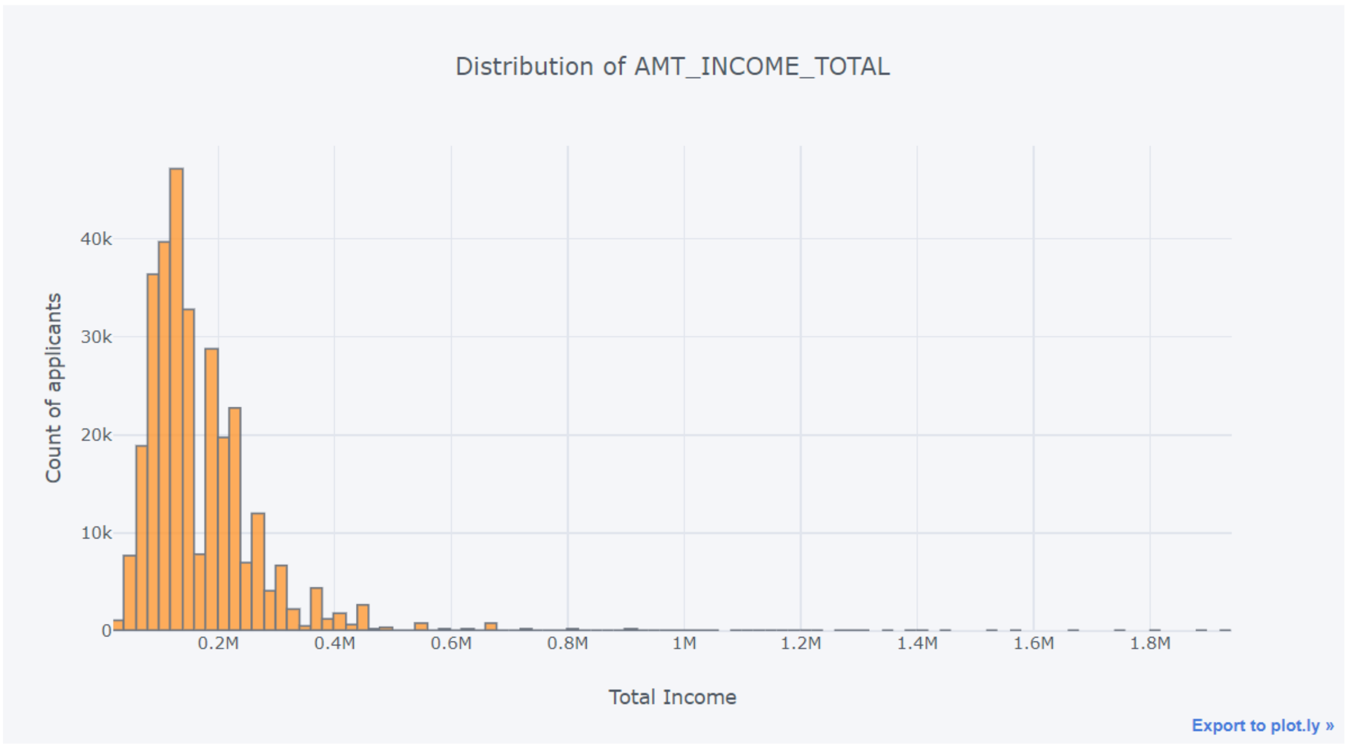


Observations:

1. The data is imbalanced (91.9%(Loan repayed-0) and 8.07%(Loan not repayed-1)) and we need to handle this problem.

Distribution of AMT_INCOME_TOTAL.

```
application[application['AMT_INCOME_TOTAL'] <
2000000]['AMT_INCOME_TOTAL'].iplot(kind='histogram', bins=100,
xTitle = 'Total Income', yTitle ='Count of applicants',
title='Distribution of AMT_INCOME_TOTAL')
```



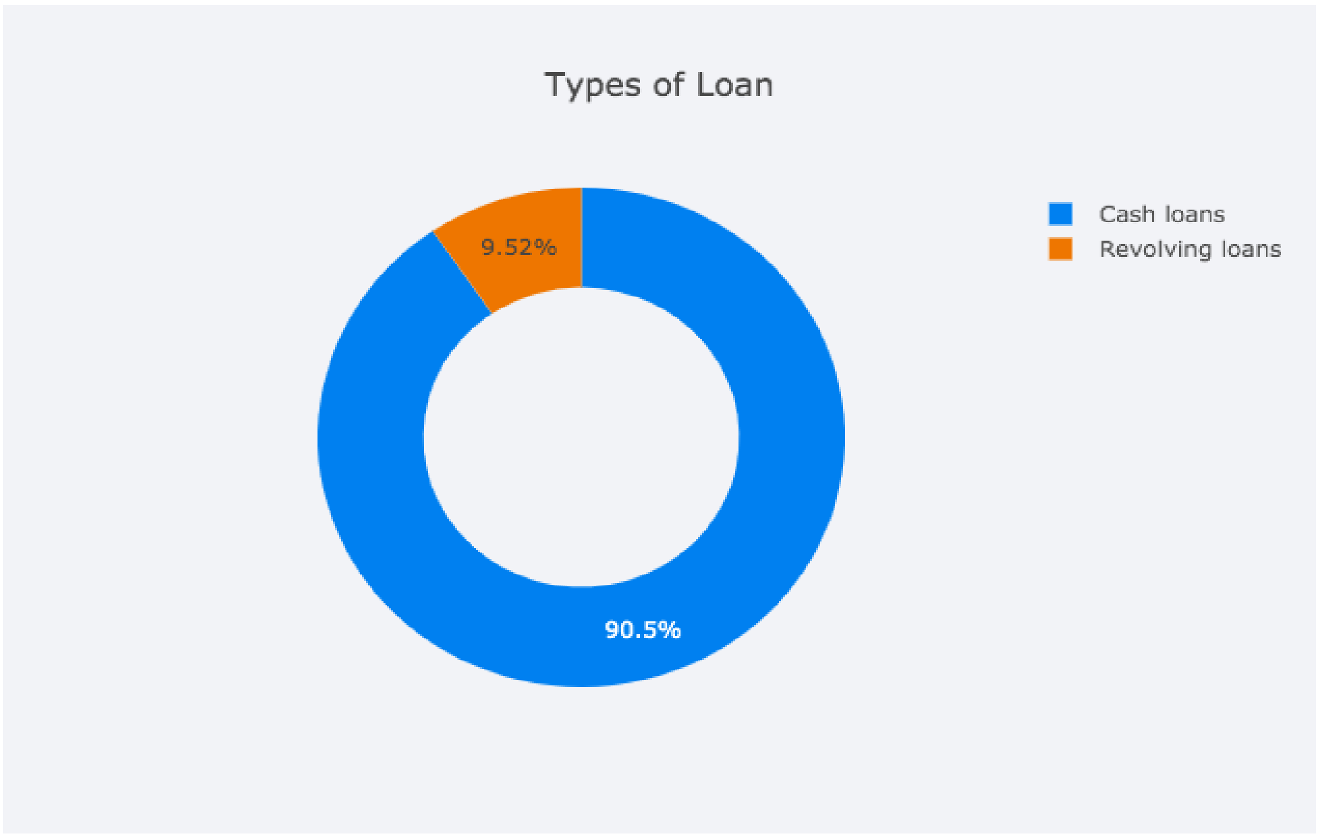
```
(application[application['AMT_INCOME_TOTAL'] >
1000000]['TARGET'].value_counts())/len(application[application['AMT_INCOME_TOTAL'] >
1000000])*100
```

```
0    94.8
1     5.2
Name: TARGET, dtype: float64
```

Observations:
1. The distribution is right skewed and there are extreme values, we can apply log distribution.
2. People with high income(>1000000) are likely to repay the loan.

Types of loan available.

```
cf.set_config_file(theme='polar')
contract_val = application['NAME_CONTRACT_TYPE'].value_counts()
contract_df = pd.DataFrame({'labels': contract_val.index,
'values': contract_val.values
})
contract_df.iplot(kind='pie',labels='labels',values='values', title='Types of Loan', hole = 0.6)
```

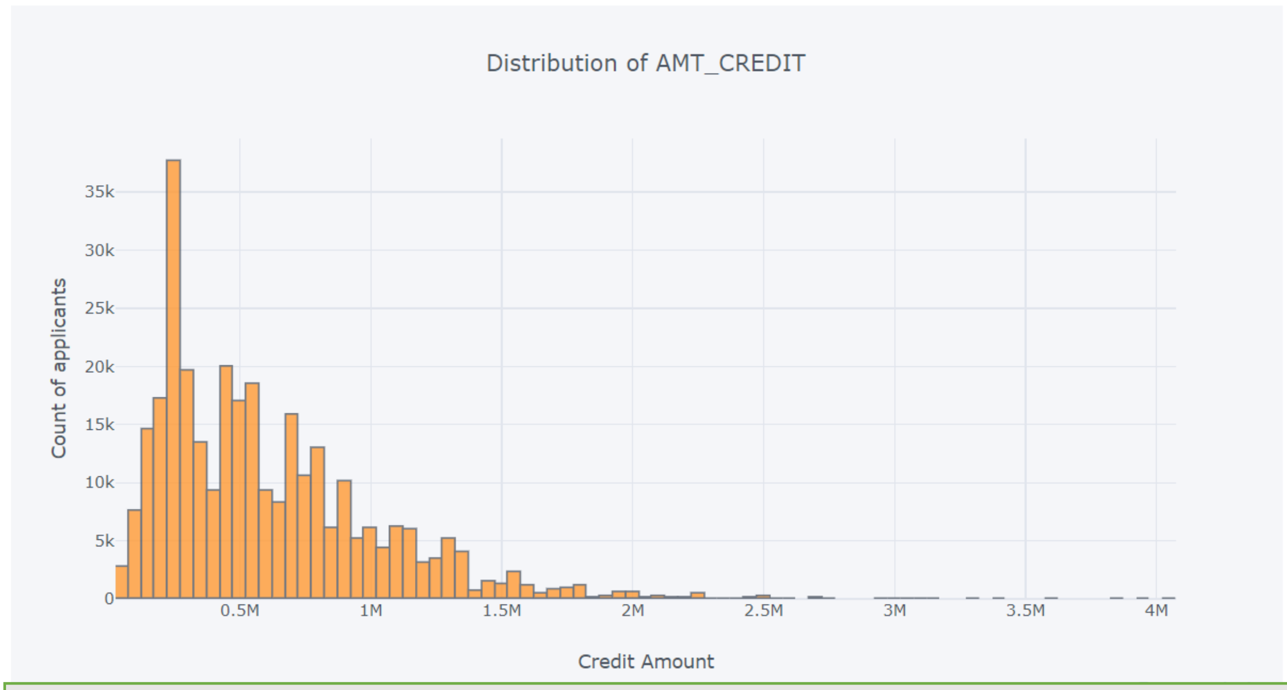


Observations:

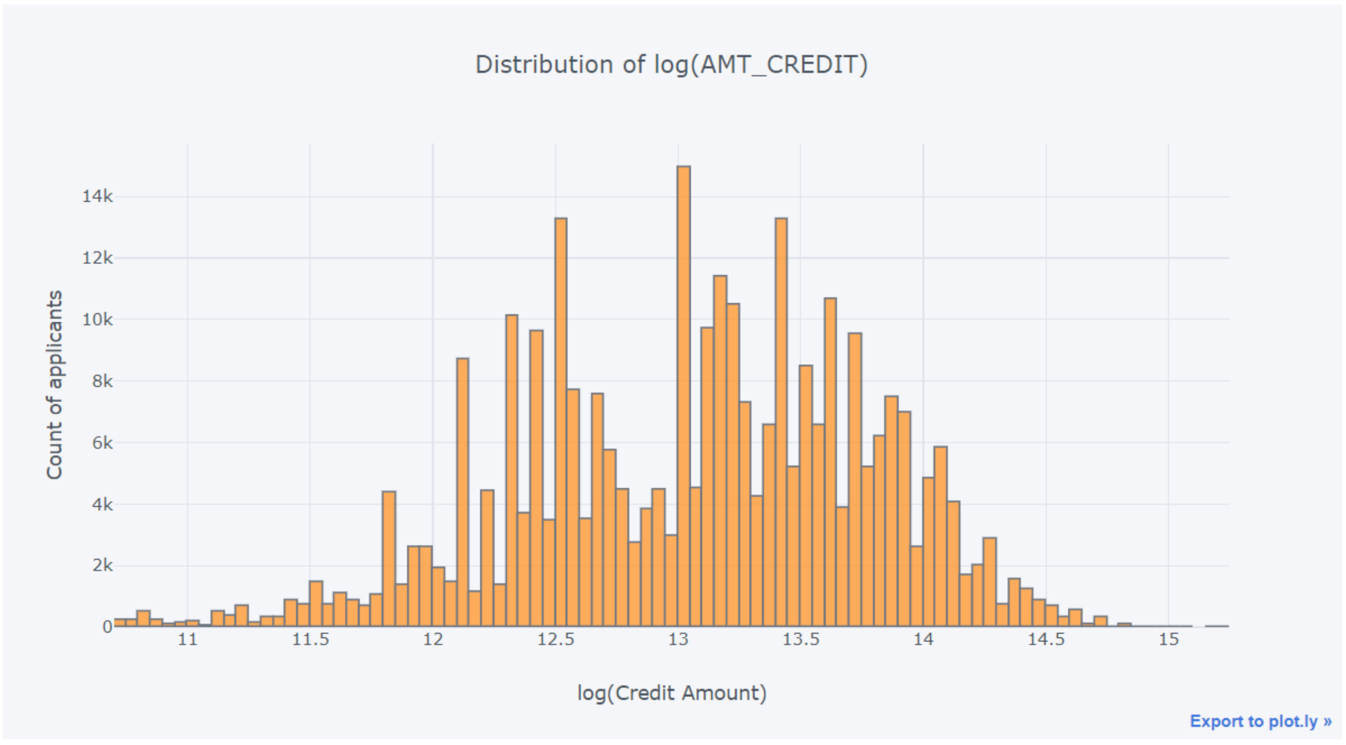
- Many people are willing to take cash loan than revolving loan

Distribution of AMT_CREDIT.

```
application['AMT_CREDIT'].iplot(kind='histogram', bins=100,
xTitle = 'Credit Amount',yTitle ='Count of applicants',
title='Distribution of AMT_CREDIT')
```



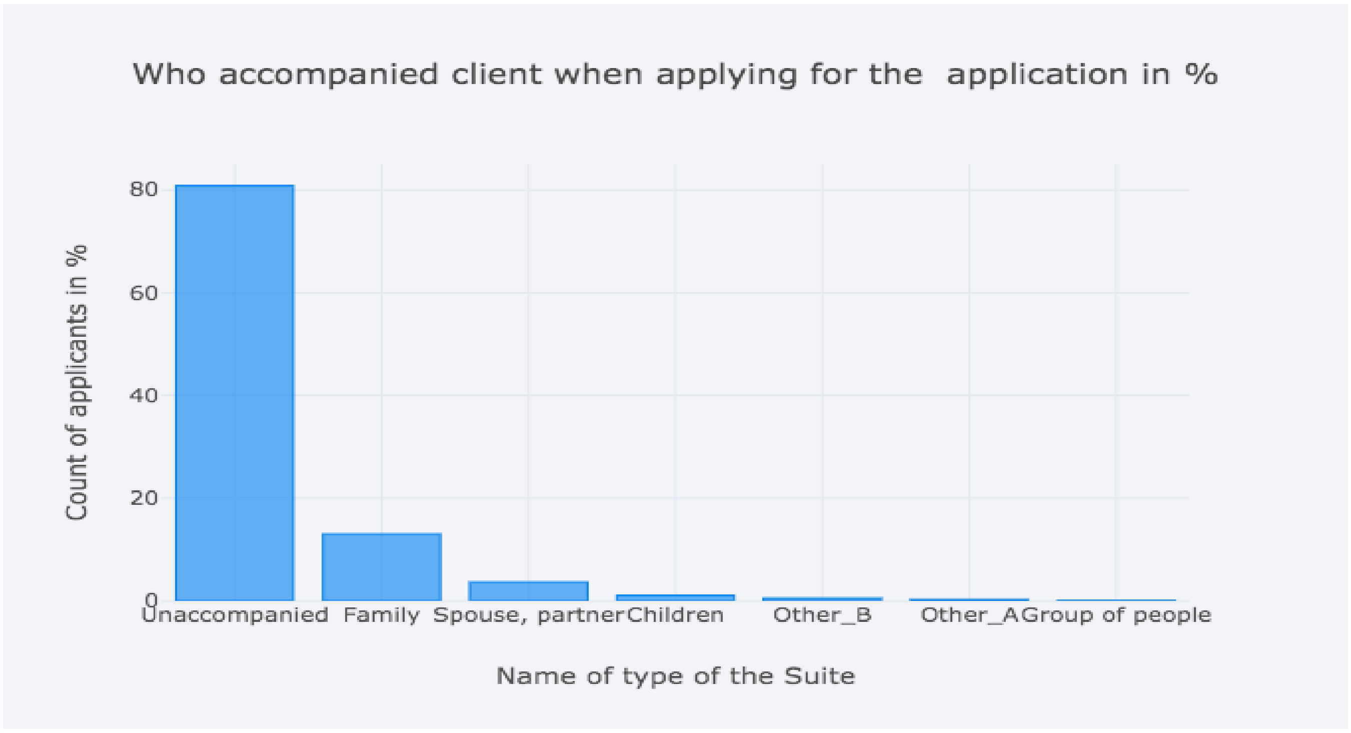
```
np.log(application['AMT_CREDIT']).iplot(kind='histogram', bins=100,
xTitle = 'log(Credit Amount)',yTitle ='Count of applicants',
title='Distribution of log(AMT_CREDIT)')
```



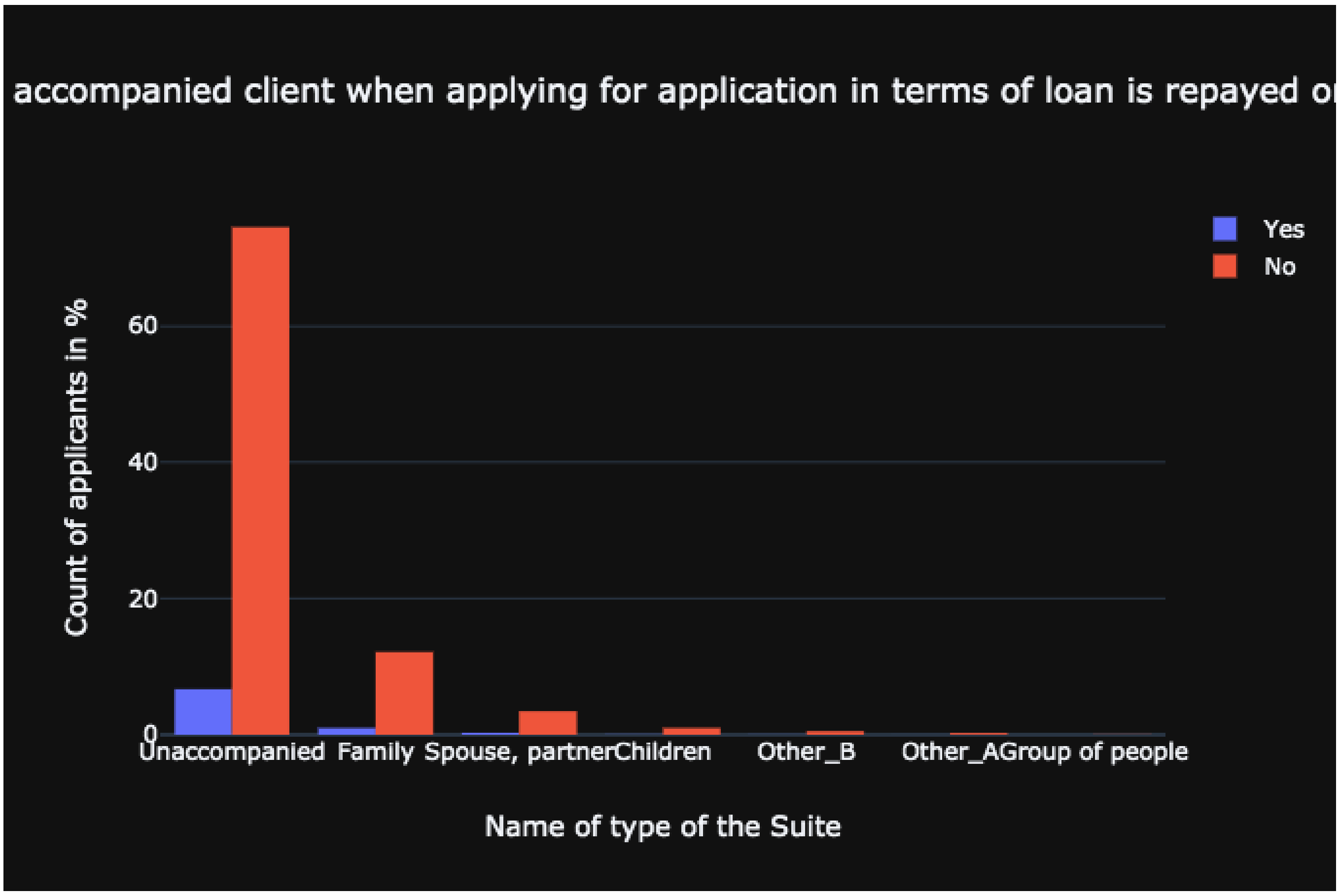
Observations:
1. People who are taking credit for large amount are very likely to repay the loan.
2. Originally the distribution is right skewed, we used log transformation to make it normal distributed.

Distribution of Name of type of the Suite in terms of loan is repayed or not.

```
cf.set_config_file(theme='polar')
suite_val = (application['NAME_TYPE_SUITE'].value_counts()/len(application))*100
suite_val.iplot(kind='bar', xTitle = 'Name of type of the Suite',
yTitle='Count of applicants in %',
title='Who accompanied client when applying for the application in % ')
```

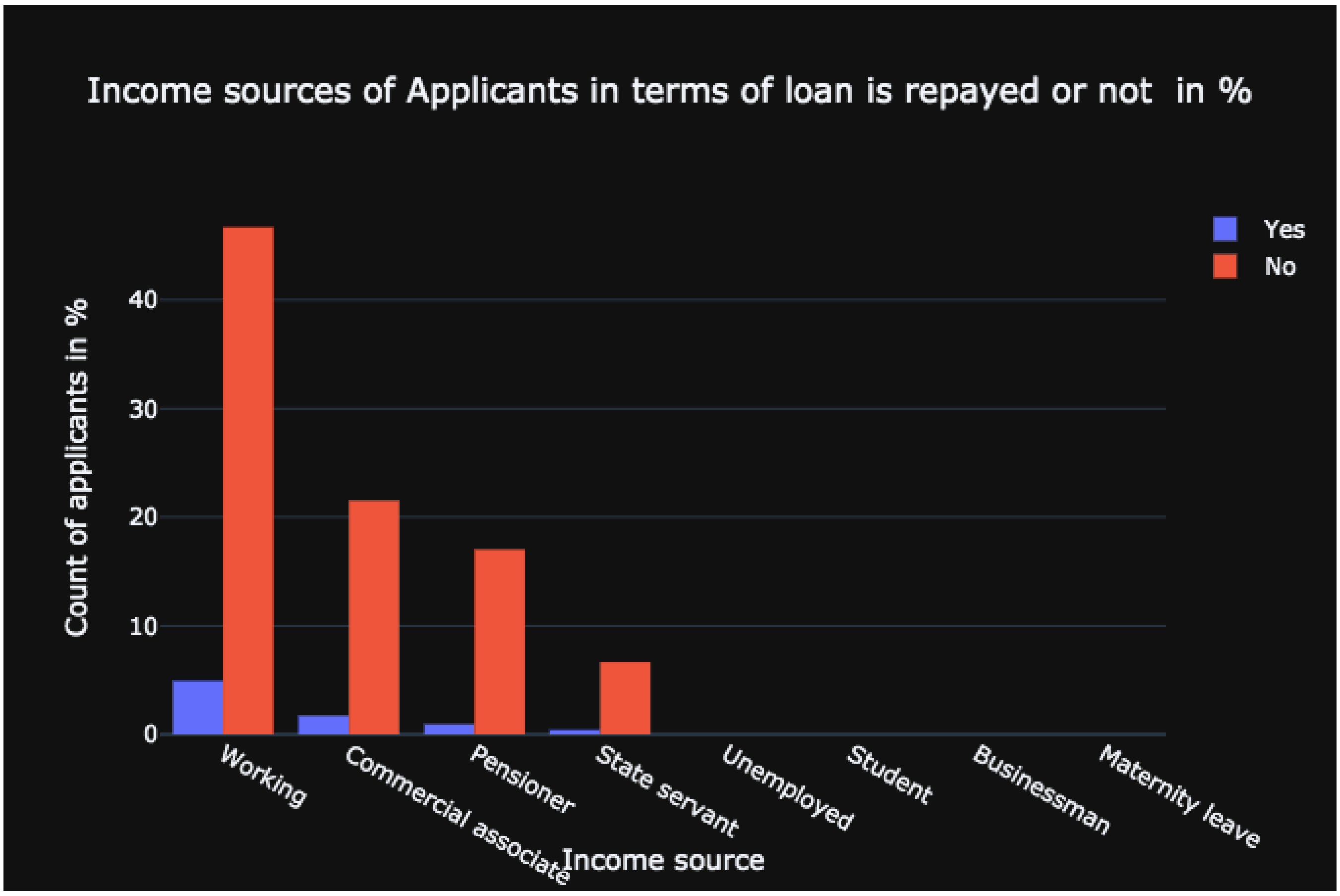


```
suite_val = application['NAME_TYPE_SUITE'].value_counts()

suite_val_y0 = []
suite_val_y1 = []

for val in suite_val.index:
    suite_val_y1.append(np.sum(application['TARGET']
[application['NAME_TYPE_SUITE']==val] == 1))
    suite_val_y0.append(np.sum(application['TARGET']
[application['NAME_TYPE_SUITE']==val] == 0))

data = [go.Bar(x = suite_val.index, y = ((suite_val_y1 / suite_val.sum()) * 100), name='Yes' ),
        go.Bar(x = suite_val.index, y = ((suite_val_y0 / suite_val.sum()) * 100), name='No' )]

layout = go.Layout(
    title = "Who accompanied client when applying for the  application in terms of loan is repayed or not in %",
    xaxis=dict(
        title='Name of type of the Suite',
       ),
    yaxis=dict(
        title='Count of applicants in %',
        )
)
```



Distribution of Income sources of Applicants in terms of loan is repayed or not.

```
income_val = application['NAME_INCOME_TYPE'].value_counts()

income_val_y0 = []
income_val_y1 = []

for val in income_val.index:
    income_val_y1.append(np.sum(application['TARGET']
[application['NAME_INCOME_TYPE']==val] == 1))
    income_val_y0.append(np.sum(application['TARGET']
[application['NAME_INCOME_TYPE']==val] == 0))

data = [go.Bar(x = income_val.index, y = ((income_val_y1 / income_val.sum()) * 100), name='Yes' ),
        go.Bar(x = income_val.index, y = ((income_val_y0 / income_val.sum()) * 100), name='No' )]

layout = go.Layout(
    title = "Income sources of Applicants in terms of loan is repayed or not  in %",
    xaxis=dict(
```

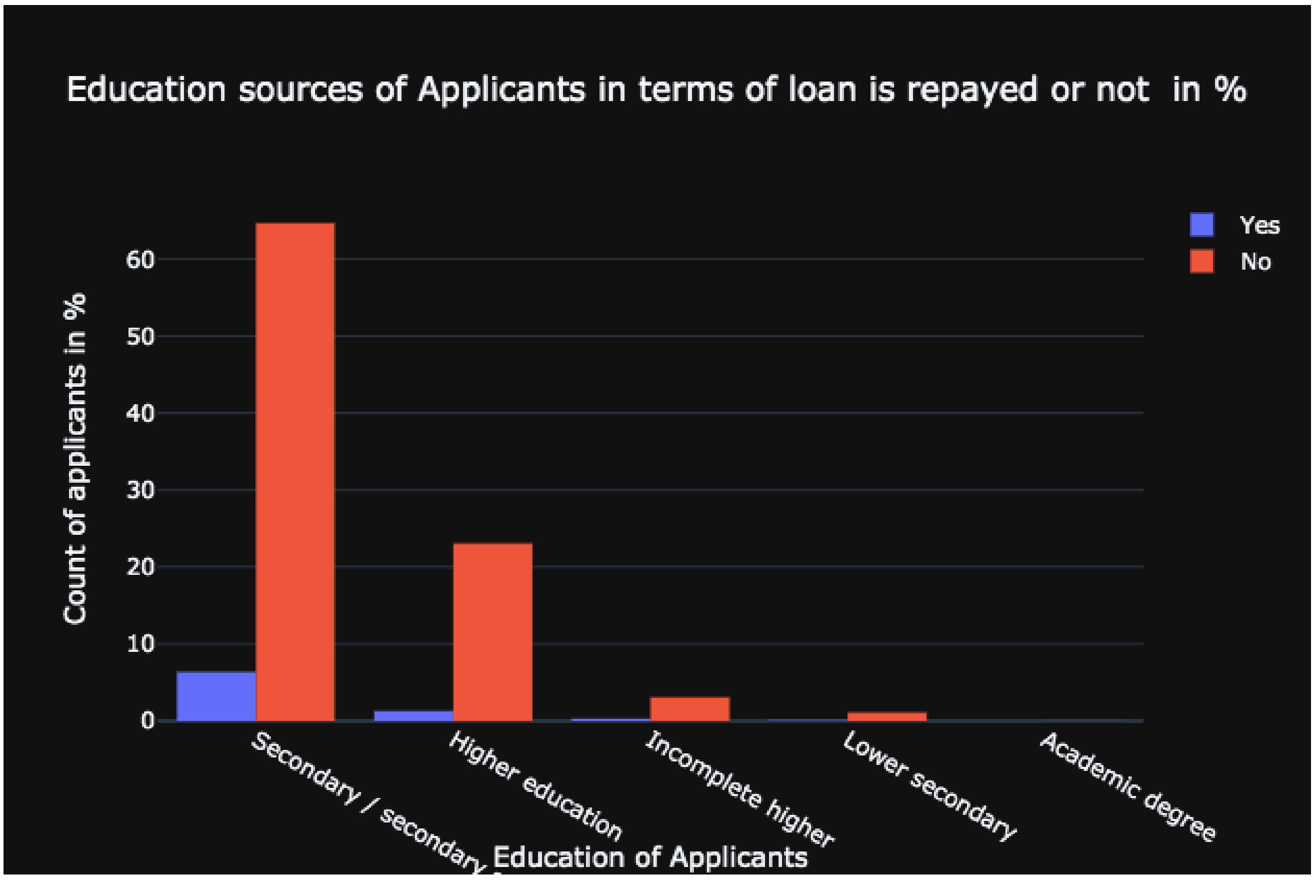


Observations:

1. All the Students and Businessman are repaying loan

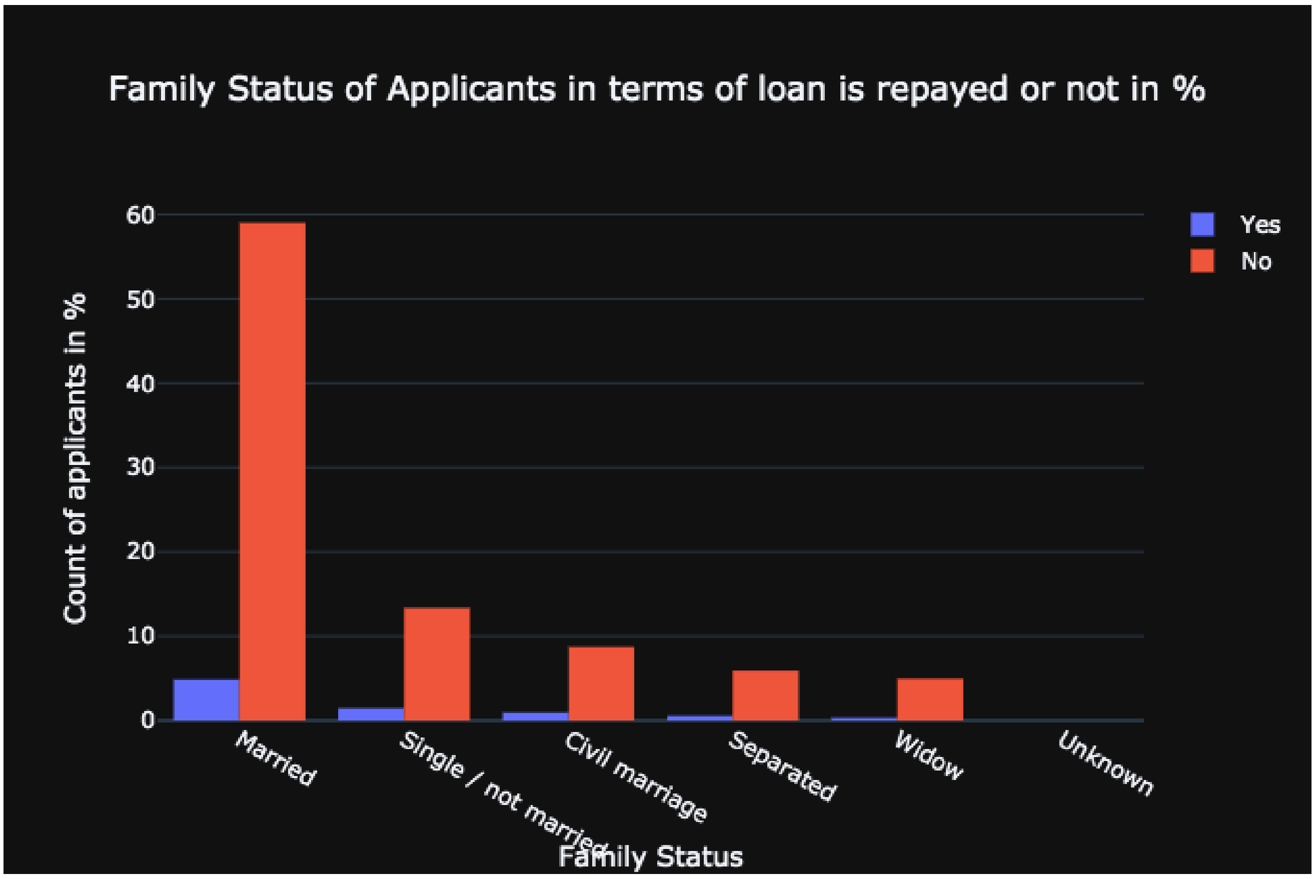
Distribution of Education of Applicants in terms of loan is repayed or not.

```
education_val = application['NAME_EDUCATION_TYPE'].value_counts()

education_val_y0 = []
education_val_y1 = []

for val in education_val.index:
    education_val_y1.append(np.sum(application['TARGET']
[application['NAME_EDUCATION_TYPE']==val] == 1))
    education_val_y0.append(np.sum(application['TARGET']
[application['NAME_EDUCATION_TYPE']==val] == 0))

data = [go.Bar(x = education_val.index, y = ((education_val_y1 / education_val.sum()) * 100),
name='Yes' ),
        go.Bar(x = education_val.index, y = ((education_val_y0 / education_val.sum()) * 100),
name='No' )]
```



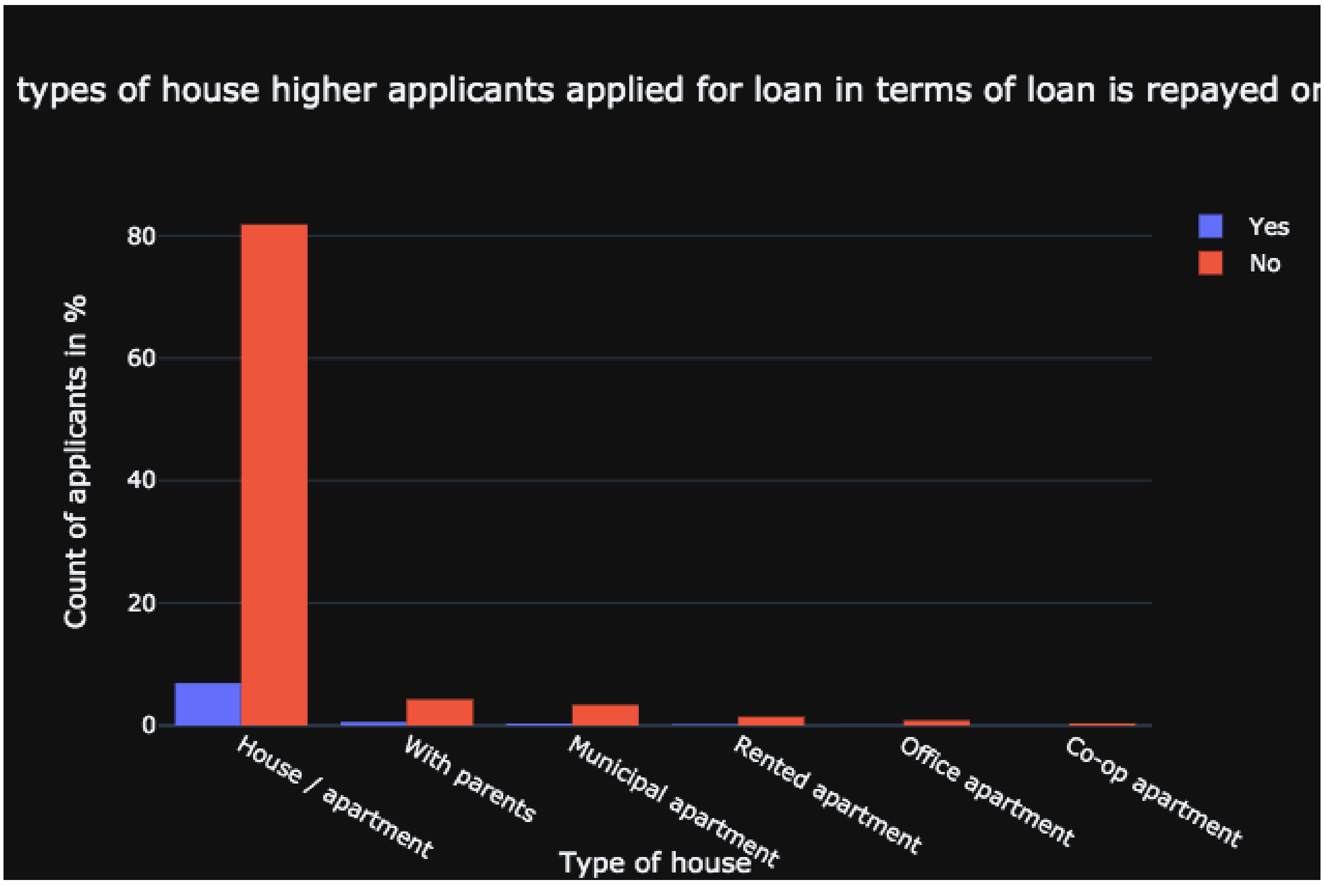
Observations:
1. People with Academic Degree are more likely to repay the loan(Out of 164, only 3 applicants are not able to repay)

Distribution of Family status of Applicants in terms of loan is repayed or not.



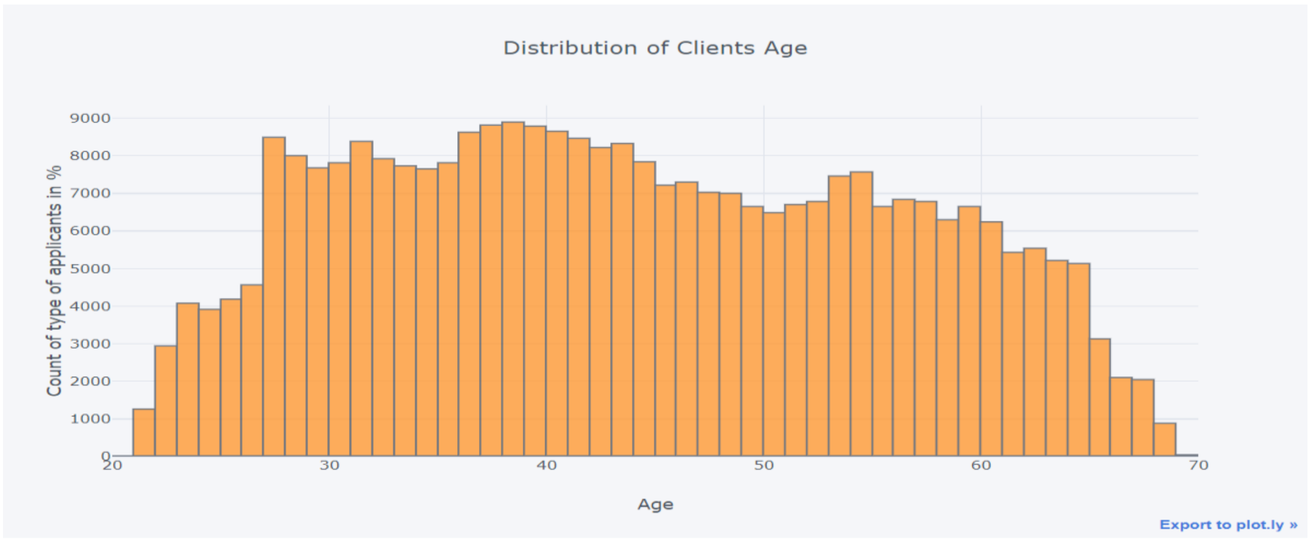
Observations:
1. Widows are more likely to repay the loan when compared to appliants with the other family statuses.

Distribution of Housing type of Applicants in terms of loan is repayed or not.



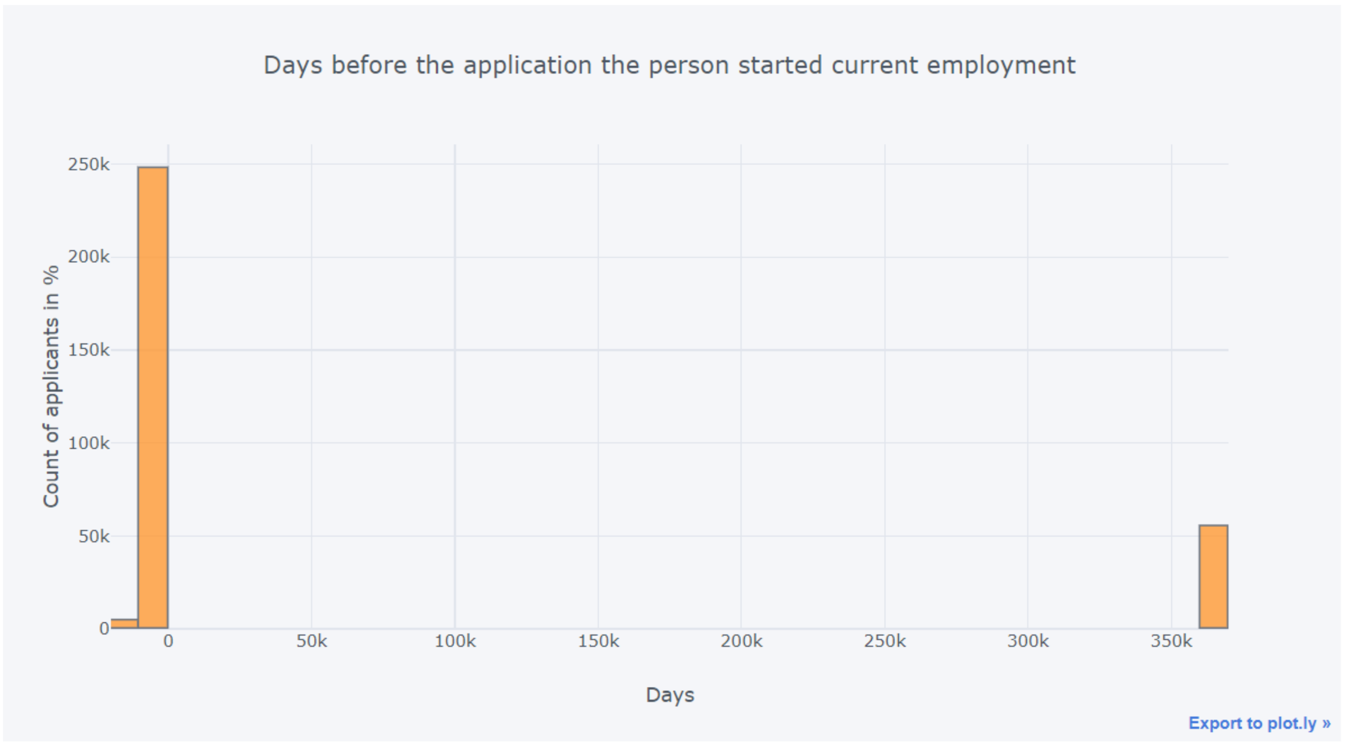
Distribution of Clients Age

```
cf.set_config_file(theme='pearl')
(application['DAYS_BIRTH']/(-365)).iplot(kind='histogram',
xTitle = 'Age', bins=50,
yTitle='Count of type of applicants in %',
title='Distribution of Clients Age')
```



Distribution of years before the application the person started current employment.

```
cf.set_config_file(theme='pearl')
(application['DAYS_EMPLOYED']).iplot(kind='histogram',
xTitle = 'Days',bins=50,
yTitle='Count of applicants in %',
title='Days before the application the person started current employment')
```



Observations:
1. The data looks strange(we have -1000.66 years(-365243 days) of employment which is impossible) looks like there is data entry error.

```
error = application[application['DAYS_EMPLOYED'] == 365243]
print('The no of errors are :', len(error))
(error['TARGET'].value_counts()/len(error))*100
```

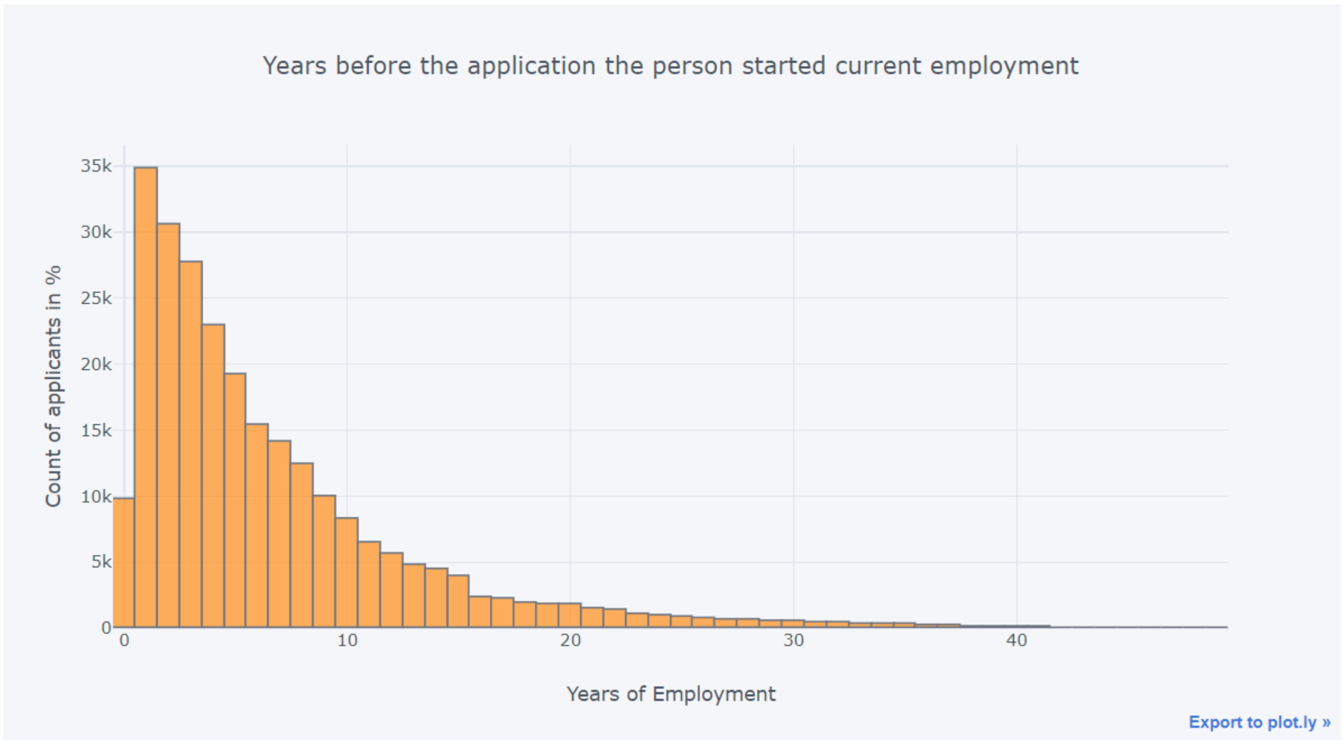
```
The no of errors are : 55374

0    94.600354
1     5.399646
Name: TARGET, dtype: float64
```

The error are default to 5.4%, so we need to handle this error

```
# Create an error flag column
application['DAYS_EMPLOYED_ERROR'] = application["DAYS_EMPLOYED"] == 365243# Replace the error values with nan
application['DAYS_EMPLOYED'].replace({365243: np.nan}, inplace = True)
```

Created a seperate column 'DAYS_EMPLOYED_ERROR', which flags the error.

```
cf.set_config_file(theme='pearl')
(application['DAYS_EMPLOYED']/(-365)).iplot(kind='histogram', xTitle = 'Years of Employment',bins=50,
yTitle='Count of applicants in %',
title='Years before the application the person started current employment')
```



```
application[application['DAYS_EMPLOYED']>(-
365*2)]['TARGET'].value_counts()/sum(application['DAYS_EMPLOYED']>(-365*2))
```

```
0    0.887924
1    0.112076
Name: TARGET, dtype: float64
```

Observations:
1. The applicants with less than 2 years of employment are less likely to repay the loan.

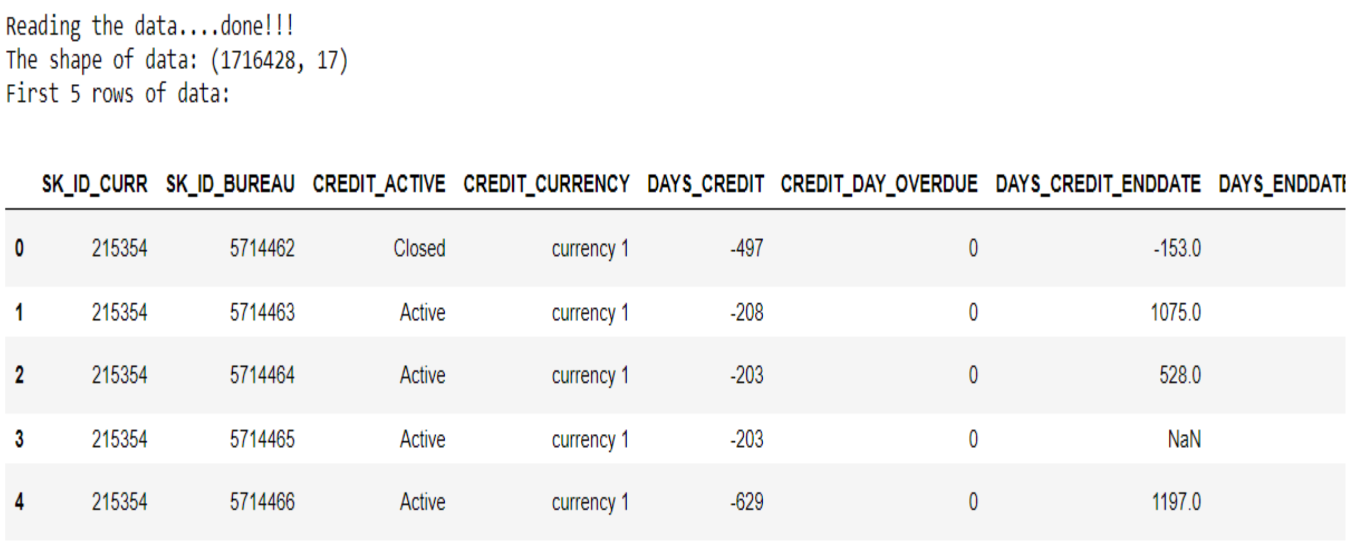
Data Preparation:

```
# Flag to represent when Total income is greater than Credit
application['INCOME_GT_CREDIT_FLAG'] = application['AMT_INCOME_TOTAL'] >
application['AMT_CREDIT']  #Column to represent Credit Income Percent

application['CREDIT_INCOME_PERCENT'] = application['AMT_CREDIT'] /
application['AMT_INCOME_TOTAL'].  #Column to represent Annuity Income percent

application['ANNUITY_INCOME_PERCENT'] = application['AMT_ANNUITY'] /
application['AMT_INCOME_TOTAL']  #Column to represent Credit Term

application['CREDIT_TERM'] = application['AMT_CREDIT'] / application['AMT_ANNUITY']
#Column to represent Days Employed percent in his life

application['DAYS_EMPLOYED_PERCENT'] = application['DAYS_EMPLOYED'] /
application['DAYS_BIRTH']  #Shape of Application data

print('The shape of application data:',application.shape)
```

```
The shape of application data: (307511, 128)
```

Using Bureau Data:

```
print('Reading the data....', end='')
bureau = pd.read_csv('bureau.csv')
print('done!!!')
print('The shape of data:',bureau.shape)
print('First 5 rows of data:')
```

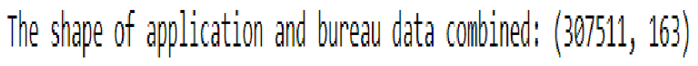


```
Reading the data....done!!!
The shape of data: (1716428, 17)
First 5 rows of data:
```

	SK_ID_CURR	SK_ID_BUREAU	CREDIT_ACTIVE	CREDIT_CURRENCY	DAYS_CREDIT	CREDIT_DAY_OVERDUE	DAYS_CREDIT_ENDDATE	DAYS_ENDDATI
0	215354	5714462	Closed	currency 1	-497	0	-153.0	
1	215354	5714463	Active	currency 1	-208	0	1075.0	
2	215354	5714464	Active	currency 1	-203	0	528.0	
3	215354	5714465	Active	currency 1	-203	0	NaN	
4	215354	5714466	Active	currency 1	-629	0	1197.0	

Joining Bureau data to Application data:

```
# Combining numerical features
grp = bureau.drop(['SK_ID_BUREAU'], axis = 1).groupby(by=['SK_ID_CURR']).mean().reset_index()
grp.columns = ['BUREAU_'+column if column !='SK_ID_CURR' else column for column in grp.columns]
application_bureau = application.merge(grp, on='SK_ID_CURR', how='left')
application_bureau.update(application_bureau[grp.columns].fillna(0))


# Combining categorical features
bureau_categorical = pd.get_dummies(bureau.select_dtypes('object'))
bureau_categorical['SK_ID_CURR'] = bureau['SK_ID_CURR']

grp = bureau_categorical.groupby(by = ['SK_ID_CURR']).mean().reset_index()
grp.columns = ['BUREAU_'+column if column !='SK_ID_CURR' else column for column in grp.columns]
application_bureau = application_bureau.merge(grp, on='SK_ID_CURR', how='left')
application_bureau.update(application_bureau[grp.columns].fillna(0))



# Shape of application and bureau data combined
print('The shape application and bureau data combined:',application_bureau.shape)
```



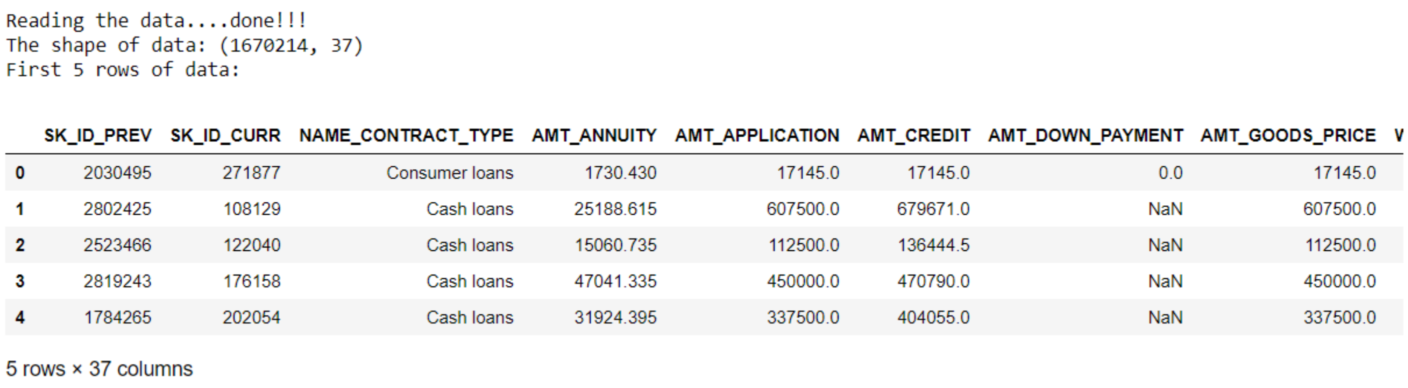
```
The shape of application and bureau data combined: (307511, 163)
```

Feature Engineering of Bureau Data:

```
# Number of past loans per customer
grp = bureau.groupby(by = ['SK_ID_CURR'])['SK_ID_BUREAU'].count().reset_index().rename(columns =
{'SK_ID_BUREAU': 'BUREAU_LOAN_COUNT'})

application_bureau = application_bureau.merge(grp, on='SK_ID_CURR', how='left')
application_bureau['BUREAU_LOAN_COUNT'] = application_bureau['BUREAU_LOAN_COUNT'].fillna(0)


# Number of types of past loans per customer
grp = bureau[['SK_ID_CURR', 'CREDIT_TYPE']].groupby(by =
['SK_ID_CURR'])['CREDIT_TYPE'].nunique().reset_index().rename(columns={'CREDIT_TYPE':
'BUREAU_LOAN_TYPES'})

application_bureau = application_bureau.merge(grp, on='SK_ID_CURR', how='left')
application_bureau['BUREAU_LOAN_TYPES'] = application_bureau['BUREAU_LOAN_TYPES'].fillna(0)

# Debt over credit ratio
bureau['AMT_CREDIT_SUM'] = bureau['AMT_CREDIT_SUM'].fillna(0)
bureau['AMT_CREDIT_SUM_DEBT'] = bureau['AMT_CREDIT_SUM_DEBT'].fillna(0)

grp1 =
bureau[['SK_ID_CURR','AMT_CREDIT_SUM']].groupby(by=['SK_ID_CURR'])['AMT_CREDIT_SUM'].sum().reset_in
dex().rename(columns={'AMT_CREDIT_SUM': 'TOTAL_CREDIT_SUM'})

grp2 =
bureau[['SK_ID_CURR','AMT_CREDIT_SUM_DEBT']].groupby(by=['SK_ID_CURR'])['AMT_CREDIT_SUM_DEBT']
.sum().reset_index().rename(columns={'AMT_CREDIT_SUM_DEBT':'TOTAL_CREDIT_SUM_DEBT'})

grp1['DEBT_CREDIT_RATIO'] = grp2['TOTAL_CREDIT_SUM_DEBT']/grp1['TOTAL_CREDIT_SUM']

del grp1['TOTAL_CREDIT_SUM']

application_bureau = application_bureau.merge(grp1, on='SK_ID_CURR', how='left')
application_bureau['DEBT_CREDIT_RATIO'] = application_bureau['DEBT_CREDIT_RATIO'].fillna(0)
application_bureau['DEBT_CREDIT_RATIO'] = application_bureau.replace([np.inf, -np.inf], 0)
application_bureau['DEBT_CREDIT_RATIO'] = pd.to_numeric(application_bureau['DEBT_CREDIT_RATIO']
```

Using Previous Application Data:

```
print('Reading the data....', end='')
previous_applicaton = pd.read_csv('previous_application.csv')
print('done!!!')
print('The shape of data:',previous_applicaton.shape)
print('First 5 rows of data:')
previous_applicaton.head()
```



```
Reading the data....done!!!
The shape of data: (1670214, 37)
First 5 rows of data:
```

	SK_ID_PREV	SK_ID_CURR	NAME_CONTRACT_TYPE	AMT_ANNUITY	AMT_APPLICATION	AMT_CREDIT	AMT_DOWN_PAYMENT	AMT_GOODS_PRICE	V
0	2030495	271877	Consumer loans	1730.430	17145.0	17145.0	0.0	17145.0	
1	2802425	108129	Cash loans	25188.615	607500.0	679671.0	NaN	607500.0	
2	2523466	122040	Cash loans	15060.735	112500.0	136444.5	NaN	112500.0	
3	2819243	176158	Cash loans	47041.335	450000.0	470790.0	NaN	450000.0	
4	1784265	202054	Cash loans	31924.395	337500.0	404055.0	NaN	337500.0	

5 rows × 37 columns

Joining Previous Application data to Application Bureau data:

```
# Number of previous applications per customer
grp =
previous_applicaton[['SK_ID_CURR','SK_ID_PREV']].groupby(by=['SK_ID_CURR'])['SK_ID_PR
EV'].count().reset_index().rename(columns={'SK_ID_PREV':'PREV_APP_COUNT'})
application_bureau_prev = application_bureau.merge(grp, on =['SK_ID_CURR'], how = 'left')
application_bureau_prev['PREV_APP_COUNT'] =
application_bureau_prev['PREV_APP_COUNT'].fillna(0)
```

Using POS_CASH_balance data:

```
print('Reading the data....', end='')
pos_cash = pd.read_csv('POS_CASH_balance.csv')
print('done!!!')
print('The shape of data:',pos_cash.shape)
print('First 5 rows of data:')
pos_cash.head()
```

```
Reading the data....done!!!
The shape of data: (10001358, 8)
First 5 rows of data:
```

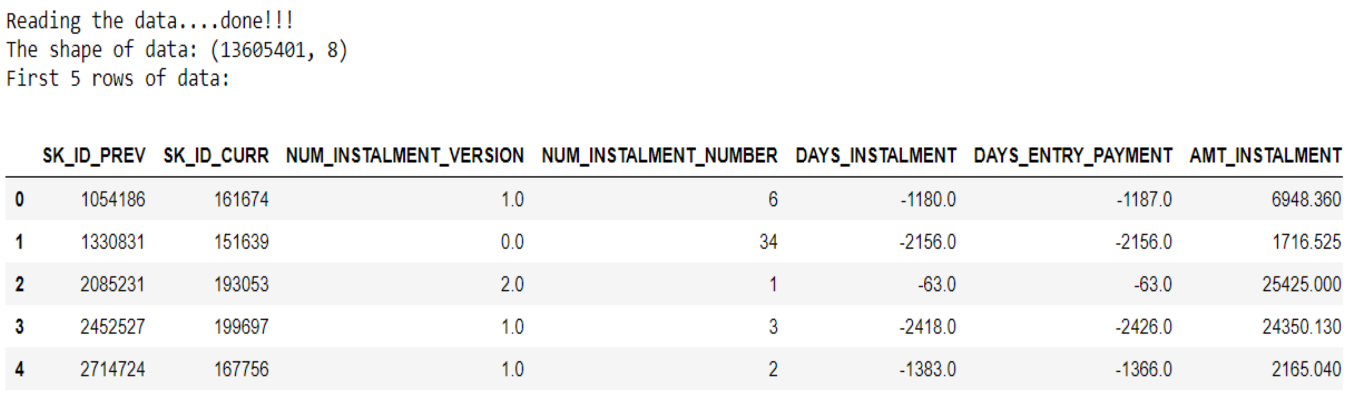
	SK_ID_PREV	SK_ID_CURR	MONTHS_BALANCE	CNT_INSTALMENT	CNT_INSTALMENT_FUTURE	NAME_CONTRACT_STATUS	SK_DPD	SK_DPD_DEF
0	1803195	182943	-31	48.0	45.0	Active	0	0
1	1715348	367990	-33	36.0	35.0	Active	0	0
2	1784872	397406	-32	12.0	9.0	Active	0	0
3	1903291	269225	-35	48.0	42.0	Active	0	0
4	2341044	334279	-35	36.0	35.0	Active	0	0

Joining POS_CASH_balance data to application_bureau_prev_data:

```
# Combining numerical features
grp = pos_cash.drop('SK_ID_PREV', axis =1).groupby(by=['SK_ID_CURR']).mean().reset_index()
prev_columns = ['POS_'+column if column != 'SK_ID_CURR' else column for column in grp.columns
]

grp.columns = prev_columns
application_bureau_prev = application_bureau_prev.merge(grp, on =['SK_ID_CURR'], how = 'left')
application_bureau_prev.update(application_bureau_prev[grp.columns].fillna(0))
```

Using installments_payments data:

```
print('Reading the data....', end='')
insta_payments = pd.read_csv('installments_payments.csv')
print('done!!!')
print('The shape of data:',insta_payments.shape)
print('First 5 rows of data:')
insta_payments.head()
```



```
Reading the data....done!!!
The shape of data: (13605401, 8)
First 5 rows of data:
```

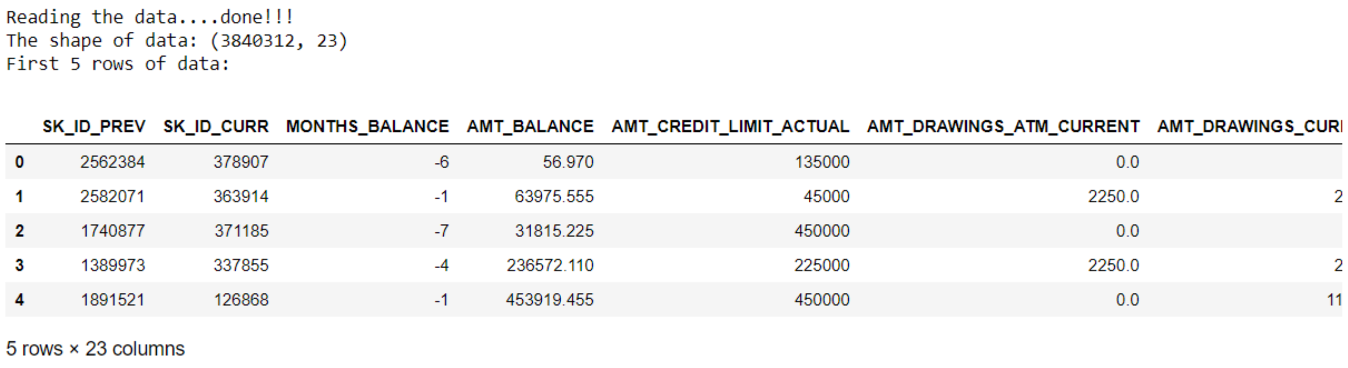
	SK_ID_PREV	SK_ID_CURR	NUM_INSTALMENT_VERSION	NUM_INSTALMENT_NUMBER	DAYS_INSTALMENT	DAYS_ENTRY_PAYMENT	AMT_INSTALMENT
0	1054186	161674	1.0	6	-1180.0	-1187.0	6948.360
1	1330831	151639	0.0	34	-2156.0	-2156.0	1716.525
2	2085231	193053	2.0	1	-63.0	-63.0	25425.000
3	2452527	199697	1.0	3	-2418.0	-2426.0	24350.130
4	2714724	167756	1.0	2	-1383.0	-1366.0	2165.040

Joining Installments Payments data to application_bureau_prev_data:

```
# Combining numerical features and there are no categorical features in this dataset
grp = insta_payments.drop('SK_ID_PREV', axis =1).groupby(by=['SK_ID_CURR']).mean().reset_index()

prev_columns = ['INSTA_'+column if column != 'SK_ID_CURR' else column for column in grp.columns ]

grp.columns = prev_columns

application_bureau_prev = application_bureau_prev.merge(grp, on =['SK_ID_CURR'], how = 'left')
application_bureau_prev.update(application_bureau_prev[grp.columns].fillna(0))
```

Using Credit card balance data:

```
print('Reading the data....', end='')
credit_card = pd.read_csv('credit_card_balance.csv')
print('done!!!')
print('The shape of data:',credit_card.shape)
print('First 5 rows of data:')
credit_card.head()
```



```
Reading the data....done!!!
The shape of data: (3840312, 23)
First 5 rows of data:
```

	SK_ID_PREV	SK_ID_CURR	MONTHS_BALANCE	AMT_BALANCE	AMT_CREDIT_LIMIT_ACTUAL	AMT_DRAWINGS_ATM_CURRENT	AMT_DRAWINGS_CURI
0	2562384	378907	-6	56.970	135000	0.0	
1	2582071	363914	-1	63975.555	45000	2250.0	2
2	1740877	371185	-7	31815.225	450000	0.0	
3	1389973	337855	-4	236572.110	225000	2250.0	2
4	1891521	126868	-1	453919.455	450000	0.0	11

5 rows × 23 columns

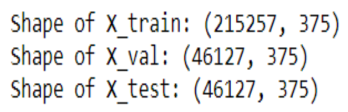
Joining Credit card balance data to application_bureau_prev data:

```
# Combining numerical features
grp = credit_card.drop('SK_ID_PREV', axis =1).groupby(by=['SK_ID_CURR']).mean().reset_index()
prev_columns = ['CREDIT_'+column if column != 'SK_ID_CURR' else column for column in
grp.columns ]
grp.columns = prev_columns
application_bureau_prev = application_bureau_prev.merge(grp, on =['SK_ID_CURR'], how = 'left')
application_bureau_prev.update(application_bureau_prev[grp.columns].fillna(0))

# Combining categorical features
credit_categorical = pd.get_dummies(credit_card.select_dtypes('object'))
credit_categorical['SK_ID_CURR'] = credit_card['SK_ID_CURR']

grp = credit_categorical.groupby('SK_ID_CURR').mean().reset_index()
grp.columns = ['CREDIT_'+column if column != 'SK_ID_CURR' else column for column in
grp.columns]

application_bureau_prev = application_bureau_prev.merge(grp, on=['SK_ID_CURR'], how='left')
application_bureau_prev.update(application_bureau_prev[grp.columns].fillna(0))
```

Shape of final prepared data: (307511, 377)

Dividing final data into train, valid and test datasets:

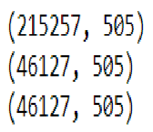
```
y = application_bureau_prev.pop('TARGET').valuesX_train, X_temp, y_train, y_temp =
train_test_split(application_bureau_prev.drop(['SK_ID_CURR'],axis=1), y, stratify = y, test_size=0.3,
random_state=42)
```



```
Shape of X_train: (215257, 375)
Shape of X_val: (46127, 375)
Shape of X_test: (46127, 375)
```

Featurizing the data:

```
#Seperation of columns into numeric and categorical columns
types = np.array([dt for dt in X_train.dtypes])

all_columns = X_train.columns.values
is_num = types != 'object'

num_cols = all_columns[is_num]
cat_cols = all_columns[~is_num]

#Featurization of numeric data
imputer_num = SimpleImputer(strategy='median')
X_train_num = imputer_num.fit_transform(X_train[num_cols])
X_val_num = imputer_num.transform(X_val[num_cols])
X_test_num = imputer_num.transform(X_test[num_cols])

scaler_num = StandardScaler()
X_train_num1 = scaler_num.fit_transform(X_train_num)
X_val_num1 = scaler_num.transform(X_val_num)
X_test_num1 = scaler_num.transform(X_test_num)

X_train_num_final = pd.DataFrame(X_train_num1, columns=num_cols)
X_val_num_final = pd.DataFrame(X_val_num1, columns=num_cols)
X_test_num_final = pd.DataFrame(X_test_num1, columns=num_cols)

# Featurization of categorical data
imputer_cat = SimpleImputer(strategy='constant', fill_value='MISSING')
X_train_cat = imputer_cat.fit_transform(X_train[cat_cols])
X_val_cat = imputer_cat.transform(X_val[cat_cols])
X_test_cat = imputer_cat.transform(X_test[cat_cols])
```



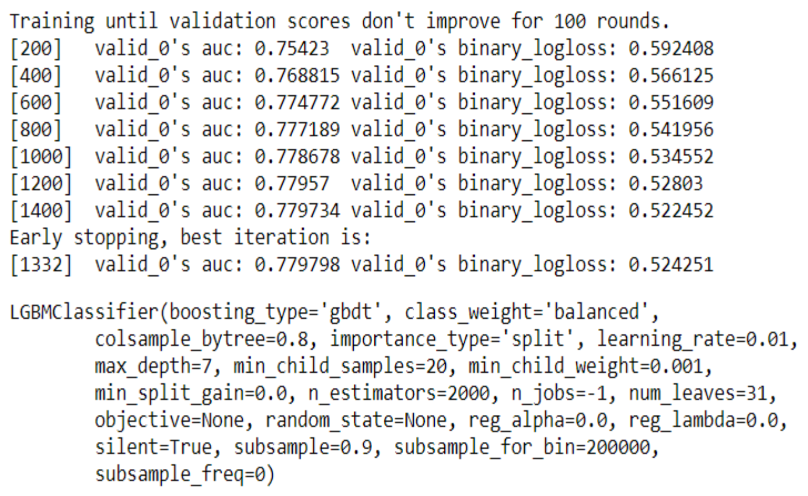
```
(215257, 505)
(46127, 505)
(46127, 505)
```

Saving the files for future use:

```
# Saving the Dataframes into CSV files for future use
X_train_final.to_csv('X_train_final.csv')
X_val_final.to_csv('X_val_final.csv')
X_test_final.to_csv('X_test_final.csv')

# Saving the numpy arrays into text files for future use
np.savetxt('y.txt', y)
np.savetxt('y_train.txt', y_train)
np.savetxt('y_val.txt', y_val)
np.savetxt('y_test.txt', y_test)
```

Selection of features:

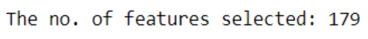
```
model_sk = lgb.LGBMClassifier(boosting_type='gbdt', max_depth=7, learning_rate=0.01,
n_estimators= 2000,
                class_weight='balanced', subsample=0.9, colsample_bytree= 0.8, n_jobs=-1)

train_features, valid_features, train_y, valid_y = train_test_split(X_train_final, y_train, test_size =
0.15, random_state = 42)

model_sk.fit(train_features, train_y, early_stopping_rounds=100, eval_set = [(valid_features,
valid_y)], eval_metric = 'auc', verbose = 200)
```



```
Training until validation scores don't improve for 100 rounds.
[200]   valid_0's auc: 0.75423  valid_0's binary_logloss: 0.592408
[400]   valid_0's auc: 0.768815 valid_0's binary_logloss: 0.566125
[600]   valid_0's auc: 0.774772 valid_0's binary_logloss: 0.551609
[800]   valid_0's auc: 0.777189 valid_0's binary_logloss: 0.541956
[1000]  valid_0's auc: 0.778678 valid_0's binary_logloss: 0.534552
[1200]  valid_0's auc: 0.77957  valid_0's binary_logloss: 0.52803
[1400]  valid_0's auc: 0.779734 valid_0's binary_logloss: 0.522452
Early stopping, best iteration is:
[1332]  valid_0's auc: 0.779798 valid_0's binary_logloss: 0.524251

LGBMClassifier(boosting_type='gbdt', class_weight='balanced',
        colsample_bytree=0.8, importance_type='split', learning_rate=0.01,
        max_depth=7, min_child_samples=20, min_child_weight=0.001,
        min_split_gain=0.0, n_estimators=2000, n_jobs=-1, num_leaves=31,
        objective=None, random_state=None, reg_alpha=0.0, reg_lambda=0.0,
        silent=True, subsample=0.9, subsample_for_bin=200000,
        subsample_freq=0)
```

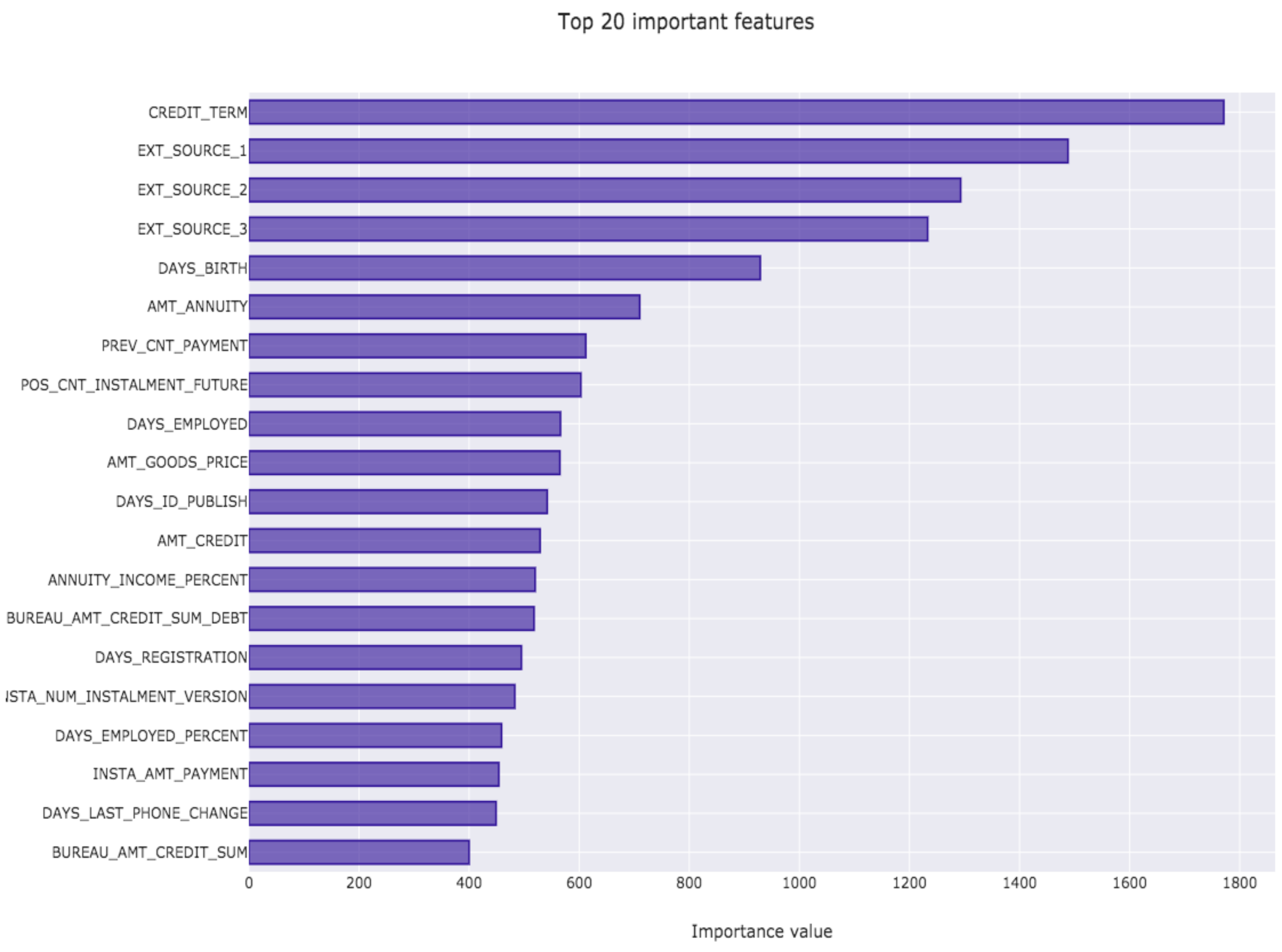
Training LGBM

```
feature_imp = pd.DataFrame(sorted(zip(model_sk.feature_importances_, X_train_final.columns)),
columns=['Value','Feature'])
features_df = feature_imp.sort_values(by="Value", ascending=False)
selected_features = list(features_df[features_df['Value']>=50]['Feature'])

# Saving the selected features into pickle file
with open('select_features.txt','wb') as fp:
    pickle.dump(selected_features, fp)

print('The no. of features selected:',len(selected_features))
```



```
The no. of features selected: 179
```

```
# Feature importance Plot
data1 = features_df.head(20)
data = [go.Bar(x =data1.sort_values(by='Value')['Value'] , y = data1.sort_values(by='Value')['Feature'],
orientation = 'h',
              marker = dict(
        color = 'rgba(43, 13, 150, 0.6)',
        line = dict(
            color = 'rgba(43, 13, 150, 1.0)',
            width = 1.5)
    )) ]

layout = go.Layout(
    autosize=False,
```



Top 20 important features
CREDIT_TERM
EXT_SOURCE_1
EXT_SOURCE_2
EXT_SOURCE_3
DAYS_BIRTH
AMT_ANNUITY
PREV_CNT_PAYMENT
POS_CNT_INSTALMENT_FUTURE
DAYS_EMPLOYED
AMT_GOODS_PRICE
DAYS_ID_PUBLISH
AMT_CREDIT
ANNUITY_INCOME_PERCENT
BUREAU_AMT_CREDIT_SUM_DEBT
DAYS_REGISTRATION
ISTA_NUM_INSTALMENT_VERSION
DAYS_EMPLOYED_PERCENT
INSTA_AMT_PAYMENT
DAYS_LAST_PHONE_CHANGE
BUREAU_AMT_CREDIT_SUM
0
200
400
600
800
1000
1200
1400
1600
1800
Importance value

Machine Learning Models:

I have tried Logistic **Regression, Random Forest and LightGBM machine** learning models.

Reusable functions for plotting Confusion matrix and CV plot.

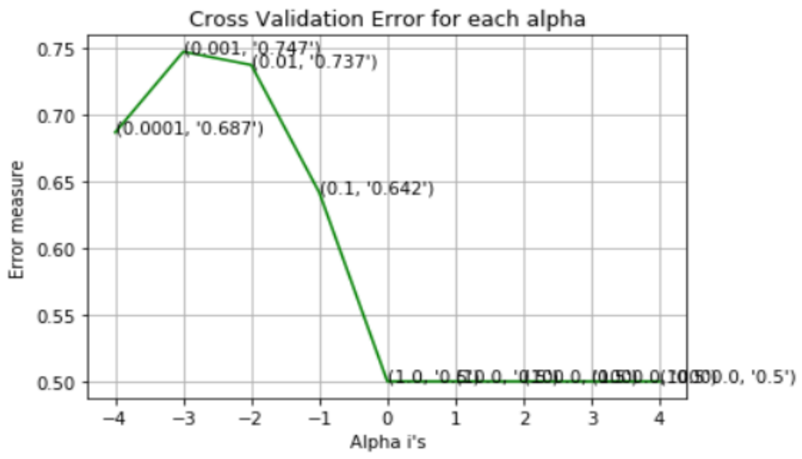
```
def plot_confusion_matrix(test_y, predicted_y):
    # Confusion matrix
    C = confusion_matrix(test_y, predicted_y)

    # Recall matrix
    A = (((C.T)/(C.sum(axis=1))).T)

    # Precision matrix
    B = (C/C.sum(axis=0))

    plt.figure(figsize=(20,4))

    labels = ['Re-paid(0)','Not Re-paid(1)']
    cmap=sns.light_palette("purple")
    plt.subplot(1,3,1)
    sns.heatmap(C, annot=True, cmap=cmap,fmt="d", xticklabels = labels, yticklabels=labels)
    plt.xlabel('Predicted Class')
    plt.ylabel('Orignal Class')
    plt.title('Confusion matrix')

    plt.subplot(1,3,2)
    sns.heatmap(A, annot=True, cmap=cmap, xticklabels = labels, yticklabels=labels)
    plt.xlabel('Predicted Class')
    plt.ylabel('Orignal Class')
    plt.title('Recall matrix')

    plt.subplot(1,3,3)
    sns.heatmap(B, annot=True, cmap=cmap, xticklabels = labels, yticklabels=labels)
    plt.xlabel('Predicted Class')
    plt.ylabel('Orignal Class')
    plt.title('Precision matrix')

    plt.show()
    def cv_plot(alpha, cv_auc):

    fig, ax = plt.subplots()
    ax.plot(np.log10(alpha), cv_auc,c='g')
    for i, txt in enumerate(np.round(cv_auc,3)):
        ax.annotate((alpha[i],str(txt)), (np.log10(alpha[i]),cv_auc[i]))
    plt.grid()
    plt.xticks(np.log10(alpha))
    plt.title("Cross Validation Error for each alpha")
    plt.xlabel("Alpha i's")
    plt.ylabel("Error measure")
```

Logistic regression with selected features:

Logistic Regression finds a hyperplane which best seperates the given positive and negative data points.

```
alpha = np.logspace(-4,4,9)
cv_auc_score = []

for i in alpha:
    clf = SGDClassifier(alpha=i, penalty='l1',class_weight = 'balanced', loss='log', random_state=28)
    clf.fit(X_train_final[selected_features], y_train)
    sig_clf = CalibratedClassifierCV(clf, method='sigmoid')
    sig_clf.fit(X_train_final[selected_features], y_train)
    y_pred_prob = sig_clf.predict_proba(X_val_final[selected_features])[:,1]
    cv_auc_score.append(roc_auc_score(y_val,y_pred_prob))
    print('For alpha {0}, cross validation AUC score
{1}'.format(i,roc_auc_score(y_val,y_pred_prob)))

cv_plot(alpha, cv_auc_score)

print('The Optimal C value is:', alpha[np.argmax(cv_auc_score)])
```

```
For alpha 0.0001, cross validation AUC score 0.6866034586096332
For alpha 0.001, cross validation AUC score 0.7470986349004096
For alpha 0.01, cross validation AUC score 0.737171244672842
For alpha 0.1, cross validation AUC score 0.641540949352706
For alpha 1.0, cross validation AUC score 0.5
For alpha 10.0, cross validation AUC score 0.5
For alpha 100.0, cross validation AUC score 0.5
For alpha 1000.0, cross validation AUC score 0.5
For alpha 10000.0, cross validation AUC score 0.5
```

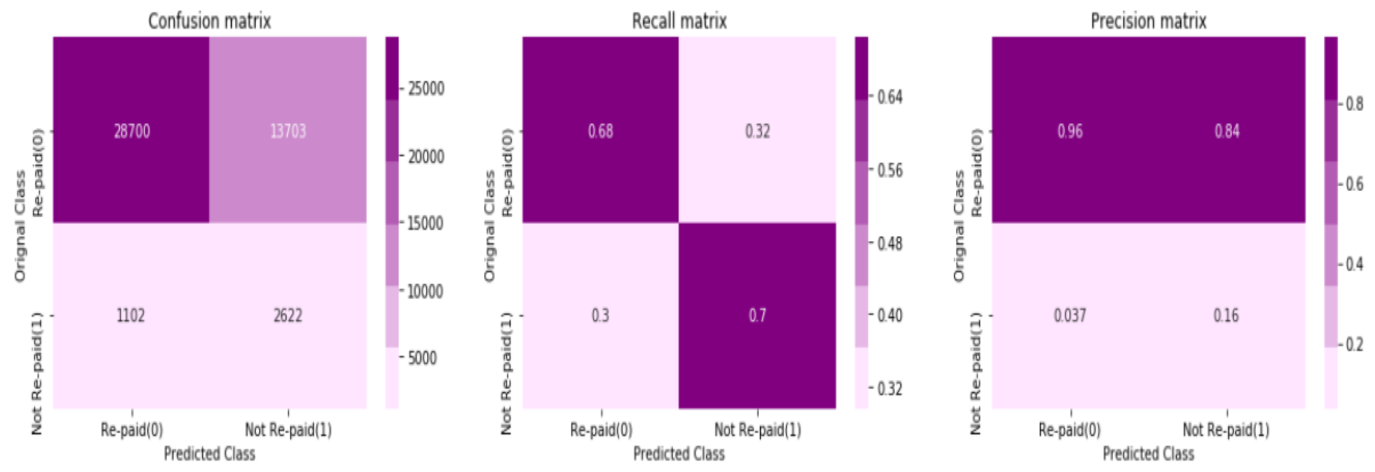


```
The Optimal C value is: 0.001
```

Cross validation results and plot for Logistic Regression model.

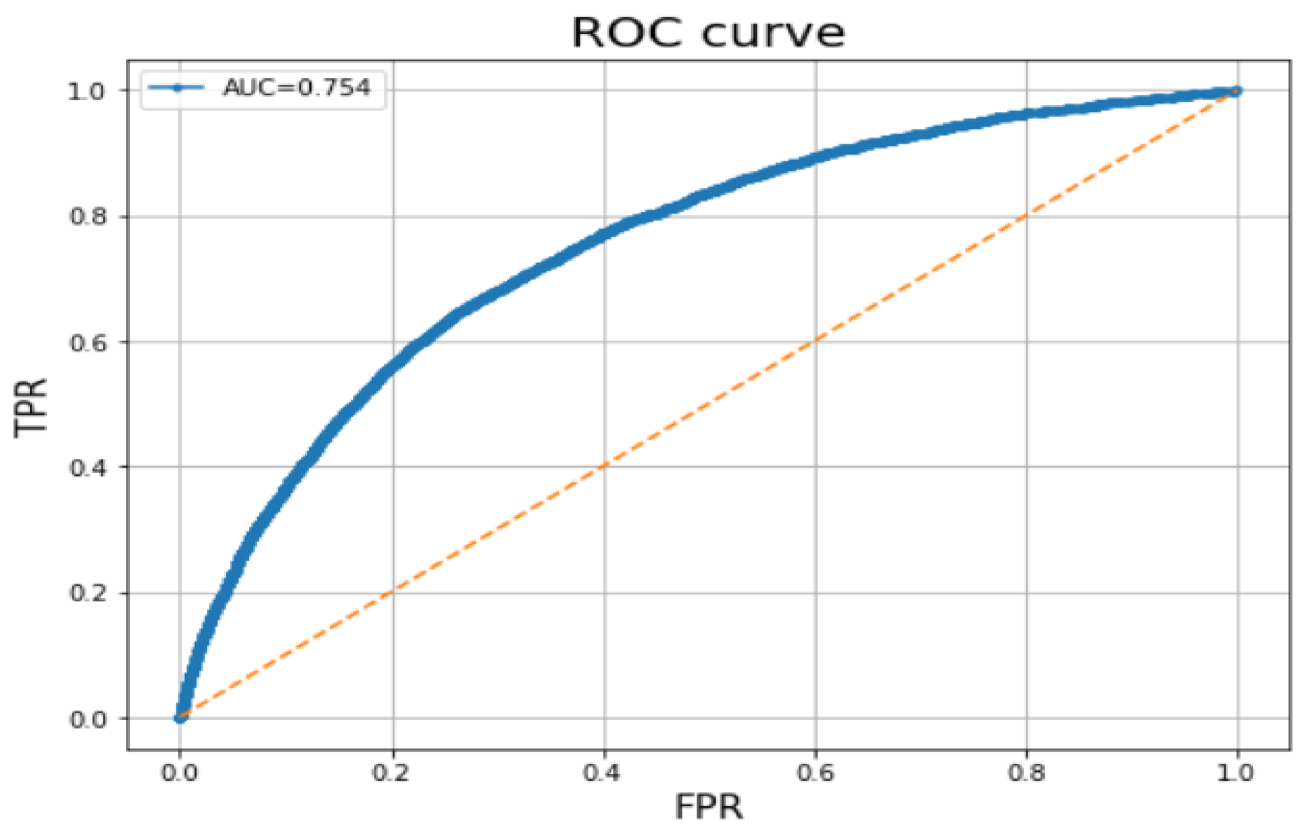
```
best_alpha = alpha[np.argmax(cv_auc_score)]
logreg = SGDClassifier(alpha = best_alpha, class_weight = 'balanced', penalty = 'l1', loss='log',
random_state = 28)
logreg.fit(X_train_final[selected_features], y_train)
logreg_sig_clf = CalibratedClassifierCV(logreg, method='sigmoid')
logreg_sig_clf.fit(X_train_final[selected_features], y_train)
y_pred_prob = logreg_sig_clf.predict_proba(X_train_final[selected_features])[:,1]
print('For best alpha {0}, The Train AUC score is {1}'.format(best_alpha,
roc_auc_score(y_train,y_pred_prob) ))
y_pred_prob = logreg_sig_clf.predict_proba(X_val_final[selected_features])[:,1]
print('For best alpha {0}, The Cross validated AUC score is {1}'.format(best_alpha,
roc_auc_score(y_val,y_pred_prob) ))
y_pred_prob = logreg_sig_clf.predict_proba(X_test_final[selected_features])[:,1]
print('For best alpha {0}, The Test AUC score is {1}'.format(best_alpha,
roc_auc_score(y_test,y_pred_prob) ))

y_pred = logreg.predict(X_test_final[selected_features])
print('The test AUC score is :', roc_auc_score(y_test,y_pred_prob))
print('The percentage of misclassified points {:05.2f}% :'.format((1-accuracy_score(y_test,
y_pred))*100))
plot_confusion_matrix(y_test, y_pred)
```

```
For best alpha 0.001, The Train AUC score is 0.7561013753905573
For best alpha 0.001, The Cross validated AUC score is 0.7470986349004096
For best alpha 0.001, The Test AUC score is 0.7536075069977747
The test AUC score is : 0.7536075069977747
The percentage of misclassified points 32.10% :
```



Logistic Regression model results

```
from sklearn.metrics import roc_curve
fpr, tpr, thresholds = roc_curve(y_test, y_pred_prob)
auc = roc_auc_score(y_test,y_pred_prob)plt.figure(figsize=(8,6))

plt.plot(fpr, tpr, marker='.')
plt.plot([0, 1], [0, 1], linestyle='--')
plt.title('ROC curve', fontsize = 20)
plt.xlabel('FPR', fontsize=15)
plt.ylabel('TPR', fontsize=15)
plt.grid()

plt.legend(["AUC=%.3f"%auc])

plt.show()
```



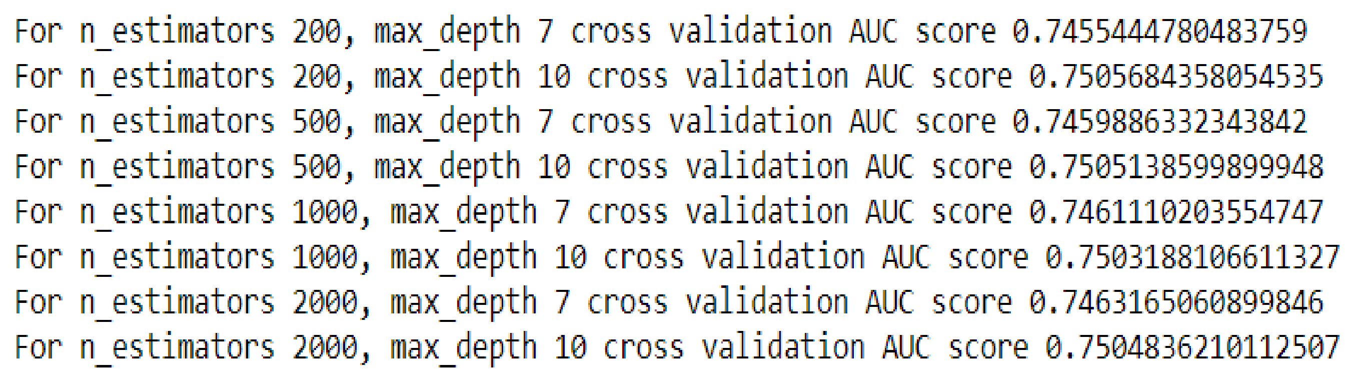
ROC curve for Logistic Regression model with AUC=0.754

Random Forest with selected features:

The Random Forest is a model made up of many decision trees. Rather than just simply averaging the prediction of trees (which we could call a “forest”), this model uses two key concepts that gives it the name random:

1. Random sampling of training data points when building trees
2. Random subsets of features considered when splitting nodes

```
alpha = [200,500,1000,2000]
max_depth = [7, 10]
cv_auc_score = []

for i in alpha:
    for j in max_depth:
        clf = RandomForestClassifier(n_estimators=i, criterion='gini',
        max_depth=j,class_weight='balanced',
        random_state=42, n_jobs=-1)
        clf.fit(X_train_final[selected_features], y_train)
        sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
        sig_clf.fit(X_train_final[selected_features], y_train)
        y_pred_prob = sig_clf.predict_proba(X_val_final[selected_features])[:,1]
        cv_auc_score.append(roc_auc_score(y_val,y_pred_prob))
        print('For n_estimators {0}, max_depth {1} cross validation AUC score {2}'.
        format(i,j,roc_auc_score(y_val,y_pred_prob)))
```

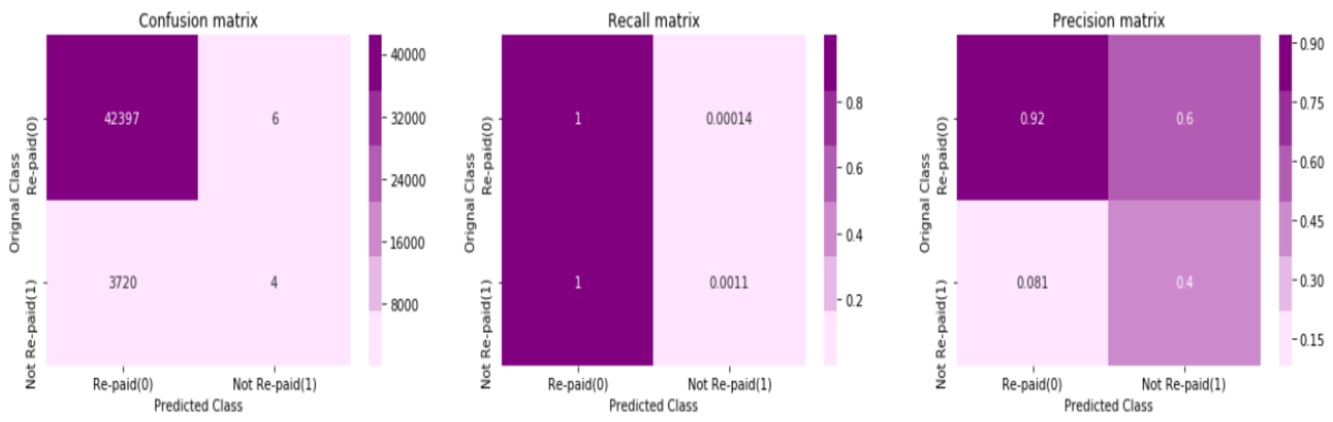


```
For n_estimators 200, max_depth 7 cross validation AUC score 0.7455444780483759
For n_estimators 200, max_depth 10 cross validation AUC score 0.7505684358054535
For n_estimators 500, max_depth 7 cross validation AUC score 0.7459886332343842
For n_estimators 500, max_depth 10 cross validation AUC score 0.7505138599899948
For n_estimators 1000, max_depth 7 cross validation AUC score 0.7461110203554747
For n_estimators 1000, max_depth 10 cross validation AUC score 0.7503188106611327
For n_estimators 2000, max_depth 7 cross validation AUC score 0.7463165060899846
For n_estimators 2000, max_depth 10 cross validation AUC score 0.7504836210112507
```

Cross validation results for Random Forest model.

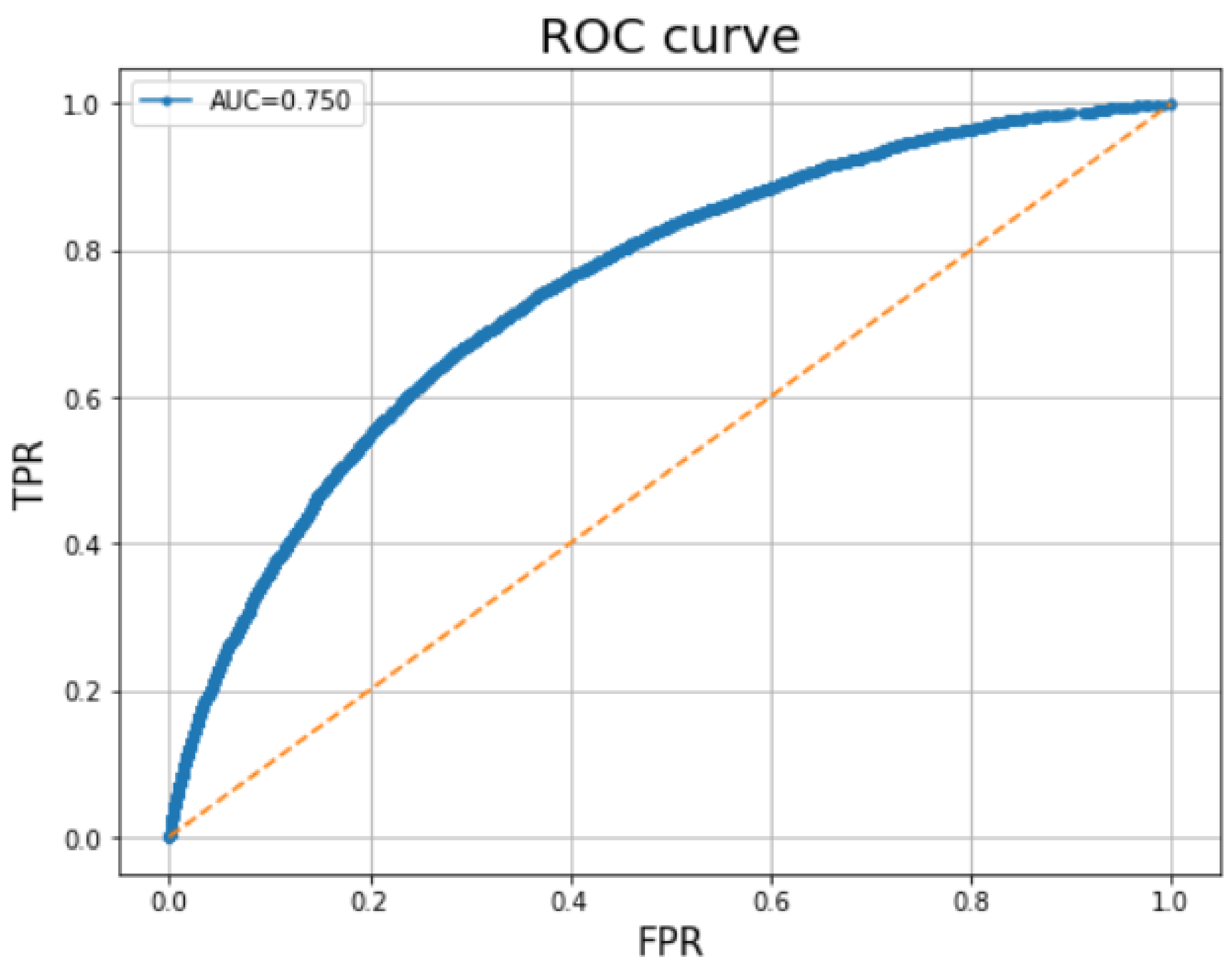
```
best_alpha = np.argmax(cv_auc_score)
print('The optimal values are: n_estimators {0}, max_depth {1} '.format(alpha[int(best_alpha/2)],
max_depth[int(best_alpha%2)]))

rf = RandomForestClassifier(n_estimators=alpha[int(best_alpha/2)], criterion='gini',
max_depth=max_depth[int(best_alpha%2)],
                            class_weight='balanced', random_state=42, n_jobs=-1)


rf.fit(X_train_final[selected_features], y_train)
rf_sig_clf = CalibratedClassifierCV(rf, method="sigmoid")
rf_sig_clf.fit(X_train_final[selected_features], y_train)


y_pred_prob = rf_sig_clf.predict_proba(X_train_final[selected_features])[:,1]
print('For best n_estimators {0} best max_depth {1}, The Train AUC score is
{2}'.format(alpha[int(best_alpha/2)],
                                                    max_depth[int(best_alpha%2)],roc_auc_score(y_train,y_pred_prob)))


y_pred_prob = rf_sig_clf.predict_proba(X_val_final[selected_features])[:,1]
print('For best n_estimators {0} best max_depth {1}, The Validation AUC score is
{2}'.format(alpha[int(best_alpha/2)],
                                                          max_depth[int(best_alpha%2)],roc_auc_score(y_val,y_pred_prob)))


y_pred_prob = rf_sig_clf.predict_proba(X_test_final[selected_features])[:,1]
print('For best n_estimators {0} best max_depth {1}, The Test AUC score is
{2}'.format(alpha[int(best_alpha/2)],
                                                       max_depth[int(best_alpha%2)],roc_auc_score(y_test,y_pred_prob)))


y_pred = rf_sig_clf.predict(X_test_final[selected_features])
print('The test AUC score is :', roc_auc_score(y_test,y_pred_prob))
print('The percentage of misclassified points {:05.2f}% :'.format((1-accuracy_score(y_test, y_pred))*100))
plot_confusion_matrix(y_test, y_pred)
```

```
The optimal values are: n_estimators 200, max_depth 10
For best n_estimators 200 best max_depth 10, The Train AUC score is 0.8417031819440642
For best n_estimators 200 best max_depth 10, The Validation AUC score is 0.7505684358054535
For best n_estimators 200 best max_depth 10, The Test AUC score is 0.7504063992087786
The test AUC score is : 0.7504063992087786
The percentage of misclassified points 08.08% :
```



Random Forest model results.

```
from sklearn.metrics import roc_curve
fpr, tpr, thresholds = roc_curve(y_test, y_pred_prob)

auc = roc_auc_score(y_test,y_pred_prob)plt.figure(figsize=(8,6))
plt.plot(fpr, tpr, marker='.')
plt.plot([0, 1], [0, 1], linestyle='--')
plt.title('ROC curve', fontsize = 20)
plt.xlabel('FPR', fontsize=15)
plt.ylabel('TPR', fontsize=15)
plt.grid()
plt.legend(["AUC=%.3f"%auc])
plt.show()
```



ROC curve for Random Forest model with AUC=0.75

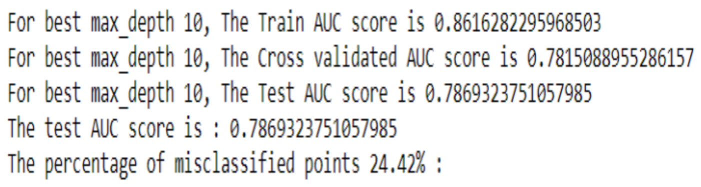
LightGBM with selected features:

Light GBM is a fast, distributed, high-performance gradient boosting framework based on decision tree algorithm, used for ranking, classification and many other machine learning tasks.

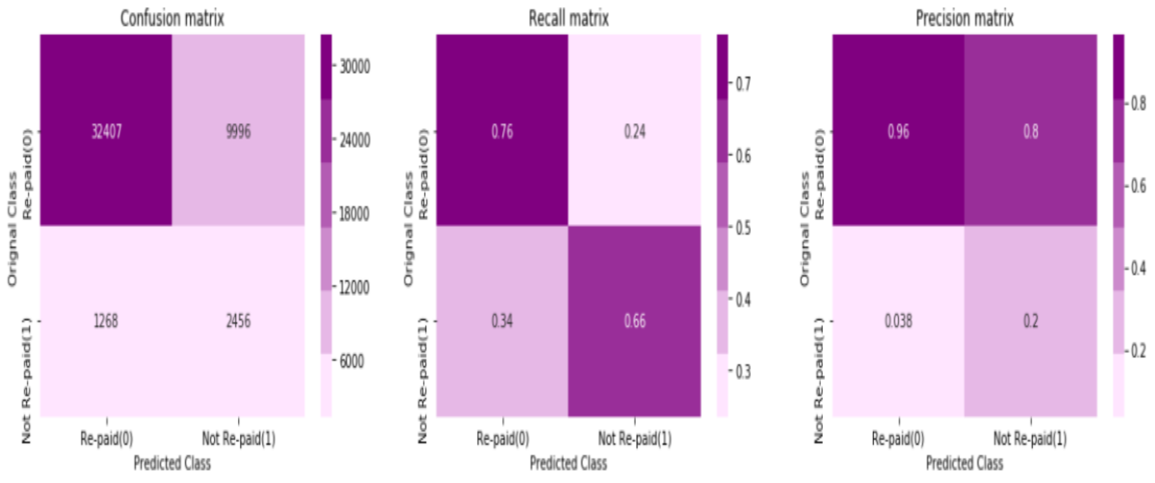
Since it is based on decision tree algorithms, it splits the tree leaf wise with the best fit whereas other boosting algorithms split the tree depth wise or level wise rather than leaf-wise. So when growing on the same leaf in Light GBM, the leaf-wise algorithm can reduce more loss than the level-wise algorithm and hence results in much better accuracy which can rarely be achieved by any of the existing boosting algorithms. Also, it is surprisingly very fast, hence the word ‘Light’.

```
weight = np.ones((len(X_train_final),), dtype=int)
for i in range(len(X_train_final)):
    if y_train[i]== 0:
        weight[i]=1
    else:
        weight[i]=11
train_data=lgb.Dataset(X_train_final[selected_features], label = y_train, weight= weight )
valid_data=lgb.Dataset(X_val_final[selected_features], label = y_val)cv_auc_score = []
max_depth = [3, 5, 7, 10]for i in max_depth:

    params = {'boosting_type': 'gbdt',
          'max_depth' : i,
          'objective': 'binary',
          'nthread': 5,
          'num_leaves': 32,
          'learning_rate': 0.05,
          'max_bin': 512,
          'subsample_for_bin': 200,
          'subsample': 0.7,
          'subsample_freq': 1,
          'colsample_bytree': 0.8,
          'reg_alpha': 20,
          'reg_lambda': 20,
          'min_split_gain': 0.5,
          'min_child_weight': 1,
          'min_child_samples': 10,
          'scale_pos_weight': 1,
          'num_class' : 1,
          'metric' : 'auc'
          }

lgbm = lgb.train(params,
                train_data,
                2500,
                valid_sets=valid_data,
                early_stopping_rounds= 100,
                verbose_eval= 10
                )
    y_pred_prob = lgbm.predict(X_val_final[selected_features])
    cv_auc_score.append(roc_auc_score(y_val,y_pred_prob))
    print('For  max_depth {0} and some other parameters, cross validation AUC score
{1}'.format(i,roc_auc_score(y_val,y_pred_prob)))

print('The optimal  max_depth: ', max_depth[np.argmax(cv_auc_score)])
```

```
params = {'boosting_type': 'gbdt',
          'max_depth' : max_depth[np.argmax(cv_auc_score)],
          'objective': 'binary',
          'nthread': 5,
          'num_leaves': 32,
          'learning_rate': 0.05,
          'max_bin': 512,
          'subsample_for_bin': 200,
          'subsample': 0.7,
          'subsample_freq': 1,
          'colsample_bytree': 0.8,
          'reg_alpha': 20,
          'reg_lambda': 20,
          'min_split_gain': 0.5,
          'min_child_weight': 1,
          'min_child_samples': 10,
          'scale_pos_weight': 1,
          'num_class' : 1,
          'metric' : 'auc'
          }

lgbm = lgb.train(params,
                 train_data,
                 2500,
                 valid_sets=valid_data,
                 early_stopping_rounds= 100,
                 verbose_eval= 10
                 )

y_pred_prob = lgbm.predict(X_train_final[selected_features])
print('For best max_depth {0}, The Train AUC score is
{1}'.format(max_depth[np.argmax(cv_auc_score)], roc_auc_score(y_train,y_pred_prob) ))

y_pred_prob = lgbm.predict(X_val_final[selected_features])
print('For best max_depth {0}, The Cross validated AUC score is
{1}'.format(max_depth[np.argmax(cv_auc_score)], roc_auc_score(y_val,y_pred_prob) ))

y_pred_prob = lgbm.predict(X_test_final[selected_features])
print('For best max_depth {0}, The Test AUC score is {1}'.format(max_depth[np.argmax(cv_auc_score)],

 roc_auc_score(y_test,y_pred_prob) ))

y_pred = np.ones((len(X_test_final),), dtype=int)

for i in range(len(y_pred_prob)):
    if y_pred_prob[i]<=0.5:
        y_pred[i]=0
    else:
        y_pred[i]=1

print('The test AUC score is :', roc_auc_score(y_test,y_pred_prob))

print('The percentage of misclassified points {:05.2f}% :'.format((1-accuracy_score(y_test,
y_pred))*100))

plot_confusion_matrix(y_test, y_pred)
```

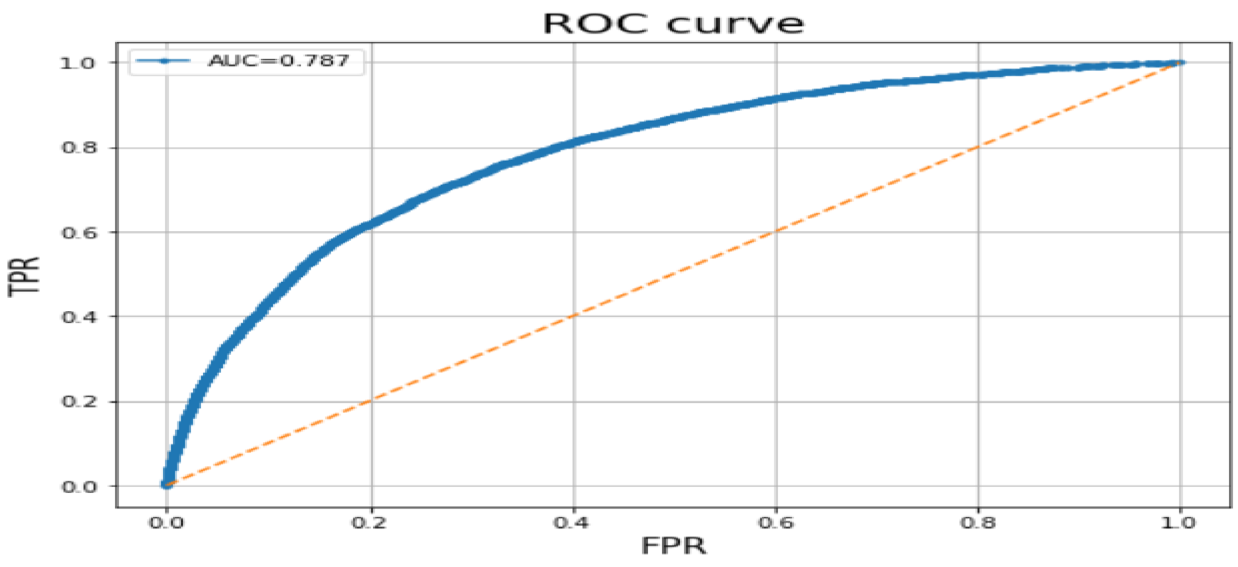


```
For best max_depth 10, The Train AUC score is 0.8616282295968503
For best max_depth 10, The Cross validated AUC score is 0.7815088955286157
For best max_depth 10, The Test AUC score is 0.7869323751057985
The test AUC score is : 0.7869323751057985
The percentage of misclassified points 24.42% :
```



LightGBM model Results

```
from sklearn.metrics import roc_curve
fpr, tpr, thresholds = roc_curve(y_test, y_pred_prob)
auc = roc_auc_score(y_test,y_pred_prob)plt.figure(figsize=(8,6))

plt.plot(fpr, tpr, marker='.')
plt.plot([0, 1], [0, 1], linestyle='--')
plt.title('ROC curve', fontsize = 20)
plt.xlabel('FPR', fontsize=15)
plt.ylabel('TPR', fontsize=15)
plt.grid()
plt.legend(["AUC=%.3f"%auc])
plt.show()
```



ROC curve for LightGBM model with AUC=0.787

Overview of Results:

Model	Train AUC	Validation AUC	Test AUC
Logistic Regression with Selected features	0.756	0.747	0.753
Random Forest with Selected features	0.841	0.751	0.751
LightGBM with Selected features	**0.861**	**0.781**	**0.787**

LightGBM gives the best performance and it is also faster to train when compared to Xgboost.