The objective this project work is to use historical loan application data to predict whether or not an applicant will be able to repay the loan.

This is a standard Supervised classification task.

Supervised: The labels are included in the training data and the goal is to train a model to lear to predict the lables from the features

Classification: The target label ('TARGET') is a binary variable having values 0 and 1 Target = 0 will repay loan on time & Target = 1 will have diffulty repaying loan

In [3]:
```
#Importing data manipulation packages
import numpy as np
import pandas as pd

# sklearn preprocessing for dealing with categorical variables
from sklearn.preprocessing import LabelEncoder
from sklearn.metrics import classification_report, confusion_matrix,accuracy_scoı


# File system manangement
import os

# Suppress warnings
import warnings
warnings.filterwarnings('ignore')

# matplotlib and seaborn for plotting
import matplotlib.pyplot as plt
import seaborn as sns

import matplotlib
import matplotlib.pyplot as plt # for plotting
color = sns.color_palette()
import plotly.offline as py
py.init_notebook_mode(connected=True)
from plotly.offline import init_notebook_mode, iplot
init_notebook_mode(connected=True)
import plotly.graph_objs as go
import plotly.offline as offline
offline.init_notebook_mode()
# from plotly import tools
# import plotly.tools as tls
# import squarify
# from mpl_toolkits.basemap import Basemap
# from numpy import array
# from matplotlib import cm

# import cufflinks and offline mode
import cufflinks as cf
cf.go_offline()
```


DATA

application_{train|test}.csv :

This is the main table, broken into two files for Train (with TARGET) and Test (without TARGET). Static data for all applications. One row represents one loan in our data sample.

bureau.csv

All client's previous credits provided by other financial institutions that were reported to Credit Bureau (for clients who have a loan in our sample).For every loan in our sample, there are as many rows as number of credits the client had in Credit Bureau before the application date.

bureau_balance.csv

Monthly balances of previous credits in Credit Bureau.This table has one row for each month of history of every previous credit reported to Credit Bureau – i.e the table has (#loans in sample * # of relative previous credits * # of months where we have some history observable for the previous credits) rows.

previous_application.csv

All previous applications for Home Credit loans of clients who have loans in our sample. There is one row for each previous application related to loans in our data sample.

installments_payments.csv

Repayment history for the previously disbursed credits in Home Credit related to the loans in our sample. There is a) one row for every payment that was made plus b) one row each for missed payment. One row is equivalent to one payment of one installment OR one installment corresponding to one payment of one previous Home Credit credit related to loans in our sample.

In [4]:
```
# reading CSV files data and storing them into data pandas data frames
train = pd.read_csv('C:/Users/amite/Downloads/application_train.csv')
#Since test data doesn't have target variable, we decided to split train data lat
#test = pd.read_csv('application_test.csv')
```

In [5]:
```
#We are getting key memory errors subesequently while joiningg below files with r
#reading partial percentage of data, re-iterated and changed percentages to neces
#we chose to run subsequently. Due to missing values we have to re-iterate severa
#to resolve memory error issue

previous_application = pd.read_csv('C:/Users/amite/Downloads/previous_applicatio
installments_payments = pd.read_csv('C:/Users/amite/Downloads/installments_paymen
bureau = pd.read_csv('C:/Users/amite/Downloads/bureau.csv').sample(frac =.2)
```

In [6]:
```
print("TRAIN DATA",train.shape)
train.head()
```

```
TRAIN DATA (307511, 122)
```

Out[6]:

	SK_ID_CURR	TARGET	NAME_CONTRACT_TYPE	CODE_GENDER	FLAG_OWN_CAR	FLAG_OWN
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 [7]:
```
print("BUREAU DATA",bureau.shape)
bureau.head()
```

```
BUREAU DATA (343286, 17)
```

Out[7]:

	SK_ID_CURR	SK_ID_BUREAU	CREDIT_ACTIVE	CREDIT_CURRENCY	DAYS_CREDIT	CRE
720610	314201	6493695	Active	currency 1	-151	
154920	323475	6593290	Active	currency 1	-1064	
820011	397787	6794248	Active	currency 1	-88	
296380	418107	5174888	Closed	currency 1	-1297	
1543523	371623	6678664	Active	currency 1	-293	

In [8]:
```
#Keeping Train data as main dataset and left join with Bureau data
train_bureau = pd.merge(train,bureau,how='left',on='SK_ID_CURR')
train_bureau.shape
```

Out[8]:
```
(441236, 138)
```

In [9]:
```
train_bureau.columns
```

Out[9]:
```
Index(['SK_ID_CURR', 'TARGET', 'NAME_CONTRACT_TYPE', 'CODE_GENDER',
       'FLAG_OWN_CAR', 'FLAG_OWN_REALTY', 'CNT_CHILDREN', 'AMT_INCOME_TOTAL',
       'AMT_CREDIT', 'AMT_ANNUITY_x',
       ...
       'DAYS_ENDDATE_FACT', 'AMT_CREDIT_MAX_OVERDUE', 'CNT_CREDIT_PROLONG',
       'AMT_CREDIT_SUM', 'AMT_CREDIT_SUM_DEBT', 'AMT_CREDIT_SUM_LIMIT',
       'AMT_CREDIT_SUM_OVERDUE', 'CREDIT_TYPE', 'DAYS_CREDIT_UPDATE',
       'AMT_ANNUITY_y'],
      dtype='object', length=138)
```

In [10]:
```
# data exploration glimpse for Installment payment data
print("Installments Payments DATA",installments_payments.shape)
installments_payments.head()
```

Installments Payments DATA (2721080, 8)

Out[10]:

	SK_ID_PREV	SK_ID_CURR	NUM_INSTALMENT_VERSION	NUM_INSTALMENT_NUMBER	D
12283802	2239099	452035	3.0	7	
1940157	2177717	106243	1.0	6	
1834769	2049485	177933	1.0	32	
6475795	1029619	295132	0.0	78	
401186	2409684	143622	1.0	10	

In [11]:
```
# data exploration glimpse for previous application data
print("Previous Application DATA",previous_application.shape)
previous_application.head()
```

Previous Application DATA (334043, 37)

Out[11]:

	SK_ID_PREV	SK_ID_CURR	NAME_CONTRACT_TYPE	AMT_ANNUITY	AMT_APPLICATION
1286571	1508511	249817	Cash loans	23844.780	454500.0
1162999	1079215	210156	Consumer loans	5152.230	57555.0
71572	2828848	392220	Cash loans	34117.155	1125000.0
253384	1006325	268383	Consumer loans	4795.515	106330.5
405929	2346080	419957	Consumer loans	6386.940	57028.5

5 rows × 37 columns

In [12]:
```
# Innner join previous Loan application data and corresponding installement payme

previous_payments = pd.merge(previous_application,installments_payments,how='inn
previous_payments.shape
```

Out[12]: (496043, 44)

In [13]:
```
# Data exploration glimpse on previous loan and corresponding payments data
# Revolving loan/credit : Isn't issued in a perdetermined amount. Credit cars are
#           Most credit scoring models penlize you for using over 30% of your ava
# Consumer loan/Installment credit - Any loan(car loan, cash loan etc) except Hom
#                                    is detrmined at the time of loan approval.
# cash loan - Credit card cash advance usually have higher interest rate.This is

print("Previous Application and Payments DATA",previous_payments.shape)
previous_payments.head(10)
```

Previous Application and Payments DATA (496043, 44)

Out[13]:

	SK_ID_PREV	SK_ID_CURR_x	NAME_CONTRACT_TYPE	AMT_ANNUITY	AMT_APPLICATION
0	1006325	268383	Consumer loans	4795.515	106330.5
1	1006325	268383	Consumer loans	4795.515	106330.5
2	1006325	268383	Consumer loans	4795.515	106330.5
3	2346080	419957	Consumer loans	6386.940	57028.5
4	2346080	419957	Consumer loans	6386.940	57028.5
5	2346080	419957	Consumer loans	6386.940	57028.5
6	1521919	422902	Cash loans	19172.655	180000.0
7	1521919	422902	Cash loans	19172.655	180000.0
8	1258043	187310	Consumer loans	3579.030	22432.5

In [14]:
```
# Merging payment data with main application-bureaue dataset (left join)
merged_data = pd.merge(train_bureau,previous_payments,how='inner',left_on='SK_ID_
merged_data.shape
```

Out[14]: (643379, 182)

In [15]:
```
# Data exploration glimpse on consolidated final dataset
print("merged_data DATA",merged_data.shape)
merged_data.head()
```

```
merged_data DATA (643379, 182)
```

Out[15]:

	SK_ID_CURR	TARGET	NAME_CONTRACT_TYPE_x	CODE_GENDER	FLAG_OWN_CAR	FLAG_O\
0	100002	1	Cash loans	M	N	
1	100002	1	Cash loans	M	N	
2	100002	1	Cash loans	M	N	
3	100002	1	Cash loans	M	N	
4	100002	1	Cash loans	M	N	

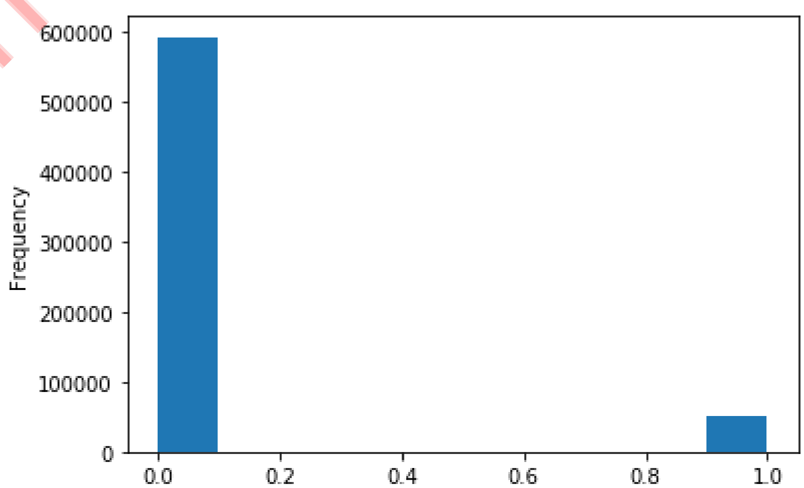
5 rows × 182 columns

In [16]:
```
# Checking balance /unbalance on target variable and data shows it is unbalanced

merged_data['TARGET'].value_counts()
```

Out[16]:
```
0    592011
1     51368
Name: TARGET, dtype: int64
```

In [17]:
```
merged_data['TARGET'].astype(int).plot.hist()
```

Out[17]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x27e4d9889b0>
```



In [18]:
```
#Checking null values in text format such as 'Unknown'
z = []
i = len(merged_data.columns)
for i in merged_data.columns:
    if merged_data[i].dtypes == 'object':
        z.append(i)

for i in z:
    print(i,merged_data[i].unique())
```

```
NAME_CONTRACT_TYPE_x ['Cash loans' 'Revolving loans']
CODE_GENDER ['M' 'F' 'XNA']
FLAG_OWN_CAR ['N' 'Y']
FLAG_OWN_REALTY ['Y' 'N']
NAME_TYPE_SUITE_x ['Unaccompanied' 'Spouse, partner' 'Children' 'Family' nan 'O
ther_A'
 'Other_B' 'Group of people']
NAME_INCOME_TYPE ['Working' 'State servant' 'Pensioner' 'Commercial associate'
'Unemployed'
 'Student' 'Maternity leave']
NAME_EDUCATION_TYPE ['Secondary / secondary special' 'Higher education' 'Incomp
lete higher'
 'Lower secondary' 'Academic degree']
NAME_FAMILY_STATUS ['Single / not married' 'Married' 'Civil marriage' 'Separate
d' 'Widow']
NAME_HOUSING_TYPE ['House / apartment' 'With parents' 'Municipal apartment'
 'Office apartment' 'Rented apartment' 'Co-op apartment']
OCCUPATION_TYPE ['Laborers' 'Core staff' nan 'Drivers' 'Cleaning staff'
 'Private service staff' 'Sales staff' 'Medicine staff' 'Managers'
 'Waiters/barmen staff' 'Realty agents' 'High skill tech staff'
 'Accountants' 'Cooking staff' 'Secretaries' 'Security staff'
 'Low-skill Laborers' 'HR staff' 'IT staff']
WEEKDAY_APPR_PROCESS_START_x ['WEDNESDAY' 'THURSDAY' 'FRIDAY' 'MONDAY' 'SATURDA
Y' 'TUESDAY' 'SUNDAY']
ORGANIZATION_TYPE ['Business Entity Type 3' 'Religion' 'Other' 'XNA' 'Electrici
ty'
 'Business Entity Type 2' 'Transport: type 2' 'Construction'
 'Industry: type 11' 'Transport: type 4' 'Self-employed' 'Services'
 'Medicine' 'Trade: type 2' 'University' 'Government' 'School' 'Postal'
 'Industry: type 4' 'Restaurant' 'Kindergarten' 'Culture' 'Trade: type 7'
 'Hotel' 'Industry: type 3' 'Bank' 'Military' 'Trade: type 3' 'Housing'
 'Business Entity Type 1' 'Agriculture' 'Police' 'Industry: type 9'
 'Industry: type 12' 'Transport: type 3' 'Security Ministries' 'Security'
 'Industry: type 7' 'Industry: type 5' 'Industry: type 1' 'Trade: type 6'
 'Emergency' 'Industry: type 10' 'Industry: type 13' 'Industry: type 2'
 'Industry: type 8' 'Advertising' 'Insurance' 'Legal Services' 'Mobile'
 'Telecom' 'Realtor' 'Trade: type 1' 'Industry: type 6'
 'Transport: type 1' 'Cleaning' 'Trade: type 5' 'Trade: type 4']
FONDKAPREMONT_MODE ['reg oper account' nan 'reg oper spec account' 'not specifi
ed'
 'org spec account']
HOUSETYPE_MODE ['block of flats' nan 'terraced house' 'specific housing']
WALLSMATERIAL_MODE ['Stone, brick' nan 'Panel' 'Others' 'Monolithic' 'Block' 'W
ooden' 'Mixed']
EMERGENCYSTATE_MODE ['No' nan 'Yes']
CREDIT_ACTIVE [nan 'Closed' 'Active' 'Sold']
```

```
CREDIT_CURRENCY [nan 'currency 1' 'currency 2' 'currency 3' 'currency 4']
CREDIT_TYPE [nan 'Consumer credit' 'Credit card' 'Mortgage' 'Microloan' 'Car lo
an'
 'Loan for working capital replenishment' 'Loan for business development'
 'Loan for the purchase of equipment' 'Another type of loan'
 'Cash loan (non-earmarked)' 'Real estate loan' 'Unknown type of loan']
NAME_CONTRACT_TYPE_y ['Consumer loans' 'Cash loans' 'Revolving loans']
WEEKDAY_APPR_PROCESS_START_y ['SATURDAY' 'THURSDAY' 'SUNDAY' 'MONDAY' 'FRIDAY'
'WEDNESDAY' 'TUESDAY']
FLAG_LAST_APPL_PER_CONTRACT ['Y']
NAME_CASH_LOAN_PURPOSE ['XAP' 'XNA' 'Medicine' 'Repairs' 'Buying a holiday home
/ land'
 'Buying a new car' 'Other' 'Journey' 'Everyday expenses'
 'Purchase of electronic equipment' 'Urgent needs' 'Buying a used car'
 'Car repairs' 'Wedding / gift / holiday' 'Building a house or an annex'
 'Payments on other loans' 'Furniture' 'Education' 'Buying a home'
 'Gasification / water supply' 'Business development' 'Hobby'
 'Buying a garage' 'Refusal to name the goal']
NAME_CONTRACT_STATUS ['Approved']
NAME_PAYMENT_TYPE ['XNA' 'Cash through the bank' 'Cashless from the account of
the employer'
 'Non-cash from your account']
CODE_REJECT_REASON ['XAP' 'XNA']
NAME_TYPE_SUITE_y [nan 'Unaccompanied' 'Spouse, partner' 'Family' 'Other_A' 'Ot
her_B'
 'Children' 'Group of people']
NAME_CLIENT_TYPE ['New' 'Repeater' 'Refreshed' 'XNA']
NAME_GOODS_CATEGORY ['Vehicles' 'XNA' 'Consumer Electronics' 'Clothing and Acce
ssories'
 'Construction Materials' 'Computers' 'Mobile' 'Audio/Video' 'Furniture'
 'Other' 'Photo / Cinema Equipment' 'Office Appliances' 'Homewares'
 'Jewelry' 'Medical Supplies' 'Auto Accessories' 'Sport and Leisure'
 'Gardening' 'Tourism' 'Weapon' 'Fitness' 'Medicine' 'Additional Service'
 'Insurance' 'Direct Sales' 'Education']
NAME_PORTFOLIO ['POS' 'Cash' 'Cards' 'Cars']
NAME_PRODUCT_TYPE ['XNA' 'x-sell' 'walk-in']
CHANNEL_TYPE ['Stone' 'Country-wide' 'Regional / Local' 'Contact center'
 'Credit and cash offices' 'AP+ (Cash loan)' 'Car dealer'
 'Channel of corporate sales']
NAME_SELLER_INDUSTRY ['Auto technology' 'Consumer electronics' 'XNA' 'Clothing'
'Construction'
 'Connectivity' 'Furniture' 'Industry' 'Jewelry' 'Tourism' 'MLM partners']
NAME_YIELD_GROUP ['low_normal' 'middle' 'high' 'XNA' 'low_action']
PRODUCT_COMBINATION ['POS other with interest' 'Cash X-Sell: middle' 'Cash Stre
et: high'
 'POS household with interest' 'Card X-Sell' 'Cash Street: low'
 'Cash X-Sell: high' 'POS industry with interest'
 'POS mobile with interest' 'Cash Street: middle'
 'POS household without interest' 'Cash X-Sell: low'
 'POS industry without interest' 'Card Street'
 'POS mobile without interest' 'POS others without interest']
```

In [19]:
```
# Data clean up for the following and these are very small percentage so clean up
merged_data=merged_data[merged_data!='XNA']
merged_data=merged_data[merged_data!='Unknown']
merged_data=merged_data[merged_data!='not specified']
```

In [20]:
```
merged_data.CODE_GENDER.unique()   # xna removed
```

Out[20]:
```
array(['M', 'F', nan], dtype=object)
```

In [21]:
```
merged_data.NAME_SELLER_INDUSTRY.unique()   #xna dropped
```

Out[21]:
```
array(['Auto technology', 'Consumer electronics', nan, 'Clothing',
       'Construction', 'Connectivity', 'Furniture', 'Industry', 'Jewelry',
       'Tourism', 'MLM partners'], dtype=object)
```

In [22]:
```
merged_data.shape
```

Out[22]:
```
(643379, 182)
```

In [23]:
```
# Columns review on merged_data data structure
# checking same column names how appeared in final dataset
#application_train: 'NAME_CONTRACT_TYPE_x','AMT_CREDIT_x', 'AMT_ANNUITY_x', 'AMT_
#previous_application:'NAME_CONTRACT_TYPE_y','AMT_CREDIT_y','AMT_ANNUITY_y','AMT_


list(merged_data.columns)
```

Out[23]:
```
['SK_ID_CURR',
 'TARGET',
 'NAME_CONTRACT_TYPE_x',
 'CODE_GENDER',
 'FLAG_OWN_CAR',
 'FLAG_OWN_REALTY',
 'CNT_CHILDREN',
 'AMT_INCOME_TOTAL',
 'AMT_CREDIT_x',
 'AMT_ANNUITY_x',
 'AMT_GOODS_PRICE_x',
 'NAME_TYPE_SUITE_x',
 'NAME_INCOME_TYPE',
 'NAME_EDUCATION_TYPE',
 'NAME_FAMILY_STATUS',
 'NAME_HOUSING_TYPE',
 'REGION_POPULATION_RELATIVE',
 'DAYS_BIRTH',
 'DAYS_EMPLOYED',
```

In [24]:
```
#Checking null values in text format such as XNA
z = []
i = len(merged_data.columns)
for i in merged_data.columns:
    if merged_data[i].dtypes == 'object':
        z.append(i)

for i in z:
    print(i,merged_data[i].unique())
```

```
NAME_CONTRACT_TYPE_x ['Cash loans' 'Revolving loans']
CODE_GENDER ['M' 'F' nan]
FLAG_OWN_CAR ['N' 'Y']
FLAG_OWN_REALTY ['Y' 'N']
NAME_TYPE_SUITE_x ['Unaccompanied' 'Spouse, partner' 'Children' 'Family' nan 'O
ther_A'
 'Other_B' 'Group of people']
NAME_INCOME_TYPE ['Working' 'State servant' 'Pensioner' 'Commercial associate'
'Unemployed'
 'Student' 'Maternity leave']
NAME_EDUCATION_TYPE ['Secondary / secondary special' 'Higher education' 'Incomp
lete higher'
 'Lower secondary' 'Academic degree']
NAME_FAMILY_STATUS ['Single / not married' 'Married' 'Civil marriage' 'Separate
d' 'Widow']
NAME_HOUSING_TYPE ['House / apartment' 'With parents' 'Municipal apartment'
 'Office apartment' 'Rented apartment' 'Co-op apartment']
OCCUPATION_TYPE ['Laborers' 'Core staff' nan 'Drivers' 'Cleaning staff'
 'Private service staff' 'Sales staff' 'Medicine staff' 'Managers'
 'Waiters/barmen staff' 'Realty agents' 'High skill tech staff'
 'Accountants' 'Cooking staff' 'Secretaries' 'Security staff'
 'Low-skill Laborers' 'HR staff' 'IT staff']
WEEKDAY_APPR_PROCESS_START_x ['WEDNESDAY' 'THURSDAY' 'FRIDAY' 'MONDAY' 'SATURDA
Y' 'TUESDAY' 'SUNDAY']
ORGANIZATION_TYPE ['Business Entity Type 3' 'Religion' 'Other' nan 'Electricit
y'
 'Business Entity Type 2' 'Transport: type 2' 'Construction'
 'Industry: type 11' 'Transport: type 4' 'Self-employed' 'Services'
 'Medicine' 'Trade: type 2' 'University' 'Government' 'School' 'Postal'
 'Industry: type 4' 'Restaurant' 'Kindergarten' 'Culture' 'Trade: type 7'
 'Hotel' 'Industry: type 3' 'Bank' 'Military' 'Trade: type 3' 'Housing'
 'Business Entity Type 1' 'Agriculture' 'Police' 'Industry: type 9'
 'Industry: type 12' 'Transport: type 3' 'Security Ministries' 'Security'
 'Industry: type 7' 'Industry: type 5' 'Industry: type 1' 'Trade: type 6'
 'Emergency' 'Industry: type 10' 'Industry: type 13' 'Industry: type 2'
 'Industry: type 8' 'Advertising' 'Insurance' 'Legal Services' 'Mobile'
 'Telecom' 'Realtor' 'Trade: type 1' 'Industry: type 6'
 'Transport: type 1' 'Cleaning' 'Trade: type 5' 'Trade: type 4']
FONDKAPREMONT_MODE ['reg oper account' nan 'reg oper spec account' 'org spec ac
count']
HOUSETYPE_MODE ['block of flats' nan 'terraced house' 'specific housing']
WALLSMATERIAL_MODE ['Stone, brick' nan 'Panel' 'Others' 'Monolithic' 'Block' 'W
ooden' 'Mixed']
EMERGENCYSTATE_MODE ['No' nan 'Yes']
CREDIT_ACTIVE [nan 'Closed' 'Active' 'Sold']
CREDIT_CURRENCY [nan 'currency 1' 'currency 2' 'currency 3' 'currency 4']
CREDIT_TYPE [nan 'Consumer credit' 'Credit card' 'Mortgage' 'Microloan' 'Car lo
```

```
an'
 'Loan for working capital replenishment' 'Loan for business development'
 'Loan for the purchase of equipment' 'Another type of loan'
 'Cash loan (non-earmarked)' 'Real estate loan' 'Unknown type of loan']
NAME_CONTRACT_TYPE_y ['Consumer loans' 'Cash loans' 'Revolving loans']
WEEKDAY_APPR_PROCESS_START_y ['SATURDAY' 'THURSDAY' 'SUNDAY' 'MONDAY' 'FRIDAY'
'WEDNESDAY' 'TUESDAY']
FLAG_LAST_APPL_PER_CONTRACT ['Y']
NAME_CASH_LOAN_PURPOSE ['XAP' nan 'Medicine' 'Repairs' 'Buying a holiday home /
land'
 'Buying a new car' 'Other' 'Journey' 'Everyday expenses'
 'Purchase of electronic equipment' 'Urgent needs' 'Buying a used car'
 'Car repairs' 'Wedding / gift / holiday' 'Building a house or an annex'
 'Payments on other loans' 'Furniture' 'Education' 'Buying a home'
 'Gasification / water supply' 'Business development' 'Hobby'
 'Buying a garage' 'Refusal to name the goal']
NAME_CONTRACT_STATUS ['Approved']
NAME_PAYMENT_TYPE [nan 'Cash through the bank' 'Cashless from the account of th
e employer'
 'Non-cash from your account']
CODE_REJECT_REASON ['XAP' nan]
NAME_TYPE_SUITE_y [nan 'Unaccompanied' 'Spouse, partner' 'Family' 'Other_A' 'Ot
her_B'
 'Children' 'Group of people']
NAME_CLIENT_TYPE ['New' 'Repeater' 'Refreshed' nan]
NAME_GOODS_CATEGORY ['Vehicles' nan 'Consumer Electronics' 'Clothing and Access
ories'
 'Construction Materials' 'Computers' 'Mobile' 'Audio/Video' 'Furniture'
 'Other' 'Photo / Cinema Equipment' 'Office Appliances' 'Homewares'
 'Jewelry' 'Medical Supplies' 'Auto Accessories' 'Sport and Leisure'
 'Gardening' 'Tourism' 'Weapon' 'Fitness' 'Medicine' 'Additional Service'
 'Insurance' 'Direct Sales' 'Education']
NAME_PORTFOLIO ['POS' 'Cash' 'Cards' 'Cars']
NAME_PRODUCT_TYPE [nan 'x-sell' 'walk-in']
CHANNEL_TYPE ['Stone' 'Country-wide' 'Regional / Local' 'Contact center'
 'Credit and cash offices' 'AP+ (Cash loan)' 'Car dealer'
 'Channel of corporate sales']
NAME_SELLER_INDUSTRY ['Auto technology' 'Consumer electronics' nan 'Clothing'
'Construction'
 'Connectivity' 'Furniture' 'Industry' 'Jewelry' 'Tourism' 'MLM partners']
NAME_YIELD_GROUP ['low_normal' 'middle' 'high' nan 'low_action']
PRODUCT_COMBINATION ['POS other with interest' 'Cash X-Sell: middle' 'Cash Stre
et: high'
 'POS household with interest' 'Card X-Sell' 'Cash Street: low'
 'Cash X-Sell: high' 'POS industry with interest'
 'POS mobile with interest' 'Cash Street: middle'
 'POS household without interest' 'Cash X-Sell: low'
 'POS industry without interest' 'Card Street'
 'POS mobile without interest' 'POS others without interest']
```

In [25]:
```
# checking the percentage of missing values in each variable
# notice we have missing percentages more than 100% , which is because of one to
# saving missing values in a variable
missing=merged_data.isnull().sum()/len(train)*100
missing.sort_values(ascending=False).head(100)
```

Out[25]:
```
RATE_INTEREST_PRIVILEGED      208.880983
RATE_INTEREST_PRIMARY         208.880983
AMT_ANNUITY_y                 175.479576
AMT_CREDIT_MAX_OVERDUE        157.426564
FONDKAPREMONT_MODE            145.062453
COMMONAREA_AVG                144.758724
COMMONAREA_MODE               144.758724
COMMONAREA_MEDI               144.758724
NONLIVINGAPARTMENTS_AVG       143.907698
NONLIVINGAPARTMENTS_MODE      143.907698
NONLIVINGAPARTMENTS_MEDI      143.907698
LIVINGAPARTMENTS_AVG          141.616723
LIVINGAPARTMENTS_MEDI         141.616723
LIVINGAPARTMENTS_MODE         141.616723
FLOORSMIN_MEDI                140.333191
FLOORSMIN_MODE                140.333191
FLOORSMIN_AVG                 140.333191
OWN_CAR_AGE                   137.567437
YEARS_BUILD_AVG               137.287121
YEARS_BUILD_MODE              137.287121
YEARS_BUILD_MEDI              137.287121
LANDAREA_MODE                 122.407654
LANDAREA_MEDI                 122.407654
LANDAREA_AVG                  122.407654
BASEMENTAREA_AVG              120.461382
BASEMENTAREA_MEDI             120.461382
BASEMENTAREA_MODE             120.461382
NAME_GOODS_CATEGORY           115.538956
DAYS_ENDDATE_FACT             115.315875
EXT_SOURCE_1                  114.839469
                                 ...
CNT_CREDIT_PROLONG             61.496987
CREDIT_DAY_OVERDUE             61.496987
DAYS_CREDIT                    61.496987
CREDIT_CURRENCY                61.496987
SK_ID_BUREAU                   61.496987
NAME_CASH_LOAN_PURPOSE         57.843459
NAME_YIELD_GROUP               52.872580
AMT_GOODS_PRICE_y              42.214100
ORGANIZATION_TYPE              38.957306
EXT_SOURCE_3                   25.614043
AMT_REQ_CREDIT_BUREAU_WEEK     16.514856
AMT_REQ_CREDIT_BUREAU_YEAR     16.514856
AMT_REQ_CREDIT_BUREAU_DAY      16.514856
AMT_REQ_CREDIT_BUREAU_HOUR     16.514856
AMT_REQ_CREDIT_BUREAU_MON      16.514856
AMT_REQ_CREDIT_BUREAU_QRT      16.514856
NAME_TYPE_SUITE_x               0.382100
DEF_60_CNT_SOCIAL_CIRCLE        0.343402
OBS_60_CNT_SOCIAL_CIRCLE        0.343402
```

```
OBS_30_CNT_SOCIAL_CIRCLE        0.343402
DEF_30_CNT_SOCIAL_CIRCLE        0.343402
EXT_SOURCE_2                    0.257552
AMT_GOODS_PRICE_x               0.119345
DAYS_FIRST_DUE                  0.056258
DAYS_LAST_DUE_1ST_VERSION       0.056258
DAYS_FIRST_DRAWING              0.056258
DAYS_LAST_DUE                   0.056258
DAYS_TERMINATION                0.056258
NFLAG_INSURED_ON_APPROVAL       0.056258
AMT_PAYMENT                     0.036421
Length: 100, dtype: float64
```

In [26]:

```
# saving column names in a variable
variables = merged_data.columns
drop_variables = [ ]
for i in variables:
    if missing[i]>=20:   #setting the threshold as 20%
        drop_variables.append(i)
print(drop_variables)
```

```
['OWN_CAR_AGE', 'OCCUPATION_TYPE', 'ORGANIZATION_TYPE', 'EXT_SOURCE_1', 'EXT_SO
URCE_3', 'APARTMENTS_AVG', 'BASEMENTAREA_AVG', 'YEARS_BEGINEXPLUATATION_AVG',
'YEARS_BUILD_AVG', 'COMMONAREA_AVG', 'ELEVATORS_AVG', 'ENTRANCES_AVG', 'FLOORSM
AX_AVG', 'FLOORSMIN_AVG', 'LANDAREA_AVG', 'LIVINGAPARTMENTS_AVG', 'LIVINGAREA_A
VG', 'NONLIVINGAPARTMENTS_AVG', 'NONLIVINGAREA_AVG', 'APARTMENTS_MODE', 'BASEME
NTAREA_MODE', 'YEARS_BEGINEXPLUATATION_MODE', 'YEARS_BUILD_MODE', 'COMMONAREA_M
ODE', 'ELEVATORS_MODE', 'ENTRANCES_MODE', 'FLOORSMAX_MODE', 'FLOORSMIN_MODE',
'LANDAREA_MODE', 'LIVINGAPARTMENTS_MODE', 'LIVINGAREA_MODE', 'NONLIVINGAPARTMEN
TS_MODE', 'NONLIVINGAREA_MODE', 'APARTMENTS_MEDI', 'BASEMENTAREA_MEDI', 'YEARS_
BEGINEXPLUATATION_MEDI', 'YEARS_BUILD_MEDI', 'COMMONAREA_MEDI', 'ELEVATORS_MED
I', 'ENTRANCES_MEDI', 'FLOORSMAX_MEDI', 'FLOORSMIN_MEDI', 'LANDAREA_MEDI', 'LIV
INGAPARTMENTS_MEDI', 'LIVINGAREA_MEDI', 'NONLIVINGAPARTMENTS_MEDI', 'NONLIVINGA
REA_MEDI', 'FONDKAPREMONT_MODE', 'HOUSETYPE_MODE', 'TOTALAREA_MODE', 'WALLSMATE
RIAL_MODE', 'EMERGENCYSTATE_MODE', 'SK_ID_BUREAU', 'CREDIT_ACTIVE', 'CREDIT_CUR
RENCY', 'DAYS_CREDIT', 'CREDIT_DAY_OVERDUE', 'DAYS_CREDIT_ENDDATE', 'DAYS_ENDDA
TE_FACT', 'AMT_CREDIT_MAX_OVERDUE', 'CNT_CREDIT_PROLONG', 'AMT_CREDIT_SUM', 'AM
T_CREDIT_SUM_DEBT', 'AMT_CREDIT_SUM_LIMIT', 'AMT_CREDIT_SUM_OVERDUE', 'CREDIT_T
YPE', 'DAYS_CREDIT_UPDATE', 'AMT_ANNUITY_y', 'AMT_DOWN_PAYMENT', 'AMT_GOODS_PRI
CE_y', 'RATE_DOWN_PAYMENT', 'RATE_INTEREST_PRIMARY', 'RATE_INTEREST_PRIVILEGE
D', 'NAME_CASH_LOAN_PURPOSE', 'NAME_PAYMENT_TYPE', 'NAME_TYPE_SUITE_y', 'NAME_G
OODS_CATEGORY', 'NAME_PRODUCT_TYPE', 'NAME_SELLER_INDUSTRY', 'NAME_YIELD_GROU
P']
```

In [27]:

```
# dropped categorical columns from the merged_data that have more than 20% missi
print(merged_data.shape)
#dropping  columns
df=merged_data.drop(drop_variables,axis=1)
len(drop_variables)
print(df.shape)
#  102 columns in final dataset after data cleaning and data imputation, its dro
```

```
(643379, 182)
(643379, 102)
```

In [28]:
```
new_variables=df.columns
print(new_variables)
```

```
Index(['SK_ID_CURR', 'TARGET', 'NAME_CONTRACT_TYPE_x', 'CODE_GENDER',
       'FLAG_OWN_CAR', 'FLAG_OWN_REALTY', 'CNT_CHILDREN', 'AMT_INCOME_TOTAL',
       'AMT_CREDIT_x', 'AMT_ANNUITY_x',
       ...
       'DAYS_LAST_DUE', 'DAYS_TERMINATION', 'NFLAG_INSURED_ON_APPROVAL',
       'SK_ID_CURR_y', 'NUM_INSTALMENT_VERSION', 'NUM_INSTALMENT_NUMBER',
       'DAYS_INSTALMENT', 'DAYS_ENTRY_PAYMENT', 'AMT_INSTALMENT',
       'AMT_PAYMENT'],
      dtype='object', length=102)
```

In [29]:
```
# Number of each type of column
df.dtypes.value_counts()
```

Out[29]:
```
int64      49
float64    34
object     19
dtype: int64
```

In [30]:
```
#Imputing null values on continuous variables with median
for col in df.columns:
        if df[col].dtype=='int64' or df[col].dtype=='float64':
             df[col].fillna(df[col].median(), inplace=True)
```

In [31]:
```
#list categorical variables missing values
cat_variables=[]
for col in df.columns:
        if df[col].dtype=='object':
            cat_variables.append(col)


print(cat_variables)
```

```
['NAME_CONTRACT_TYPE_x', 'CODE_GENDER', 'FLAG_OWN_CAR', 'FLAG_OWN_REALTY', 'NAM
E_TYPE_SUITE_x', 'NAME_INCOME_TYPE', 'NAME_EDUCATION_TYPE', 'NAME_FAMILY_STATU
S', 'NAME_HOUSING_TYPE', 'WEEKDAY_APPR_PROCESS_START_x', 'NAME_CONTRACT_TYPE_
y', 'WEEKDAY_APPR_PROCESS_START_y', 'FLAG_LAST_APPL_PER_CONTRACT', 'NAME_CONTRA
CT_STATUS', 'CODE_REJECT_REASON', 'NAME_CLIENT_TYPE', 'NAME_PORTFOLIO', 'CHANNE
L_TYPE', 'PRODUCT_COMBINATION']
```

In [32]:
```
# Building trmporary data frame for digging underlying issue to see we need to ad

venkat =pd.DataFrame(df[cat_variables])
```

In [33]:
```
venkat.head()
```

Out[33]:

	NAME_CONTRACT_TYPE_x	CODE_GENDER	FLAG_OWN_CAR	FLAG_OWN_REALTY	NAME_TYPI
0	Cash loans	M	N	Y	Unac
1	Cash loans	M	N	Y	Unac
2	Cash loans	M	N	Y	Unac
3	Cash loans	M	N	Y	Unac
4	Cash loans	M	N	Y	Unac

In [34]:
```
# checking the percentage of missing values in each variable
missing=df.isnull().sum()/len(train)*100
missing.sort_values(ascending=False).head(28)
```

Out[34]:
```
NAME_TYPE_SUITE_x              0.382100
NAME_CLIENT_TYPE               0.021463
CODE_GENDER                    0.005853
CODE_REJECT_REASON             0.001626
AMT_PAYMENT                    0.000000
REG_CITY_NOT_WORK_CITY         0.000000
CNT_FAM_MEMBERS                0.000000
REGION_RATING_CLIENT           0.000000
REGION_RATING_CLIENT_W_CITY    0.000000
WEEKDAY_APPR_PROCESS_START_x   0.000000
HOUR_APPR_PROCESS_START_x      0.000000
REG_REGION_NOT_LIVE_REGION     0.000000
REG_REGION_NOT_WORK_REGION     0.000000
LIVE_REGION_NOT_WORK_REGION    0.000000
REG_CITY_NOT_LIVE_CITY         0.000000
LIVE_CITY_NOT_WORK_CITY        0.000000
FLAG_PHONE                     0.000000
EXT_SOURCE_2                   0.000000
OBS_30_CNT_SOCIAL_CIRCLE       0.000000
DEF_30_CNT_SOCIAL_CIRCLE       0.000000
OBS_60_CNT_SOCIAL_CIRCLE       0.000000
DEF_60_CNT_SOCIAL_CIRCLE       0.000000
DAYS_LAST_PHONE_CHANGE         0.000000
FLAG_DOCUMENT_2                0.000000
FLAG_DOCUMENT_3                0.000000
FLAG_DOCUMENT_4                0.000000
FLAG_DOCUMENT_5                0.000000
FLAG_EMAIL                     0.000000
dtype: float64
```

In [35]:
```
df.shape     # 102 columns in final dataset after data cleaning and data imputat
```
Out[35]:
```
(643379, 102)
```

In [36]:
```
# resolving one hot label encoding issue in downstream process due to missing va
modDF = df.dropna(axis=0,how='any',inplace=False,subset=cat_variables)
```

In [37]:
```
modDF.shape
```
Out[37]:
```
(642115, 102)
```

In [38]:
```python
# checking latest data structure

list(modDF.columns)
```

Out[38]:
```
['SK_ID_CURR',
 'TARGET',
 'NAME_CONTRACT_TYPE_x',
 'CODE_GENDER',
 'FLAG_OWN_CAR',
 'FLAG_OWN_REALTY',
 'CNT_CHILDREN',
 'AMT_INCOME_TOTAL',
 'AMT_CREDIT_x',
 'AMT_ANNUITY_x',
 'AMT_GOODS_PRICE_x',
 'NAME_TYPE_SUITE_x',
 'NAME_INCOME_TYPE',
 'NAME_EDUCATION_TYPE',
 'NAME_FAMILY_STATUS',
 'NAME_HOUSING_TYPE',
 'REGION_POPULATION_RELATIVE',
 'DAYS_BIRTH',
 'DAYS_EMPLOYED',
 'DAYS_REGISTRATION',
 'DAYS_ID_PUBLISH',
 'FLAG_MOBIL',
 'FLAG_EMP_PHONE',
 'FLAG_WORK_PHONE',
 'FLAG_CONT_MOBILE',
 'FLAG_PHONE',
 'FLAG_EMAIL',
 'CNT_FAM_MEMBERS',
 'REGION_RATING_CLIENT',
 'REGION_RATING_CLIENT_W_CITY',
 'WEEKDAY_APPR_PROCESS_START_x',
 'HOUR_APPR_PROCESS_START_x',
 'REG_REGION_NOT_LIVE_REGION',
 'REG_REGION_NOT_WORK_REGION',
 'LIVE_REGION_NOT_WORK_REGION',
 'REG_CITY_NOT_LIVE_CITY',
 'REG_CITY_NOT_WORK_CITY',
 'LIVE_CITY_NOT_WORK_CITY',
 'EXT_SOURCE_2',
 'OBS_30_CNT_SOCIAL_CIRCLE',
 'DEF_30_CNT_SOCIAL_CIRCLE',
 'OBS_60_CNT_SOCIAL_CIRCLE',
 'DEF_60_CNT_SOCIAL_CIRCLE',
 'DAYS_LAST_PHONE_CHANGE',
 'FLAG_DOCUMENT_2',
 'FLAG_DOCUMENT_3',
 'FLAG_DOCUMENT_4',
 'FLAG_DOCUMENT_5',
 'FLAG_DOCUMENT_6',
 'FLAG_DOCUMENT_7',
 'FLAG_DOCUMENT_8',
 'FLAG_DOCUMENT_9',
 'FLAG_DOCUMENT_10',
```

```
 'FLAG_DOCUMENT_11',
 'FLAG_DOCUMENT_12',
 'FLAG_DOCUMENT_13',
 'FLAG_DOCUMENT_14',
 'FLAG_DOCUMENT_15',
 'FLAG_DOCUMENT_16',
 'FLAG_DOCUMENT_17',
 'FLAG_DOCUMENT_18',
 'FLAG_DOCUMENT_19',
 'FLAG_DOCUMENT_20',
 'FLAG_DOCUMENT_21',
 'AMT_REQ_CREDIT_BUREAU_HOUR',
 'AMT_REQ_CREDIT_BUREAU_DAY',
 'AMT_REQ_CREDIT_BUREAU_WEEK',
 'AMT_REQ_CREDIT_BUREAU_MON',
 'AMT_REQ_CREDIT_BUREAU_QRT',
 'AMT_REQ_CREDIT_BUREAU_YEAR',
 'SK_ID_PREV',
 'SK_ID_CURR_x',
 'NAME_CONTRACT_TYPE_y',
 'AMT_ANNUITY',
 'AMT_APPLICATION',
 'AMT_CREDIT_y',
 'WEEKDAY_APPR_PROCESS_START_y',
 'HOUR_APPR_PROCESS_START_y',
 'FLAG_LAST_APPL_PER_CONTRACT',
 'NFLAG_LAST_APPL_IN_DAY',
 'NAME_CONTRACT_STATUS',
 'DAYS_DECISION',
 'CODE_REJECT_REASON',
 'NAME_CLIENT_TYPE',
 'NAME_PORTFOLIO',
 'CHANNEL_TYPE',
 'SELLERPLACE_AREA',
 'CNT_PAYMENT',
 'PRODUCT_COMBINATION',
 'DAYS_FIRST_DRAWING',
 'DAYS_FIRST_DUE',
 'DAYS_LAST_DUE_1ST_VERSION',
 'DAYS_LAST_DUE',
 'DAYS_TERMINATION',
 'NFLAG_INSURED_ON_APPROVAL',
 'SK_ID_CURR_y',
 'NUM_INSTALMENT_VERSION',
 'NUM_INSTALMENT_NUMBER',
 'DAYS_INSTALMENT',
 'DAYS_ENTRY_PAYMENT',
 'AMT_INSTALMENT',
 'AMT_PAYMENT']
```

In [39]:
```
# Create a label encoder object
#Initially this code is failing due to NaN in some of the categorical variables,
le = LabelEncoder()
le_count = 0

# Iterate through the columns
for col in df:
    if modDF[col].dtype == 'object':
        # If 2 or fewer unique categories
        if len(list(modDF[col].unique())) <= 2:
            # Train on the training data
            le.fit(modDF[col])
            # Transform both training and testing data
            modDF[col] = le.transform(modDF[col])
           # test[col] = le.transform(test[col])

            # Keep track of how many columns were label encoded
            le_count += 1

print('%d columns were label encoded.' % le_count)
```

```
7 columns were label encoded.
```

In [40]:
```
# creating dummy variables on categorical columns

df = pd.get_dummies(modDF)

print('Training Features shape: ', df.shape)
```

```
Training Features shape:  (642115, 168)
```

In [41]:
```
df_main = df
```

In [42]:
```
# checking the percentage of missing values in each variable
missing=df.isnull().sum()/len(train)*100
missing.sort_values(ascending=False).head()
```

Out[42]:
```
PRODUCT_COMBINATION_POS others without interest    0.0
FLAG_DOCUMENT_16                                   0.0
AMT_REQ_CREDIT_BUREAU_WEEK                         0.0
AMT_REQ_CREDIT_BUREAU_DAY                          0.0
AMT_REQ_CREDIT_BUREAU_HOUR                         0.0
dtype: float64
```

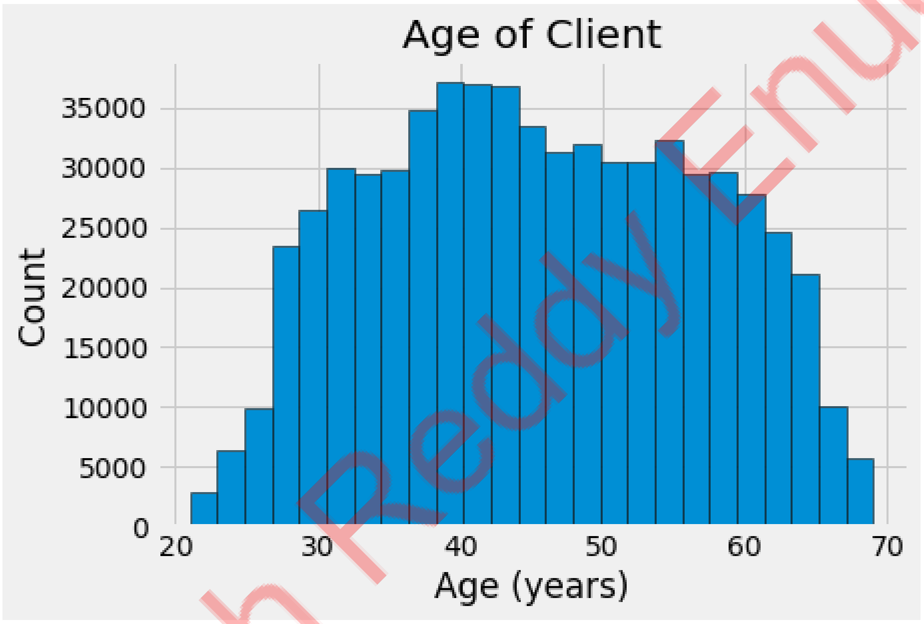
In [43]:
```
 #Find correlations with the target and sort
correlations = df.corr()['TARGET'].sort_values()

# Display correlations
print('Most Positive Correlations:\n', correlations.tail(15))
print('\nMost Negative Correlations:\n', correlations.head(15))
```

```
Most Positive Correlations:
 DAYS_ENTRY_PAYMENT                                 0.039345
DAYS_DECISION                                      0.039589
CODE_GENDER                                        0.043367
NAME_EDUCATION_TYPE_Secondary / secondary special  0.043602
NAME_INCOME_TYPE_Working                           0.051156
DAYS_LAST_PHONE_CHANGE                             0.051806
REGION_RATING_CLIENT                               0.056814
REGION_RATING_CLIENT_W_CITY                        0.058563
DAYS_BIRTH                                         0.066016
TARGET                                             1.000000
FLAG_MOBIL                                              NaN
FLAG_DOCUMENT_12                                        NaN
FLAG_LAST_APPL_PER_CONTRACT                             NaN
NAME_CONTRACT_STATUS                                    NaN
CODE_REJECT_REASON                                      NaN
Name: TARGET, dtype: float64

Most Negative Correlations:
 EXT_SOURCE_2                            -0.140517
NAME_EDUCATION_TYPE_Higher education   -0.045405
REGION_POPULATION_RELATIVE             -0.036175
NAME_INCOME_TYPE_Pensioner             -0.032143
NAME_CONTRACT_TYPE_x                   -0.031231
DAYS_EMPLOYED                          -0.031058
PRODUCT_COMBINATION_Cash X-Sell: low   -0.028543
NAME_HOUSING_TYPE_House / apartment    -0.028158
AMT_GOODS_PRICE_x                      -0.027192
DAYS_FIRST_DRAWING                     -0.026624
NAME_INCOME_TYPE_State servant         -0.023915
FLAG_OWN_CAR                           -0.022978
HOUR_APPR_PROCESS_START_y              -0.021324
HOUR_APPR_PROCESS_START_x              -0.020796
AMT_CREDIT_x                           -0.019056
Name: TARGET, dtype: float64
```

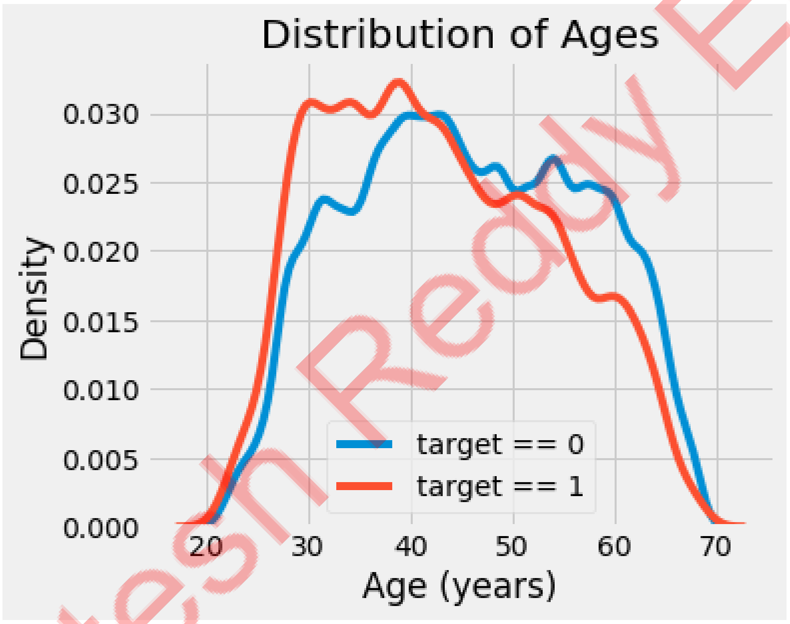
In [44]:
```
# Find the correlation of the positive days since birth and target
df['DAYS_BIRTH'] = abs(df['DAYS_BIRTH'])
df['DAYS_BIRTH'].corr(df['TARGET'])
```

Out[44]:
```
-0.066016087646578
```

In [45]:
```
# Set the style of plots
plt.style.use('fivethirtyeight')

# Plot the distribution of ages in years
plt.hist(df['DAYS_BIRTH'] / 365, edgecolor = 'k', bins = 25)
plt.title('Age of Client'); plt.xlabel('Age (years)'); plt.ylabel('Count');
```



In [46]:
```
plt.figure(figsize = (5, 4))

# KDE plot of loans that were repaid on time
sns.kdeplot(df.loc[df['TARGET'] == 0, 'DAYS_BIRTH'] / 365, label = 'target == 0'

# KDE plot of loans which were not repaid on time
sns.kdeplot(df.loc[df['TARGET'] == 1, 'DAYS_BIRTH'] / 365, label = 'target == 1'

# Labeling of plot
plt.xlabel('Age (years)'); plt.ylabel('Density'); plt.title('Distribution of Ages
```



In [47]:
```
# Age information into a separate dataframe
age_data = df[['TARGET', 'DAYS_BIRTH']]
age_data['YEARS_BIRTH'] = age_data['DAYS_BIRTH'] / 365

# Bin the age data
age_data['YEARS_BINNED'] = pd.cut(age_data['YEARS_BIRTH'], bins = np.linspace(20,
age_data.head(10)
```

Out[47]:

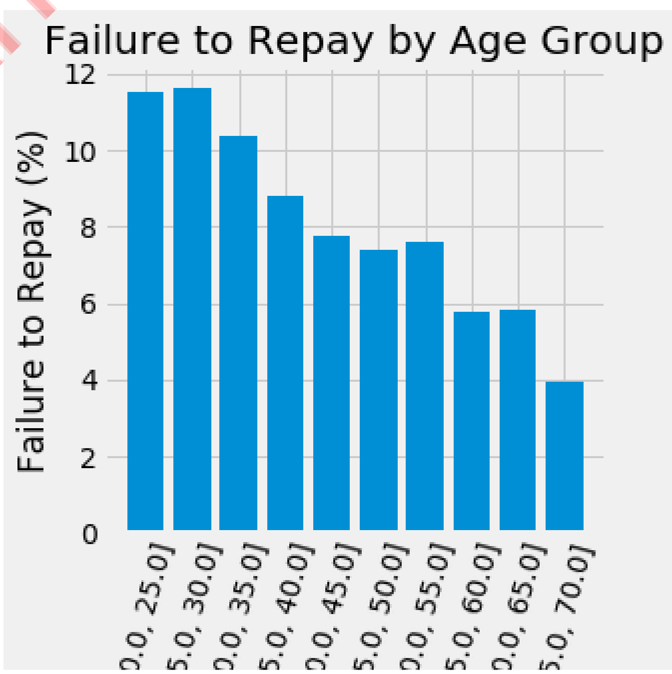
	TARGET	DAYS_BIRTH	YEARS_BIRTH	YEARS_BINNED
0	1	9461	25.920548	(25.0, 30.0]
1	1	9461	25.920548	(25.0, 30.0]
2	1	9461	25.920548	(25.0, 30.0]
3	1	9461	25.920548	(25.0, 30.0]
4	1	9461	25.920548	(25.0, 30.0]
5	1	9461	25.920548	(25.0, 30.0]
6	0	19932	54.608219	(50.0, 55.0]
7	0	19932	54.608219	(50.0, 55.0]
8	0	19932	54.608219	(50.0, 55.0]
9	0	19932	54.608219	(50.0, 55.0]

In [48]:
```
# Group by the bin and calculate averages
age_groups  = age_data.groupby('YEARS_BINNED').mean()
age_groups
```

Out[48]:

YEARS_BINNED	TARGET	DAYS_BIRTH	YEARS_BIRTH
(20.0, 25.0]	0.115054	8572.944128	23.487518
(25.0, 30.0]	0.116158	10224.909313	28.013450
(30.0, 35.0]	0.103747	11859.008857	32.490435
(35.0, 40.0]	0.088329	13733.699476	37.626574
(40.0, 45.0]	0.077784	15511.901263	42.498360
(45.0, 50.0]	0.074198	17334.578339	47.491995
(50.0, 55.0]	0.076119	19193.576445	52.585141
(55.0, 60.0]	0.057664	20993.721464	57.517045
(60.0, 65.0]	0.058324	22764.815840	62.369358
(65.0, 70.0]	0.039277	24290.660695	66.549755

In [49]:
```
plt.figure(figsize = (4,4))

# Graph the age bins and the average of the target as a bar plot
plt.bar(age_groups.index.astype(str), 100 * age_groups['TARGET'])

# Plot labeling
plt.xticks(rotation = 75); plt.xlabel('Age Group (years)'); plt.ylabel('Failure t
plt.title('Failure to Repay by Age Group');
```

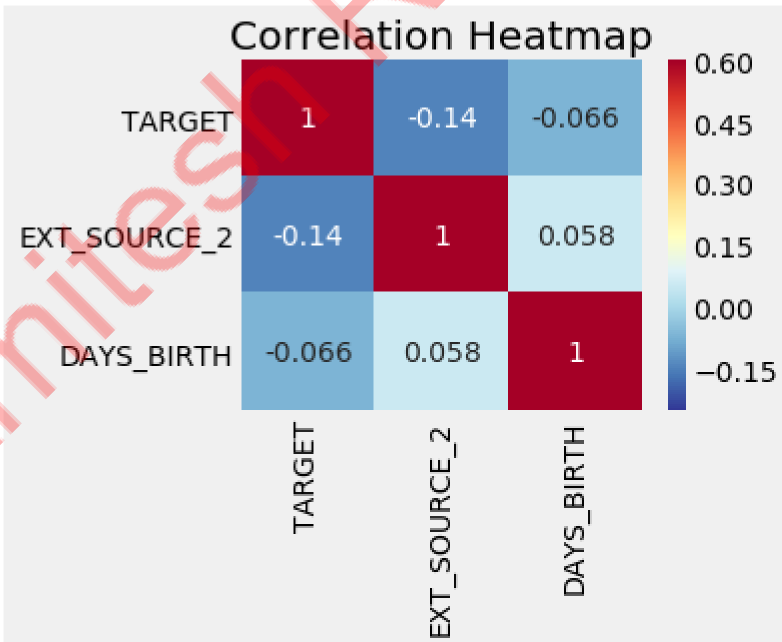


In [50]:
```
# Extract the EXT_SOURCE variables and show correlations  --rem   , 'EXT_SOURCE_1
ext_data = df[['TARGET', 'EXT_SOURCE_2', 'DAYS_BIRTH']]
ext_data_corrs = ext_data.corr()
ext_data_corrs
```

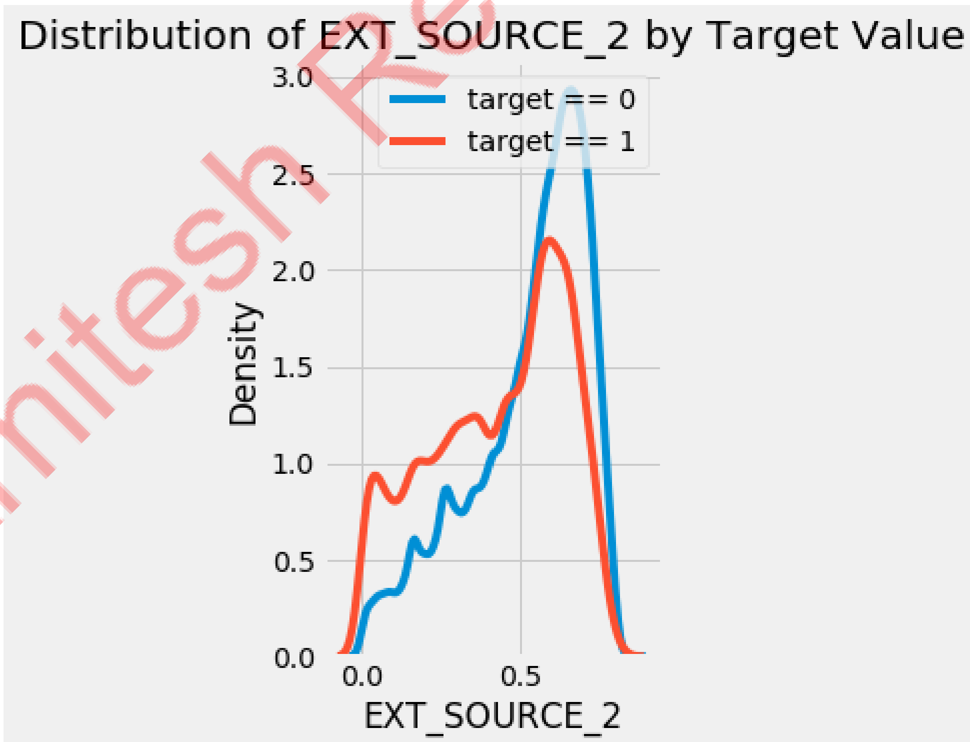
Out[50]:

	TARGET	EXT_SOURCE_2	DAYS_BIRTH
TARGET	1.000000	-0.140517	-0.066016
EXT_SOURCE_2	-0.140517	1.000000	0.058498
DAYS_BIRTH	-0.066016	0.058498	1.000000

In [51]:
```
plt.figure(figsize = (4, 3))

# Heatmap of correlations
sns.heatmap(ext_data_corrs, cmap = plt.cm.RdYlBu_r, vmin = -0.25, annot = True, v
plt.title('Correlation Heatmap');
```



In [52]:
```
plt.figure(figsize = (5,10))

# iterate through the sources  rm , 'EXT_SOURCE_1'
for i, source in enumerate([ 'EXT_SOURCE_2']):

    # create a new subplot for each source
    plt.subplot(2, 1, i + 1)
    # plot repaid loans
    sns.kdeplot(df.loc[df['TARGET'] == 0, source], label = 'target == 0')
    # plot loans that were not repaid
    sns.kdeplot(df.loc[df['TARGET'] == 1, source], label = 'target == 1')

    # Label the plots
    plt.title('Distribution of %s by Target Value' % source)
    plt.xlabel('%s' % source); plt.ylabel('Density');

plt.tight_layout(h_pad = 2.5)
```

In [53]:
```
data_clean=merged_data
```

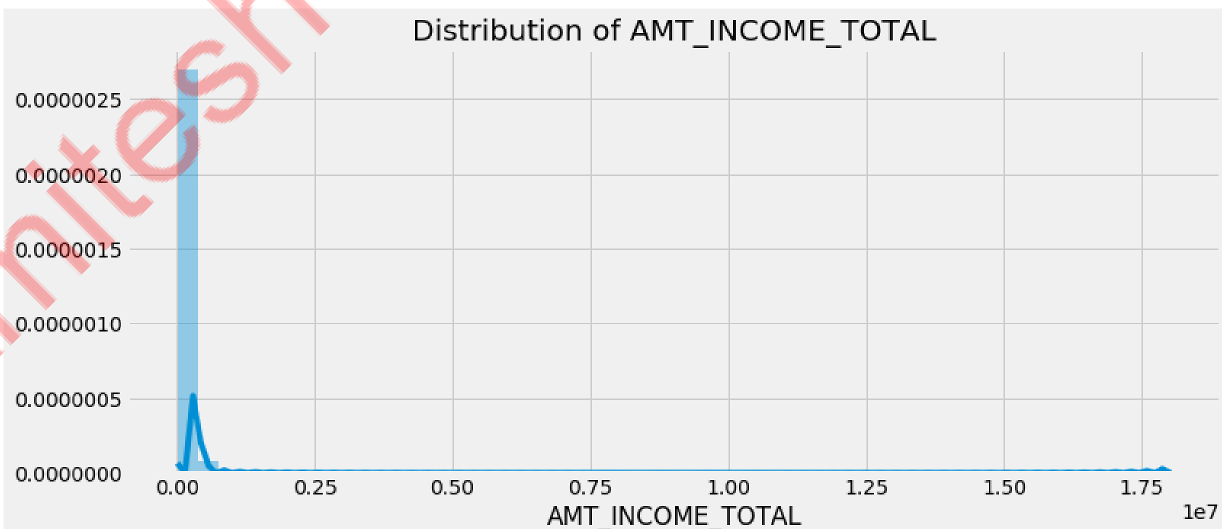
In [54]:
```
#visualization
data_clean.head()
```

Out[54]:

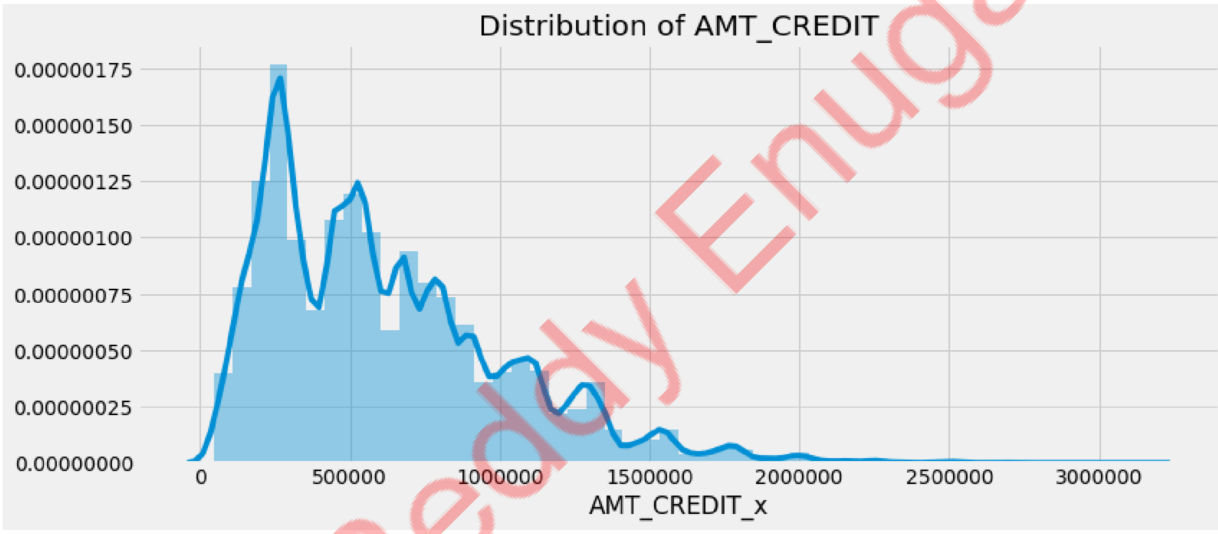
	SK_ID_CURR	TARGET	NAME_CONTRACT_TYPE_x	CODE_GENDER	FLAG_OWN_CAR	FLAG_O\
0	100002	1	Cash loans	M	N	
1	100002	1	Cash loans	M	N	
2	100002	1	Cash loans	M	N	
3	100002	1	Cash loans	M	N	
4	100002	1	Cash loans	M	N	

5 rows × 182 columns

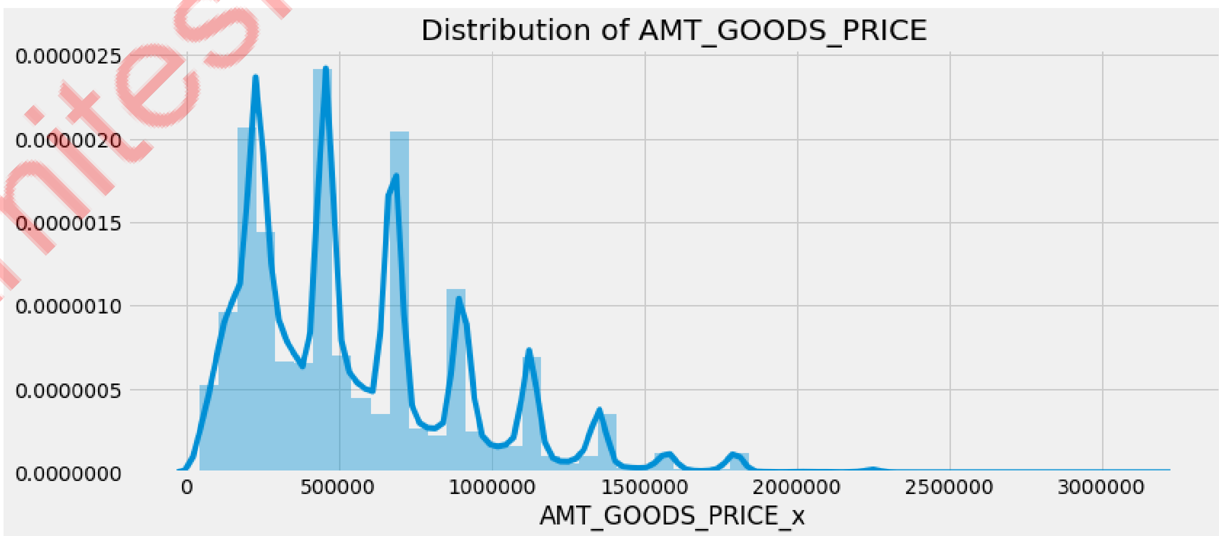
In [55]:
```
#Distribution of AMT_INCOME_TOTAL
plt.figure(figsize=(12,5))
plt.title("Distribution of AMT_INCOME_TOTAL")
ax = sns.distplot(data_clean["AMT_INCOME_TOTAL"].dropna())
```



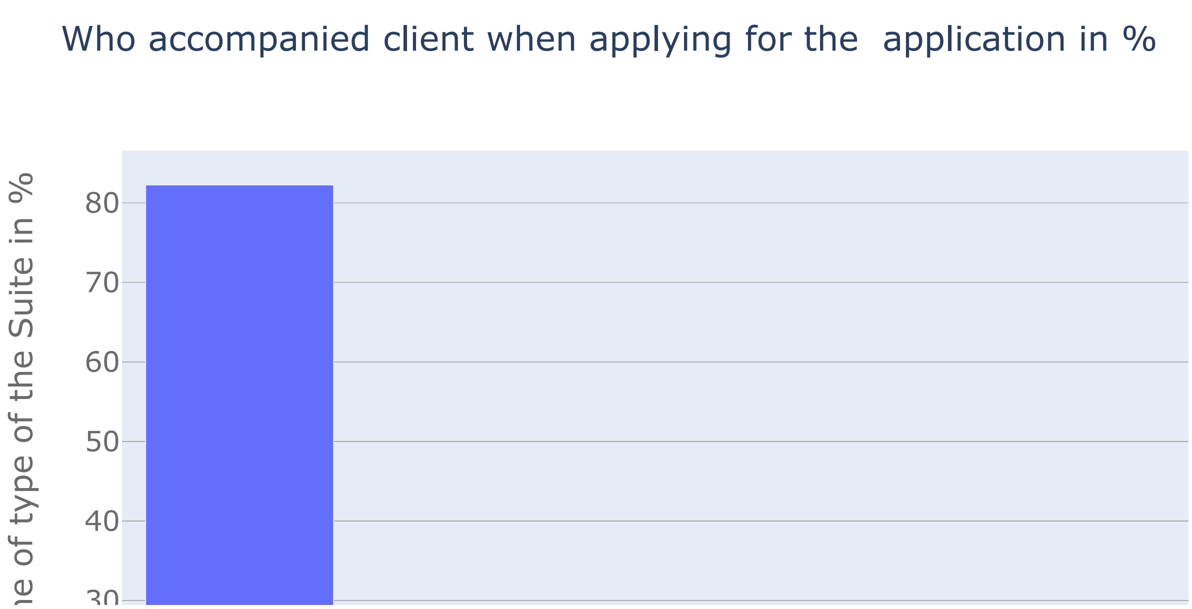
In [56]:
```
#Distribution of AMT_CREDIT
plt.figure(figsize=(12,5))
plt.title("Distribution of AMT_CREDIT")
ax = sns.distplot(data_clean["AMT_CREDIT_x"])
```



In [57]:
```
#Distribution of AMT_GOODS_PRICE
plt.figure(figsize=(12,5))
plt.title("Distribution of AMT_GOODS_PRICE")
ax = sns.distplot(data_clean["AMT_GOODS_PRICE_x"].dropna())
```



In [58]:
```
# Who accompanied client when applying for the application
temp = data_clean["NAME_TYPE_SUITE_x"].value_counts()
#print("Total number of states : ",len(temp))
trace = go.Bar(
    x = temp.index,
    y = (temp / temp.sum())*100,
)
data = [trace]
layout = go.Layout(
    title = "Who accompanied client when applying for the  application in % ",
    xaxis=dict(
        title='Name of type of the Suite',
        tickfont=dict(
            size=14,
            color='rgb(107, 107, 107)'
        )
    ),
    yaxis=dict(
        title='Count of Name of type of the Suite in %',
        titlefont=dict(
            size=16,
            color='rgb(107, 107, 107)'
        ),
        tickfont=dict(
            size=14,
            color='rgb(107, 107, 107)'
        )
)
)
fig = go.Figure(data=data, layout=layout)
py.iplot(fig, filename='schoolStateNames')
```

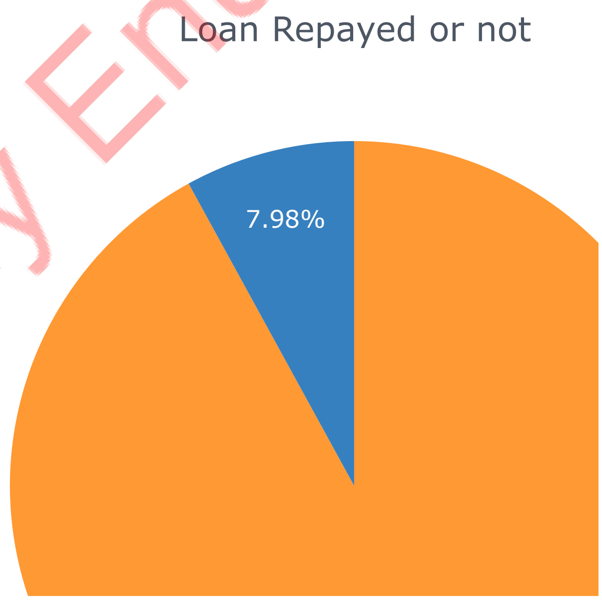
In [59]:
```
data_clean.head()
```

Out[59]:

	SK_ID_CURR	TARGET	NAME_CONTRACT_TYPE_x	CODE_GENDER	FLAG_OWN_CAR	FLAG_O\
0	100002	1	Cash loans	M	N	
1	100002	1	Cash loans	M	N	
2	100002	1	Cash loans	M	N	
3	100002	1	Cash loans	M	N	
4	100002	1	Cash loans	M	N	

5 rows × 182 columns

In [60]:
```
#Data is balanced or imbalanced
temp = data_clean["TARGET"].value_counts()
df = pd.DataFrame({'labels': temp.index,
                   'values': temp.values
                  })
df.iplot(kind='pie',labels='labels',values='values', title='Loan Repayed or not'
```



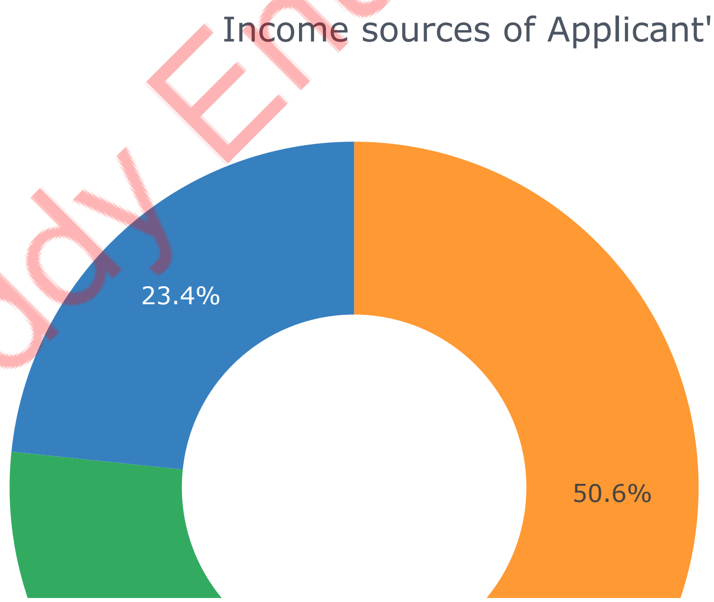
In [61]:
```
data_clean.head()
```

Out[61]:

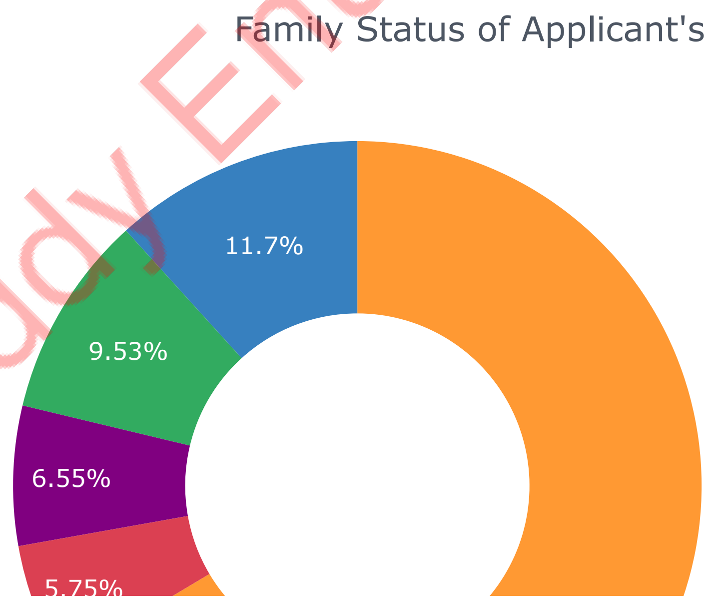
	SK_ID_CURR	TARGET	NAME_CONTRACT_TYPE_x	CODE_GENDER	FLAG_OWN_CAR	FLAG_OV
0	100002	1	Cash loans	M	N	
1	100002	1	Cash loans	M	N	
2	100002	1	Cash loans	M	N	
3	100002	1	Cash loans	M	N	
4	100002	1	Cash loans	M	N	

5 rows × 182 columns

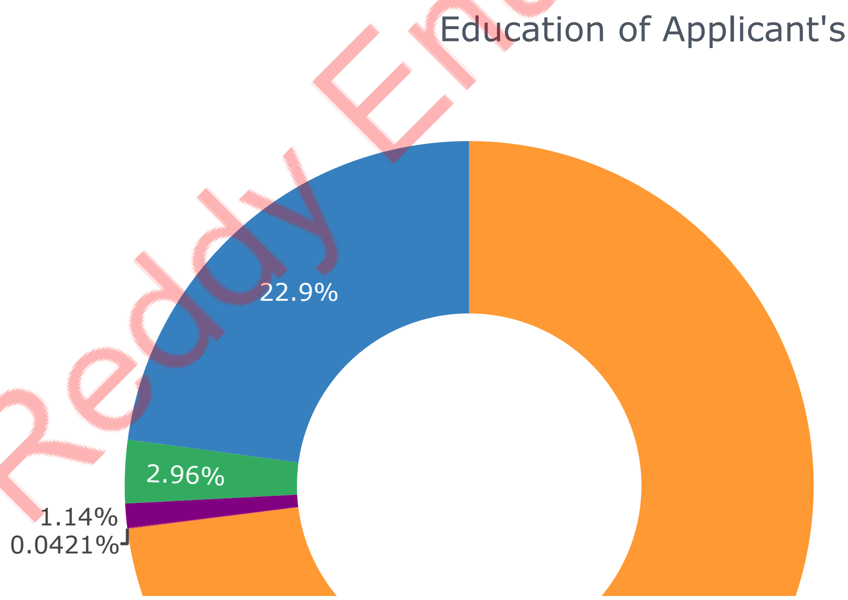
In [62]:
```
#Income sources of Applicant's who applied for Loan
temp = data_clean["NAME_INCOME_TYPE"].value_counts()
df = pd.DataFrame({'labels': temp.index,
                   'values': temp.values
                  })
df.iplot(kind='pie',labels='labels',values='values', title='Income sources of App
```



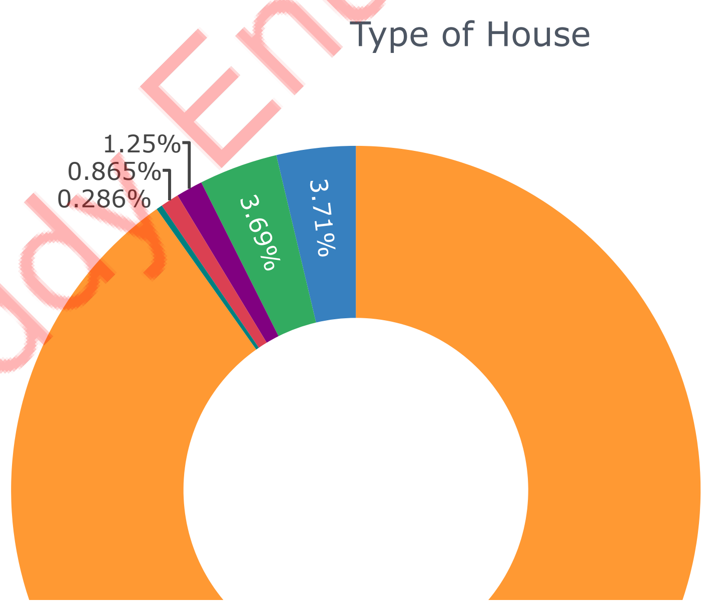
In [63]:
```
#Family Status of Applicant's who applied for Loan
temp = data_clean["NAME_FAMILY_STATUS"].value_counts()
df = pd.DataFrame({'labels': temp.index,
                   'values': temp.values
                  })
df.iplot(kind='pie',labels='labels',values='values', title='Family Status of Appl
```



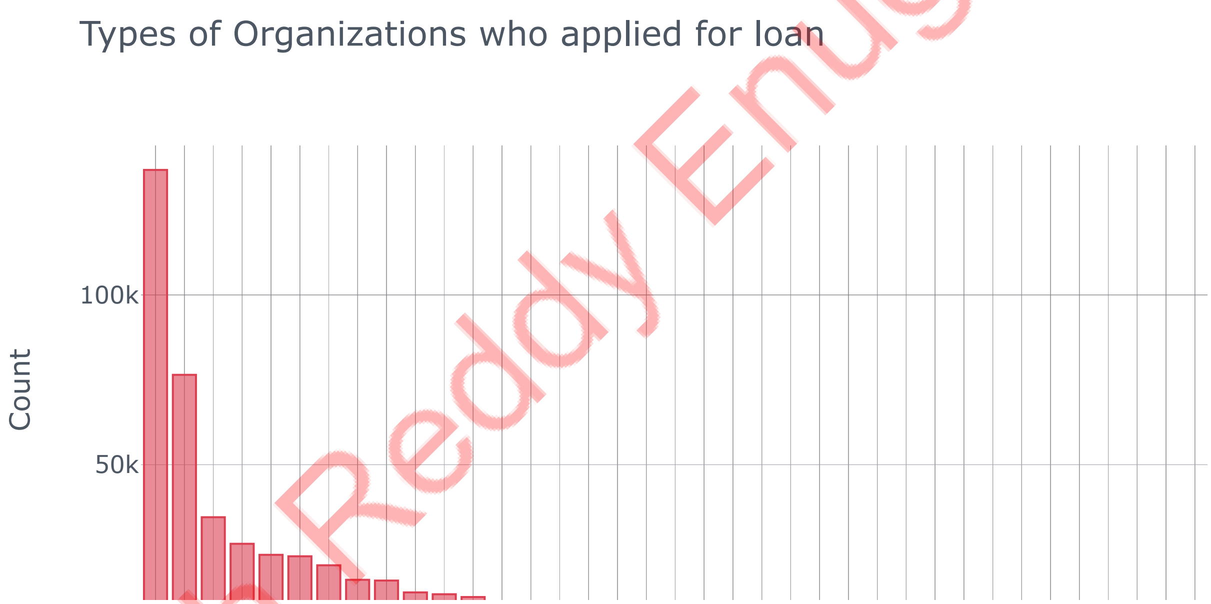
In [64]:
```
#Education of Applicant's who applied for loan¶
temp = data_clean["NAME_EDUCATION_TYPE"].value_counts()
df = pd.DataFrame({'labels': temp.index,
                   'values': temp.values
                  })
df.iplot(kind='pie',labels='labels',values='values', title='Education of Applica
```



In [65]:
```
#For which types of house higher applicant's applied for loan ?
temp = data_clean["NAME_HOUSING_TYPE"].value_counts()
df = pd.DataFrame({'labels': temp.index,
                   'values': temp.values
                  })
df.iplot(kind='pie',labels='labels',values='values', title='Type of House', hole
```



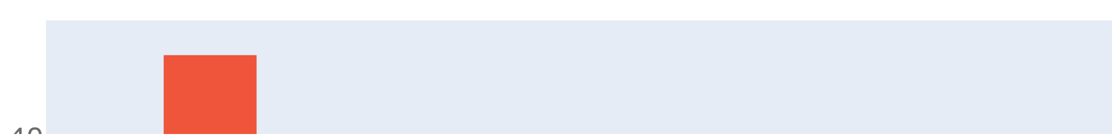
In [66]:
```
#Types of Organizations who applied for loan
temp =data_clean["ORGANIZATION_TYPE"].value_counts()
temp.iplot(kind='bar', xTitle = 'Organization Name', yTitle = "Count", title = ''
```



In [67]:

```
#Exploration in terms of loan is repayed or not
#Income sources of Applicant's in terms of loan is repayed or not in
temp = data_clean["NAME_INCOME_TYPE"].value_counts()
#print(temp.values)
temp_y0 = []
temp_y1 = []
for val in temp.index:
    temp_y1.append(np.sum(data_clean["TARGET"][data_clean["NAME_INCOME_TYPE"]==va
    temp_y0.append(np.sum(data_clean["TARGET"][data_clean["NAME_INCOME_TYPE"]==va
trace1 = go.Bar(
    x = temp.index,
    y = (temp_y1 / temp.sum()) * 100,
    name='YES'
)
trace2 = go.Bar(
    x = temp.index,
    y = (temp_y0 / temp.sum()) * 100,
    name='NO'
)

data = [trace1, trace2]
layout = go.Layout(
    title = "Income sources of Applicant's in terms of loan is repayed or not  in
    #barmode='stack',
    width = 1000,
    xaxis=dict(
        title='Income source',
        tickfont=dict(
            size=14,
            color='rgb(107, 107, 107)'
        )
    ),
    yaxis=dict(
        title='Count in %',
        titlefont=dict(
            size=16,
            color='rgb(107, 107, 107)'
        ),
        tickfont=dict(
            size=14,
            color='rgb(107, 107, 107)'
        )
)
)

fig = go.Figure(data=data, layout=layout)
iplot(fig)
```

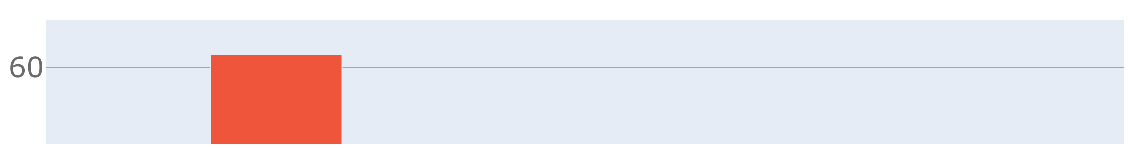
Income sources of Applicant's in terms of loan is repayed or not in

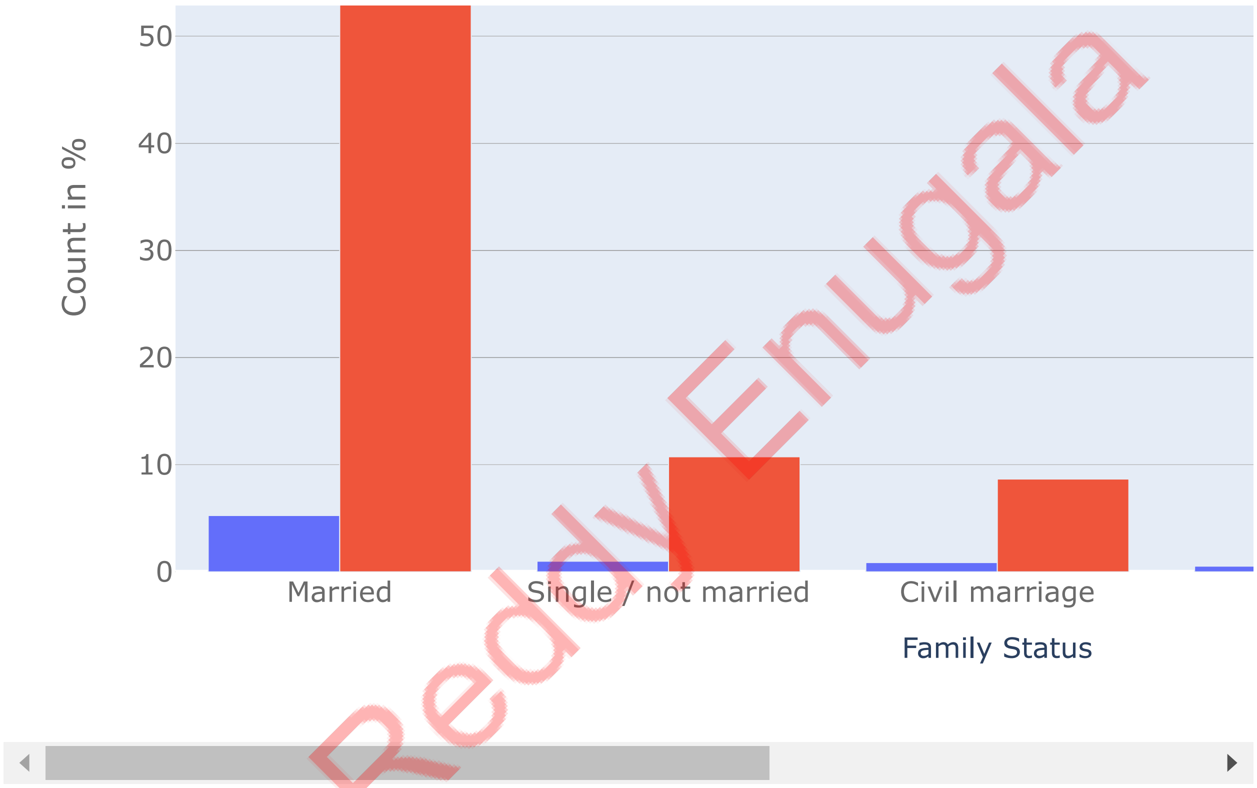



40
30
20
10
0
Count in %
Working
Commercial associate
Pensioner
State servant
Unempl
Income source

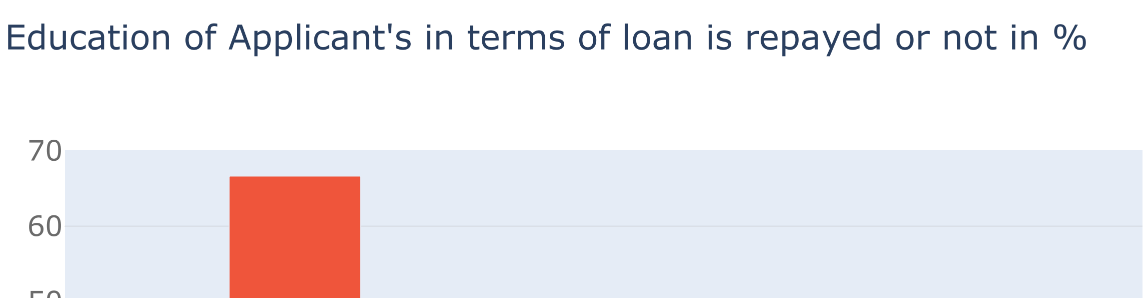
In [68]:
```
#Family Status of Applicant's in terms of loan is repayed or not in %
temp = data_clean["NAME_FAMILY_STATUS"].value_counts()
#print(temp.values)
temp_y0 = []
temp_y1 = []
for val in temp.index:
    temp_y1.append(np.sum(data_clean["TARGET"][data_clean["NAME_FAMILY_STATUS"]=
    temp_y0.append(np.sum(data_clean["TARGET"][data_clean["NAME_FAMILY_STATUS"]=
trace1 = go.Bar(
    x = temp.index,
    y = (temp_y1 / temp.sum()) * 100,
    name='YES'
)
trace2 = go.Bar(
    x = temp.index,
    y = (temp_y0 / temp.sum()) * 100,
    name='NO'
)

data = [trace1, trace2]
layout = go.Layout(
    title = "Family Status of Applicant's in terms of loan is repayed or not in %
    #barmode='stack',
    width = 1000,
    xaxis=dict(
        title='Family Status',
        tickfont=dict(
            size=14,
            color='rgb(107, 107, 107)'
        )
    ),
    yaxis=dict(
        title='Count in %',
        titlefont=dict(
            size=16,
            color='rgb(107, 107, 107)'
        ),
        tickfont=dict(
            size=14,
            color='rgb(107, 107, 107)'
        )
)
)

fig = go.Figure(data=data, layout=layout)
iplot(fig)
```

Family Status of Applicant's in terms of loan is repayed or not in %



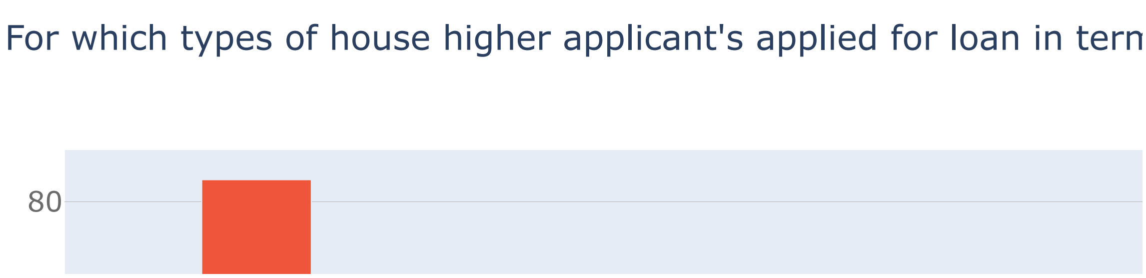

Count in %
50
40
30
20
10
0
Married
Single / not married
Civil marriage
Family Status

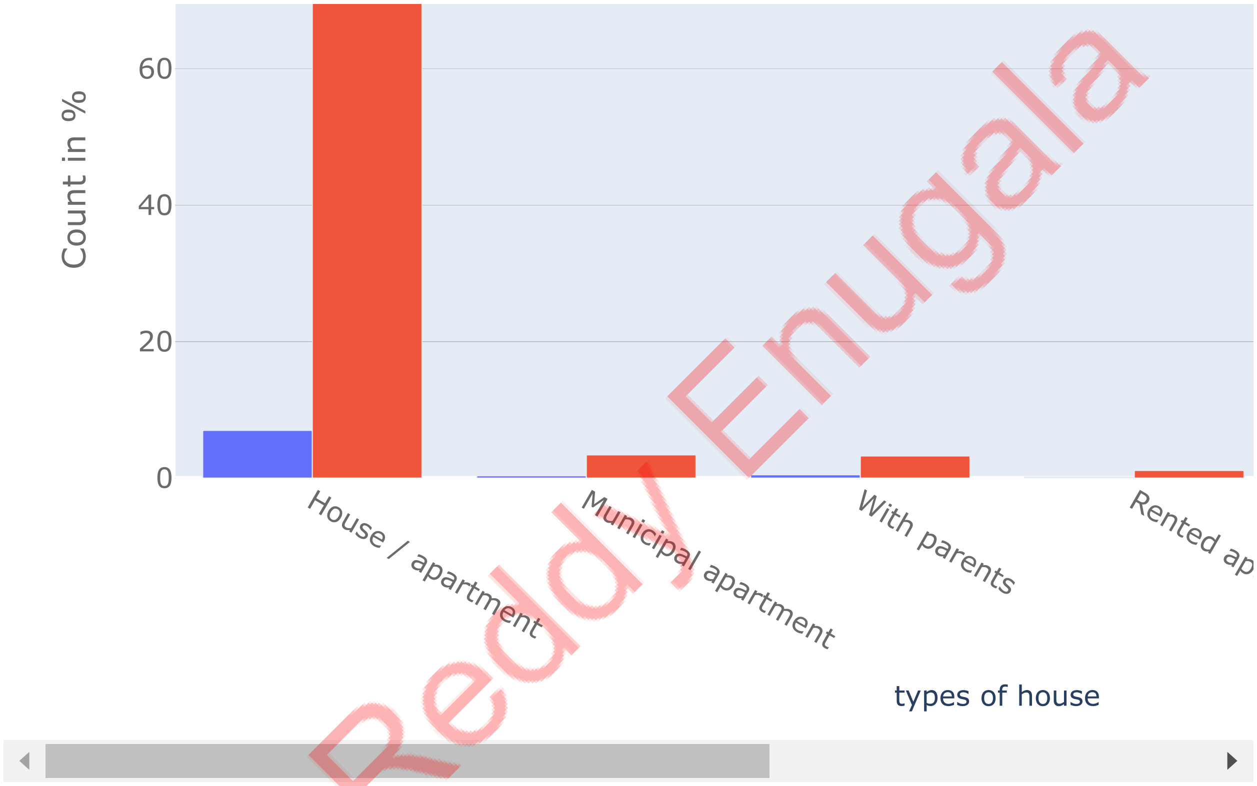
In [69]:
```python
#Education of Applicant's in terms of Loan is repayed or not in %
temp = data_clean["NAME_EDUCATION_TYPE"].value_counts()
#print(temp.values)
temp_y0 = []
temp_y1 = []
for val in temp.index:
    temp_y1.append(np.sum(data_clean["TARGET"][data_clean["NAME_EDUCATION_TYPE"]
    temp_y0.append(np.sum(data_clean["TARGET"][data_clean["NAME_EDUCATION_TYPE"]
trace1 = go.Bar(
    x = temp.index,
    y = (temp_y1 / temp.sum()) * 100,
    name='YES'
)
trace2 = go.Bar(
    x = temp.index,
    y = (temp_y0 / temp.sum()) * 100,
    name='NO'
)

data = [trace1, trace2]
layout = go.Layout(
    title = "Education of Applicant's in terms of loan is repayed or not in %",
    #barmode='stack',
    width = 1000,
    xaxis=dict(
        title='Education of Applicant\'s',
        tickfont=dict(
            size=14,
            color='rgb(107, 107, 107)'
        )
    ),
    yaxis=dict(
        title='Count in %',
        titlefont=dict(
            size=16,
            color='rgb(107, 107, 107)'
        ),
        tickfont=dict(
            size=14,
            color='rgb(107, 107, 107)'
        )
)
)

fig = go.Figure(data=data, layout=layout)
iplot(fig)
```




Count in %
50
40
30
20
10
0
Secondary / secondary special
Higher education
Incomplete higher
Education of Applicant's

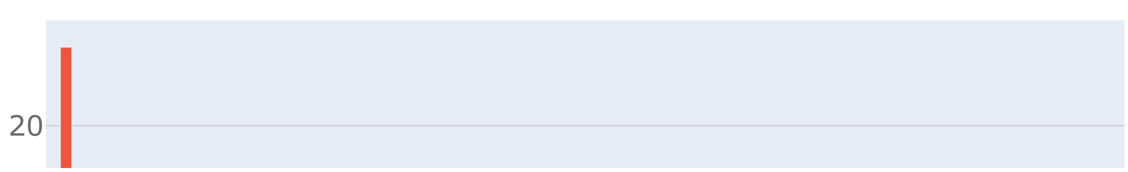
In [70]:
```
#For which types of house higher applicant's applied for loan in terms of loan is
temp = data_clean["NAME_HOUSING_TYPE"].value_counts()
#print(temp.values)
temp_y0 = []
temp_y1 = []
for val in temp.index:
    temp_y1.append(np.sum(data_clean["TARGET"][data_clean["NAME_HOUSING_TYPE"]==v
    temp_y0.append(np.sum(data_clean["TARGET"][data_clean["NAME_HOUSING_TYPE"]==v
trace1 = go.Bar(
    x = temp.index,
    y = (temp_y1 / temp.sum()) * 100,
    name='YES'
)
trace2 = go.Bar(
    x = temp.index,
    y = (temp_y0 / temp.sum()) * 100,
    name='NO'
)

data = [trace1, trace2]
layout = go.Layout(
    title = "For which types of house higher applicant's applied for loan in term
    #barmode='stack',
    width = 1000,
    xaxis=dict(
        title='types of house',
        tickfont=dict(
            size=14,
            color='rgb(107, 107, 107)'
        )
    ),
    yaxis=dict(
        title='Count in %',
        titlefont=dict(
            size=16,
            color='rgb(107, 107, 107)'
        ),
        tickfont=dict(
            size=14,
            color='rgb(107, 107, 107)'
        )
)
)

fig = go.Figure(data=data, layout=layout)
iplot(fig)
```

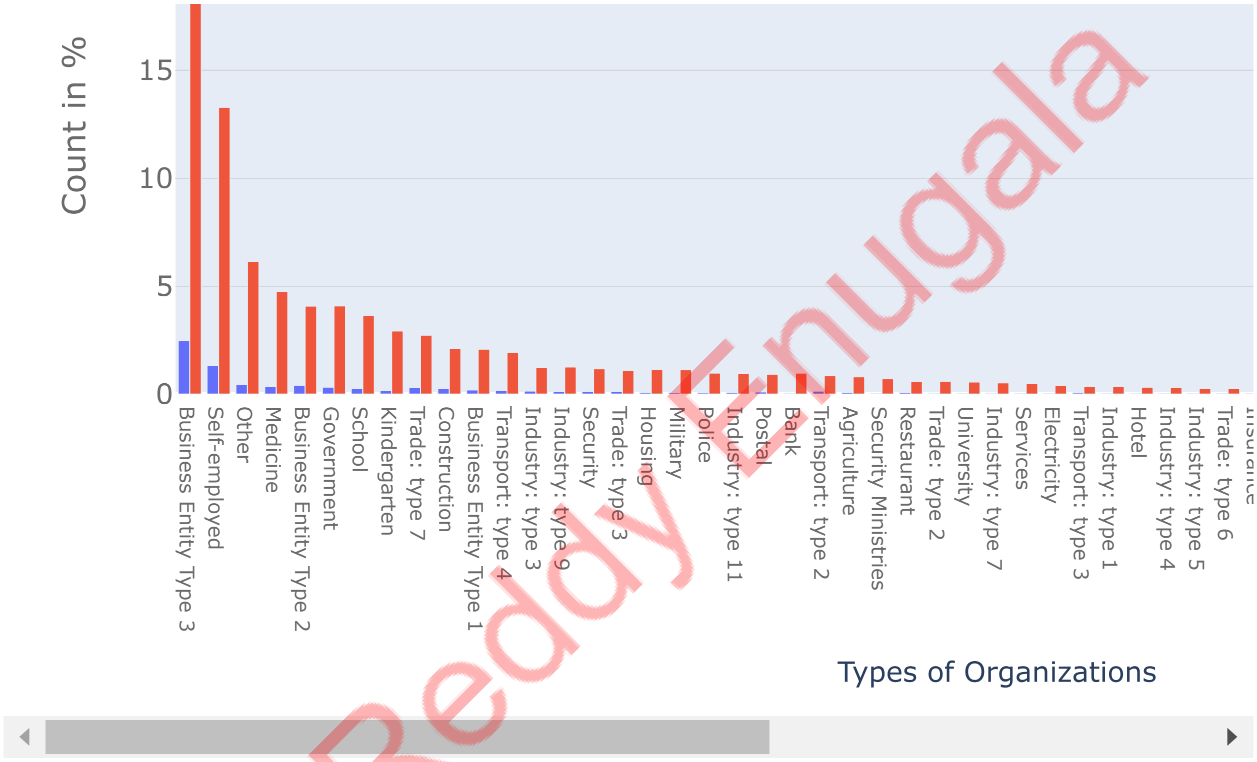



Count in %
60
40
20
0
House / apartment
Municipal apartment
With parents
Rented ap
types of house

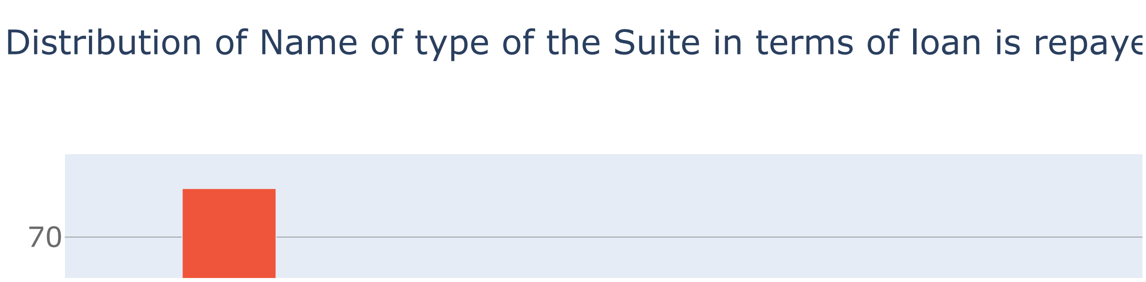
In [71]:
```
#Types of Organizations in terms of loan is repayed or not in %
temp = data_clean["ORGANIZATION_TYPE"].value_counts()
#print(temp.values)
temp_y0 = []
temp_y1 = []
for val in temp.index:
    temp_y1.append(np.sum(data_clean["TARGET"][data_clean["ORGANIZATION_TYPE"]==
    temp_y0.append(np.sum(data_clean["TARGET"][data_clean["ORGANIZATION_TYPE"]==
trace1 = go.Bar(
    x = temp.index,
    y = (temp_y1 / temp.sum()) * 100,
    name='YES'
)
trace2 = go.Bar(
    x = temp.index,
    y = (temp_y0 / temp.sum()) * 100,
    name='NO'
)

data = [trace1, trace2]
layout = go.Layout(
    title = "Types of Organizations in terms of loan is repayed or not in %",
    #barmode='stack',
    width = 1000,
    xaxis=dict(
        title='Types of Organizations',
        tickfont=dict(
            size=10,
            color='rgb(107, 107, 107)'
        )
    ),
    yaxis=dict(
        title='Count in %',
        titlefont=dict(
            size=16,
            color='rgb(107, 107, 107)'
        ),
        tickfont=dict(
            size=14,
            color='rgb(107, 107, 107)'
        )
)
)

fig = go.Figure(data=data, layout=layout)
iplot(fig)
```

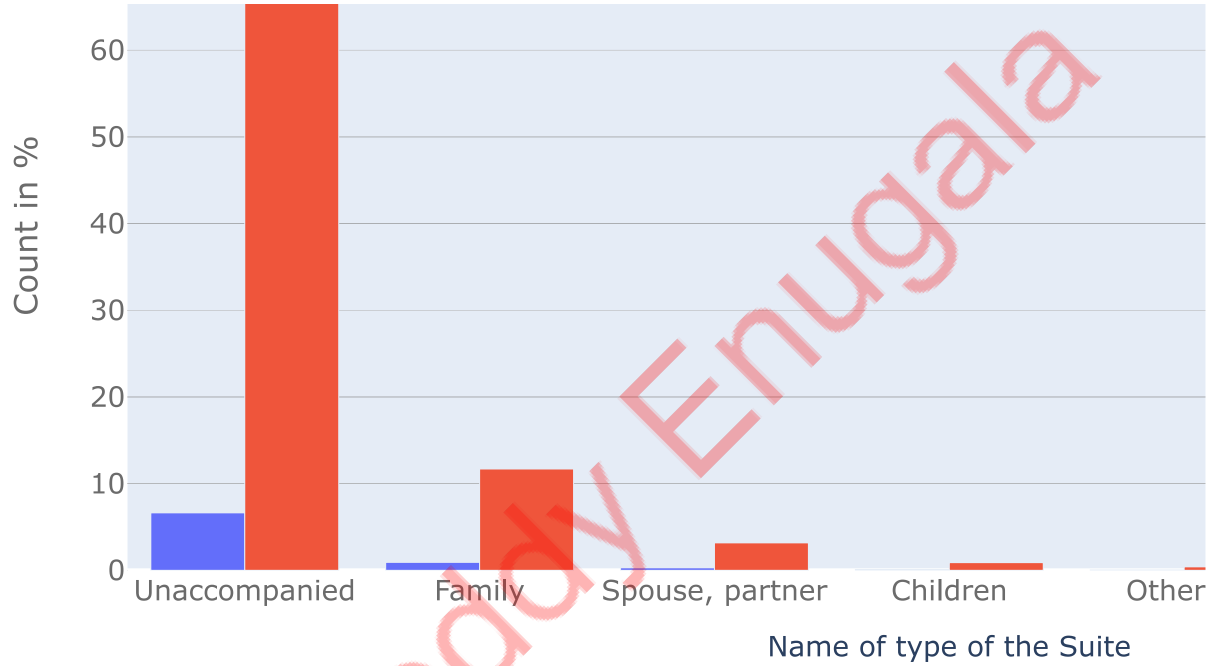

Types of Organizations in terms of loan is repayed or not in %




Count in %
15
10
5
0
Business Entity Type 3
Self-employed
Other
Medicine
Business Entity Type 2
Government
School
Kindergarten
Trade: type 7
Construction
Business Entity Type 1
Transport: type 4
Industry: type 3
Industry: type 9
Security
Trade: type 3
Housing
Military
Police
Industry: type 11
Postal
Bank
Transport: type 2
Agriculture
Security Ministries
Restaurant
Trade: type 2
University
Industry: type 7
Services
Electricity
Transport: type 3
Industry: type 1
Hotel
Industry: type 4
Industry: type 5
Trade: type 6
Types of Organizations

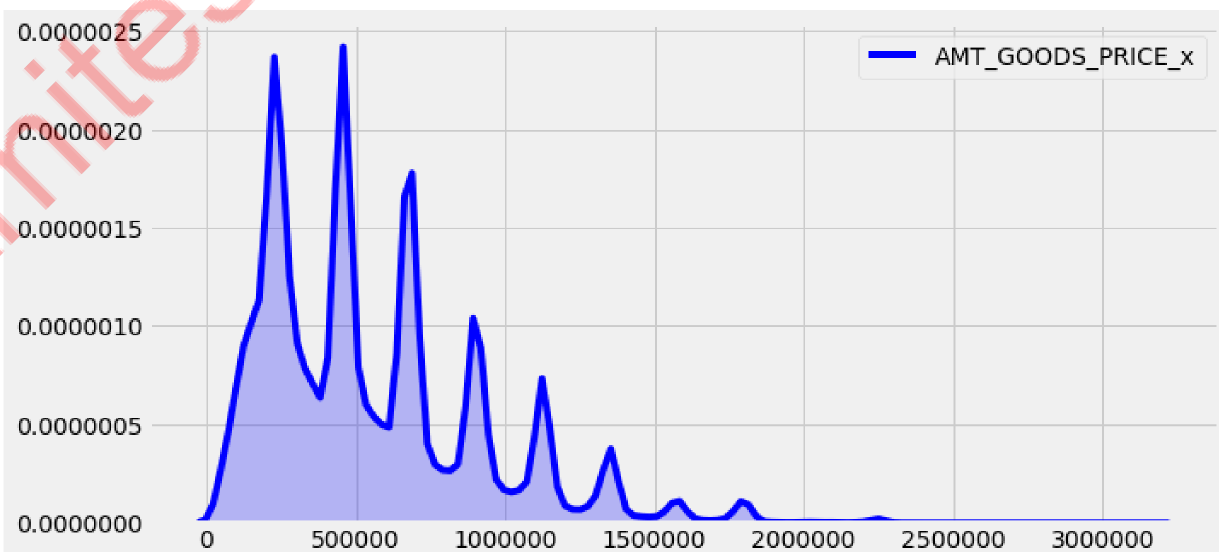
In [72]:
```
# Distribution of Name of type of the Suite in terms of loan is repayed or not i
temp = data_clean["NAME_TYPE_SUITE_x"].value_counts()
#print(temp.values)
temp_y0 = []
temp_y1 = []
for val in temp.index:
    temp_y1.append(np.sum(data_clean["TARGET"][data_clean["NAME_TYPE_SUITE_x"]==
    temp_y0.append(np.sum(data_clean["TARGET"][data_clean["NAME_TYPE_SUITE_x"]==
trace1 = go.Bar(
    x = temp.index,
    y = (temp_y1 / temp.sum()) * 100,
    name='YES'
)
trace2 = go.Bar(
    x = temp.index,
    y = (temp_y0 / temp.sum()) * 100,
    name='NO'
)

data = [trace1, trace2]
layout = go.Layout(
    title = "Distribution of Name of type of the Suite in terms of loan is repaye
    #barmode='stack',
    width = 1000,
    xaxis=dict(
        title='Name of type of the Suite',
        tickfont=dict(
            size=14,
            color='rgb(107, 107, 107)'
        )
    ),
    yaxis=dict(
        title='Count in %',
        titlefont=dict(
            size=16,
            color='rgb(107, 107, 107)'
        ),
        tickfont=dict(
            size=14,
            color='rgb(107, 107, 107)'
        )
)
)

fig = go.Figure(data=data, layout=layout)
iplot(fig)
```





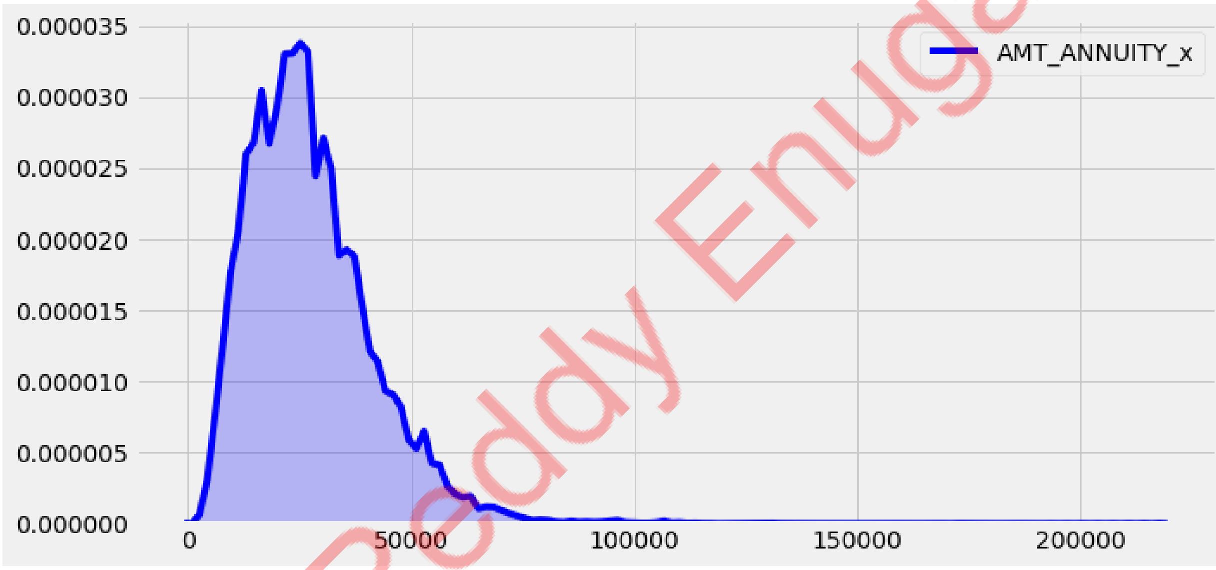
In [73]:
```
plt.figure(figsize=(10,5))
sns.kdeplot(modDF['AMT_GOODS_PRICE_x'], shade=True, color="b")
```

Out[73]: <matplotlib.axes._subplots.AxesSubplot at 0x27e821a9be0>



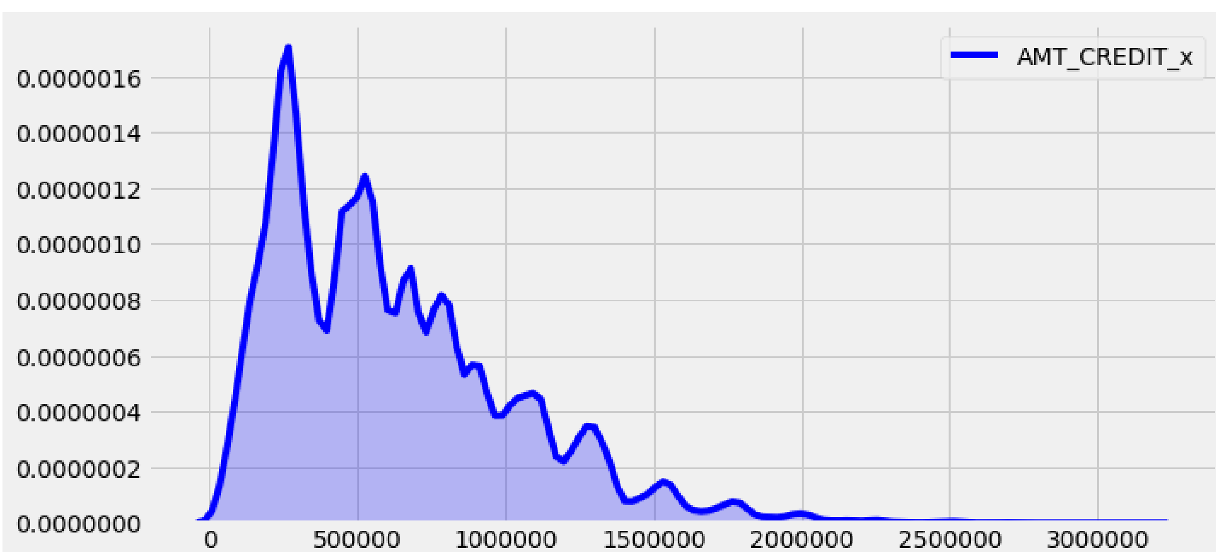
In [74]:
```
plt.figure(figsize=(10,5))
sns.kdeplot(modDF['AMT_ANNUITY_x'], shade=True, color="b")
```

Out[74]: <matplotlib.axes._subplots.AxesSubplot at 0x27e7fec5d30>



In [75]:
```
plt.figure(figsize=(10,5))
sns.kdeplot(modDF['AMT_CREDIT_x'], shade=True, color="b")
```

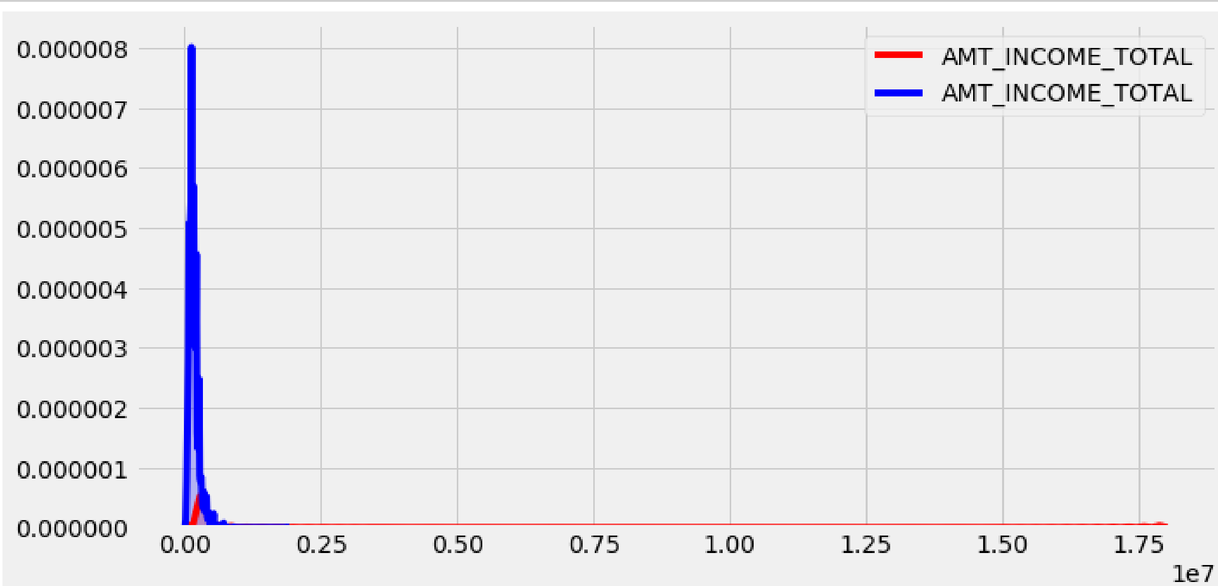
Out[75]: <matplotlib.axes._subplots.AxesSubplot at 0x27e97a00b00>



In [76]:
```
plt.figure(figsize=(10,5))
sns.kdeplot(modDF['AMT_INCOME_TOTAL'], shade=True, color="b")
```

Out[76]: <matplotlib.axes._subplots.AxesSubplot at 0x27e811d3a20>

In [77]:
```
#finding out that people with default(Target = 1) and non default(Target = 0) has
plt.figure(figsize=(10,5))
s1 = modDF[modDF['TARGET'] == 0]
s2 = modDF[modDF['TARGET'] == 1]

p1=sns.kdeplot(s1['AMT_INCOME_TOTAL'], shade=True, color="r")
p1=sns.kdeplot(s2['AMT_INCOME_TOTAL'], shade=True, color="b")
```



In [78]:
```
#as data is not normal we can use non-parametric tests
#H0 : Same Distribution
#Ha : Not same Distribution

from scipy.stats import mannwhitneyu

stat, p = mannwhitneyu(s1['AMT_INCOME_TOTAL'], s2['AMT_INCOME_TOTAL'])
print('Statistics=%.3f, p=%.3f' % (stat, p))
# interpret
alpha = 0.05
if p > alpha:
    print('Same distribution (fail to reject H0)')
else:
    print('Different distribution (reject H0)')

#this test statistically says that there is difference in
#total income for people with approved and rejected credit cards
```

```
Statistics=14905569801.500, p=0.000
Different distribution (reject H0)
```

In [79]:
```
from statsmodels.formula.api import ols
import statsmodels.api as sm
results = ols('AMT_CREDIT_x ~ C(NAME_INCOME_TYPE)', data=modDF).fit()
print(results.summary())
aov_table = sm.stats.anova_lm(results, typ=2)
print(aov_table)

# we can reject null hypothesis and say that total amount credit given is not sar
# similarly we can try it for NAME_EDUCATION_TYPE instead of NAME_INCOME_TYPE
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:           AMT_CREDIT_x   R-squared:                       0.013
Model:                            OLS   Adj. R-squared:                  0.013
Method:                 Least Squares   F-statistic:                     1433.
Date:                Sun, 01 Dec 2019   Prob (F-statistic):               0.00
Time:                        01:55:41   Log-Likelihood:            -9.1835e+06
No. Observations:              642115   AIC:                         1.837e+07
Df Residuals:                  642108   BIC:                         1.837e+07
Df Model:                           6
Covariance Type:            nonrobust
================================================================================
===========================
                                              coef    std err          t      P>
|t|      [0.025      0.975]
--------------------------------------------------------------------------------
---------------------------
Intercept                               6.812e+05   1016.414    670.178      0.
000    6.79e+05    6.83e+05
C(NAME_INCOME_TYPE)[T.Maternity leave] -6.347e+04   7.72e+04     -0.822      0.
411   -2.15e+05    8.78e+04
C(NAME_INCOME_TYPE)[T.Pensioner]        -1.26e+05   1525.975    -82.572      0.
000   -1.29e+05   -1.23e+05
C(NAME_INCOME_TYPE)[T.State servant]    2595.4858   2070.609      1.253      0.
210   -1462.841    6653.812
C(NAME_INCOME_TYPE)[T.Student]         -1.468e+05   1.97e+05     -0.746      0.
456   -5.32e+05    2.39e+05
C(NAME_INCOME_TYPE)[T.Unemployed]       3.582e+04   7.19e+04      0.498      0.
618   -1.05e+05    1.77e+05
C(NAME_INCOME_TYPE)[T.Working]         -7.631e+04   1228.682    -62.104      0.
000   -7.87e+04   -7.39e+04
==============================================================================
Omnibus:                    90658.643   Durbin-Watson:                   0.322
Prob(Omnibus):                  0.000   Jarque-Bera (JB):           138493.575
Skew:                           1.018   Prob(JB):                         0.00
Kurtosis:                       4.016   Cond. No.                         464.
==============================================================================

Warnings:
[1] Standard Errors assume that the covariance matrix of the errors is correctl
y specified.
                           sum_sq        df            F  PR(>F)
C(NAME_INCOME_TYPE)  1.331408e+15       6.0  1432.732882     0.0
Residual             9.944957e+16  642108.0          NaN     NaN
```

In [80]:

```
from scipy import stats
m = modDF['NAME_INCOME_TYPE'].unique()
for i in m:
    for j in m:
        if j<i:
            stat, p = stats.ttest_ind(modDF[modDF['NAME_INCOME_TYPE'] == i]['AMT_
            if p < 0.05:
                print(f'between group {i} and {j}')
                print('Statistics=%.3f, p=%.3f' % (stat, p))
# if p > 0.05 it says that we cannot reject null hypothesis and the AMT_CREDIT_x
```

```
between group Working and State servant
Statistics=-37.600, p=0.000
between group Working and Pensioner
Statistics=38.349, p=0.000
between group Working and Commercial associate
Statistics=-60.424, p=0.000
between group Working and Unemployed
Statistics=-2.162, p=0.039
between group State servant and Pensioner
Statistics=56.432, p=0.000
between group Pensioner and Commercial associate
Statistics=-81.600, p=0.000
between group Unemployed and Pensioner
Statistics=3.120, p=0.004
```

In [81]:
```
cat_col = [c for i, c in enumerate(modDF.columns) if modDF.dtypes[i] in [np.obje
e = cat_col
df2 = modDF.sample(frac = 0.01)
d = e
for i in d:
    for j in d:
        if i != j:
            crosstab = pd.crosstab(df2[i], df2[j])
            X_squared,p,dfdm,array = stats.chi2_contingency(crosstab)
            if p < 0.05:
                print(f'Collinearity exists between {i} and {j} with p-value {p}
    d.remove(i)
#As there are many cases where p<0.05 where we cant reject null hypothesis and co
#that multi collinearity exists in this dataset
```

```
Collinearity exists between NAME_TYPE_SUITE_x and NAME_INCOME_TYPE with p-value
0.0007702161963116009 and F-value 52.05222692299388
Collinearity exists between NAME_TYPE_SUITE_x and NAME_EDUCATION_TYPE with p-va
lue 0.03579698451234921 and F-value 37.86441999754221
Collinearity exists between NAME_TYPE_SUITE_x and NAME_FAMILY_STATUS with p-val
ue 2.9034992978826276e-15 and F-value 122.93429201919143
Collinearity exists between NAME_TYPE_SUITE_x and PRODUCT_COMBINATION with p-va
lue 2.5401227603464696e-08 and F-value 183.10500447748558
Collinearity exists between NAME_EDUCATION_TYPE and NAME_INCOME_TYPE with p-val
ue 1.0193080841425923e-37 and F-value 219.18841246892515
Collinearity exists between NAME_EDUCATION_TYPE and NAME_FAMILY_STATUS with p-v
alue 1.4077937320003786e-05 and F-value 51.32341874236803
Collinearity exists between NAME_EDUCATION_TYPE and NAME_HOUSING_TYPE with p-va
lue 4.311130852632195e-07 and F-value 67.69588373539352
Collinearity exists between NAME_EDUCATION_TYPE and NAME_CLIENT_TYPE with p-val
ue 0.013291281755638307 and F-value 19.309078787050176
Collinearity exists between NAME_EDUCATION_TYPE and NAME_PORTFOLIO with p-value
0.0010426995131564035 and F-value 32.79293071567131
Collinearity exists between NAME_EDUCATION_TYPE and CHANNEL_TYPE with p-value
0.0008281076838211928 and F-value 57.55423935831748
Collinearity exists between NAME_EDUCATION_TYPE and PRODUCT_COMBINATION with p-
value 7.791732938267256e-05 and F-value 110.52033031617327
Collinearity exists between NAME_HOUSING_TYPE and NAME_INCOME_TYPE with p-value
1.4867193336007554e-12 and F-value 99.59754892925947
Collinearity exists between NAME_HOUSING_TYPE and NAME_FAMILY_STATUS with p-val
ue 4.81651925193653e-20 and F-value 140.11349882496728
Collinearity exists between NAME_HOUSING_TYPE and NAME_CONTRACT_TYPE_y with p-v
alue 0.00187865083824763 and F-value 27.891819898597646
Collinearity exists between NAME_HOUSING_TYPE and NAME_PORTFOLIO with p-value
0.01860164092182557 and F-value 28.507154971813478
Collinearity exists between NAME_HOUSING_TYPE and PRODUCT_COMBINATION with p-va
lue 8.705139739251828e-13 and F-value 196.17683128356904
Collinearity exists between NAME_CONTRACT_TYPE_y and NAME_INCOME_TYPE with p-va
lue 1.4428985743887694e-24 and F-value 131.4103125340399
Collinearity exists between NAME_CONTRACT_TYPE_y and NAME_FAMILY_STATUS with p-
value 0.00020755629196381062 and F-value 30.044825752887938
Collinearity exists between NAME_CONTRACT_TYPE_y and WEEKDAY_APPR_PROCESS_START
_y with p-value 7.806233793227065e-47 and F-value 251.16851956801796
Collinearity exists between NAME_CONTRACT_TYPE_y and NAME_CLIENT_TYPE with p-va
lue 1.0239881932342834e-300 and F-value 1394.6009447760844
Collinearity exists between NAME_CONTRACT_TYPE_y and NAME_PORTFOLIO with p-valu
e 0.0 and F-value 12842.0
```

```
Collinearity exists between NAME_CONTRACT_TYPE_y and CHANNEL_TYPE with p-value
0.0 and F-value 4924.19918415451
Collinearity exists between NAME_CONTRACT_TYPE_y and PRODUCT_COMBINATION with p
-value 0.0 and F-value 12842.0
Collinearity exists between NAME_CLIENT_TYPE and NAME_FAMILY_STATUS with p-valu
e 0.0006964396763041452 and F-value 27.03779758205485
Collinearity exists between NAME_CLIENT_TYPE and WEEKDAY_APPR_PROCESS_START_y w
ith p-value 2.899245076768612e-07 and F-value 53.856057087638455
Collinearity exists between NAME_CLIENT_TYPE and NAME_PORTFOLIO with p-value 2.
780612088885714e-298 and F-value 1395.1051247080159
Collinearity exists between NAME_CLIENT_TYPE and CHANNEL_TYPE with p-value 4.09
298539214967e-174 and F-value 858.0893936975092
Collinearity exists between NAME_CLIENT_TYPE and PRODUCT_COMBINATION with p-val
ue 0.0 and F-value 1616.011771486401
Collinearity exists between CHANNEL_TYPE and NAME_INCOME_TYPE with p-value 6.37
1067287514089e-09 and F-value 93.1357501255334
Collinearity exists between CHANNEL_TYPE and WEEKDAY_APPR_PROCESS_START_y with
p-value 3.1114291955299435e-24 and F-value 210.1950554386279
Collinearity exists between CHANNEL_TYPE and NAME_PORTFOLIO with p-value 0.0 an
d F-value 11342.68630452074
Collinearity exists between CHANNEL_TYPE and PRODUCT_COMBINATION with p-value
0.0 and F-value 7352.615646711538
```

In [82]:

```
# Import your necessary dependencies
from sklearn.feature_selection import RFE
from sklearn.linear_model import LogisticRegression
```

In [83]:
```
# Feature extraction
model = LogisticRegression(solver='lbfgs')
rfe = RFE(model, 20)
df1 = df_main.drop(columns=['TARGET','SK_ID_CURR','SK_ID_CURR_x','SK_ID_CURR_x',
fit = rfe.fit(df1, df_main.TARGET)
print("Num Features: %s" % (fit.n_features_))
print("Selected Features: %s" % (fit.support_))
print("Feature Ranking: %s" % (fit.ranking_))
```

```
Num Features: 20
Selected Features: [False False False False False  True  True  True  True False
True False
  True  True False False False False False False False False False False
 False False False False False False False False False False False  True
 False False False False False False False False False False False False
 False False False False False False False False False False False False
 False False  True False  True False False False False  True False False
 False  True  True  True  True  True False False False False  True  True
  True  True False False False False False False False False False False
 False False False False False False False False False False False False
 False False False False False False False False False False False False
 False False False False False False False False False False False False
 False False False False False False False False False False False False
 False False False False False False False False False False False False
 False False False False False False False False]
Feature Ranking: [ 32  18  22  71  37   1   1   1   1  96   1   3   1   1 106 1
30  34 111
  49 109  31  14  13   9 140  69  62  38  25  43  15  11  21  10  27   1
 121  24 129 115 101 131  36 124 136 122 145  97 103 126  87 120  99 123
 119 138 128  93  82  30  53  12   1   4   1   8 144 100 143   1 142   2
   7   1   1   1   1   1  73   5  42   6   1   1   1   1  84  58 107 116
 110 102  65  23 113  98  29 139 125  19 118  17  91 135  16  57  67 104
  83  60 127  28  80  85  74  39  77  64  46  78  68  56  86  55  50 134
  79  75  88  72  89  70  52  20  45  26 133 137  54  47  41 141 112  44
  51  35  61  90  94  95  48 108  92  40  33 105  81  59  63  76  66 117
 132 114]
```

In [84]:
```
rank = fit.ranking_.tolist()
column = df1.columns.tolist()
z = []
for i in rank:
    if i == 1:
        a = rank.index(i)
        rank[a] = 0
        m = column[a]
        z.append(m)
z
```

Out[84]:
```
['AMT_INCOME_TOTAL',
 'AMT_CREDIT_x',
 'AMT_ANNUITY_x',
 'AMT_GOODS_PRICE_x',
 'DAYS_BIRTH',
 'DAYS_REGISTRATION',
 'DAYS_ID_PUBLISH',
 'DAYS_LAST_PHONE_CHANGE',
 'AMT_ANNUITY',
 'AMT_CREDIT_y',
 'DAYS_DECISION',
 'DAYS_FIRST_DRAWING',
 'DAYS_FIRST_DUE',
 'DAYS_LAST_DUE_1ST_VERSION',
 'DAYS_LAST_DUE',
 'DAYS_TERMINATION',
 'DAYS_INSTALMENT',
 'DAYS_ENTRY_PAYMENT',
 'AMT_INSTALMENT',
 'AMT_PAYMENT']
```

In [85]:
```
#Feature Selection
X_file = df_main[z]
y_file = df_main['TARGET']
df_main.columns.tolist()
```

Out[85]:
```
['SK_ID_CURR',
 'TARGET',
 'NAME_CONTRACT_TYPE_x',
 'CODE_GENDER',
 'FLAG_OWN_CAR',
 'FLAG_OWN_REALTY',
 'CNT_CHILDREN',
 'AMT_INCOME_TOTAL',
 'AMT_CREDIT_x',
 'AMT_ANNUITY_x',
 'AMT_GOODS_PRICE_x',
 'REGION_POPULATION_RELATIVE',
 'DAYS_BIRTH',
 'DAYS_EMPLOYED',
 'DAYS_REGISTRATION',
 'DAYS_ID_PUBLISH',
 'FLAG_MOBIL',
 'FLAG_EMP_PHONE',
 'FLAG_WORK_PHONE',
```

In [86]:
```
from sklearn.model_selection import train_test_split

# Partition it randomly into train and test set using a 70/30 split.
X_train,X_test,y_train,y_test=train_test_split(X_file,y_file,test_size=0.4,randor
```

In [87]:
```
def Report(y_test,y_pred):
   print(confusion_matrix(y_test, y_pred))
   print(classification_report(y_test, y_pred))
```

In [88]:
```
#Logistic
log_reg = LogisticRegression(C = 0.0001)
result = log_reg.fit(X_train, y_train)
y_pred = np.where(log_reg.predict_proba(X_test)[:, 1]> 0.1, 1, 0)

Report(y_test, y_pred)
```

```
[[184764  51579]
 [ 12752   7751]]
              precision    recall  f1-score   support

           0       0.94      0.78      0.85    236343
           1       0.13      0.38      0.19     20503

    accuracy                           0.75    256846
   macro avg       0.53      0.58      0.52    256846
weighted avg       0.87      0.75      0.80    256846
```

In [99]:
```
#parameter selection for Decision Tree and Random Forest
from sklearn.model_selection import GridSearchCV

def dt_param_selection(X, y, nfolds):
    opt_tree = DecisionTreeClassifier(random_state=42)
    param_DT = {"max_depth": range(1,120),
           "min_samples_split": range(2,3,4),
           "max_leaf_nodes": range(2,5)}
    grid_tree = GridSearchCV(opt_tree,param_DT,cv=nfolds)
    grid_tree.fit(X ,y)
    grid_tree.best_params_
    return grid_tree.best_params_

dt_param_selection(X_train, y_train, 5)
```

Out[99]:
```
{'max_depth': 1, 'max_leaf_nodes': 2, 'min_samples_split': 2}
```

In [98]:
```
# Create Decision Tree classifer object
from sklearn.tree import DecisionTreeClassifier # Import Decision Tree Classifie
from sklearn import metrics #Import scikit-learn metrics module for accuracy calc

clf = DecisionTreeClassifier(max_depth=1, max_leaf_nodes=2, min_samples_split=2,

# Train Decision Tree Classifer
clf = clf.fit(X_train,y_train)
y_pred3 = np.where(clf.predict_proba(X_test)[:, 1] > 0.1, 1, 0)
Report(y_test,y_pred3)
```

```
[[233800   2543]
 [  2254  18249]]
              precision    recall  f1-score   support

           0       0.99      0.99      0.99    236343
           1       0.88      0.89      0.88     20503

    accuracy                           0.98    256846
   macro avg       0.93      0.94      0.94    256846
weighted avg       0.98      0.98      0.98    256846
```

In [89]:
```
# Random Forest
features = list(X_train.columns)
from sklearn.ensemble import RandomForestClassifier
random_forest = RandomForestClassifier(max_depth=1, max_leaf_nodes=2, min_sample
random_forest.fit(X_train, y_train)
y_pred1 = np.where(random_forest.predict_proba(X_test)[:, 1] > 0.1, 1, 0)
Report(y_test, y_pred1)
```

```
[Parallel(n_jobs=-1)]: Using backend ThreadingBackend with 12 concurrent worker
s.
[Parallel(n_jobs=-1)]: Done  26 tasks      | elapsed:   11.1s
[Parallel(n_jobs=-1)]: Done 100 out of 100 | elapsed:   32.9s finished
[Parallel(n_jobs=12)]: Using backend ThreadingBackend with 12 concurrent worker
s.
[Parallel(n_jobs=12)]: Done  26 tasks      | elapsed:    0.4s
[Parallel(n_jobs=12)]: Done 100 out of 100 | elapsed:    1.4s finished

[[225446  10897]
 [  1035  19468]]
              precision    recall  f1-score   support

           0       1.00      0.95      0.97    236343
           1       0.64      0.95      0.77     20503

    accuracy                           0.95    256846
   macro avg       0.82      0.95      0.87    256846
weighted avg       0.97      0.95      0.96    256846
```

In [92]:
```
#KNN
from sklearn.neighbors import KNeighborsClassifier
classifier = KNeighborsClassifier(n_neighbors=3)
cls = classifier.fit(X_train, y_train)
y_pred4 = np.where(cls.predict_proba(X_test)[:, 1] > 0.1, 1, 0)
Report(y_test,y_pred4)
```

```
[[226538   9805]
 [  2065  18438]]
              precision    recall  f1-score   support

           0       0.99      0.96      0.97    236343
           1       0.65      0.90      0.76     20503

    accuracy                           0.95    256846
   macro avg       0.82      0.93      0.87    256846
weighted avg       0.96      0.95      0.96    256846
```

In [94]:
```
from xgboost.sklearn import XGBClassifier
from xgboost.sklearn import XGBRegressor
```

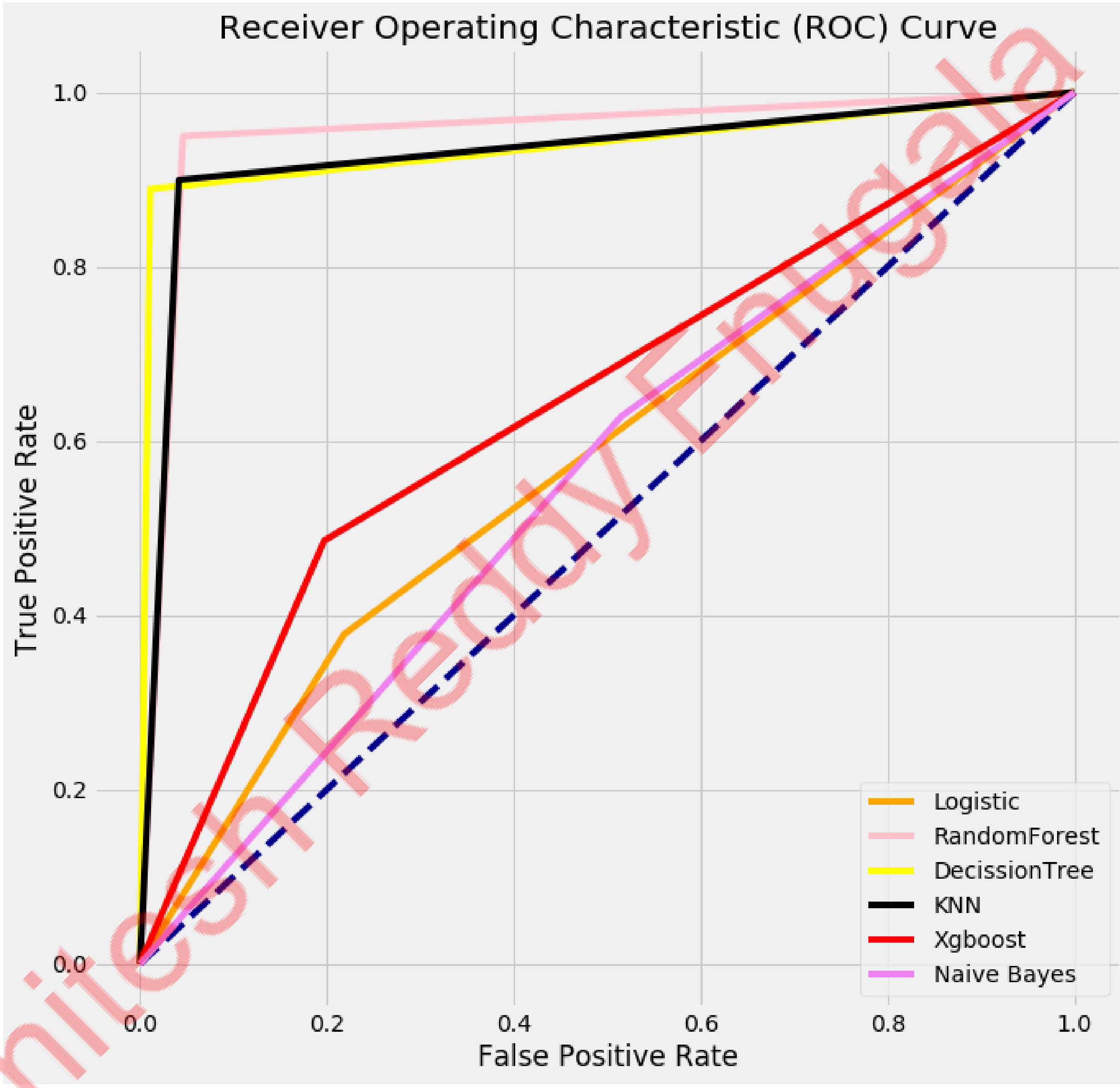
In [95]:
```
# fit model no training data
model = XGBClassifier()
mdl = model.fit(X_train, y_train)
y_pred5 =  np.where(mdl.predict_proba(X_test)[:, 1] > 0.1, 1, 0)
Report(y_test,y_pred5)
```

```
[[189759  46584]
 [ 10548   9955]]
              precision    recall  f1-score   support

           0       0.95      0.80      0.87    236343
           1       0.18      0.49      0.26     20503

    accuracy                           0.78    256846
   macro avg       0.56      0.64      0.56    256846
weighted avg       0.89      0.78      0.82    256846
```

In [96]:
```
from sklearn.naive_bayes import GaussianNB
model = GaussianNB()
gnb = model.fit(X_train, y_train)
y_pred6 =  np.where(gnb.predict_proba(X_test)[:, 1] > 0.1, 1, 0)
Report(y_test,y_pred6)
```

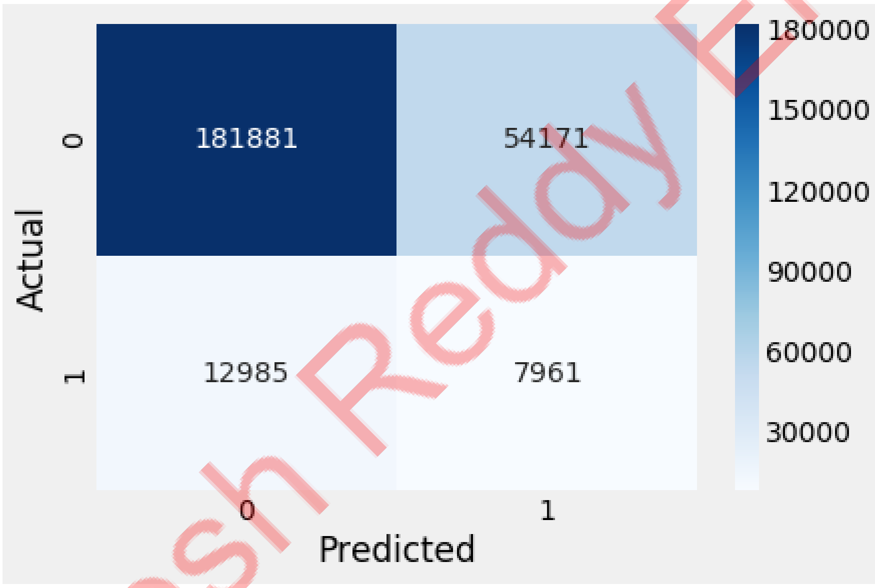
```
[[114584 121759]
 [  7621  12882]]
              precision    recall  f1-score   support

           0       0.94      0.48      0.64    236343
           1       0.10      0.63      0.17     20503

    accuracy                           0.50    256846
   macro avg       0.52      0.56      0.40    256846
weighted avg       0.87      0.50      0.60    256846
```

In [97]:
```
from sklearn.metrics import roc_curve
from sklearn.metrics import roc_auc_score

plt.figure(figsize = (10,10))
fpr, tpr, thresholds = roc_curve(y_test, y_pred)
plt.plot(fpr, tpr, color='orange', label='Logistic')
plt.plot([0, 1], [0, 1], color='darkblue', linestyle='--')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic (ROC) Curve')

fpr1, tpr1, thresholds = roc_curve(y_test, y_pred1)
plt.plot(fpr1, tpr1, color='pink', label='RandomForest')

fpr3, tpr3, thresholds = roc_curve(y_test, y_pred3)
plt.plot(fpr3, tpr3, color='yellow', label='DecissionTree')

fpr4, tpr4, thresholds = roc_curve(y_test, y_pred4)
plt.plot(fpr4, tpr4, color='black', label='KNN')

fpr5, tpr5, thresholds = roc_curve(y_test, y_pred5)
plt.plot(fpr5, tpr5, color='red', label='Xgboost')

fpr6, tpr6, thresholds = roc_curve(y_test, y_pred6)
plt.plot(fpr6, tpr6, color='violet', label='Naive Bayes')

plt.legend()
plt.show()
```


Receiver Operating Characteristic (ROC) Curve
True Positive Rate
False Positive Rate
0.0
0.2
0.4
0.6
0.8
1.0
Logistic
RandomForest
DecissionTree
KNN
Xgboost
Naive Bayes

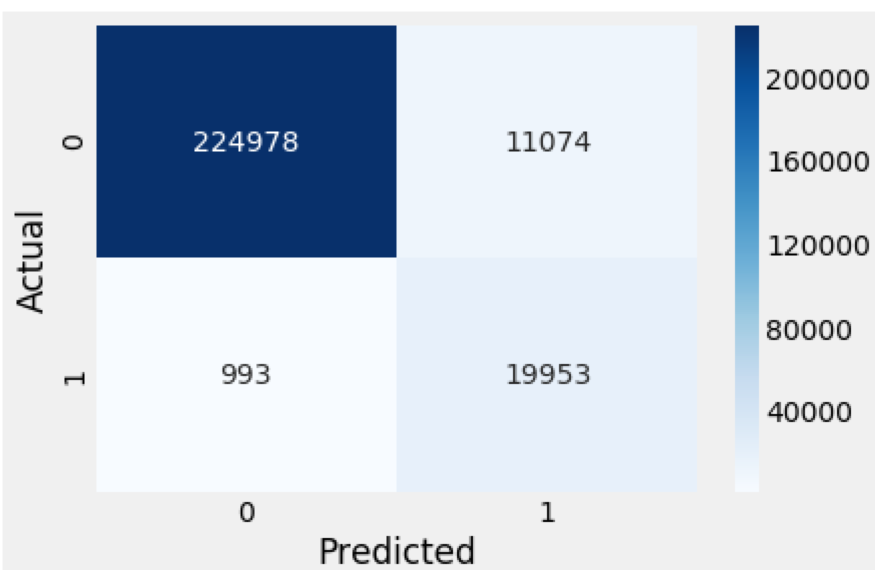
In [239]:
```
import seaborn as sn
#logistic
import seaborn as sn
confusion_matrix = pd.crosstab(y_test, y_pred, rownames=['Actual'], colnames=['Pr

sn.heatmap(confusion_matrix, annot=True, cmap="Blues",fmt='g')
```

Out[239]: <matplotlib.axes._subplots.AxesSubplot at 0x24199e86668>



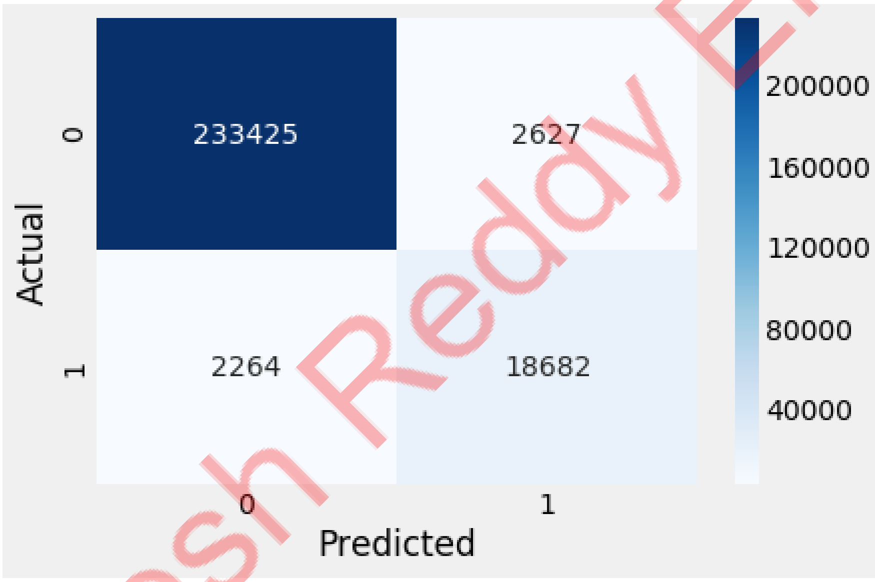
In [240]:
```
#decision tree
import seaborn as sn
confusion_matrix = pd.crosstab(y_test, y_pred1, rownames=['Actual'], colnames=['P

sn.heatmap(confusion_matrix, annot=True, cmap="Blues",fmt='g')
```

Out[240]: <matplotlib.axes._subplots.AxesSubplot at 0x24199843048>



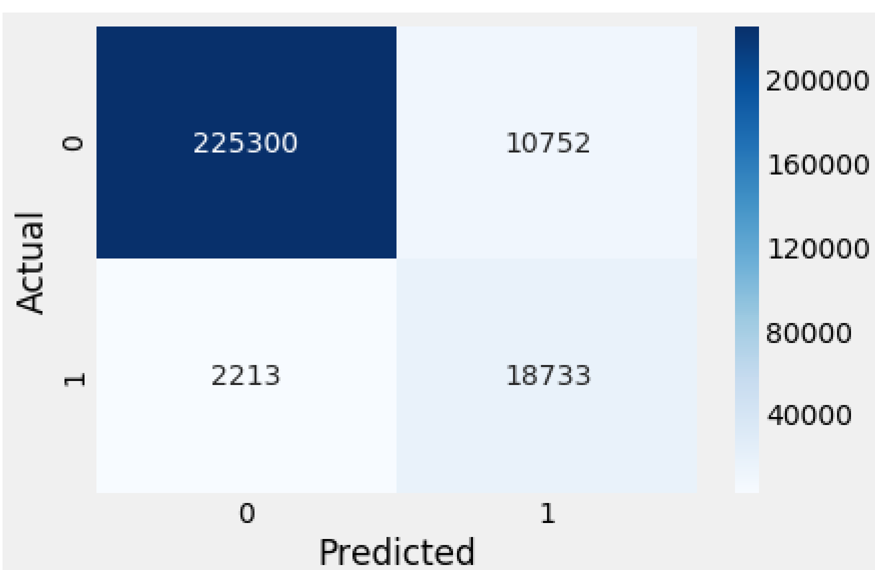
In [241]:
```
#random forest
import seaborn as sn
confusion_matrix = pd.crosstab(y_test, y_pred3, rownames=['Actual'], colnames=['

sn.heatmap(confusion_matrix, annot=True, cmap="Blues",fmt='g')
```

Out[241]: <matplotlib.axes._subplots.AxesSubplot at 0x2431125e550>



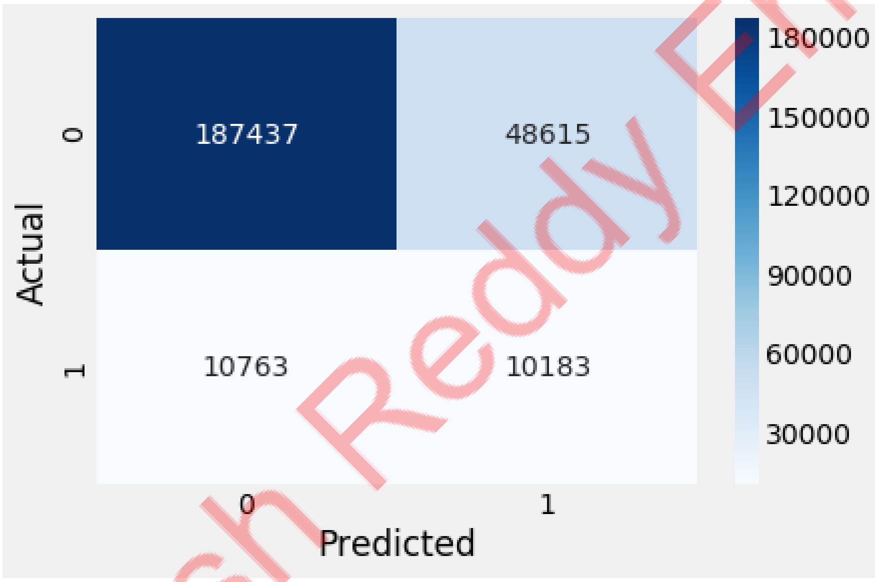
In [242]:
```
#Knn
import seaborn as sn
confusion_matrix = pd.crosstab(y_test, y_pred4, rownames=['Actual'], colnames=['

sn.heatmap(confusion_matrix, annot=True, cmap="Blues",fmt='g')
```

Out[242]: <matplotlib.axes._subplots.AxesSubplot at 0x2412e68c208>



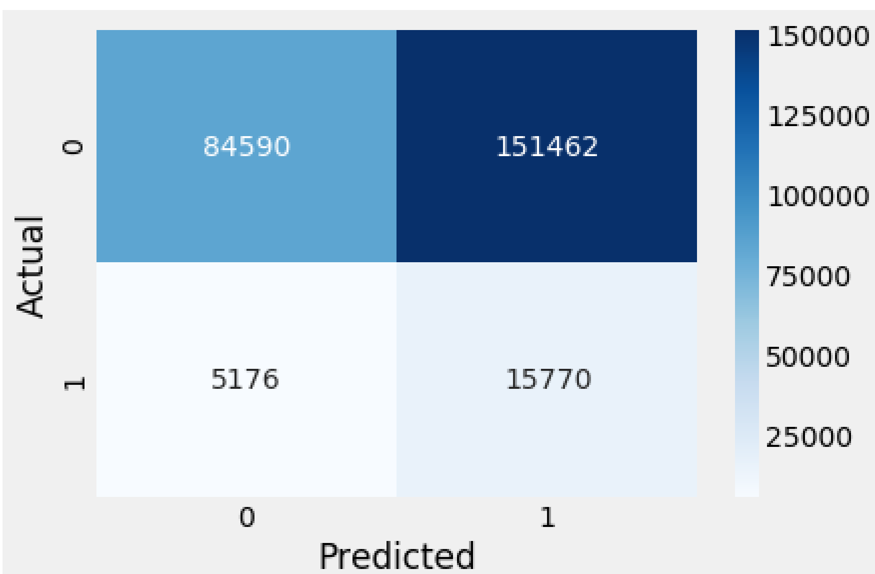
In [243]:
```
#XGboost
import seaborn as sn
confusion_matrix = pd.crosstab(y_test, y_pred5, rownames=['Actual'], colnames=['

sn.heatmap(confusion_matrix, annot=True, cmap="Blues",fmt='g')
```

Out[243]: <matplotlib.axes._subplots.AxesSubplot at 0x2418359a438>



In [244]:
```
#Naive Bayes
import seaborn as sn
confusion_matrix = pd.crosstab(y_test, y_pred6, rownames=['Actual'], colnames=['

sn.heatmap(confusion_matrix, annot=True, cmap="Blues",fmt='g')
```

Out[244]: <matplotlib.axes._subplots.AxesSubplot at 0x2417a7d4588>



In [245]:
```
from sklearn.metrics import precision_score, recall_score, accuracy_score, f1_sc
#logistic
print('logistic')
print('Precision score: {:.4f}'.format(precision_score(y_test,y_pred)))
print('Recall score: {:.4f}'.format(recall_score(y_test,y_pred)))
print('Accuracy score: {:.4f}'.format(accuracy_score(y_test,y_pred)))
print('F1 score: {:.4f}'.format(f1_score(y_test,y_pred)))

#decision tree
print('')
print('decision tree')
print('Precision score: {:.4f}'.format(precision_score(y_test,y_pred1)))
print('Recall score: {:.4f}'.format(recall_score(y_test,y_pred1)))
print('Accuracy score: {:.4f}'.format(accuracy_score(y_test,y_pred1)))
print('F1 score: {:.4f}'.format(f1_score(y_test,y_pred1)))

#random forest
print('')
print('random forest')
print('Precision score: {:.4f}'.format(precision_score(y_test,y_pred3)))
print('Recall score: {:.4f}'.format(recall_score(y_test,y_pred3)))
print('Accuracy score: {:.4f}'.format(accuracy_score(y_test,y_pred3)))
print('F1 score: {:.4f}'.format(f1_score(y_test,y_pred3)))

#knn
print('')
print('knn')
print('Precision score: {:.4f}'.format(precision_score(y_test,y_pred4)))
print('Recall score: {:.4f}'.format(recall_score(y_test,y_pred4)))
print('Accuracy score: {:.4f}'.format(accuracy_score(y_test,y_pred4)))
print('F1 score: {:.4f}'.format(f1_score(y_test,y_pred4)))

#XGBoost
print('')
print('XGBoost')
print('Precision score: {:.4f}'.format(precision_score(y_test,y_pred5)))
print('Recall score: {:.4f}'.format(recall_score(y_test,y_pred5)))
print('Accuracy score: {:.4f}'.format(accuracy_score(y_test,y_pred5)))
print('F1 score: {:.4f}'.format(f1_score(y_test,y_pred5)))

#Naive Bayes
print('')
print('Naive Bayes')
print('Precision score: {:.4f}'.format(precision_score(y_test,y_pred6)))
print('Recall score: {:.4f}'.format(recall_score(y_test,y_pred6)))
print('Accuracy score: {:.4f}'.format(accuracy_score(y_test,y_pred6)))
print('F1 score: {:.4f}'.format(f1_score(y_test,y_pred6)))
```

```
logistic
Precision score: 0.1281
Recall score: 0.3801
Accuracy score: 0.7387
F1 score: 0.1917

decision tree
```

```
Precision score: 0.6431
Recall score: 0.9526
Accuracy score: 0.9530
F1 score: 0.7678

random forest
Precision score: 0.8767
Recall score: 0.8919
Accuracy score: 0.9810
F1 score: 0.8843

knn
Precision score: 0.6353
Recall score: 0.8943
Accuracy score: 0.9496
F1 score: 0.7429

XGBoost
Precision score: 0.1732
Recall score: 0.4862
Accuracy score: 0.7690
F1 score: 0.2554

Naive Bayes
Precision score: 0.0943
Recall score: 0.7529
Accuracy score: 0.3905
F1 score: 0.1676
```

In [ ]: