Importing necessary modules

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
```

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

In [2]:
```
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[2]:

	SK_ID_CURR	TARGET	NAME_CONTRACT_TYPE	CODE_GENDER	FLAG_OWN_CAR	FLAG_OWN_REALTY	CNT_CHILDREN	AMT_INCOME_TOTAL	AMT_CREDIT	AMT_ANNUITY	...	FLAG_DOCUMENT_18	FLAG_DOCUMENT
0	100002	1	Cash loans	M	N	Y	0	202500.0	406597.5	24700.5	...	0	
1	100003	0	Cash loans	F	N	N	0	270000.0	1293502.5	35698.5	...	0	
2	100004	0	Revolving loans	M	Y	Y	0	67500.0	135000.0	6750.0	...	0	
3	100006	0	Cash loans	F	N	Y	0	135000.0	312682.5	29686.5	...	0	
4	100007	0	Cash loans	M	N	Y	0	121500.0	513000.0	21865.5	...	0	

5 rows × 122 columns

In [3]:
```
#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.
```

Checking for missing values in each column.

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

	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

In [5]:
```
# A lot of missing values are present , will handle them later on..
```

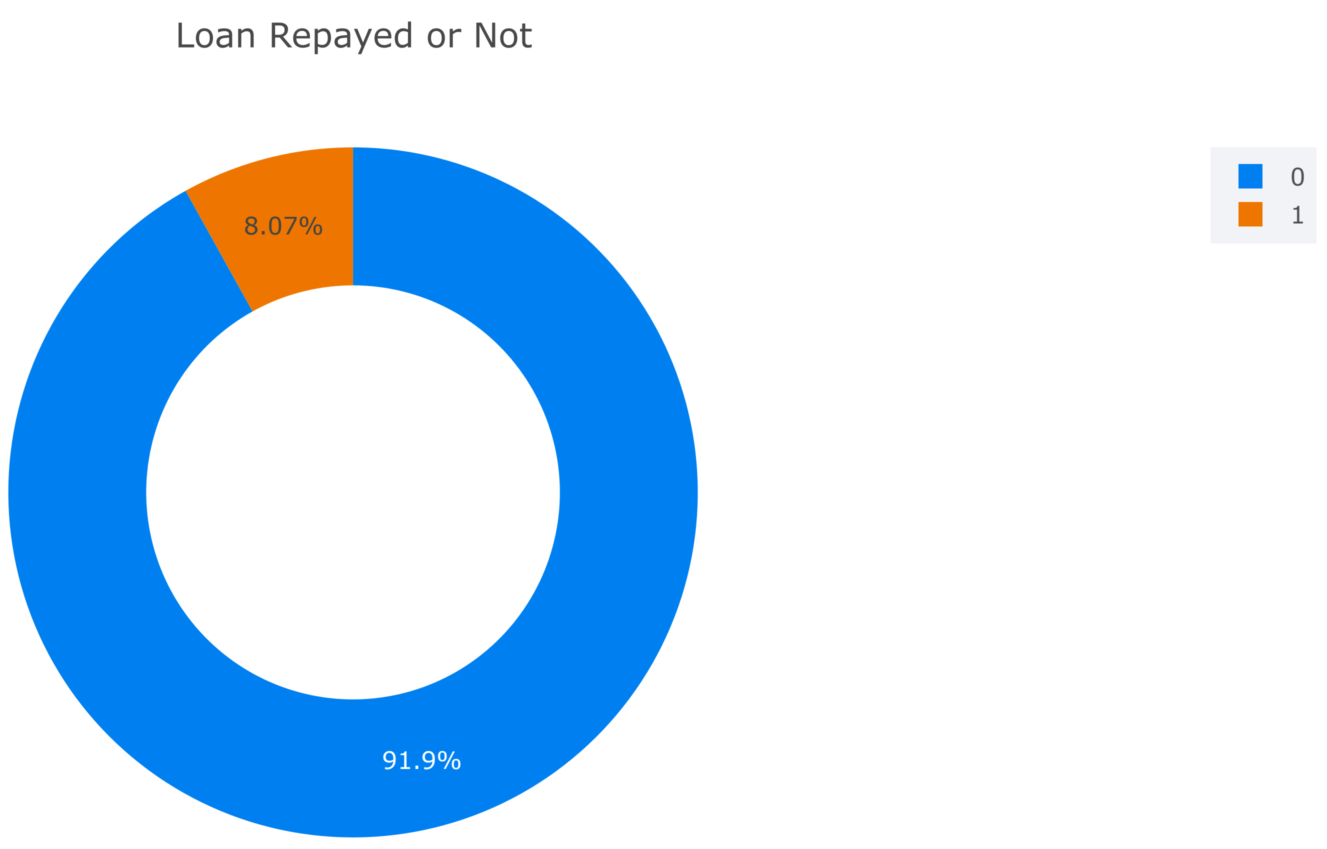
Checking for duplicate(redundant) data

In [6]:
```
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
```

Checking distribution of data points among output class

In [7]:
```
cf.set_config_file(theme='polar')
contract_val = application['TARGET'].value_counts()
contract_df = pd.DataFrame({'labels': contract_val.index,
                   'values': contract_val.values
                  })
contract_df.iplot(kind='pie',labels='labels',values='values', title='Loan Repayed or Not', hole = 0.6)
```

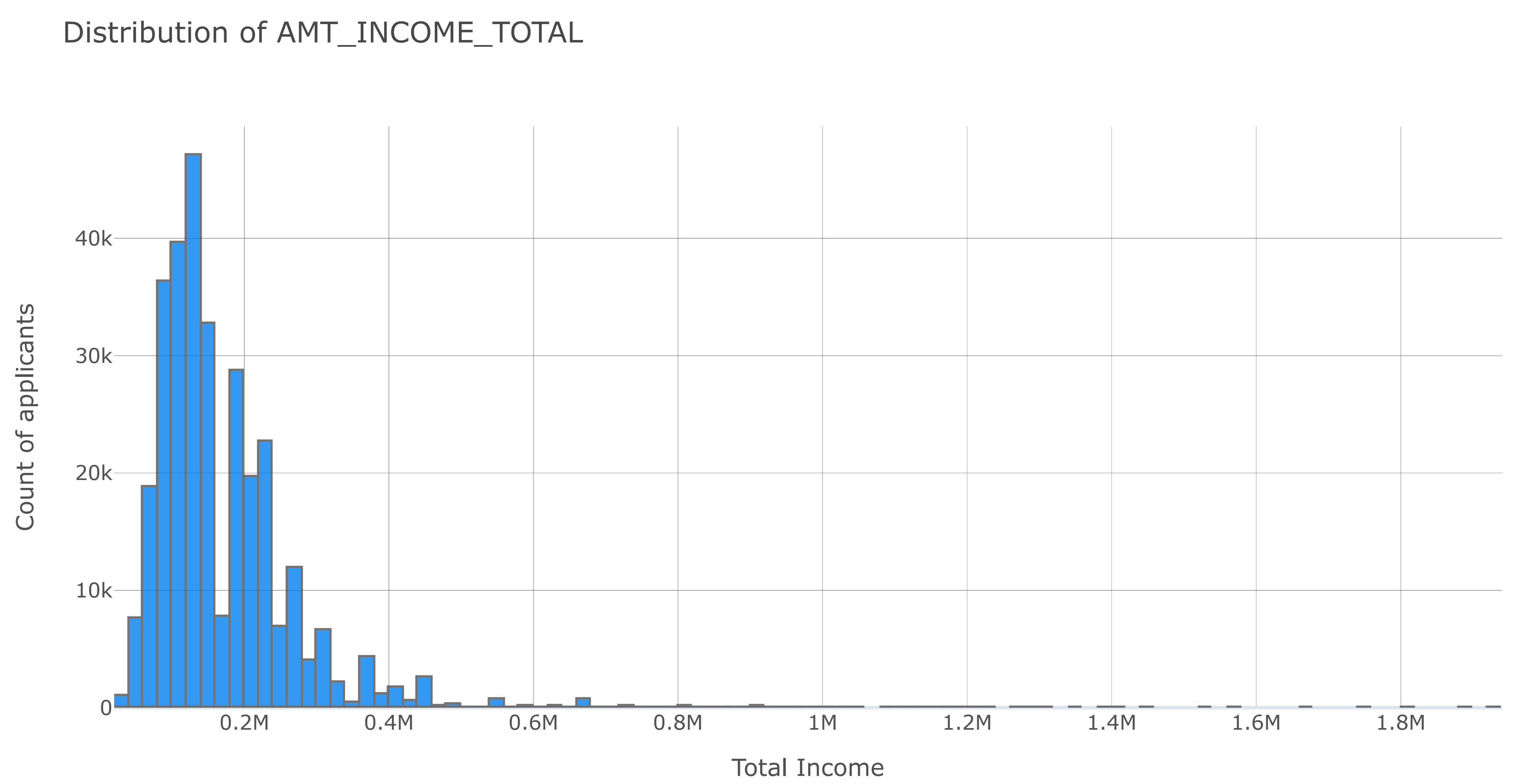


Export to plot.ly »

In [8]:
```
#The data is imbalanced (91.9%(Loan repayed-0) and 8.07%(Loan not repayed-1)).
```

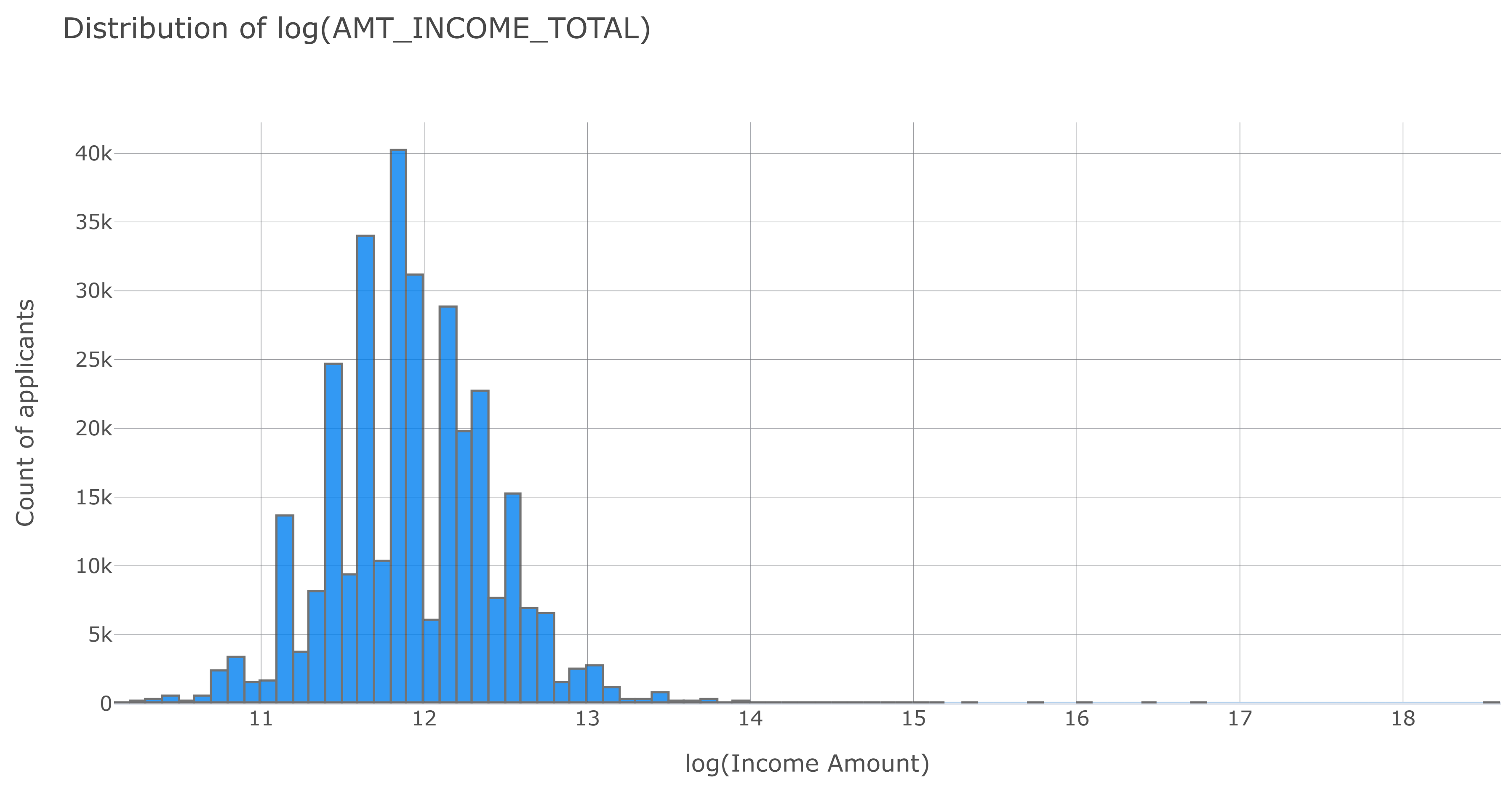
Distribution of AMT_INCOME_TOTAL.

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



Export to plot.ly »

In [10]:
```
np.log(application['AMT_INCOME_TOTAL']).iplot(kind='histogram', bins=100,
        xTitle = 'log(Income Amount)',yTitle ='Count of applicants',
        title='Distribution of log(AMT_INCOME_TOTAL)')
```



Export to plot.ly »

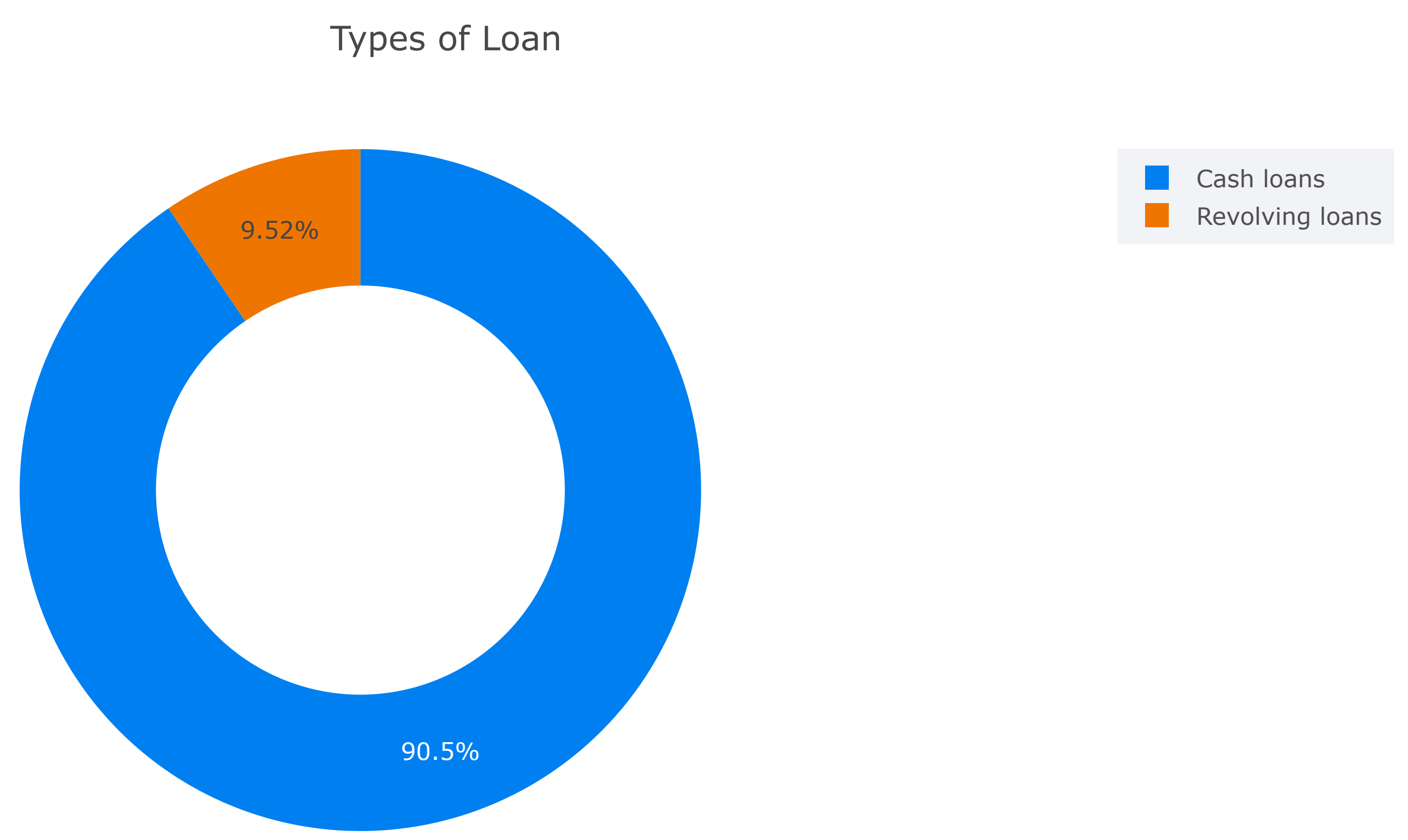
In [11]:
```
(application[application['AMT_INCOME_TOTAL'] > 1000000]['TARGET'].value_counts())/len(application[application['AMT_INCOME_TOTAL'] > 1000000])*100
```

Out[11]:
```
0    94.8
1     5.2
Name: TARGET, dtype: float64
```

In [12]:
```
#Observations:
#1. The distribution is right skewed and there are extreme values, we applied log distribution.
#2. People with high income(>1000000) are likely to repay the loan.
```

Distributing Types of loans available

In [13]:
```
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)
```

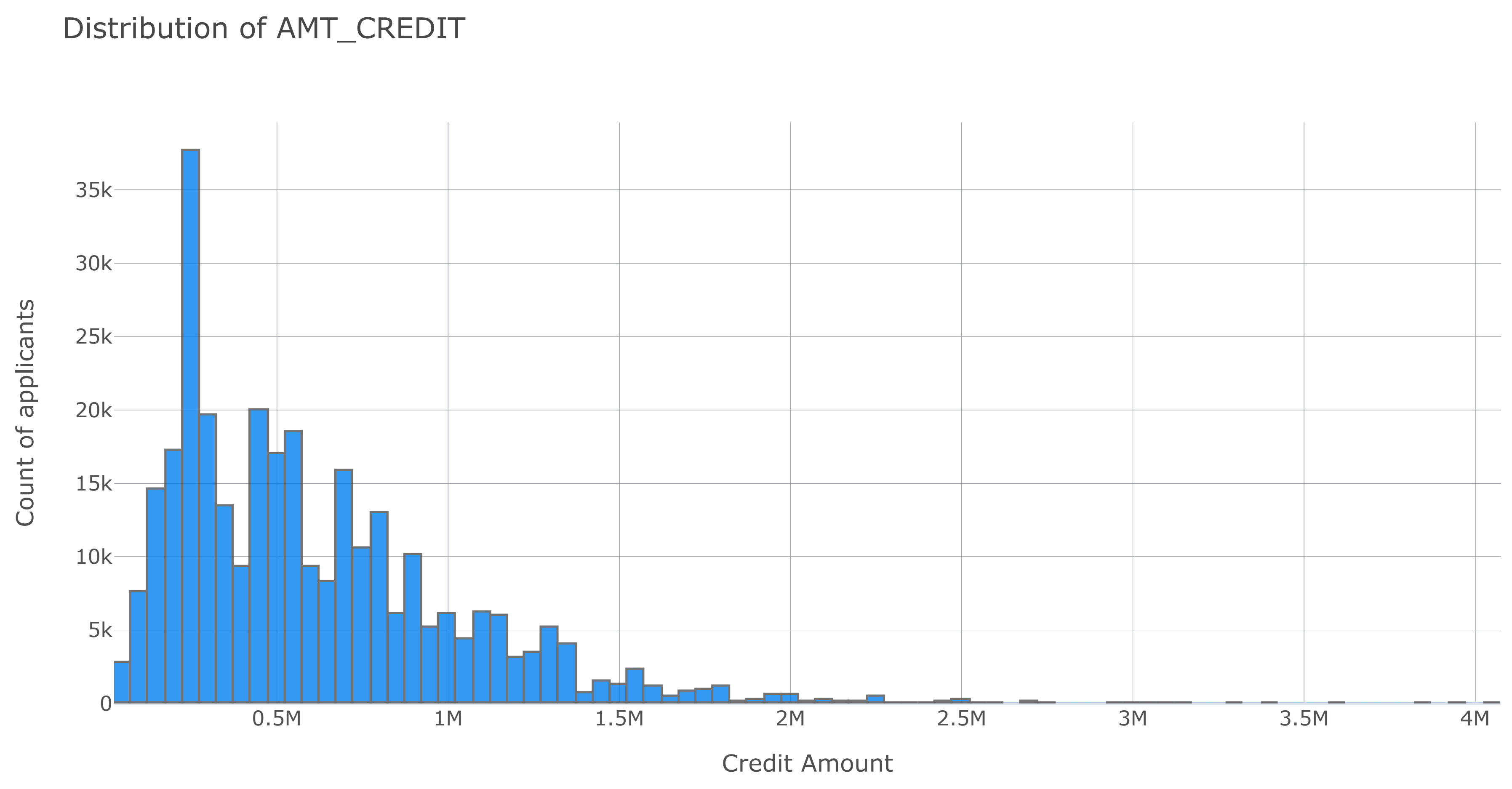


Export to plot.ly »

In [14]:
```
#Observations : Majority of the people prefer taking cash loans compared to revolving loans
```

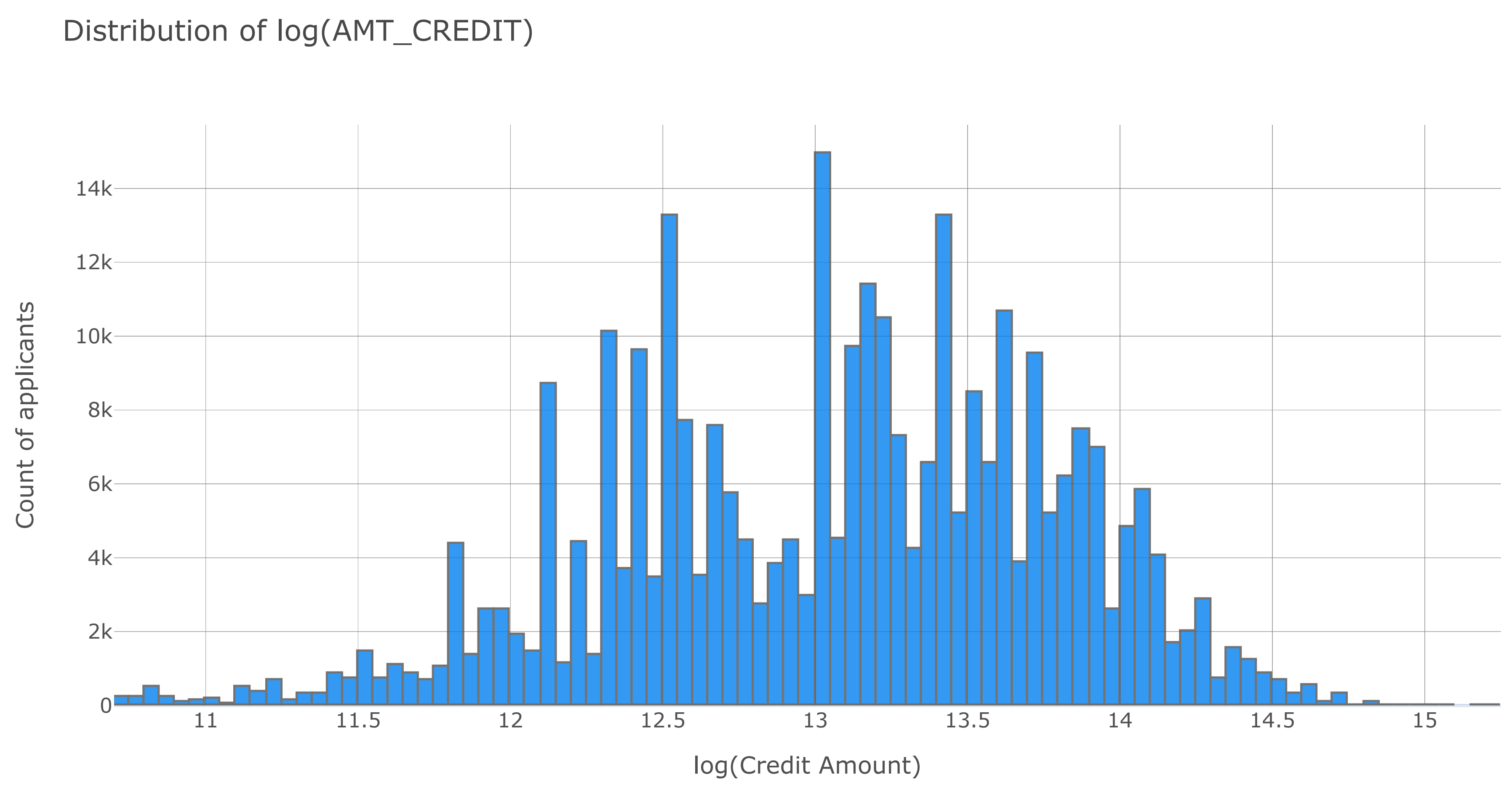
Distribution of AMT_CREDIT

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



Export to plot.ly »

In [16]:
```
np.log(application['AMT_CREDIT']).iplot(kind='histogram', bins=100,
        xTitle = 'log(Credit Amount)',yTitle ='Count of applicants',
        title='Distribution of log(AMT_CREDIT)')
```

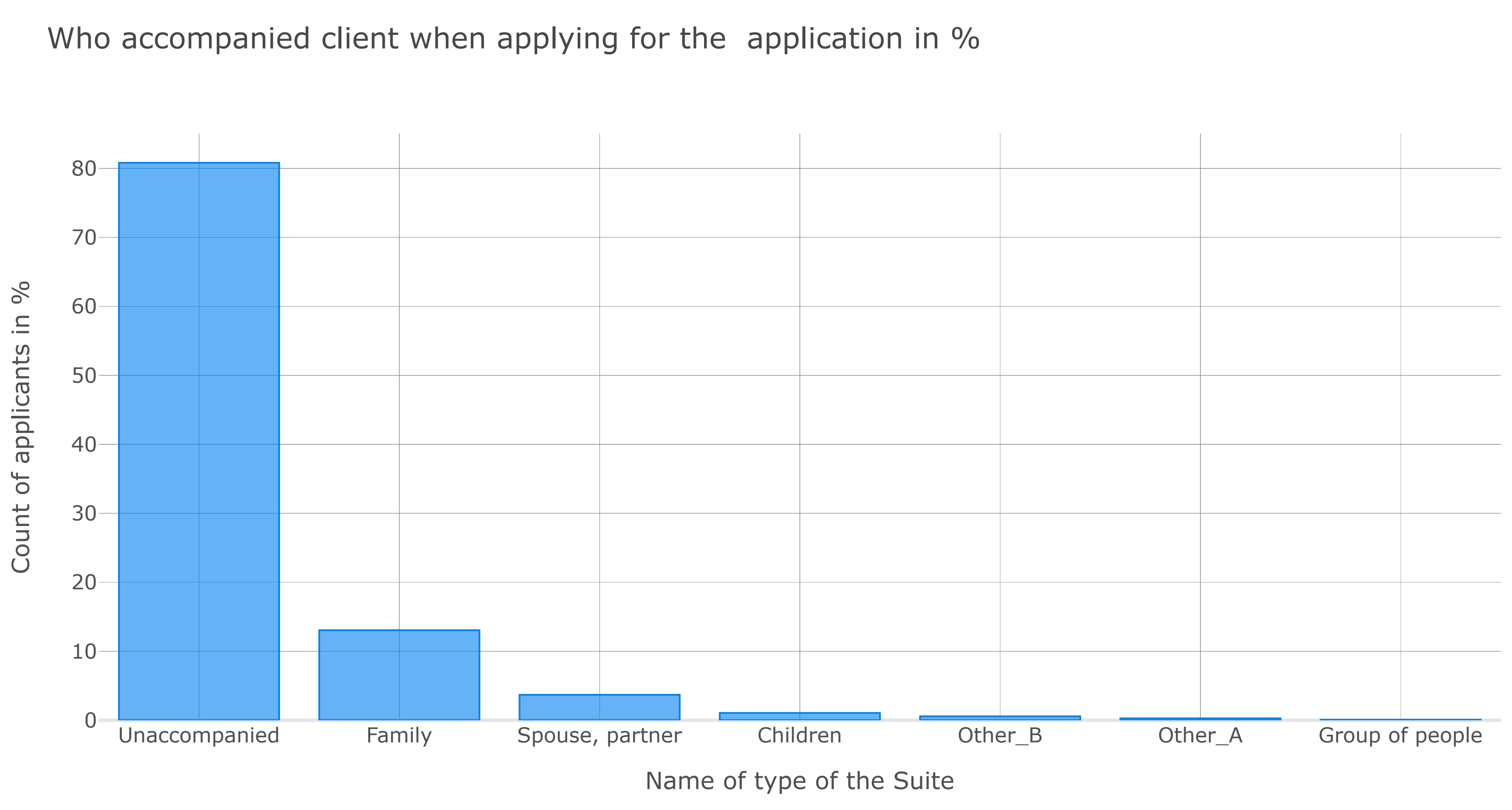


Export to plot.ly »

In [17]:
```
#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.
```

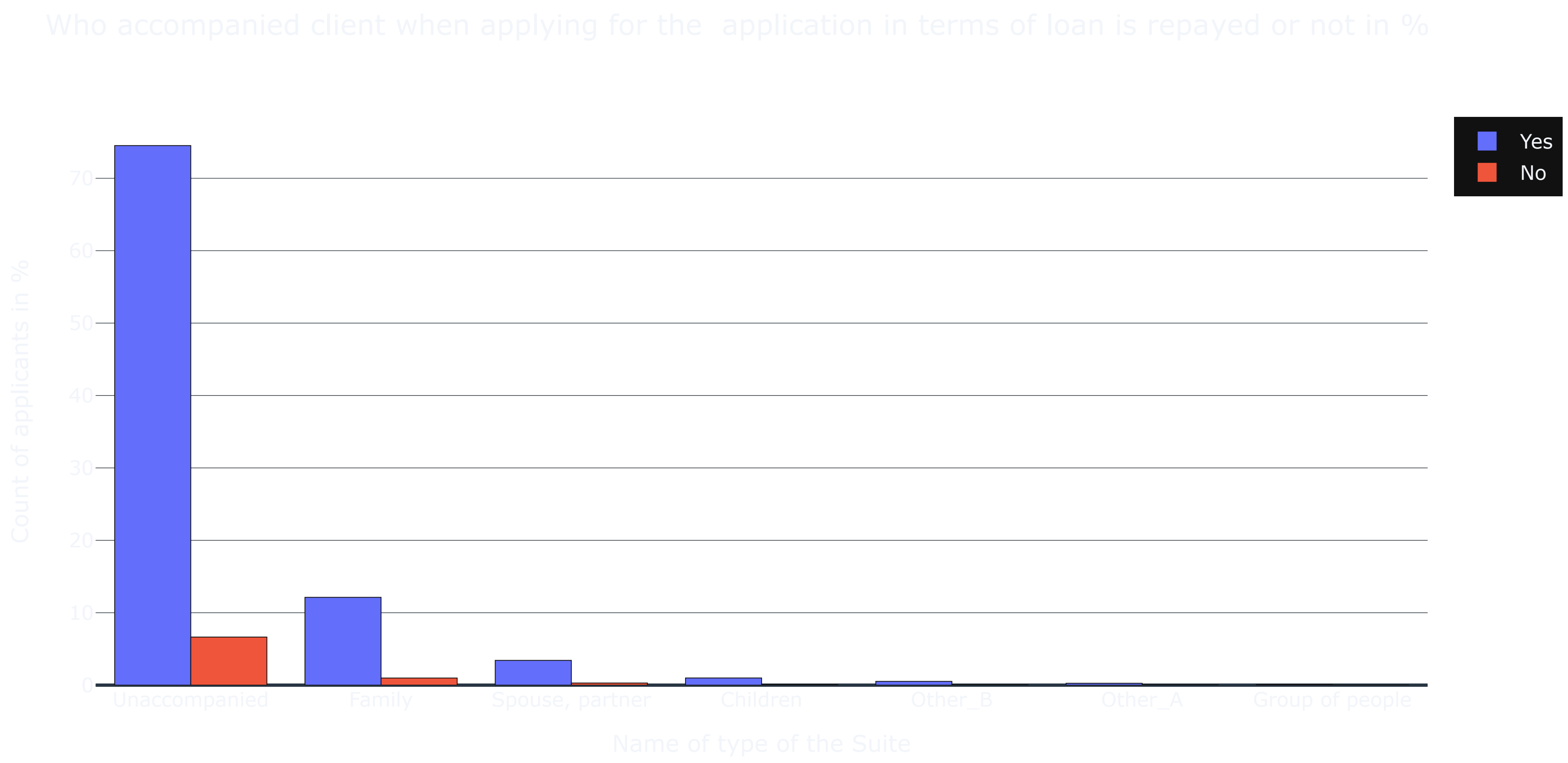
Distribution of Name of type of the Suite in terms of loan is repayed or not.

In [18]:
```
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 % ')
```



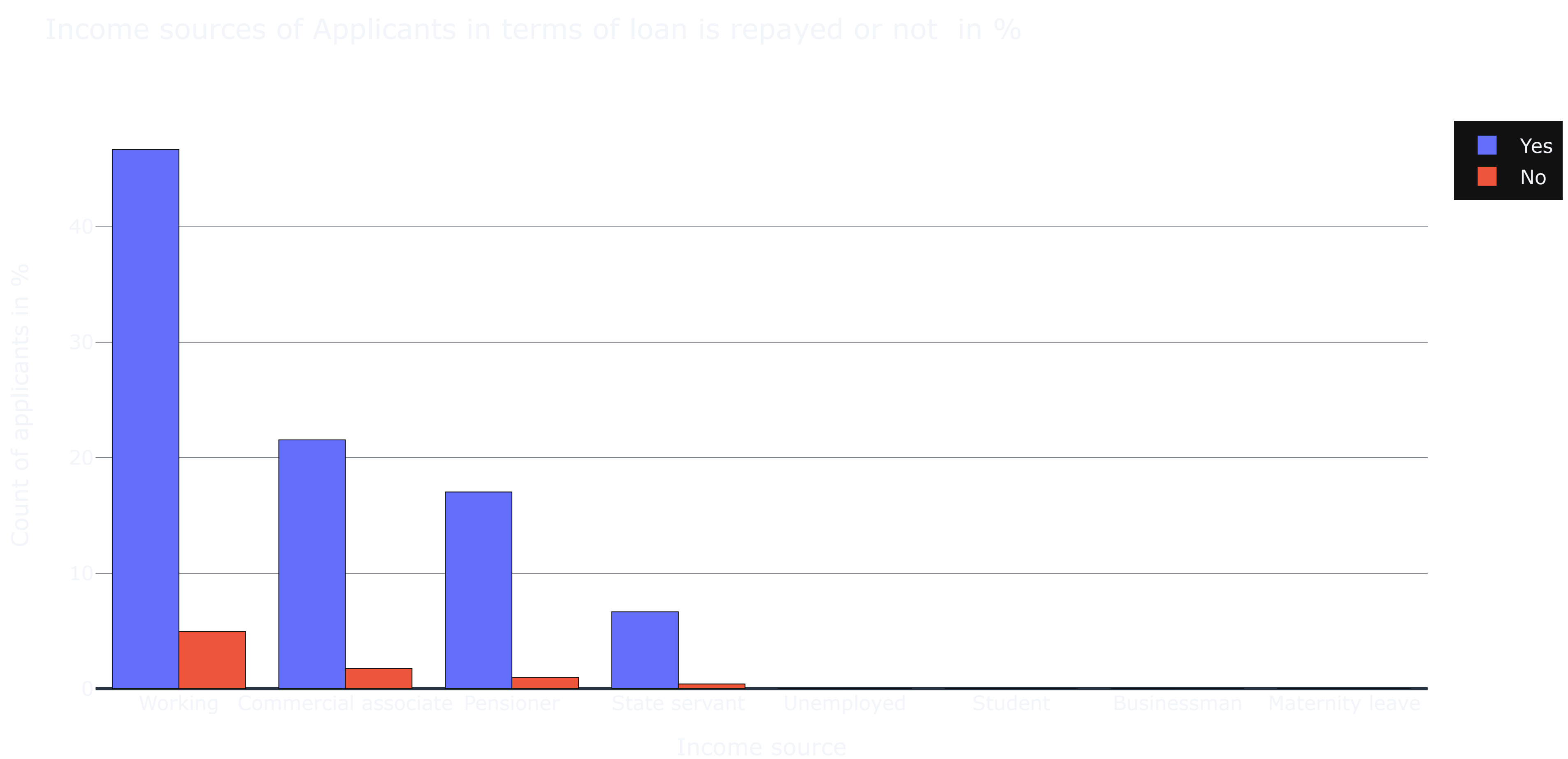
Export to plot.ly »

In [19]:
```
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] == 0))
    suite_val_y0.append(np.sum(application['TARGET'][application['NAME_TYPE_SUITE']==val] == 1))
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)
```



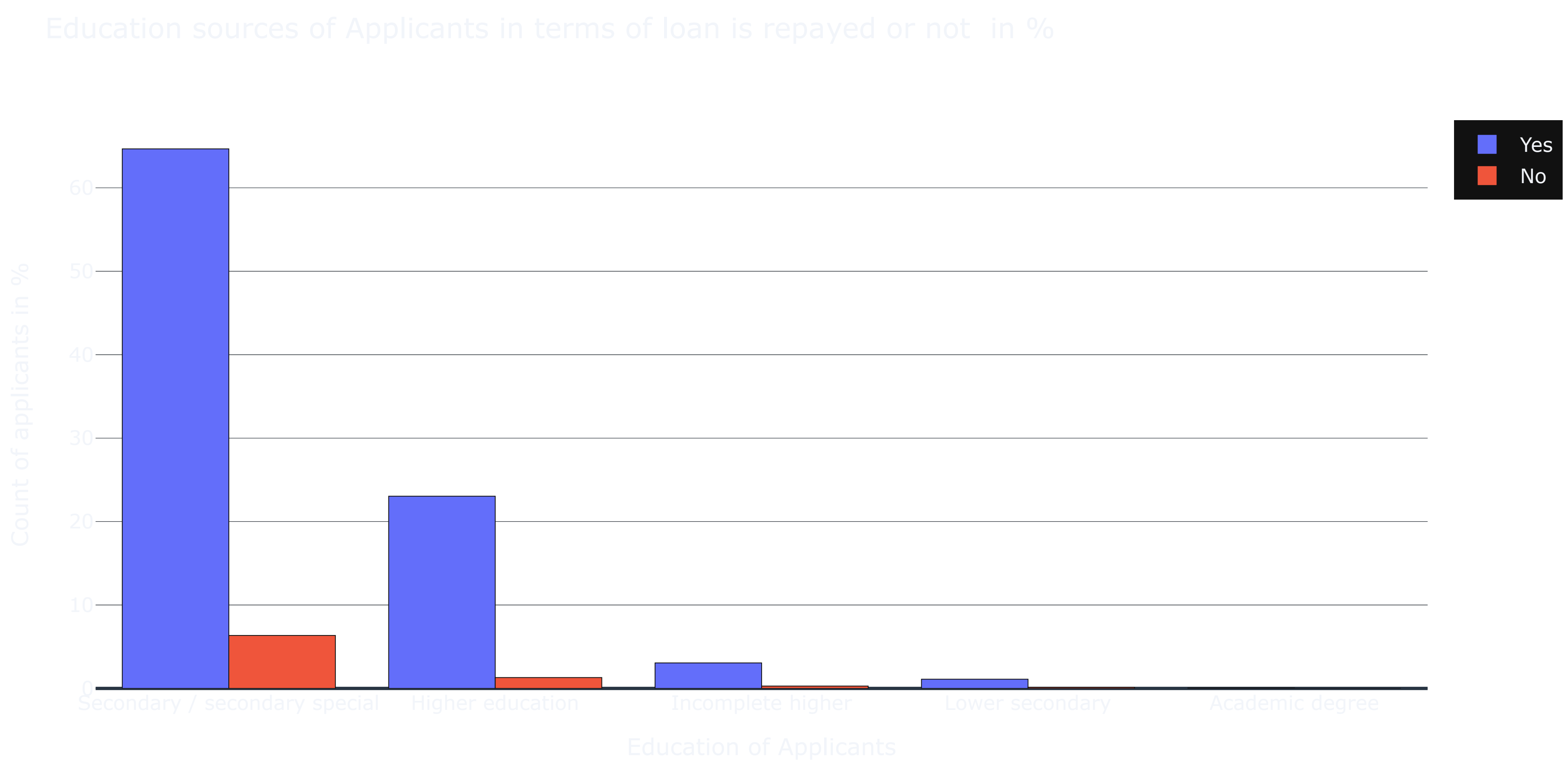
Distribution of Income sources of Applicants in terms of loan is repayed or not.

In [20]:
```
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] == 0))
    income_val_y0.append(np.sum(application['TARGET'][application['NAME_INCOME_TYPE']==val] == 1))
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)
```



In [21]:
```
#Observations:
#1. All the Students and Businessman are repaying loan.(Open the chart and hover over the plot to observe).
```

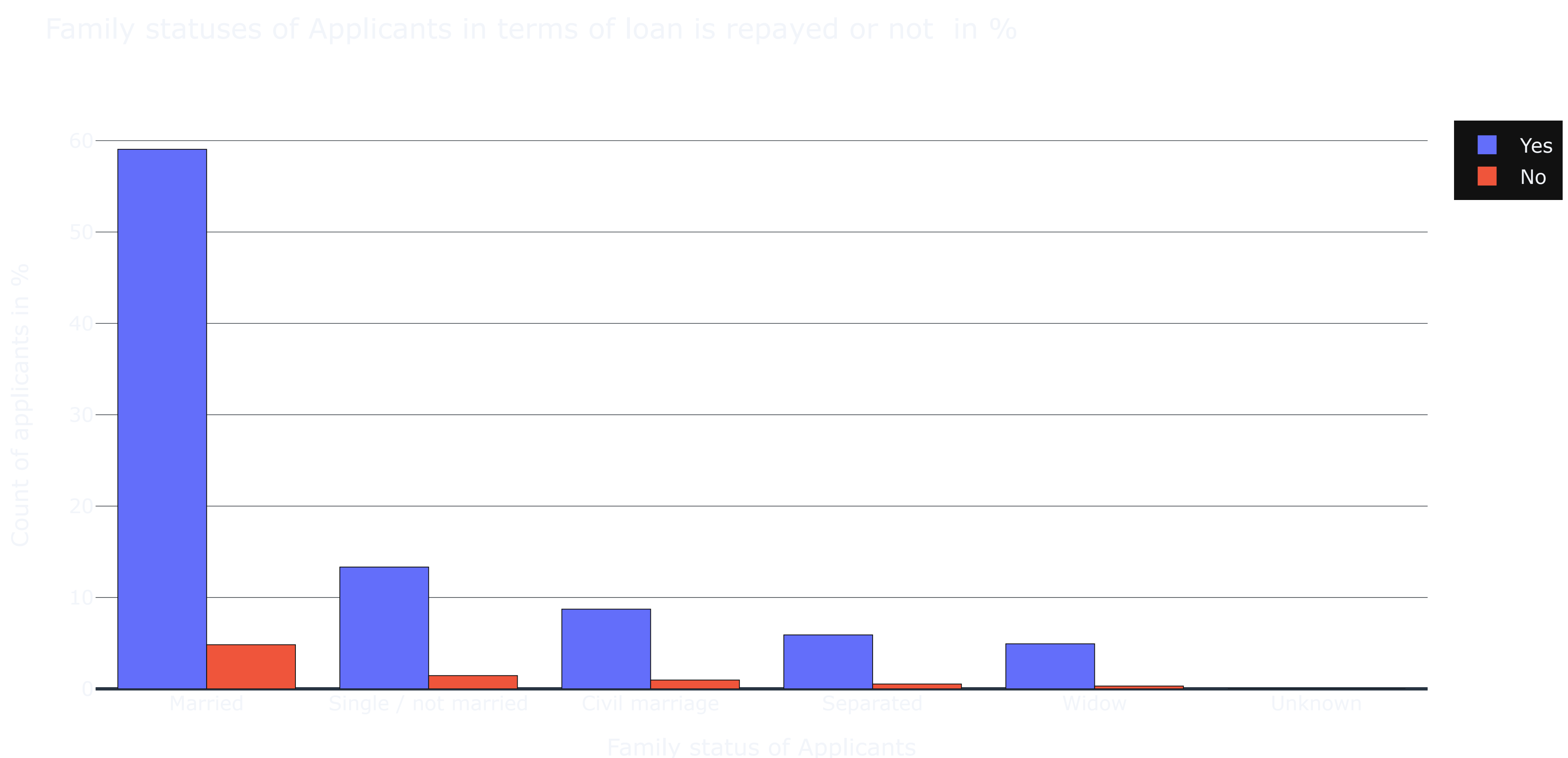
In [22]:
```
education_val = application['NAME_EDUCATION_TYPE'].value_counts()
education_val_y0 = []
education_val_y1 = []
for val in education_val.index:
    education_val_y1.append(np.sum(application['TARGET'][application['NAME_EDUCATION_TYPE']==val] == 0))
    education_val_y0.append(np.sum(application['TARGET'][application['NAME_EDUCATION_TYPE']==val] == 1))
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)
```



In [23]:
```
# 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.

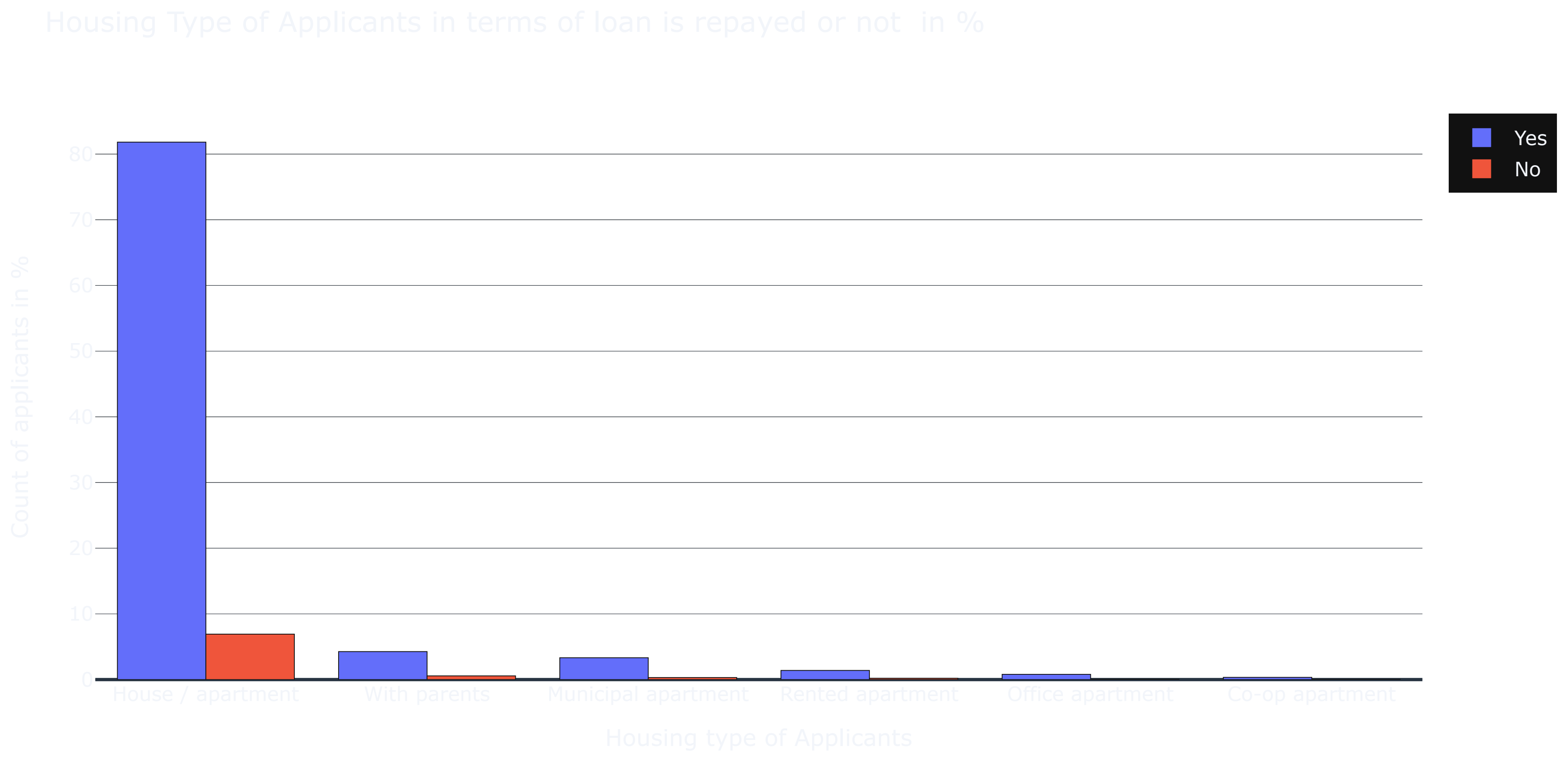
In [24]:
```
family_val = application['NAME_FAMILY_STATUS'].value_counts()
family_val_y0 = []
family_val_y1 = []
for val in family_val.index:
    family_val_y1.append(np.sum(application['TARGET'][application['NAME_FAMILY_STATUS']==val] == 0))
    family_val_y0.append(np.sum(application['TARGET'][application['NAME_FAMILY_STATUS']==val] == 1))
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 statuses of Applicants in terms of loan is repayed or not  in %",
    xaxis=dict(
        title='Family status of Applicants',
       ),
    yaxis=dict(
        title='Count of applicants in %',
        )
)
fig = go.Figure(data = data, layout=layout)
fig.layout.template = 'plotly_dark'
py.iplot(fig)
```



In [25]:
```
#Observations:
#1. Widows are more likely to repay the loan when compared to appliants with the other family statuses.
```

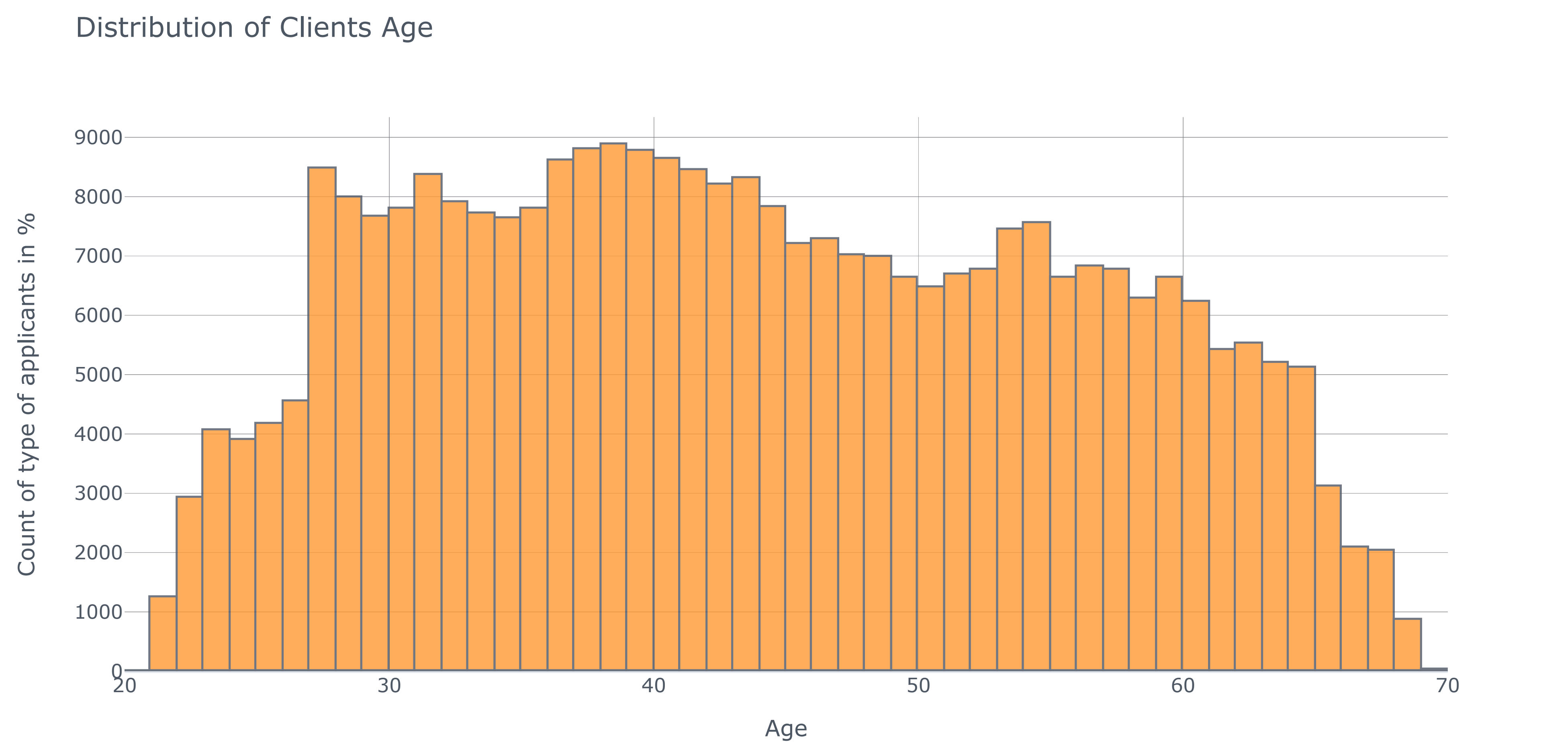
Distribution of Housing type of Applicants in terms of loan is repayed or not.

In [26]:
```
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] == 0))
    housing_val_y0.append(np.sum(application['TARGET'][application['NAME_HOUSING_TYPE']==val] == 1))
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 = "Housing Type of Applicants in terms of loan is repayed or not  in %",
    xaxis=dict(
        title='Housing type of Applicants',
       ),
    yaxis=dict(
        title='Count of applicants in %',
        )
)
fig = go.Figure(data = data, layout=layout)
fig.layout.template = 'plotly_dark'
py.iplot(fig)
```



In [27]:
```
# Distribution of Clients Age
```

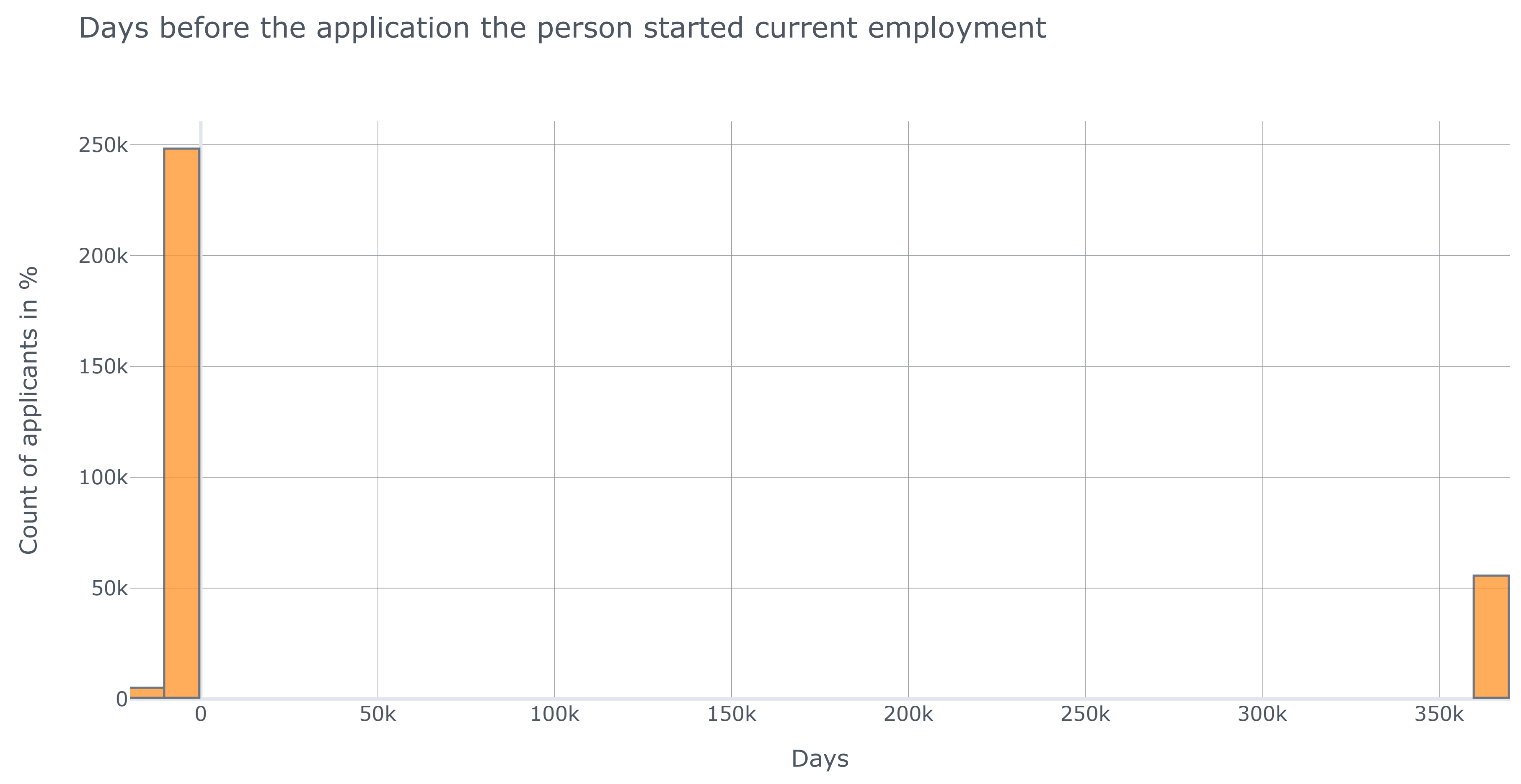
In [28]:
```
cf.set_config_file(theme='pearl')
(application['DAYS_BIRTH']/(-365)).iplot(kind='histogram',
             xTitle = 'Age', bins=50,
             yTitle='Count of type of applicants in %',
             title='Distribution of Clients Age')
```



Export to plot.ly »

In [29]:
```
# Distribution of years before the application the person started current employment.
```

In [30]:
```
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')
```



Export to plot.ly »

In [31]:
```
#Observations:
#The data looks strange
#(we have 1000.66 years(365243 days) of employment which is impossible) looks like there is data entry error.
```

In [32]:
```
error = application[application['DAYS_EMPLOYED'] == 365243]
print('The no of errors are :', len(error))
(error['TARGET'].value_counts()/len(error))*100
```

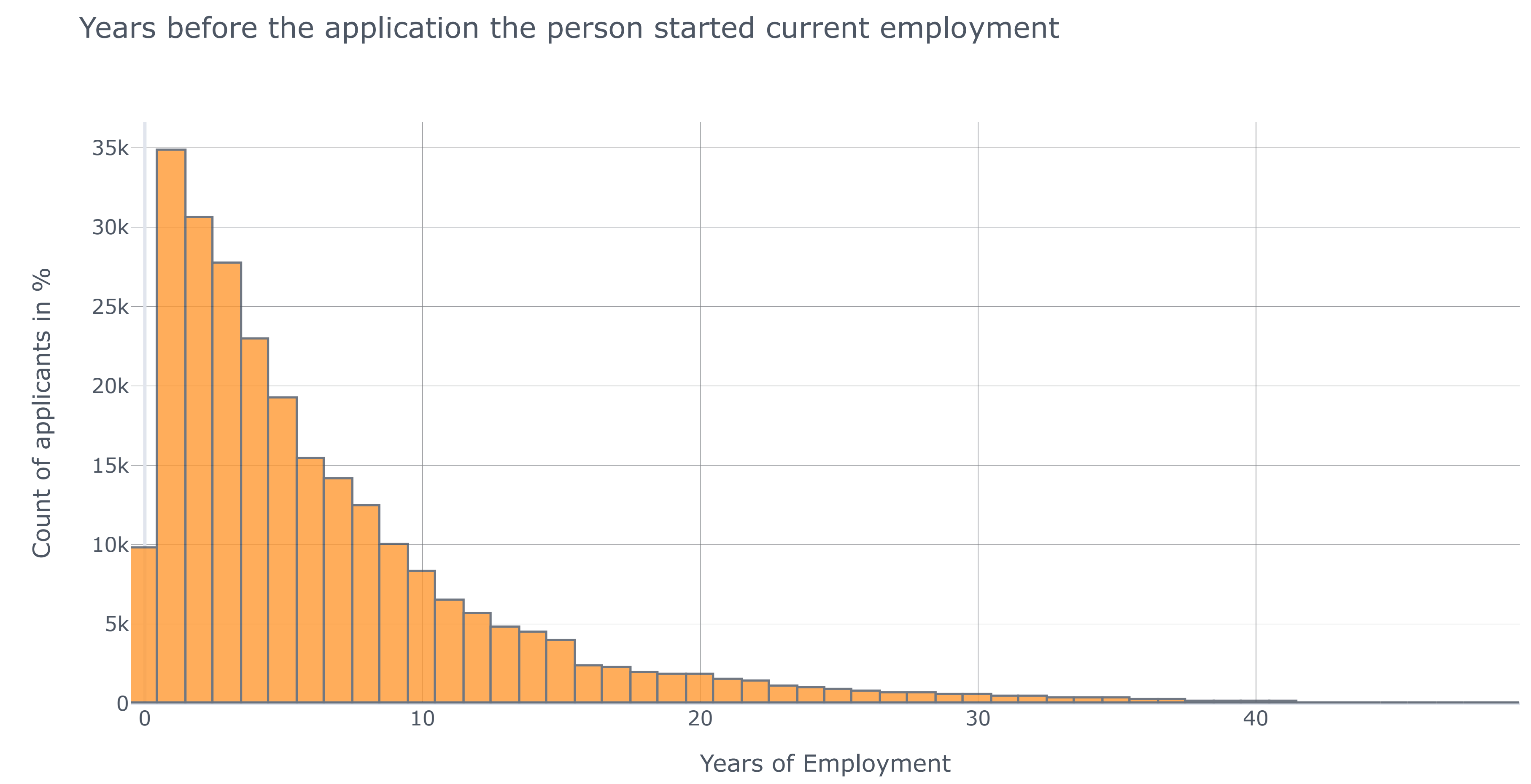
```
The no of errors are : 55374
```

Out[32]:
```
0    94.600354
1     5.399646
Name: TARGET, dtype: float64
```

In [33]:
```
# The error are default to 5.4% , so we need to handle this
```

In [34]:
```
# 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 [35]:
```
#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')
```



Export to plot.ly »

In [36]:
```
application[application['DAYS_EMPLOYED']>(-365*2)]['TARGET'].value_counts()/sum(application['DAYS_EMPLOYED']>(-365*2))
```

Out[36]:
```
0    0.887924
1    0.112076
Name: TARGET, dtype: float64
```

In [37]:
```
#Observations:
#The applicants with lesser years of employment are less likely to repay the loan(<2 years is least likely)
```

Data Preparation:

#Feature Engineering of Application data:

In [38]:
```
# 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:

In [39]:
```
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[39]:

	SK_ID_CURR	SK_ID_BUREAU	CREDIT_ACTIVE	CREDIT_CURRENCY	DAYS_CREDIT	CREDIT_DAY_OVERDUE	DAYS_CREDIT_ENDDATE	DAYS_ENDDATE_FACT	AMT_CREDIT_MAX_OVERDUE	CNT_CREDIT_PROLONG	AMT_
0	215354	5714462	Closed	currency 1	-497	0	-153.0	-153.0	NaN	0	
1	215354	5714463	Active	currency 1	-208	0	1075.0	NaN	NaN	0	
2	215354	5714464	Active	currency 1	-203	0	528.0	NaN	NaN	0	
3	215354	5714465	Active	currency 1	-203	0	NaN	NaN	NaN	0	
4	215354	5714466	Active	currency 1	-629	0	1197.0	NaN	77674.5	0	

Joining Bureau data to Application data:

In [40]:
```
# 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 application and bureau data combined: (307511, 163)
```

Feature Engineering of Bureau Data:

In [41]:
```
# 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)
```

In [42]:
```
# 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 [43]:
```
# 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)
```

In [44]:
```
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'})
```

In [45]:
```
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_DEB
```

In [46]:
```
grp1['DEBT_CREDIT_RATIO'] = grp2['TOTAL_CREDIT_SUM_DEBT']/grp1['TOTAL_CREDIT_SUM']
```

In [47]:
```
del grp1['TOTAL_CREDIT_SUM']
```

In [48]:
```
application_bureau = application_bureau.merge(grp1, on='SK_ID_CURR', how='left')
```

In [49]:
```
application_bureau['DEBT_CREDIT_RATIO'] = application_bureau['DEBT_CREDIT_RATIO'].fillna(0)
```

In [50]:
```
application_bureau['DEBT_CREDIT_RATIO'] = pd.to_numeric(application_bureau['DEBT_CREDIT_RATIO'], downcast='float')
```

In [51]:
```
# 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)
```

In [52]:
```
grp1 = bureau[['SK_ID_CURR','AMT_CREDIT_SUM_OVERDUE']].groupby(by=['SK_ID_CURR'])['AMT_CREDIT_SUM_OVERDUE'].sum().reset_index().rename(columns={'AMT_CREDIT_SUM_OVERDUE': 'TOTAL_CUST
```

In [53]:
```
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_CUSTOMER_DEBT'
```

In [54]:
```
grp1['OVERDUE_DEBT_RATIO'] = grp1['TOTAL_CUSTOMER_OVERDUE']/grp2['TOTAL_CUSTOMER_DEBT']
```

In [55]:
```
del grp1['TOTAL_CUSTOMER_OVERDUE']
```

In [56]:
```
application_bureau = application_bureau.merge(grp1, on='SK_ID_CURR', how='left')
application_bureau['OVERDUE_DEBT_RATIO'] = application_bureau['OVERDUE_DEBT_RATIO'].fillna(0)
```

In [57]:
```
application_bureau['OVERDUE_DEBT_RATIO'] = pd.to_numeric(application_bureau['OVERDUE_DEBT_RATIO'], downcast='float')
#dropping irrelevant/already analysed features
application_bureau = application_bureau.drop(['NAME_CONTRACT_TYPE', 'CODE_GENDER', 'FLAG_OWN_CAR', 'FLAG_OWN_REALTY',
                                              'NAME_TYPE_SUITE', 'NAME_INCOME_TYPE', 'NAME_EDUCATION_TYPE',
                                              'NAME_FAMILY_STATUS',
                                              'NAME_HOUSING_TYPE', 'OCCUPATION_TYPE', 'WEEKDAY_APPR_PROCESS_START',
                                              'ORGANIZATION_TYPE', 'FONDKAPREMONT_MODE', 'HOUSETYPE_MODE',
                                               'WALLSMATERIAL_MODE', 'EMERGENCYSTATE_MODE'] , axis = 1)
```

In [58]:
```
application_bureau = application_bureau.fillna(0)
print('The shape application and bureau data combined:',application_bureau.shape)
```

```
The shape application and bureau data combined: (307511, 151)
```

In [59]:
```
# Not considering irrelevant datasets (already analysed , they have very low implications in predicting target and are
#very large thus creating memory issues also , slowing the response.
# The datasets not included are : previous_applications , installments_payments , credit_card_balance , pos data)
# Proxy for these are : repayment behaviour , if existing cust. returns , also CC data to be included in bureau data
```

Dividing final data into train, valid and test sets

In [60]:
```
y = application_bureau.pop('TARGET').values
X_train, X_temp, y_train, y_temp = train_test_split(application_bureau.drop(['SK_ID_CURR'],axis=1), y, stratify = 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_state=42)
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, 149)
Shape of X_val: (46127, 149)
Shape of X_test: (46127, 149)
```

Selection of features and plotting feature importance

In [61]:
```
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, 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)
```

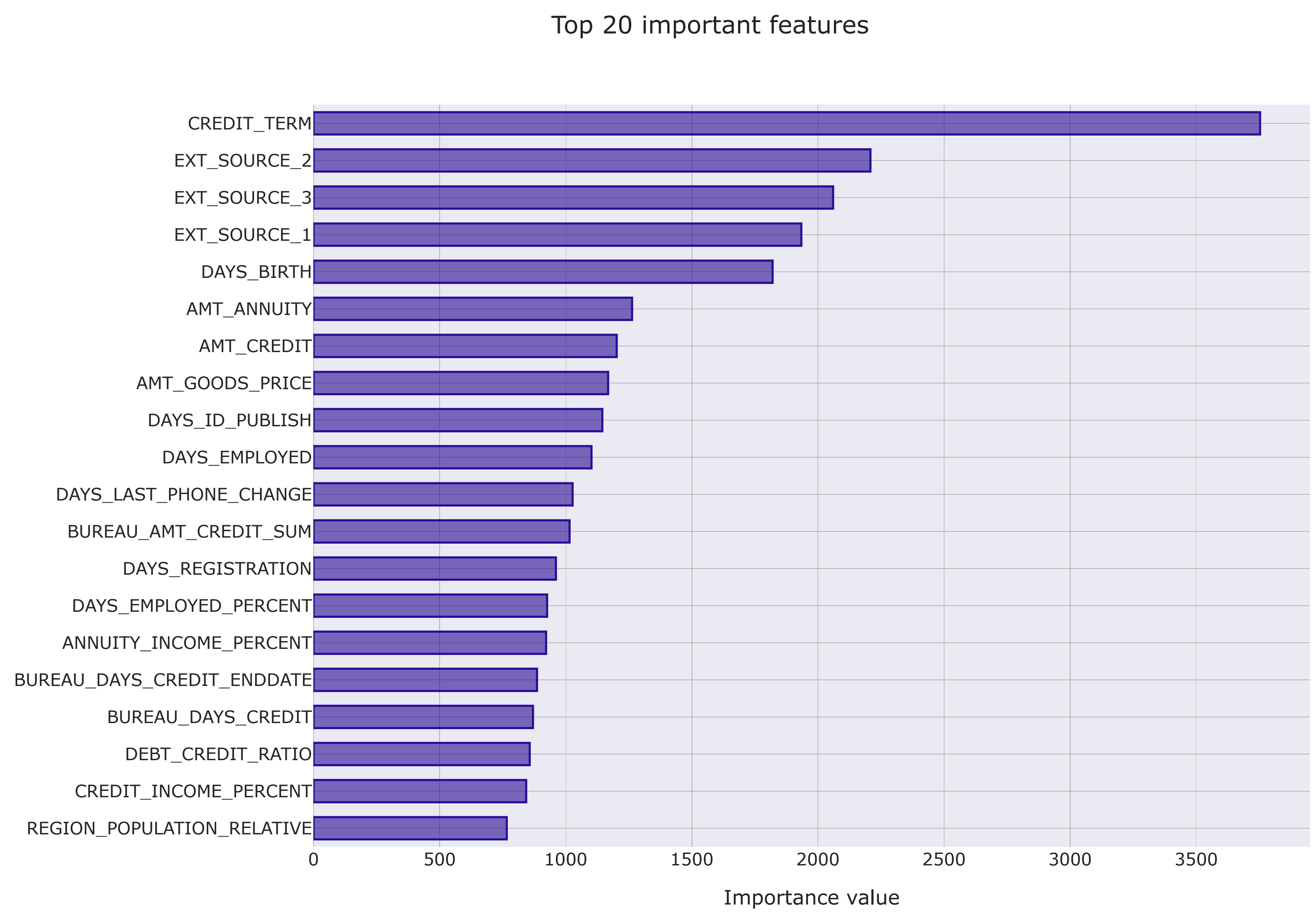
```
Training until validation scores don't improve for 100 rounds
[200]   valid_0's auc: 0.744263 valid_0's binary_logloss: 0.600395
[400]   valid_0's auc: 0.755893 valid_0's binary_logloss: 0.579162
[600]   valid_0's auc: 0.761146 valid_0's binary_logloss: 0.567717
[800]   valid_0's auc: 0.763691 valid_0's binary_logloss: 0.559939
[1000]  valid_0's auc: 0.764767 valid_0's binary_logloss: 0.553989
[1200]  valid_0's auc: 0.765469 valid_0's binary_logloss: 0.548871
[1400]  valid_0's auc: 0.765742 valid_0's binary_logloss: 0.544172
Early stopping, best iteration is:
[1458]  valid_0's auc: 0.76586  valid_0's binary_logloss: 0.542804
```

Out[61]:
```
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 [62]:
```
feature_imp = pd.DataFrame(sorted(zip(model_sk.feature_importances_, X_train.columns)), columns=['Value','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: 94
```

In [63]:
```
# 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,
    width=1000,
    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)
```



Machine Learning Models

I used Random Forest , Logistic Regression and LightGBM, out of which LightGBM performed best and is also faster compartively(only uploading code for LightGBM)

In [68]:
```
# Reusable function 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")
    plt.show()
```

In [69]:

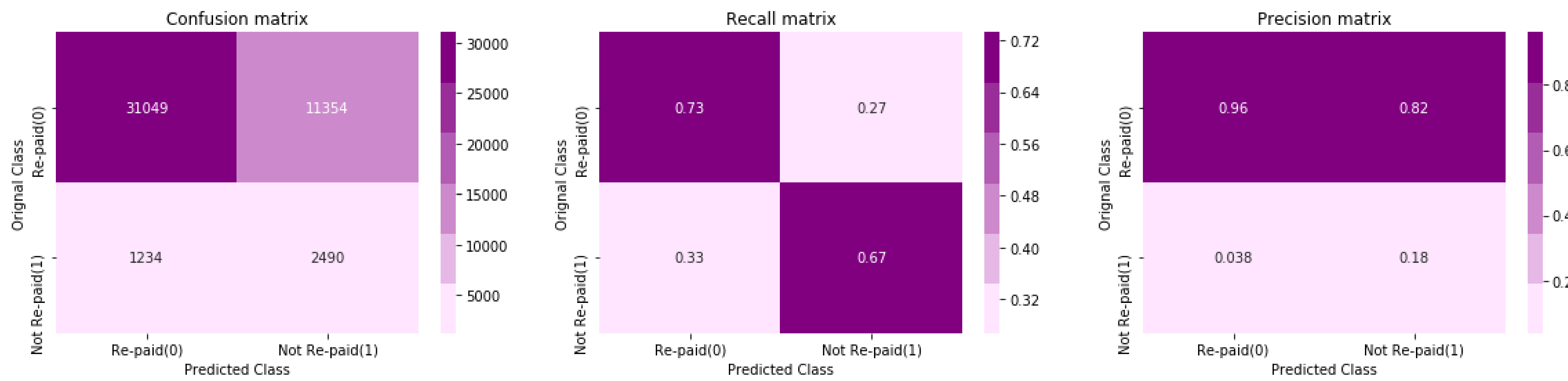
```
weight = np.ones((len(X_train),), dtype=int)
for i in range(len(X_train)):
    if y_train[i]== 0:
        weight[i]=1
    else:
        weight[i]=11

train_data=lgb.Dataset(X_train[selected_features], label = y_train, weight= weight )
valid_data=lgb.Dataset(X_val[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[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[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[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[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),), 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))
```

...

In [70]:

```
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 test AUC score is : 0.770126576628688
The percentage of misclassified points 27.29% :
```



In [71]:
```
# ROC Curve for LightGBM Model with AUC = 0.77
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()
```