Home Credit Default Risk

1. Business Problem

1.1 Description

Home Credit offers easy, simple and fast loans for a range of Home Appliances, Mobile Phones, Laptops, Two Wheelers , and varied personal needs. Home Credit strives to broaden financial inclusion for the unbanked population by providing a positive and safe borrowing experience. In order to make sure this underserved population has a positive loan experience, Home Credit makes use of a variety of alternative data--including telco and transactional information--to predict their clients' repayment abilities.

This project focuses on the problem of predicting the capability of each applicant of repaying a loan, given the applicant data, all credits data from Credit Bureau, previous applications data from Home Credit and some more data.

Problem Statemtent
To predict how capable each applicant is of repaying a loan, so that sanctioning loan only for the applicants who are likely to repay the loan.

Source:
https://www.kaggle.com/c/home-credit-default-risk

1.2 Sources/Useful Links

Data Source : https://www.kaggle.com/c/home-credit-default-risk/data (https://www.kaggle.com/c/home-credit-default-risk/data)

1.3 Real World / Business Objectives and Constraints

1. No strict latency constraints.
2. Predict the probability of capability of each applicant of repaying a loan, so that you can choose any threshold of choice.
3. The cost of a mis-classification can be very high(Loss for the organization).
4. Interpretability is partially important.

2. Machine Learning Problem

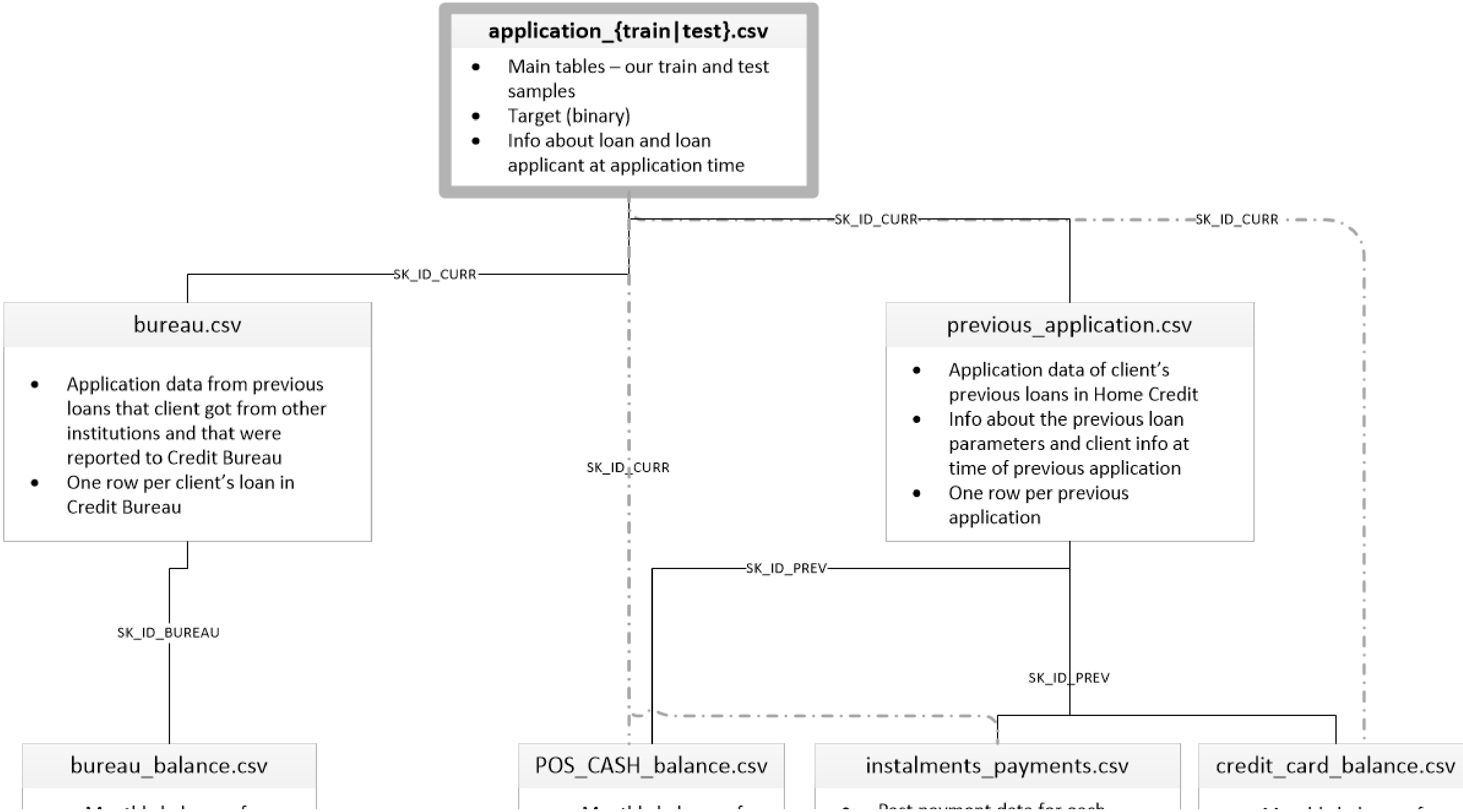
2.1 Data

The data is provided by Home Credit, a service dedicated to provided lines of credit (loans) to the unbanked population.

There are 7 different sources of data:

- **application_train/application_test**: The main training data with information about each loan application at Home Credit. Every loan has its own row and is identified by the feature SK_ID_CURR. The training application data comes with the TARGET indicating 0: the loan was repaid or 1: the loan was not repaid. Here we will use only the Training data.
- **bureau**: In this dataset it consists of data concerning client's previous credits from other financial institutions. Each previous credit has its own row in bureau, but one loan in the application data can have multiple previous credits.
- **bureau_balance**: It consists of monthly data about the previous credits in bureau. Each row is one month of a previous credit, and a single previous credit can have multiple rows, one for each month of the credit length.
- **previous_application**:The data of previous applications for loans at Home Credit of clients who have loans in the application data. Each current loan in the application data can have multiple previous loans. Each previous application has one row and is identified by the feature SK_ID_PREV.
- **POS_CASH_BALANCE**: It consists of monthly data about previous point of sale or cash loans clients have had with Home Credit. Each row is one month of a previous point of sale or cash loan, and a single previous loan can have many rows.
- **credit_card_balance**:The monthly data about previous credit cards clients have had with Home Credit. Each row is one month of a credit card balance, and a single credit card can have many rows.
- **installments_payment**:The data of payment history for previous loans at Home Credit. There is one row for every made payment and one row for every missed payment.

  The below diagram shows how the data is related:



2.1.1 Data Overview

- Data will be in a file Train.csv
- Train.csv contains 5 columns : qid1, qid2, question1, question2, is_duplicate
- Size of Train.csv - 60MB
- Number of rows in Train.csv = 404,290

```
"id","qid1","qid2","question1","question2","is_duplicate"
"0","1","2","What is the step by step guide to invest in share market in india?","What is the step by step guide t
o invest in share market?","0"
"1","3","4","What is the story of Kohinoor (Koh-i-Noor) Diamond?","What would happen if the Indian government stol
e the Kohinoor (Koh-i-Noor) diamond back?","0"
"7","15","16","How can I be a good geologist?","What should I do to be a great geologist?","1"
"11","23","24","How do I read and find my YouTube comments?","How can I see all my Youtube comments?","1"
```

2.2 Mapping the real world problem to an ML problem

2.2.1 Type of Machine Learning Problem

It is a binary classification problem, for the given applicant data we need to predict if they are capable to repay the loan or not.

2.2.2 Performance Metric

Source: https://www.kaggle.com/c/home-credit-default-risk#evaluation (https://www.kaggle.com/c/home-credit-default-risk#evaluation)

Metric(s):

- AUC : https://en.wikipedia.org/wiki/Receiver_operating_characteristic (https://en.wikipedia.org/wiki/Receiver_operating_characteristic)
- Binary Confusion Matrix

2.3 Train and Test Construction

We build train, validation and test datasets by stratified splitting with target in the ratio of 70:15:15 .

3. Exploratory Data Analysis

In [1]:

```
import pandas as pd
import sklearn
import numpy as np
import matplotlib.pyplot as plt
import os
import warnings
import seaborn as sns

from 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 SGDClassifier

import plotly.offline as py
import plotly.graph_objs as go

from 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 lgb

warnings.filterwarnings('ignore')
%matplotlib inline
```

In [2]:
```
os.chdir('/Users/ANANDAPAVANI/Desktop/Praveen/Applied AI/loan/input')
```

3.1 Reading data and basic stats

- In this case study, we have multiple datasets from different data sources to deal with. First, we will start with the application dataset(Main table) and proceed further with the other datasets.

In [4]:
```
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:
```

Out[4]:

	SK_ID_CURR	TARGET	NAME_CONTRACT_TYPE	CODE_GENDER	FLAG_OWN_CAR	FLAG_OWN_REALTY	CNT_CHILDREN	AMT_INC
0	100002	1	Cash loans	M	N	Y	0	
1	100003	0	Cash loans	F	N	N	0	
2	100004	0	Revolving loans	M	Y	Y	0	
3	100006	0	Cash loans	F	N	Y	0	
4	100007	0	Cash loans	M	N	Y	0	

5 rows × 122 columns

In [5]:
```
application.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 307511 entries, 0 to 307510
Columns: 122 entries, SK_ID_CURR to AMT_REQ_CREDIT_BUREAU_YEAR
dtypes: float64(65), int64(41), object(16)
memory usage: 286.2+ MB
```

We are using 'application_train.csv' file :

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

3.2 Basic Analysis

3.2.1 Checking for Missing values

In [6]:
```python
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:
```

Out[6]:

	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
FLOORSMIN_MEDI	208642	67.848630
FLOORSMIN_MODE	208642	67.848630
FLOORSMIN_AVG	208642	67.848630
YEARS_BUILD_MEDI	204488	66.497784
YEARS_BUILD_AVG	204488	66.497784
YEARS_BUILD_MODE	204488	66.497784
OWN_CAR_AGE	202929	65.990810
LANDAREA_MODE	182590	59.376738
LANDAREA_AVG	182590	59.376738
LANDAREA_MEDI	182590	59.376738

Observations:

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

3.2.2 Checking for Duplicates

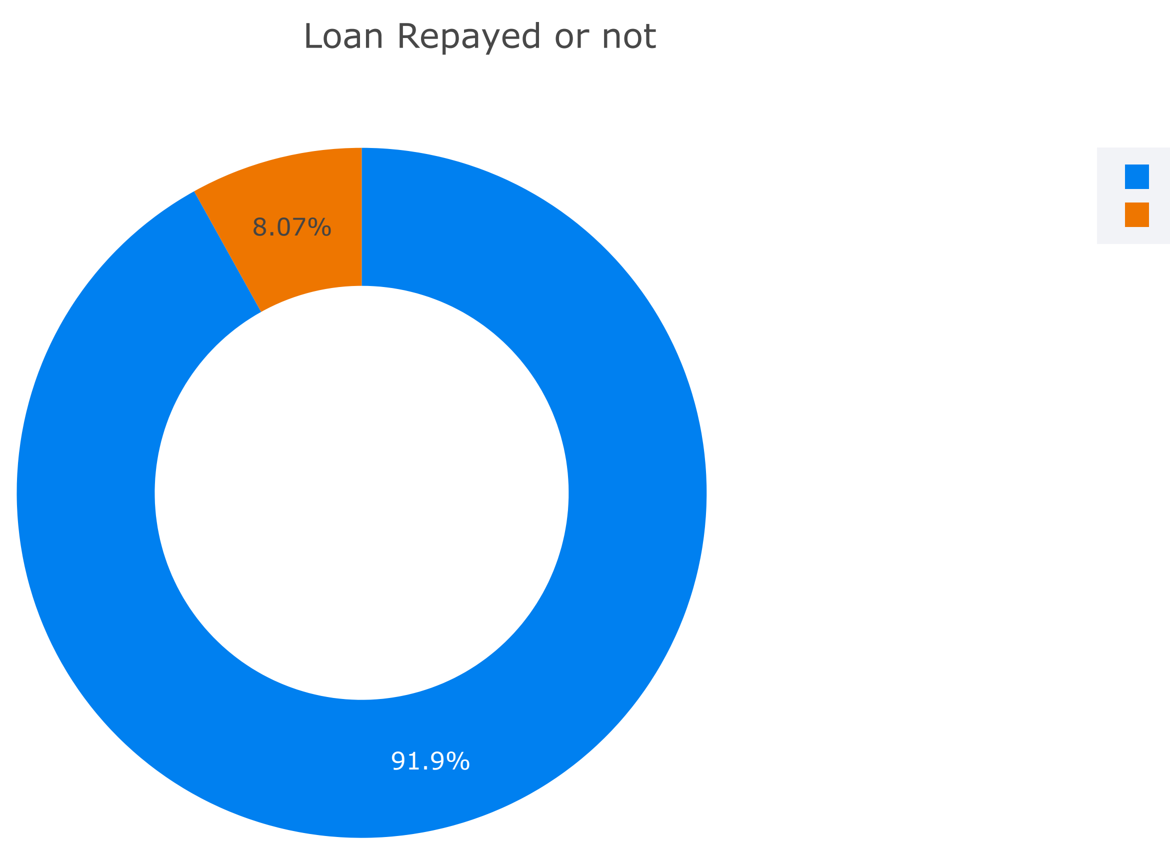
In [7]:
```
columns_without_id = [col for col in application.columns if col!='SK_ID_CURR']

#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, kee
p=False)]
      .shape[0])
```

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

3.2.3 Distribution of data points among output classes

Most of the analysis are plotted using **Plotly** , you can hover over the plot to see the overview of data.

In [8]:
```
cf.set_config_file(theme='polar')
target_val = application['TARGET'].value_counts()
target_df = pd.DataFrame({'labels': ['Loan Repayed (0)','Loan not Repayed(1)'],
                   'values': target_val.values
                  })
target_df.iplot(kind='pie',labels='labels',values='values', title='Loan Repayed or not', hole = 0.6)
```



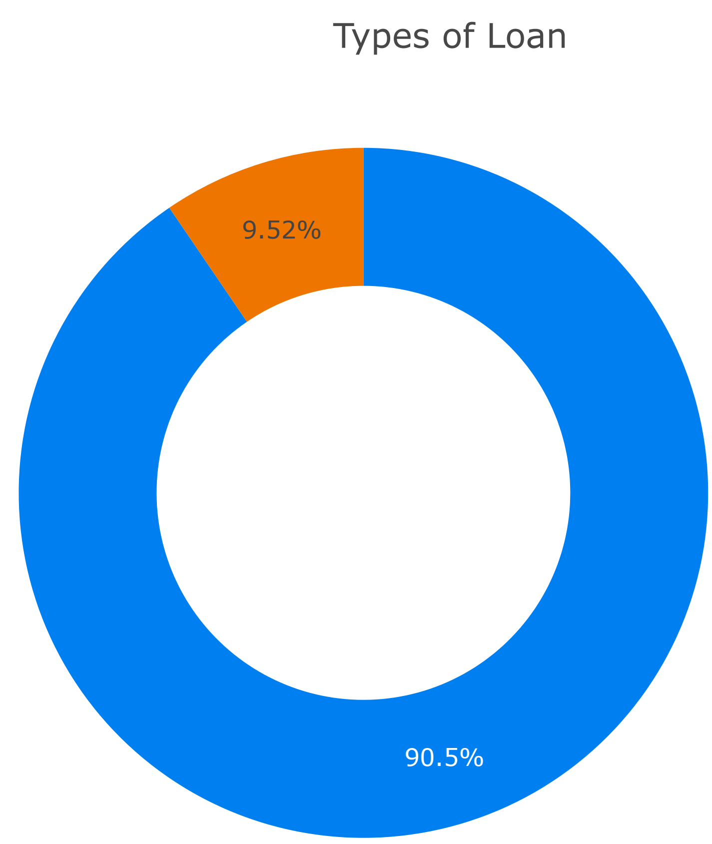
Observations:

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

3.3 Data Analysis

3.3.1 Types of loan

In [9]:
```
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)
```

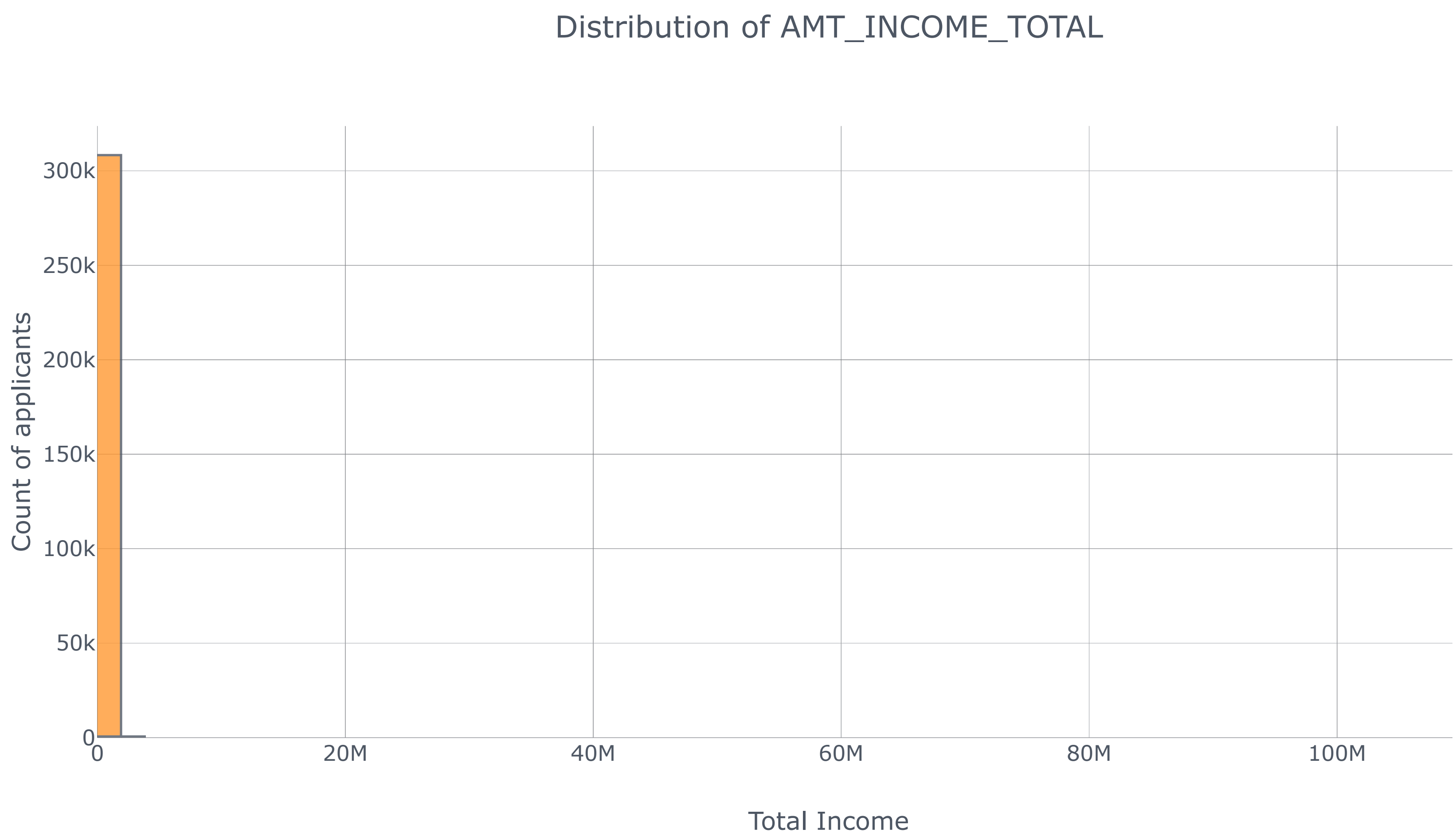


Observations:

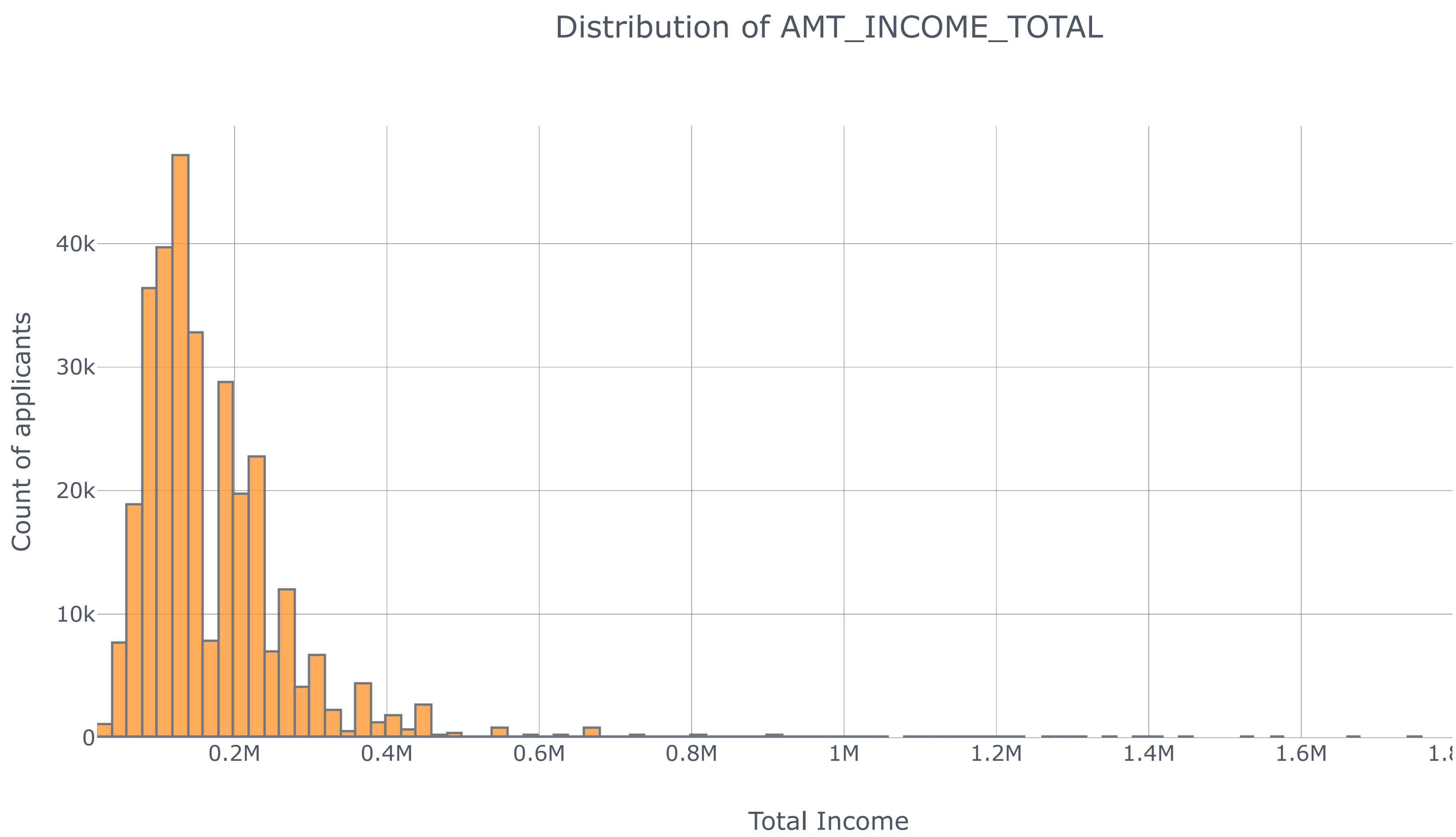
- Many people are willing to take cash loan than revolving loan (https://www.investopedia.com/terms/r/revolving-loan-facility.asp (https://www.investopedia.com/terms/r/revolving-loan-facility.asp)).

3.3.2 Distribution of AMT_INCOME_TOTAL

In [10]:
```
cf.set_config_file(theme='pearl')
application['AMT_INCOME_TOTAL'].iplot(kind='histogram', bins=100,
                                      xTitle = 'Total Income',yTitle ='Count of applicants',
                                     title='Distribution of AMT_INCOME_TOTAL')
```



```
In [11]: 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')
```



In [12]:
```
(application[application['AMT_INCOME_TOTAL'] > 1000000]['TARGET'].value_counts())/len(application[application
['AMT_INCOME_TOTAL'] > 1000000])*100
```

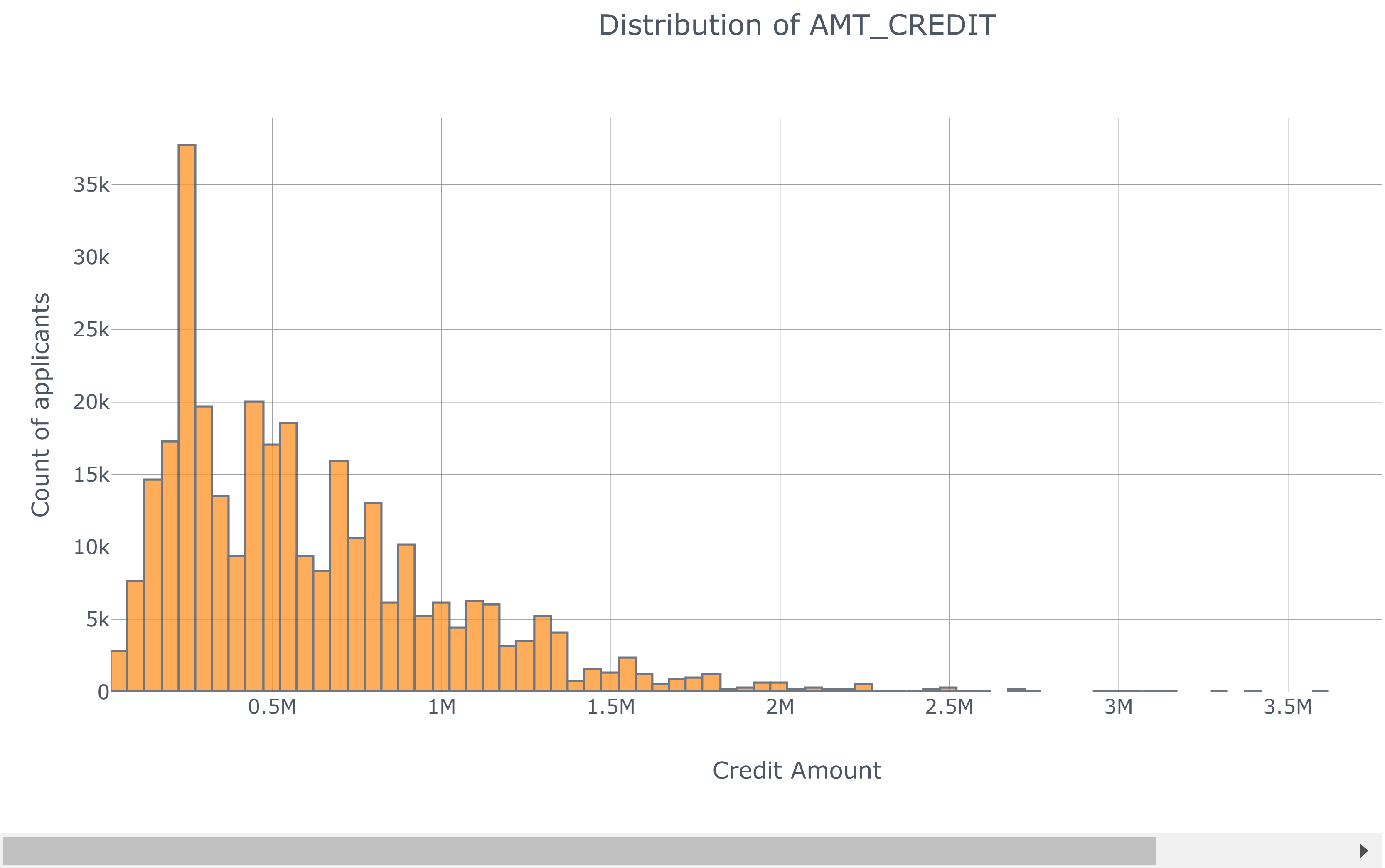
Out[12]:
```
0    94.8
1     5.2
Name: TARGET, dtype: float64
```

Observations:

- The distribution is right skewed and there are extreme values, we can apply log distribution.
- People with high income are likely to repay the loan.

3.3.3 Distribution of AMT_CREDIT

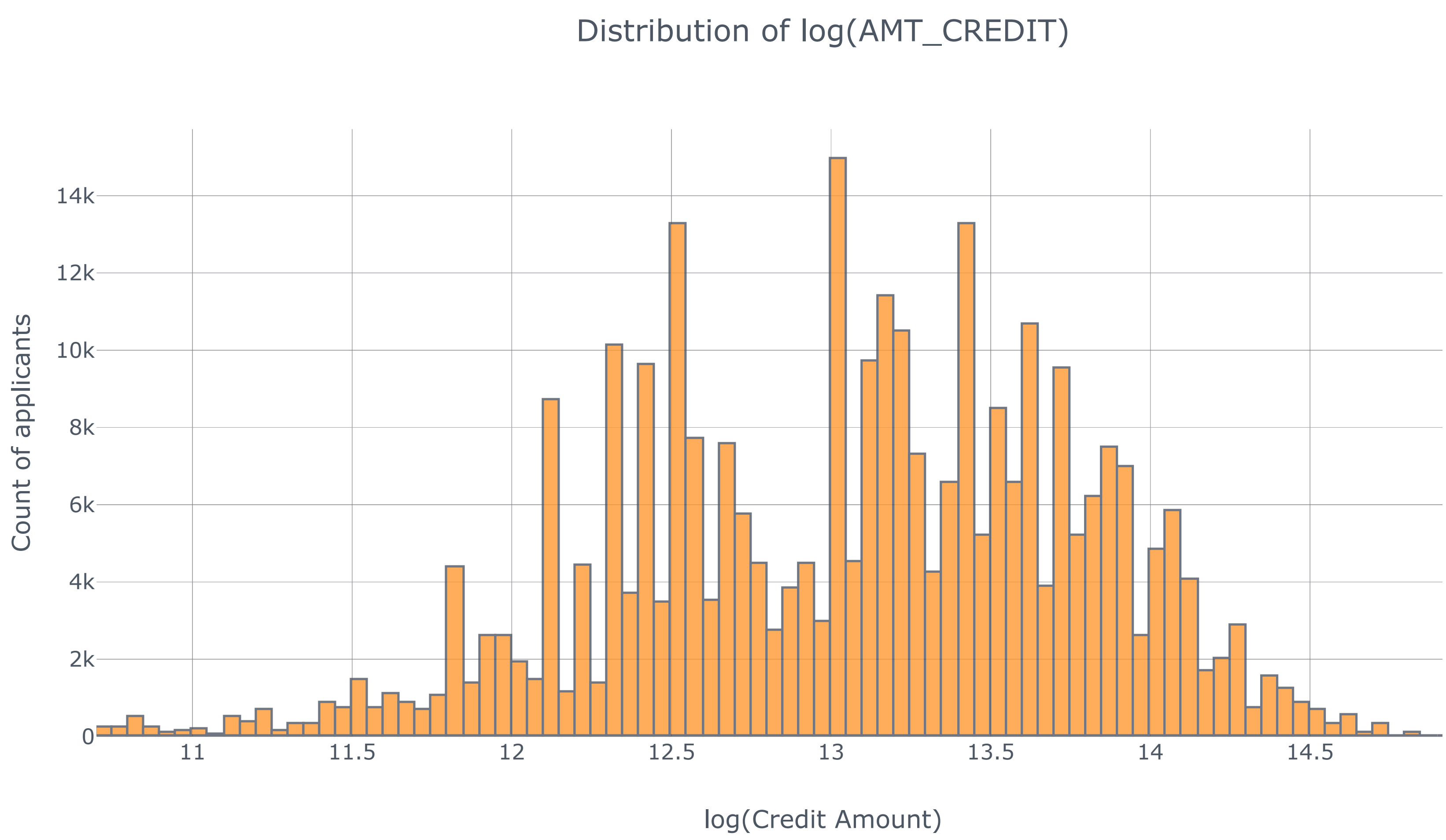
In [13]:
```
application['AMT_CREDIT'].iplot(kind='histogram', bins=100,
                               xTitle = 'Credit Amount',yTitle ='Count of applicants',
                               title='Distribution of AMT_CREDIT')
```



In [14]:
```
(application[application['AMT_CREDIT'] > 2000000]['TARGET'].value_counts())/len(application[application['AMT_
CREDIT'] > 2000000])*100
```

Out[14]:
```
0    96.747166
1     3.252834
Name: TARGET, dtype: float64
```

In [15]:
```
np.log(application['AMT_CREDIT']).iplot(kind='histogram', bins=100,
                               xTitle = 'log(Credit Amount)',yTitle ='Count of applicants',
                               title='Distribution of log(AMT_CREDIT)')
```

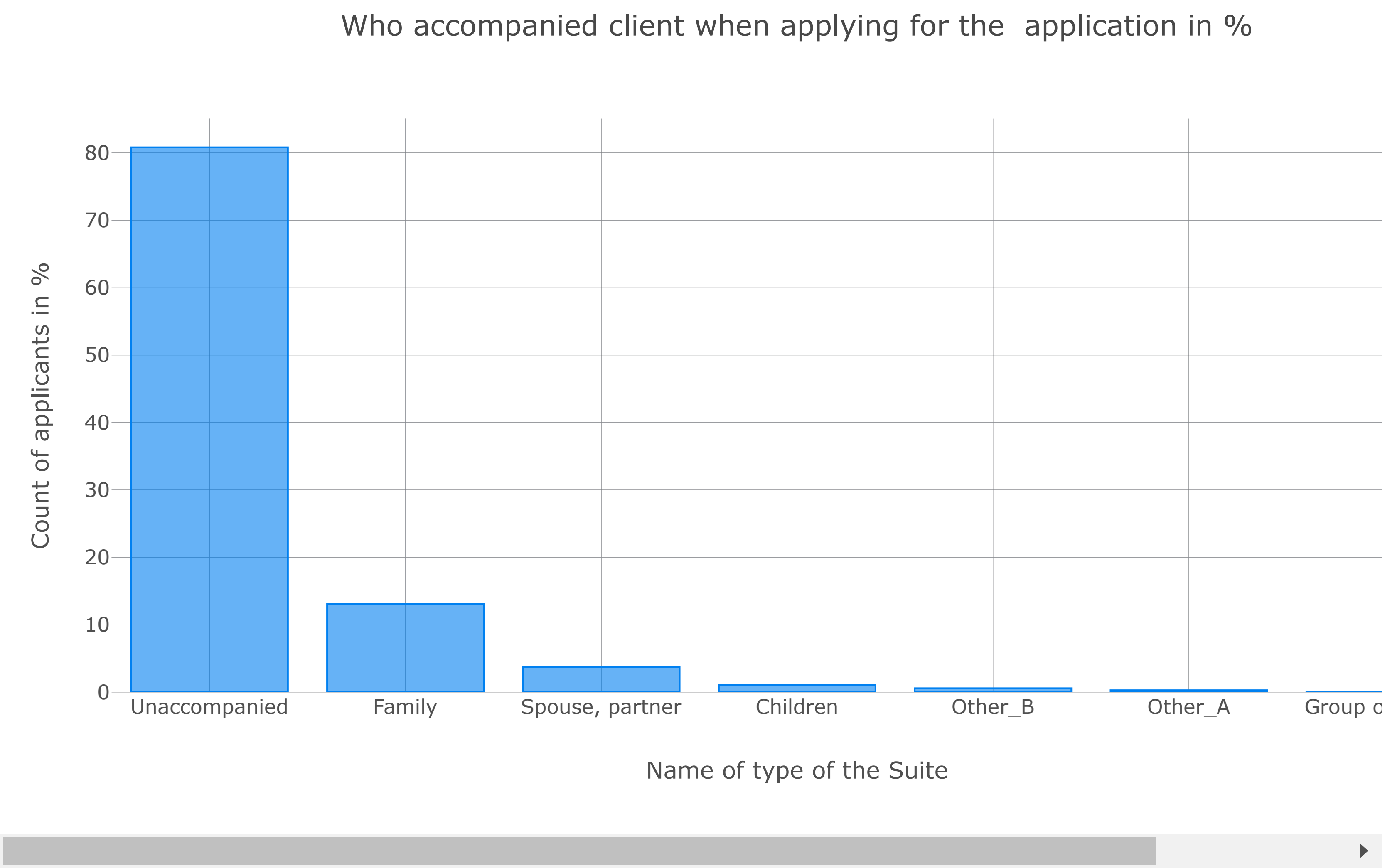


Observations:

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

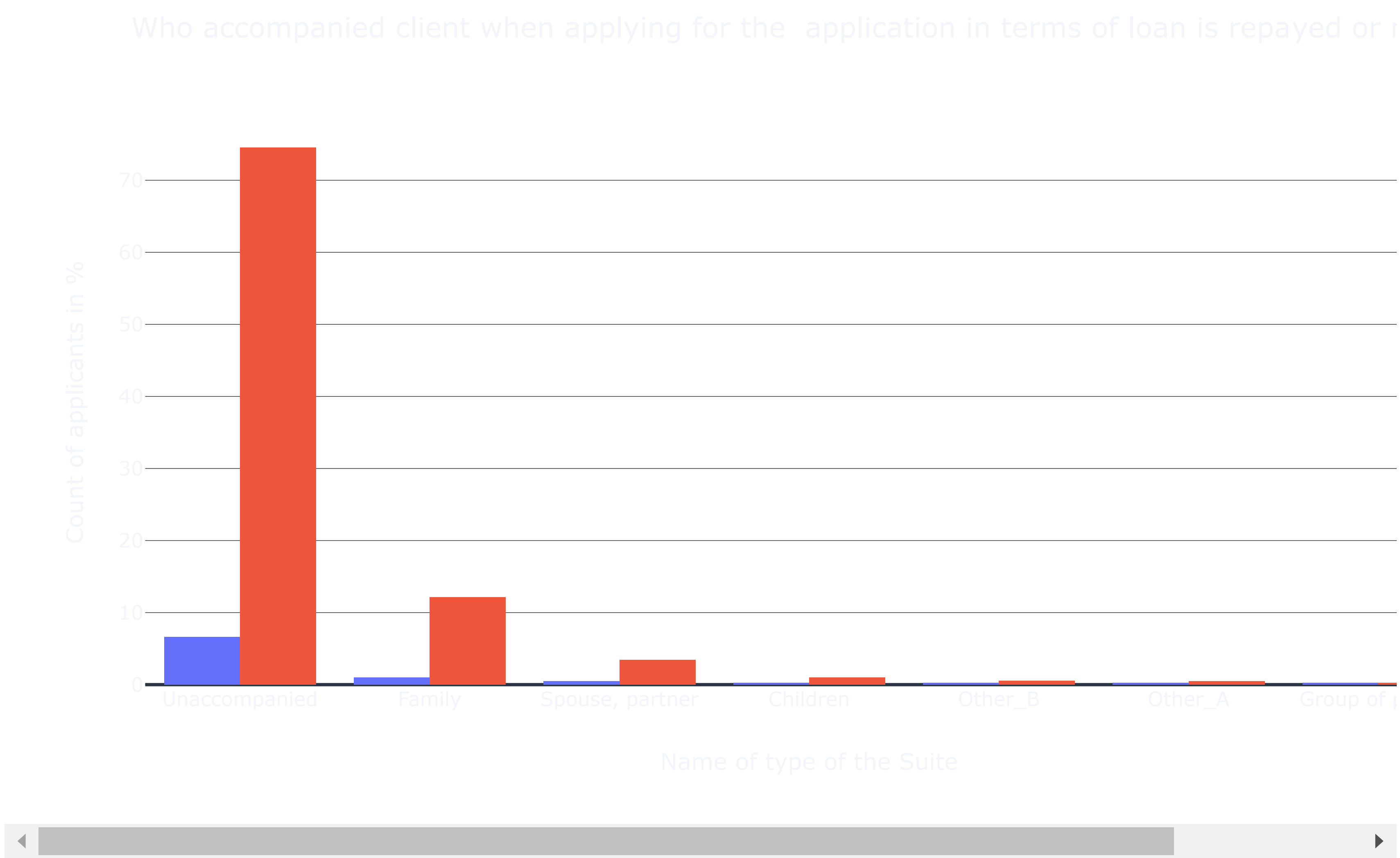
3.3.4 Distribution of Name of type of the Suite in terms of loan is repayed or not

In [16]:
```
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 % ')
```



In [17]:

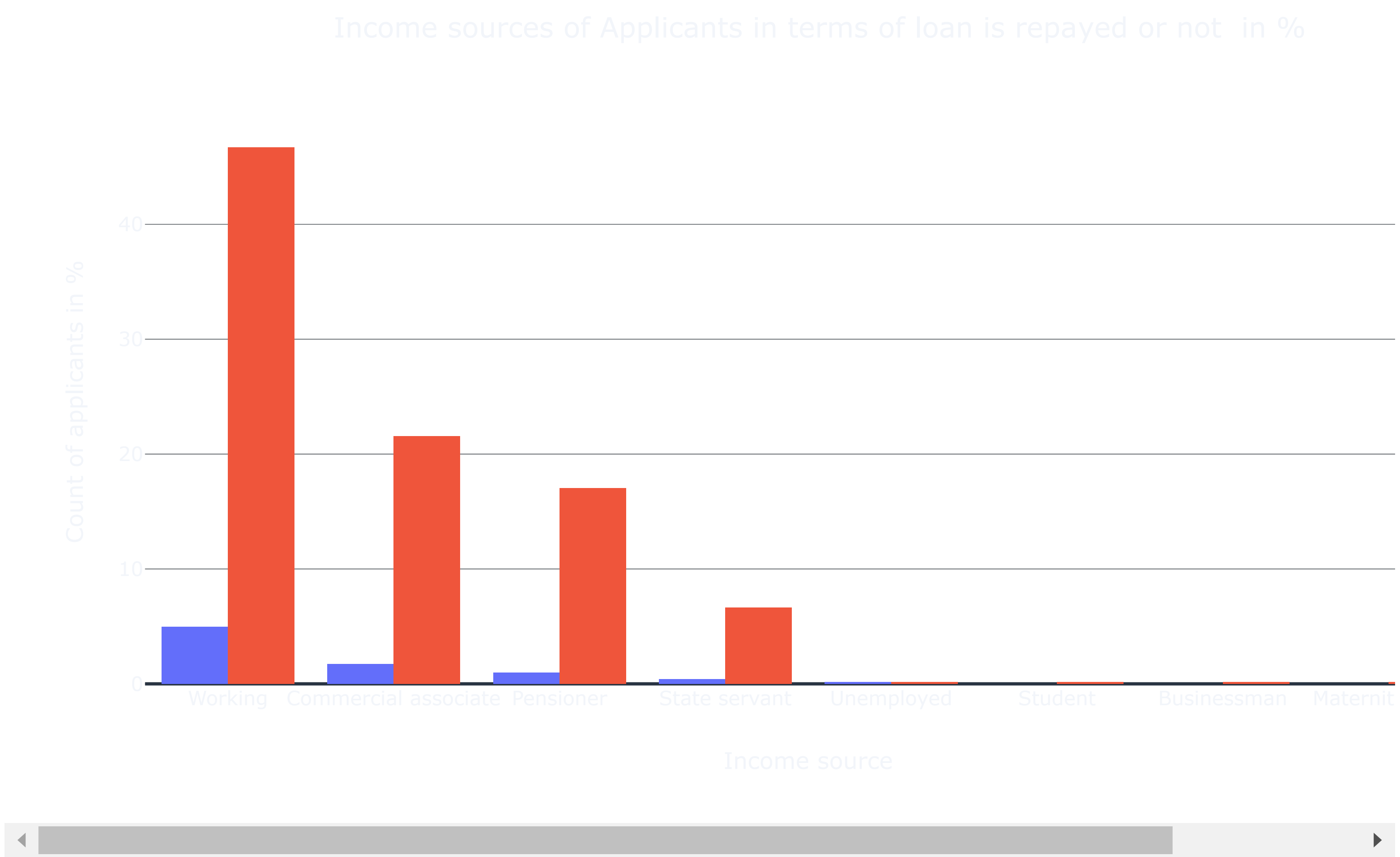
```
suite_val = application['NAME_TYPE_SUITE'].value_counts()

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 %',
        )
)

fig = go.Figure(data = data, layout=layout)
fig.layout.template = 'plotly_dark'


py.iplot(fig)
```



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

In [18]:

```
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(
        title='Income source',
       ),
    yaxis=dict(
        title='Count of applicants in %',
        )
)

fig = go.Figure(data = data, layout=layout)
fig.layout.template = 'plotly_dark'


py.iplot(fig)
```



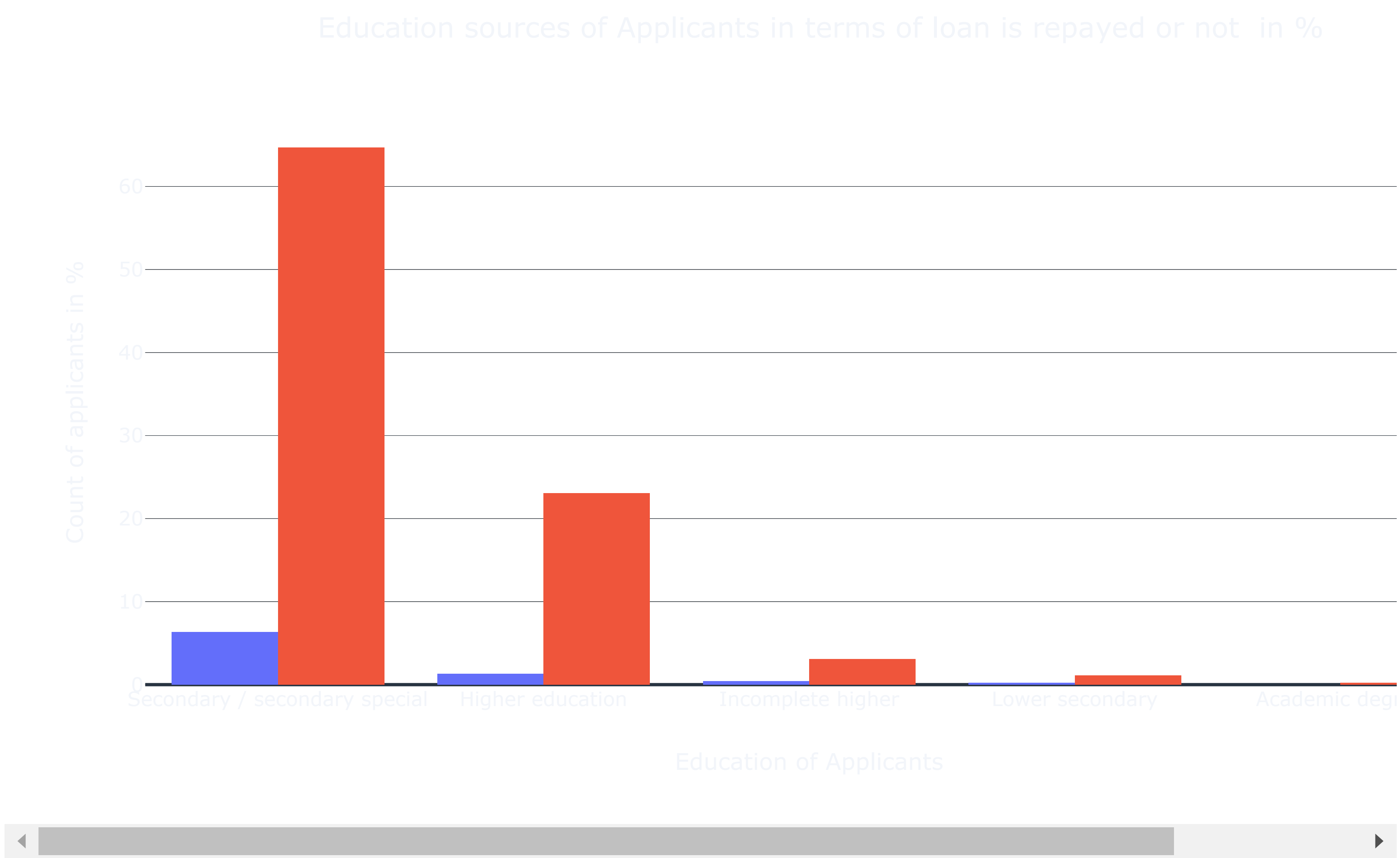
Observations:

- All the Students and Businessman are repaying loan.(Hover over the plot to observe)

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

In [19]:

```
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' )]

layout = go.Layout(
    title = "Education sources of Applicants in terms of loan is repayed or not  in %",
    xaxis=dict(
        title='Education of Applicants',
       ),
    yaxis=dict(
        title='Count of applicants in %',
        )
)

fig = go.Figure(data = data, layout=layout)
fig.layout.template = 'plotly_dark'


py.iplot(fig)
```

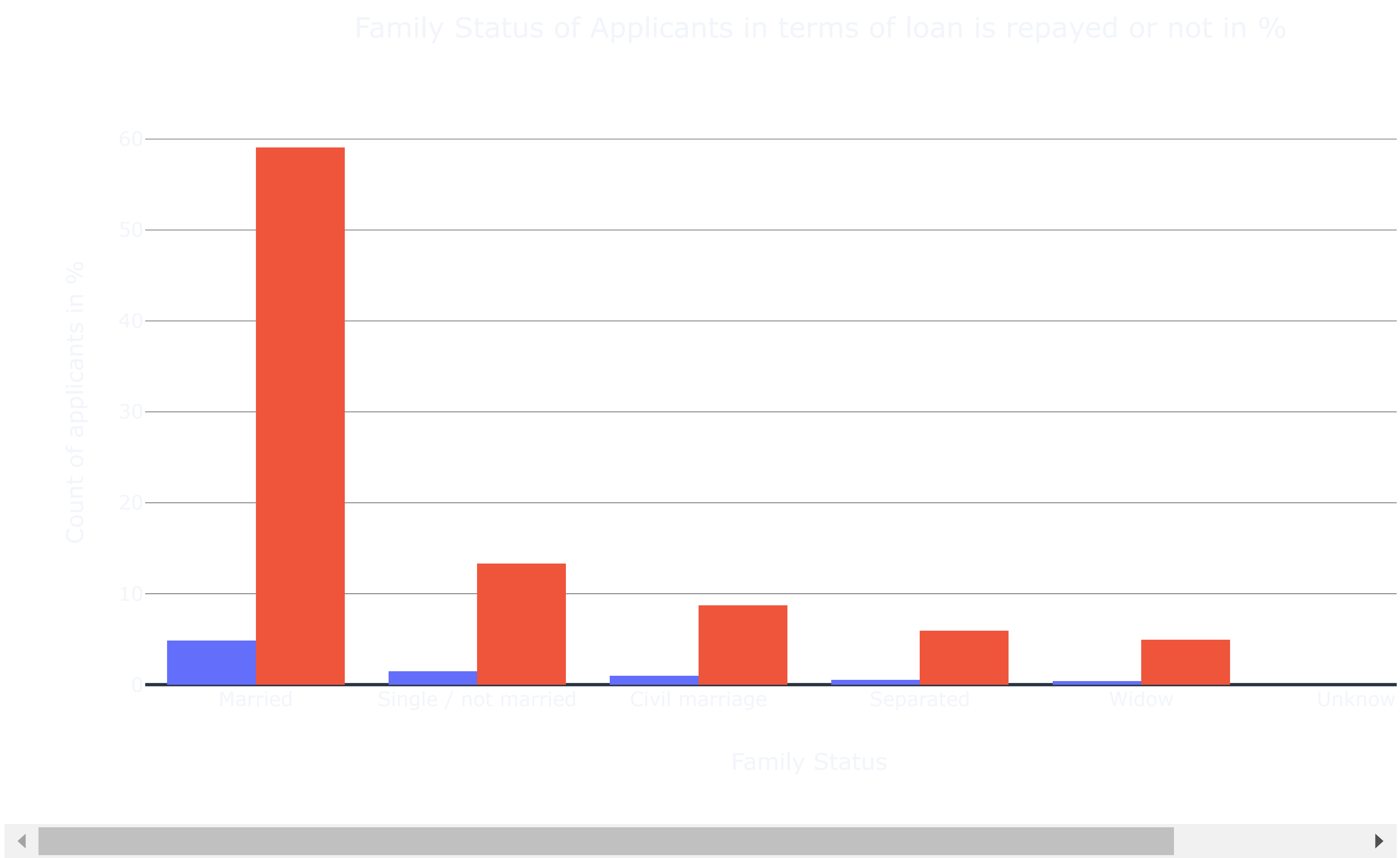


Observations:

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

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

In [20]:
```
family_val = application["NAME_FAMILY_STATUS"].value_counts()

family_val_y0 = []
family_val_y1 = []
for val in family_val.index:
    family_val_y1.append(np.sum(application["TARGET"][application["NAME_FAMILY_STATUS"]==val] == 1))
    family_val_y0.append(np.sum(application["TARGET"][application["NAME_FAMILY_STATUS"]==val] == 0))

data = [go.Bar(x = family_val.index, y = ((family_val_y1 / family_val.sum()) * 100), name='Yes' ),
        go.Bar(x = family_val.index, y = ((family_val_y0 / family_val.sum()) * 100), name='No' )]

layout = go.Layout(
    title = "Family Status of Applicants in terms of loan is repayed or not in %",
    xaxis=dict(
        title='Family Status',
       ),
    yaxis=dict(
        title='Count of applicants in %',
        )
)

fig = go.Figure(data = data, layout=layout)
fig.layout.template = 'plotly_dark'

py.iplot(fig)
```

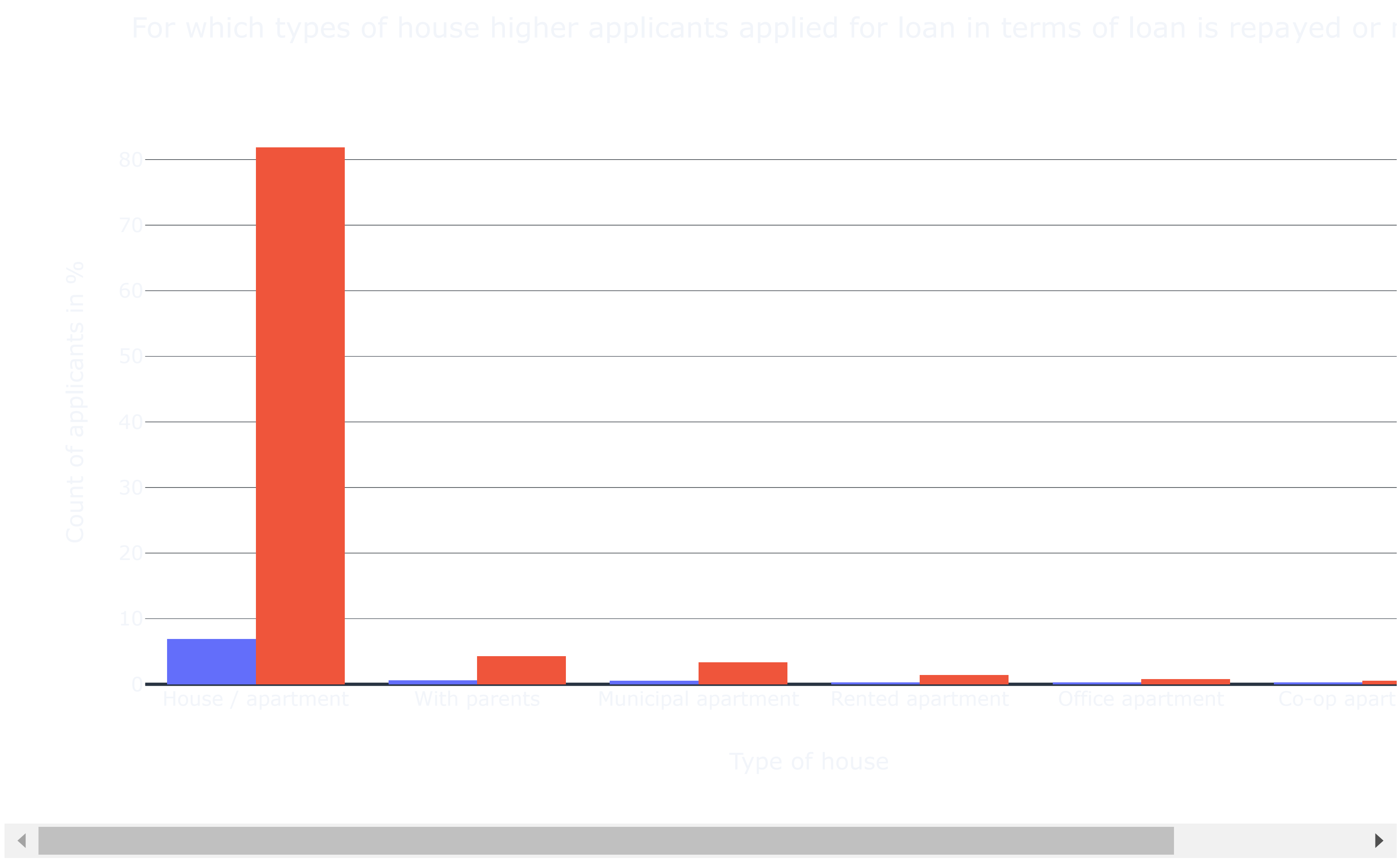


Observations:

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

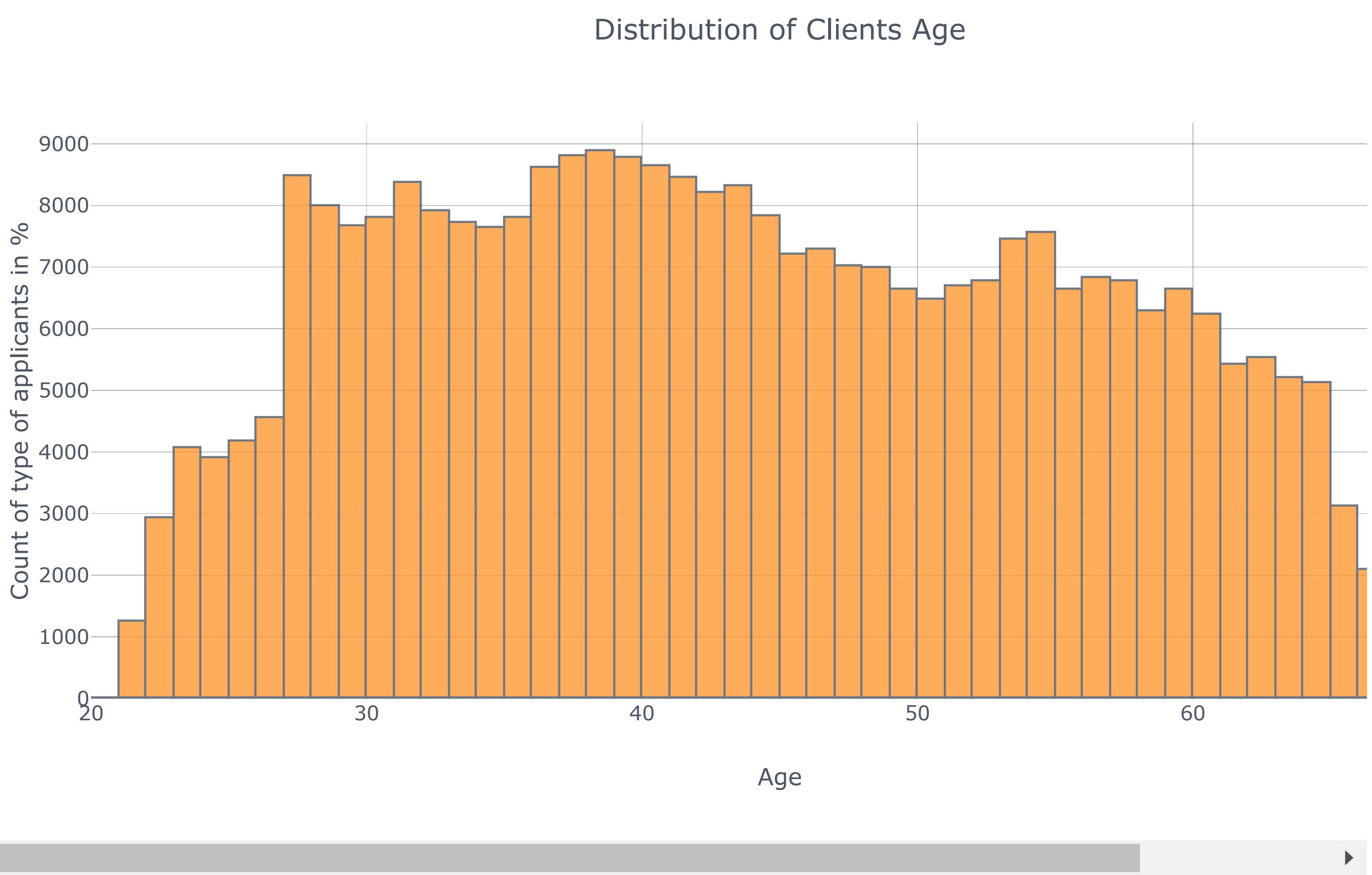
3.3.8 Distribution of Housing type of Applicants in terms of loan is repayed or not

In [21]:
```
housing_val = application['NAME_HOUSING_TYPE'].value_counts()

housing_val_y0 = []
housing_val_y1 = []
for val in housing_val.index:
    housing_val_y1.append(np.sum(application['TARGET'][application['NAME_HOUSING_TYPE']==val] == 1))
    housing_val_y0.append(np.sum(application['TARGET'][application['NAME_HOUSING_TYPE']==val] == 0))

data = [go.Bar(x = housing_val.index, y = ((housing_val_y1 / housing_val.sum()) * 100), name='Yes' ),
        go.Bar(x = housing_val.index, y = ((housing_val_y0 / housing_val.sum()) * 100), name='No' )]

layout = go.Layout(
    title = "For which types of house higher applicants applied for loan in terms of loan is repayed or not i
n %",
    xaxis=dict(
        title='Type of house',
       ),
    yaxis=dict(
        title='Count of applicants in %',
        )
)

fig = go.Figure(data = data, layout=layout)
fig.layout.template = 'plotly_dark'


py.iplot(fig)
```



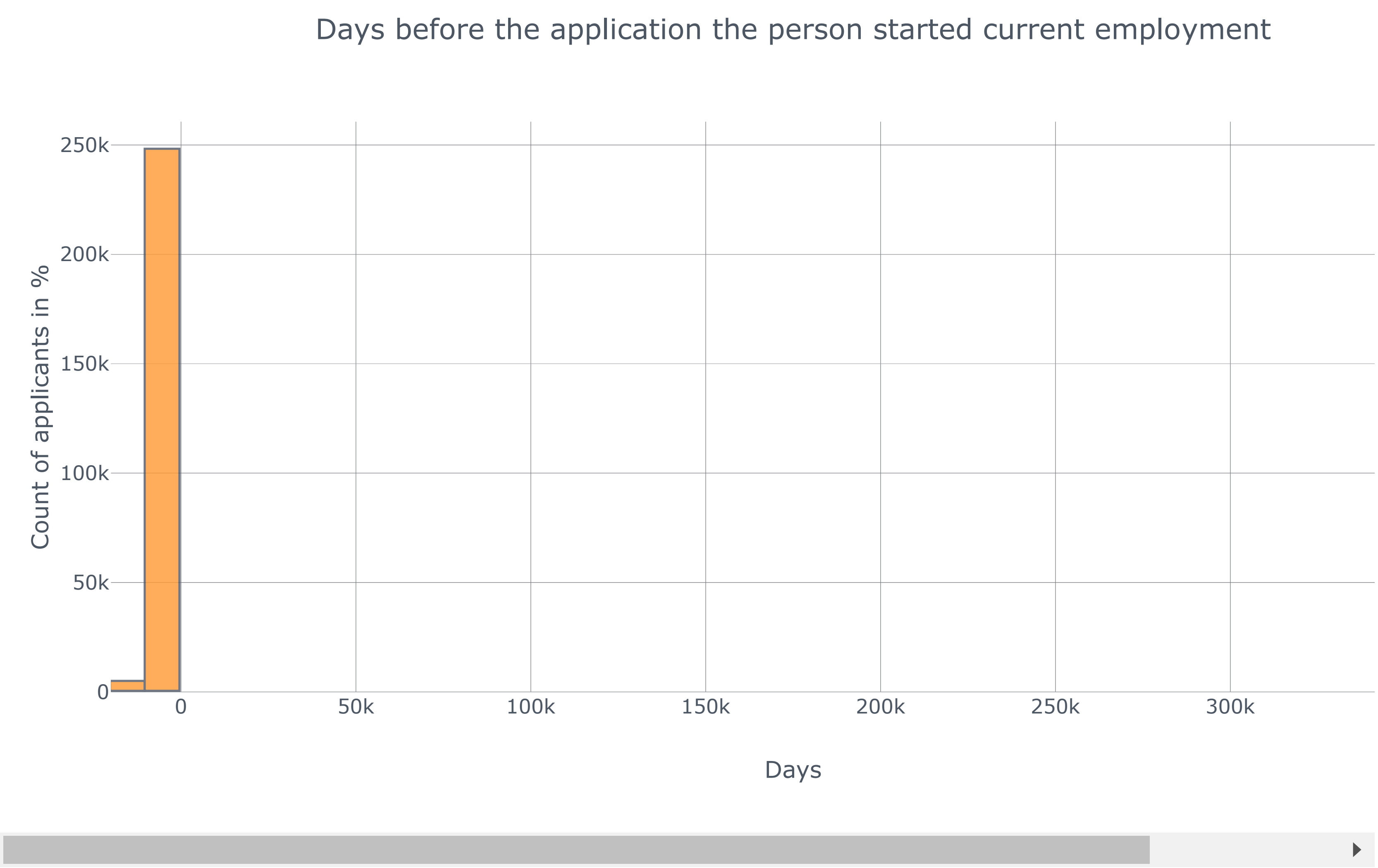
3.3.9 Distribution of Clients Age

In [22]:
```
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')
```



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

In [23]:
```
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')
```



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

In [24]:
```
application['DAYS_EMPLOYED'].describe()
```

Out[24]:
```
count    307511.000000
mean      63815.045904
std      141275.766519
min      -17912.000000
25%       -2760.000000
50%       -1213.000000
75%        -289.000000
max      365243.000000
Name: DAYS_EMPLOYED, dtype: float64
```

In [25]:
```
error = application[application['DAYS_EMPLOYED'] == 365243]
print('The no of errors are :', len(error))
(error['TARGET'].value_counts()/len(error))*100
```

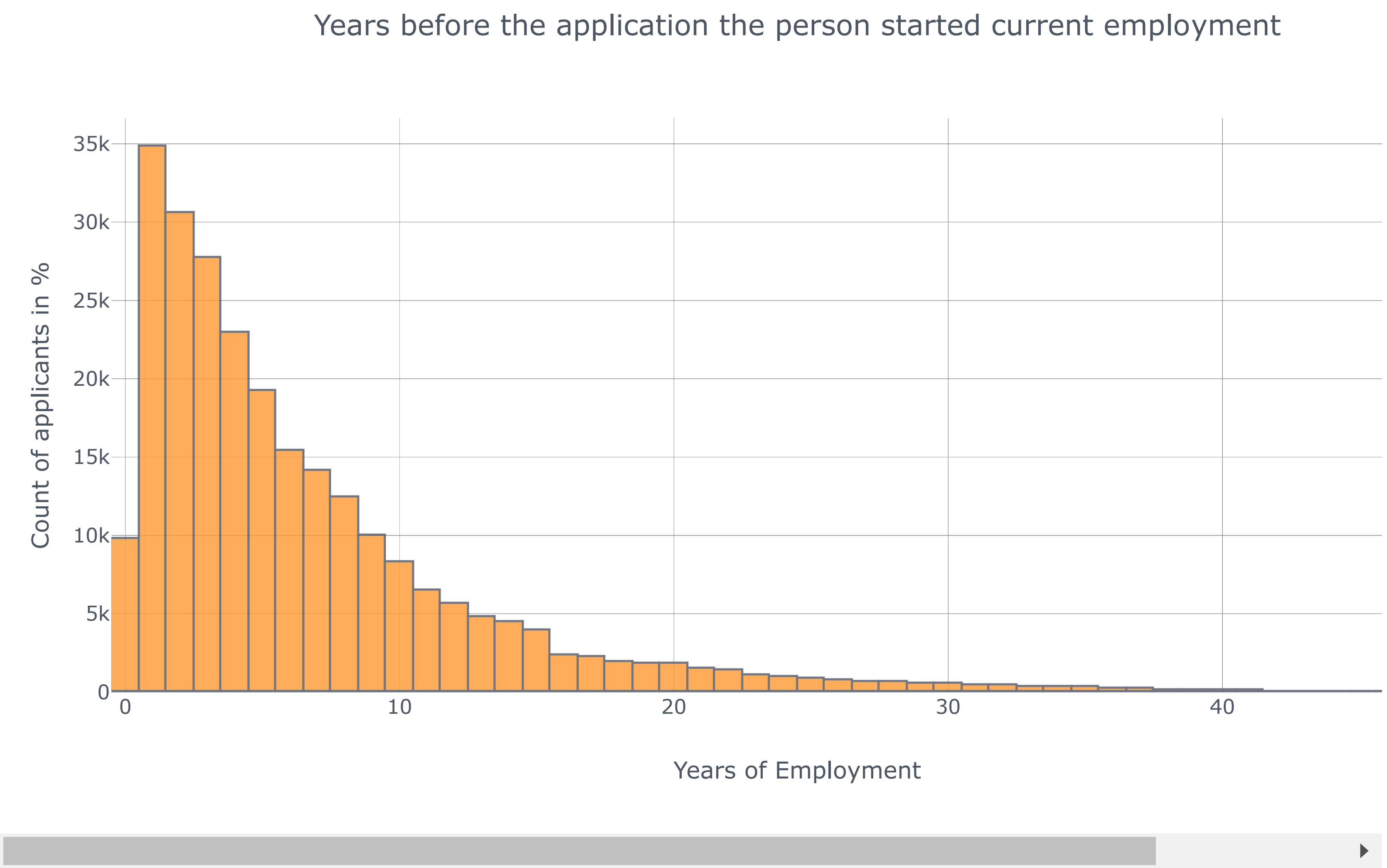
```
The no of errors are : 55374
```

Out[25]:
```
0    94.600354
1     5.399646
Name: TARGET, dtype: float64
```

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

In [26]:
```
# 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)
```

In [27]:
```
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')
```



In [28]:
```
application[application['DAYS_EMPLOYED']>(-365*2)]['TARGET'].value_counts()/sum(application['DAYS_EMPLOYED']>
(-365*2))
```

Out[28]:
```
0    0.887924
1    0.112076
Name: TARGET, dtype: float64
```

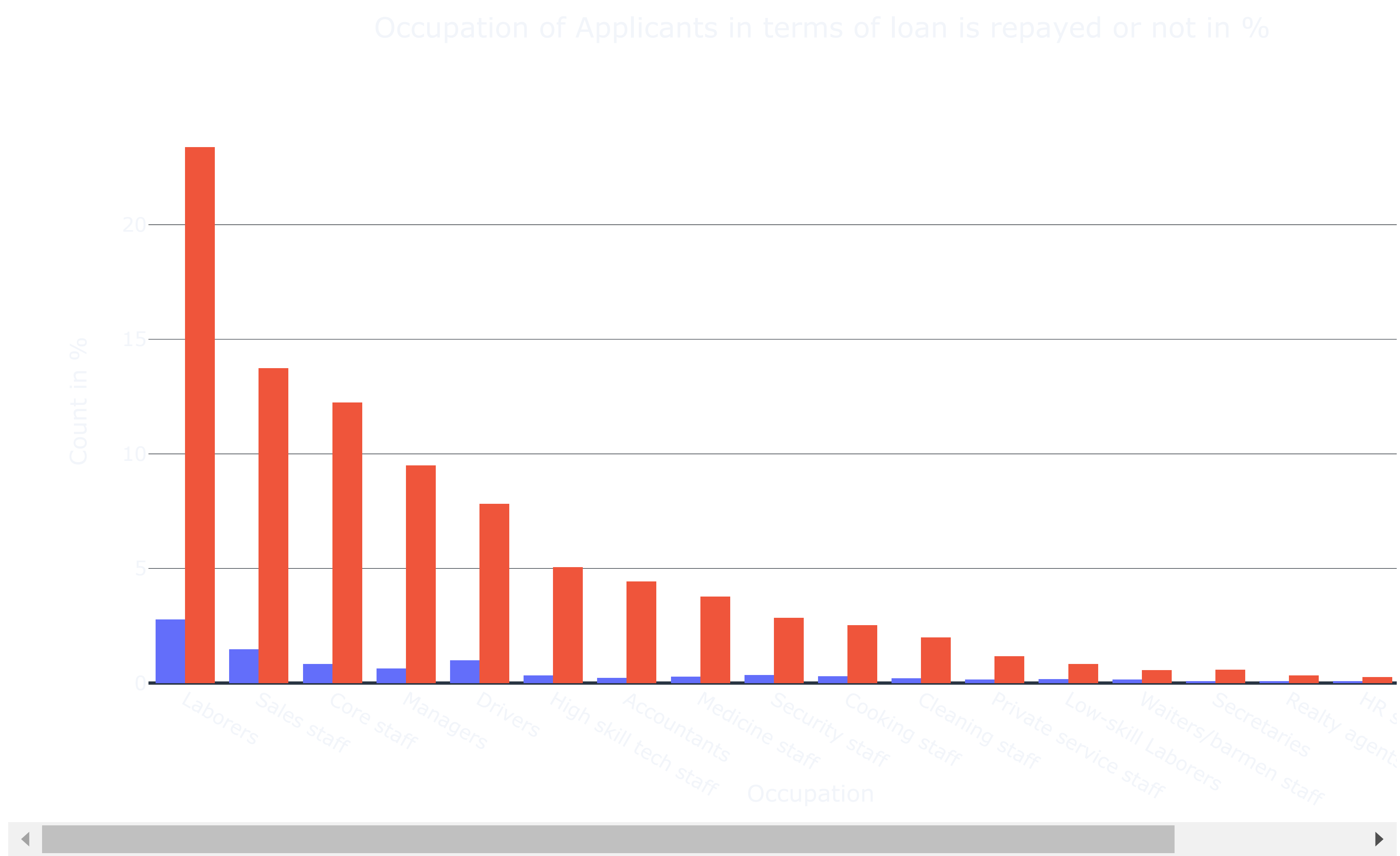
Observations:

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

3.3.11 Occupation of Applicants in terms of loan is repayed or not

In [29]:

```
occupation_val = application['OCCUPATION_TYPE'].value_counts()

occupation_val_y0 = []
occupation_val_y1 = []
for val in occupation_val.index:
    occupation_val_y1.append(np.sum(application['TARGET'][application['OCCUPATION_TYPE']==val] == 1))
    occupation_val_y0.append(np.sum(application['TARGET'][application['OCCUPATION_TYPE']==val] == 0))

data = [go.Bar(x = occupation_val.index, y = ((occupation_val_y1 / occupation_val.sum()) * 100), name='Yes'
),
        go.Bar(x = occupation_val.index, y = ((occupation_val_y0 / occupation_val.sum()) * 100), name='No' )]

layout = go.Layout(
    title = "Occupation of Applicants in terms of loan is repayed or not in %",
    xaxis=dict(
        title='Occupation ',
       ),
    yaxis=dict(
        title='Count in %',
        )
)

fig = go.Figure(data = data, layout=layout)
fig.layout.template = 'plotly_dark'


py.iplot(fig)
```



Observations:

- Core staff ,Managers, High skill tech staff, Accountants are more likely to repay when compared to Laborers, Sales staff, Drivers, Low-skill Laborers(very less likely to repay).

```
In [30]: application.shape
```

```
Out[30]: (307511, 123)
```

3.4 Preparation of data

3.4.1 Feature Engineering of Application data

```
In [31]: # 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']
```

```
In [32]: application.shape
```

```
Out[32]: (307511, 128)
```

3.4.2 Using Bureau Data

In [33]:
```python
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:')
bureau.head()
```

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

Out[33]:

	SK_ID_CURR	SK_ID_BUREAU	CREDIT_ACTIVE	CREDIT_CURRENCY	DAYS_CREDIT	CREDIT_DAY_OVERDUE	DAYS_CREDIT_ENDDA
0	215354	5714462	Closed	currency 1	-497	0	-15
1	215354	5714463	Active	currency 1	-208	0	107
2	215354	5714464	Active	currency 1	-203	0	52
3	215354	5714465	Active	currency 1	-203	0	N
4	215354	5714466	Active	currency 1	-629	0	119

In [34]:
```python
# 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))
```

In [35]:
```python
# 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))
```

In [36]:
```
application_bureau.shape
```

Out[36]:
```
(307511, 163)
```

3.4.3 Feature Engineering of Bureau Data

In [37]:
```
# Number of past loans per customer
grp = bureau.groupby(by = ['SK_ID_CURR'])['SK_ID_BUREAU'].count().reset_index().rename(columns = {'SK_ID_BURE
AU': '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)
```

In [38]:
```
# 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)
```

In [39]:
```
# 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_index
().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().r
eset_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'], downcast='fl
oat')
```

In [40]:
```
(application_bureau[application_bureau['DEBT_CREDIT_RATIO'] > 0.5]['TARGET'].value_counts()/len(application_b
ureau[application_bureau['DEBT_CREDIT_RATIO'] > 0.5]))*100
```

Out[40]:
```
0    91.927118
1     8.072882
Name: TARGET, dtype: float64
```

In [41]:
```
# Overdue over debt ratio
bureau['AMT_CREDIT_SUM_OVERDUE'] = bureau['AMT_CREDIT_SUM_OVERDUE'].fillna(0)
bureau['AMT_CREDIT_SUM_DEBT'] = bureau['AMT_CREDIT_SUM_DEBT'].fillna(0)

grp1 = bureau[['SK_ID_CURR','AMT_CREDIT_SUM_OVERDUE']].groupby(by=['SK_ID_CURR'])['AMT_CREDIT_SUM_OVERDUE'].s
um().reset_index().rename(columns={'AMT_CREDIT_SUM_OVERDUE': 'TOTAL_CUSTOMER_OVERDUE'})

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

grp1['OVERDUE_DEBT_RATIO'] = grp1['TOTAL_CUSTOMER_OVERDUE']/grp2['TOTAL_CUSTOMER_DEBT']

del grp1['TOTAL_CUSTOMER_OVERDUE']

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

application_bureau['OVERDUE_DEBT_RATIO'] = pd.to_numeric(application_bureau['OVERDUE_DEBT_RATIO'], downcast=
'float')
```

In [42]:
```
application_bureau.shape
```

Out[42]:
```
(307511, 167)
```

In [43]:
```
gc.collect()
```

Out[43]:
```
44354
```

3.4.4 Using Previous Application Data

In [44]:
```python
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:
```

Out[44]:

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

5 rows × 37 columns

In [45]:
```python
# Number of previous applications per customer
grp = previous_applicaton[['SK_ID_CURR','SK_ID_PREV']].groupby(by=['SK_ID_CURR'])['SK_ID_PREV'].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)
```

In [46]:
```python
# Combining numerical features
grp = previous_applicaton.drop('SK_ID_PREV', axis =1).groupby(by=['SK_ID_CURR']).mean().reset_index()
prev_columns = ['PREV_'+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))
```

In [47]:
```
# Combining categorical features
prev_categorical = pd.get_dummies(previous_applicaton.select_dtypes('object'))
prev_categorical['SK_ID_CURR'] = previous_applicaton['SK_ID_CURR']
prev_categorical.head()

grp = prev_categorical.groupby('SK_ID_CURR').mean().reset_index()
grp.columns = ['PREV_'+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))
```

3.4.5 Using POS_CASH_balance data

In [48]:
```
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:
```

Out[48]:

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

In [49]:
```
# 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))
```

In [50]:
```
# Combining categorical features
pos_cash_categorical = pd.get_dummies(pos_cash.select_dtypes('object'))
pos_cash_categorical['SK_ID_CURR'] = pos_cash['SK_ID_CURR']

grp = pos_cash_categorical.groupby('SK_ID_CURR').mean().reset_index()
grp.columns = ['POS_'+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))
```

3.4.6 Using installments_payments data

In [51]:
```
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:
```

Out[51]:

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

In [52]:
```
# 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))
```

3.4.7 Using Credit card balance data

In [53]:
```
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:
```

Out[53]:

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

5 rows × 23 columns

In [54]:
```
# 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))
```

In [55]:
```
# 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))
```

In [56]:
```
application_bureau_prev.shape
```
Out[56]:
```
(307511, 377)
```

3.5 Dividing data into train, valid and test

In [57]:
```
y = application_bureau_prev.pop('TARGET').values
```

In [58]:
```
X_train, X_temp, y_train, y_temp = train_test_split(application_bureau_prev.drop(['SK_ID_CURR'],axis=1), y, s
tratify = y, test_size=0.3, random_state=42)
X_val, X_test, y_val, y_test = train_test_split(X_temp, y_temp, stratify = y_temp, test_size=0.5, random_stat
e=42)
```

In [59]:
```
print('Shape of X_train:',X_train.shape)
print('Shape of X_val:',X_val.shape)
print('Shape of X_test:',X_test.shape)
```
```
Shape of X_train: (215257, 375)
Shape of X_val: (46127, 375)
Shape of X_test: (46127, 375)
```

3.6 Featurizing the data

In [60]:
```
# 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]
```

In [61]:
```python
# 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)
```

In [62]:
```python
# 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])

X_train_cat1= pd.DataFrame(X_train_cat, columns=cat_cols)
X_val_cat1= pd.DataFrame(X_val_cat, columns=cat_cols)
X_test_cat1= pd.DataFrame(X_test_cat, columns=cat_cols)

ohe = OneHotEncoder(sparse=False,handle_unknown='ignore')
X_train_cat2 = ohe.fit_transform(X_train_cat1)
X_val_cat2 = ohe.transform(X_val_cat1)
X_test_cat2 = ohe.transform(X_test_cat1)

cat_cols_ohe = list(ohe.get_feature_names(input_features=cat_cols))

X_train_cat_final = pd.DataFrame(X_train_cat2, columns = cat_cols_ohe)
X_val_cat_final = pd.DataFrame(X_val_cat2, columns = cat_cols_ohe)
X_test_cat_final = pd.DataFrame(X_test_cat2, columns = cat_cols_ohe)
```

In [63]:
```python
# To free up the unused memory
gc.collect()
```

Out[63]:
```
105
```

In [64]:
```python
# Final complete data
X_train_final = pd.concat([X_train_num_final,X_train_cat_final], axis = 1)
X_val_final = pd.concat([X_val_num_final,X_val_cat_final], axis = 1)
X_test_final = pd.concat([X_test_num_final,X_test_cat_final], axis = 1)
print(X_train_final.shape)
print(X_val_final.shape)
print(X_test_final.shape)
```

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

In [65]:
```python
# 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')
```

In [67]:
```python
# 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)
```

3.7 Selection of features

In [69]:
```
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
```

Out[69]:
```
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)
```

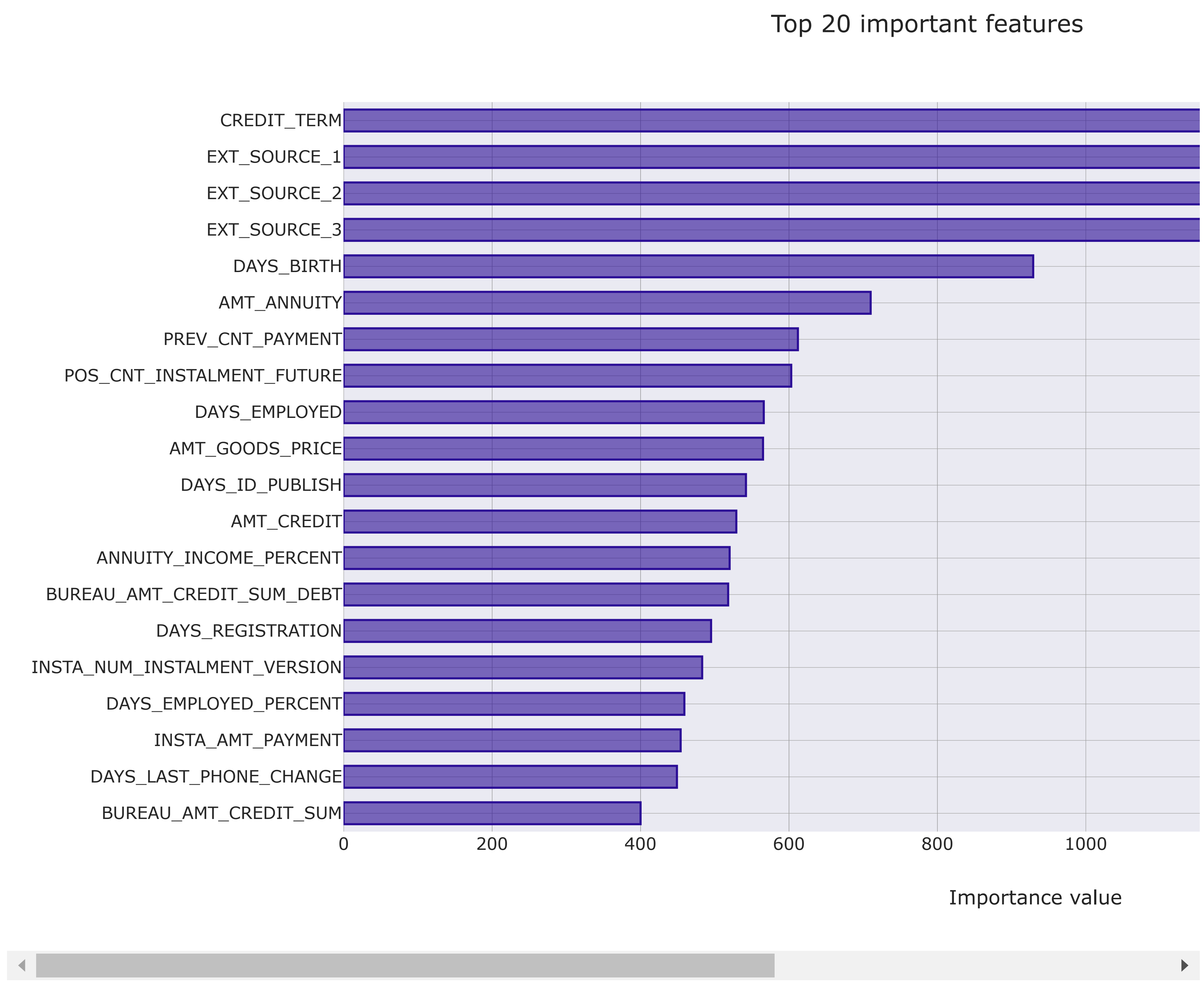
In [80]:
```
feature_imp = pd.DataFrame(sorted(zip(model_sk.feature_importances_, X_train_final.columns)), columns=['Valu
e','Feature'])
features_df = feature_imp.sort_values(by="Value", ascending=False)
selected_features = list(features_df[features_df['Value']>=50]['Feature'])
print('The no. of features selected:',len(selected_features))
```

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

In [75]:
```
# Saving the selected features into pickle file
with open('select_features.txt','wb') as fp:
    pickle.dump(selected_features, fp)
```

In [77]:

```
# 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'], orie
ntation = '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,
    width=1300,
    height=700,
    title = "Top 20 important features",
    xaxis=dict(
        title='Importance value'
        ),
    yaxis=dict(
        automargin=True
        ),
    bargap=0.4
    )

fig = go.Figure(data = data, layout=layout)
fig.layout.template = 'seaborn'


py.iplot(fig)
```


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
INSTA_NUM_INSTALMENT_VERSION
DAYS_EMPLOYED_PERCENT
INSTA_AMT_PAYMENT
DAYS_LAST_PHONE_CHANGE
BUREAU_AMT_CREDIT_SUM
0
200
400
600
800
1000
Importance value

4. Machine Learning Models

In [4]:
```
X_train_final = pd.read_csv('X_train_final.csv')
X_val_final = pd.read_csv('X_val_final.csv')
X_test_final = pd.read_csv('X_test_final.csv')

print('X_train_final',X_train_final.shape)
print('X_val_final',X_val_final.shape)
print('X_test_final',X_test_final.shape)
```

```
X_train_final (215257, 506)
X_val_final (46127, 506)
X_test_final (46127, 506)
```

In [5]:
```
with open('select_features.txt', 'rb') as fp:
    selected_features = pickle.load(fp)
```

In [6]:
```
y_train = np.loadtxt('y_train.txt')
y_val = np.loadtxt('y_val.txt')
y_test = np.loadtxt('y_test.txt')
```

In [7]:
```
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()
```

In [8]:

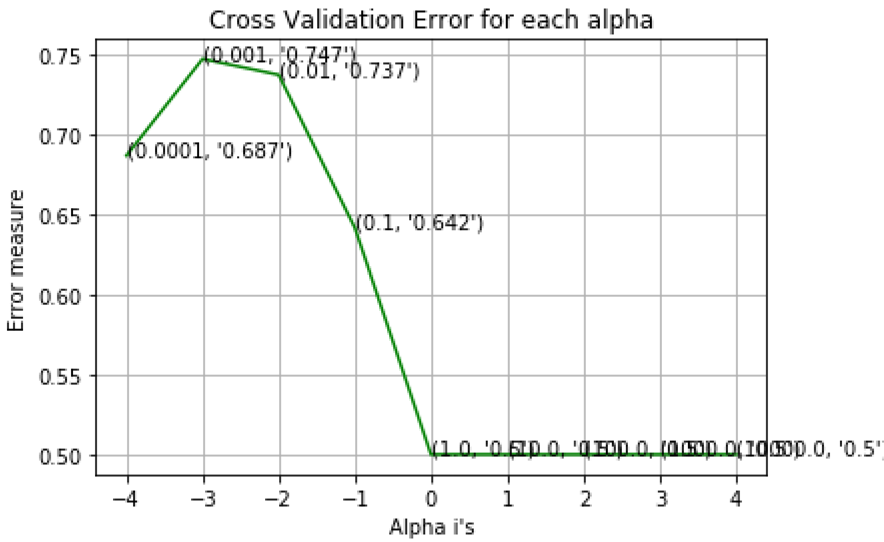
```
def cv_plot(alpha, cv_auc):

    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")
    plt.show()
```

4.1 Logistic regression with selected features

In [13]:

```
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)])
```

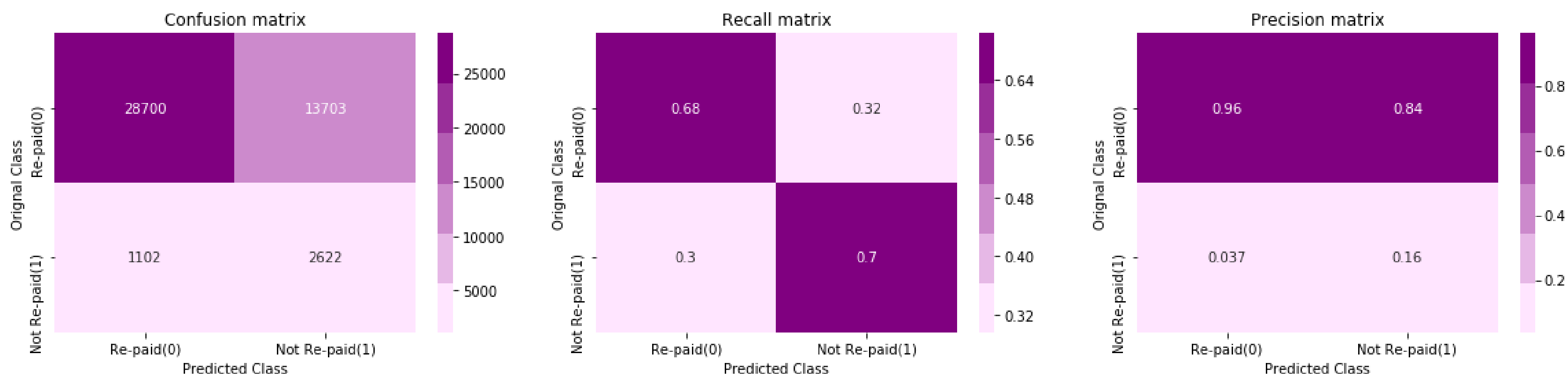
```
For alpha 0.0001, cross validation AUC score 0.6866034586096332
For alpha 0.001, cross validation AUC score 0.7470986349004096
For alpha 0.01, cross validation AUC score 0.737171244672842
For alpha 0.1, cross validation AUC score 0.641540949352706
For alpha 1.0, cross validation AUC score 0.5
For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



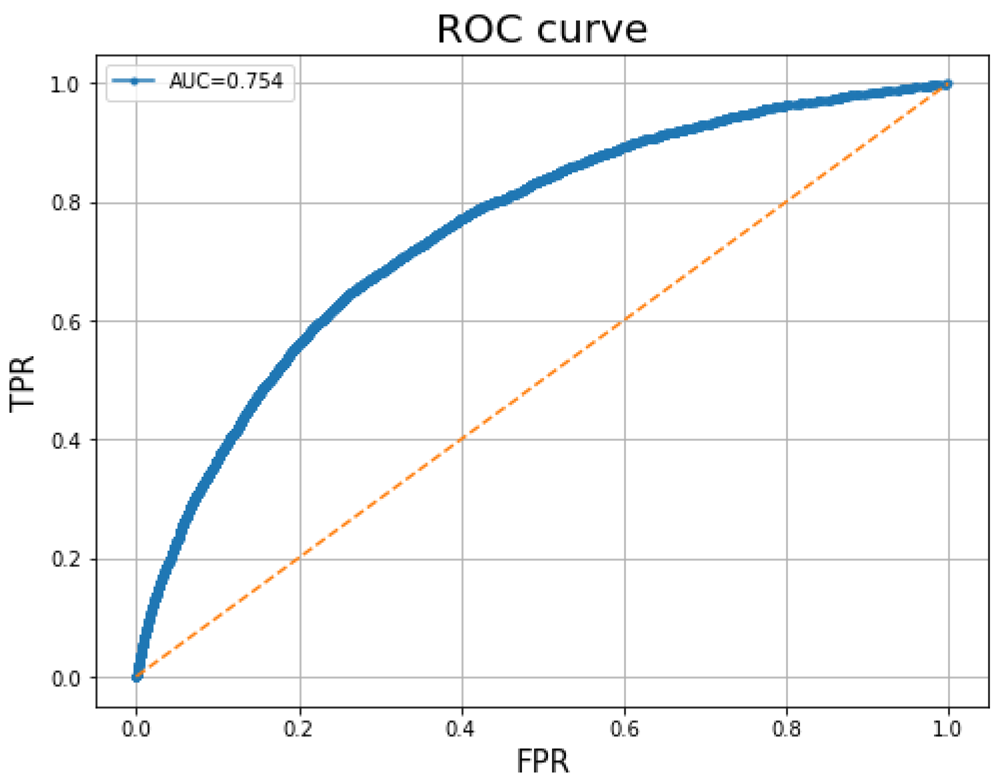

```
The Optimal C value is: 0.001
```

In [14]:
```
best_alpha = alpha[np.argmax(cv_auc_score)]
logreg = SGDClassifier(alpha = best_alpha, class_weight = 'balanced', penalty = 'l1', loss='log', random_stat
e = 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_pre
d_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% :
```



In [22]:
```
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()
```



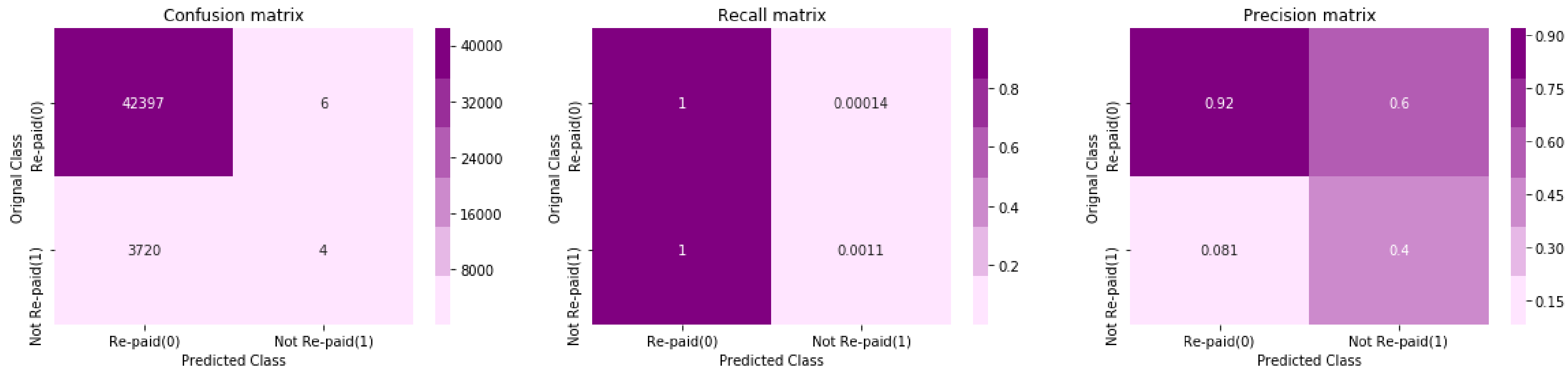
4.2 Random Forest with selected features

In [23]:
```
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
```

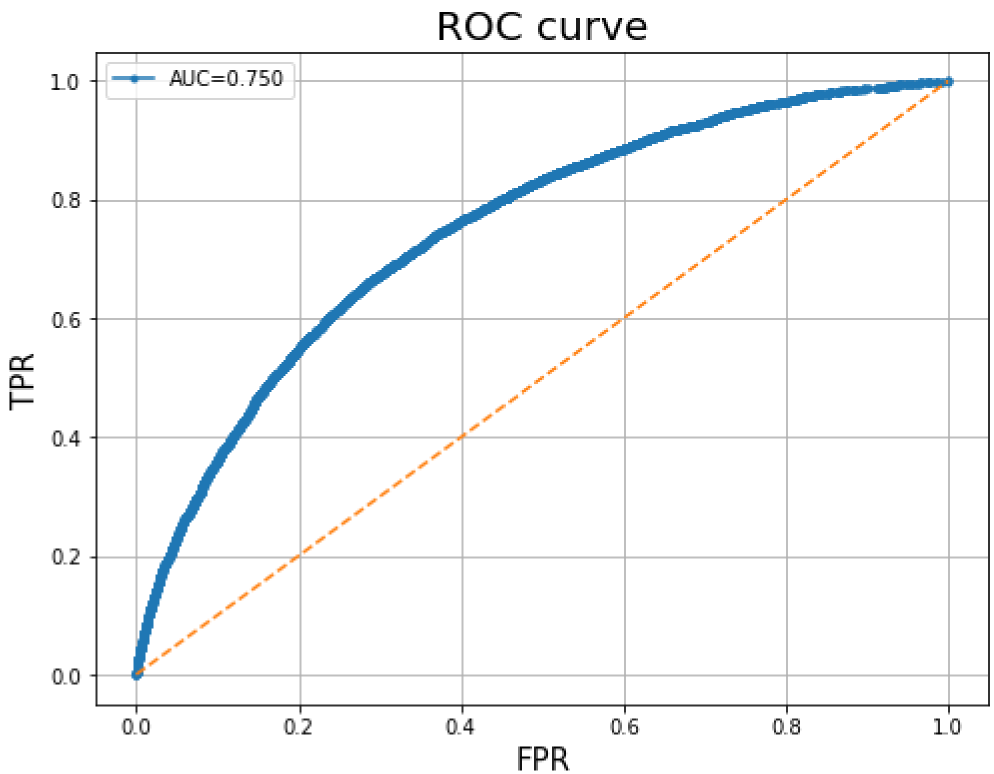
In [24]:
```
best_alpha = np.argmax(cv_auc_score)
print('The optimal values are: n_estimators {0}, max_depth {1} '.format(alpha[int(best_alpha/2)],
                                                                         max_depth[int(best_alpha%2)]))
rf = RandomForestClassifier(n_estimators=alpha[int(best_alpha/2)], 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_a
lpha/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_p
red_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% :
```



In [25]:

```
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()
```



4.3 LightGBM with selected features

In [26]:
```
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
```

In [27]:

```
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_sco
re(y_val,y_pred_prob)))
```

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

```
Training until validation scores don't improve for 100 rounds.
[10]	valid_0's auc: 0.719903
[20]	valid_0's auc: 0.72316
[30]	valid_0's auc: 0.728471
[40]	valid_0's auc: 0.734165
[50]	valid_0's auc: 0.738478
[60]	valid_0's auc: 0.743034
[70]	valid_0's auc: 0.747074
[80]	valid_0's auc: 0.750124
[90]	valid_0's auc: 0.752539
[100]	valid_0's auc: 0.754766
[110]	valid_0's auc: 0.75667
[120]	valid_0's auc: 0.758271
[130]	valid_0's auc: 0.759587
[140]	valid_0's auc: 0.761137
[150]	valid_0's auc: 0.762085
[160]	valid_0's auc: 0.76321
[170]	valid_0's auc: 0.764334
[180]	valid_0's auc: 0.765275
[190]	valid_0's auc: 0.766105
[200]	valid_0's auc: 0.766597
[210]	valid_0's auc: 0.767176
[220]	valid_0's auc: 0.767686
[230]	valid_0's auc: 0.76829
[240]	valid_0's auc: 0.768839
[250]	valid_0's auc: 0.769324
[260]	valid_0's auc: 0.76984
[270]	valid_0's auc: 0.770288
[280]	valid_0's auc: 0.770764
[290]	valid_0's auc: 0.771161
[300]	valid_0's auc: 0.771475
[310]	valid_0's auc: 0.771749
[320]	valid_0's auc: 0.772057
[330]	valid_0's auc: 0.772475
[340]	valid_0's auc: 0.772851
[350]	valid_0's auc: 0.773227
[360]	valid_0's auc: 0.773536
[370]	valid_0's auc: 0.773817
[380]	valid_0's auc: 0.7742
[390]	valid_0's auc: 0.774464
[400]	valid_0's auc: 0.77469
[410]	valid_0's auc: 0.774908
[420]	valid_0's auc: 0.775107
```

```
[430]	valid_0's auc: 0.775212
[440]	valid_0's auc: 0.775434
[450]	valid_0's auc: 0.775802
[460]	valid_0's auc: 0.775911
[470]	valid_0's auc: 0.776008
[480]	valid_0's auc: 0.776169
[490]	valid_0's auc: 0.776496
[500]	valid_0's auc: 0.776681
[510]	valid_0's auc: 0.776883
[520]	valid_0's auc: 0.776977
[530]	valid_0's auc: 0.777041
[540]	valid_0's auc: 0.777226
[550]	valid_0's auc: 0.777433
[560]	valid_0's auc: 0.777432
[570]	valid_0's auc: 0.777643
[580]	valid_0's auc: 0.777695
[590]	valid_0's auc: 0.777852
[600]	valid_0's auc: 0.777906
[610]	valid_0's auc: 0.777971
[620]	valid_0's auc: 0.778076
[630]	valid_0's auc: 0.778072
[640]	valid_0's auc: 0.778174
[650]	valid_0's auc: 0.778262
[660]	valid_0's auc: 0.778346
[670]	valid_0's auc: 0.778521
[680]	valid_0's auc: 0.778642
[690]	valid_0's auc: 0.778633
[700]	valid_0's auc: 0.7787
[710]	valid_0's auc: 0.778882
[720]	valid_0's auc: 0.778934
[730]	valid_0's auc: 0.778975
[740]	valid_0's auc: 0.778991
[750]	valid_0's auc: 0.77905
[760]	valid_0's auc: 0.779022
[770]	valid_0's auc: 0.779133
[780]	valid_0's auc: 0.779165
[790]	valid_0's auc: 0.779279
[800]	valid_0's auc: 0.77933
[810]	valid_0's auc: 0.779416
[820]	valid_0's auc: 0.779411
[830]	valid_0's auc: 0.779436
[840]	valid_0's auc: 0.779407
[850]	valid_0's auc: 0.779483
```

```
[860]   valid_0's auc: 0.779577
[870]   valid_0's auc: 0.779561
[880]   valid_0's auc: 0.779577
[890]   valid_0's auc: 0.77962
[900]   valid_0's auc: 0.779638
[910]   valid_0's auc: 0.779663
[920]   valid_0's auc: 0.779772
[930]   valid_0's auc: 0.779788
[940]   valid_0's auc: 0.779811
[950]   valid_0's auc: 0.77983
[960]   valid_0's auc: 0.779831
[970]   valid_0's auc: 0.779886
[980]   valid_0's auc: 0.779945
[990]   valid_0's auc: 0.780039
[1000]  valid_0's auc: 0.780126
[1010]  valid_0's auc: 0.780016
[1020]  valid_0's auc: 0.780022
[1030]  valid_0's auc: 0.780057
[1040]  valid_0's auc: 0.780037
[1050]  valid_0's auc: 0.780094
[1060]  valid_0's auc: 0.78006
[1070]  valid_0's auc: 0.780048
[1080]  valid_0's auc: 0.780095
[1090]  valid_0's auc: 0.780195
[1100]  valid_0's auc: 0.780256
[1110]  valid_0's auc: 0.780274
[1120]  valid_0's auc: 0.780329
[1130]  valid_0's auc: 0.780401
[1140]  valid_0's auc: 0.780471
[1150]  valid_0's auc: 0.780498
[1160]  valid_0's auc: 0.780556
[1170]  valid_0's auc: 0.78059
[1180]  valid_0's auc: 0.780581
[1190]  valid_0's auc: 0.780567
[1200]  valid_0's auc: 0.780627
[1210]  valid_0's auc: 0.780617
[1220]  valid_0's auc: 0.780644
[1230]  valid_0's auc: 0.780606
[1240]  valid_0's auc: 0.780643
[1250]  valid_0's auc: 0.78061
[1260]  valid_0's auc: 0.780591
[1270]  valid_0's auc: 0.780644
[1280]  valid_0's auc: 0.780644
```

```
[1290]  valid_0's auc: 0.780583
[1300]  valid_0's auc: 0.780601
[1310]  valid_0's auc: 0.780647
[1320]  valid_0's auc: 0.780676
[1330]  valid_0's auc: 0.780691
[1340]  valid_0's auc: 0.780621
[1350]  valid_0's auc: 0.780568
[1360]  valid_0's auc: 0.78059
[1370]  valid_0's auc: 0.780568
[1380]  valid_0's auc: 0.78058
[1390]  valid_0's auc: 0.78067
[1400]  valid_0's auc: 0.780699
[1410]  valid_0's auc: 0.780801
[1420]  valid_0's auc: 0.780819
[1430]  valid_0's auc: 0.780803
[1440]  valid_0's auc: 0.780871
[1450]  valid_0's auc: 0.780888
[1460]  valid_0's auc: 0.780891
[1470]  valid_0's auc: 0.780884
[1480]  valid_0's auc: 0.780899
[1490]  valid_0's auc: 0.780928
[1500]  valid_0's auc: 0.780963
[1510]  valid_0's auc: 0.780992
[1520]  valid_0's auc: 0.78097
[1530]  valid_0's auc: 0.781021
[1540]  valid_0's auc: 0.781005
[1550]  valid_0's auc: 0.781004
[1560]  valid_0's auc: 0.781033
[1570]  valid_0's auc: 0.781022
[1580]  valid_0's auc: 0.780992
[1590]  valid_0's auc: 0.781013
[1600]  valid_0's auc: 0.780994
[1610]  valid_0's auc: 0.780967
[1620]  valid_0's auc: 0.780907
[1630]  valid_0's auc: 0.780929
[1640]  valid_0's auc: 0.780968
[1650]  valid_0's auc: 0.78094
[1660]  valid_0's auc: 0.780975
Early stopping, best iteration is:
[1562]  valid_0's auc: 0.781065
For  max_depth 3 and some other parameters, cross validation AUC score 0.7810649999861946
Training until validation scores don't improve for 100 rounds.
[10]    valid_0's auc: 0.733883
```

```
[20]	valid_0's auc: 0.737245
[30]	valid_0's auc: 0.741839
[40]	valid_0's auc: 0.746952
[50]	valid_0's auc: 0.751237
[60]	valid_0's auc: 0.75459
[70]	valid_0's auc: 0.758037
[80]	valid_0's auc: 0.7607
[90]	valid_0's auc: 0.76256
[100]	valid_0's auc: 0.764271
[110]	valid_0's auc: 0.765793
[120]	valid_0's auc: 0.766989
[130]	valid_0's auc: 0.76796
[140]	valid_0's auc: 0.768908
[150]	valid_0's auc: 0.769634
[160]	valid_0's auc: 0.770513
[170]	valid_0's auc: 0.771374
[180]	valid_0's auc: 0.772108
[190]	valid_0's auc: 0.772857
[200]	valid_0's auc: 0.773255
[210]	valid_0's auc: 0.77366
[220]	valid_0's auc: 0.774211
[230]	valid_0's auc: 0.77459
[240]	valid_0's auc: 0.774998
[250]	valid_0's auc: 0.775301
[260]	valid_0's auc: 0.77566
[270]	valid_0's auc: 0.775976
[280]	valid_0's auc: 0.776231
[290]	valid_0's auc: 0.776509
[300]	valid_0's auc: 0.776696
[310]	valid_0's auc: 0.776899
[320]	valid_0's auc: 0.777152
[330]	valid_0's auc: 0.777459
[340]	valid_0's auc: 0.777564
[350]	valid_0's auc: 0.777707
[360]	valid_0's auc: 0.777745
[370]	valid_0's auc: 0.777858
[380]	valid_0's auc: 0.778203
[390]	valid_0's auc: 0.77829
[400]	valid_0's auc: 0.778348
[410]	valid_0's auc: 0.77859
[420]	valid_0's auc: 0.778649
[430]	valid_0's auc: 0.778658
[440]	valid_0's auc: 0.778667
```

```
[450]	valid_0's auc: 0.778798
[460]	valid_0's auc: 0.778814
[470]	valid_0's auc: 0.778876
[480]	valid_0's auc: 0.779002
[490]	valid_0's auc: 0.779204
[500]	valid_0's auc: 0.779257
[510]	valid_0's auc: 0.779323
[520]	valid_0's auc: 0.779418
[530]	valid_0's auc: 0.779519
[540]	valid_0's auc: 0.779597
[550]	valid_0's auc: 0.779662
[560]	valid_0's auc: 0.779657
[570]	valid_0's auc: 0.779608
[580]	valid_0's auc: 0.779635
[590]	valid_0's auc: 0.779724
[600]	valid_0's auc: 0.779884
[610]	valid_0's auc: 0.779861
[620]	valid_0's auc: 0.779825
[630]	valid_0's auc: 0.77969
[640]	valid_0's auc: 0.77967
[650]	valid_0's auc: 0.779685
[660]	valid_0's auc: 0.779592
[670]	valid_0's auc: 0.779645
[680]	valid_0's auc: 0.779641
[690]	valid_0's auc: 0.779677
[700]	valid_0's auc: 0.779674
Early stopping, best iteration is:
[608]	valid_0's auc: 0.779903
For  max_depth 5 and some other parameters, cross validation AUC score 0.779903493898363
Training until validation scores don't improve for 100 rounds.
[10]	valid_0's auc: 0.737725
[20]	valid_0's auc: 0.74166
[30]	valid_0's auc: 0.747049
[40]	valid_0's auc: 0.751797
[50]	valid_0's auc: 0.756442
[60]	valid_0's auc: 0.759065
[70]	valid_0's auc: 0.762413
[80]	valid_0's auc: 0.765047
[90]	valid_0's auc: 0.766645
[100]	valid_0's auc: 0.768242
[110]	valid_0's auc: 0.770086
[120]	valid_0's auc: 0.771561
[130]	valid_0's auc: 0.772272
```

```
[140]	valid_0's auc: 0.77291
[150]	valid_0's auc: 0.773725
[160]	valid_0's auc: 0.774545
[170]	valid_0's auc: 0.775105
[180]	valid_0's auc: 0.775824
[190]	valid_0's auc: 0.776511
[200]	valid_0's auc: 0.776782
[210]	valid_0's auc: 0.777065
[220]	valid_0's auc: 0.777558
[230]	valid_0's auc: 0.77797
[240]	valid_0's auc: 0.778404
[250]	valid_0's auc: 0.778523
[260]	valid_0's auc: 0.778894
[270]	valid_0's auc: 0.779132
[280]	valid_0's auc: 0.779316
[290]	valid_0's auc: 0.779443
[300]	valid_0's auc: 0.779676
[310]	valid_0's auc: 0.77978
[320]	valid_0's auc: 0.780035
[330]	valid_0's auc: 0.780217
[340]	valid_0's auc: 0.78036
[350]	valid_0's auc: 0.780468
[360]	valid_0's auc: 0.780719
[370]	valid_0's auc: 0.780864
[380]	valid_0's auc: 0.780844
[390]	valid_0's auc: 0.780873
[400]	valid_0's auc: 0.78074
[410]	valid_0's auc: 0.780753
[420]	valid_0's auc: 0.7808
[430]	valid_0's auc: 0.780764
[440]	valid_0's auc: 0.780751
[450]	valid_0's auc: 0.780818
[460]	valid_0's auc: 0.780784
[470]	valid_0's auc: 0.780938
[480]	valid_0's auc: 0.781042
[490]	valid_0's auc: 0.781158
[500]	valid_0's auc: 0.781212
[510]	valid_0's auc: 0.781226
[520]	valid_0's auc: 0.781167
[530]	valid_0's auc: 0.781237
[540]	valid_0's auc: 0.781231
[550]	valid_0's auc: 0.781285
[560]	valid_0's auc: 0.781262
```

```
[570]	valid_0's auc: 0.781187
[580]	valid_0's auc: 0.781177
[590]	valid_0's auc: 0.78128
[600]	valid_0's auc: 0.781183
[610]	valid_0's auc: 0.781143
[620]	valid_0's auc: 0.781088
[630]	valid_0's auc: 0.780825
[640]	valid_0's auc: 0.78099
[650]	valid_0's auc: 0.780951
[660]	valid_0's auc: 0.780821
Early stopping, best iteration is:
[565]	valid_0's auc: 0.781356
For  max_depth 7 and some other parameters, cross validation AUC score 0.7813557818054592
Training until validation scores don't improve for 100 rounds.
[10]	valid_0's auc: 0.737512
[20]	valid_0's auc: 0.742569
[30]	valid_0's auc: 0.748226
[40]	valid_0's auc: 0.752636
[50]	valid_0's auc: 0.757438
[60]	valid_0's auc: 0.760858
[70]	valid_0's auc: 0.764328
[80]	valid_0's auc: 0.766632
[90]	valid_0's auc: 0.768313
[100]	valid_0's auc: 0.76989
[110]	valid_0's auc: 0.771375
[120]	valid_0's auc: 0.772301
[130]	valid_0's auc: 0.773232
[140]	valid_0's auc: 0.774115
[150]	valid_0's auc: 0.774698
[160]	valid_0's auc: 0.775521
[170]	valid_0's auc: 0.776208
[180]	valid_0's auc: 0.777143
[190]	valid_0's auc: 0.777731
[200]	valid_0's auc: 0.777994
[210]	valid_0's auc: 0.778372
[220]	valid_0's auc: 0.778724
[230]	valid_0's auc: 0.778998
[240]	valid_0's auc: 0.779136
[250]	valid_0's auc: 0.779341
[260]	valid_0's auc: 0.779481
[270]	valid_0's auc: 0.779541
[280]	valid_0's auc: 0.779698
[290]	valid_0's auc: 0.779839
```

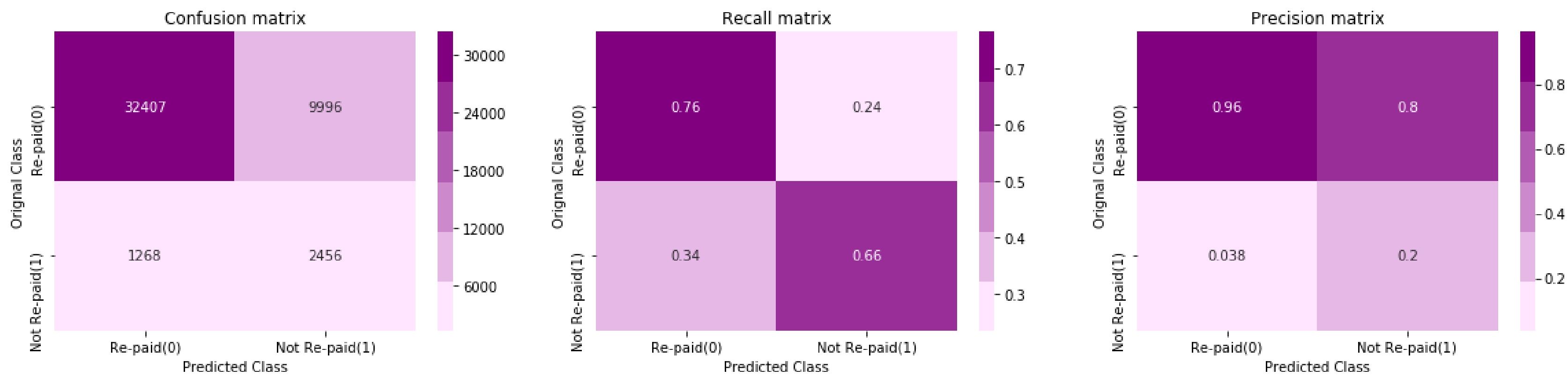
```
[300]	valid_0's auc: 0.780145
[310]	valid_0's auc: 0.780344
[320]	valid_0's auc: 0.780523
[330]	valid_0's auc: 0.780515
[340]	valid_0's auc: 0.780579
[350]	valid_0's auc: 0.780562
[360]	valid_0's auc: 0.780642
[370]	valid_0's auc: 0.780547
[380]	valid_0's auc: 0.780662
[390]	valid_0's auc: 0.780882
[400]	valid_0's auc: 0.780954
[410]	valid_0's auc: 0.781015
[420]	valid_0's auc: 0.781127
[430]	valid_0's auc: 0.781108
[440]	valid_0's auc: 0.78118
[450]	valid_0's auc: 0.781291
[460]	valid_0's auc: 0.781473
[470]	valid_0's auc: 0.781412
[480]	valid_0's auc: 0.781371
[490]	valid_0's auc: 0.781442
[500]	valid_0's auc: 0.781396
[510]	valid_0's auc: 0.781355
[520]	valid_0's auc: 0.781295
[530]	valid_0's auc: 0.781213
[540]	valid_0's auc: 0.781226
[550]	valid_0's auc: 0.78122
[560]	valid_0's auc: 0.781258
[570]	valid_0's auc: 0.781161
[580]	valid_0's auc: 0.781066
Early stopping, best iteration is:
[483]	valid_0's auc: 0.781509
For  max_depth 10 and some other parameters, cross validation AUC score 0.7815088955286157
The optimal  max_depth:  10
```

In [28]:
```
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)
```

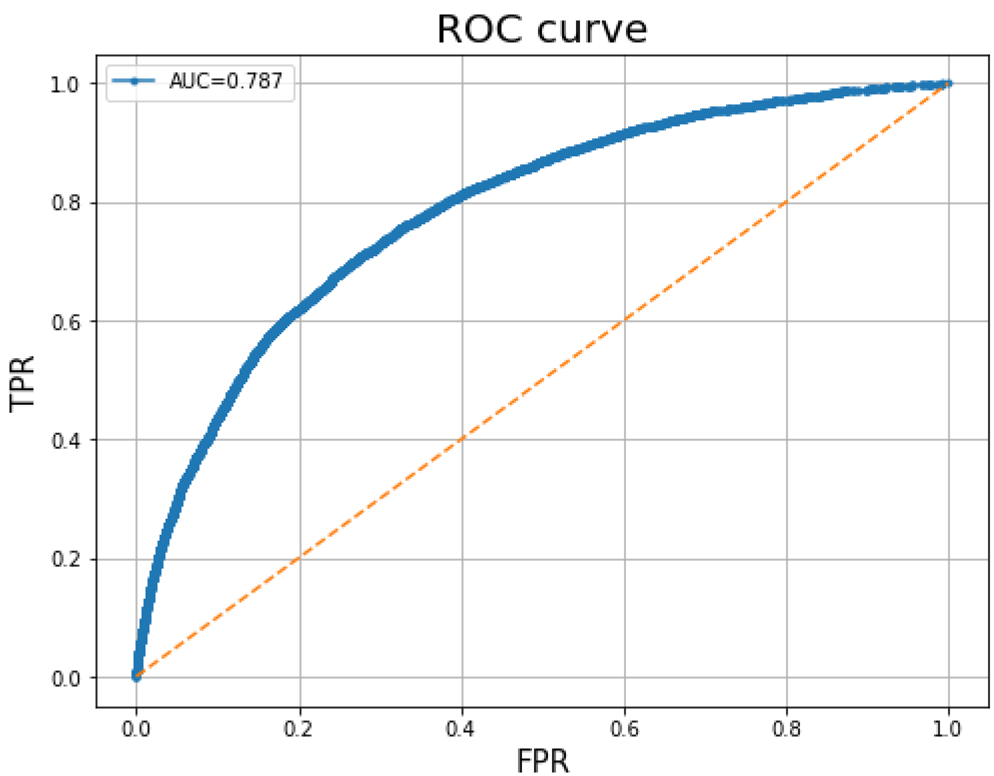
```
for i in range(len(y_pred_prob)):
    if y_pred_prob[i]<=0.5:
        y_pred[i]=0
    else:
        y_pred[i]=1

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)
```

```
Training until validation scores don't improve for 100 rounds.
[10]	valid_0's auc: 0.737512
[20]	valid_0's auc: 0.742569
[30]	valid_0's auc: 0.748226
[40]	valid_0's auc: 0.752636
[50]	valid_0's auc: 0.757438
[60]	valid_0's auc: 0.760858
[70]	valid_0's auc: 0.764328
[80]	valid_0's auc: 0.766632
[90]	valid_0's auc: 0.768313
[100]	valid_0's auc: 0.76989
[110]	valid_0's auc: 0.771375
[120]	valid_0's auc: 0.772301
[130]	valid_0's auc: 0.773232
[140]	valid_0's auc: 0.774115
[150]	valid_0's auc: 0.774698
[160]	valid_0's auc: 0.775521
[170]	valid_0's auc: 0.776208
[180]	valid_0's auc: 0.777143
[190]	valid_0's auc: 0.777731
[200]	valid_0's auc: 0.777994
[210]	valid_0's auc: 0.778372
[220]	valid_0's auc: 0.778724
[230]	valid_0's auc: 0.778998
[240]	valid_0's auc: 0.779136
[250]	valid_0's auc: 0.779341
[260]	valid_0's auc: 0.779481
[270]	valid_0's auc: 0.779541
[280]	valid_0's auc: 0.779698
[290]	valid_0's auc: 0.779839
[300]	valid_0's auc: 0.780145
[310]	valid_0's auc: 0.780344
[320]	valid_0's auc: 0.780523
[330]	valid_0's auc: 0.780515
[340]	valid_0's auc: 0.780579
[350]	valid_0's auc: 0.780562
[360]	valid_0's auc: 0.780642
[370]	valid_0's auc: 0.780547
[380]	valid_0's auc: 0.780662
[390]	valid_0's auc: 0.780882
[400]	valid_0's auc: 0.780954
[410]	valid_0's auc: 0.781015
[420]	valid_0's auc: 0.781127
```

```
[430]	valid_0's auc: 0.781108
[440]	valid_0's auc: 0.78118
[450]	valid_0's auc: 0.781291
[460]	valid_0's auc: 0.781473
[470]	valid_0's auc: 0.781412
[480]	valid_0's auc: 0.781371
[490]	valid_0's auc: 0.781442
[500]	valid_0's auc: 0.781396
[510]	valid_0's auc: 0.781355
[520]	valid_0's auc: 0.781295
[530]	valid_0's auc: 0.781213
[540]	valid_0's auc: 0.781226
[550]	valid_0's auc: 0.78122
[560]	valid_0's auc: 0.781258
[570]	valid_0's auc: 0.781161
[580]	valid_0's auc: 0.781066
Early stopping, best iteration is:
[483]	valid_0's auc: 0.781509
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% :
```



In [29]:
```
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()
```



5.Conclusion

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

- Of all the models that I have trained, LightGBDT gives the best performance and it is also faster to train when compared to Xgboost.

In [30]:

```
# Saving the final model LightGBM as pickle file for the future use in productionizing the model
with open('final_model.pkl','wb') as fp:
    pickle.dump(lgbm, fp)
```