1. Business Problem

1.1 Problem Description

Home Credit comes up with a Kaggle challenge to find out the loan applicants who is capable of repaying a loan, given the applicant data, all credits data from Credit Bureau, previous applications data from Home Credit and some more data.

1.2 Problem Statement

Building a model to predict how capable each applicant is of repaying a loan.

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.
3. The cost of a mis-classification is very high.
4. Interpretability is partially important.

1.4 Performance Metric:

1. Area Under Curve (AUC)
2. Confusion Matrix

Data Description and Overview:

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:

1. 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.
2. 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.
3. 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.
4. 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.

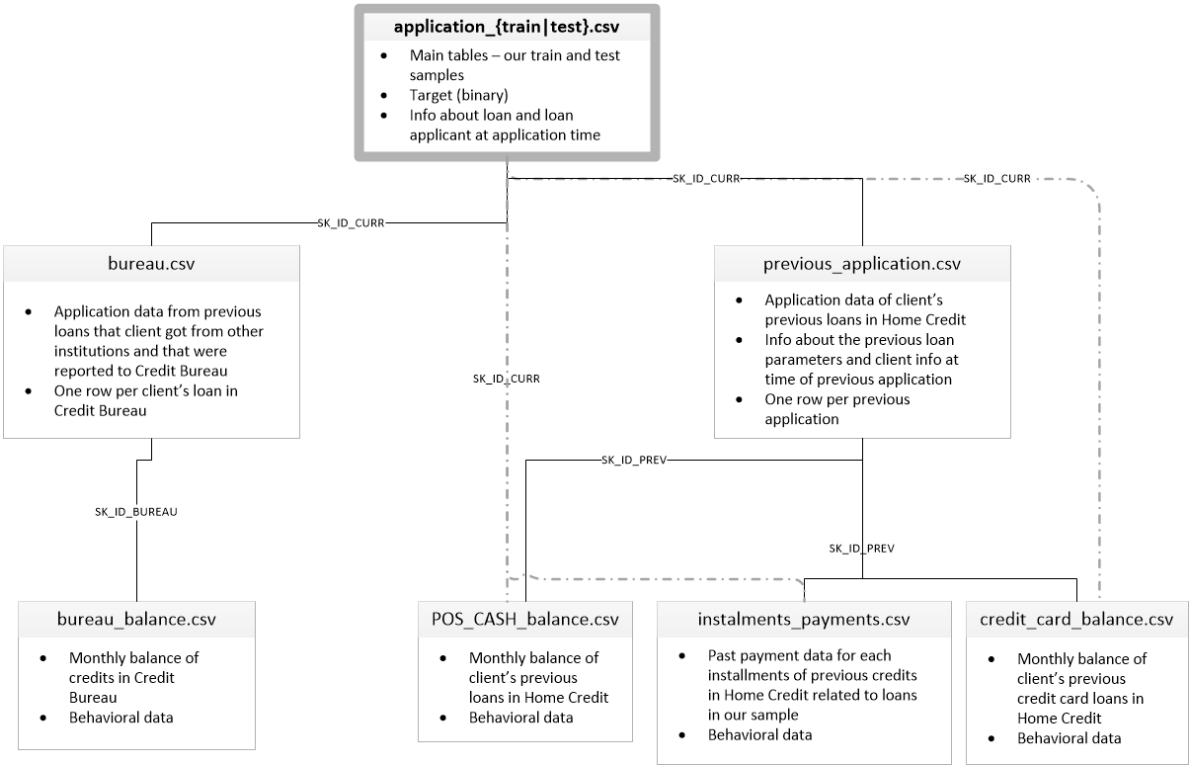
5. POS_CASH_BALANCE: It consists of monthly data about previous point of sale or cash loans clients have had with Home Credit. Each row is one month of a previous point of sale or cash loan, and a single previous loan can have many rows.
6. credit_card_balance: The monthly data about previous credit 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.
7. 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:

In [2]:

```
from IPython.display import Image
Image(filename='selfcaseStudy.png')
```

Out[2]:



In [3]:

```
import pandas as pd
import matplotlib.pyplot as plt
import re
import time
import warnings
import numpy as np
from nltk.corpus import stopwords
from sklearn.preprocessing import normalize
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.feature_extraction.text import TfidfVectorizer
warnings.filterwarnings("ignore")
import sys
import os
from tqdm import tqdm
import sqlite3
from sqlalchemy import create_engine # database connection
import csv
warnings.filterwarnings("ignore")
import datetime as dt
from nltk.corpus import stopwords
from sklearn.decomposition import TruncatedSVD
from sklearn.preprocessing import normalize
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.manifold import TSNE
import seaborn as sns
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import confusion_matrix
from sklearn.metrics.classification import accuracy_score, log_loss
from sklearn.feature_extraction.text import TfidfVectorizer
from collections import Counter
from scipy.sparse import hstack
from sklearn.multiclass import OneVsRestClassifier
from sklearn.svm import SVC
from collections import Counter, defaultdict
from sklearn.calibration import CalibratedClassifierCV
from sklearn.naive_bayes import MultinomialNB
from sklearn.naive_bayes import GaussianNB
from sklearn.model_selection import train_test_split
from sklearn.model_selection import GridSearchCV
import math
from sklearn.metrics import normalized_mutual_info_score
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import cross_val_score
from sklearn.linear_model import SGDClassifier
from mlxtend.classifier import StackingClassifier
from sklearn import model_selection
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import precision_recall_curve, auc, roc_curve

import cufflinks as cf
cf.go_offline()
```

In [4]:

```
#traing data
df_train = pd.read_csv("application_train.csv")
df_train.head()
```

Out[4]:

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

5 rows × 122 columns

In [5]:

```
#test data
df_test = pd.read_csv("application_test.csv")
df_test.head()
```

Out[5]:

	SK_ID_CURR	NAME_CONTRACT_TYPE	CODE_GENDER	FLAG_OWN_CAR	FLAG_OWN_REAL
0	100001	Cash loans	F	N	
1	100005	Cash loans	M	N	
2	100013	Cash loans	M	Y	
3	100028	Cash loans	F	N	
4	100038	Cash loans	M	Y	

5 rows × 121 columns

finding missing value

In [6]:

```
count_train = df_train.isnull().sum().sort_values(ascending=False)
percentage_train = ((df_train.isnull().sum()/len(df_train)*100)).sort_values(ascending=Fals
missing_df_train = pd.concat([count_train, percentage_train], axis=1, keys=['Count_train','
print('Count_train and Percentage_train of missing values for top 38 columns:')
missing_df_train.head(38)
```

```
Count_train and Percentage_train of missing values for top 38 columns:
```

Out[6]:

	Count_train	Percentage_train
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
BASEMENTAREA_MEDI	179943	58.515956
BASEMENTAREA_AVG	179943	58.515956
BASEMENTAREA_MODE	179943	58.515956
EXT_SOURCE_1	173378	56.381073
NONLIVINGAREA_MEDI	169682	55.179164
NONLIVINGAREA_AVG	169682	55.179164
NONLIVINGAREA_MODE	169682	55.179164
ELEVATORS_MODE	163891	53.295980
ELEVATORS_AVG	163891	53.295980

	Count_train	Percentage_train
ELEVATORS_MEDI	163891	53.295980
WALLSMATERIAL_MODE	156341	50.840783
APARTMENTS_MODE	156061	50.749729
APARTMENTS_AVG	156061	50.749729
APARTMENTS_MEDI	156061	50.749729
ENTRANCES_MEDI	154828	50.348768
ENTRANCES_MODE	154828	50.348768
ENTRANCES_AVG	154828	50.348768

check for duplicate data

In [8]:

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

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

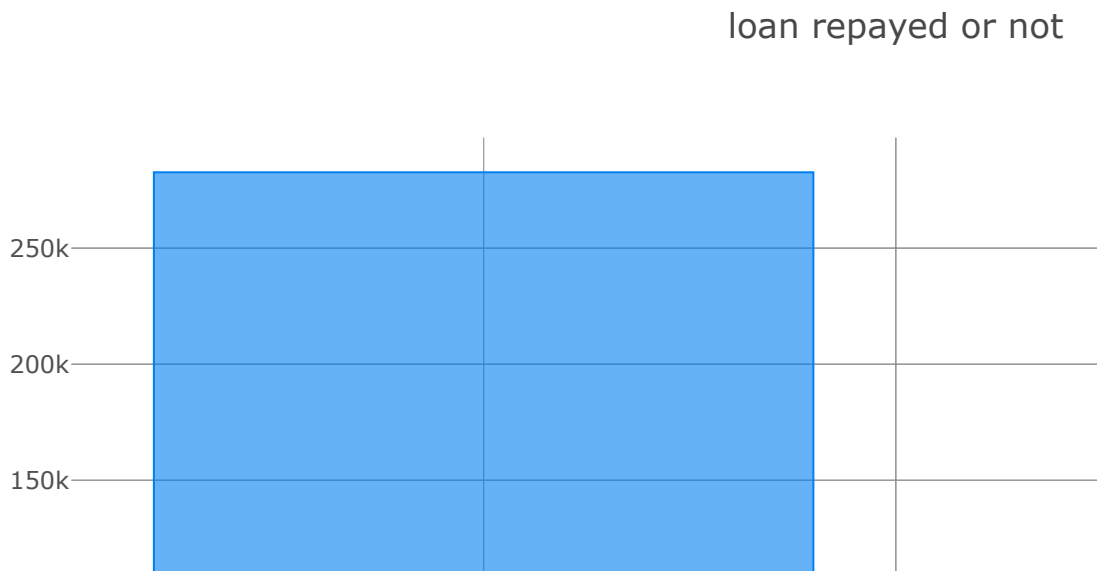
In [9]:

```
def plot_hist(col, func=None, **args):
    args['bins'] = args.get('bins', 50)
    args['alpha'] = args.get('alpha', .9)
    fig, axes = plt.subplots(figsize=(12, 3), ncols=2)
    train = df_train[col].dropna()
    test = df_test[col].dropna()
    if func:
        axes[0].hist(func(train[df_train.TARGET==0]), label='0', **args)
        axes[0].hist(func(train[df_train.TARGET==1]), label='1', **args)
        axes[1].hist(func(train), label='train', **args)
        axes[1].hist(func(test), label='test', **args)
    else:
        axes[0].hist(train[df_train.TARGET==0], label='0', **args)
        axes[0].hist(train[df_train.TARGET==1], label='1', **args)
        axes[1].hist(train, label='train', **args)
        axes[1].hist(test, label='test', **args)
    axes[0].set_ylabel(col)
    axes[0].legend()
    axes[1].legend()
    plt.show()
```

check the distribution of data points among output class.

In [18]:

```
cf.set_config_file(theme='polar')
contract_val = df_train['TARGET'].value_counts()
contract_val.iplot(kind='bar',labels='labels',values='values', title='loan repayed or not')
```

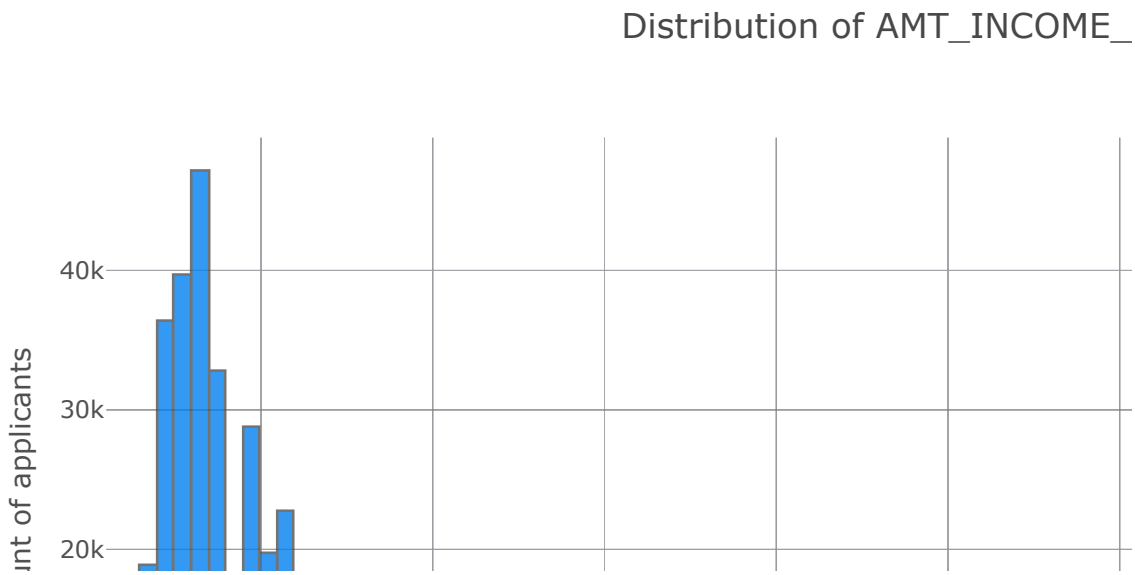


As we can see above graph people are able to repayed a loan is greater than not able to repayed loan, therefore this is imbalance data set

Distribution of AMT_INCOME_TOTAL.

In [20]:

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



In [21]:

```
(df_train[df_train['AMT_INCOME_TOTAL'] > 1000000]['TARGET'].value_counts())/len(df_train[df
```

Out[21]:

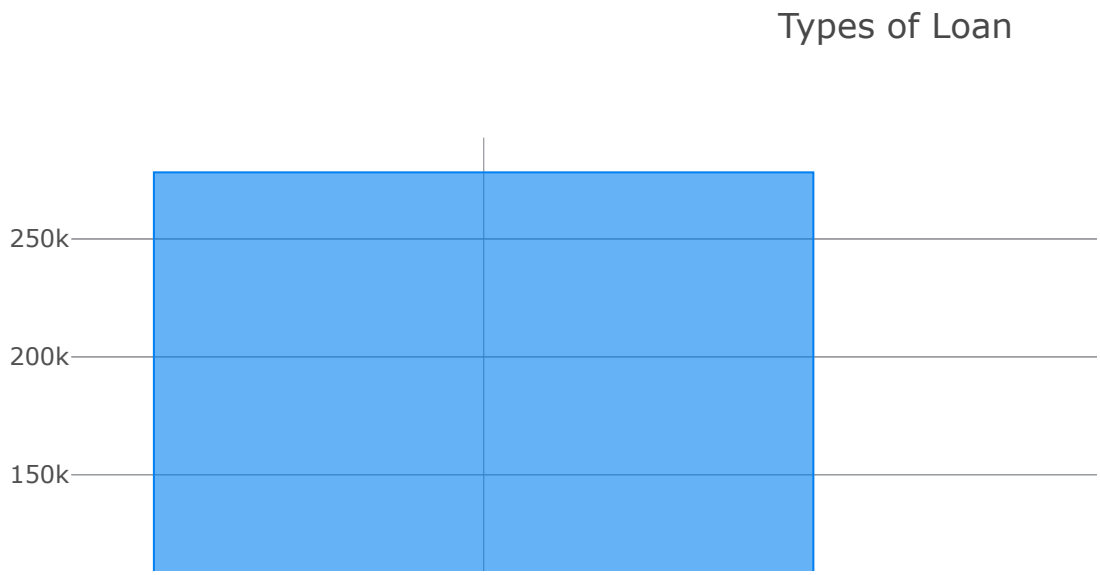
```
0    94.8
1     5.2
Name: TARGET, dtype: float64
```

people AMT_INCOME_TOTAL> 1000k are 94.8% able to repayed a loan

Types of loan available

In [22]:

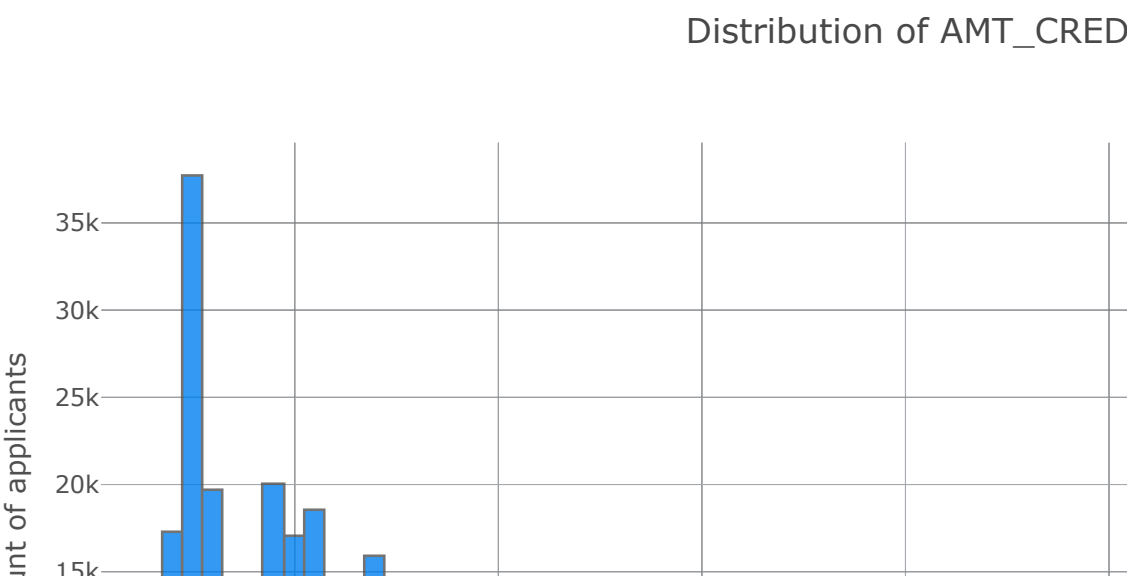
```
cf.set_config_file(theme='polar')
contract_val = df_train['NAME_CONTRACT_TYPE'].value_counts()
contract_val.iplot(kind='bar',labels='labels',values='values', title='Types of Loan')
```



Distribution of AMT_CREDIT

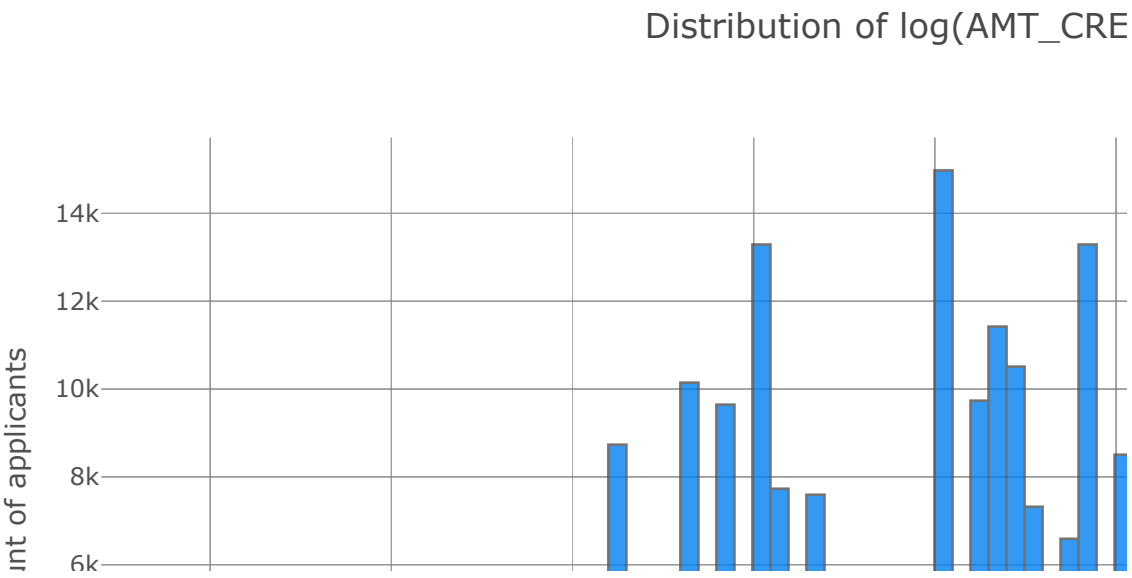
In [23]:

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



In [24]:

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



Observations:

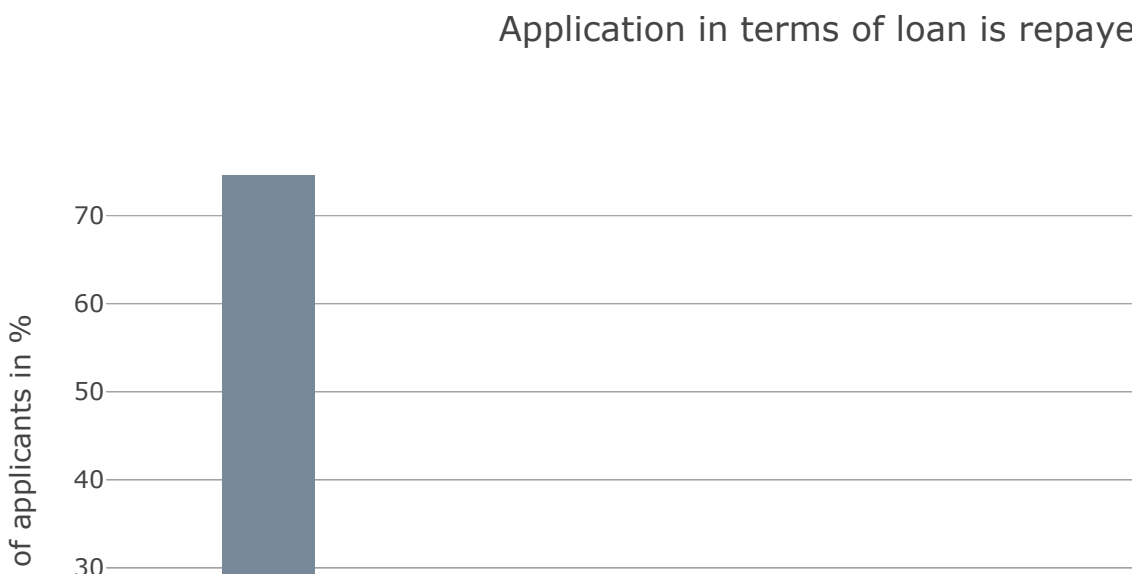
1. People who are taking credit for large amount are very likely to repay the loan.
2. Originally the distribution is right skewed, we used log transformation to make it normal distributed.

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

In [41]:

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

suite = df_train['NAME_TYPE_SUITE'].value_counts()
suite_y0 = []
suite_y1 = []
for i in suite.index:
    suite_y1.append(np.sum(df_train['TARGET'][df_train['NAME_TYPE_SUITE']==i] == 1))
    suite_y0.append(np.sum(df_train['TARGET'][df_train['NAME_TYPE_SUITE']==i] == 0))
data_suite = [go.Bar(x = suite.index, y = ((suite_y1 / suite.sum()) * 100), marker_color='c
        go.Bar(x = suite.index, y = ((suite_y0 / suite.sum()) * 100),marker_color='lightsla
layout_suite = go.Layout(
    title = "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_suite, layout=layout_suite)
py.iplot(fig)
```

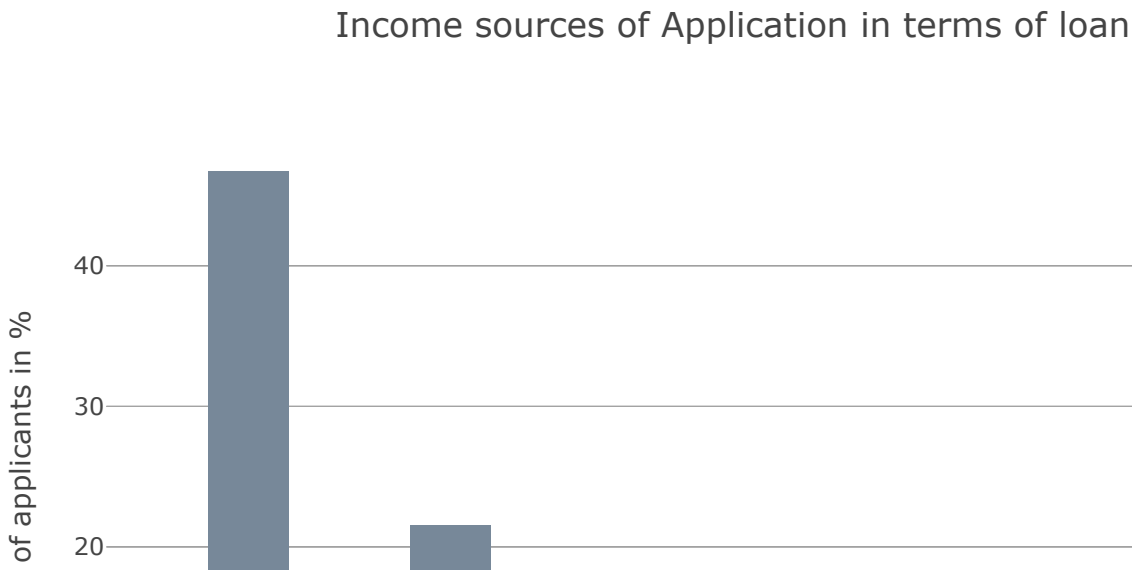


Income sources of Applicants in terms of loan is repayed or not.

In [42]:

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

income = df_train['NAME_INCOME_TYPE'].value_counts()
income_y0 = []
income_y1 = []
for i in income.index:
    income_y1.append(np.sum(df_train['TARGET'][df_train['NAME_INCOME_TYPE']==i] == 1))
    income_y0.append(np.sum(df_train['TARGET'][df_train['NAME_INCOME_TYPE']==i] == 0))
data_income = [go.Bar(x = income.index, y = ((income_y1 / income.sum()) * 100), marker_colo
        go.Bar(x = income.index, y = ((income_y0 / income.sum()) * 100),marker_color='light
layout_income = go.Layout(
    title = "Income sources of Application in terms of loan is repayed or not in %",
    xaxis=dict(
        title='Name of Income sources',
       ),
    yaxis=dict(
        title='Count of applicants in %',
        )
)
fig = go.Figure(data = data_income, layout=layout_income)
py.iplot(fig)
```



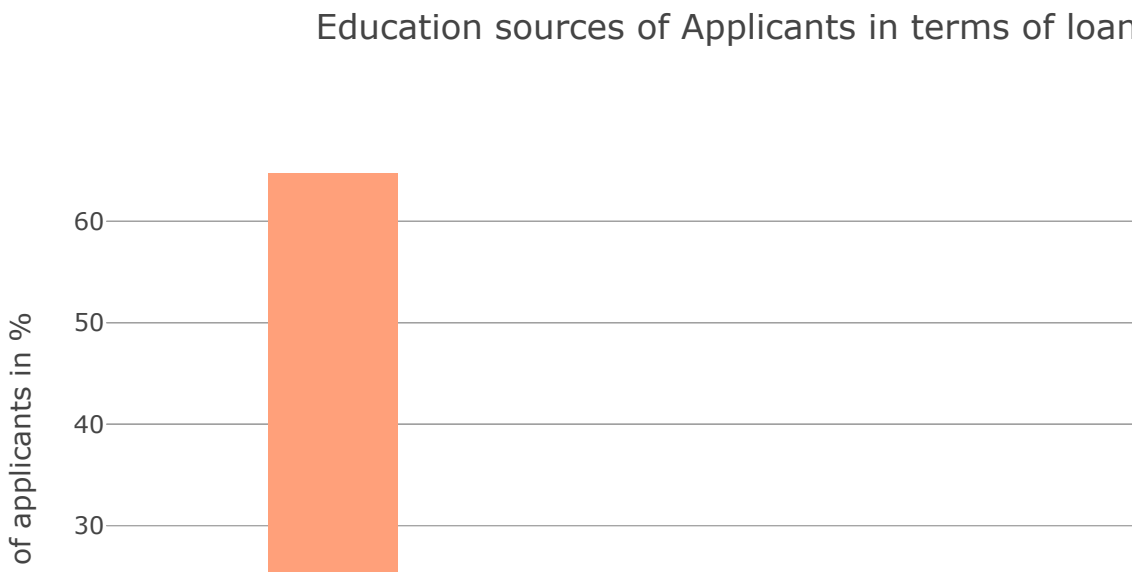
Observations:

1. All the Students and Businessman are repaying loan.

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

In [43]:

```
education = df_train['NAME_EDUCATION_TYPE'].value_counts()
education_y0 = []
education_y1 = []
for i in education.index:
    education_y1.append(np.sum(df_train['TARGET'][df_train['NAME_EDUCATION_TYPE']==i] == 1)
    education_y0.append(np.sum(df_train['TARGET'][df_train['NAME_EDUCATION_TYPE']==i] == 0)
data_education = [go.Bar(x = education.index, y = ((education_y1 / education.sum()) * 100),
        go.Bar(x = education.index, y = ((education_y0 / education.sum()) * 100),marker_col
layout_education = 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_education, layout=layout_education)
py.iplot(fig)
```



In [ ]:

```
print(np.sum(df_train['TARGET'][df_train['NAME_EDUCATION_TYPE']==i] == 1))
```

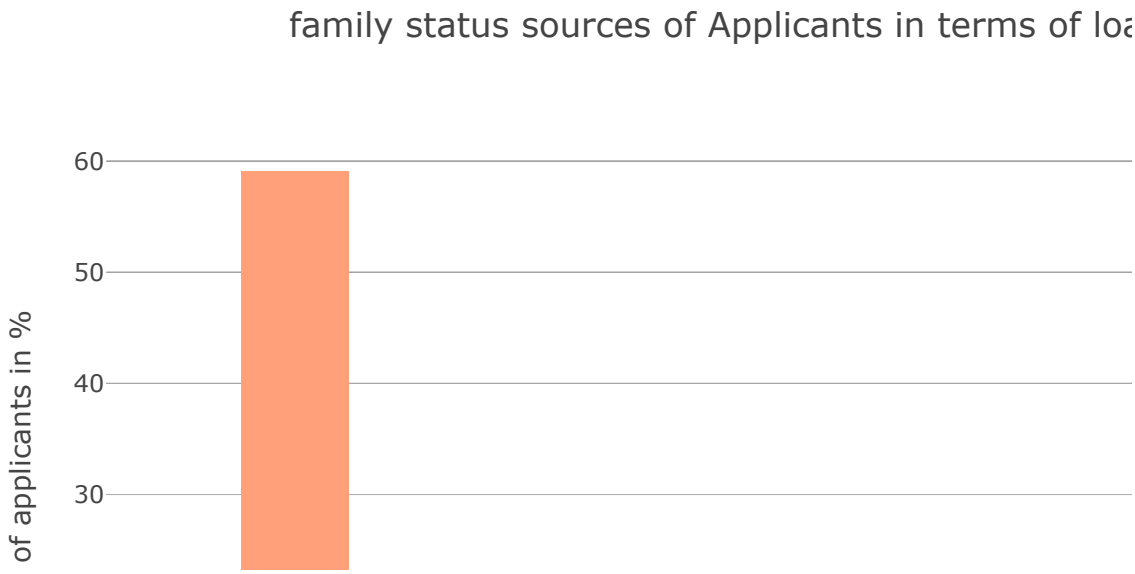
Observations:

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 [44]:

```
family_status= df_train['NAME_FAMILY_STATUS'].value_counts()
family_status_y0 = []
family_status_y1 = []
for i in family_status.index:
    family_status_y1.append(np.sum(df_train['TARGET'][df_train['NAME_FAMILY_STATUS']==i] ==
    family_status_y0.append(np.sum(df_train['TARGET'][df_train['NAME_FAMILY_STATUS']==i] ==
data_family_status = [go.Bar(x = family_status.index, y = ((family_status_y1 / family_statu
        go.Bar(x = family_status.index, y = ((family_status_y0 / family_status.sum()) * 100
layout_family_status = go.Layout(
    title = "family status sources 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_family_status, layout=layout_family_status)
py.iplot(fig)
```



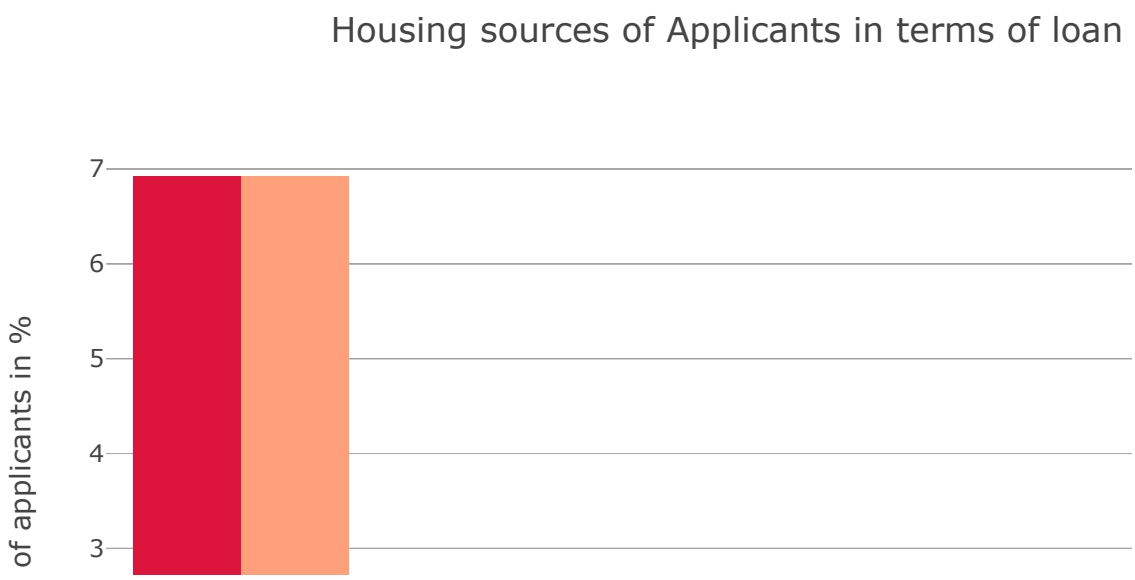
Observations:

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 [45]:

```
Housing_val = df_train['NAME_HOUSING_TYPE'].value_counts()
Housing_val_y0 = []
Housing_val_y1 = []
for val in Housing_val.index:
    Housing_val_y1.append(np.sum(df_train['TARGET'][df_train['NAME_HOUSING_TYPE']==val] ==
    Housing_val_y0.append(np.sum(df_train['TARGET'][df_train['NAME_HOUSING_TYPE']==val] ==
data = [go.Bar(x = Housing_val.index, y = ((Housing_val_y1 / Housing_val.sum()) * 100),mark
        go.Bar(x = Housing_val.index, y = ((Housing_val_y1 / Housing_val.sum()) * 100),mark
layout = go.Layout(
    title = "Housing sources of Applicants in terms of loan is repayed or not  in %",
    xaxis=dict(
        title='Type of House',
       ),
    yaxis=dict(
        title='Count of applicants in %',
        )
)
fig = go.Figure(data = data, layout=layout)
py.iplot(fig)
```



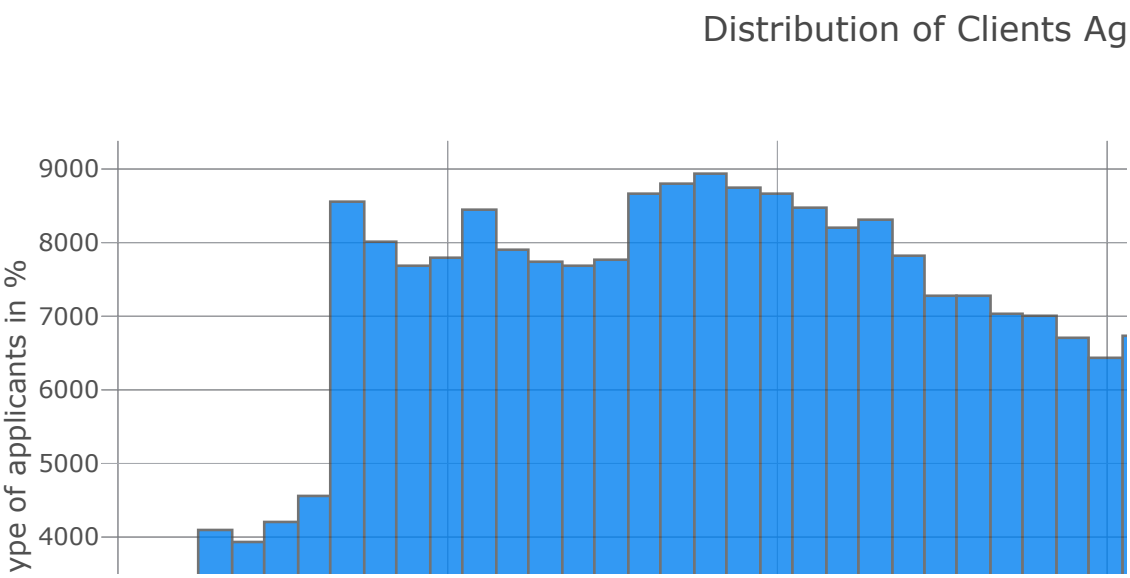
Distribution of Clients Age

In [46]:

```
df_train ['AGE'] = - (df_train ['DAYS_BIRTH']/365.25).astype('int32')
```

In [47]:

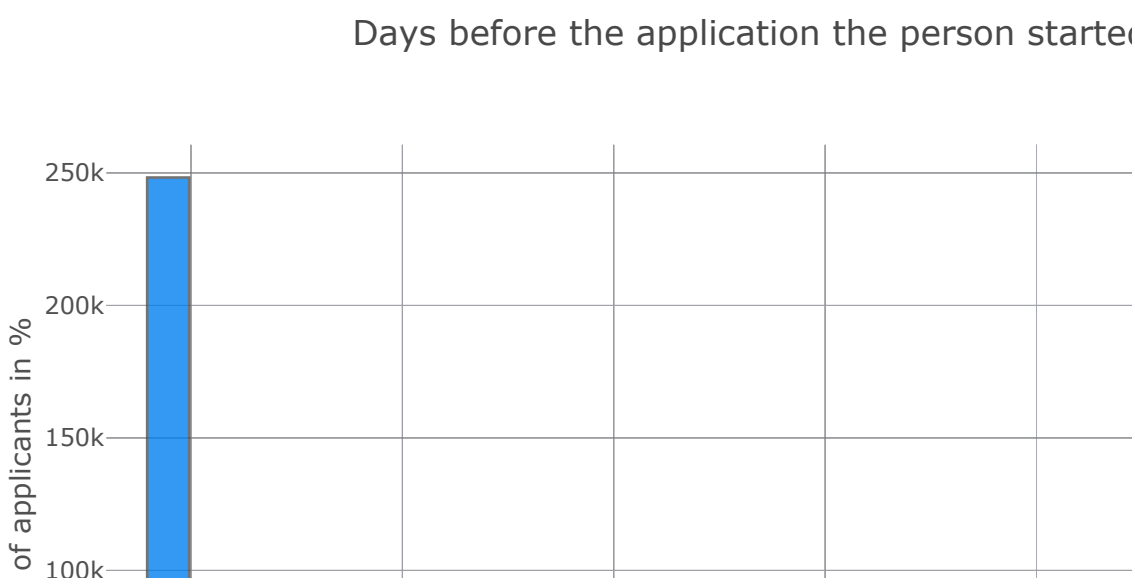
```
cf.set_config_file(theme='polar')
(df_train['AGE']).iplot(kind='histogram',
             xTitle = 'Age', bins=50,
             yTitle='Count of type of applicants in %',
             title='Distribution of Clients Age')
```



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

In [48]:

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



In [7]:

```
df_train['DAYS_EMPLOYED']
```

Out[7]:

```
0           -637
1          -1188
2           -225
3          -3039
4          -3038
           ...
307506      -236
307507    365243
307508     -7921
307509     -4786
307510     -1262
Name: DAYS_EMPLOYED, Length: 307511, dtype: int64
```

Observations:

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

In [49]:

```
error = df_train[df_train['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[49]:

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

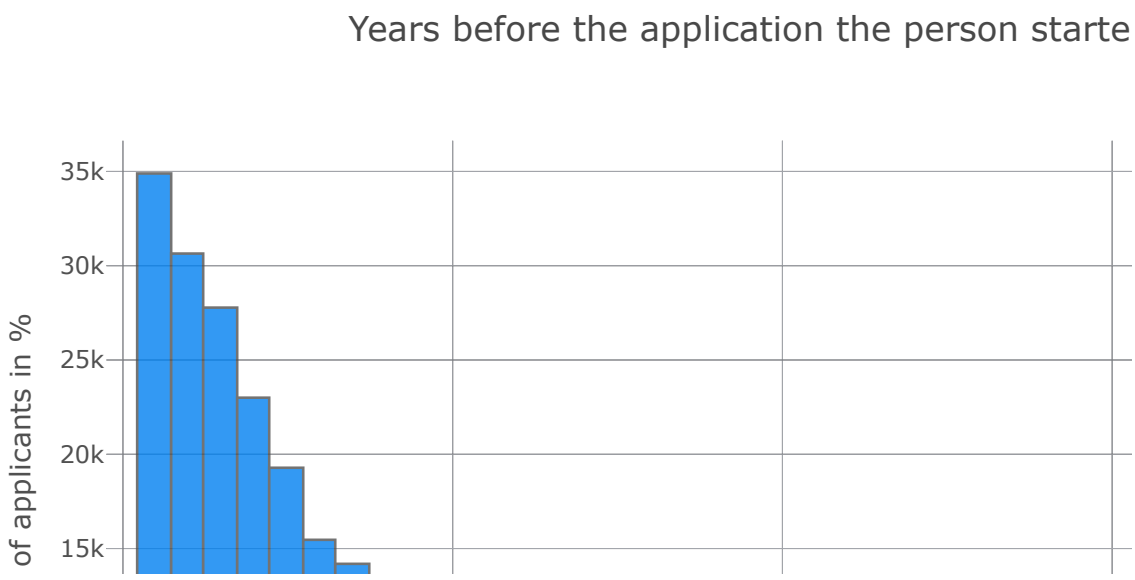
In [50]:

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

Created a seperate column ‘DAYS_EMPLOYED_ERROR’, which flags the error.

In [51]:

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



In [52]:

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

Out[52]:

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

Observations:

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