Introduction

This case study aims to give us an idea of applying EDA in a real business scenario. In this case study, apart from applying the techniques that we have learnt in the EDA module, we will also develop a basic understanding of risk analytics in banking and financial services and understand how data is used to minimise the risk of losing money while lending to customers.

Business Understanding

The loan providing companies find it hard to give loans to the people due to their insufficient or non-existent credit history. Because of that, some consumers use it as their advantage by becoming a defaulter. Suppose you work for a consumer finance company which specialises in lending various types of loans to urban customers. You have to use EDA to analyse the patterns present in the data. This will ensure that the applicants are capable of repaying the loan are not rejected.

When the company receives a loan application, the company has to decide for loan approval based on the applicant's profile. Two types of risks are associated with the bank's decision:

If the applicant is likely to repay the loan, then not approving the loan results in a loss of business to the company

If the applicant is not likely to repay the loan, i.e. he/she is likely to default, then approving the loan may lead to a financial loss for the company.

The data given below contains the information about the loan application at the time of applying for the loan. It contains two types of scenarios:

The client with payment difficulties: he/she had late payment more than X days on at least one of the first Y instalments of the loan in our sample,

All other cases: All other cases when the payment is paid on time.

When a client applies for a loan, there are four types of decisions that could be taken by the client/company):

Approved:

The Company has approved loan Application

Cancelled:

The client cancelled the application sometime during approval. Either the client changed her/his mind about the loan or in some cases due to a higher risk of the client he received worse pricing which he did not want.

Refused:

The company had rejected the loan (because the client does not meet their requirements etc.).

Unused offer:

Loan has been cancelled by the client but on different stages of the process.

Business Objectives

The case study aims to identify patterns which indicate if a client has difficulty paying their installments which may be used for taking actions such as denying the loan, reducing the amount of loan, lending (to risky applicants) at a higher interest rate, etc. This will ensure that the consumers capable of repaying the loan are not rejected. Identification of such applicants using EDA is the aim of this case study.

In other words, the company wants to understand the driving factors (or driver variables) behind loan default, i.e. the variables which are strong indicators of default. The company can utilise this knowledge for its portfolio and risk assessment.

Data Understanding

1. 'application_data.csv'

It contains all the information of the client at the time of application. The data is about whether a client has payment difficulties.

2. 'previous_application.csv'

It contains information about the client's previous loan data. It contains the data whether the previous application had been Approved, Cancelled, Refused or Unused offer.

3. 'columns_description.csv'

It is data dictionary which describes the meaning of the variables.

The solution is made in 2 different ipymb files

- 1st file contains detailed analysis (EDA) on application_data to identify the important features which help us to identify the defaulters
- 2nd file contains data where we inner join the records (application_data, previous_application) with same the SK_ID_CURR

IMPORTING LIBRARIES

```
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
import pandas as pd
import seaborn as sns
import warnings
warnings.filterwarnings("ignore")
import itertools
```

Importing Data

```
application_data = pd.read_csv(r'/kaggle/input/application_data.csv')
previous_application = pd.read_csv(r'/kaggle/input/previous_application.csv')
columns_description = pd.read_csv(r'/kaggle/input/columns_description.csv',skiprows=1)
```

Data Dimensions

```
print ("application_data     :",application_data.shape)
print ("previous_application :",previous_application.shape)
print ("columns_description  :",columns_description.shape)
```

```
application_data     : (307511, 122)
previous_application : (1670214, 37)
columns_description  : (159, 5)
```

First Few rows of Data

```
pd.set_option("display.max_rows", None, "display.max_columns", None)
display("application_data")
display(application_data.head(3))
```

```
'application_data'
```

	SK_ID_CURR	TARGET	NAME_CONTRACT_TYPE	CODE_GENDER	FLAG_OWN_CAR	FLAG_OWN_REALTY	CNT_CHILDREN	AMT_INCOME_TOTAL	AMT_CREDIT
0	100002	1	Cash loans	M	N	Y	0	202500.0	406597.5
1	100003	0	Cash loans	F	N	N	0	270000.0	1293502.5
2	100004	0	Revolving loans	M	Y	Y	0	67500.0	135000.0

```
display("previous_application ")
display(previous_application.head(3))
```

'previous_application '

	SK_ID_PREV	SK_ID_CURR	NAME_CONTRACT_TYPE	AMT_ANNUITY	AMT_APPLICATION	AMT_CREDIT	AMT_DOWN_PAYMENT	AMT_GOODS_PRICE	WEEKDA
0	2030495	271877	Consumer loans	1730.430	17145.0	17145.0	0.0	17145.0	

Term Dictionary

```
display("columns_description")
pd.set_option('display.max_colwidth', -1)
columns_description=columns_description.drop(['1'],axis=1)
display(columns_description)
```

'columns_description'

	application_data	SK_ID_CURR	ID of loan in our sample	Unnamed: 4
0	application_data	TARGET	Target variable (1 - client with payment difficulties: he/she had late payment more than X days on at least one of the first Y installments of the loan in our sample, 0 - all other cases)	NaN
1	application_data	NAME_CONTRACT_TYPE	Identification if loan is cash or revolving	NaN
2	application_data	CODE_GENDER	Gender of the client	NaN
3	application_data	FLAG_OWN_CAR	Flag if the client owns a car	NaN
4	application_data	FLAG_OWN_REALTY	Flag if client owns a house or flat	NaN
5	application_data	CNT_CHILDREN	Number of children the client has	NaN
6	application_data	AMT_INCOME_TOTAL	Income of the client	NaN
7	application_data	AMT_CREDIT	Credit amount of the loan	NaN
8	application_data	AMT_ANNUITY	Loan annuity	NaN
9	application_data	AMT_GOODS_PRICE	For consumer loans it is the price of the goods for which the loan is given	NaN
10	application_data	NAME_TYPE_SUITE	Who was accompanying client when he was applying for the loan	NaN
11	application_data	NAME_INCOME_TYPE	Clients income type (businessman, working, maternity leave,…)	NaN
12	application_data	NAME_EDUCATION_TYPE	Level of highest education the client achieved	NaN
13	application_data	NAME_FAMILY_STATUS	Family status of the client	NaN
14	application_data	NAME_HOUSING_TYPE	What is the housing situation of the client (renting, living with parents, ...)	NaN
15	application_data	REGION_POPULATION_RELATIVE	Normalized population of region where client lives (higher number means the client lives in more populated region)	normalized
16	application_data	DAYS_BIRTH	Client's age in days at the time of application	time only relative to the application

17	application_data	DAYS_EMPLOYED	How many days before the application the person started current employment	time only relative to the application
18	application_data	DAYS_REGISTRATION	How many days before the application did client change his registration	time only relative to the application
19	application_data	DAYS_ID_PUBLISH	How many days before the application did client change the identity document with which he applied for the loan	time only relative to the application
20	application_data	OWN_CAR_AGE	Age of client's car	NaN
21	application_data	FLAG_MOBIL	Did client provide mobile phone (1=YES, 0=NO)	NaN
22	application_data	FLAG_EMP_PHONE	Did client provide work phone (1=YES, 0=NO)	NaN
23	application_data	FLAG_WORK_PHONE	Did client provide home phone (1=YES, 0=NO)	NaN
24	application_data	FLAG_CONT_MOBILE	Was mobile phone reachable (1=YES, 0=NO)	NaN
25	application_data	FLAG_PHONE	Did client provide home phone (1=YES, 0=NO)	NaN
26	application_data	FLAG_EMAIL	Did client provide email (1=YES, 0=NO)	NaN
27	application_data	OCCUPATION_TYPE	What kind of occupation does the client have	NaN
28	application_data	CNT_FAM_MEMBERS	How many family members does client have	NaN
29	application_data	REGION_RATING_CLIENT	Our rating of the region where client lives (1,2,3)	NaN
30	application_data	REGION_RATING_CLIENT_W_CITY	Our rating of the region where client lives with taking city into account (1,2,3)	NaN
31	application_data	WEEKDAY_APPR_PROCESS_START	On which day of the week did the client apply for the loan	NaN
32	application_data	HOUR_APPR_PROCESS_START	Approximately at what hour did the client apply for the loan	rounded
33	application_data	REG_REGION_NOT_LIVE_REGION	Flag if client's permanent address does not match contact address (1=different, 0=same, at region level)	NaN
			Flag if client's permanent address does not match work address	

34	application_data	REG_REGION_NOT_WORK_REGION	Flag if client's permanent address does not match work address (1=different, 0=same, at region level)	NaN
35	application_data	LIVE_REGION_NOT_WORK_REGION	Flag if client's contact address does not match work address (1=different, 0=same, at region level)	NaN
36	application_data	REG_CITY_NOT_LIVE_CITY	Flag if client's permanent address does not match contact address (1=different, 0=same, at city level)	NaN
37	application_data	REG_CITY_NOT_WORK_CITY	Flag if client's permanent address does not match work address (1=different, 0=same, at city level)	NaN
38	application_data	LIVE_CITY_NOT_WORK_CITY	Flag if client's contact address does not match work address (1=different, 0=same, at city level)	NaN
39	application_data	ORGANIZATION_TYPE	Type of organization where client works	NaN
40	application_data	EXT_SOURCE_1	Normalized score from external data source	normalized
41	application_data	EXT_SOURCE_2	Normalized score from external data source	normalized
42	application_data	EXT_SOURCE_3	Normalized score from external data source	normalized
43	application_data	APARTMENTS_AVG	Normalized information about building where the client lives, What is average (_AVG suffix), modus (_MODE suffix), median (_MEDI suffix) apartment size, common area, living area, age of building, number of elevators, number of entrances, state of the building, number of floor	normalized
44	application_data	BASEMENTAREA_AVG	Normalized information about building where the client lives, What is average (_AVG suffix), modus (_MODE suffix), median (_MEDI suffix) apartment size, common area, living area, age of building, number of elevators, number of entrances, state of the building, number of floor	normalized
45	application_data	YEARS_BEGINEXPLUATATION_AVG	Normalized information about building where the client lives, What is average (_AVG suffix), modus (_MODE suffix), median (_MEDI suffix) apartment size, common area, living area, age of building, number of elevators, number of entrances, state of the building, number of floor	normalized
46	application_data	YEARS_BUILD_AVG	Normalized information about building where the client lives, What is average (_AVG suffix), modus (_MODE suffix), median (_MEDI suffix) apartment size, common area, living area, age of building, number of elevators, number of entrances, state of the building, number of floor	normalized

			number of floor	
47	application_data	COMMONAREA_AVG	Normalized information about building where the client lives, What is average (_AVG suffix), modus (_MODE suffix), median (_MEDI suffix) apartment size, common area, living area, age of building, number of elevators, number of entrances, state of the building, number of floor	normalized
48	application_data	ELEVATORS_AVG	Normalized information about building where the client lives, What is average (_AVG suffix), modus (_MODE suffix), median (_MEDI suffix) apartment size, common area, living area, age of building, number of elevators, number of entrances, state of the building, number of floor	normalized
49	application_data	ENTRANCES_AVG	Normalized information about building where the client lives, What is average (_AVG suffix), modus (_MODE suffix), median (_MEDI suffix) apartment size, common area, living area, age of building, number of elevators, number of entrances, state of the building, number of floor	normalized
50	application_data	FLOORSMAX_AVG	Normalized information about building where the client lives, What is average (_AVG suffix), modus (_MODE suffix), median (_MEDI suffix) apartment size, common area, living area, age of building, number of elevators, number of entrances, state of the building, number of floor	normalized
51	application_data	FLOORSMIN_AVG	Normalized information about building where the client lives, What is average (_AVG suffix), modus (_MODE suffix), median (_MEDI suffix) apartment size, common area, living area, age of building, number of elevators, number of entrances, state of the building, number of floor	normalized
52	application_data	LANDAREA_AVG	Normalized information about building where the client lives, What is average (_AVG suffix), modus (_MODE suffix), median (_MEDI suffix) apartment size, common area, living area, age of building, number of elevators, number of entrances, state of the building, number of floor	normalized
53	application_data	LIVINGAPARTMENTS_AVG	Normalized information about building where the client lives, What is average (_AVG suffix), modus (_MODE suffix), median (_MEDI suffix) apartment size, common area, living area, age of building, number of elevators, number of entrances, state of the building, number of floor	normalized
			Normalized information about building where the client lives, What is	

54	application_data	LIVINGAREA_AVG	average (_AVG suffix), modus (_MODE suffix), median (_MEDI suffix) apartment size, common area, living area, age of building, number of elevators, number of entrances, state of the building, number of floor	normalized
55	application_data	NONLIVINGAPARTMENTS_AVG	Normalized information about building where the client lives, What is average (_AVG suffix), modus (_MODE suffix), median (_MEDI suffix) apartment size, common area, living area, age of building, number of elevators, number of entrances, state of the building, number of floor	normalized
56	application_data	NONLIVINGAREA_AVG	Normalized information about building where the client lives, What is average (_AVG suffix), modus (_MODE suffix), median (_MEDI suffix) apartment size, common area, living area, age of building, number of elevators, number of entrances, state of the building, number of floor	normalized
57	application_data	APARTMENTS_MODE	Normalized information about building where the client lives, What is average (_AVG suffix), modus (_MODE suffix), median (_MEDI suffix) apartment size, common area, living area, age of building, number of elevators, number of entrances, state of the building, number of floor	normalized
58	application_data	BASEMENTAREA_MODE	Normalized information about building where the client lives, What is average (_AVG suffix), modus (_MODE suffix), median (_MEDI suffix) apartment size, common area, living area, age of building, number of elevators, number of entrances, state of the building, number of floor	normalized
59	application_data	YEARS_BEGINEXPLUATATION_MODE	Normalized information about building where the client lives, What is average (_AVG suffix), modus (_MODE suffix), median (_MEDI suffix) apartment size, common area, living area, age of building, number of elevators, number of entrances, state of the building, number of floor	normalized
60	application_data	YEARS_BUILD_MODE	Normalized information about building where the client lives, What is average (_AVG suffix), modus (_MODE suffix), median (_MEDI suffix) apartment size, common area, living area, age of building, number of elevators, number of entrances, state of the building, number of floor	normalized
61	application_data	COMMONAREA_MODE	Normalized information about building where the client lives, What is average (_AVG suffix), modus (_MODE suffix), median (_MEDI suffix) apartment size, common area, living area, age of building,	normalized

			number of elevators, number of entrances, state of the building, number of floor	
62	application_data	ELEVATORS_MODE	Normalized information about building where the client lives, What is average (_AVG suffix), modus (_MODE suffix), median (_MEDI suffix) apartment size, common area, living area, age of building, number of elevators, number of entrances, state of the building, number of floor	normalized
63	application_data	ENTRANCES_MODE	Normalized information about building where the client lives, What is average (_AVG suffix), modus (_MODE suffix), median (_MEDI suffix) apartment size, common area, living area, age of building, number of elevators, number of entrances, state of the building, number of floor	normalized
64	application_data	FLOORSMAX_MODE	Normalized information about building where the client lives, What is average (_AVG suffix), modus (_MODE suffix), median (_MEDI suffix) apartment size, common area, living area, age of building, number of elevators, number of entrances, state of the building, number of floor	normalized
65	application_data	FLOORSMIN_MODE	Normalized information about building where the client lives, What is average (_AVG suffix), modus (_MODE suffix), median (_MEDI suffix) apartment size, common area, living area, age of building, number of elevators, number of entrances, state of the building, number of floor	normalized
66	application_data	LANDAREA_MODE	Normalized information about building where the client lives, What is average (_AVG suffix), modus (_MODE suffix), median (_MEDI suffix) apartment size, common area, living area, age of building, number of elevators, number of entrances, state of the building, number of floor	normalized
67	application_data	LIVINGAPARTMENTS_MODE	Normalized information about building where the client lives, What is average (_AVG suffix), modus (_MODE suffix), median (_MEDI suffix) apartment size, common area, living area, age of building, number of elevators, number of entrances, state of the building, number of floor	normalized
68	application_data	LIVINGAREA_MODE	Normalized information about building where the client lives, What is average (_AVG suffix), modus (_MODE suffix), median (_MEDI suffix) apartment size, common area, living area, age of building, number of elevators, number of entrances, state of the building, number of floor	normalized

69	application_data	NONLIVINGAPARTMENTS_MODE	Normalized information about building where the client lives, What is average (_AVG suffix), modus (_MODE suffix), median (_MEDI suffix) apartment size, common area, living area, age of building, number of elevators, number of entrances, state of the building, number of floor	normalized
70	application_data	NONLIVINGAREA_MODE	Normalized information about building where the client lives, What is average (_AVG suffix), modus (_MODE suffix), median (_MEDI suffix) apartment size, common area, living area, age of building, number of elevators, number of entrances, state of the building, number of floor	normalized
71	application_data	APARTMENTS_MEDI	Normalized information about building where the client lives, What is average (_AVG suffix), modus (_MODE suffix), median (_MEDI suffix) apartment size, common area, living area, age of building, number of elevators, number of entrances, state of the building, number of floor	normalized
72	application_data	BASEMENTAREA_MEDI	Normalized information about building where the client lives, What is average (_AVG suffix), modus (_MODE suffix), median (_MEDI suffix) apartment size, common area, living area, age of building, number of elevators, number of entrances, state of the building, number of floor	normalized
73	application_data	YEARS_BEGINEXPLUATATION_MEDI	Normalized information about building where the client lives, What is average (_AVG suffix), modus (_MODE suffix), median (_MEDI suffix) apartment size, common area, living area, age of building, number of elevators, number of entrances, state of the building, number of floor	normalized
74	application_data	YEARS_BUILD_MEDI	Normalized information about building where the client lives, What is average (_AVG suffix), modus (_MODE suffix), median (_MEDI suffix) apartment size, common area, living area, age of building, number of elevators, number of entrances, state of the building, number of floor	normalized
75	application_data	COMMONAREA_MEDI	Normalized information about building where the client lives, What is average (_AVG suffix), modus (_MODE suffix), median (_MEDI suffix) apartment size, common area, living area, age of building, number of elevators, number of entrances, state of the building, number of floor	normalized
			Normalized information about building where the client lives, What is average (_AVG suffix), modus (_MODE suffix), median (_MEDI	

76	application_data	ELEVATORS_MEDI	suffix) apartment size, common area, living area, age of building, number of elevators, number of entrances, state of the building, number of floor	normalized
77	application_data	ENTRANCES_MEDI	Normalized information about building where the client lives, What is average (_AVG suffix), modus (_MODE suffix), median (_MEDI suffix) apartment size, common area, living area, age of building, number of elevators, number of entrances, state of the building, number of floor	normalized
78	application_data	FLOORSMAX_MEDI	Normalized information about building where the client lives, What is average (_AVG suffix), modus (_MODE suffix), median (_MEDI suffix) apartment size, common area, living area, age of building, number of elevators, number of entrances, state of the building, number of floor	normalized
79	application_data	FLOORSMIN_MEDI	Normalized information about building where the client lives, What is average (_AVG suffix), modus (_MODE suffix), median (_MEDI suffix) apartment size, common area, living area, age of building, number of elevators, number of entrances, state of the building, number of floor	normalized
80	application_data	LANDAREA_MEDI	Normalized information about building where the client lives, What is average (_AVG suffix), modus (_MODE suffix), median (_MEDI suffix) apartment size, common area, living area, age of building, number of elevators, number of entrances, state of the building, number of floor	normalized
81	application_data	LIVINGAPARTMENTS_MEDI	Normalized information about building where the client lives, What is average (_AVG suffix), modus (_MODE suffix), median (_MEDI suffix) apartment size, common area, living area, age of building, number of elevators, number of entrances, state of the building, number of floor	normalized
82	application_data	LIVINGAREA_MEDI	Normalized information about building where the client lives, What is average (_AVG suffix), modus (_MODE suffix), median (_MEDI suffix) apartment size, common area, living area, age of building, number of elevators, number of entrances, state of the building, number of floor	normalized
83	application_data	NONLIVINGAPARTMENTS_MEDI	Normalized information about building where the client lives, What is average (_AVG suffix), modus (_MODE suffix), median (_MEDI suffix) apartment size, common area, living area, age of building, number of elevators, number of entrances, state of the building, number of floor	normalized

			number of floor	
84	application_data	NONLIVINGAREA_MEDI	Normalized information about building where the client lives, What is average (_AVG suffix), modus (_MODE suffix), median (_MEDI suffix) apartment size, common area, living area, age of building, number of elevators, number of entrances, state of the building, number of floor	normalized
85	application_data	FONDKAPREMONT_MODE	Normalized information about building where the client lives, What is average (_AVG suffix), modus (_MODE suffix), median (_MEDI suffix) apartment size, common area, living area, age of building, number of elevators, number of entrances, state of the building, number of floor	normalized
86	application_data	HOUSETYPE_MODE	Normalized information about building where the client lives, What is average (_AVG suffix), modus (_MODE suffix), median (_MEDI suffix) apartment size, common area, living area, age of building, number of elevators, number of entrances, state of the building, number of floor	normalized
87	application_data	TOTALAREA_MODE	Normalized information about building where the client lives, What is average (_AVG suffix), modus (_MODE suffix), median (_MEDI suffix) apartment size, common area, living area, age of building, number of elevators, number of entrances, state of the building, number of floor	normalized
88	application_data	WALLSMATERIAL_MODE	Normalized information about building where the client lives, What is average (_AVG suffix), modus (_MODE suffix), median (_MEDI suffix) apartment size, common area, living area, age of building, number of elevators, number of entrances, state of the building, number of floor	normalized
89	application_data	EMERGENCYSTATE_MODE	Normalized information about building where the client lives, What is average (_AVG suffix), modus (_MODE suffix), median (_MEDI suffix) apartment size, common area, living area, age of building, number of elevators, number of entrances, state of the building, number of floor	normalized
90	application_data	OBS_30_CNT_SOCIAL_CIRCLE	How many observation of client's social surroundings with observable 30 DPD (days past due) default	NaN
91	application_data	DEF_30_CNT_SOCIAL_CIRCLE	How many observation of client's social surroundings defaulted on 30 DPD (days past due)	NaN
92	application_data	OBS_60_CNT_SOCIAL_CIRCLE	How many observation of client's social surroundings with observable	NaN

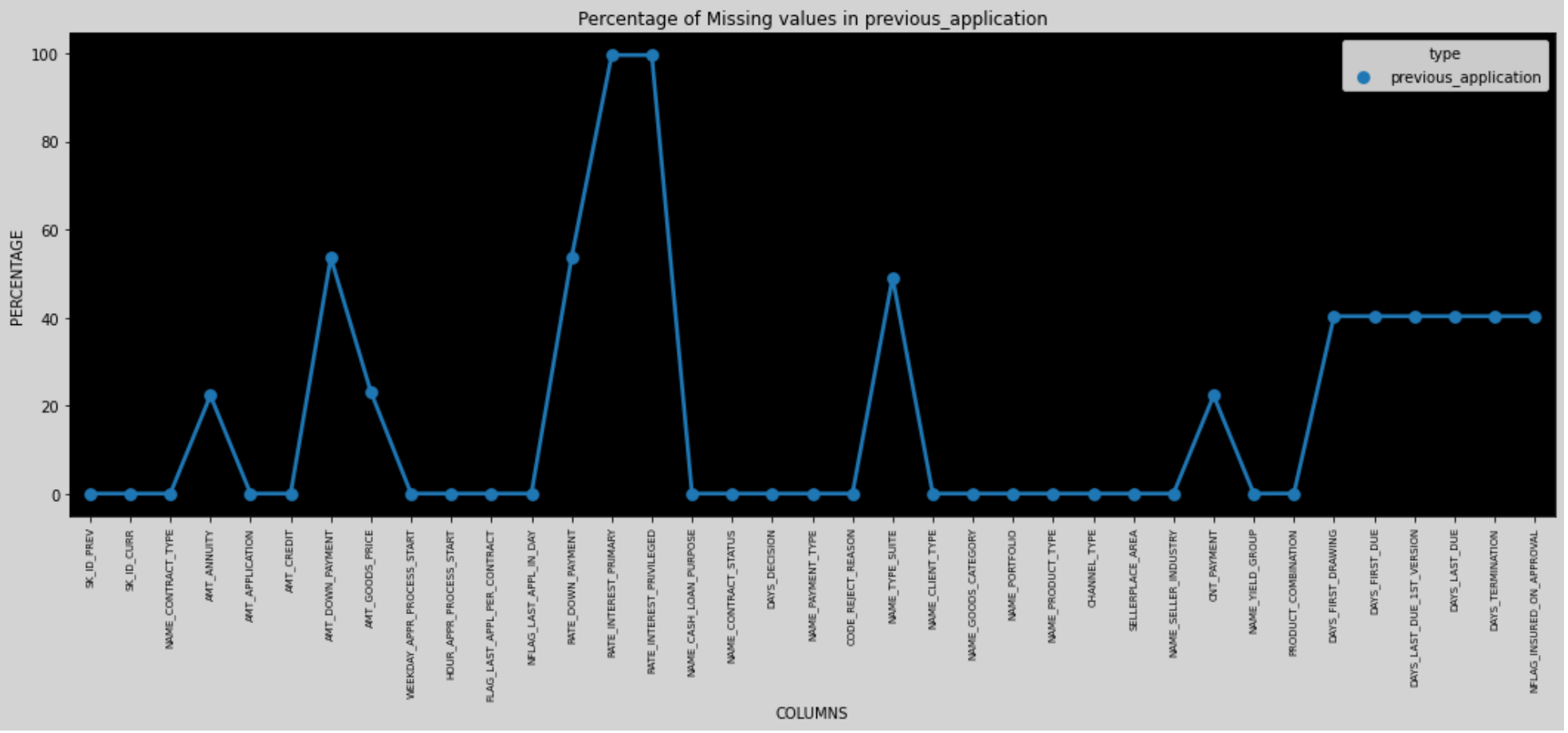
92	application_data	OBS_60_CNT_SOCIAL_CIRCLE	60 DPD (days past due) default	NaN
93	application_data	DEF_60_CNT_SOCIAL_CIRCLE	How many observation of client's social surroundings defaulted on 60 (days past due) DPD	NaN
94	application_data	DAYS_LAST_PHONE_CHANGE	How many days before application did client change phone	NaN
95	application_data	FLAG_DOCUMENT_2	Did client provide document 2	NaN
96	application_data	FLAG_DOCUMENT_3	Did client provide document 3	NaN
97	application_data	FLAG_DOCUMENT_4	Did client provide document 4	NaN
98	application_data	FLAG_DOCUMENT_5	Did client provide document 5	NaN
99	application_data	FLAG_DOCUMENT_6	Did client provide document 6	NaN
100	application_data	FLAG_DOCUMENT_7	Did client provide document 7	NaN
101	application_data	FLAG_DOCUMENT_8	Did client provide document 8	NaN
102	application_data	FLAG_DOCUMENT_9	Did client provide document 9	NaN
103	application_data	FLAG_DOCUMENT_10	Did client provide document 10	NaN
104	application_data	FLAG_DOCUMENT_11	Did client provide document 11	NaN
105	application_data	FLAG_DOCUMENT_12	Did client provide document 12	NaN
106	application_data	FLAG_DOCUMENT_13	Did client provide document 13	NaN
107	application_data	FLAG_DOCUMENT_14	Did client provide document 14	NaN
108	application_data	FLAG_DOCUMENT_15	Did client provide document 15	NaN
109	application_data	FLAG_DOCUMENT_16	Did client provide document 16	NaN
110	application_data	FLAG_DOCUMENT_17	Did client provide document 17	NaN
111	application_data	FLAG_DOCUMENT_18	Did client provide document 18	NaN
112	application_data	FLAG_DOCUMENT_19	Did client provide document 19	NaN
113	application_data	FLAG_DOCUMENT_20	Did client provide document 20	NaN
114	application_data	FLAG_DOCUMENT_21	Did client provide document 21	NaN

115	application_data	AMT_REQ_CREDIT_BUREAU_HOUR	Number of enquiries to Credit Bureau about the client one hour before application	NaN
116	application_data	AMT_REQ_CREDIT_BUREAU_DAY	Number of enquiries to Credit Bureau about the client one day before application (excluding one hour before application)	NaN
117	application_data	AMT_REQ_CREDIT_BUREAU_WEEK	Number of enquiries to Credit Bureau about the client one week before application (excluding one day before application)	NaN
118	application_data	AMT_REQ_CREDIT_BUREAU_MON	Number of enquiries to Credit Bureau about the client one month before application (excluding one week before application)	NaN
119	application_data	AMT_REQ_CREDIT_BUREAU_QRT	Number of enquiries to Credit Bureau about the client 3 month before application (excluding one month before application)	NaN
120	application_data	AMT_REQ_CREDIT_BUREAU_YEAR	Number of enquiries to Credit Bureau about the client one day year (excluding last 3 months before application)	NaN
121	previous_application.csv	SK_ID_PREV	ID of previous credit in Home credit related to loan in our sample. (One loan in our sample can have 0,1,2 or more previous loan applications in Home Credit, previous application could, but not necessarily have to lead to credit)	hashed
122	previous_application.csv	SK_ID_CURR	ID of loan in our sample	hashed
123	previous_application.csv	NAME_CONTRACT_TYPE	Contract product type (Cash loan, consumer loan [POS] ,...) of the previous application	NaN
124	previous_application.csv	AMT_ANNUITY	Annuity of previous application	NaN
125	previous_application.csv	AMT_APPLICATION	For how much credit did client ask on the previous application	NaN
			Final credit amount on the previous application. This differs from AMT_APPLICATION in a way that the AMT_APPLICATION is the	

Percentage of Missing values in previous_application

```
fig = plt.figure(figsize=(18,6))
miss_previous_application = pd.DataFrame((previous_application.isnull().sum())*100/previous_application.shape[0]).reset_index()
miss_previous_application["type"] = "previous_application"
ax = sns.pointplot("index",0,data=miss_previous_application,hue="type")
plt.xticks(rotation =90,fontsize =7)
plt.title("Percentage of Missing values in previous_application")
```

```
plt.ylabel("PERCENTAGE")
plt.xlabel("COLUMNS")
ax.set_facecolor("k")
fig.set_facecolor("lightgrey")
```




```
round(100*(previous_application.isnull().sum()/len(previous_application.index)),2)
```

```
SK_ID_PREV                   0.00
SK_ID_CURR                   0.00
NAME_CONTRACT_TYPE           0.00
AMT_ANNUITY                 22.29
```

```
AMT_APPLICATION               0.00
AMT_CREDIT                    0.00
AMT_DOWN_PAYMENT             53.64
AMT_GOODS_PRICE              23.08
WEEKDAY_APPR_PROCESS_START    0.00
HOUR_APPR_PROCESS_START       0.00
FLAG_LAST_APPL_PER_CONTRACT   0.00
NFLAG_LAST_APPL_IN_DAY        0.00
RATE_DOWN_PAYMENT            53.64
RATE_INTEREST_PRIMARY        99.64
RATE_INTEREST_PRIVILEGED     99.64
NAME_CASH_LOAN_PURPOSE        0.00
NAME_CONTRACT_STATUS          0.00
DAYS_DECISION                 0.00
NAME_PAYMENT_TYPE             0.00
CODE_REJECT_REASON            0.00
NAME_TYPE_SUITE              49.12
NAME_CLIENT_TYPE              0.00
NAME_GOODS_CATEGORY           0.00
NAME_PORTFOLIO                0.00
NAME_PRODUCT_TYPE             0.00
CHANNEL_TYPE                  0.00
SELLERPLACE_AREA              0.00
NAME_SELLER_INDUSTRY          0.00
CNT_PAYMENT                  22.29
NAME_YIELD_GROUP              0.00
PRODUCT_COMBINATION           0.02
DAYS_FIRST_DRAWING           40.30
DAYS_FIRST_DUE               40.30
DAYS_LAST_DUE_1ST_VERSION    40.30
DAYS_LAST_DUE                40.30
DAYS_TERMINATION             40.30
NFLAG_INSURED_ON_APPROVAL    40.30
dtype: float64
```

Removing columns with missing values more than 50%

key point

As per Industrial Standard, max Threshold limit can be between 40% to 50 % depending upon the data acquired in specific sector.

```
previous_application=previous_application.drop([ 'AMT_DOWN_PAYMENT', 'RATE_DOWN_PAYMENT', 'RATE_INTEREST_PRIMARY',
       "RATE_INTEREST_PRIVILEGED"],axis=1)


fig = plt.figure(figsize=(18,6))
miss_previous_application = pd.DataFrame((previous_application.isnull().sum())*100/previous_application.shape[0]).reset_index()
miss_previous_application["type"] = "previous_application"
ax = sns.pointplot("index",0,data=miss_previous_application,hue="type")
plt.xticks(rotation =90,fontsize =7)
plt.title("Percentage of Missing values in previous_application")
plt.ylabel("PERCENTAGE")
plt.xlabel("COLUMNS")
ax.set_facecolor("k")
fig.set_facecolor("lightgrey")
```


```
round(100*(previous_application.isnull().sum()/len(previous_application.index)),2)
```

```
SK_ID_PREV                    0.00
SK_ID_CURR                    0.00
NAME_CONTRACT_TYPE            0.00
AMT_ANNUITY                  22.29
AMT_APPLICATION               0.00
AMT_CREDIT                    0.00
AMT_GOODS_PRICE              23.08
WEEKDAY_APPR_PROCESS_START    0.00
HOUR_APPR_PROCESS_START       0.00
FLAG_LAST_APPL_PER_CONTRACT   0.00
NFLAG_LAST_APPL_IN_DAY        0.00
NAME_CASH_LOAN_PURPOSE        0.00
NAME_CONTRACT_STATUS          0.00
DAYS_DECISION                 0.00
NAME_PAYMENT_TYPE             0.00
CODE_REJECT_REASON            0.00
NAME_TYPE_SUITE              49.12
NAME_CLIENT_TYPE              0.00
NAME_GOODS_CATEGORY           0.00
NAME_PORTFOLIO                0.00
NAME_PRODUCT_TYPE             0.00
CHANNEL_TYPE                  0.00
SELLERPLACE_AREA              0.00
NAME_SELLER_INDUSTRY          0.00
CNT_PAYMENT                  22.29
NAME_YIELD_GROUP              0.00
PRODUCT_COMBINATION           0.02
DAYS_FIRST_DRAWING           40.30
DAYS_FIRST_DUE               40.30
DAYS_LAST_DUE_1ST_VERSION    40.30
DAYS_LAST_DUE                40.30
DAYS_TERMINATION             40.30
NFLAG_INSURED_ON_APPROVAL    40.30
dtype: float64
```

MISSING values Suggestion

```
print("AMT_ANNUITY NULL COUNT:" ,previous_application['AMT_ANNUITY'].isnull().sum())
```

```
AMT_ANNUITY NULL COUNT: 372235
```

```
previous_application['AMT_ANNUITY'].describe()
```

```
count    1.297979e+06
mean     1.595512e+04
std      1.478214e+04
min      0.000000e+00
25%      6.321780e+03
50%      1.125000e+04
75%      2.065842e+04
max      4.180581e+05
Name: AMT_ANNUITY, dtype: float64
```

```
sns.set_style('whitegrid')
sns.distplot(previous_application['AMT_ANNUITY'])
plt.show()
```

Suggestion

We can Fill NA with 15955 i.e. Mean for this field

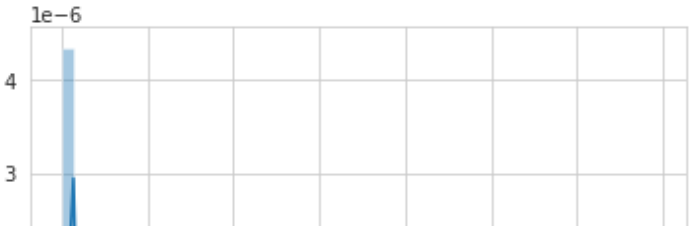
```
print("AMT_GOODS_PRICE NULL COUNT:" ,previous_application['AMT_GOODS_PRICE'].isnull().sum())
```

```
AMT_GOODS_PRICE NULL COUNT: 385515
```

```
previous_application['AMT_GOODS_PRICE'].describe()
```

```
count    1.284699e+06
mean     2.278473e+05
std      3.153966e+05
min      0.000000e+00
25%      5.084100e+04
50%      1.123200e+05
75%      2.340000e+05
max      6.905160e+06
Name: AMT_GOODS_PRICE, dtype: float64
```

```
sns.set_style('whitegrid')
sns.distplot(previous_application['AMT_GOODS_PRICE'])
plt.show()
```



Suggestion

We can Fill NA with 112320 i.e. Median for this field

```
print("NAME_TYPE_SUITE NULL COUNT:" ,previous_application['NAME_TYPE_SUITE'].isnull().sum())
```

```
NAME_TYPE_SUITE NULL COUNT: 820405
```

```
previous_application['NAME_TYPE_SUITE'].value_counts()
```

```
Unaccompanied      508970
Family             213263
Spouse, partner    67069
Children           31566
Other_B            17624
Other_A            9077
Group of people    2240
Name: NAME_TYPE_SUITE, dtype: int64
```

Suggestion

We can Fill NA with Unaccompanied i.e. Mode for this field

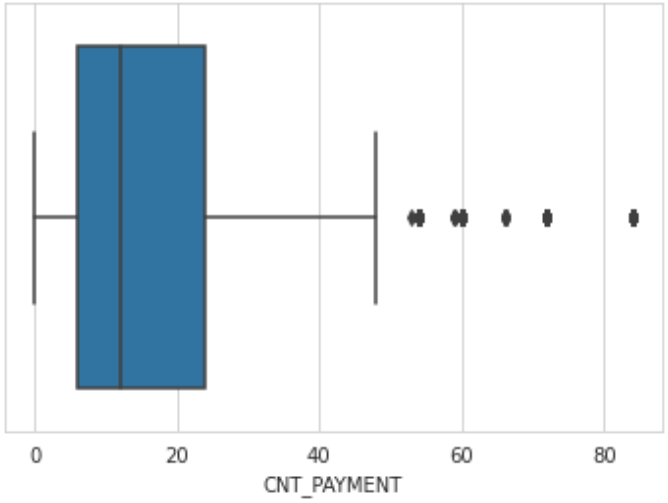
```
print("CNT_PAYMENT NULL COUNT:" ,previous_application['CNT_PAYMENT'].isnull().sum())
```

```
CNT_PAYMENT NULL COUNT: 372230
```

```
previous_application['CNT_PAYMENT'].describe()
```

```
count    1.297984e+06
mean     1.605408e+01
std      1.456729e+01
min      0.000000e+00
25%      6.000000e+00
50%      1.200000e+01
75%      2.400000e+01
max      8.400000e+01
Name: CNT_PAYMENT, dtype: float64
```

```
sns.set_style('whitegrid')
sns.boxplot(previous_application['CNT_PAYMENT'])
plt.show()
```



Suggestion

We can Fill NA with 12 i.e. Median for this field

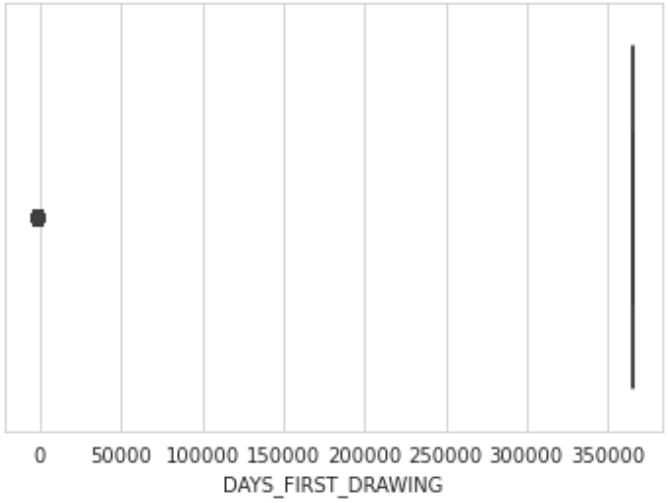
```
print("DAYS_FIRST_DRAWING :" ,previous_application['CNT_PAYMENT'].isnull().sum())
```

```
DAYS_FIRST_DRAWING : 372230
```

```
previous_application['DAYS_FIRST_DRAWING'].describe()
```

```
count    997149.000000
mean     342209.855039
std      88916.115834
min     -2922.000000
25%      365243.000000
50%      365243.000000
75%      365243.000000
max      365243.000000
Name: DAYS_FIRST_DRAWING, dtype: float64
```

```
sns.set_style('whitegrid')
sns.boxplot(previous_application['DAYS_FIRST_DRAWING'])
plt.show()
```



Suggestion

We can Fill NA with 365243 i.e. Median for this field

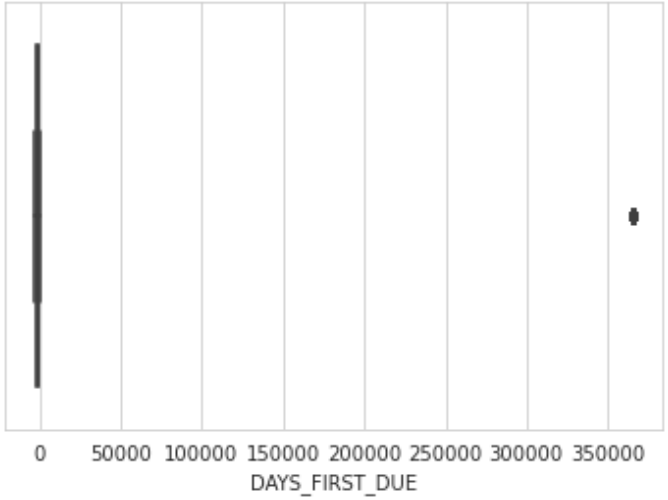
```
print("DAYS_FIRST_DUE :" ,previous_application['DAYS_FIRST_DUE'].isnull().sum())
```

```
DAYS_FIRST_DUE : 673065
```

```
previous_application['DAYS_FIRST_DUE'].describe()
```

```
count    997149.000000
mean      13826.269337
std       72444.869708
min      -2892.000000
25%      -1628.000000
50%      -831.000000
75%      -411.000000
max       365243.000000
Name: DAYS_FIRST_DUE, dtype: float64
```

```
sns.set_style('whitegrid')
sns.boxplot(previous_application['DAYS_FIRST_DUE'])
plt.show()
```



Suggestion

We can Fill NA with -831 i.e. Median for this field

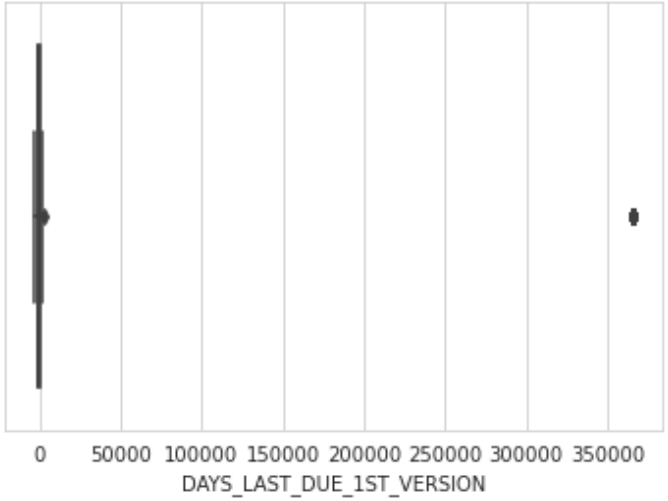
```
print("DAYS_LAST_DUE_1ST_VERSION :" ,previous_application['DAYS_LAST_DUE_1ST_VERSION'].isnull().sum())
```

```
DAYS_LAST_DUE_1ST_VERSION : 673065
```

```
previous_application['DAYS_LAST_DUE_1ST_VERSION'].describe()
```

```
count    997149.000000
mean      33767.774054
std      106857.034789
min     -2801.000000
25%     -1242.000000
50%     -361.000000
75%      129.000000
max      365243.000000
Name: DAYS_LAST_DUE_1ST_VERSION, dtype: float64
```

```
sns.set_style('whitegrid')
sns.boxplot(previous_application['DAYS_LAST_DUE_1ST_VERSION'])
plt.show()
```



Suggestion

We can Fill NA with -361 i.e. Median for this field

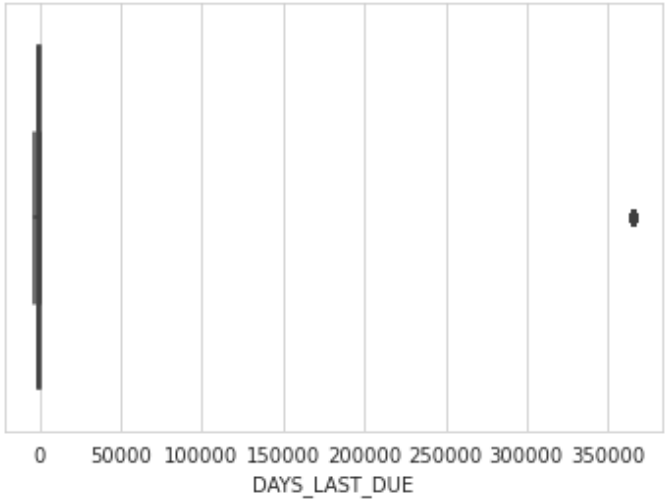
```
print("DAYS_LAST_DUE:" ,previous_application['DAYS_LAST_DUE'].isnull().sum())
```

```
DAYS_LAST_DUE: 673065
```

```
previous_application['DAYS_LAST_DUE'].describe()
```

```
count    997149.000000
mean      76582.403064
std      149647.415123
min       -2889.000000
25%       -1314.000000
50%       -537.000000
75%       -74.000000
max       365243.000000
Name: DAYS_LAST_DUE, dtype: float64
```

```
sns.set_style('whitegrid')
sns.boxplot(previous_application['DAYS_LAST_DUE'])
plt.show()
```



Suggestion

We can Fill NA with -537 i.e. Median for this field

```
print("DAYS_TERMINATION :" ,previous_application['DAYS_TERMINATION'].isnull().sum())
```

```
DAYS_TERMINATION : 673065
```

```
previous_application['DAYS_TERMINATION'].describe()
```

```
count    997149.000000
mean      81992.343838
std      153303.516729
min     -2874.000000
25%     -1270.000000
50%     -499.000000
75%     -44.000000
max      365243.000000
Name: DAYS_TERMINATION, dtype: float64
```

```
sns.set_style('whitegrid')
sns.boxplot(previous_application['DAYS_TERMINATION'])
plt.show()
```

Suggestion

We can Fill NA with -499 i.e. Median for this field

```
print("NFLAG_INSURED_ON_APPROVAL:" ,previous_application['NFLAG_INSURED_ON_APPROVAL'].isnull().sum())
```

```
NFLAG_INSURED_ON_APPROVAL: 673065
```

```
previous_application['NFLAG_INSURED_ON_APPROVAL'].value_counts()
```

```
0.0    665527
1.0    331622
Name: NFLAG_INSURED_ON_APPROVAL, dtype: int64
```

Suggestion

We can Fill NA with 0 i.e. Mode for this field

```
previous_application.isnull().sum()
```

```
SK_ID_PREV                    0
SK_ID_CURR                    0
NAME_CONTRACT_TYPE            0
AMT_ANNUITY                372235
AMT_APPLICATION               0
AMT_CREDIT                    1
AMT_GOODS_PRICE             385515
WEEKDAY_APPR_PROCESS_START    0
HOUR_APPR_PROCESS_START       0
FLAG_LAST_APPL_PER_CONTRACT   0
NFLAG_LAST_APPL_IN_DAY        0
NAME_CASH_LOAN_PURPOSE        0
NAME_CONTRACT_STATUS          0
```

```
DAYS_DECISION                 0
NAME_PAYMENT_TYPE             0
CODE_REJECT_REASON            0
NAME_TYPE_SUITE               820405
NAME_CLIENT_TYPE              0
NAME_GOODS_CATEGORY           0
NAME_PORTFOLIO                0
NAME_PRODUCT_TYPE             0
CHANNEL_TYPE                  0
SELLERPLACE_AREA              0
NAME_SELLER_INDUSTRY          0
CNT_PAYMENT                   372230
NAME_YIELD_GROUP              0
PRODUCT_COMBINATION           346
DAYS_FIRST_DRAWING            673065
DAYS_FIRST_DUE                673065
DAYS_LAST_DUE_1ST_VERSION     673065
DAYS_LAST_DUE                 673065
DAYS_TERMINATION              673065
NFLAG_INSURED_ON_APPROVAL     673065
dtype: int64
```

```
print("AMT_CREDIT :" ,previous_application['AMT_CREDIT'].isnull().sum())
```

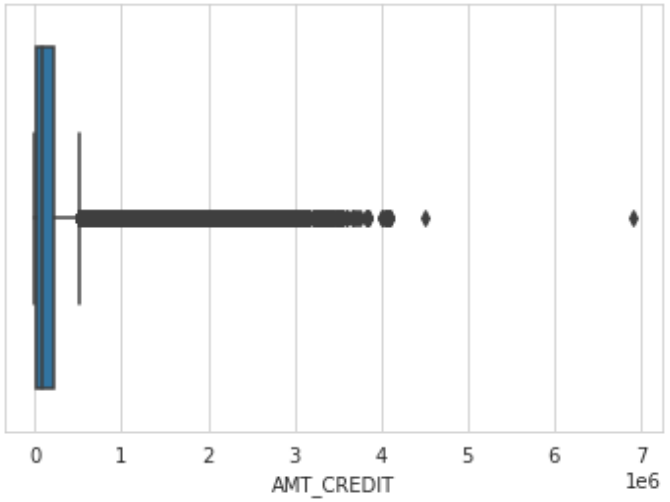
```
AMT_CREDIT : 1
```

```
previous_application['AMT_CREDIT'].describe()
```

```
count    1.670213e+06
mean     1.961140e+05
std      3.185746e+05
min      0.000000e+00
25%      2.416050e+04
50%      8.054100e+04
75%      2.164185e+05
max      6.905160e+06
Name: AMT_CREDIT, dtype: float64
```

```
sns.set_style('whitegrid')
sns.boxplot(previous_application['AMT_CREDIT'])
```

```
plt.show()
```



Suggestion

We can Fill NA with 80541 i.e. Median for this field

```
print("PRODUCT_COMBINATION :" ,previous_application['PRODUCT_COMBINATION'].isnull().sum())
```

```
PRODUCT_COMBINATION : 346
```

```
previous_application['PRODUCT_COMBINATION'].value_counts()
```

```
Cash                              285990
POS household with interest       263622
POS mobile with interest          220670
Cash X-Sell: middle               143883
Cash X-Sell: low                  130248
Card Street                       112582
POS industry with interest         98833
POS household without interest     82908
Card X-Sell                        80582
```

```
Cash Street: high                 59639
Cash X-Sell: high                 59301
Cash Street: middle               34658
Cash Street: low                  33834
POS mobile without interest       24082
POS other with interest           23879
POS industry without interest     12602
POS others without interest       2555
Name: PRODUCT_COMBINATION, dtype: int64
```

Suggestion

We can Fill NA with Cash i.e. Mode for this field

```
class color:
   PURPLE = '\033[95m'
   CYAN = '\033[96m'
   DARKCYAN = '\033[36m'
   BLUE = '\033[94m'
   GREEN = '\033[92m'
   YELLOW = '\033[93m'
   RED = '\033[91m'
   BOLD = '\033[1m'
   UNDERLINE = '\033[4m'
   END = '\033[0m'
```

Separating numerical and categorical columns from previous_application

```
obj_dtypes = [i for i in previous_application.select_dtypes(include=np.object).columns if i not in ["type"] ]
num_dtypes = [i for i in previous_application.select_dtypes(include = np.number).columns if i not in ['SK_ID_CURR'] + [ 'TARGET']]
```

```
print(color.BOLD + color.PURPLE + 'Categorical Columns' + color.END, "\n")
for x in range(len(obj_dtypes)):
    print(obj_dtypes[x])
```

Categorical Columns

NAME_CONTRACT_TYPE
WEEKDAY_APPR_PROCESS_START
FLAG_LAST_APPL_PER_CONTRACT
NAME_CASH_LOAN_PURPOSE
NAME_CONTRACT_STATUS
NAME_PAYMENT_TYPE
CODE_REJECT_REASON
NAME_TYPE_SUITE
NAME_CLIENT_TYPE
NAME_GOODS_CATEGORY
NAME_PORTFOLIO
NAME_PRODUCT_TYPE
CHANNEL_TYPE
NAME_SELLER_INDUSTRY
NAME_YIELD_GROUP
PRODUCT_COMBINATION

```
print(color.BOLD + color.PURPLE + 'Numerical' + color.END, "\n")
for x in range(len(obj_dtypes)):
    print(obj_dtypes[x])
```

Numerical

NAME_CONTRACT_TYPE
WEEKDAY_APPR_PROCESS_START
FLAG_LAST_APPL_PER_CONTRACT
NAME_CASH_LOAN_PURPOSE
NAME_CONTRACT_STATUS
NAME_PAYMENT_TYPE
CODE_REJECT_REASON
NAME_TYPE_SUITE
NAME_CLIENT_TYPE
NAME_GOODS_CATEGORY
NAME_PORTFOLIO
NAME_PRODUCT_TYPE
CHANNEL_TYPE
NAME_SELLER_INDUSTRY

```
NAME_YIELD_GROUP
PRODUCT_COMBINATION
```

Percentage of Missing values in application_data

```
fig = plt.figure(figsize=(18,6))
miss_application_data = pd.DataFrame((application_data.isnull().sum())*100/application_data.shape[0]).reset_index()
miss_application_data["type"] = "application_data"
ax = sns.pointplot("index",0,data=miss_application_data,hue="type")
plt.xticks(rotation =90,fontsize =7)
plt.title("Percentage of Missing values in application_data")
plt.ylabel("PERCENTAGE")
plt.xlabel("COLUMNS")
ax.set_facecolor("k")
fig.set_facecolor("lightgrey")
```


```
round(100*(application_data.isnull().sum()/len(application_data.index)),2)
```

```
ENTRANCES_MODE                  50.35
FLOORSMAX_MODE                  49.76
FLOORSMIN_MODE                  67.85
LANDAREA_MODE                   59.38
LIVINGAPARTMENTS_MODE           68.35
LIVINGAREA_MODE                 50.19
NONLIVINGAPARTMENTS_MODE        69.43
NONLIVINGAREA_MODE              55.18
APARTMENTS_MEDI                 50.75
BASEMENTAREA_MEDI               58.52
YEARS_BEGINEXPLUATATION_MEDI    48.78
YEARS_BUILD_MEDI                66.50
COMMONAREA_MEDI                 69.87
ELEVATORS_MEDI                  53.30
ENTRANCES_MEDI                  50.35
FLOORSMAX_MEDI                  49.76
FLOORSMIN_MEDI                  67.85
LANDAREA_MEDI                   59.38
LIVINGAPARTMENTS_MEDI           68.35
LIVINGAREA_MEDI                 50.19
NONLIVINGAPARTMENTS_MEDI        69.43
NONLIVINGAREA_MEDI              55.18
FONDKAPREMONT_MODE              68.39
HOUSETYPE_MODE                  50.18
TOTALAREA_MODE                  48.27
WALLSMATERIAL_MODE              50.84
EMERGENCYSTATE_MODE             47.40
OBS_30_CNT_SOCIAL_CIRCLE        0.33
DEF_30_CNT_SOCIAL_CIRCLE        0.33
OBS_60_CNT_SOCIAL_CIRCLE        0.33
DEF_60_CNT_SOCIAL_CIRCLE        0.33
DAYS_LAST_PHONE_CHANGE          0.00
FLAG_DOCUMENT_2                 0.00
FLAG_DOCUMENT_3                 0.00
```

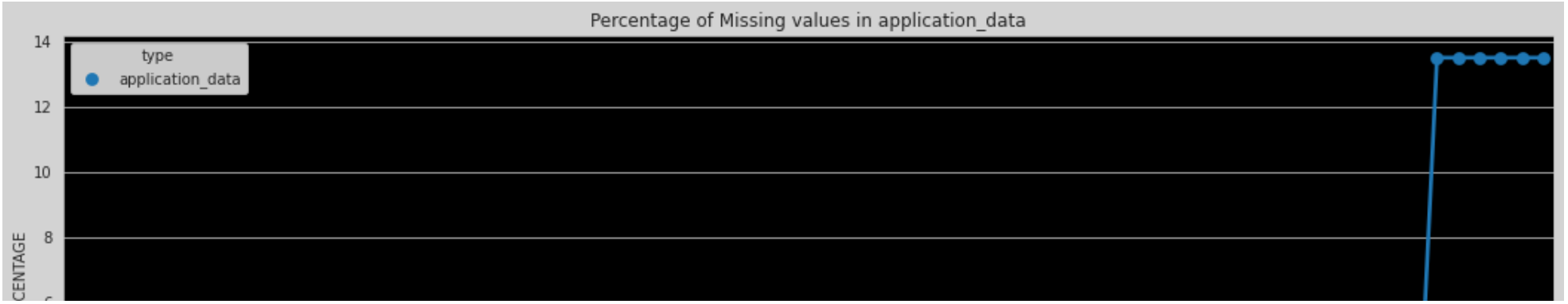
```
FLAG_DOCUMENT_4                0.00
FLAG_DOCUMENT_5                0.00
FLAG_DOCUMENT_6                0.00
FLAG_DOCUMENT_7                0.00
FLAG_DOCUMENT_8                0.00
FLAG_DOCUMENT_9                0.00
FLAG_DOCUMENT_10               0.00
FLAG_DOCUMENT_11               0.00
FLAG_DOCUMENT_12               0.00
FLAG_DOCUMENT_13               0.00
FLAG_DOCUMENT_14               0.00
FLAG_DOCUMENT_15               0.00
FLAG_DOCUMENT_16               0.00
FLAG_DOCUMENT_17               0.00
FLAG_DOCUMENT_18               0.00
FLAG_DOCUMENT_19               0.00
FLAG_DOCUMENT_20               0.00
FLAG_DOCUMENT_21               0.00
AMT_REQ_CREDIT_BUREAU_HOUR     13.50
AMT_REQ_CREDIT_BUREAU_DAY      13.50
AMT_REQ_CREDIT_BUREAU_WEEK     13.50
AMT_REQ_CREDIT_BUREAU_MON      13.50
AMT_REQ_CREDIT_BUREAU_QRT      13.50
AMT_REQ_CREDIT_BUREAU_YEAR     13.50
dtype: float64
```

Removing columns with missing values more than 40%

As per Industrial Standard, max Threshold limit can be between 40% to 50 % depending upon the data acquired in specific sector.

```
application_data=application_data.drop([ 'EXT_SOURCE_1', 'EXT_SOURCE_2', 'EXT_SOURCE_3',
       'APARTMENTS_AVG', 'BASEMENTAREA_AVG', 'YEARS_BEGINEXPLUATATION_AVG',
       'YEARS_BUILD_AVG', 'COMMONAREA_AVG', 'ELEVATORS_AVG', 'ENTRANCES_AVG',
       'FLOORSMAX_AVG', 'FLOORSMIN_AVG', 'LANDAREA_AVG',
       'LIVINGAPARTMENTS_AVG', 'LIVINGAREA_AVG', 'NONLIVINGAPARTMENTS_AVG',
       'NONLIVINGAREA_AVG', 'APARTMENTS_MODE', 'BASEMENTAREA_MODE',
       'YEARS_BEGINEXPLUATATION_MODE', 'YEARS_BUILD_MODE', 'COMMONAREA_MODE',
       'ELEVATORS_MODE', 'ENTRANCES_MODE', 'FLOORSMAX_MODE', 'FLOORSMIN_MODE',
       'LANDAREA_MODE', 'LIVINGAPARTMENTS_MODE', 'LIVINGAREA_MODE',
```

```
        'NONLIVINGAPARTMENTS_MODE', 'NONLIVINGAREA_MODE', 'APARTMENTS_MEDI',
        'BASEMENTAREA_MEDI', 'YEARS_BEGINEXPLUATATION_MEDI', 'YEARS_BUILD_MEDI',
        'COMMONAREA_MEDI', 'ELEVATORS_MEDI', 'ENTRANCES_MEDI', 'FLOORSMAX_MEDI',
        'FLOORSMIN_MEDI', 'LANDAREA_MEDI', 'LIVINGAPARTMENTS_MEDI',
        'LIVINGAREA_MEDI', 'NONLIVINGAPARTMENTS_MEDI', 'NONLIVINGAREA_MEDI',
        'FONDKAPREMONT_MODE', 'HOUSETYPE_MODE', 'TOTALAREA_MODE',
        'WALLSMATERIAL_MODE', 'EMERGENCYSTATE_MODE',"OWN_CAR_AGE","OCCUPATION_TYPE"],axis=1)


fig = plt.figure(figsize=(18,6))
miss_application_data = pd.DataFrame((application_data.isnull().sum())*100/application_data.shape[0]).reset_index()
miss_application_data["type"] = "application_data"
ax = sns.pointplot("index",0,data=miss_application_data,hue="type")
plt.xticks(rotation =90,fontsize =7)
plt.title("Percentage of Missing values in application_data")
plt.ylabel("PERCENTAGE")
plt.xlabel("COLUMNS")
ax.set_facecolor("k")
fig.set_facecolor("lightgrey")
```




```
round(100*(application_data.isnull().sum()/len(application_data.index)),2)
```

```
NAME_INCOME_TYPE              0.00
NAME_EDUCATION_TYPE           0.00
NAME_FAMILY_STATUS            0.00
NAME_HOUSING_TYPE             0.00
REGION_POPULATION_RELATIVE    0.00
DAYS_BIRTH                    0.00
DAYS_EMPLOYED                 0.00
DAYS_REGISTRATION             0.00
DAYS_ID_PUBLISH               0.00
FLAG_MOBIL                    0.00
FLAG_EMP_PHONE                0.00
FLAG_WORK_PHONE               0.00
FLAG_CONT_MOBILE              0.00
FLAG_PHONE                    0.00
FLAG_EMAIL                    0.00
CNT_FAM_MEMBERS               0.00

REGION_RATING_CLIENT          0.00
REGION_RATING_CLIENT_W_CITY   0.00
WEEKDAY_APPR_PROCESS_START    0.00
HOUR_APPR_PROCESS_START       0.00
REG_REGION_NOT_LIVE_REGION    0.00
REG_REGION_NOT_WORK_REGION    0.00
LIVE_REGION_NOT_WORK_REGION   0.00
REG_CITY_NOT_LIVE_CITY        0.00
REG_CITY_NOT_WORK_CITY        0.00
LIVE_CITY_NOT_WORK_CITY       0.00
ORGANIZATION_TYPE             0.00
OBS_30_CNT_SOCIAL_CIRCLE      0.33
DEF_30_CNT_SOCIAL_CIRCLE      0.33
```

```
OBS_60_CNT_SOCIAL_CIRCLE       0.33
DEF_60_CNT_SOCIAL_CIRCLE       0.33
DAYS_LAST_PHONE_CHANGE         0.00
FLAG_DOCUMENT_2                0.00
FLAG_DOCUMENT_3                0.00
FLAG_DOCUMENT_4                0.00
FLAG_DOCUMENT_5                0.00
FLAG_DOCUMENT_6                0.00
FLAG_DOCUMENT_7                0.00
FLAG_DOCUMENT_8                0.00
FLAG_DOCUMENT_9                0.00
FLAG_DOCUMENT_10               0.00
FLAG_DOCUMENT_11               0.00
FLAG_DOCUMENT_12               0.00
FLAG_DOCUMENT_13               0.00
FLAG_DOCUMENT_14               0.00
FLAG_DOCUMENT_15               0.00
FLAG_DOCUMENT_16               0.00
FLAG_DOCUMENT_17               0.00
FLAG_DOCUMENT_18               0.00
FLAG_DOCUMENT_19               0.00
FLAG_DOCUMENT_20               0.00
FLAG_DOCUMENT_21               0.00
AMT_REQ_CREDIT_BUREAU_HOUR    13.50
AMT_REQ_CREDIT_BUREAU_DAY     13.50
AMT_REQ_CREDIT_BUREAU_WEEK    13.50
AMT_REQ_CREDIT_BUREAU_MON     13.50
AMT_REQ_CREDIT_BUREAU_QRT     13.50
AMT_REQ_CREDIT_BUREAU_YEAR    13.50
dtype: float64
```

MISSING values Suggestion

```
print("AMT_REQ_CREDIT_BUREAU_DAY NAN COUNT :" ,application_data['AMT_REQ_CREDIT_BUREAU_DAY'].isnull().sum())
```

```
AMT_REQ_CREDIT_BUREAU_DAY NAN COUNT : 41519
```

```
application_data['AMT_REQ_CREDIT_BUREAU_DAY'].describe()
```

```
count    265992.000000
mean     0.007000
std      0.110757
min      0.000000
25%      0.000000
50%      0.000000
75%      0.000000
max      9.000000
Name: AMT_REQ_CREDIT_BUREAU_DAY, dtype: float64
```

Suggestion

We can Fill NA with 0 i.e. Median for this field

```
print("AMT_REQ_CREDIT_BUREAU_HOUR NAN COUNT :" ,application_data['AMT_REQ_CREDIT_BUREAU_HOUR'].isnull().sum())
```

```
AMT_REQ_CREDIT_BUREAU_HOUR NAN COUNT : 41519
```

```
application_data['AMT_REQ_CREDIT_BUREAU_HOUR'].describe()
```

```
count    265992.000000
mean     0.006402
std      0.083849
min      0.000000
25%      0.000000
50%      0.000000
75%      0.000000
max      4.000000
Name: AMT_REQ_CREDIT_BUREAU_HOUR, dtype: float64
```

Suggestion

We can Fill NA with 0 i.e. Median for this field

```
print("AMT_REQ_CREDIT_BUREAU_MON NAN COUNT :" ,application_data['AMT_REQ_CREDIT_BUREAU_MON'].isnull().sum())
```

```
AMT_REQ_CREDIT_BUREAU_MON NAN COUNT : 41519
```

```
application_data['AMT_REQ_CREDIT_BUREAU_MON'].describe()
```

```
count    265992.000000
mean     0.267395
std      0.916002
min      0.000000
25%      0.000000
50%      0.000000
75%      0.000000
max      27.000000
Name: AMT_REQ_CREDIT_BUREAU_MON, dtype: float64
```

Suggestion

We can Fill NA with 0 i.e. Median for this field

```
print("AMT_REQ_CREDIT_BUREAU_QRT NAN COUNT :" ,application_data['AMT_REQ_CREDIT_BUREAU_QRT'].isnull().sum())
```

```
AMT_REQ_CREDIT_BUREAU_QRT NAN COUNT : 41519
```

Suggestion

We can Fill NA with 0 i.e. Median for this field

```
print("AMT_REQ_CREDIT_BUREAU_WEEK NAN COUNT :" ,application_data['AMT_REQ_CREDIT_BUREAU_WEEK'].isnull().sum())
```

```
AMT_REQ_CREDIT_BUREAU_WEEK NAN COUNT : 41519
```

```
application_data['AMT_REQ_CREDIT_BUREAU_WEEK'].describe()
```

```
count    265992.000000
mean     0.034362
```

```
std      0.204685
min      0.000000
25%      0.000000
50%      0.000000
75%      0.000000
max      8.000000
Name: AMT_REQ_CREDIT_BUREAU_WEEK, dtype: float64
```

Suggestion

We can Fill NA with 0 i.e. Median for this field

```
print("AMT_REQ_CREDIT_BUREAU_YEAR NAN COUNT :" ,application_data['AMT_REQ_CREDIT_BUREAU_YEAR'].isnull().sum())
```

```
AMT_REQ_CREDIT_BUREAU_YEAR NAN COUNT : 41519
```

```
application_data['AMT_REQ_CREDIT_BUREAU_YEAR'].describe()
```

```
count    265992.000000
mean     1.899974
std      1.869295
min      0.000000
25%      0.000000
50%      1.000000
75%      3.000000
max      25.000000
Name: AMT_REQ_CREDIT_BUREAU_YEAR, dtype: float64
```

Suggestion

We can Fill NA with 0 i.e. Median for this field

```
print("DEF_30_CNT_SOCIAL_CIRCLE NAN COUNT :" ,application_data['DEF_30_CNT_SOCIAL_CIRCLE'].isnull().sum())
```

```
DEF_30_CNT_SOCIAL_CIRCLE NAN COUNT : 1021
```

```
application_data['DEF_30_CNT_SOCIAL_CIRCLE'].describe()
```

```
count    306490.000000
mean     0.143421
std      0.446698
min      0.000000
25%      0.000000
50%      0.000000
75%      0.000000
max      34.000000
Name: DEF_30_CNT_SOCIAL_CIRCLE, dtype: float64
```

Suggestion

We can Fill NA with 0 i.e. Median for this field

```
print("DEF_30_CNT_SOCIAL_CIRCLE :" ,application_data['DEF_30_CNT_SOCIAL_CIRCLE'].isnull().sum())
```

```
DEF_30_CNT_SOCIAL_CIRCLE : 1021
```

```
application_data['DEF_30_CNT_SOCIAL_CIRCLE'].describe()
```

```
count    306490.000000
mean     0.143421
std      0.446698
min      0.000000
25%      0.000000
50%      0.000000
75%      0.000000
max      34.000000
Name: DEF_30_CNT_SOCIAL_CIRCLE, dtype: float64
```

Suggestion

We can Fill NA with 0 i.e. Median for this field

```
print("OBS_60_CNT_SOCIAL_CIRCLE :" ,application_data['OBS_60_CNT_SOCIAL_CIRCLE'].isnull().sum())
```

```
OBS_60_CNT_SOCIAL_CIRCLE : 1021
```

```
application_data['OBS_60_CNT_SOCIAL_CIRCLE'].describe()
```

```
count    306490.000000
mean     1.405292
std      2.379803
min      0.000000
25%      0.000000
50%      0.000000
75%      2.000000
max      344.000000
Name: OBS_60_CNT_SOCIAL_CIRCLE, dtype: float64
```

Suggestion

We can Fill NA with 0 i.e. Median for this field

```
print("DEF_60_CNT_SOCIAL_CIRCLE :" ,application_data['DEF_60_CNT_SOCIAL_CIRCLE'].isnull().sum())
```

```
DEF_60_CNT_SOCIAL_CIRCLE : 1021
```

```
application_data['DEF_60_CNT_SOCIAL_CIRCLE'].describe()
```

```
count    306490.000000
mean     0.100049
std      0.362291
min      0.000000
25%      0.000000
50%      0.000000
75%      0.000000
max      24.000000
Name: DEF_60_CNT_SOCIAL_CIRCLE, dtype: float64
```

Suggestion

We can Fill NA with 0 i.e. Median for this field

```
application_data.isnull().sum()
```

```
NAME_INCOME_TYPE               0
NAME_EDUCATION_TYPE            0
NAME_FAMILY_STATUS             0
NAME_HOUSING_TYPE              0
REGION_POPULATION_RELATIVE     0
DAYS_BIRTH                     0
DAYS_EMPLOYED                  0
DAYS_REGISTRATION              0
DAYS_ID_PUBLISH                0
FLAG_MOBIL                     0
FLAG_EMP_PHONE                 0
FLAG_WORK_PHONE                0
FLAG_CONT_MOBILE               0
FLAG_PHONE                     0
FLAG_EMAIL                     0
CNT_FAM_MEMBERS                2
REGION_RATING_CLIENT           0
REGION_RATING_CLIENT_W_CITY    0
WEEKDAY_APPR_PROCESS_START     0
HOUR_APPR_PROCESS_START        0
REG_REGION_NOT_LIVE_REGION     0
REG_REGION_NOT_WORK_REGION     0
LIVE_REGION_NOT_WORK_REGION    0
REG_CITY_NOT_LIVE_CITY         0
REG_CITY_NOT_WORK_CITY         0
LIVE_CITY_NOT_WORK_CITY        0
ORGANIZATION_TYPE              0
OBS_30_CNT_SOCIAL_CIRCLE       1021
DEF_30_CNT_SOCIAL_CIRCLE       1021
OBS_60_CNT_SOCIAL_CIRCLE       1021
DEF_60_CNT_SOCIAL_CIRCLE       1021
DAYS_LAST_PHONE_CHANGE         1
FLAG_DOCUMENT_2                0
FLAG_DOCUMENT_3                0
```

```
FLAG_DOCUMENT_4               0
FLAG_DOCUMENT_5               0
FLAG_DOCUMENT_6               0
FLAG_DOCUMENT_7               0
FLAG_DOCUMENT_8               0
FLAG_DOCUMENT_9               0
FLAG_DOCUMENT_10              0
FLAG_DOCUMENT_11              0
FLAG_DOCUMENT_12              0
FLAG_DOCUMENT_13              0
FLAG_DOCUMENT_14              0
FLAG_DOCUMENT_15              0
FLAG_DOCUMENT_16              0
FLAG_DOCUMENT_17              0
FLAG_DOCUMENT_18              0
FLAG_DOCUMENT_19              0
FLAG_DOCUMENT_20              0
FLAG_DOCUMENT_21              0
AMT_REQ_CREDIT_BUREAU_HOUR    41519
AMT_REQ_CREDIT_BUREAU_DAY     41519
AMT_REQ_CREDIT_BUREAU_WEEK    41519
AMT_REQ_CREDIT_BUREAU_MON     41519
AMT_REQ_CREDIT_BUREAU_QRT     41519
AMT_REQ_CREDIT_BUREAU_YEAR    41519
dtype: int64
```

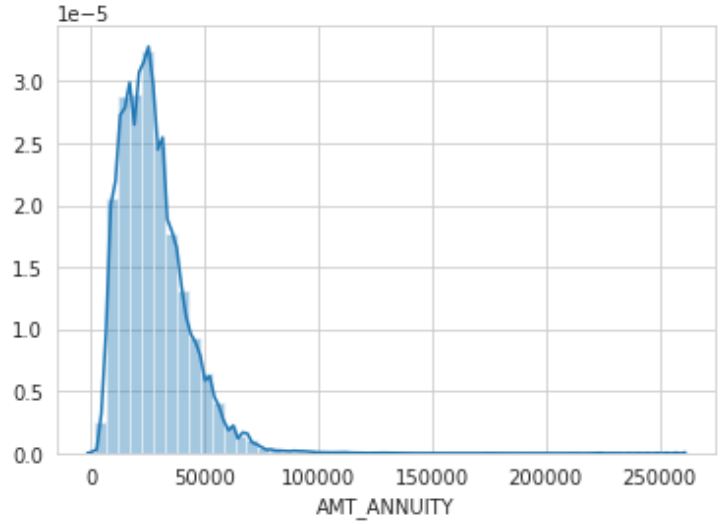
```
print("AMT_ANNUITY  :" ,application_data['AMT_ANNUITY'].isnull().sum())
```

```
AMT_ANNUITY  : 12
```

```
application_data['AMT_ANNUITY'].describe()
```

```
count    307499.000000
mean      27108.573909
std       14493.737315
min        1615.500000
25%       16524.000000
50%       24903.000000
75%       34596.000000
max      258025.500000
Name: AMT_ANNUITY, dtype: float64
```

```
sns.set_style('whitegrid')
sns.distplot(application_data['AMT_ANNUITY'])
plt.show()
```



Suggestion

We can Fill NA with 0 i.e. Mean for this field as it's right skewed graph

```
print("AMT_GOODS_PRICE   :" ,application_data['AMT_GOODS_PRICE'].isnull().sum())
```

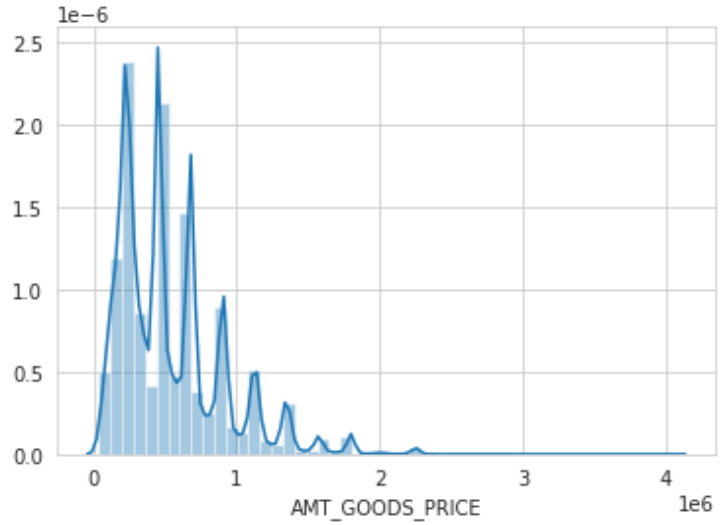
```
AMT_GOODS_PRICE   : 278
```

```
application_data['AMT_GOODS_PRICE'].describe()
```

```
count    3.072330e+05
mean     5.383962e+05
std      3.694465e+05
min      4.050000e+04
25%      2.385000e+05
```

```
50%      4.500000e+05
75%      6.795000e+05
max      4.050000e+06
Name: AMT_GOODS_PRICE, dtype: float64
```

```
sns.set_style('whitegrid')
sns.distplot(application_data['AMT_GOODS_PRICE'])
plt.show()
```



Suggestion

We can Fill NA with 0 i.e. Mean for this field as it's right skewed graph

```
print("NAME_TYPE_SUITE :" ,application_data['NAME_TYPE_SUITE'].isnull().sum())
```

```
NAME_TYPE_SUITE : 1292
```

```
application_data['NAME_TYPE_SUITE'].value_counts()
```

```
Unaccompanied      248526
```

```
Family              40149
Spouse, partner     11370
Children             3267
Other_B              1770
Other_A               866
Group of people       271
Name: NAME_TYPE_SUITE, dtype: int64
```

Suggestion

We can Fill NA with "Unaccompanied" i.e. Mode for this field

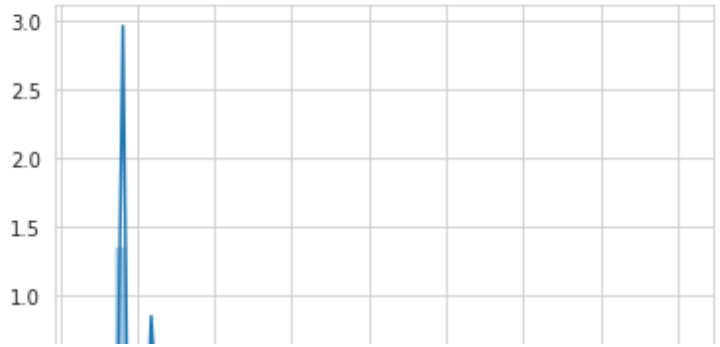
```
print("CNT_FAM_MEMBERS :" ,application_data['CNT_FAM_MEMBERS'].isnull().sum())
```

```
CNT_FAM_MEMBERS : 2
```

```
application_data['CNT_FAM_MEMBERS'].describe()
```

```
count    307509.000000
mean          2.152665
std           0.910682
min           1.000000
25%           2.000000
50%           2.000000
75%           3.000000
max          20.000000
Name: CNT_FAM_MEMBERS, dtype: float64
```

```
sns.set_style('whitegrid')
sns.distplot(application_data['CNT_FAM_MEMBERS'])
plt.show()
```



Suggestion

We can Fill NA with 2 i.e. Median for this field, Mean is not be used as this field needs to be Whole number

```
print("DAYS_LAST_PHONE_CHANGE :" ,application_data['DAYS_LAST_PHONE_CHANGE'].isnull().sum())
```

```
DAYS_LAST_PHONE_CHANGE : 1
```

```
application_data['DAYS_LAST_PHONE_CHANGE'].describe()
```

```
count    307510.000000
mean    -962.858788
std      826.808487
min     -4292.000000
25%     -1570.000000
50%     -757.000000
75%     -274.000000
max      0.000000
Name: DAYS_LAST_PHONE_CHANGE, dtype: float64
```

```
import statistics
statistics.mode(application_data['DAYS_LAST_PHONE_CHANGE'])
```

```
0.0
```

Suggestion

We can Fill NA with 0 i.e. Mode for this field

Print the information about the attributes of application_data

```
print(type(application_data.info()))
```

```
      14  NAME_FAMILY_STATUS           307511 non-null  object
      15  NAME_HOUSING_TYPE            307511 non-null  object
      16  REGION_POPULATION_RELATIVE   307511 non-null  float64
      17  DAYS_BIRTH                   307511 non-null  int64
      18  DAYS_EMPLOYED                307511 non-null  int64
      19  DAYS_REGISTRATION            307511 non-null  float64
      20  DAYS_ID_PUBLISH              307511 non-null  int64
      21  FLAG_MOBIL                   307511 non-null  int64
      22  FLAG_EMP_PHONE               307511 non-null  int64
      23  FLAG_WORK_PHONE              307511 non-null  int64
      24  FLAG_CONT_MOBILE             307511 non-null  int64
      25  FLAG_PHONE                   307511 non-null  int64
      26  FLAG_EMAIL                   307511 non-null  int64
      27  CNT_FAM_MEMBERS              307509 non-null  float64
      28  REGION_RATING_CLIENT         307511 non-null  int64
      29  REGION_RATING_CLIENT_W_CITY  307511 non-null  int64
      30  WEEKDAY_APPR_PROCESS_START   307511 non-null  object
      31  HOUR_APPR_PROCESS_START      307511 non-null  int64
      32  REG_REGION_NOT_LIVE_REGION   307511 non-null  int64
      33  REG_REGION_NOT_WORK_REGION   307511 non-null  int64
      34  LIVE_REGION_NOT_WORK_REGION  307511 non-null  int64
      35  REG_CITY_NOT_LIVE_CITY       307511 non-null  int64
      36  REG_CITY_NOT_WORK_CITY       307511 non-null  int64
      37  LIVE_CITY_NOT_WORK_CITY      307511 non-null  int64
      38  ORGANIZATION_TYPE            307511 non-null  object
      39  OBS_30_CNT_SOCIAL_CIRCLE     306490 non-null  float64
      40  DEF_30_CNT_SOCIAL_CIRCLE     306490 non-null  float64
      41  OBS_60_CNT_SOCIAL_CIRCLE     306490 non-null  float64
      42  DEF_60_CNT_SOCIAL_CIRCLE     306490 non-null  float64
      43  DAYS_LAST_PHONE_CHANGE       307510 non-null  float64
      44  FLAG_DOCUMENT_2              307511 non-null  int64
      45  FLAG_DOCUMENT_3              307511 non-null  int64
```

```
 46  FLAG_DOCUMENT_4              307511 non-null  int64
 47  FLAG_DOCUMENT_5              307511 non-null  int64
 48  FLAG_DOCUMENT_6              307511 non-null  int64
 49  FLAG_DOCUMENT_7              307511 non-null  int64
 50  FLAG_DOCUMENT_8              307511 non-null  int64
 51  FLAG_DOCUMENT_9              307511 non-null  int64
 52  FLAG_DOCUMENT_10             307511 non-null  int64
 53  FLAG_DOCUMENT_11             307511 non-null  int64
 54  FLAG_DOCUMENT_12             307511 non-null  int64
 55  FLAG_DOCUMENT_13             307511 non-null  int64
 56  FLAG_DOCUMENT_14             307511 non-null  int64
 57  FLAG_DOCUMENT_15             307511 non-null  int64
 58  FLAG_DOCUMENT_16             307511 non-null  int64
 59  FLAG_DOCUMENT_17             307511 non-null  int64
 60  FLAG_DOCUMENT_18             307511 non-null  int64
 61  FLAG_DOCUMENT_19             307511 non-null  int64
 62  FLAG_DOCUMENT_20             307511 non-null  int64
 63  FLAG_DOCUMENT_21             307511 non-null  int64
 64  AMT_REQ_CREDIT_BUREAU_HOUR   265992 non-null  float64
 65  AMT_REQ_CREDIT_BUREAU_DAY    265992 non-null  float64
 66  AMT_REQ_CREDIT_BUREAU_WEEK   265992 non-null  float64
 67  AMT_REQ_CREDIT_BUREAU_MON    265992 non-null  float64
 68  AMT_REQ_CREDIT_BUREAU_QRT    265992 non-null  float64
 69  AMT_REQ_CREDIT_BUREAU_YEAR   265992 non-null  float64
dtypes: float64(18), int64(41), object(11)
memory usage: 164.2+ MB
<class 'NoneType'>
```

Converting negative values to absolute values

```
application_data['DAYS_BIRTH'] = abs(application_data['DAYS_BIRTH'])
application_data['DAYS_ID_PUBLISH'] = abs(application_data['DAYS_ID_PUBLISH'])
application_data['DAYS_ID_PUBLISH'] = abs(application_data['DAYS_ID_PUBLISH'])
application_data['DAYS_LAST_PHONE_CHANGE'] = abs(application_data['DAYS_LAST_PHONE_CHANGE'])
```

```
display("application_data")
```

```
display(application_data.head())
```

'application_data'

	SK_ID_CURR	TARGET	NAME_CONTRACT_TYPE	CODE_GENDER	FLAG_OWN_CAR	FLAG_OWN_REALTY	CNT_CHILDREN	AMT_INCOME_TOTAL	AMT_CREDIT
0	100002	1	Cash loans	M	N	Y	0	202500.0	406597.5
1	100003	0	Cash loans	F	N	N	0	270000.0	1293502.5
2	100004	0	Revolving loans	M	Y	Y	0	67500.0	135000.0
3	100006	0	Cash loans	F	N	Y	0	135000.0	312682.5
4	100007	0	Cash loans	M	N	Y	0	121500.0	513000.0

Separating numerical and categorical in application_data

```
obj_dtypes = [i for i in application_data.select_dtypes(include=np.object).columns if i not in ["type"] ]
num_dtypes = [i for i in application_data.select_dtypes(include = np.number).columns if i not in ['SK_ID_CURR'] + [ 'TARGET']]
```

```
print(color.BOLD + color.PURPLE + 'Categorical Columns' + color.END, "\n")
for x in range(len(obj_dtypes)):
    print(obj_dtypes[x])
```

Categorical Columns

NAME_CONTRACT_TYPE
CODE_GENDER
FLAG_OWN_CAR
FLAG_OWN_REALTY
NAME_TYPE_SUITE
NAME_INCOME_TYPE
NAME_EDUCATION_TYPE

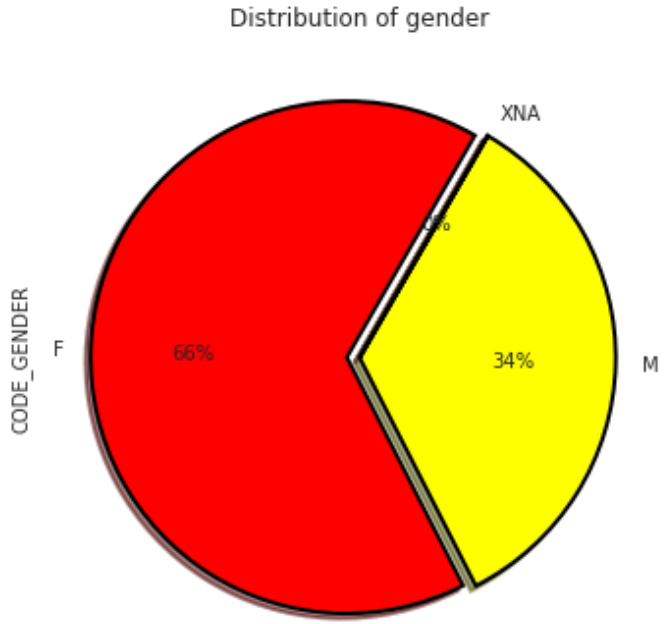
```
      NAME_FAMILY_STATUS
      NAME_HOUSING_TYPE
      WEEKDAY_APPR_PROCESS_START
      ORGANIZATION_TYPE


print(color.BOLD + color.PURPLE +"Numerical Columns" + color.END, "\n")
for x in range(len(num_dtypes)):
    print(num_dtypes[x])
```

```
Numerical Columns

CNT_CHILDREN
AMT_INCOME_TOTAL
AMT_CREDIT
AMT_ANNUITY
AMT_GOODS_PRICE
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
HOUR_APPR_PROCESS_START
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
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
```

Imbalance percentage

```
fig = plt.figure(figsize=(13,6))
plt.subplot(121)
application_data["CODE_GENDER"].value_counts().plot.pie(autopct = "%1.0f%%",colors = ["red","yellow"],startangle = 60,
                                                        wedgeprops={"linewidth":2,"edgecolor":"k"},explode=[.05,0,0],sha
plt.title("Distribution of gender")
plt.show()
```



Point to infer from the graph

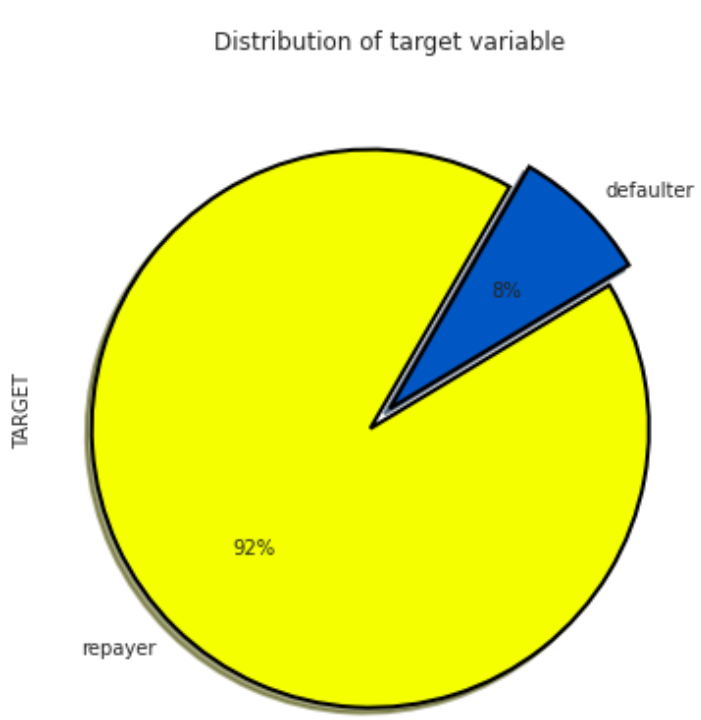
It's non balanced data

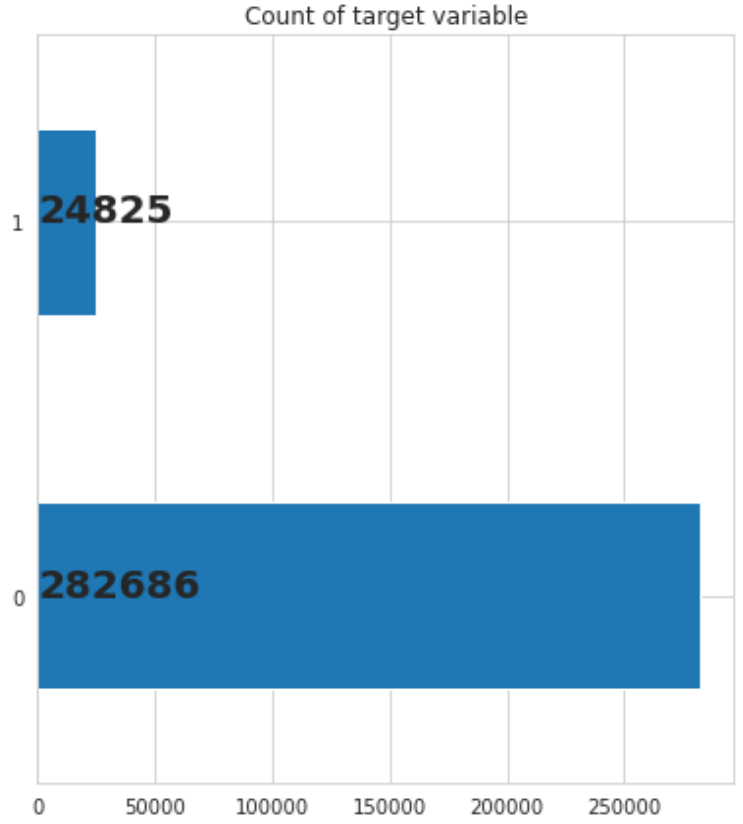
Distribution of Target variable

TARGET :Target variable (1 - client with payment difficulties: he/she had late payment more than X days on at least one of the first Y installments of the loan in sample, 0 - all other cases)

```
plt.figure(figsize=(14,7))
plt.subplot(121)
application_data["TARGET"].value_counts().plot.pie(autopct = "%1.0f%%",colors = sns.color_palette("prism",7),startangle = 60,labels=["re
                                                                                    wedgeprops={"linewidth":2,"edgecolor":"k"},explode=[.1,0],shadow
plt.title("Distribution of target variable")

plt.subplot(122)
```

```
ax = application_data["TARGET"].value_counts().plot(kind="barh")

for i,j in enumerate(application_data["TARGET"].value_counts().values):
    ax.text(.7,i,j,weight = "bold",fontsize=20)

plt.title("Count of target variable")
plt.show()
```





Point to infer from the graph

8% out of total client population have difficulties in repaying loans.

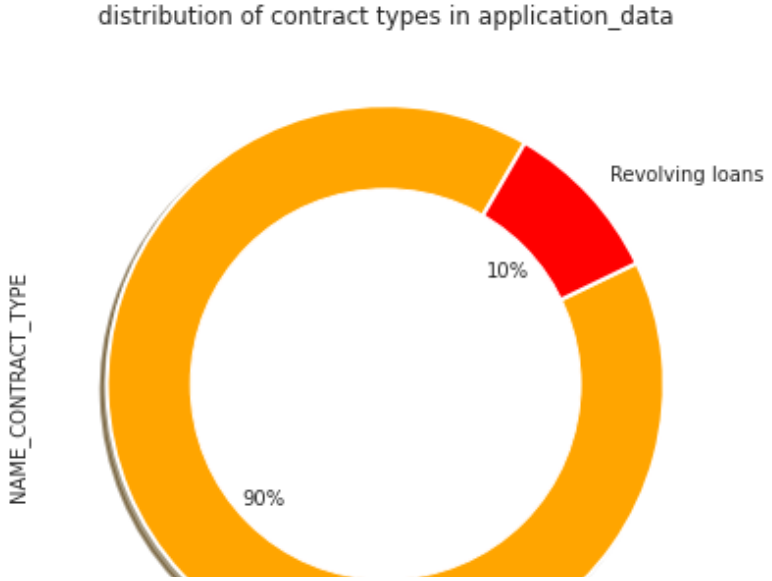
Concatenating application_data and previous_application

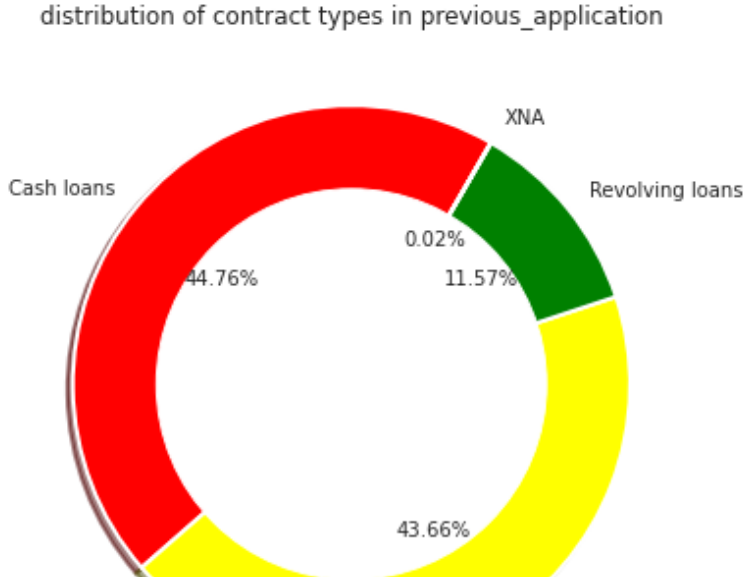
```
application_data_x = application_data[[x for x in application_data.columns if x not in ["TARGET"]]]
previous_application_x = previous_application[[x for x in previous_application.columns if x not in ["TARGET"]]]
application_data_x["type"] = "application_data"
previous_application_x["type"] = "previous_application"
data = pd.concat([application_data_x,previous_application_x],axis=0)
```

Distribution in Contract types in application_data

NAME_CONTRACT_TYPE : Identification if loan is cash , consumer or revolving

```
plt.figure(figsize=(14,7))
plt.subplot(121)
data[data["type"] == "application_data"]["NAME_CONTRACT_TYPE"].value_counts().plot.pie(autopct = "%1.0f%%",colors = ["orange","red"],sta
                                                                                       wedgeprops={"linewidth":2,"edgecolor":"white"},shadow =True)
circ = plt.Circle((0,0),.7,color="white")
plt.gca().add_artist(circ)
plt.title("distribution of contract types in application_data")

plt.subplot(122)
data[data["type"] == "previous_application"]["NAME_CONTRACT_TYPE"].value_counts().plot.pie(autopct = "%1.2f%%",colors = ["red","yellow",
                                                                                       wedgeprops={"linewidth":2,"edgecolor":"white"},shadow =True)
circ = plt.Circle((0,0),.7,color="white")
plt.gca().add_artist(circ)
plt.ylabel("")
plt.title("distribution of contract types in previous_application")
plt.show()

plt.show()
```



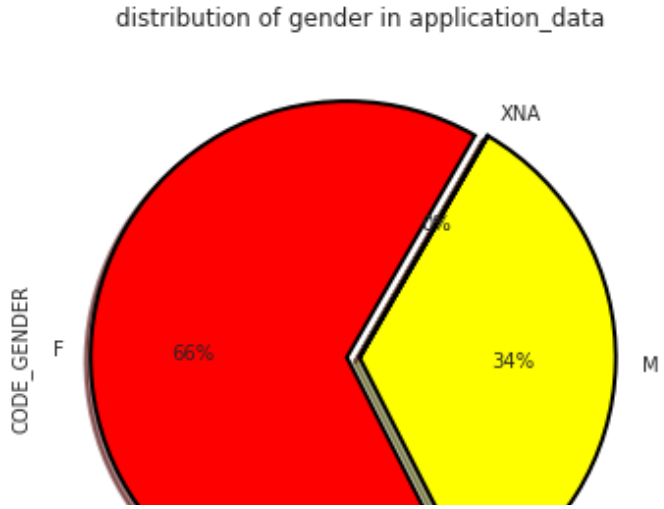


Point to infer from the graph

The percentage of revolving loans and cash loans are 10% & 90%.

Gender Distribution in application_data

```
fig = plt.figure(figsize=(13,6))
plt.subplot(121)
data[data["type"] == "application_data"]["CODE_GENDER"].value_counts().plot.pie(autopct = "%1.0f%%",colors = ["red","yellow"],startangle
                                                                                wedgeprops={"linewidth":2,"edgecolor":"k"},explode=[.05,0,0],sha
plt.title("distribution of gender in application_data")
plt.show()
```



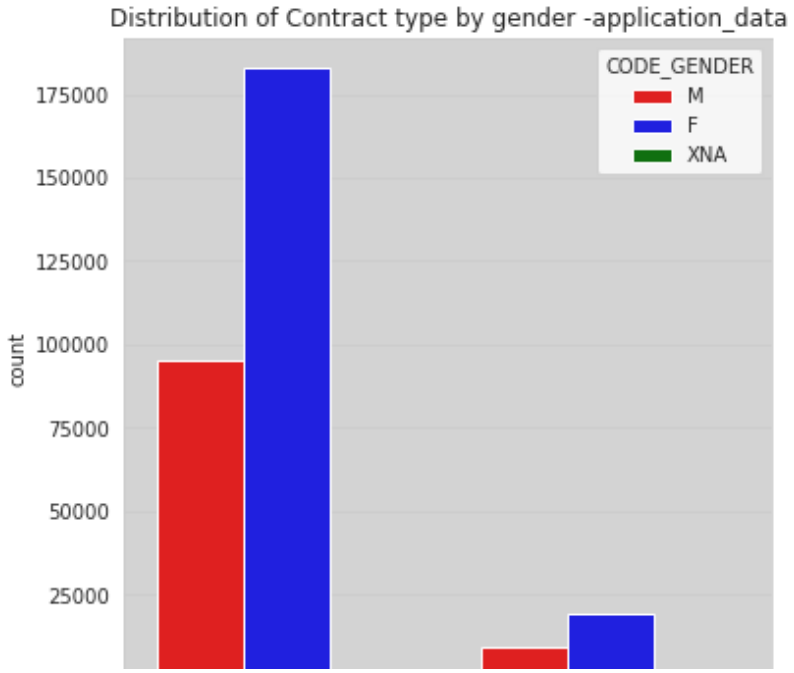
Point to infer from the graph

Female : 66%

Male : 34%

Distribution of Contract type by gender

```
fig  = plt.figure(figsize=(13,6))
plt.subplot(121)
ax = sns.countplot("NAME_CONTRACT_TYPE",hue="CODE_GENDER",data=data[data["type"] == "application_data"],palette=["r","b","g"])
ax.set_facecolor("lightgrey")
ax.set_title("Distribution of Contract type by gender -application_data")


plt.show()
```



Point to infer from the graph

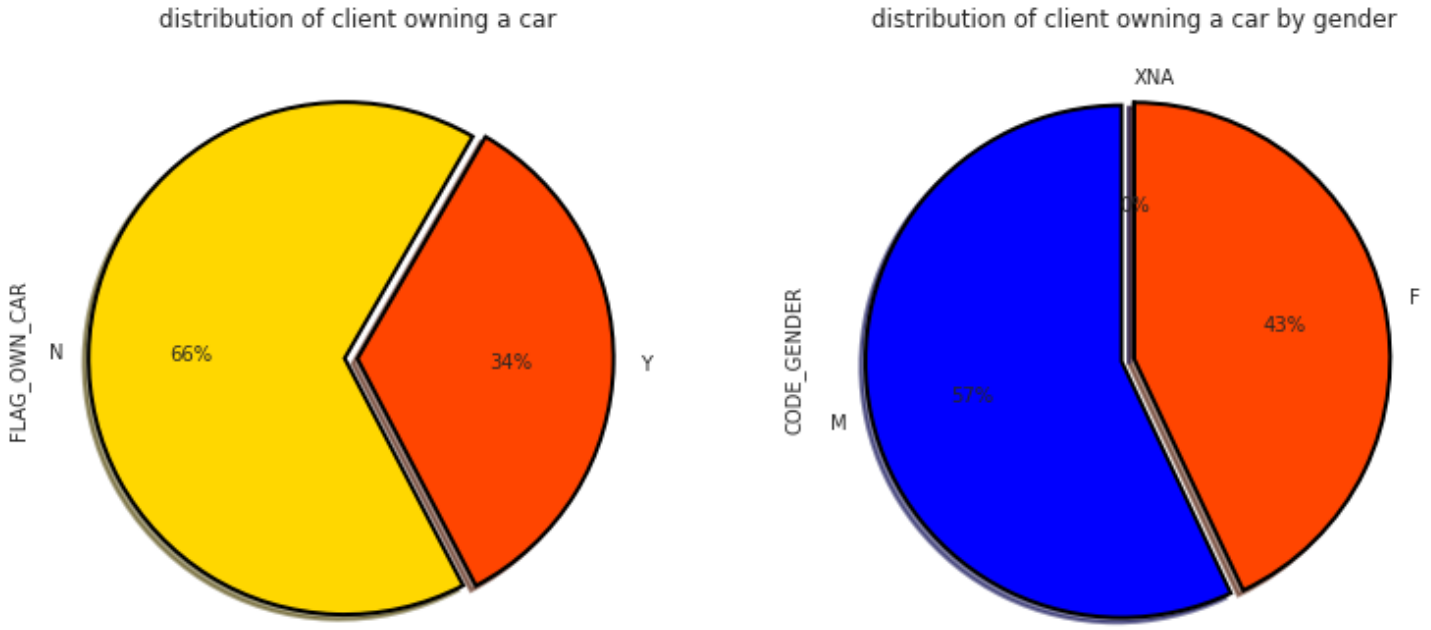
Cash loans is always prefered over Revolving loans by both genders

Distribution of client owning a car and by gender

FLAG_OWN_CAR Flag if the client owns a car .

```
fig = plt.figure(figsize=(13,6))

plt.subplot(121)
data["FLAG_OWN_CAR"].value_counts().plot.pie(autopct = "%1.0f%%",colors = ["gold","orangered"],startangle = 60,
                                                                    wedgeprops={"linewidth":2,"edgecolor":"k"},explode=[.05,0],shadc
plt.title("distribution of client owning a car")

plt.subplot(122)
data[data["FLAG_OWN_CAR"] == "Y"]["CODE_GENDER"].value_counts().plot.pie(autopct = "%1.0f%%",colors = ["b","orangered"],startangle = 90,
```

```
                                                                                wedgeprops={"linewidth":2,"edgecolor":"k"},explode=[.05,0,0],sha

plt.title("distribution of client owning a car by gender")

plt.show()
```



Point to infer from the graph

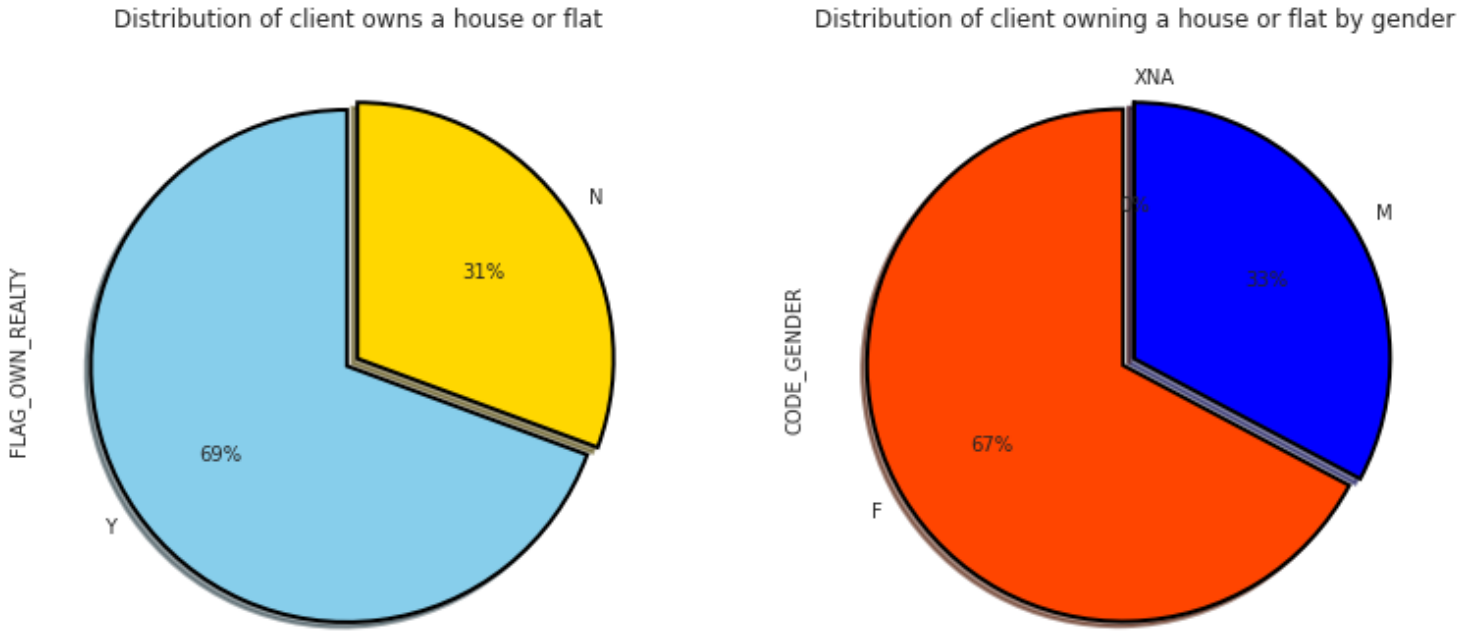
SUBPLOT 1 : Distribution of client owning a car. 34% of clients own a car .

SUBPLOT 2 : Distribution of client owning a car by gender. Out of total clients who own car 57% are male and 43% are female.

Distribution of client owning a house or flat and by gender

FLAG_OWN_REALTY - Flag if client owns a house or flat

```
plt.figure(figsize=(13,6))
plt.subplot(121)
data["FLAG_OWN_REALTY"].value_counts().plot.pie(autopct = "%1.0f%%",colors = ["skyblue","gold"],startangle = 90,
                                                wedgeprops={"linewidth":2,"edgecolor":"k"},explode=[0.05,0],shadow =True)
plt.title("Distribution of client owns a house or flat")

plt.subplot(122)
data[data["FLAG_OWN_REALTY"] == "Y"]["CODE_GENDER"].value_counts().plot.pie(autopct = "%1.0f%%",colors = ["orangered","b"],startangle =
                                                                            wedgeprops={"linewidth":2,"edgecolor":"k"},explode=[.05,0,0],sha
plt.title("Distribution of client owning a house or flat by gender")
plt.show()
```



Point to infer from the graph

SUBPLOT 1 : Distribution of client owning a house or flat . 69% of clients own a flat or house .

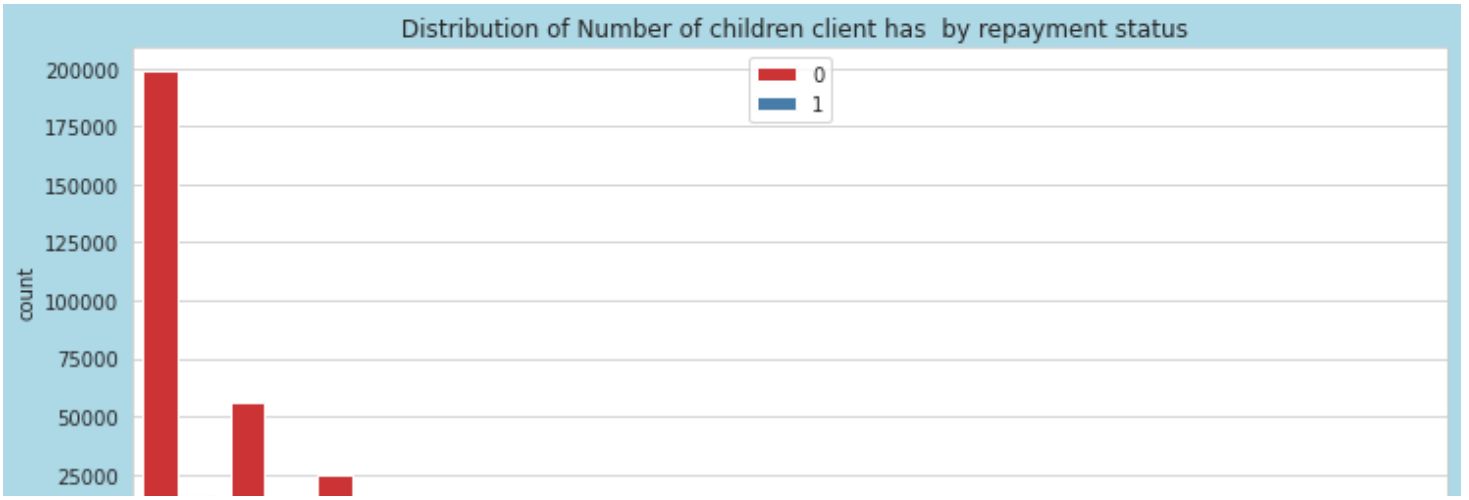
SUBPLOT 2 : Distribution of client owning a house or flat by gender . Out of total clients who own house 67% are female and 33% are male.

Distribution of Number of children and family members of client by repayment status.

CNT_CHILDREN - Number of children the client has.

CNT_FAM_MEMBERS - How many family members does client have.

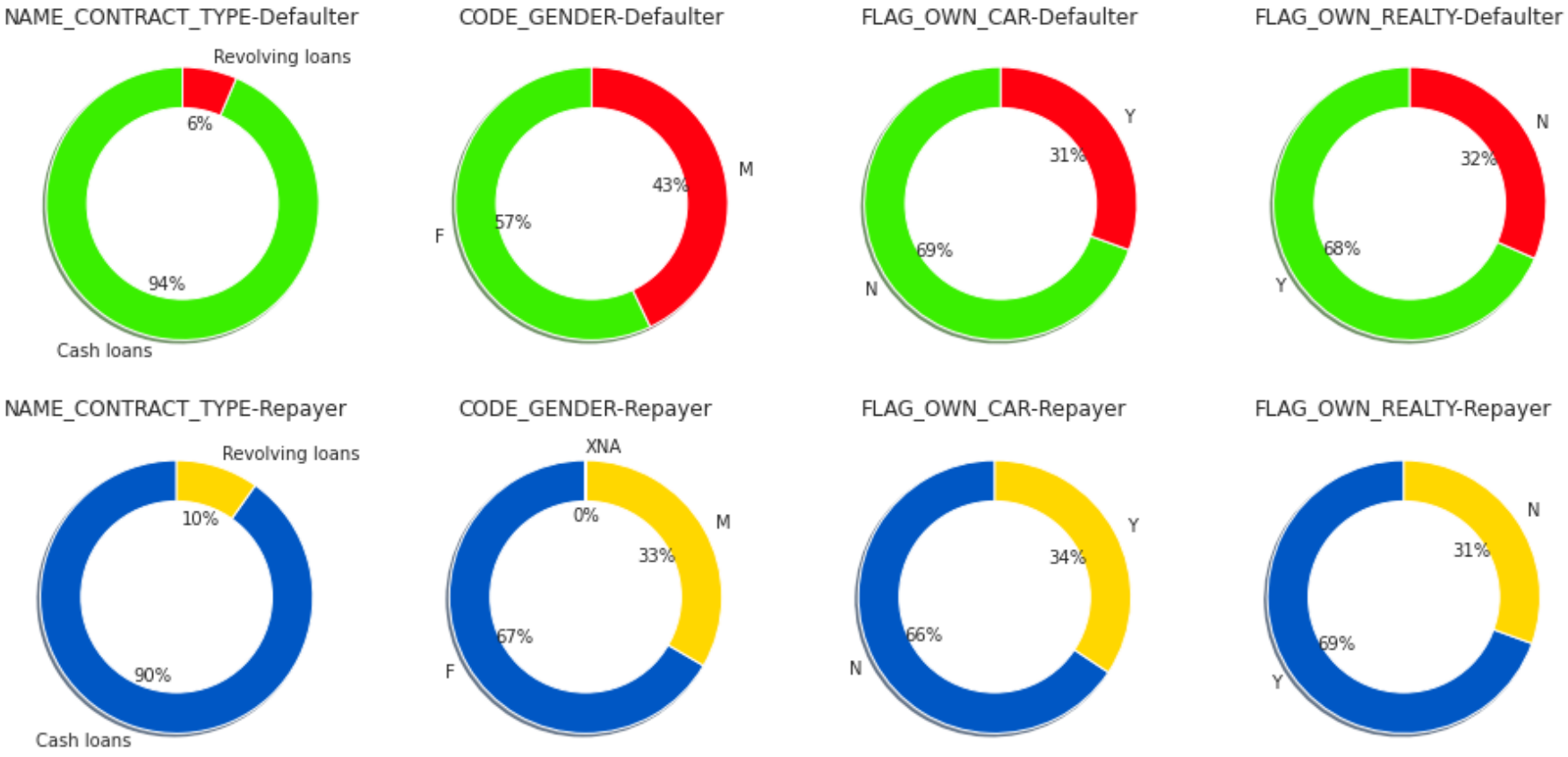
```
fig = plt.figure(figsize=(12,10))
plt.subplot(211)
sns.countplot(application_data["CNT_CHILDREN"],palette="Set1",hue=application_data["TARGET"])
plt.legend(loc="upper center")
plt.title(" Distribution of Number of children client has  by repayment status")
plt.subplot(212)
sns.countplot(application_data["CNT_FAM_MEMBERS"],palette="Set1",hue=application_data["TARGET"])
plt.legend(loc="upper center")
plt.title(" Distribution of Number of family members client has  by repayment status")
fig.set_facecolor("lightblue")
```



Distribution of contract type ,gender ,own car ,own house with respect to Repayment status(Target variable)

```
default = application_data[application_data["TARGET"]==1][[ 'NAME_CONTRACT_TYPE', 'CODE_GENDER','FLAG_OWN_CAR', 'FLAG_OWN_REALTY']]
non_default = application_data[application_data["TARGET"]==0][[ 'NAME_CONTRACT_TYPE', 'CODE_GENDER','FLAG_OWN_CAR', 'FLAG_OWN_REALTY']]

d_cols = ['NAME_CONTRACT_TYPE', 'CODE_GENDER','FLAG_OWN_CAR', 'FLAG_OWN_REALTY']
d_length = len(d_cols)

fig = plt.figure(figsize=(16,4))
for i,j in itertools.zip_longest(d_cols,range(d_length)):
    plt.subplot(1,4,j+1)
    default[i].value_counts().plot.pie(autopct = "%1.0f%%",colors = sns.color_palette("prism"),startangle = 90,
                                       wedgeprops={"linewidth":1,"edgecolor":"white"},shadow =True)
    circ = plt.Circle((0,0),.7,color="white")
    plt.gca().add_artist(circ)
    plt.ylabel("")
    plt.title(i+"-Defaulter")


fig = plt.figure(figsize=(16,4))
for i,j in itertools.zip_longest(d_cols,range(d_length)):
```

```
    plt.subplot(1,4,j+1)
    non_default[i].value_counts().plot.pie(autopct = "%1.0f%%",colors = sns.color_palette("prism",3),startangle = 90,
                                           wedgeprops={"linewidth":1,"edgecolor":"white"},shadow =True)
    circ = plt.Circle((0,0),.7,color="white")
    plt.gca().add_artist(circ)
    plt.ylabel("")
    plt.title(i+"-Repayer")
```



Point to infer from the graph

Percentage of males is 10% more in defaults than non defaulters.

Percentage of Cash Loans is 4% more in defaults than Revolving Loans.

Distribution of amount data

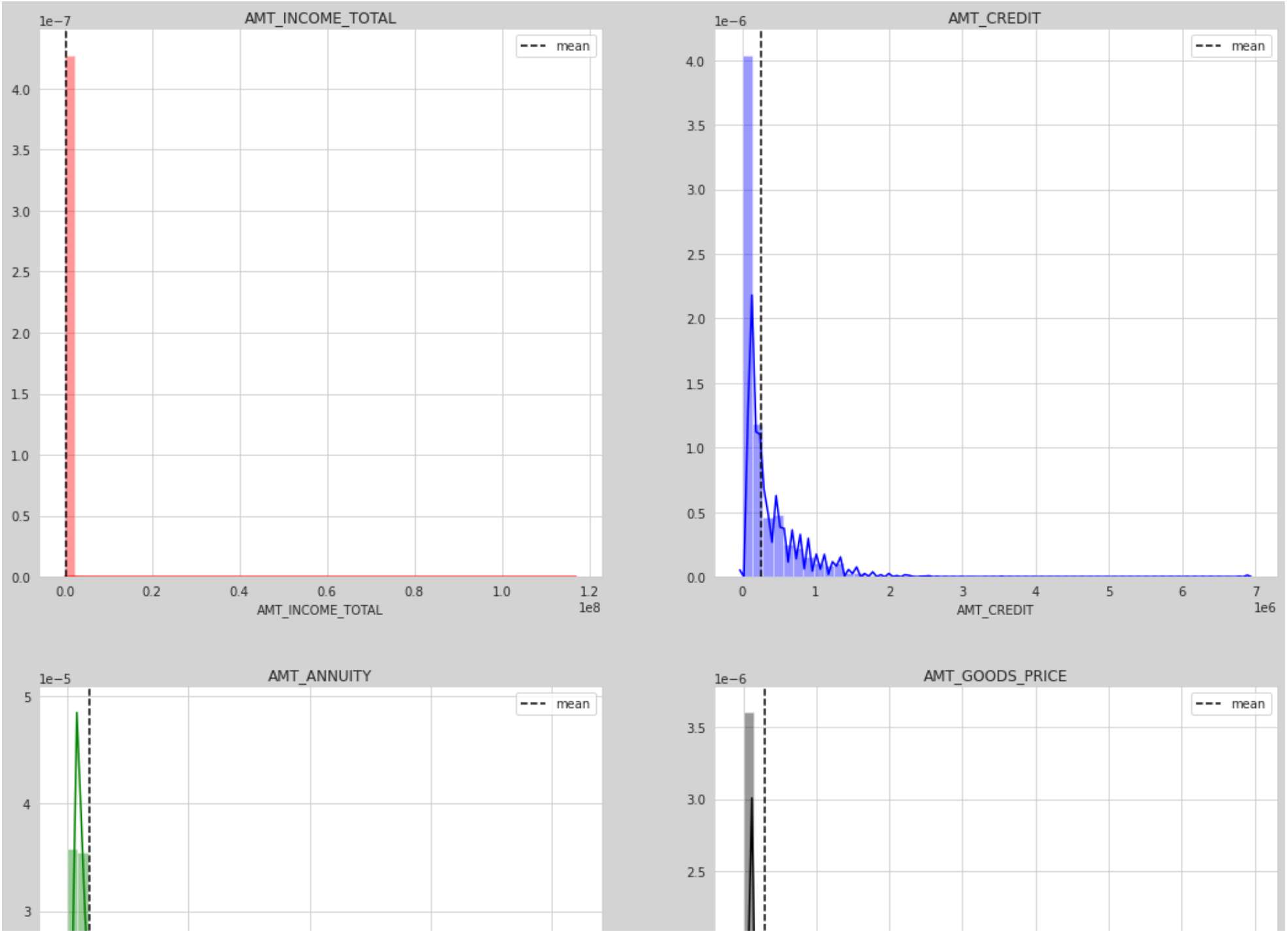
AMT_INCOME_TOTAL - Income of the client

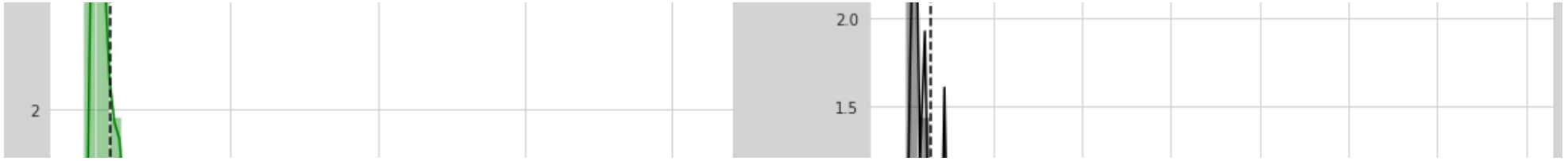
AMT_CREDIT - Credit amount of the loan

AMT_ANNUITY - Loan annuity

AMT_GOODS_PRICE - For consumer loans it is the price of the goods for which the loan is given

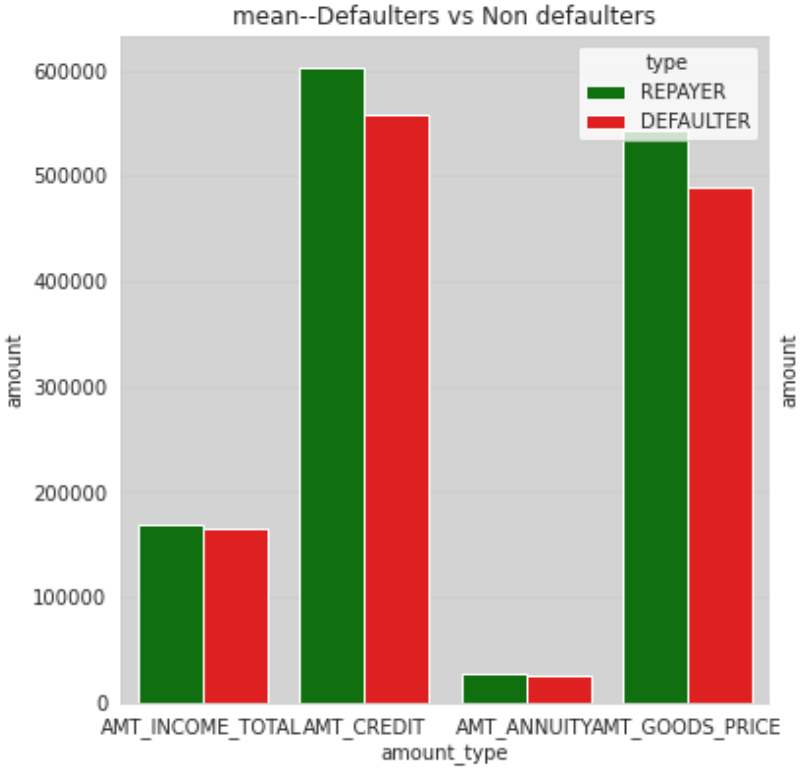
```
cols = [ 'AMT_INCOME_TOTAL', 'AMT_CREDIT','AMT_ANNUITY', 'AMT_GOODS_PRICE']
length = len(cols)
cs = ["r","b","g","k"]

ax = plt.figure(figsize=(18,18))
ax.set_facecolor("lightgrey")
for i,j,k in itertools.zip_longest(cols,range(length),cs):
    plt.subplot(2,2,j+1)
    sns.distplot(data[data[i].notnull()][i],color=k)
    plt.axvline(data[i].mean(),label = "mean",linestyle="dashed",color="k")
    plt.legend(loc="best")
    plt.title(i)
    plt.subplots_adjust(hspace = .2)
```

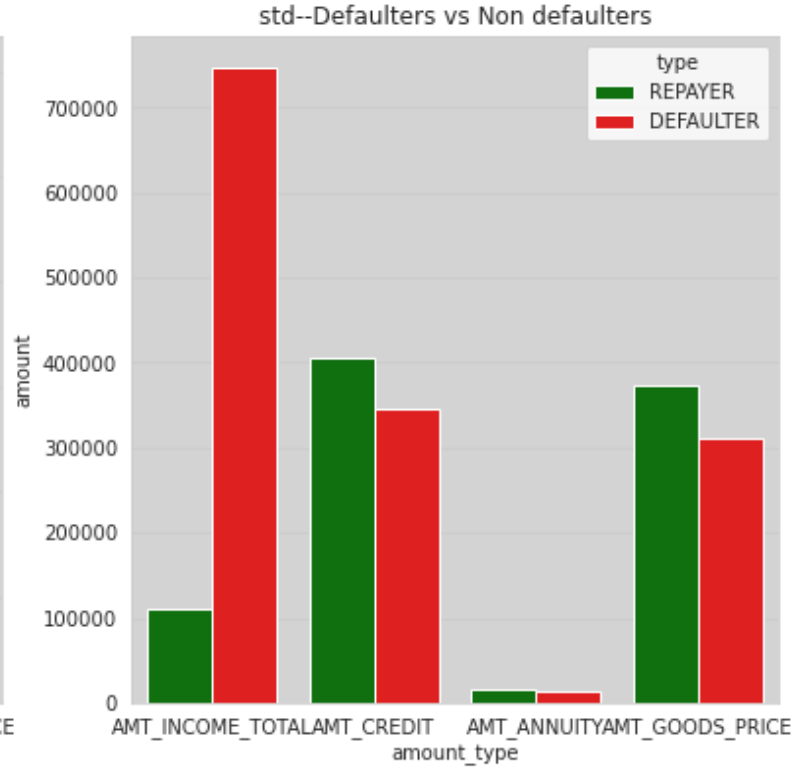

AMT_INCOME_TOTAL
1e−7
mean
4.0
3.5
3.0
2.5
2.0
1.5
1.0
0.5
0.0
0.0
0.2
0.4
0.6
0.8
1.0
1.2
1e8
AMT_INCOME_TOTAL
AMT_CREDIT
1e−6
mean
4.0
3.5
3.0
2.5
2.0
1.5
1.0
0.5
0.0
0
1
2
3
4
5
6
7
1e6
AMT_CREDIT
AMT_ANNUITY
1e−5
mean
5
4
3
AMT_GOODS_PRICE
1e−6
mean
3.5
3.0
2.5

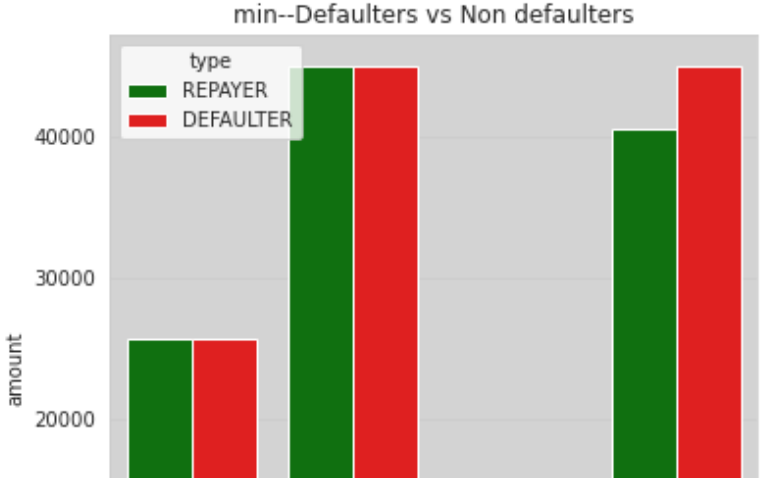


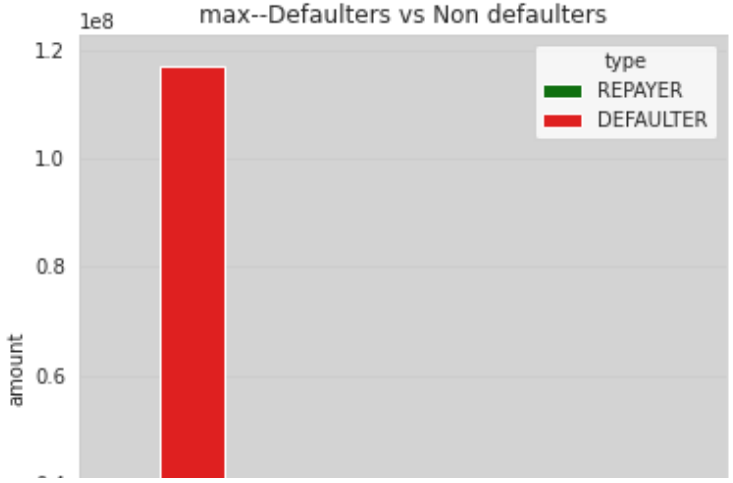
Comparing summary statistics between defaulters and non - defaulters for loan amounts.

```
df = application_data.groupby("TARGET")[cols].describe().transpose().reset_index()
df = df[df["level_1"].isin([ 'mean', 'std', 'min', 'max'])]
df_x = df[["level_0","level_1",0]]
df_y = df[["level_0","level_1",1]]
df_x = df_x.rename(columns={'level_0':"amount_type", 'level_1':"statistic", 0:"amount"})
df_x["type"] = "REPAYER"
df_y = df_y.rename(columns={'level_0':"amount_type", 'level_1':"statistic", 1:"amount"})
df_y["type"] = "DEFAULTER"
df_new = pd.concat([df_x,df_y],axis = 0)

stat = df_new["statistic"].unique().tolist()
length = len(stat)

plt.figure(figsize=(13,15))

for i,j in itertools.zip_longest(stat,range(length)):
    plt.subplot(2,2,j+1)
    fig = sns.barplot(df_new[df_new["statistic"] == i]["amount_type"],df_new[df_new["statistic"] == i]["amount"],
                hue=df_new[df_new["statistic"] == i]["type"],palette=["g","r"])
    plt.title(i + "--Defaulters vs Non defaulters")
    plt.subplots_adjust(hspace = .4)
    fig.set_facecolor("lightgrey")
```


mean--Defaulters vs Non defaulters
type
REPAYER
DEFAULTER
amount
600000
500000
400000
300000
200000
100000
0
AMT_INCOME_TOTAL
AMT_CREDIT
AMT_ANNUITY
AMT_GOODS_PRICE
amount_type


std--Defaulters vs Non defaulters
type
REPAYER
DEFAULTER
amount
700000
600000
500000
400000
300000
200000
100000
0
AMT_INCOME_TOTAL
AMT_CREDIT
AMT_ANNUITY
AMT_GOODS_PRICE
amount_type


min--Defaulters vs Non defaulters
type
REPAYER
DEFAULTER
amount
40000
30000
20000


max--Defaulters vs Non defaulters
1e8
type
REPAYER
DEFAULTER
amount
1.2
1.0
0.8
0.6

Point to infer from the graph

Income of client -

1 . Average income of clients who default and who do not are almost same.

2 . Standard deviation in income of client who default is very high compared to who do not default.

3 . Clients who default also has maximum income earnings

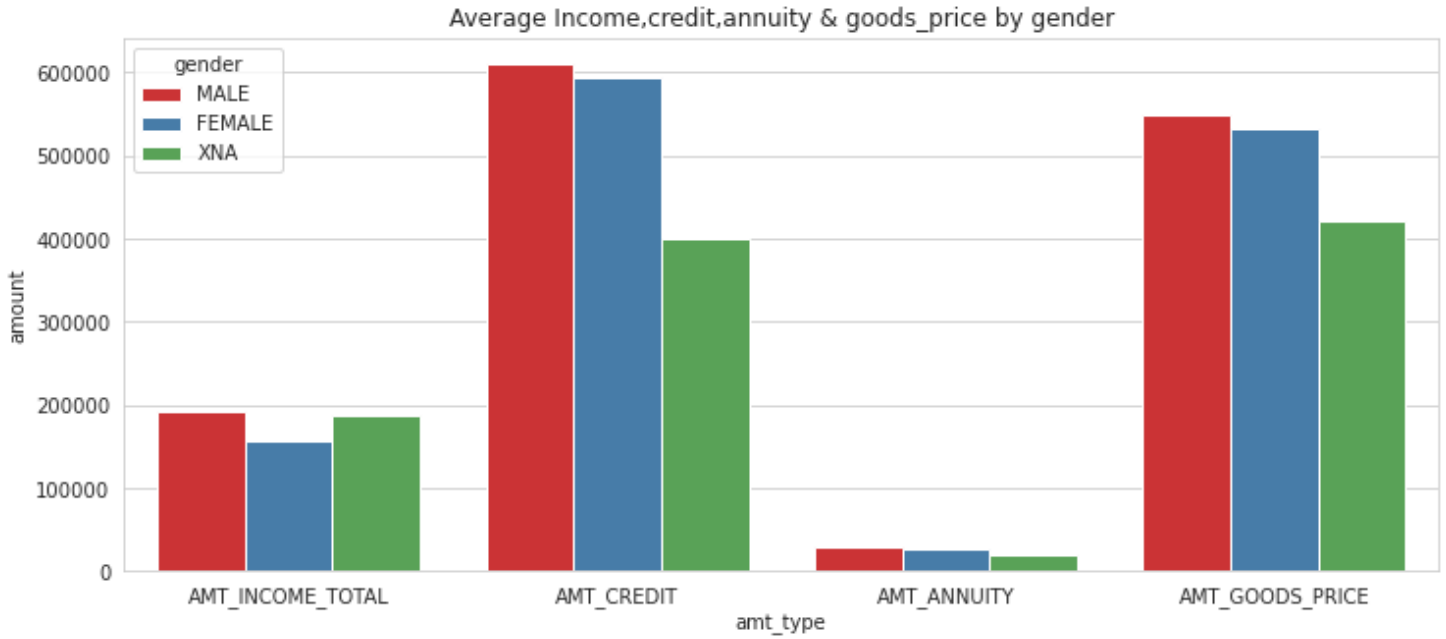
Credit amount of the loan ,Loan annuity,Amount goods price -

1 . Statistics between credit amounts,Loan annuity and Amount goods price given to cilents who default and who dont are almost similar.

Average Income,credit,annuity & goods_price by gender

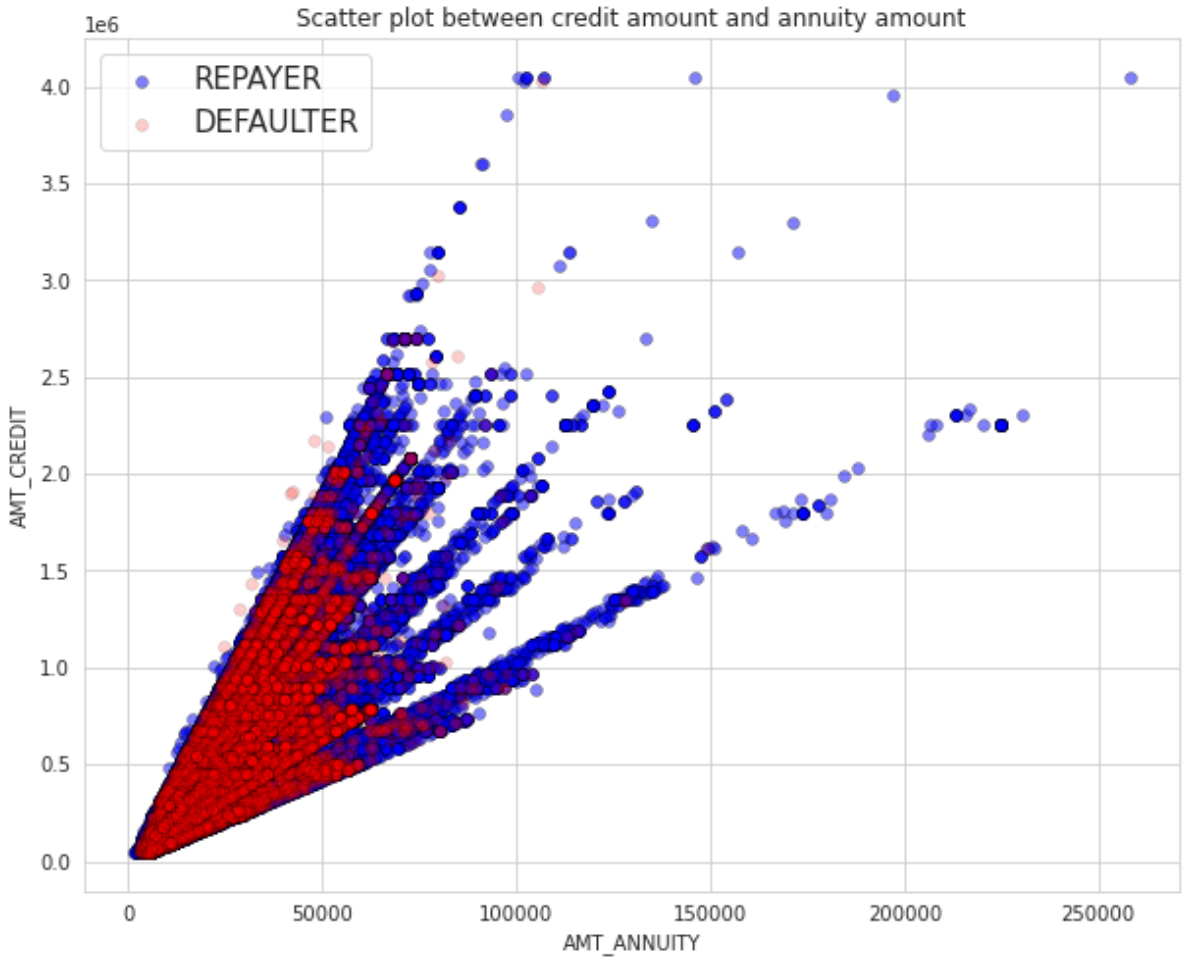
```
cols = [ 'AMT_INCOME_TOTAL', 'AMT_CREDIT','AMT_ANNUITY', 'AMT_GOODS_PRICE']

df1 = data.groupby("CODE_GENDER")[cols].mean().transpose().reset_index()

df_f   = df1[["index","F"]]
df_f   = df_f.rename(columns={'index':"amt_type", 'F':"amount"})
df_f["gender"] = "FEMALE"
df_m   = df1[["index","M"]]
df_m   = df_m.rename(columns={'index':"amt_type", 'M':"amount"})
df_m["gender"] = "MALE"
df_xna = df1[["index","XNA"]]
df_xna = df_xna.rename(columns={'index':"amt_type", 'XNA':"amount"})
df_xna["gender"] = "XNA"

df_gen = pd.concat([df_m,df_f,df_xna],axis=0)

plt.figure(figsize=(12,5))
```

```
ax = sns.barplot("amt_type","amount",data=df_gen,hue="gender",palette="Set1")
plt.title("Average Income,credit,annuity & goods_price by gender")
plt.show()
```



Scatter plot between credit amount and annuity amount

```
fig = plt.figure(figsize=(10,8))
plt.scatter(application_data[application_data["TARGET"]==0]['AMT_ANNUITY'],application_data[application_data["TARGET"]==0]['AMT_CREDIT']
            color="b",alpha=.5,label="REPAYER",linewidth=.5,edgecolor="k")
plt.scatter(application_data[application_data["TARGET"]==1]['AMT_ANNUITY'],application_data[application_data["TARGET"]==1]['AMT_CREDIT']
            color="r",alpha=.2,label="DEFAULTER",linewidth=.5,edgecolor="k")
plt.legend(loc="best",prop={"size":15})
plt.xlabel("AMT_ANNUITY")
plt.ylabel("AMT_CREDIT")
plt.title("Scatter plot between credit amount and annuity amount")
plt.show()
```



Pair Plot between amount variables

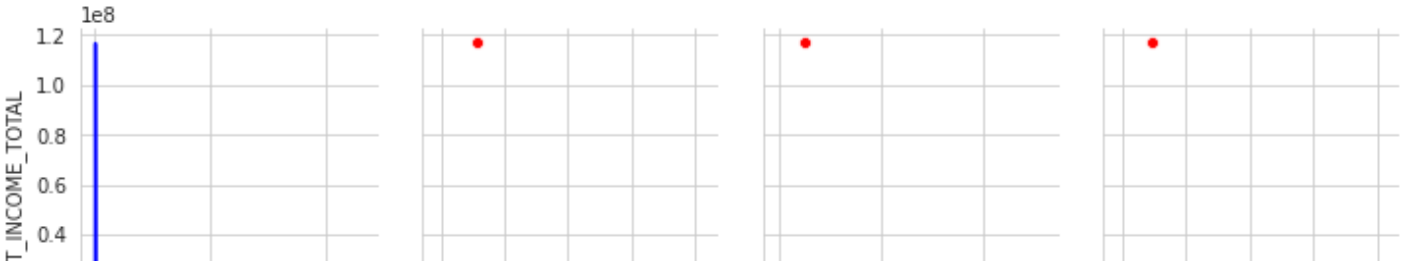
AMT_INCOME_TOTAL - Income of the client

AMT_CREDIT - Credit amount of the loan

AMT_ANNUITY - Loan annuity

AMT_GOODS_PRICE - For consumer loans it is the price of the goods for which the loan is given

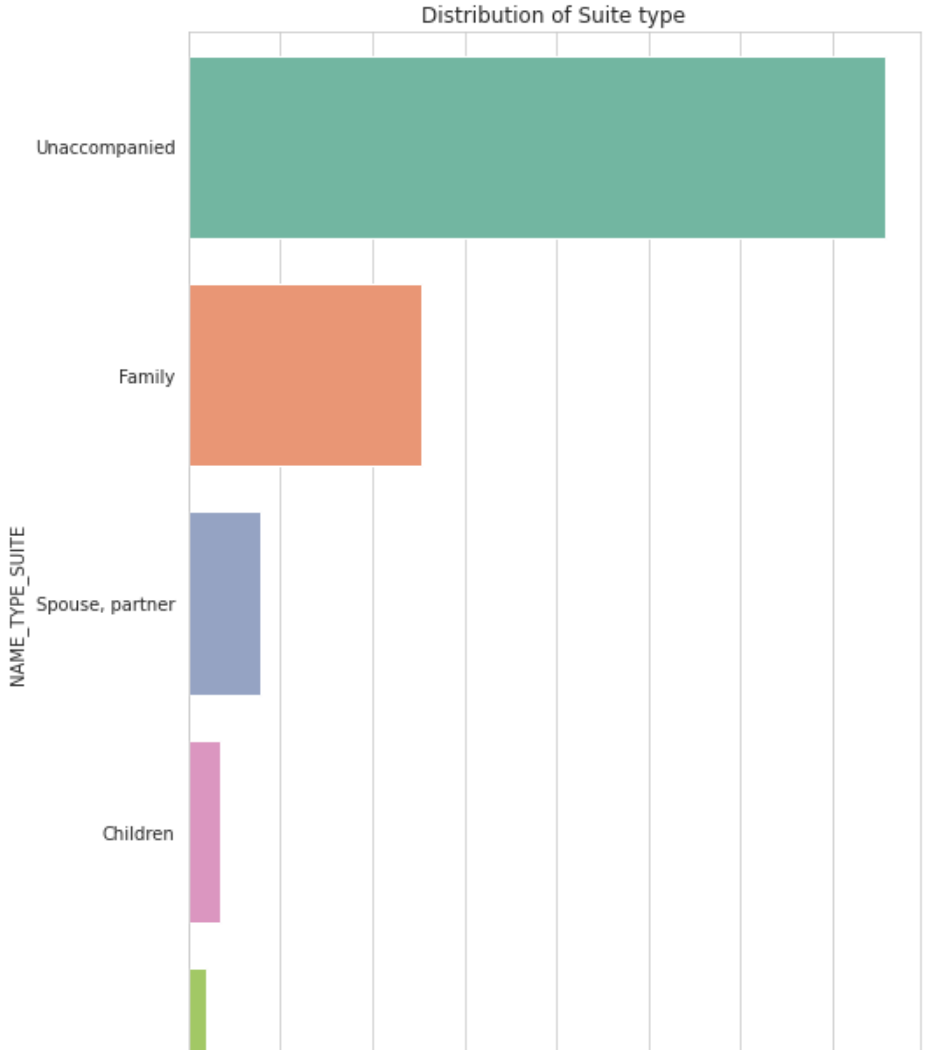
```
amt = application_data[[ 'AMT_INCOME_TOTAL','AMT_CREDIT',
                         'AMT_ANNUITY', 'AMT_GOODS_PRICE',"TARGET"]]
amt = amt[(amt["AMT_GOODS_PRICE"].notnull()) & (amt["AMT_ANNUITY"].notnull())]
sns.pairplot(amt,hue="TARGET",palette=["b","r"])
plt.show()
```

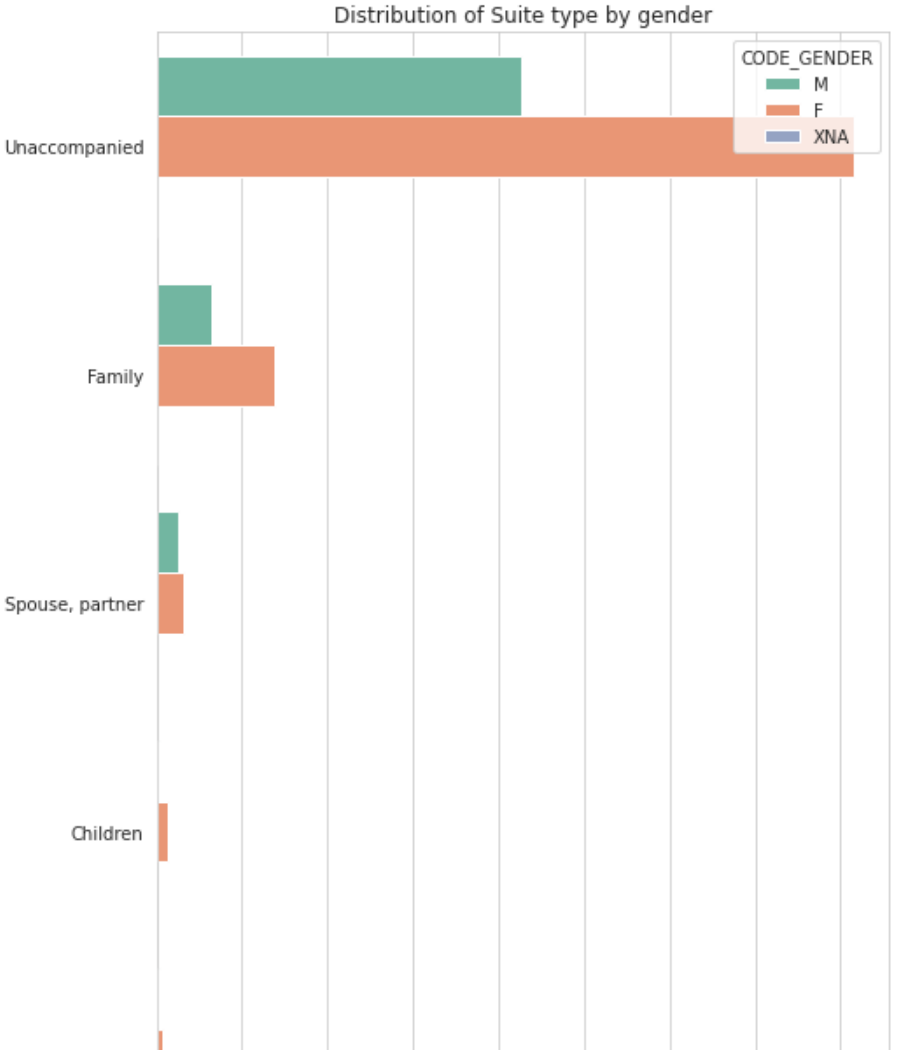


Distribution of Suite type

NAME_TYPE_SUITE - Who was accompanying client when he was applying for the loan.

```
plt.figure(figsize=(18,12))
plt.subplot(121)
sns.countplot(y=data["NAME_TYPE_SUITE"],
              palette="Set2",
              order=data["NAME_TYPE_SUITE"].value_counts().index[:5])
plt.title("Distribution of Suite type")

plt.subplot(122)
sns.countplot(y=data["NAME_TYPE_SUITE"],
              hue=data["CODE_GENDER"],palette="Set2",
              order=data["NAME_TYPE_SUITE"].value_counts().index[:5])
plt.ylabel("")
plt.title("Distribution of Suite type by gender")
plt.subplots_adjust(wspace = .4)
```



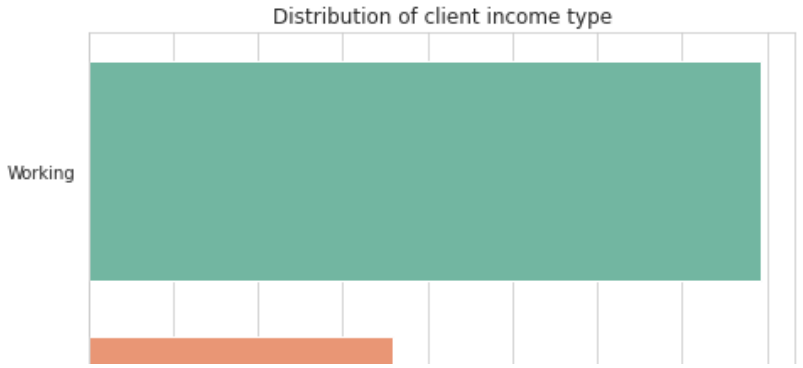


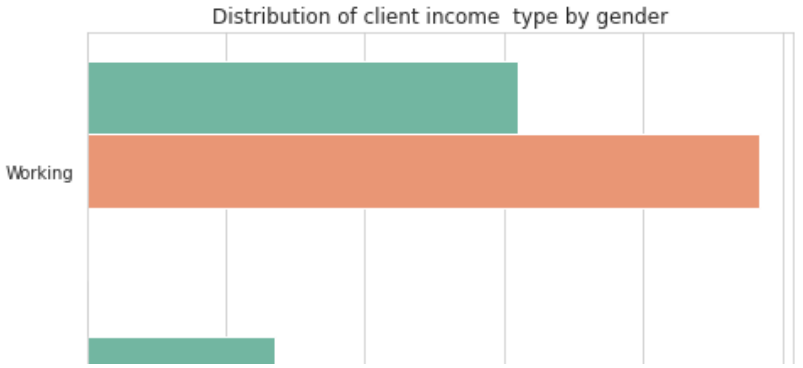
Distribution of client income type

NAME_INCOME_TYPE Clients income type (businessman, working, maternity leave,...)

```
plt.figure(figsize=(18,12))
plt.subplot(121)
sns.countplot(y=data["NAME_INCOME_TYPE"],
              palette="Set2",
```

```
              order=data["NAME_INCOME_TYPE"].value_counts().index[:4])
plt.title("Distribution of client income type")

plt.subplot(122)
sns.countplot(y=data["NAME_INCOME_TYPE"],
              hue=data["CODE_GENDER"],
              palette="Set2",
              order=data["NAME_INCOME_TYPE"].value_counts().index[:4])
plt.ylabel("")
plt.title("Distribution of client income  type by gender")
plt.subplots_adjust(wspace = .4)
```

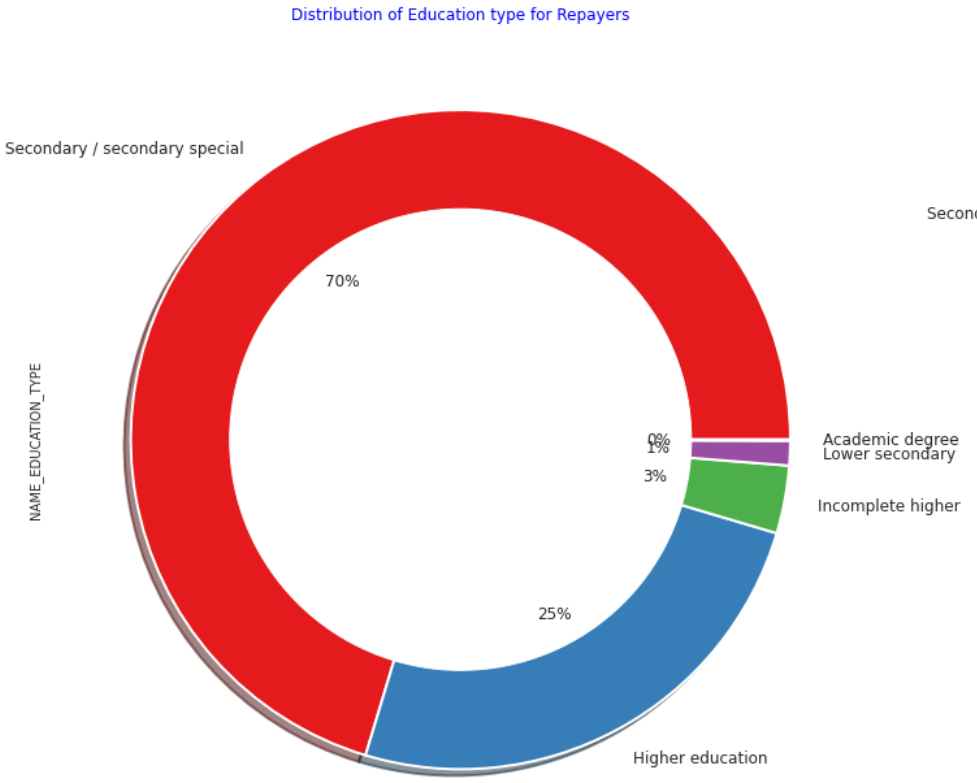


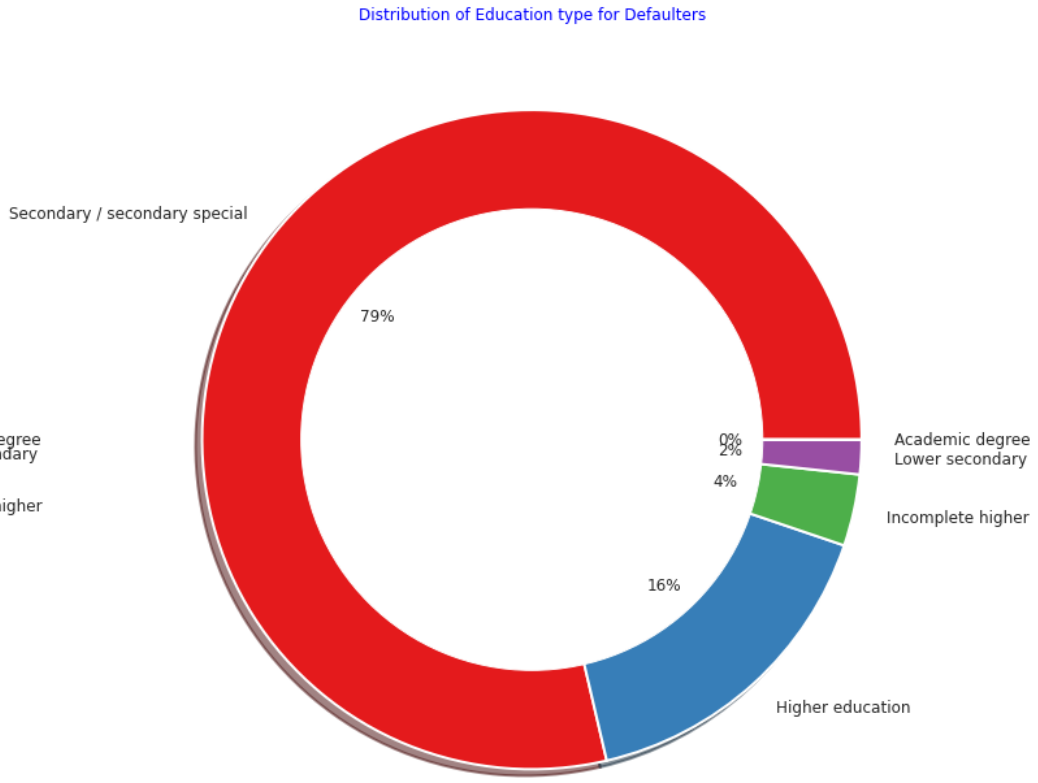


Distribution of Education type by loan repayment status

NAME_EDUCATION_TYPE Level of highest education the client achieved..


```
plt.figure(figsize=(25,25))
plt.subplot(121)
application_data[application_data["TARGET"]==0]["NAME_EDUCATION_TYPE"].value_counts().plot.pie(fontsize=12,autopct = "%1.0f%%",
                                                                                               colors = sns.color_palette("Set1"),
                                                                                               wedgeprops={"linewidth":2,"edgecolor":"white"},shadow =True)
circ = plt.Circle((0,0),.7,color="white")
plt.gca().add_artist(circ)
plt.title("Distribution of Education type for Repayers",color="b")

plt.subplot(122)
application_data[application_data["TARGET"]==1]["NAME_EDUCATION_TYPE"].value_counts().plot.pie(fontsize=12,autopct = "%1.0f%%",
                                                                                               colors = sns.color_palette("Set1"),
                                                                                               wedgeprops={"linewidth":2,"edgecolor":"white"},shadow =True)
circ = plt.Circle((0,0),.7,color="white")
plt.gca().add_artist(circ)
plt.title("Distribution of Education type for Defaulters",color="b")
plt.ylabel("")
plt.show()
```



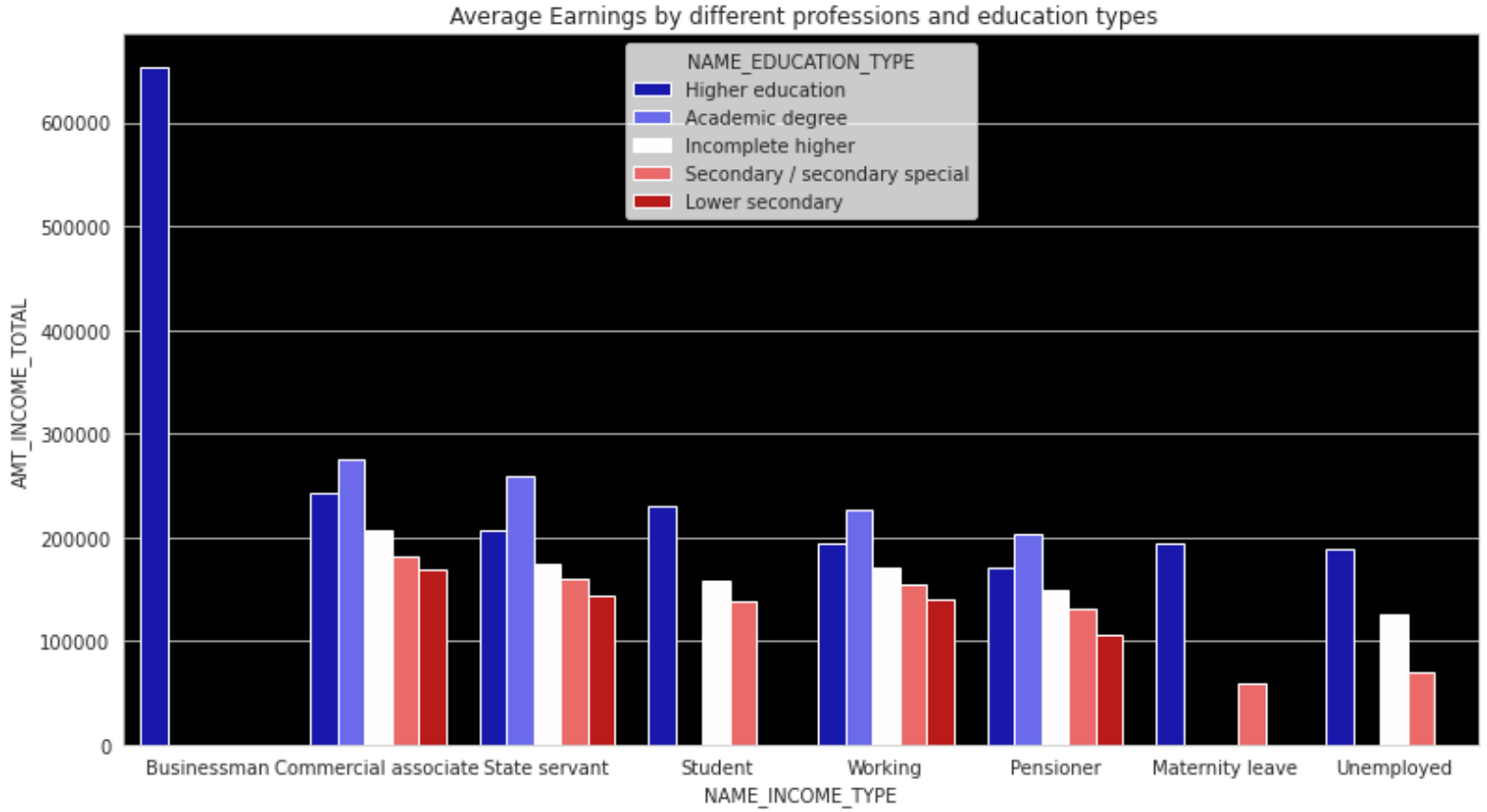


Point to infer from the graph

Clients who default have proportionally 9% less higher education compared to clients who do not default.

Average Earnings by different professions and education types

```
edu = data.groupby(['NAME_EDUCATION_TYPE','NAME_INCOME_TYPE'])['AMT_INCOME_TOTAL'].mean().reset_index().sort_values(by='AMT_INCOME_TOTAL
fig = plt.figure(figsize=(13,7))
ax = sns.barplot('NAME_INCOME_TYPE','AMT_INCOME_TOTAL',data=edu,hue='NAME_EDUCATION_TYPE',palette="seismic")
ax.set_facecolor("k")
plt.title(" Average Earnings by different professions and education types")
plt.show()
```

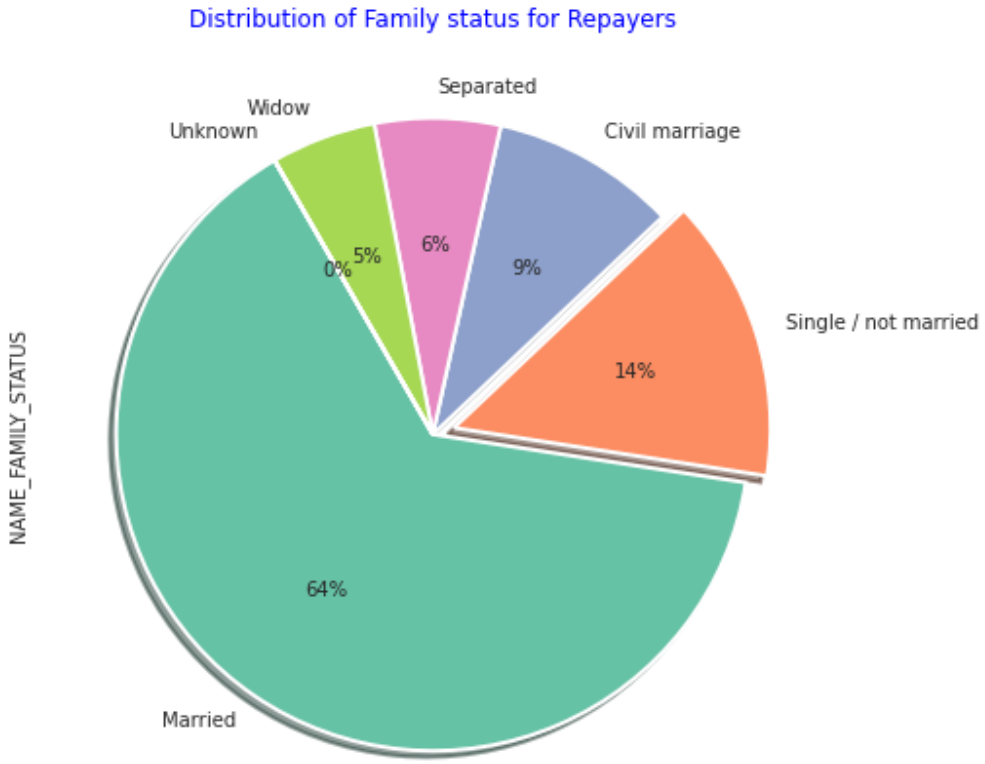


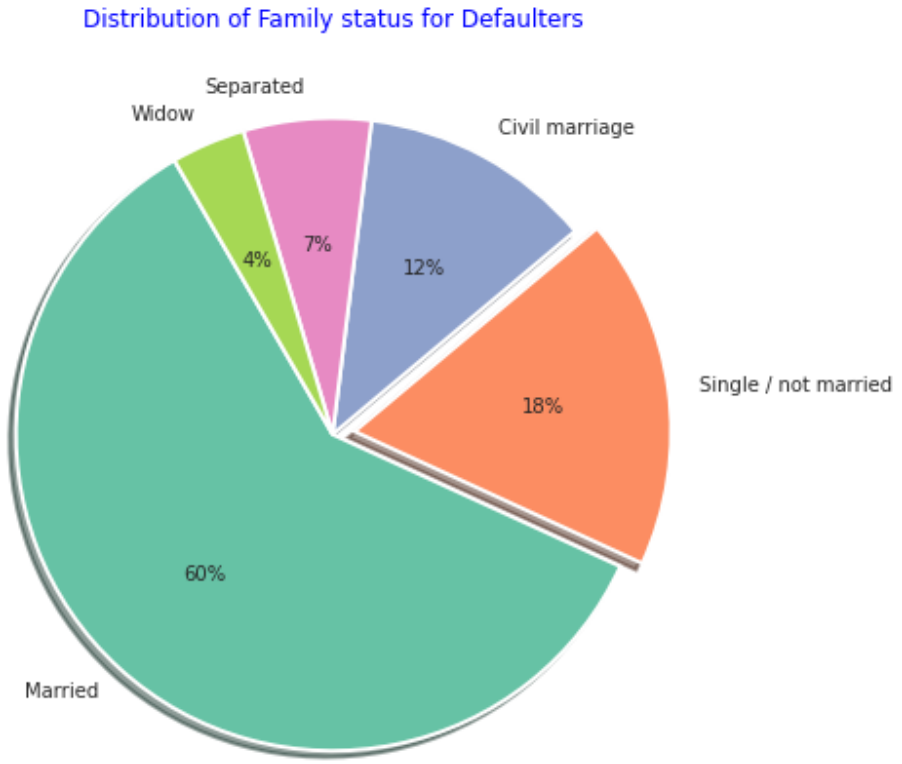
Distribution of Education type by loan repayment status

NAME_FAMILY_STATUS - Family status of the client

```
plt.figure(figsize=(16,8))
plt.subplot(121)
application_data[application_data["TARGET"]==0]["NAME_FAMILY_STATUS"].value_counts().plot.pie(autopct = "%1.0f%%",
                                                                                              startangle=120,colors = sns.color_palette("Set2",7),
                                                                                    wedgeprops={"linewidth":2,"edgecolor":"white"},shadow =True,explode=[0,.07,0,0,0,0])

plt.title("Distribution of Family status for Repayers",color="b")
```

```
plt.subplot(122)
application_data[application_data["TARGET"]==1]["NAME_FAMILY_STATUS"].value_counts().plot.pie(autopct = "%1.0f%%",
                                                    startangle=120,colors = sns.color_palette("Set2",7),
                                                   wedgeprops={"linewidth":2,"edgecolor":"white"},shadow =True,explode=[0,.07,0,0,0])


plt.title("Distribution of Family status for Defaulters",color="b")
plt.ylabel("")
plt.show()
```





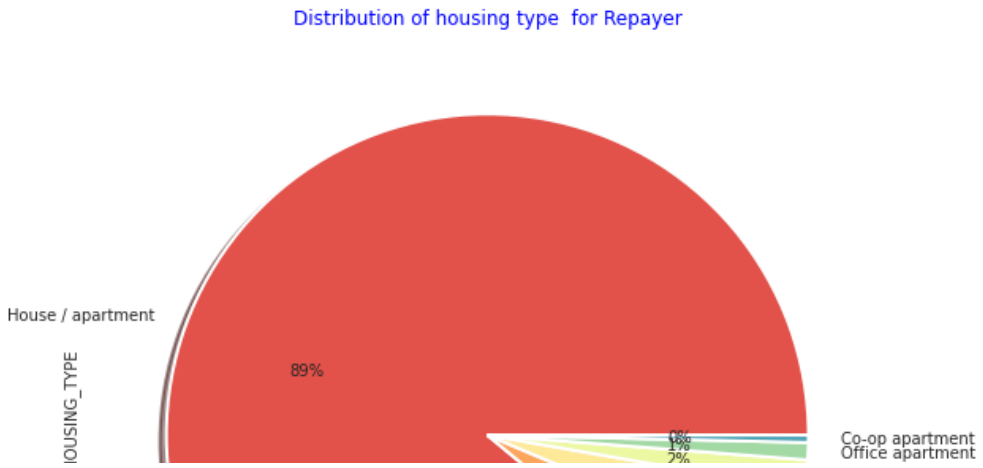
Point to infer from the graph

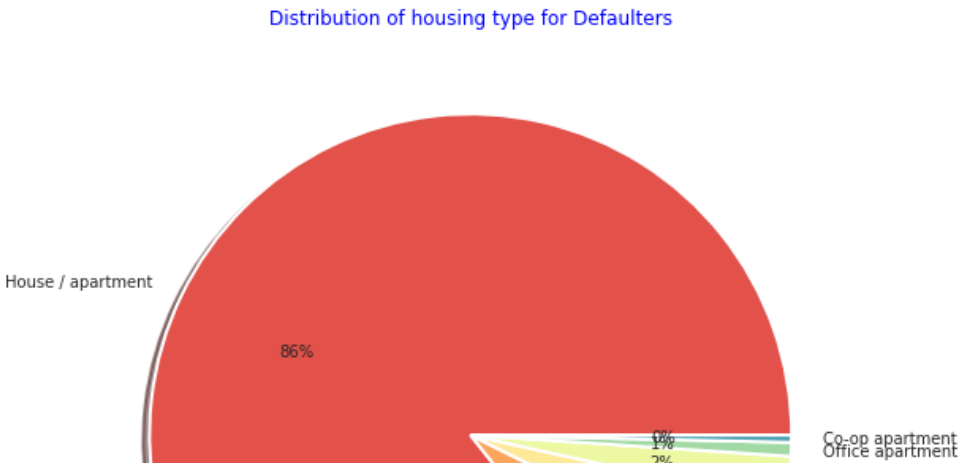
Percentage of single people are more in defaulters than non defaulters.

Distribution of Housing type by loan repayment status

NAME_HOUSING_TYPE - What is the housing situation of the client (renting, living with parents, ...)

```
plt.figure(figsize=(20,20))
plt.subplot(121)
application_data[application_data["TARGET"]==0]["NAME_HOUSING_TYPE"].value_counts().plot.pie(autopct = "%1.0f%%",fontsize=10,
                                                                colors = sns.color_palette("Spectral"),
                                                wedgeprops={"linewidth":2,"edgecolor":"white"},shadow =True)

plt.title("Distribution of housing type  for Repayer",color="b")

plt.subplot(122)
application_data[application_data["TARGET"]==1]["NAME_HOUSING_TYPE"].value_counts().plot.pie(autopct = "%1.0f%%",fontsize=10,
                                                      colors = sns.color_palette("Spectral"),
                                                wedgeprops={"linewidth":2,"edgecolor":"white"},shadow =True)


plt.title("Distribution of housing type for Defaulters",color="b")
plt.ylabel("")
plt.show()
```

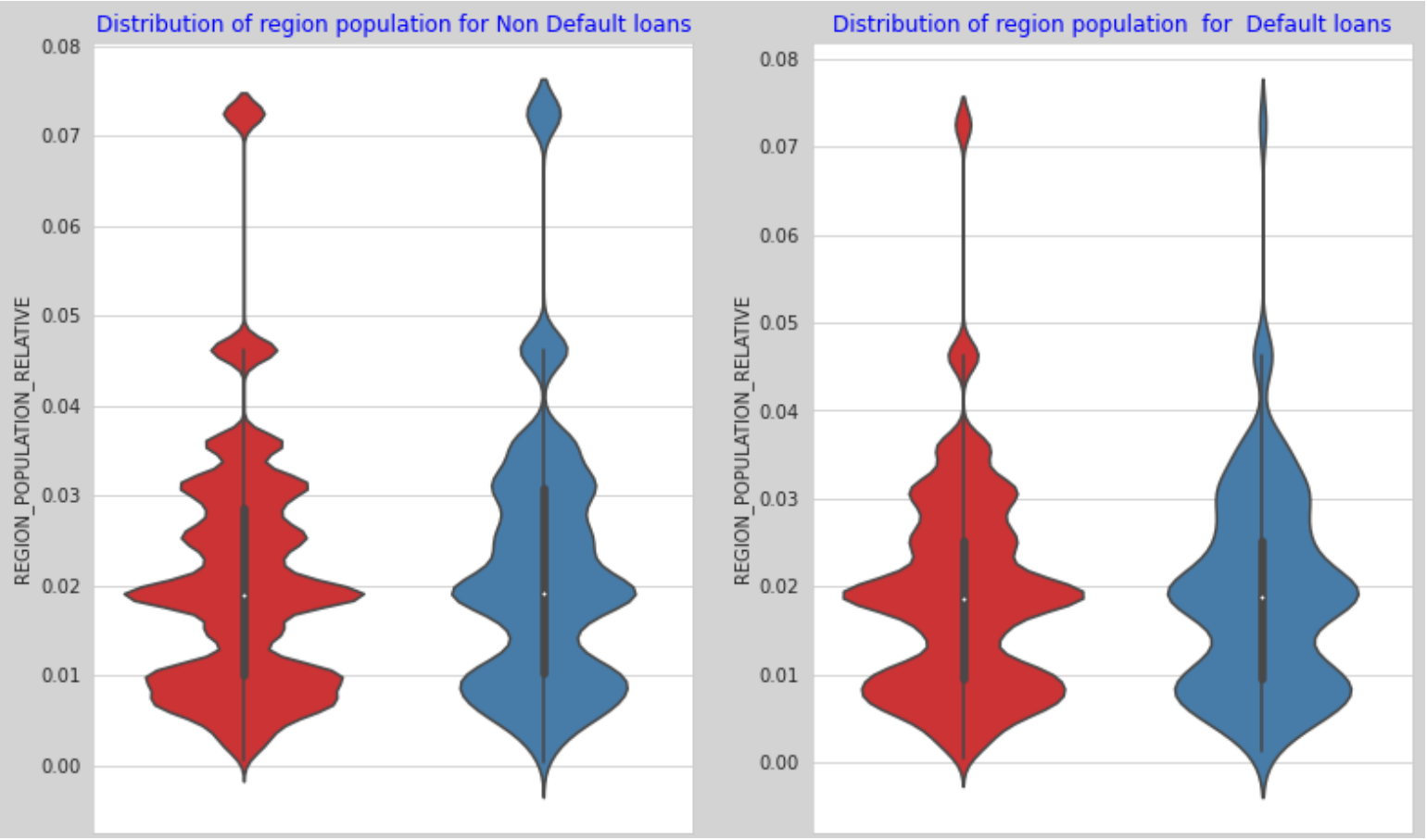




Distribution normalized population of region where client lives by loan repayment status

REGION_POPULATION_RELATIVE - Normalized population of region where client lives (higher number means the client lives in more populated region).

```
fig = plt.figure(figsize=(13,8))

plt.subplot(121)
sns.violinplot(y=application_data[application_data["TARGET"]==0]["REGION_POPULATION_RELATIVE"]
               ,x=application_data[application_data["TARGET"]==0]["NAME_CONTRACT_TYPE"],
               palette="Set1")
plt.title("Distribution of region population for Non Default loans",color="b")
plt.subplot(122)
sns.violinplot(y = application_data[application_data["TARGET"]==1]["REGION_POPULATION_RELATIVE"]
               ,x=application_data[application_data["TARGET"]==1]["NAME_CONTRACT_TYPE"]
               ,palette="Set1")
plt.title("Distribution of region population  for  Default loans",color="b")

plt.subplots_adjust(wspace = .2)
fig.set_facecolor("lightgrey")
```



Point to infer from the graph

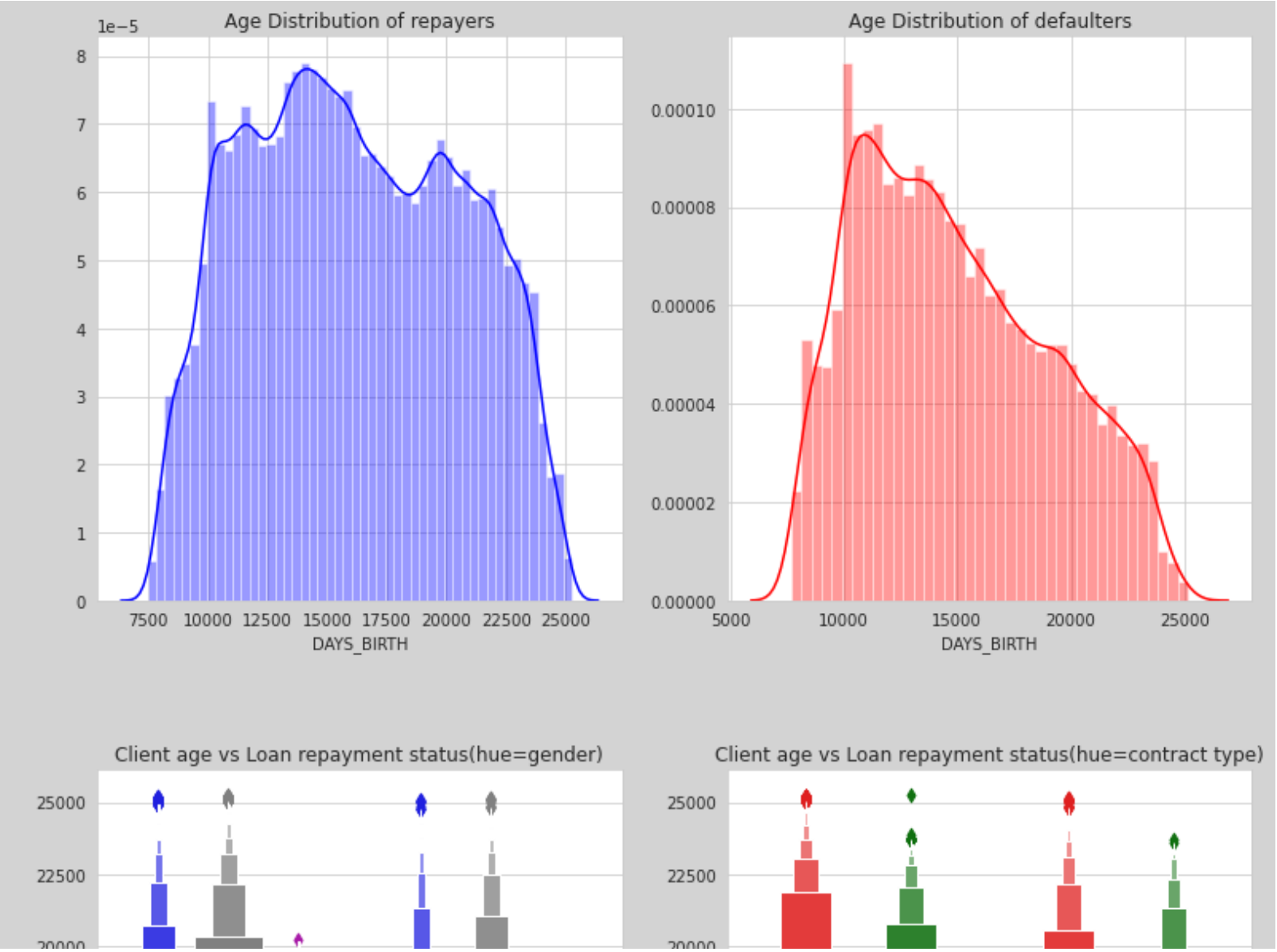
In High population density regions people are less likely to default on loans.

Client's age

DAYS_BIRTH - Client's age in days at the time of application.

```
fig = plt.figure(figsize=(13,15))

plt.subplot(221)
```

```
sns.distplot(application_data[application_data["TARGET"]==0]["DAYS_BIRTH"],color="b")
plt.title("Age Distribution of repayers")

plt.subplot(222)
sns.distplot(application_data[application_data["TARGET"]==1]["DAYS_BIRTH"],color="r")
plt.title("Age Distribution of defaulters")

plt.subplot(223)
sns.lvplot(application_data["TARGET"],application_data["DAYS_BIRTH"],hue=application_data["CODE_GENDER"],palette=["b","grey","m"])
plt.axhline(application_data["DAYS_BIRTH"].mean(),linestyle="dashed",color="k",label ="average age of client")
plt.legend(loc="lower right")
plt.title("Client age vs Loan repayment status(hue=gender)")

plt.subplot(224)
sns.lvplot(application_data["TARGET"],application_data["DAYS_BIRTH"],hue=application_data["NAME_CONTRACT_TYPE"],palette=["r","g"])
plt.axhline(application_data["DAYS_BIRTH"].mean(),linestyle="dashed",color="k",label ="average age of client")
plt.legend(loc="lower right")
plt.title("Client age vs Loan repayment status(hue=contract type)")

plt.subplots_adjust(wspace = .2,hspace = .3)

fig.set_facecolor("lightgrey")
```



Point to infer from the graph

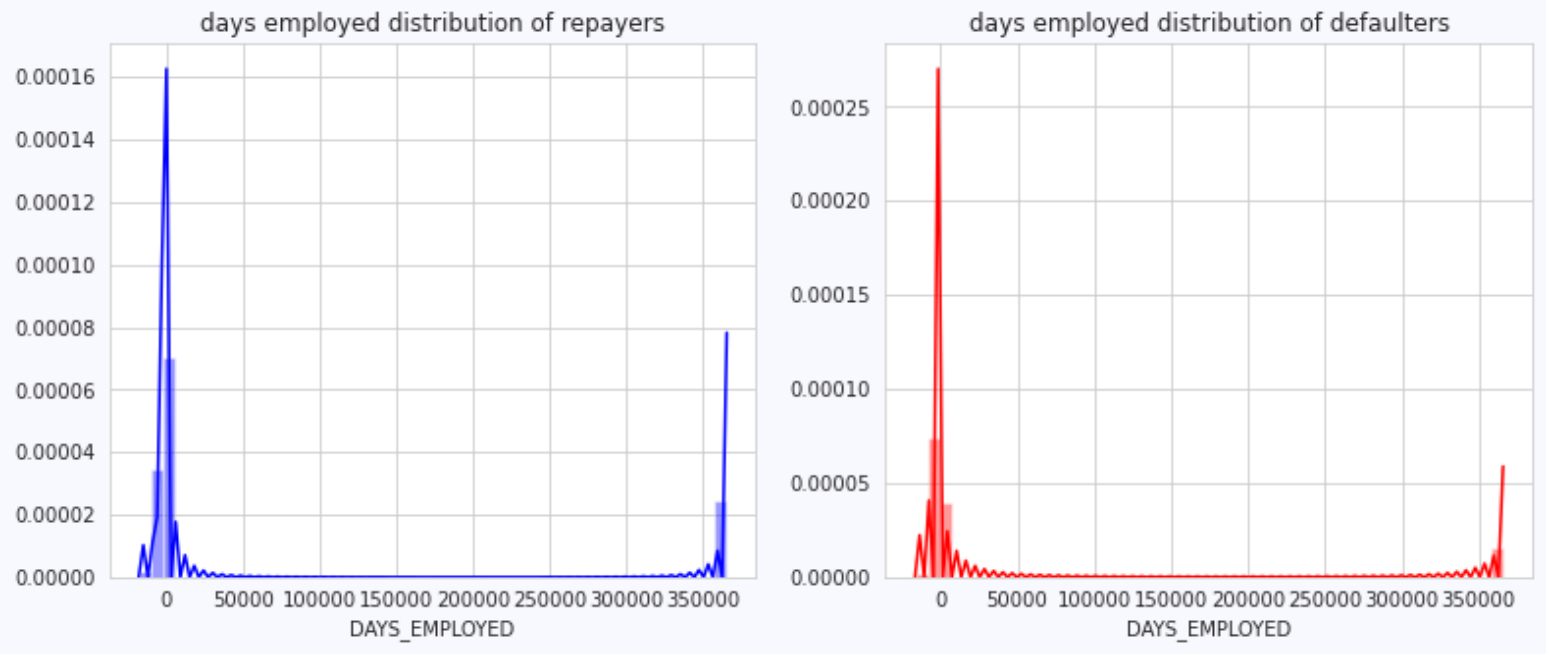
Average clients age is comparatively less in non repayers than repayers in every aspect.

Younger people tend to default more than elder people.

Distribution of days employed for target variable.

DAYS_EMPLOYED - How many days before the application for target variable the person started current employment

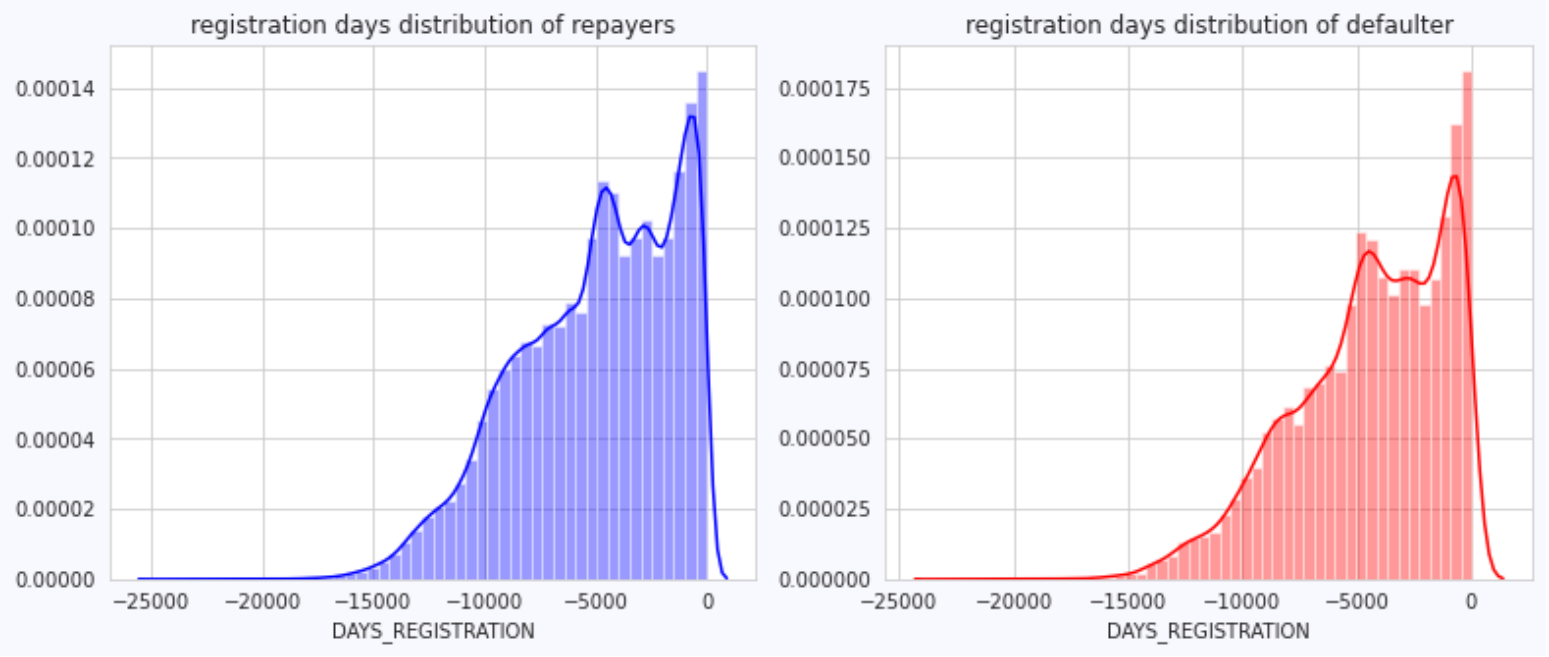
```
fig = plt.figure(figsize=(13,5))

plt.subplot(121)
sns.distplot(application_data[application_data["TARGET"]==0]["DAYS_EMPLOYED"],color="b")
plt.title("days employed distribution of repayers")

plt.subplot(122)
sns.distplot(application_data[application_data["TARGET"]==1]["DAYS_EMPLOYED"],color="r")
plt.title("days employed distribution of defaulters")

fig.set_facecolor("ghostwhite")
```



Distribution of registration days for target variable.

DAYS_REGISTRATION How many days before the application did client change his registration

```
fig = plt.figure(figsize=(13,5))

plt.subplot(121)
sns.distplot(application_data[application_data["TARGET"]==0]["DAYS_REGISTRATION"],color="b")
plt.title("registration days distribution of repayers")

plt.subplot(122)
sns.distplot(application_data[application_data["TARGET"]==1]["DAYS_REGISTRATION"],color="r")
plt.title("registration days distribution of defaulter")

fig.set_facecolor("ghostwhite")
```



Distribution in contact information provided by client

FLAG_MOBIL - Did client provide mobile phone (1=YES, 0=NO)

FLAG_EMP_PHONE - Did client provide work phone (1=YES, 0=NO)

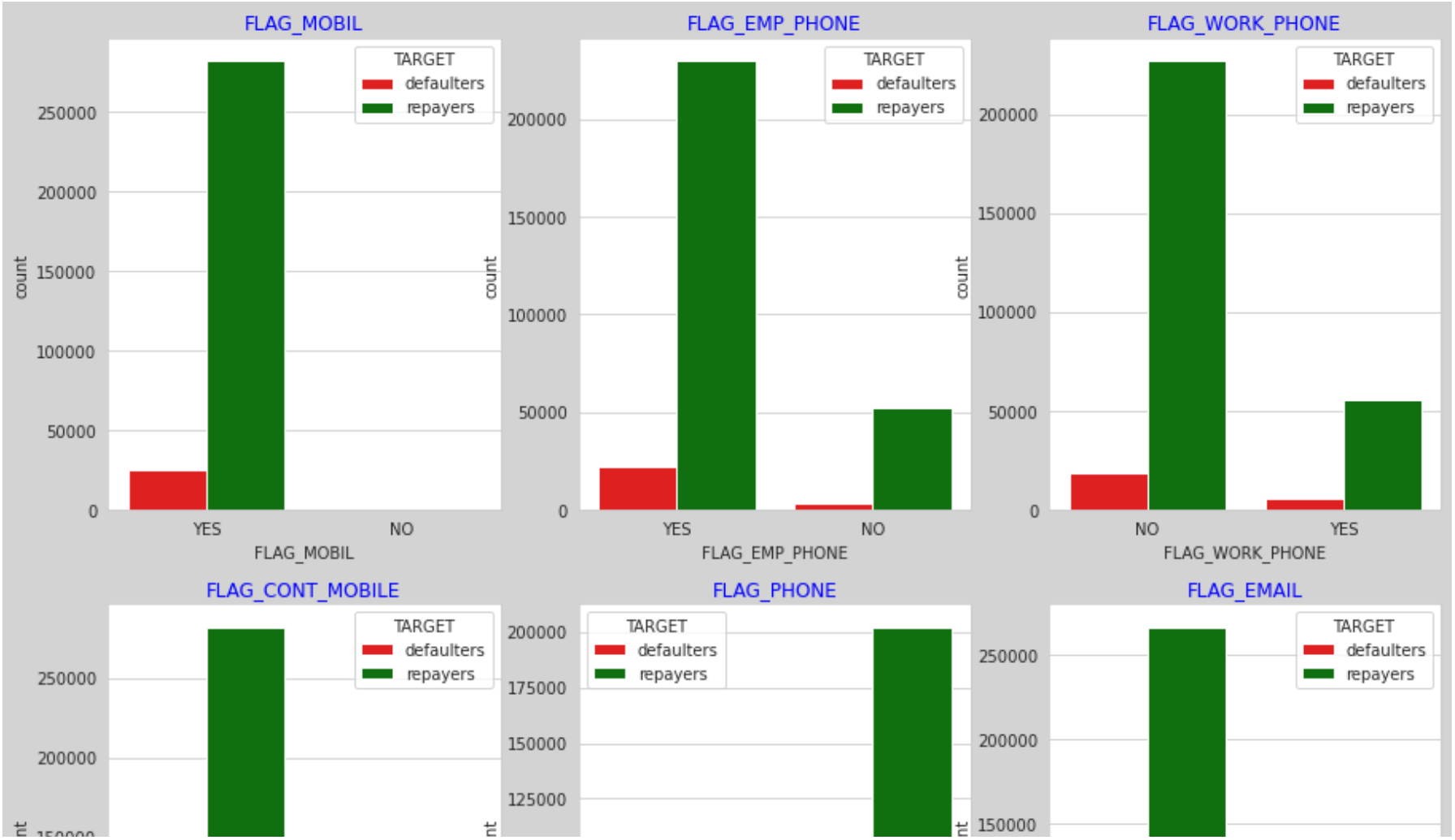
FLAG_WORK_PHONE - Did client provide home phone (1=YES, 0=NO)

FLAG_CONT_MOBILE - Was mobile phone reachable (1=YES, 0=NO)

FLAG_PHONE - Did client provide home phone (1=YES, 0=NO)

FLAG_EMAIL - Did client provide email (1=YES, 0=NO)

```
x   = application_data[['FLAG_MOBIL', 'FLAG_EMP_PHONE', 'FLAG_WORK_PHONE', 'FLAG_CONT_MOBILE',
       'FLAG_PHONE', 'FLAG_EMAIL',"TARGET"]]
x["TARGET"] = x["TARGET"].replace({0:"repayers",1:"defaulters"})
x  = x.replace({1:"YES",0:"NO"})

cols = ['FLAG_MOBIL', 'FLAG_EMP_PHONE', 'FLAG_WORK_PHONE', 'FLAG_CONT_MOBILE',
       'FLAG_PHONE', 'FLAG_EMAIL']
length = len(cols)

fig = plt.figure(figsize=(15,12))
fig.set_facecolor("lightgrey")

for i,j in itertools.zip_longest(cols,range(length)):
    plt.subplot(2,3,j+1)
    sns.countplot(x[i],hue=x["TARGET"],palette=["r","g"])
    plt.title(i,color="b")
```



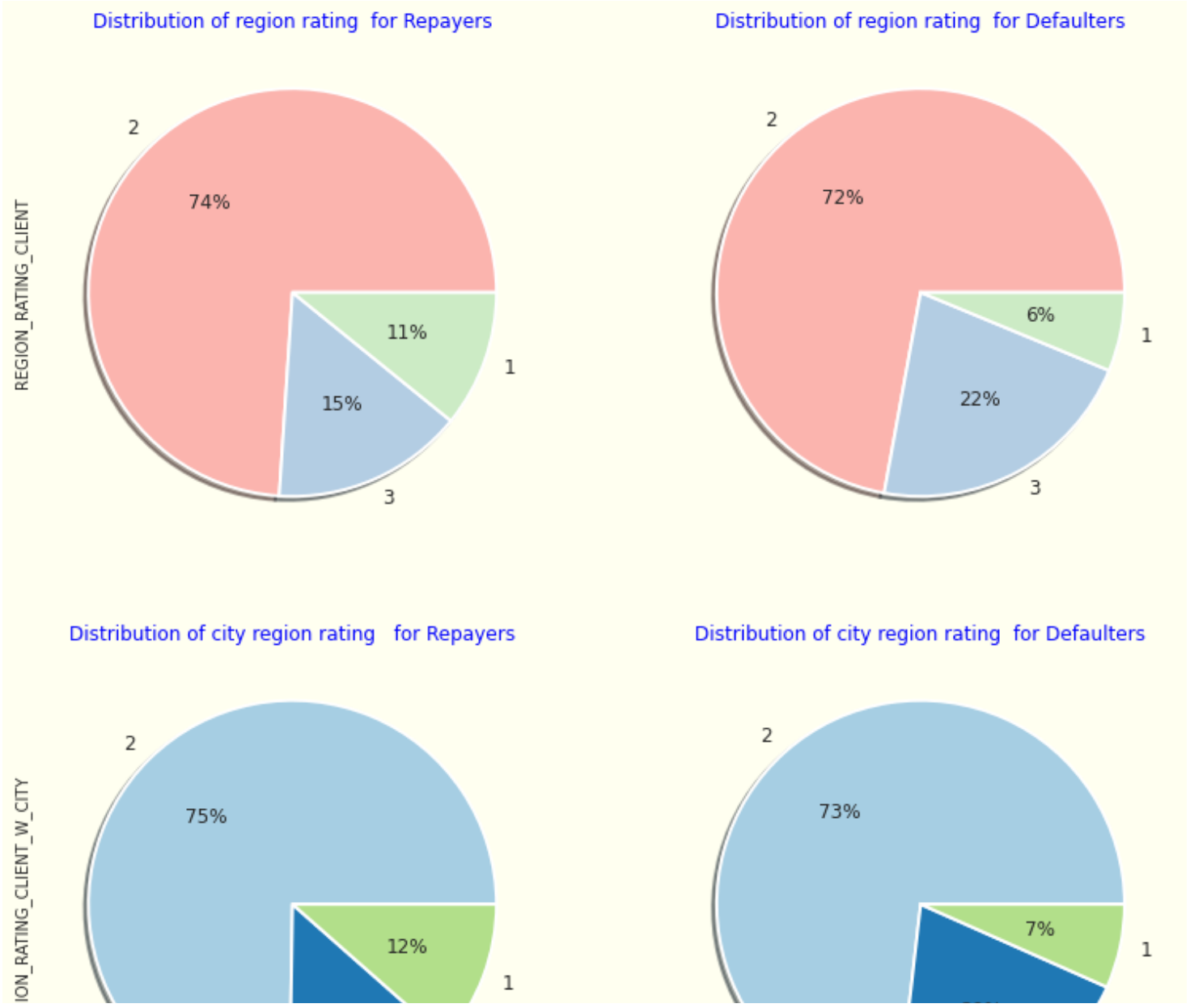
Distribution of registration days for target variable.

REGION_RATING_CLIENT - Home credit rating of the region where client lives (1,2,3).

REGION_RATING_CLIENT_W_CITY - Home credit rating of the region where client lives with taking city into account (1,2,3). Percentage of defaulters are less in 1-rated regions compared to repayers.

```
fig = plt.figure(figsize=(13,13))
plt.subplot(221)
```

```
application_data[application_data["TARGET"]==0]["REGION_RATING_CLIENT"].value_counts().plot.pie(autopct = "%1.0f%%",fontsize=12,
                                                                colors = sns.color_palette("Pastel1"),
                                                    wedgeprops={"linewidth":2,"edgecolor":"white"},shadow =True)

plt.title("Distribution of region rating  for Repayers",color="b")

plt.subplot(222)
application_data[application_data["TARGET"]==1]["REGION_RATING_CLIENT"].value_counts().plot.pie(autopct = "%1.0f%%",fontsize=12,
                                                       colors = sns.color_palette("Pastel1"),
                                                    wedgeprops={"linewidth":2,"edgecolor":"white"},shadow =True)


plt.title("Distribution of region rating  for Defaulters",color="b")
plt.ylabel("")

plt.subplot(223)
application_data[application_data["TARGET"]==0]["REGION_RATING_CLIENT_W_CITY"].value_counts().plot.pie(autopct = "%1.0f%%",fontsize=12,
                                                                colors = sns.color_palette("Paired"),
                                                    wedgeprops={"linewidth":2,"edgecolor":"white"},shadow =True)

plt.title("Distribution of city region rating   for Repayers",color="b")

plt.subplot(224)
application_data[application_data["TARGET"]==1]["REGION_RATING_CLIENT_W_CITY"].value_counts().plot.pie(autopct = "%1.0f%%",fontsize=12,
                                                       colors = sns.color_palette("Paired"),
                                                    wedgeprops={"linewidth":2,"edgecolor":"white"},shadow =True)


plt.title("Distribution of city region rating  for Defaulters",color="b")
plt.ylabel("")
fig.set_facecolor("ivory")
```



Point to infer from the graph

Percentage of defaulters are less in 1-rated regions compared to repayers.

Percentage of defaulters are more in 3-rated regions compared to repayers.

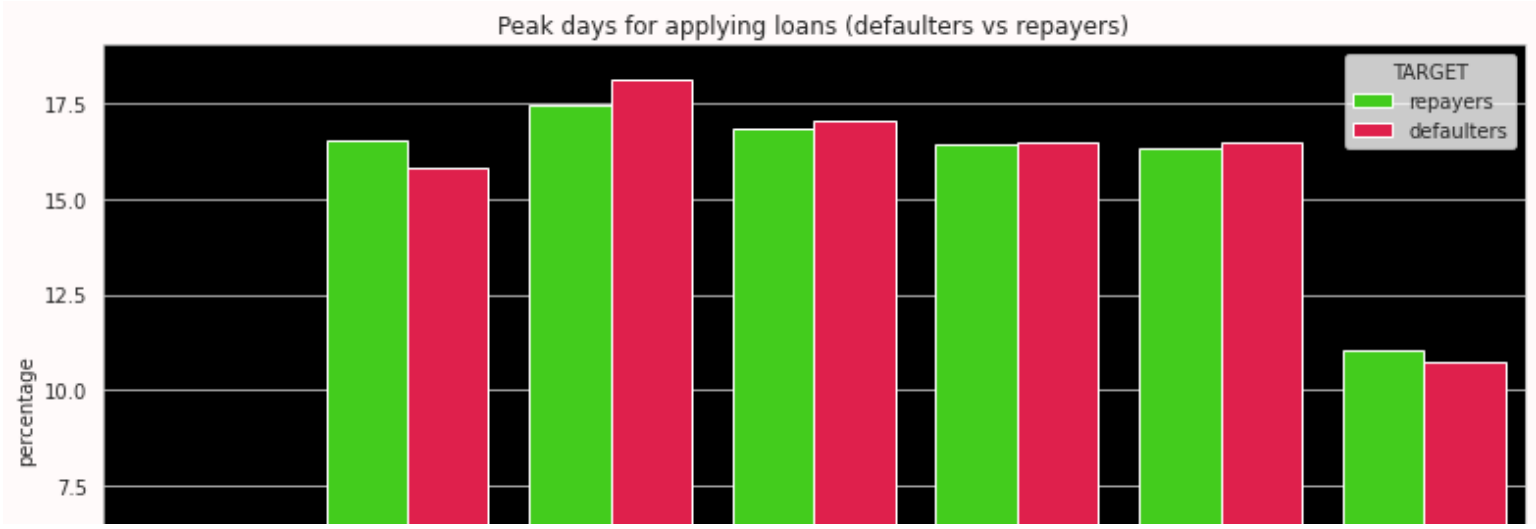
Peak days and hours for applying loans (defaulters vs repayers)

WEEKDAY_APPR_PROCESS_START - On which day of the week did the client apply for the loan.

HOUR_APPR_PROCESS_START - Approximately at what hour did the client apply for the loan.

```
day = application_data.groupby("TARGET").agg({"WEEKDAY_APPR_PROCESS_START":"value_counts"})
day = day.rename(columns={"WEEKDAY_APPR_PROCESS_START":"value_counts"})
day = day.reset_index()
day_0 = day[:7]
day_1 = day[7:]
day_0["percentage"] = day_0["value_counts"]*100/day_0["value_counts"].sum()
day_1["percentage"] = day_1["value_counts"]*100/day_1["value_counts"].sum()
days = pd.concat([day_0,day_1],axis=0)
days["TARGET"] = days.replace({1:"defaulters",0:"repayers"})

fig = plt.figure(figsize=(13,15))
plt.subplot(211)
order = ['SUNDAY', 'MONDAY','TUESDAY', 'WEDNESDAY','THURSDAY', 'FRIDAY', 'SATURDAY']
ax= sns.barplot("WEEKDAY_APPR_PROCESS_START","percentage",data=days,
                hue="TARGET",order=order,palette="prism")
ax.set_facecolor("k")
ax.set_title("Peak days for applying loans (defaulters vs repayers)")

hr = application_data.groupby("TARGET").agg({"HOUR_APPR_PROCESS_START":"value_counts"})
hr = hr.rename(columns={"HOUR_APPR_PROCESS_START":"value_counts"}).reset_index()
hr_0 = hr[hr["TARGET"]==0]
hr_1 = hr[hr["TARGET"]==1]
hr_0["percentage"] = hr_0["value_counts"]*100/hr_0["value_counts"].sum()
hr_1["percentage"] = hr_1["value_counts"]*100/hr_1["value_counts"].sum()
hrs = pd.concat([hr_0,hr_1],axis=0)
hrs["TARGET"] = hrs["TARGET"].replace({1:"defaulters",0:"repayers"})
hrs = hrs.sort_values(by="HOUR_APPR_PROCESS_START",ascending=True)

plt.subplot(212)
```

```
ax1 = sns.pointplot("HOUR_APPR_PROCESS_START","percentage",
                    data=hrs,hue="TARGET",palette="prism")
ax1.set_facecolor("k")
ax1.set_title("Peak hours for applying loans (defaulters vs repayers)")
fig.set_facecolor("snow")
```



Point to infer from the graph

On tuesdays , percentage of defaulters applying for loans is greater than that of repayers.

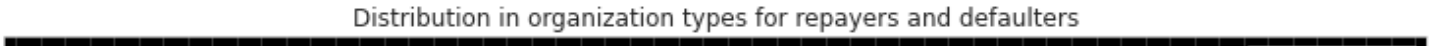
From morning 4'O clock to 9'O clock percentage of defaulters applying for loans is greater than that of repayers.

Distribution in organization types for repayers and defaulters

ORGANIZATION_TYPE - Type of organization where client works.


```
org = application_data.groupby("TARGET").agg({"ORGANIZATION_TYPE":"value_counts"})
org = org.rename(columns = {"ORGANIZATION_TYPE":"value_counts"}).reset_index()
org_0 = org[org["TARGET"] == 0]
org_1 = org[org["TARGET"] == 1]
org_0["percentage"] = org_0["value_counts"]*100/org_0["value_counts"].sum()
org_1["percentage"] = org_1["value_counts"]*100/org_1["value_counts"].sum()

organization = pd.concat([org_0,org_1],axis=0)
organization = organization.sort_values(by="ORGANIZATION_TYPE",ascending=True)

organization["TARGET"] = organization["TARGET"].replace({0:"repayers",1:"defaulters"})
```

```
organization
plt.figure(figsize=(13,7))
ax = sns.pointplot("ORGANIZATION_TYPE","percentage",
                   data=organization,hue="TARGET",palette=["b","r"])
plt.xticks(rotation=90)
plt.grid(True,alpha=.3)
ax.set_facecolor("k")
ax.set_title("Distribution in organization types for repayers and defaulters")
plt.show()
```



Point to infer from the graph

Organizations like Business Entity Type 3,Construction,Self-employed percentage of defaulters are higher than repayers.

Distribution client's social surroundings with observed and defaulted 30 DPD (days past due)

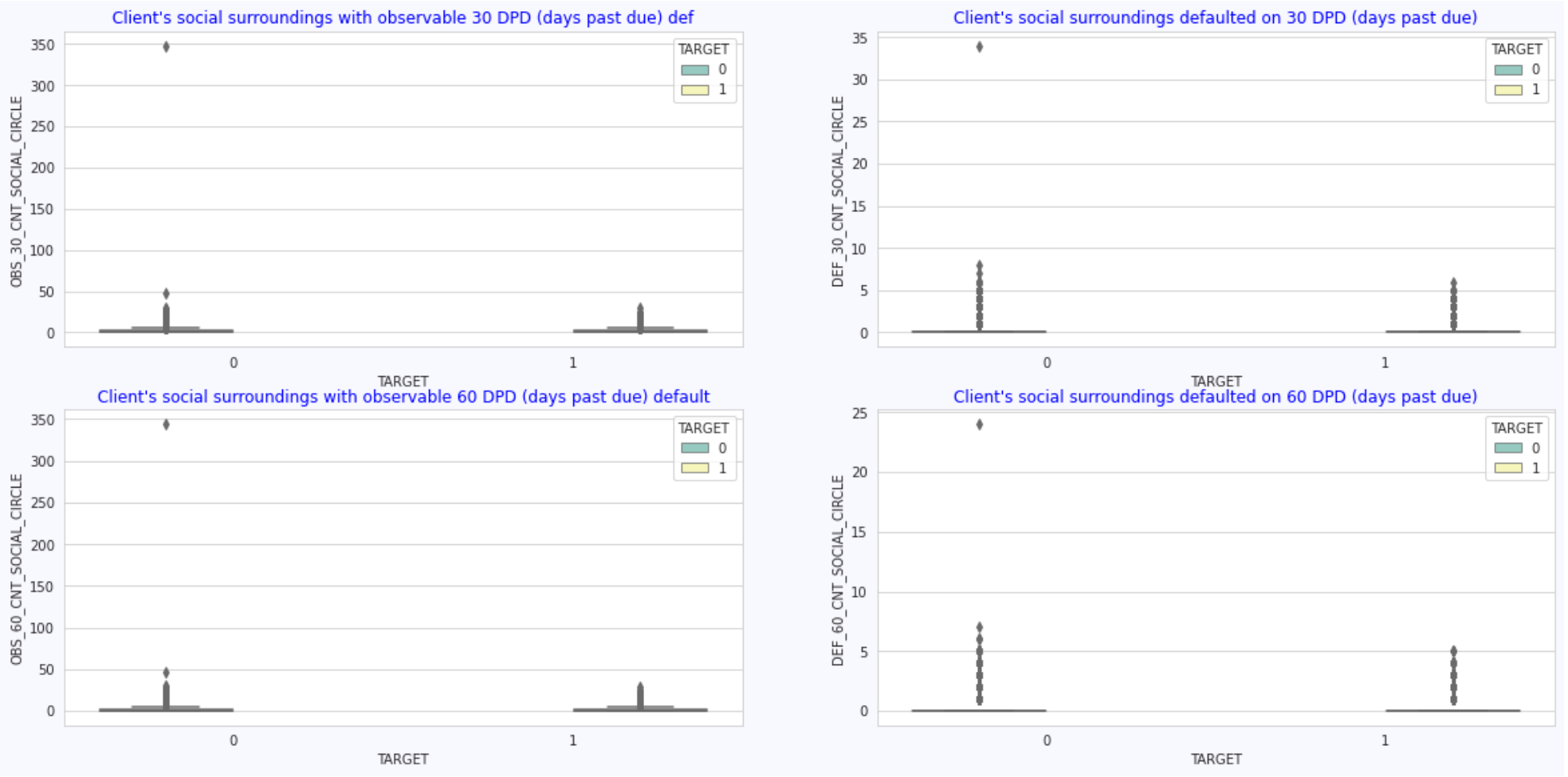
OBS_30_CNT_SOCIAL_CIRCLE- How many observation of client's social surroundings with observable 30 DPD (days past due) default.

DEF_30_CNT_SOCIAL_CIRCLE-How many observation of client's social surroundings defaulted on 30 DPD (days past due) .

OBS_60_CNT_SOCIAL_CIRCLE - How many observation of client's social surroundings with observable 60 DPD (days past due) default.

DEF_60_CNT_SOCIAL_CIRCLE - How many observation of client's social surroundings defaulted on 60 (days past due) DPD.

```
fig = plt.figure(figsize=(20,20))
plt.subplot(421)
sns.boxplot(data=application_data,x='TARGET',y='OBS_30_CNT_SOCIAL_CIRCLE',
            hue="TARGET", palette="Set3")
plt.title("Client's social surroundings with observable 30 DPD (days past due) def",color="b")
plt.subplot(422)
sns.boxplot(data=application_data,x='TARGET',y='DEF_30_CNT_SOCIAL_CIRCLE',
            hue="TARGET", palette="Set3")
plt.title("Client's social surroundings defaulted on 30 DPD (days past due)",color="b")
plt.subplot(423)
sns.boxplot(data=application_data,x='TARGET',y='OBS_60_CNT_SOCIAL_CIRCLE',
            hue="TARGET", palette="Set3")
plt.title("Client's social surroundings with observable 60 DPD (days past due) default",color="b")
plt.subplot(424)
sns.boxplot(data=application_data,x='TARGET',y='DEF_60_CNT_SOCIAL_CIRCLE',
            hue="TARGET", palette="Set3")
plt.title("Client's social surroundings defaulted on 60 DPD (days past due)",color="b")
fig.set_facecolor("ghostwhite")
```

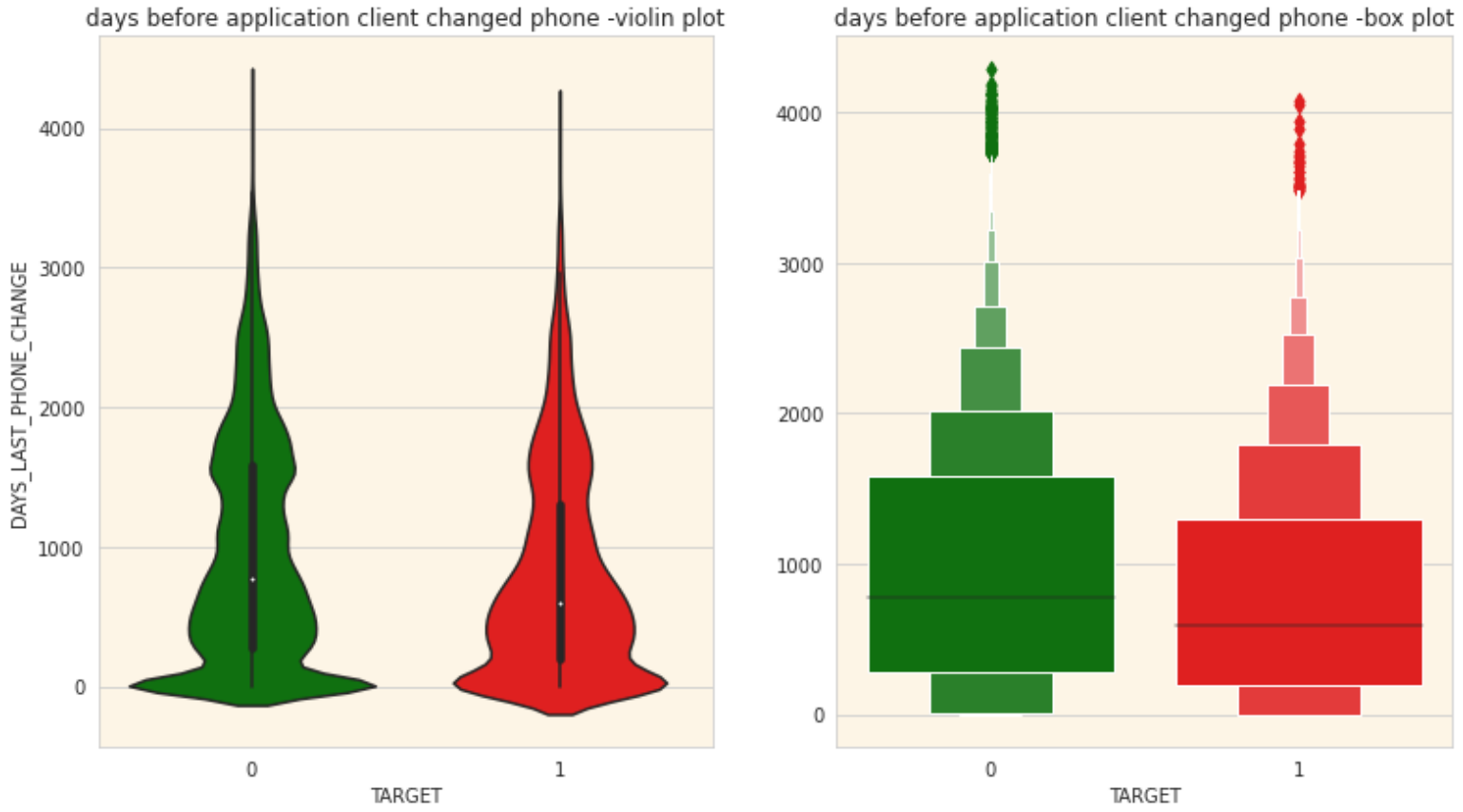


Number of days before application client changed phone .

DAYS_LAST_PHONE_CHANGE - How many days before application did client change phone.

```
plt.figure(figsize=(13,7))
plt.subplot(121)
ax = sns.violinplot(application_data["TARGET"],
                    application_data["DAYS_LAST_PHONE_CHANGE"],palette=["g","r"])
```

```
ax.set_facecolor("oldlace")
ax.set_title("days before application client changed phone -violin plot")
plt.subplot(122)
ax1 = sns.lvplot(application_data["TARGET"],
                 application_data["DAYS_LAST_PHONE_CHANGE"],palette=["g","r"])
ax1.set_facecolor("oldlace")
ax1.set_ylabel("")
ax1.set_title("days before application client changed phone -box plot")
plt.subplots_adjust(wspace = .2)
```



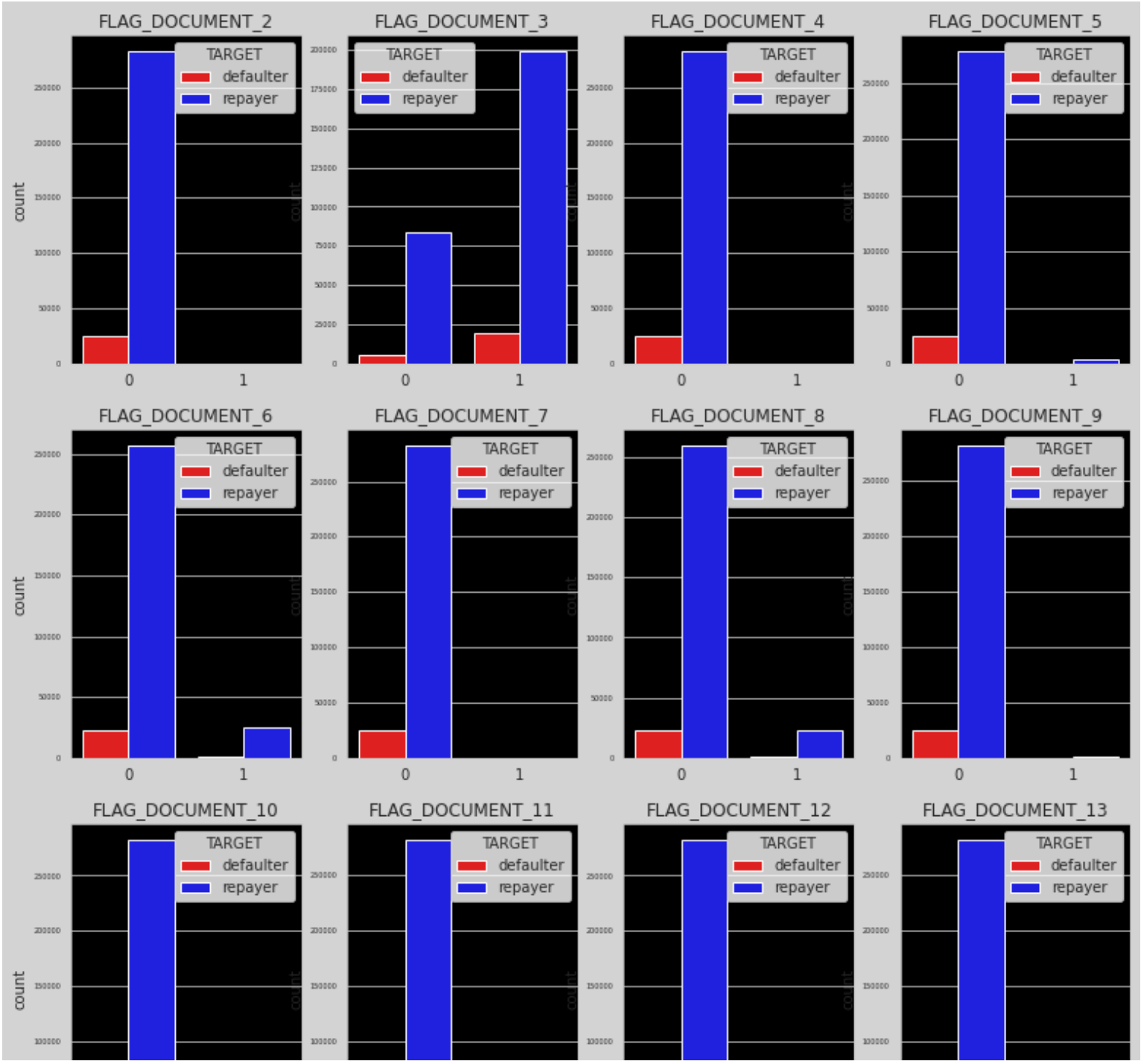
Point to infer from the graph

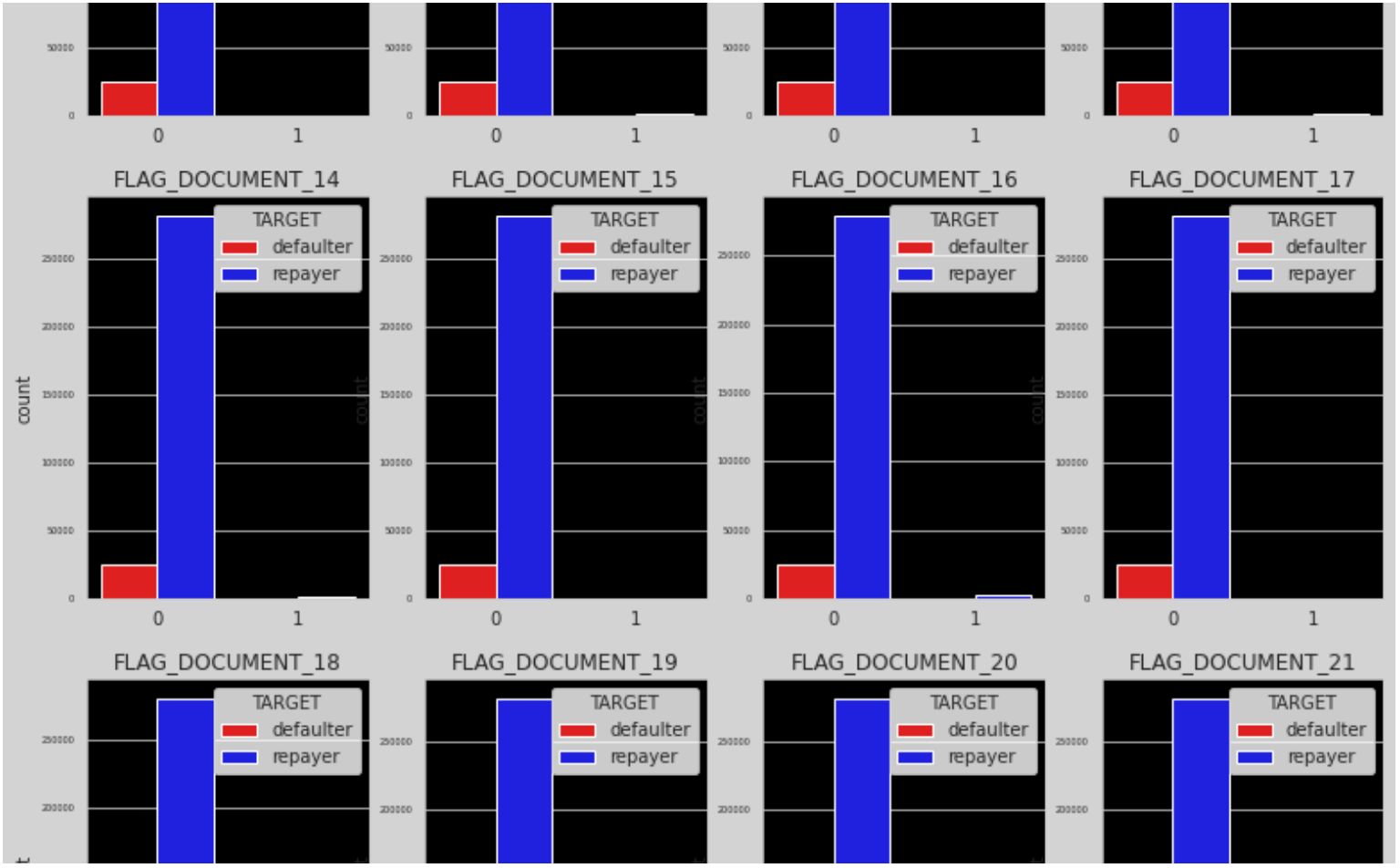
Average days of defaulters phone change is less than average days of repayers phone change.

Documents provided by the clients.

FLAG_DOCUMENT - Did client provide documents.(1,0)

```
cols = [ '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']

df_flag = application_data[cols+["TARGET"]]

length = len(cols)

df_flag["TARGET"] = df_flag["TARGET"].replace({1:"defaulter",0:"repayer"})

fig = plt.figure(figsize=(13,24))
fig.set_facecolor("lightgrey")
for i,j in itertools.zip_longest(cols,range(length)):
    plt.subplot(5,4,j+1)
    ax = sns.countplot(df_flag[i],hue=df_flag["TARGET"],palette=["r","b"])
    plt.yticks(fontsize=5)
    plt.xlabel("")
    plt.title(i)
    ax.set_facecolor("k")
```


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
TARGET
defaulter
repayer
count
0
1



Equiries to Credit Bureau about the client before application.

AMT_REQ_CREDIT_BUREAU_HOUR - Number of enquiries to Credit Bureau about the client one hour before application.

AMT_REQ_CREDIT_BUREAU_DAY - Number of enquiries to Credit Bureau about the client one day before application (excluding one hour before application).

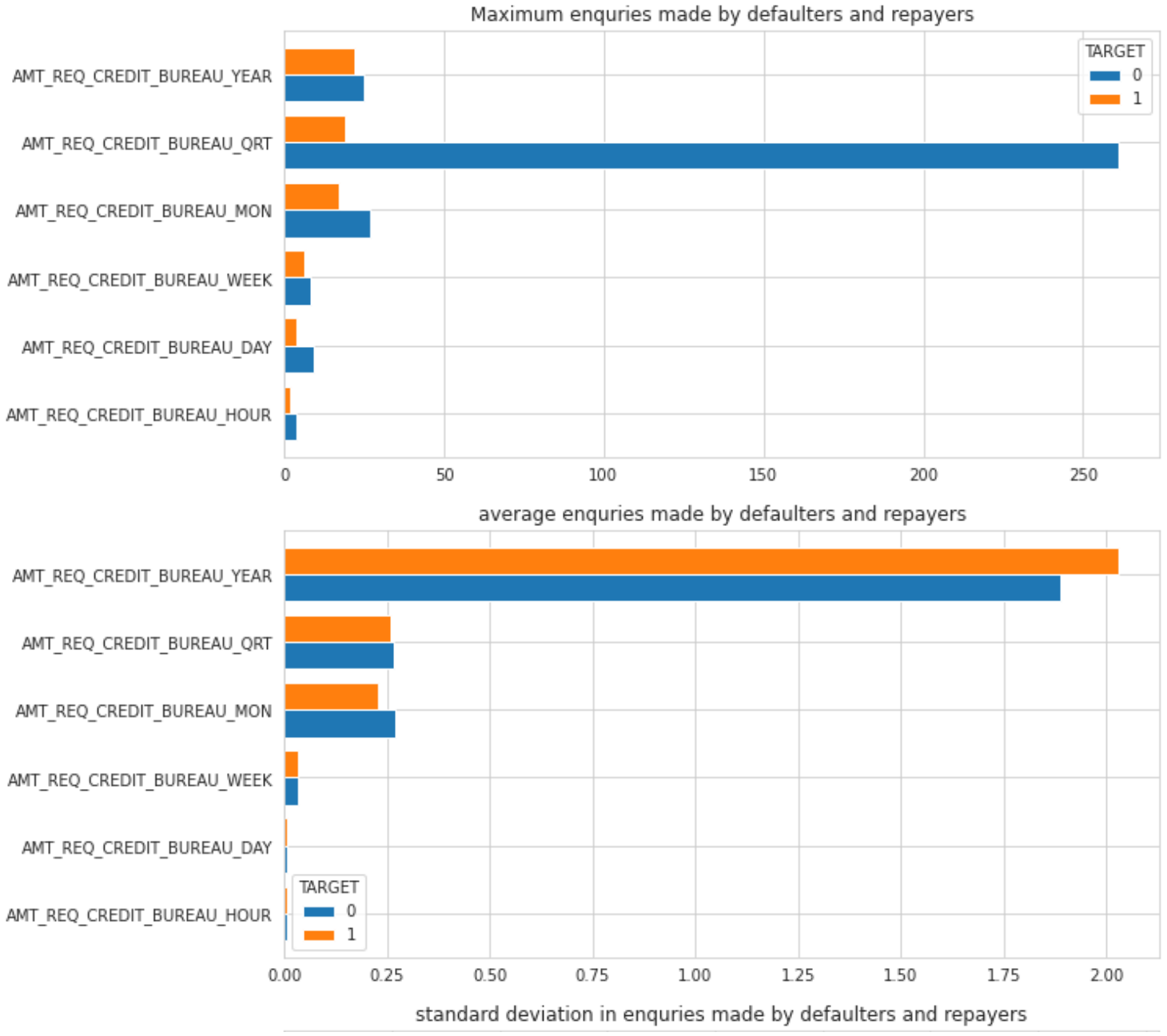
AMT_REQ_CREDIT_BUREAU_WEEK - Number of enquiries to Credit Bureau about the client one week before application (excluding one day before application).

AMT_REQ_CREDIT_BUREAU_MON - Number of enquiries to Credit Bureau about the client one month before application (excluding one week before application).

AMT_REQ_CREDIT_BUREAU_QRT - Number of enquiries to Credit Bureau about the client 3 month before application (excluding one month before application).

AMT_REQ_CREDIT_BUREAU_YEAR - Number of enquiries to Credit Bureau about the client one day year (excluding last 3 months before application).

```
cols = ['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']
application_data.groupby("TARGET")[cols].max().transpose().plot(kind="barh",
                                                                figsize=(10,5),width=.8)
plt.title("Maximum enquries made by defaulters and repayers")
application_data.groupby("TARGET")[cols].mean().transpose().plot(kind="barh",
                                                                 figsize=(10,5),width=.8)
plt.title("average enquries made by defaulters and repayers")
application_data.groupby("TARGET")[cols].std().transpose().plot(kind="barh",
                                                                figsize=(10,5),width=.8)
plt.title("standard deviation in enquries made by defaulters and repayers")
plt.show()
```

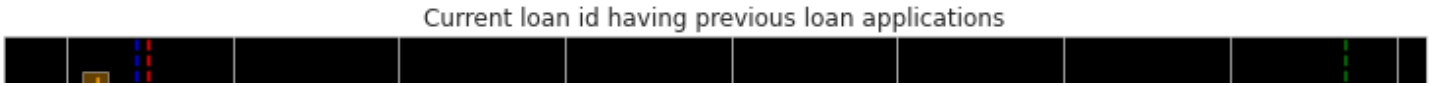


- Current loan id having previous loan applications.

SK_ID_PREV - ID of previous credit in Home credit related to loan in our sample. (One loan in our sample can have 0,1,2 or more previous loan applications in Home Credit, previous application could, but not necessarily have to lead to credit).

SK_ID_CURR ID of loan in our sample.

```
x = previous_application.groupby("SK_ID_CURR")["SK_ID_PREV"].count().reset_index()
plt.figure(figsize=(13,7))
ax = sns.distplot(x["SK_ID_PREV"],color="orange")
plt.axvline(x["SK_ID_PREV"].mean(),linestyle="dashed",color="r",label="average")
plt.axvline(x["SK_ID_PREV"].std(),linestyle="dashed",color="b",label="standard deviation")
plt.axvline(x["SK_ID_PREV"].max(),linestyle="dashed",color="g",label="maximum")
plt.legend(loc="best")
plt.title("Current loan id having previous loan applications")
ax.set_facecolor("k")
```



Point to infer from the graph

On average current loan ids have 4 to 5 loan applications previously

Contract types in previous applications

NAME_CONTRACT_TYPE Contract product type (Cash loan, consumer loan [POS] ,...) of the previous application.


```
cnts = previous_application["NAME_CONTRACT_TYPE"].value_counts()
import squarify
plt.figure(figsize=(8,6))
squarify.plot(cnts.values,label=cnts.keys(),value=cnts.values,linewidth=2,edgecolor="k",alpha=.8,color=sns.color_palette("Set1"))
plt.axis("off")
plt.title("Contaract types in previous applications")
plt.show()
```

Point to infer from the graph

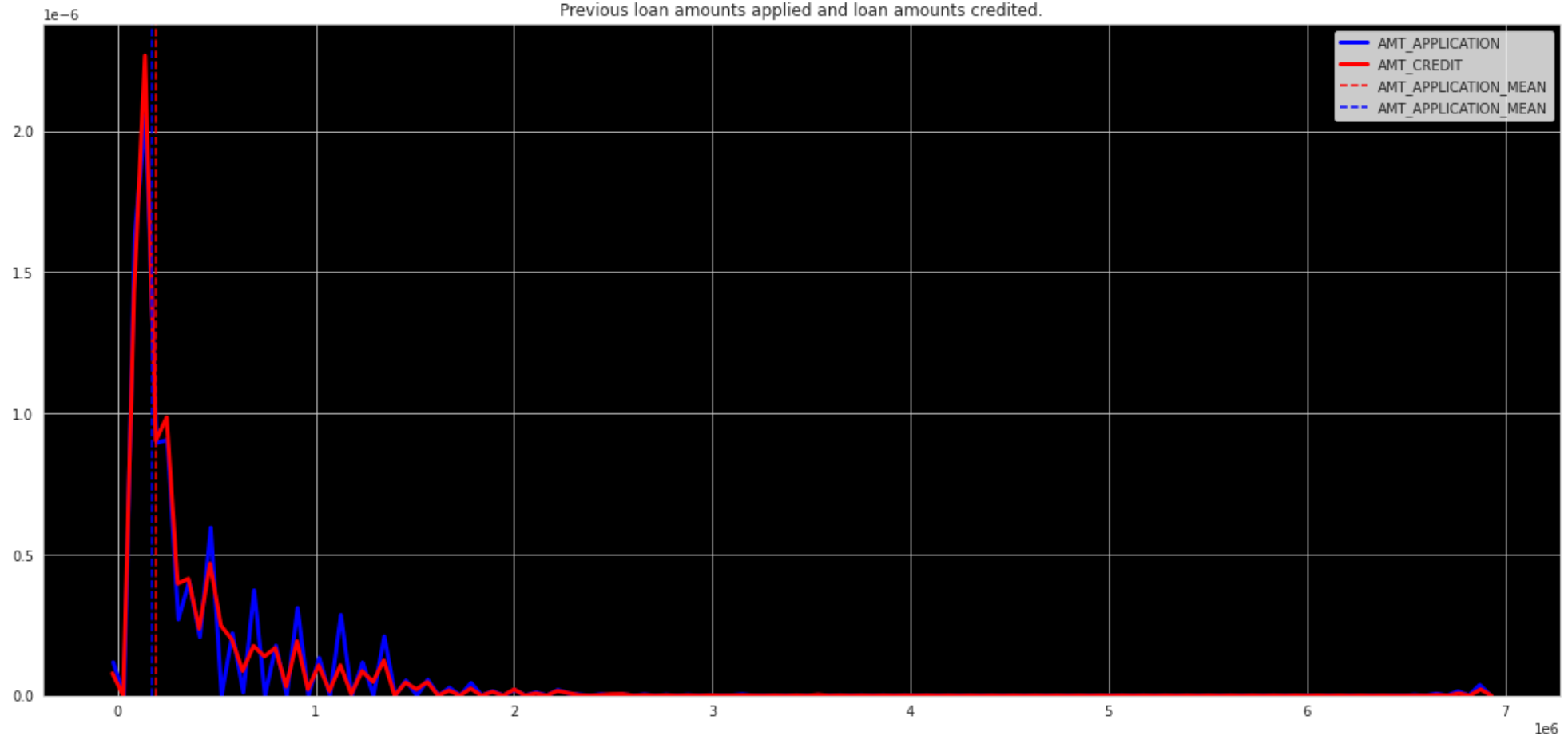
Cash loan applications are maximum followed by consumer loan applications.

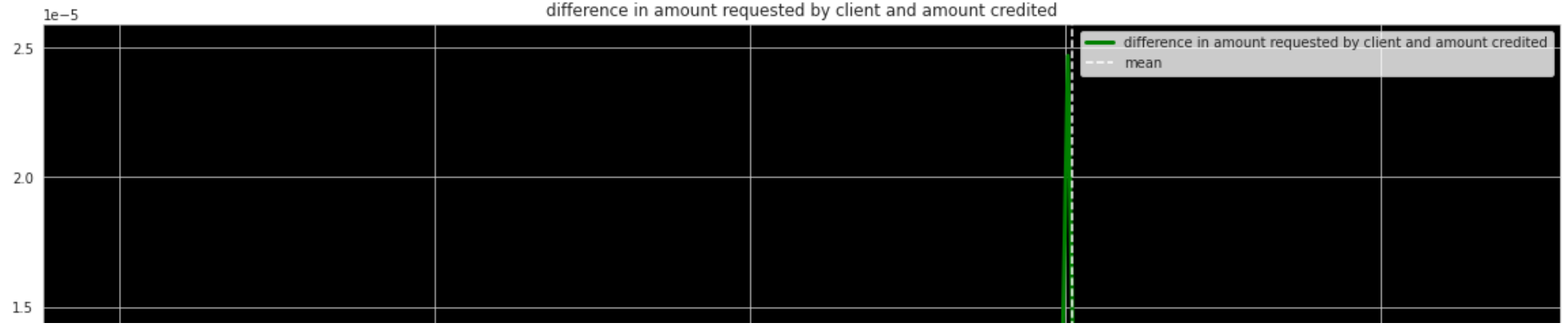
Previous loan amounts applied and loan amounts credited.

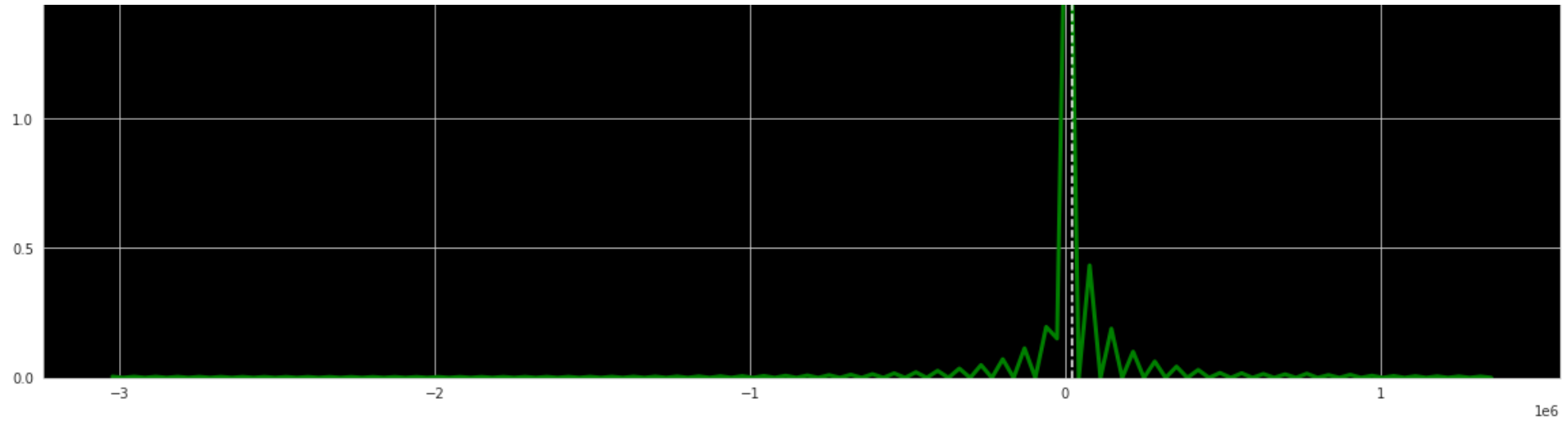
AMT_APPLICATION-For how much credit did client ask on the previous application.

AMT_CREDIT-Final credit amount on the previous application. This differs from AMT_APPLICATION in a way that the AMT_APPLICATION is the amount for which the client initially applied for, but during our approval process he could have received different amount - AMT_CREDIT.

```
plt.figure(figsize=(20,20))
plt.subplot(211)
ax = sns.kdeplot(previous_application["AMT_APPLICATION"],color="b",linewidth=3)
ax = sns.kdeplot(previous_application[previous_application["AMT_CREDIT"].notnull()]["AMT_CREDIT"],color="r",linewidth=3)
plt.axvline(previous_application[previous_application["AMT_CREDIT"].notnull()]["AMT_CREDIT"].mean(),color="r",linestyle="dashed",label="
plt.axvline(previous_application["AMT_APPLICATION"].mean(),color="b",linestyle="dashed",label="AMT_APPLICATION_MEAN")
plt.legend(loc="best")
plt.title("Previous loan amounts applied and loan amounts credited.")
ax.set_facecolor("k")

plt.subplot(212)
diff = (previous_application["AMT_CREDIT"] - previous_application["AMT_APPLICATION"]).reset_index()
diff = diff[diff[0].notnull()]
ax1 = sns.kdeplot(diff[0],color="g",linewidth=3,label = "difference in amount requested by client and amount credited")
plt.axvline(diff[0].mean(),color="white",linestyle="dashed",label = "mean")
plt.title("difference in amount requested by client and amount credited")
ax1.legend(loc="best")
ax1.set_facecolor("k")
```


Previous loan amounts applied and loan amounts credited.
1e−6
AMT_APPLICATION
AMT_CREDIT
AMT_APPLICATION_MEAN
AMT_APPLICATION_MEAN
2.0
1.5
1.0
0.5
0.0
0
1
2
3
4
5
6
7
1e6


difference in amount requested by client and amount credited
1e−5
difference in amount requested by client and amount credited
mean
2.5
2.0
1.5

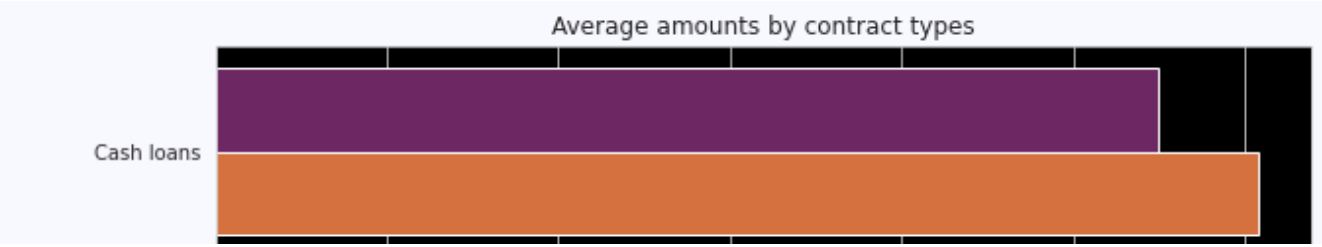


Total and average amounts applied and credited in previous applications

AMT_APPLICATION-For how much credit did client ask on the previous application. >AMT_CREDIT-Final credit amount on the previous application. This differs from AMT_APPLICATION in a way that the AMT_APPLICATION is the amount for which the client.

```
mn = previous_application.groupby("NAME_CONTRACT_TYPE")[["AMT_APPLICATION","AMT_CREDIT"]].mean().stack().reset_index()
tt = previous_application.groupby("NAME_CONTRACT_TYPE")[["AMT_APPLICATION","AMT_CREDIT"]].sum().stack().reset_index()
fig = plt.figure(figsize=(10,13))
fig.set_facecolor("ghostwhite")
plt.subplot(211)
ax = sns.barplot(0,"NAME_CONTRACT_TYPE",data=mn[:6],hue="level_1",palette="inferno")
ax.set_facecolor("k")
ax.set_xlabel("average amounts")
ax.set_title("Average amounts by contract types")

plt.subplot(212)
ax1 = sns.barplot(0,"NAME_CONTRACT_TYPE",data=tt[:6],hue="level_1",palette="magma")
ax1.set_facecolor("k")
ax1.set_xlabel("total amounts")
```

```
ax1.set_title("total amounts by contract types")
plt.subplots_adjust(hspace = .2)
plt.show()
```



Annuity of previous application

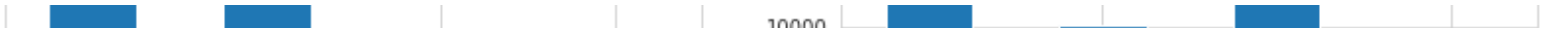
AMT_ANNUITY - Annuity of previous application

```
plt.figure(figsize=(14,5))
plt.subplot(121)
previous_application.groupby("NAME_CONTRACT_TYPE")["AMT_ANNUITY"].sum().plot(kind="bar")
plt.xticks(rotation=0)
plt.title("Total annuity amount by contract types in previous applications")
plt.subplot(122)
previous_application.groupby("NAME_CONTRACT_TYPE")["AMT_ANNUITY"].mean().plot(kind="bar")
plt.title("average annuity amount by contract types in previous applications")
plt.xticks(rotation=0)
plt.show()
```

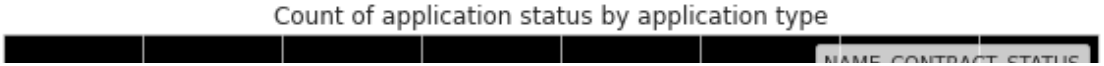
Count of application status by application type.

NAME_CONTRACT_TYPE -Contract product type (Cash loan, consumer loan [POS] ,...) of the previous application.

NAME_CONTRACT_STATUS -Contract status (approved, cancelled, ...) of previous application.




```
ax = pd.crosstab(previous_application["NAME_CONTRACT_TYPE"],previous_application["NAME_CONTRACT_STATUS"]).plot(kind="barh",figsize=(10,7
plt.xticks(rotation =0)
plt.ylabel("count")
plt.title("Count of application status by application type")
ax.set_facecolor("k")
```



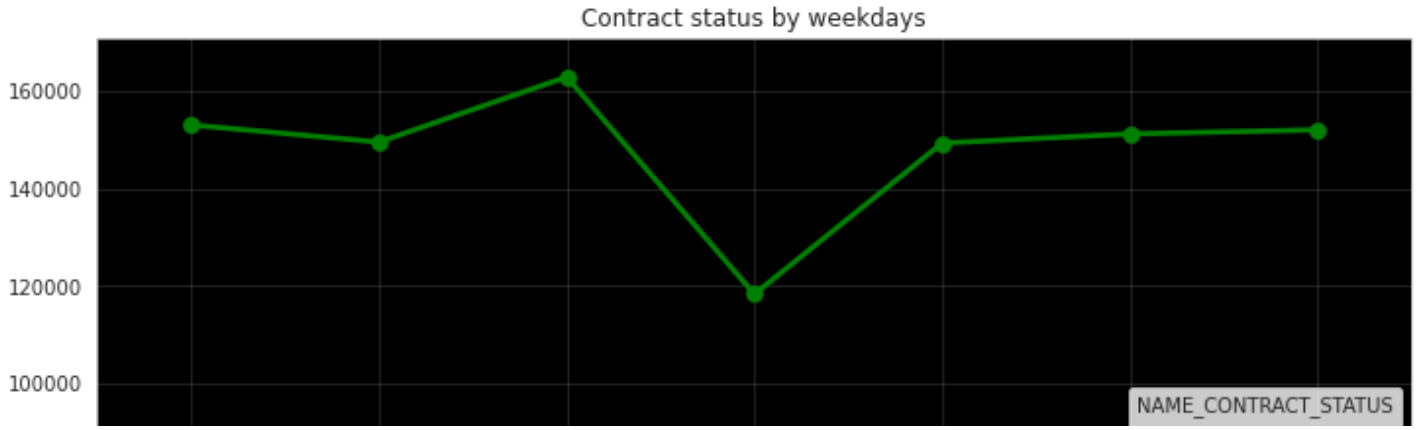
Point to infer from the graph

Consumer loan applications are most approved loans and cash loans are most cancelled and refused loans.

Contract status by weekdays

WEEKDAY_APPR_PROCESS_START - On which day of the week did the client apply for previous application

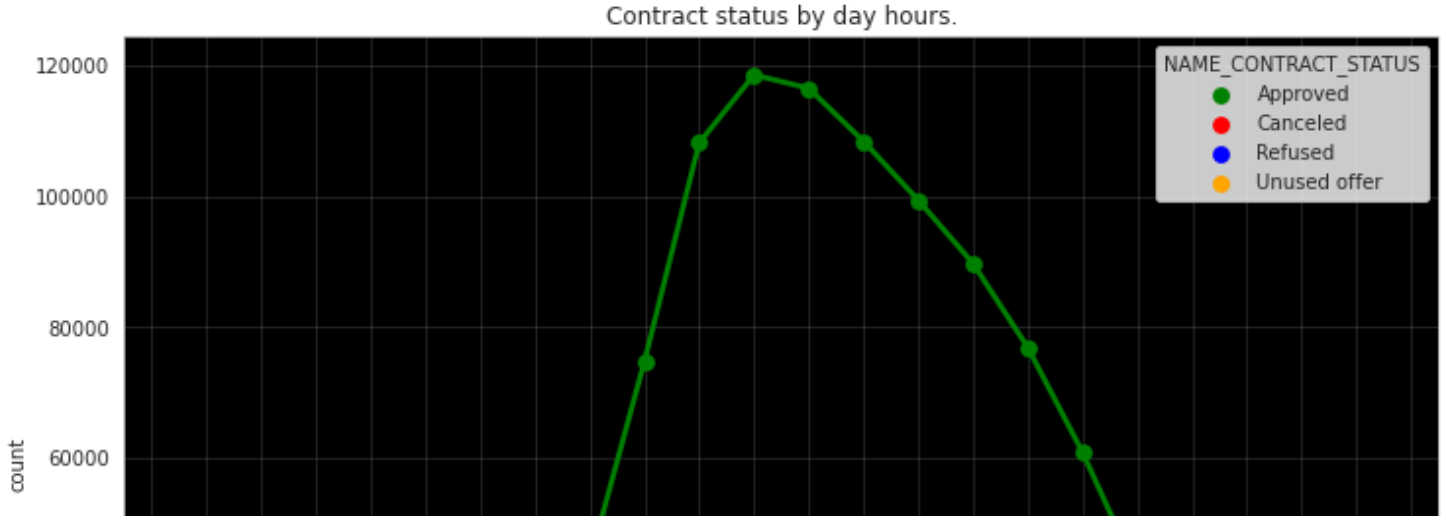
```
hr = pd.crosstab(previous_application["WEEKDAY_APPR_PROCESS_START"],previous_application["NAME_CONTRACT_STATUS"]).stack().reset_index()
plt.figure(figsize=(12,8))
ax = sns.pointplot(hr["WEEKDAY_APPR_PROCESS_START"],hr[0],hue=hr["NAME_CONTRACT_STATUS"],palette=["g","r","b","orange"],scale=1)
ax.set_facecolor("k")
ax.set_ylabel("count")
ax.set_title("Contract status by weekdays")
plt.grid(True,alpha=.2)
```



Contract status by hour of the day

HOUR_APPR_PROCESS_START - Approximately at what day hour did the client apply for the previous application.

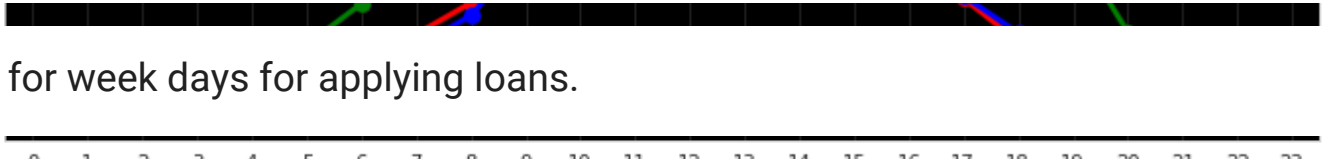
```
hr = pd.crosstab(previous_application["HOUR_APPR_PROCESS_START"],previous_application["NAME_CONTRACT_STATUS"]).stack().reset_index()
plt.figure(figsize=(12,8))
ax = sns.pointplot(hr["HOUR_APPR_PROCESS_START"],hr[0],hue=hr["NAME_CONTRACT_STATUS"],palette=["g","r","b","orange"],scale=1)
ax.set_facecolor("k")
ax.set_ylabel("count")
ax.set_title("Contract status by day hours.")
plt.grid(True,alpha=.2)
```



Point to infer from the graph

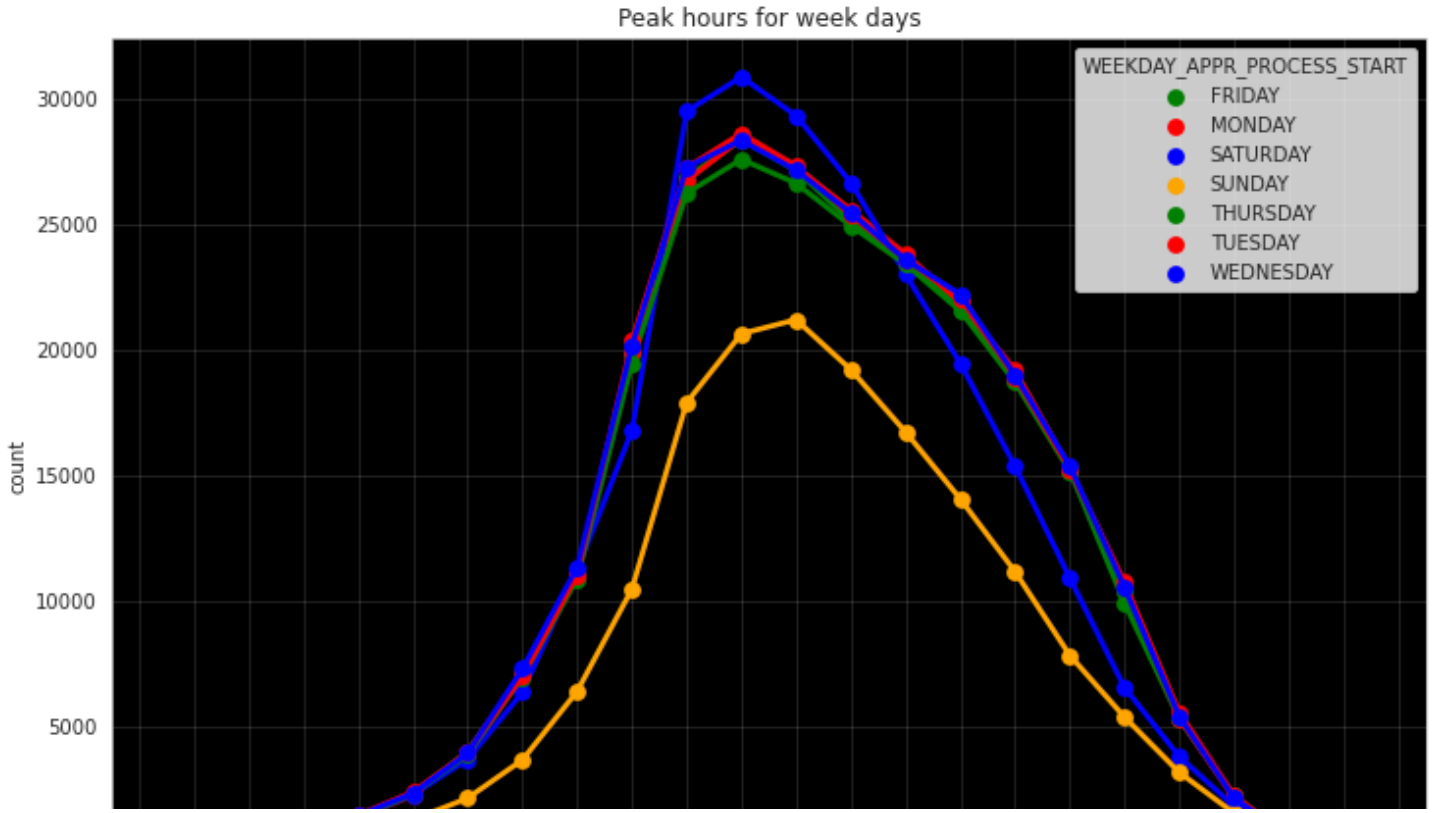
Morning 11'o clock have maximum number of approvals.

Morning 10'o clock have maximum number of refused and cancelled contracts.



Peak hours for week days for applying loans.


```
hr = pd.crosstab(previous_application["HOUR_APPR_PROCESS_START"],previous_application["WEEKDAY_APPR_PROCESS_START"]).stack().reset_index
plt.figure(figsize=(12,8))
ax = sns.pointplot(hr["HOUR_APPR_PROCESS_START"],hr[0],hue=hr["WEEKDAY_APPR_PROCESS_START"],palette=["g","r","b","orange"],scale=1)
ax.set_facecolor("k")
ax.set_ylabel("count")
ax.set_title("Peak hours for week days")
plt.grid(True,alpha=.2)
```



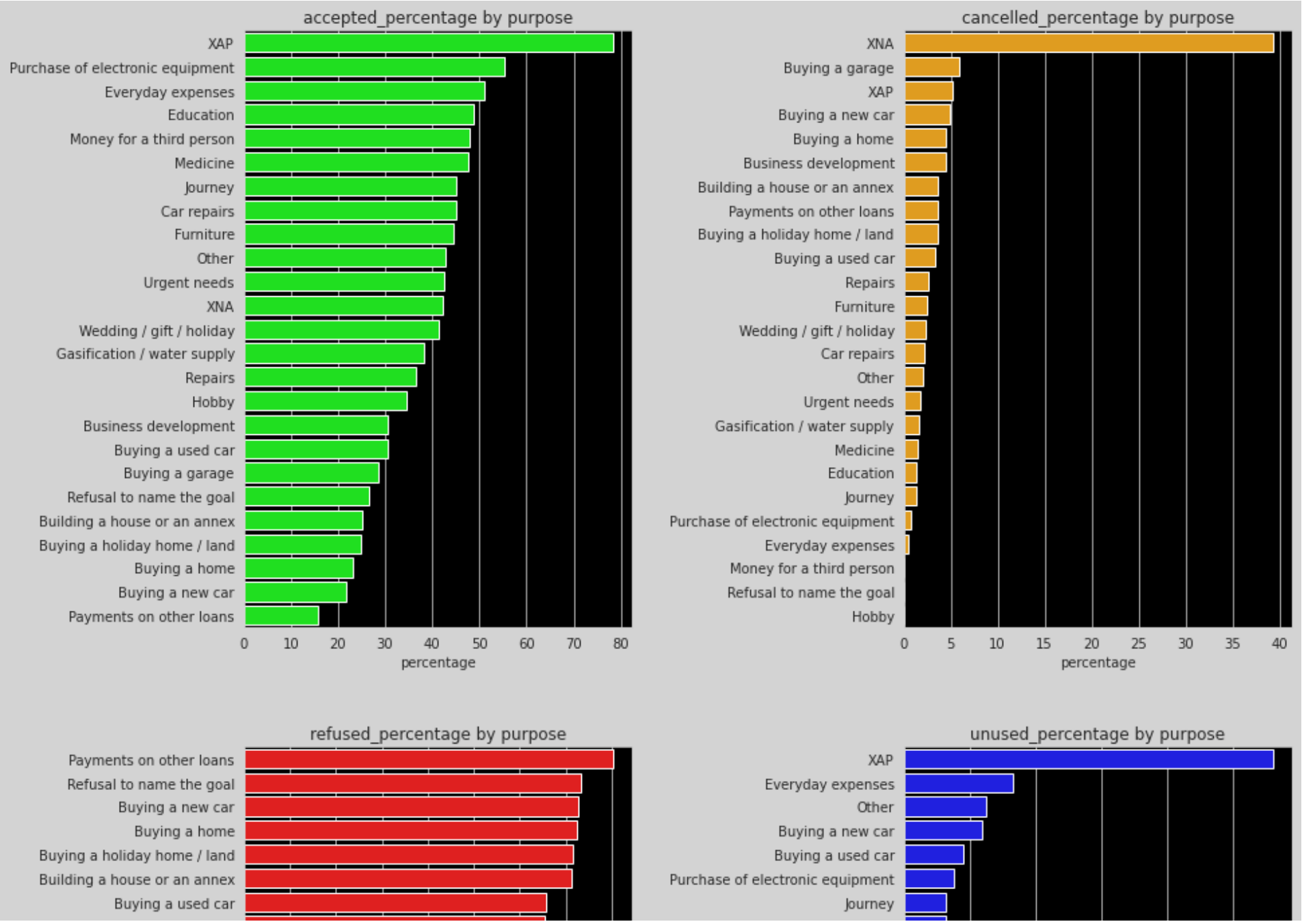
Percentage of applications accepted,cancelled,refused and unused for different loan purposes.

NAME_CASH_LOAN_PURPOSE - Purpose of the cash loan.

NAME_CONTRACT_STATUS - Contract status (approved, cancelled, ...) of previous application.

```
previous_application[["NAME_CASH_LOAN_PURPOSE","NAME_CONTRACT_STATUS"]]
purpose = pd.crosstab(previous_application["NAME_CASH_LOAN_PURPOSE"],previous_application["NAME_CONTRACT_STATUS"])
purpose["a"] = (purpose["Approved"]*100)/(purpose["Approved"]+purpose["Canceled"]+purpose["Refused"]+purpose["Unused offer"])
purpose["c"] = (purpose["Canceled"]*100)/(purpose["Approved"]+purpose["Canceled"]+purpose["Refused"]+purpose["Unused offer"])
purpose["r"] = (purpose["Refused"]*100)/(purpose["Approved"]+purpose["Canceled"]+purpose["Refused"]+purpose["Unused offer"])
purpose["u"] = (purpose["Unused offer"]*100)/(purpose["Approved"]+purpose["Canceled"]+purpose["Refused"]+purpose["Unused offer"])
purpose_new = purpose[["a","c","r","u"]]
purpose_new = purpose_new.stack().reset_index()
purpose_new["NAME_CONTRACT_STATUS"] = purpose_new["NAME_CONTRACT_STATUS"].replace({"a":"accepted percentage","c":"cancelled percentage",
```

```
                                                                    "r":"refused_percentage","u":"unused_percentage"})

lst = purpose_new["NAME_CONTRACT_STATUS"].unique().tolist()
length = len(lst)
cs = ["lime","orange","r","b"]

fig = plt.figure(figsize=(14,18))
fig.set_facecolor("lightgrey")
for i,j,k in itertools.zip_longest(lst,range(length),cs):
    plt.subplot(2,2,j+1)
    dat = purpose_new[purpose_new["NAME_CONTRACT_STATUS"] == i]
    ax = sns.barplot(0,"NAME_CASH_LOAN_PURPOSE",data=dat.sort_values(by=0,ascending=False),color=k)
    plt.ylabel("")
    plt.xlabel("percentage")
    plt.title(i+" by purpose")
    plt.subplots_adjust(wspace = .7)
    ax.set_facecolor("k")
```


accepted_percentage by purpose
XAP
Purchase of electronic equipment
Everyday expenses
Education
Money for a third person
Medicine
Journey
Car repairs
Furniture
Other
Urgent needs
XNA
Wedding / gift / holiday
Gasification / water supply
Repairs
Hobby
Business development
Buying a used car
Buying a garage
Refusal to name the goal
Building a house or an annex
Buying a holiday home / land
Buying a home
Buying a new car
Payments on other loans
0 10 20 30 40 50 60 70 80
percentage
cancelled_percentage by purpose
XNA
Buying a garage
XAP
Buying a new car
Buying a home
Business development
Building a house or an annex
Payments on other loans
Buying a holiday home / land
Buying a used car
Repairs
Furniture
Wedding / gift / holiday
Car repairs
Other
Urgent needs
Gasification / water supply
Medicine
Education
Journey
Purchase of electronic equipment
Everyday expenses
Money for a third person
Refusal to name the goal
Hobby
0 5 10 15 20 25 30 35 40
percentage
refused_percentage by purpose
Payments on other loans
Refusal to name the goal
Buying a new car
Buying a home
Buying a holiday home / land
Building a house or an annex
Buying a used car
unused_percentage by purpose
XAP
Everyday expenses
Other
Buying a new car
Buying a used car
Purchase of electronic equipment
Journey

Point to infer from the graph

Purposes like XAP ,electronic eqipment ,everey day expences and education have maximum loan acceptance.

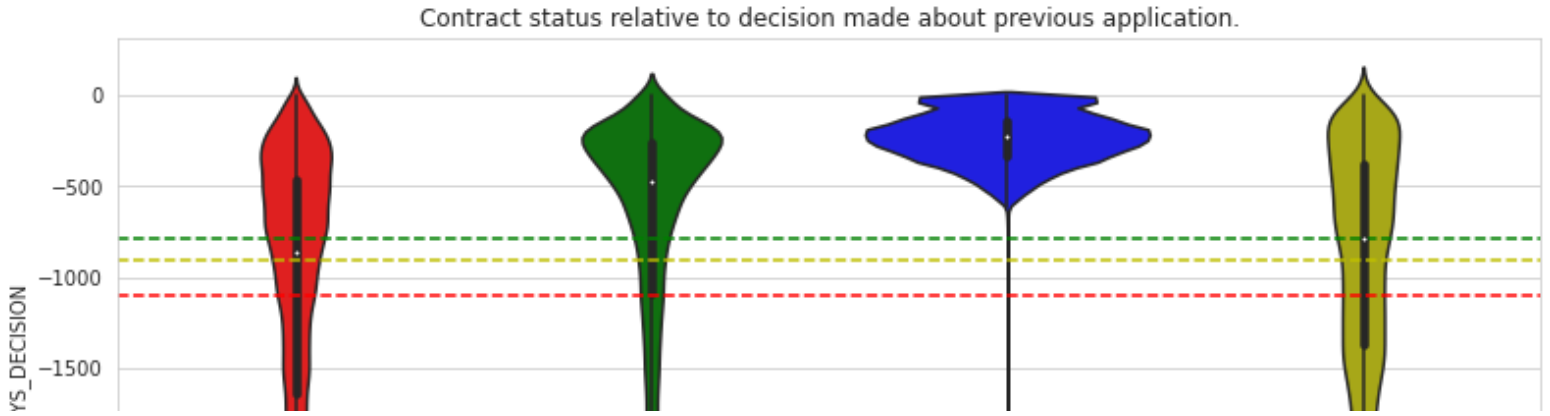
Loan puposes like payment of other loans ,refusal to name goal ,buying new home or car have most refusals.

40% of XNA purpose loans are cancalled.

Contract status relative to decision made about previous application.

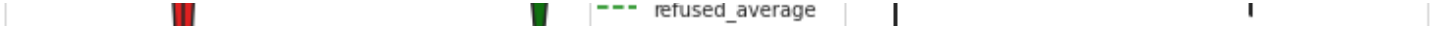
DAYS_DECISION - Relative to current application when was the decision about previous application made.

```
plt.figure(figsize=(13,6))
sns.violinplot(y= previous_application["DAYS_DECISION"],
               x = previous_application["NAME_CONTRACT_STATUS"],palette=["r","g","b","y"])
plt.axhline(previous_application[previous_application["NAME_CONTRACT_STATUS"] == "Approved"]["DAYS_DECISION"].mean(),
            color="r",linestyle="dashed",label="accepted_average")
plt.axhline(previous_application[previous_application["NAME_CONTRACT_STATUS"] == "Refused"]["DAYS_DECISION"].mean(),
            color="g",linestyle="dashed",label="refused_average")
plt.axhline(previous_application[previous_application["NAME_CONTRACT_STATUS"] == "Cancelled"]["DAYS_DECISION"].mean(),color="b",
            linestyle="dashed",label="cancelled_average")
plt.axhline(previous_application[previous_application["NAME_CONTRACT_STATUS"] == "Unused offer"]["DAYS_DECISION"].mean(),color="y",
            linestyle="dashed",label="un used_average")
plt.legend(loc="best")

plt.title("Contract status relative to decision made about previous application.")
plt.show()
```



Point to infer from the graph

On average approved contract types have higher number of decision days compared to cancelled and refused contracts.



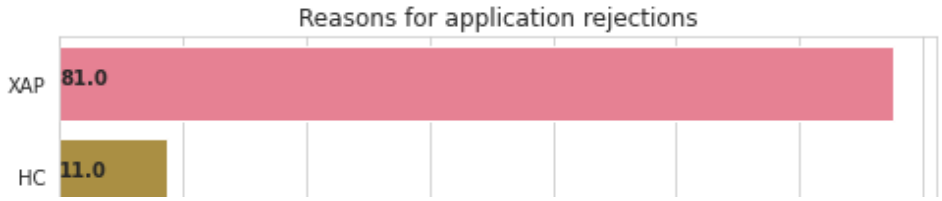
Client payment methods & reasons for application rejections

NAME_PAYMENT_TYPE - Payment method that client chose to pay for the previous application.

CODE_REJECT_REASON - Why was the previous application rejected.

```
plt.figure(figsize=(8,12))
plt.subplot(211)
rej = previous_application["CODE_REJECT_REASON"].value_counts().reset_index()
ax = sns.barplot("CODE_REJECT_REASON","index",data=rej[:6],palette="husl")
for i,j in enumerate(np.around((rej["CODE_REJECT_REASON"][:6].values*100/(rej["CODE_REJECT_REASON"][:6].sum())))):
    ax.text(.7,i,j,weight="bold")
plt.xlabel("Top as percentage & Bottom as Count")
plt.ylabel("CODE_REJECT_REASON")
plt.title("Reasons for application rejections")

plt.subplot(212)
pay = previous_application["NAME_PAYMENT_TYPE"].value_counts().reset_index()
ax1 = sns.barplot("NAME_PAYMENT_TYPE","index",data=pay,palette="husl")
for i,j in enumerate(np.around((pay["NAME_PAYMENT_TYPE"].values*100/(pay["NAME_PAYMENT_TYPE"].sum())))):
```

```
    ax1.text(.7,i,j,weight="bold")
plt.xlabel("pTop as percentage & Bottom as Count")
plt.ylabel("NAME_PAYMENT_TYPE")
plt.title("Clients payment methods")
plt.subplots_adjust(hspace = .3)
```



Point to infer from the graph

Around 81% of rejected applications the reason is XAP.

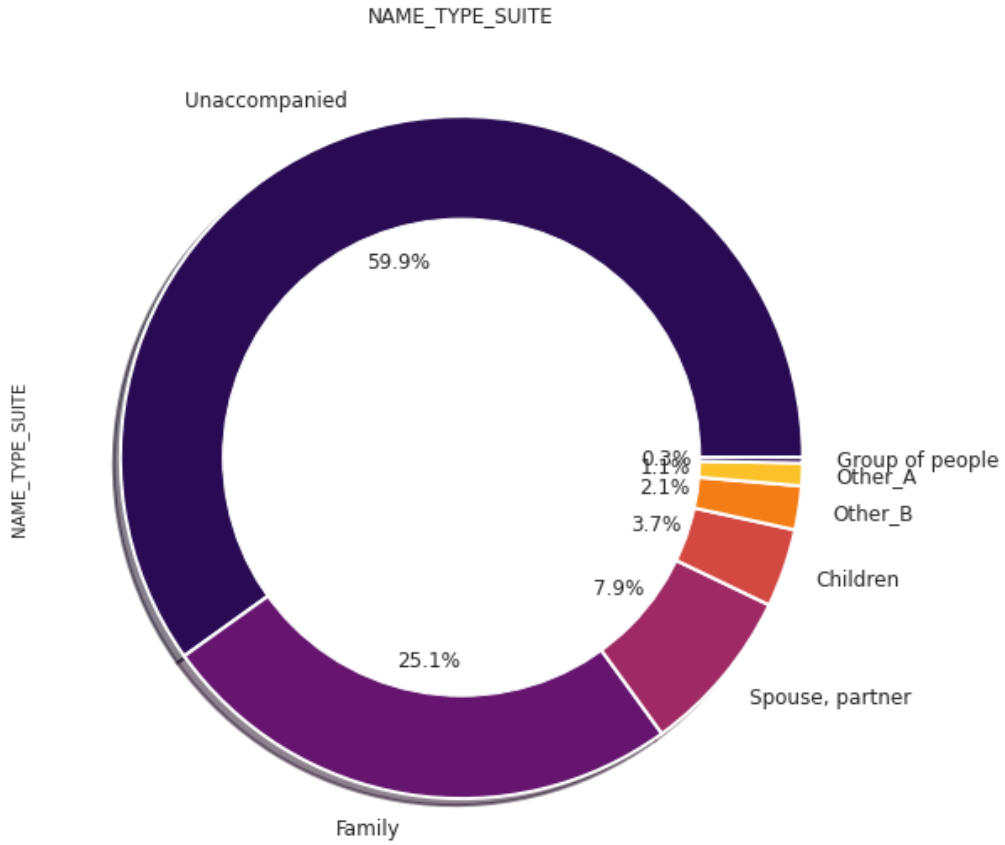
62% of chose to pay through cash by bank for previous applications.

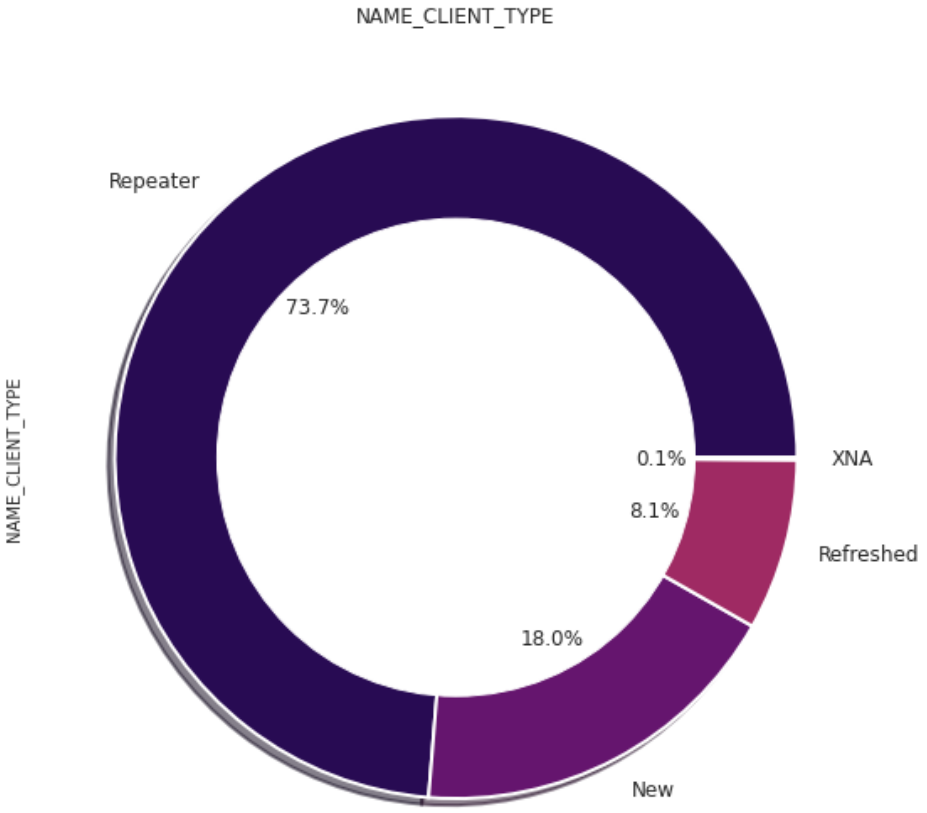
Distribution in Client suite type & client type.

NAME_TYPE_SUITE - Who accompanied client when applying for the previous application.

NAME_CLIENT_TYPE - Was the client old or new client when applying for the previous application.

```
plt.figure(figsize=(20,20))
plt.subplot(121)
previous_application["NAME_TYPE_SUITE"].value_counts().plot.pie(autopct = "%1.1f%%",fontsize=12,
                                                               colors = sns.color_palette("inferno"),
                                                    wedgeprops={"linewidth":2,"edgecolor":"white"},shadow =True)
circ = plt.Circle((0,0),.7,color="white")
plt.gca().add_artist(circ)
plt.title("NAME_TYPE_SUITE")

plt.subplot(122)
previous_application["NAME_CLIENT_TYPE"].value_counts().plot.pie(autopct = "%1.1f%%",fontsize=12,
                                                               colors = sns.color_palette("inferno"),
                                                    wedgeprops={"linewidth":2,"edgecolor":"white"},shadow =True)
circ = plt.Circle((0,0),.7,color="white")
plt.gca().add_artist(circ)
plt.title("NAME_CLIENT_TYPE")
plt.show()
```





Point to infer from the graph

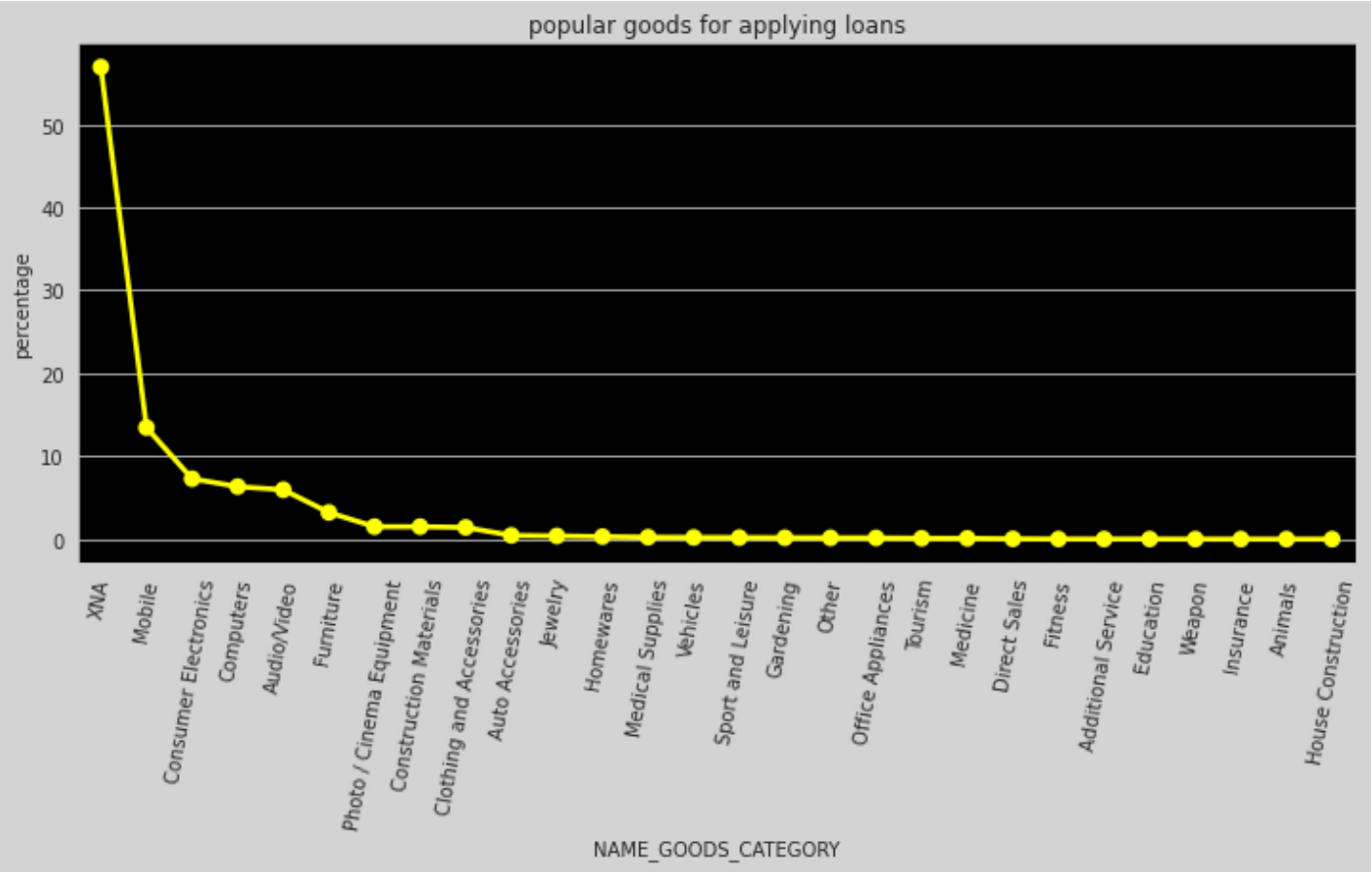
About 60% clients are un-accompained when applying for loans.

73% clients are old clients

Popular goods for applying loans

NAME_GOODS_CATEGORY - What kind of goods did the client apply for in the previous application.

```
goods = previous_application["NAME_GOODS_CATEGORY"].value_counts().reset_index()
goods["percentage"] = round(goods["NAME_GOODS_CATEGORY"]*100/goods["NAME_GOODS_CATEGORY"].sum(),2)
fig = plt.figure(figsize=(12,5))
ax = sns.pointplot("index","percentage",data=goods,color="yellow")
plt.xticks(rotation = 80)
plt.xlabel("NAME_GOODS_CATEGORY")
plt.ylabel("percentage")
plt.title("popular goods for applying loans")
ax.set_facecolor("k")
fig.set_facecolor('lightgrey')
```



Point to infer from the graph

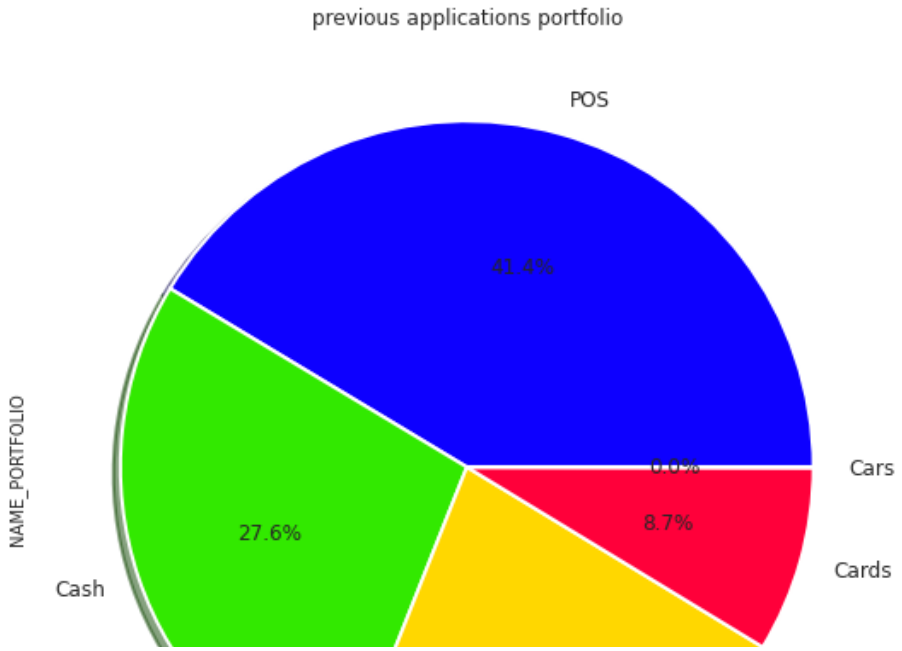
XNA ,Mobiles ,Computers and consumer electronics are popular goods for applying loans

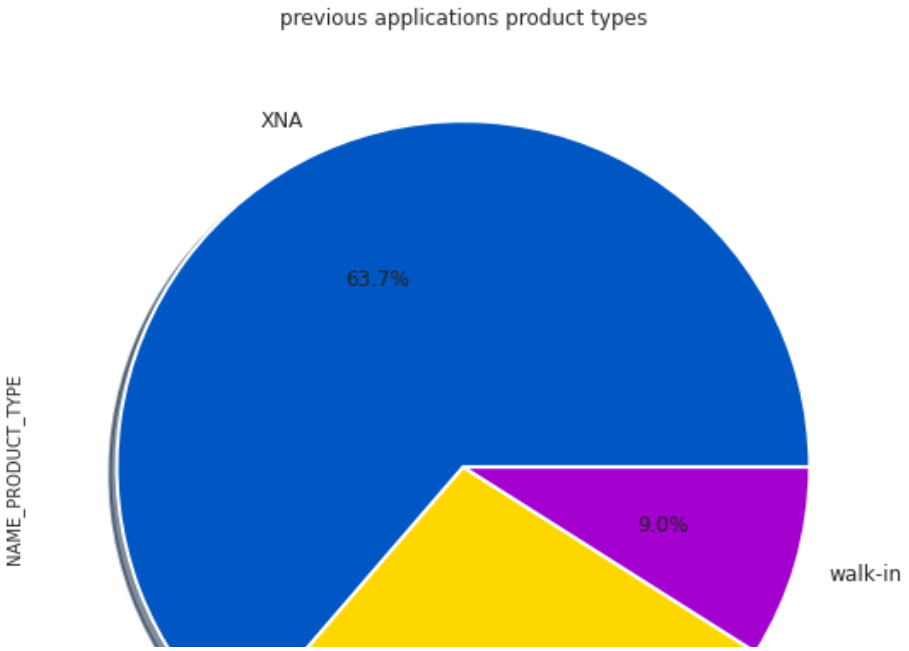
Previous applications portfolio and product types

NAME_PORTFOLIO - Was the previous application for CASH, POS, CAR, …

NAME_PRODUCT_TYPE - Was the previous application x-sell o walk-in.

```
plt.figure(figsize=(20,20))
plt.subplot(121)
previous_application["NAME_PORTFOLIO"].value_counts().plot.pie(autopct = "%1.1f%%",fontsize=12,
                                                               colors = sns.color_palette("prism",5),
                                                    wedgeprops={"linewidth":2,"edgecolor":"white"},
                                                                shadow =True)
plt.title("previous applications portfolio")
plt.subplot(122)
previous_application["NAME_PRODUCT_TYPE"].value_counts().plot.pie(autopct = "%1.1f%%",fontsize=12,
                                                               colors = sns.color_palette("prism",3),
                                                    wedgeprops={"linewidth":2,"edgecolor":"white"},
                                                                    shadow =True)
plt.title("previous applications product types")
plt.show()
```



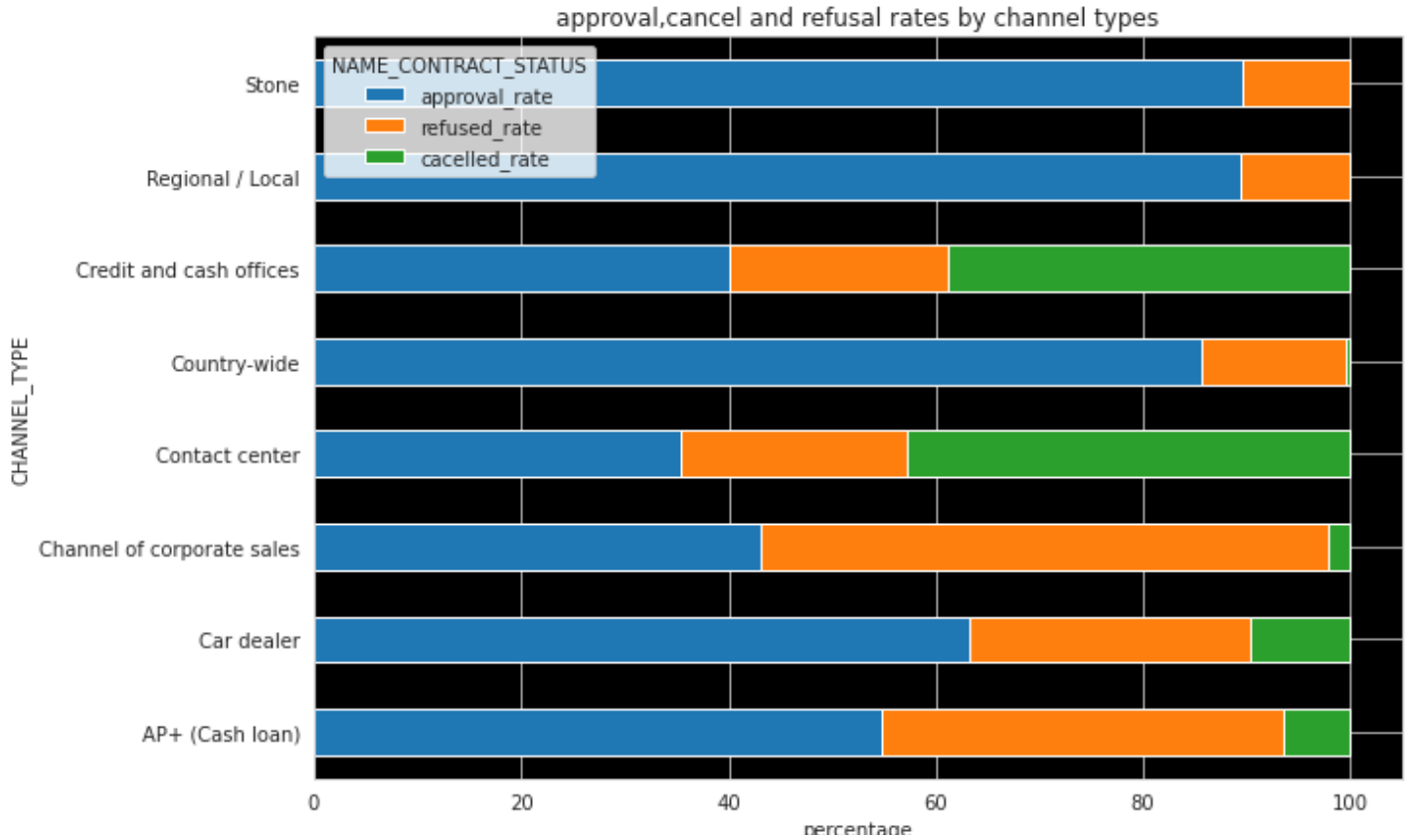


Approval,canceled and refusal rates by channel types.

CHANNEL_TYPE - Through which channel we acquired the client on the previous application.

NAME_CONTRACT_STATUS- Contract status (approved, cancelled, ...) of previous application.

```
app = pd.crosstab(previous_application["CHANNEL_TYPE"],previous_application["NAME_CONTRACT_STATUS"])
app1 = app
app1["approval_rate"] = app1["Approved"]*100/(app1["Approved"]+app1["Refused"]+app1["Canceled"])
app1["refused_rate"]  = app1["Refused"]*100/(app1["Approved"]+app1["Refused"]+app1["Canceled"])
app1["cacelled_rate"] = app1["Canceled"]*100/(app1["Approved"]+app1["Refused"]+app1["Canceled"])
app2 = app[["approval_rate","refused_rate","cacelled_rate"]]
ax = app2.plot(kind="barh",stacked=True,figsize=(10,7))
ax.set_facecolor("k")
ax.set_xlabel("percentage")
ax.set_title("approval,cancel and refusal rates by channel types")
plt.show()
```



Point to infer from the graph

Channel types like Stone ,regional and country-wide have maximum approval rates.

Channel of coorporate sales have maximum refusal rate.

Credit-cash centres and Contact centres have maximum cancellation rates.

Highest amount credited seller areas and industries.

SELLERPLACE_AREA - Selling area of seller place of the previous application.

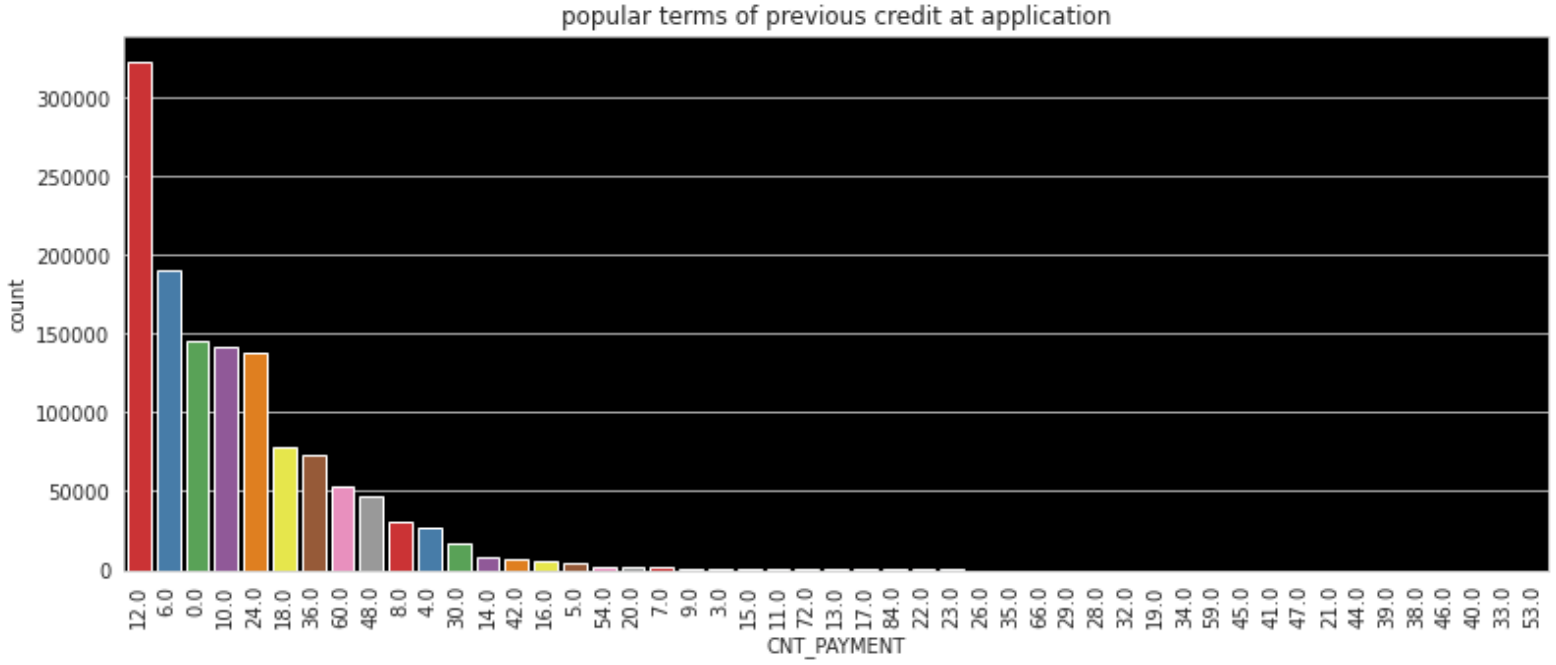
NAME_SELLER_INDUSTRY - The industry of the seller.

```
fig = plt.figure(figsize=(13,5))
plt.subplot(121)
are = previous_application.groupby("SELLERPLACE_AREA")["AMT_CREDIT"].sum().reset_index()
are = are.sort_values(by ="AMT_CREDIT",ascending = False)
ax = sns.barplot(y= "AMT_CREDIT",x ="SELLERPLACE_AREA",data=are[:15],color="r")
ax.set_facecolor("k")
ax.set_title("Highest amount credited seller place areas")

plt.subplot(122)
sell = previous_application.groupby("NAME_SELLER_INDUSTRY")["AMT_CREDIT"].sum().reset_index().sort_values(by = "AMT_CREDIT",ascending =
ax1=sns.barplot(y = "AMT_CREDIT",x = "NAME_SELLER_INDUSTRY",data=sell,color="b")
ax1.set_facecolor("k")
ax1.set_title("Highest amount credited seller industrys")
plt.xticks(rotation=90)
plt.subplots_adjust(wspace = .5)
fig.set_facecolor("lightgrey")
```

Popular terms of previous credit at application.

CNT_PAYMENT - Term of previous credit at application of the previous application.


```
plt.figure(figsize=(13,5))
ax = sns.countplot(previous_application["CNT_PAYMENT"],palette="Set1",order=previous_application["CNT_PAYMENT"].value_counts().index)
ax.set_facecolor("k")
plt.xticks(rotation = 90)
plt.title("popular terms of previous credit at application")
plt.show()
```

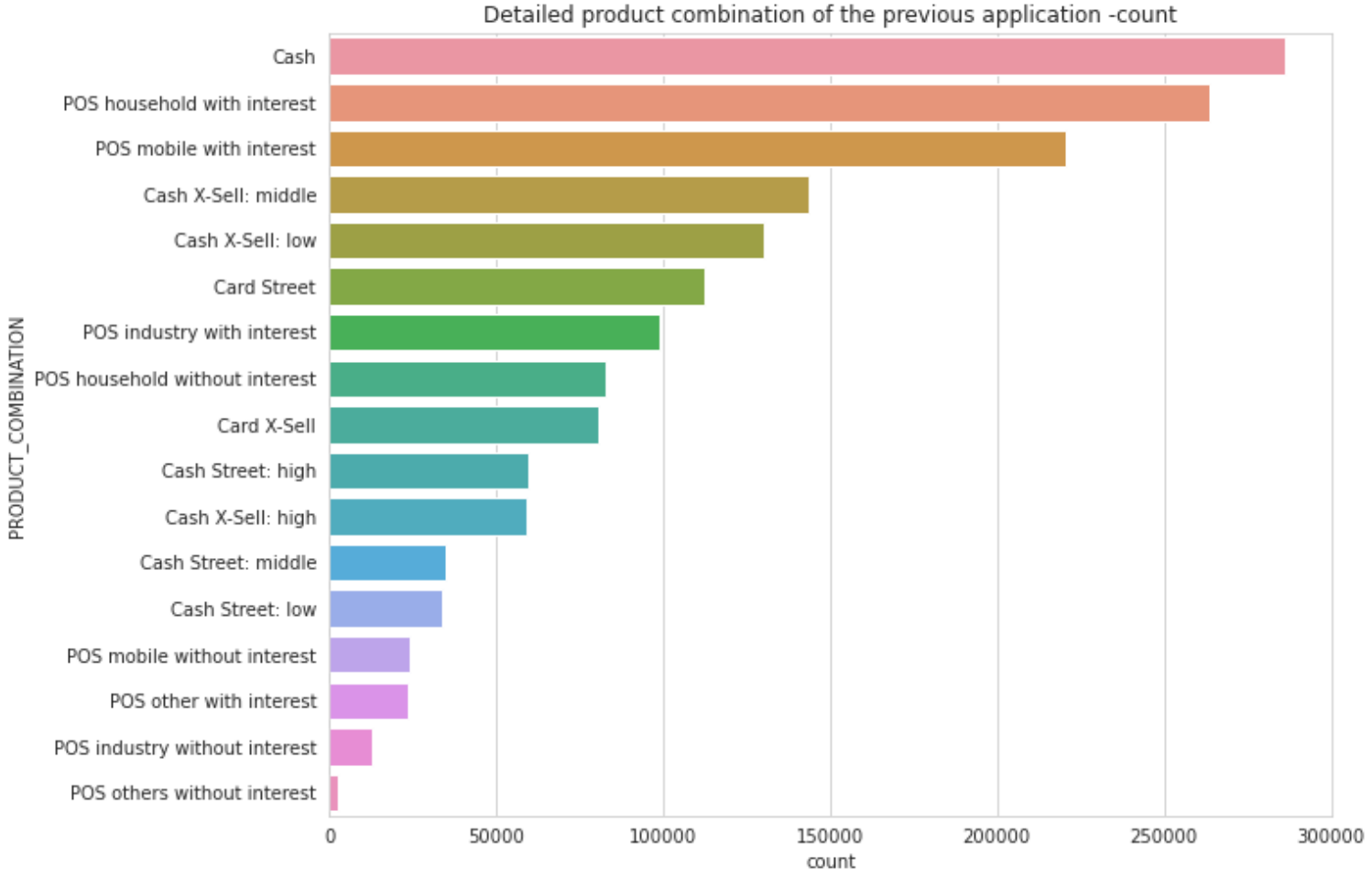


Point to infer from the graph

Popular term of previous credit are 6months ,10months ,1year ,2years & 3 years.

Detailed product combination of the previous application

```
plt.figure(figsize=(10,8))
sns.countplot(y = previous_application["PRODUCT_COMBINATION"],order=previous_application["PRODUCT_COMBINATION"].value_counts().index)
plt.title("Detailed product combination of the previous application -count")
plt.show()
```



Frequency distribution of intrest rates and client insurance requests

NAME_YIELD_GROUP - Grouped interest rate into small medium and high of the previous application.

NFLAG_INSURED_ON_APPROVAL - Did the client requested insurance during the previous application.

```
plt.figure(figsize=(12,6))
plt.subplot(121)
previous_application["NFLAG_INSURED_ON_APPROVAL"].value_counts().plot.pie(autopct = "%1.1f%%",fontsize=8,
                                                                          colors = sns.color_palette("prism",4),
                                                  wedgeprops={"linewidth":2,"edgecolor":"white"},shadow =True)
circ = plt.Circle((0,0),.7,color="white")
plt.gca().add_artist(circ)
plt.title("client requesting insurance")

plt.subplot(122)
previous_application["NAME_YIELD_GROUP"].value_counts().plot.pie(autopct = "%1.1f%%",fontsize=8,
                                                                          colors = sns.color_palette("prism",4),
                                                  wedgeprops={"linewidth":2,"edgecolor":"white"},shadow =True)
circ = plt.Circle((0,0),.7,color="white")
plt.gca().add_artist(circ)
plt.title("interest rates")
plt.show()
```

Days variables - Relative to application date of current application

DAYS_FIRST_DRAWING - Relative to application date of current application when was the first disbursement of the previous application.

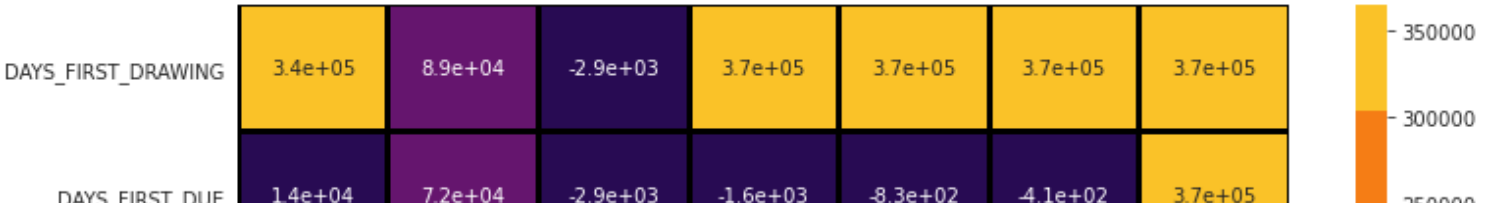
DAYS_FIRST_DUE - Relative to application date of current application when was the first due supposed to be of the previous application.

DAYS_LAST_DUE_1ST_VERSION - Relative to application date of current application when was the first due of the previous application.

DAYS_LAST_DUE -Relative to application date of current application when was the last due date of the previous application.

DAYS_TERMINATION - Relative to application date of current application when was the expected termination of the previous application.

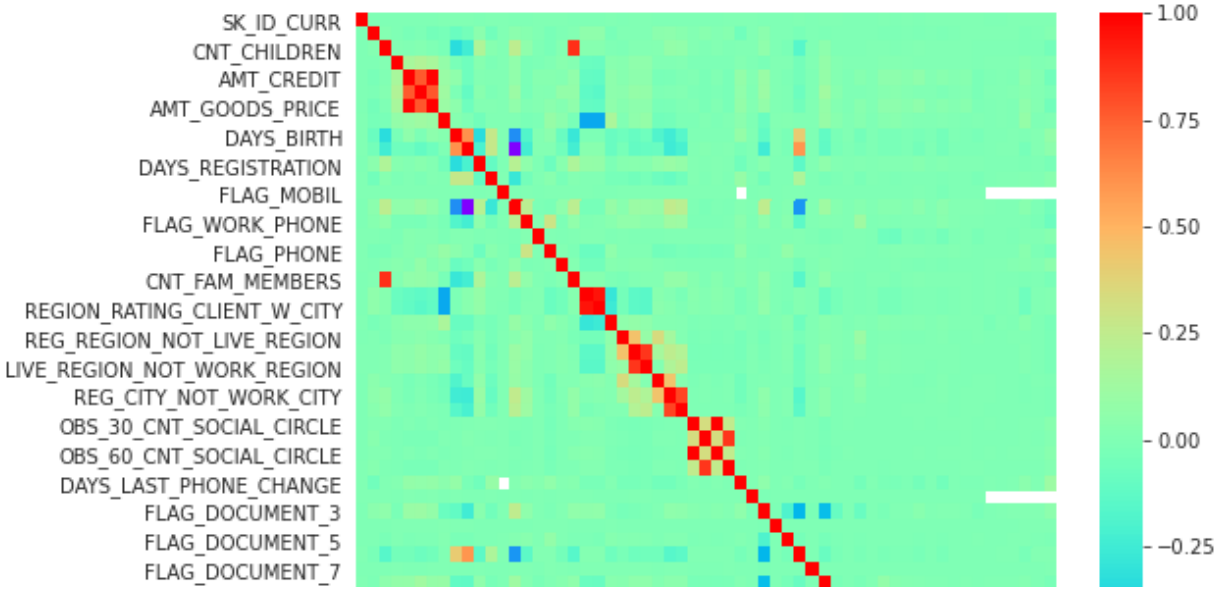
```
cols = ['DAYS_FIRST_DRAWING', 'DAYS_FIRST_DUE', 'DAYS_LAST_DUE_1ST_VERSION','DAYS_LAST_DUE', 'DAYS_TERMINATION']
plt.figure(figsize=(12,6))
sns.heatmap(previous_application[cols].describe()[1:].transpose(),
            annot=True,linewidth=2,linecolor="k",cmap=sns.color_palette("inferno"))
plt.show()
```



Corelation between variables

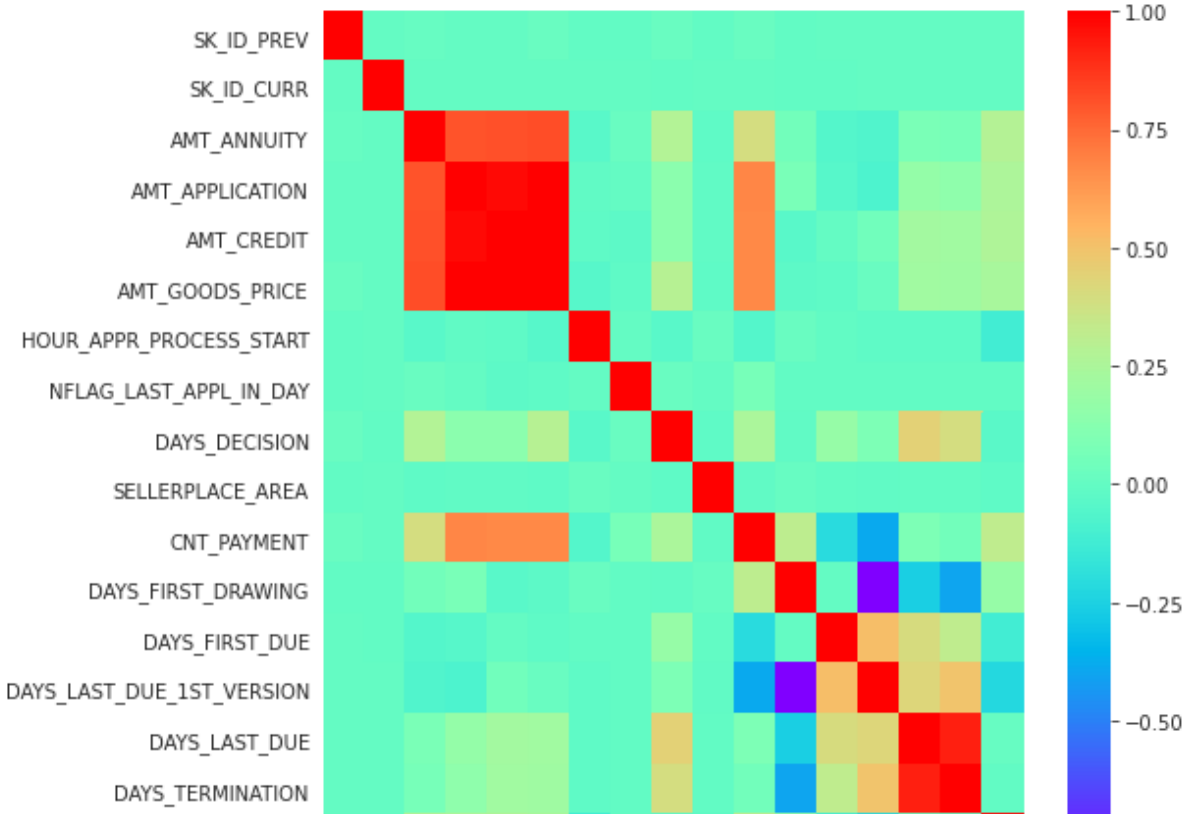
Application Data

```
corrmat = application_data.corr()

f, ax = plt.subplots(figsize =(8, 8))
sns.heatmap(corrmat, ax = ax, cmap ="rainbow")
plt.show()
```



Previous Application


```
corrmat = previous_application.corr()

f, ax = plt.subplots(figsize =(8, 8))
sns.heatmap(corrmat, ax = ax, cmap ="rainbow")
plt.show()
```




```
corrmat = previous_application.corr()
corrdf = corrmat.where(np.triu(np.ones(corrmat.shape), k=1).astype(np.bool))
corrdf = corrdf.unstack().reset_index()
corrdf.columns = ['Var1', 'Var2', 'Correlation']
corrdf.dropna(subset = ['Correlation'], inplace = True)
corrdf['Correlation'] = round(corrdf['Correlation'], 2)
corrdf['Correlation'] = abs(corrdf['Correlation'])
corrdf.sort_values(by = 'Correlation', ascending = False).head(10)
```

	Var1	Var2	Correlation
88	AMT_GOODS_PRICE	AMT_APPLICATION	1.00
89	AMT_GOODS_PRICE	AMT_CREDIT	0.99
71	AMT_CREDIT	AMT_APPLICATION	0.98
269	DAYS_TERMINATION	DAYS_LAST_DUE	0.93
87	AMT_GOODS_PRICE	AMT_ANNUITY	0.82
70	AMT_CREDIT	AMT_ANNUITY	0.82

Application Data

173	CNT_PAYMENT	AMT_APPLICATION	0.68

Top 10 Correlation Fields for Repayer

```
df_repayer = application_data[application_data['TARGET'] == 0]
df_defaulter = application_data[application_data['TARGET'] == 1]
```

```
corrmat = df_repayer.corr()
corrdf = corrmat.where(np.triu(np.ones(corrmat.shape), k=1).astype(np.bool))
corrdf = corrdf.unstack().reset_index()
corrdf.columns = ['Var1', 'Var2', 'Correlation']
corrdf.dropna(subset = ['Correlation'], inplace = True)
corrdf['Correlation'] = round(corrdf['Correlation'], 2)
corrdf['Correlation'] = abs(corrdf['Correlation'])
corrdf.sort_values(by = 'Correlation', ascending = False).head(10)
```

	Var1	Var2	Correlation
776	FLAG_EMP_PHONE	DAYS_EMPLOYED	1.00
1798	OBS_60_CNT_SOCIAL_CIRCLE	OBS_30_CNT_SOCIAL_CIRCLE	1.00
358	AMT_GOODS_PRICE	AMT_CREDIT	0.99
1199	REGION_RATING_CLIENT_W_CITY	REGION_RATING_CLIENT	0.95
1064	CNT_FAM_MEMBERS	CNT_CHILDREN	0.88
1858	DEF_60_CNT_SOCIAL_CIRCLE	DEF_30_CNT_SOCIAL_CIRCLE	0.86
1439	LIVE_REGION_NOT_WORK_REGION	REG_REGION_NOT_WORK_REGION	0.86

Top 10 Correlation Fields for Defaulter

299	AMT_ANNUITY	AMT_CREDIT	0.77

```
corrmat = df_defaulter.corr()
corrdf = corrmat.where(np.triu(np.ones(corrmat.shape), k=1).astype(np.bool))
corrdf = corrdf.unstack().reset_index()
corrdf.columns = ['Var1', 'Var2', 'Correlation']
corrdf.dropna(subset = ['Correlation'], inplace = True)
corrdf['Correlation'] = round(corrdf['Correlation'], 2)
corrdf['Correlation'] = abs(corrdf['Correlation'])
corrdf.sort_values(by = 'Correlation', ascending = False).head(10)
```

	Var1	Var2	Correlation
1798	OBS_60_CNT_SOCIAL_CIRCLE	OBS_30_CNT_SOCIAL_CIRCLE	1.00
776	FLAG_EMP_PHONE	DAYS_EMPLOYED	1.00
358	AMT_GOODS_PRICE	AMT_CREDIT	0.98

```
mergeddf =  pd.merge(application_data,previous_application,on='SK_ID_CURR')
mergeddf.head()
```

	SK_ID_CURR	TARGET	NAME_CONTRACT_TYPE_x	CODE_GENDER	FLAG_OWN_CAR	FLAG_OWN_REALTY	CNT_CHILDREN	AMT_INCOME_TOTAL	AMT_CREDI
0	100002	1	Cash loans	M	N	Y	0	202500.0	40659
1	100003	0	Cash loans	F	N	N	0	270000.0	12935(
2	100003	0	Cash loans	F	N	N	0	270000.0	12935(
3	100003	0	Cash loans	F	N	N	0	270000.0	12935(
4	100004	0	Revolving loans	M	Y	Y	0	67500.0	1350(

```
y = mergeddf.groupby('SK_ID_CURR').size()
dfA = mergeddf.groupby('SK_ID_CURR').agg({'TARGET': np.sum})
dfA['count'] = y
display(dfA.head(10))
```

SK_ID_CURR	TARGET	count
100002	1	1
100003	0	3
100004	0	1
100006	0	9
100007	0	6
100008	0	5

```
dfA.sort_values(by = 'count',ascending=False).head(10)
```

SK_ID_CURR	TARGET	count
265681	0	73
173680	0	72
242412	0	68
206783	0	67
389950	0	64
382179	0	64
198355	0	63
345161	0	62
446486	0	62
238250	0	61

```
df_repayer = dfA[dfA['TARGET'] == 0]
```

```
df_defaulter = dfA[dfA['TARGET'] == 1]
```

Repayers' Borrowing History

```
df_repayer.sort_values(by = 'count',ascending=False).head(10)
```

SK_ID_CURR	TARGET	count
265681	0	73
173680	0	72
242412	0	68
206783	0	67
382179	0	64
389950	0	64
198355	0	63
446486	0	62
345161	0	62
280586	0	61

Defaulters' Borrowing History

```
df_defaulter.sort_values(by = 'count',ascending=False).head(10)
```

SK_ID_CURR	TARGET	count
100002	1	1
333349	1	1
333587	1	1
333582	1	1
333534	1	1
333506	1	1
333419	1	1
333355	1	1

```
mergeddf.isnull().sum()
```

```
SK_ID_CURR                     0
TARGET                         0
NAME_CONTRACT_TYPE_x           0
CODE_GENDER                    0
FLAG_OWN_CAR                   0
FLAG_OWN_REALTY                0
CNT_CHILDREN                   0
AMT_INCOME_TOTAL               0
AMT_CREDIT_x                   0
AMT_ANNUITY_x                 93
AMT_GOODS_PRICE_x           1208
NAME_TYPE_SUITE_x           3526
NAME_INCOME_TYPE               0
NAME_EDUCATION_TYPE            0
NAME_FAMILY_STATUS             0
NAME_HOUSING_TYPE              0
REGION_POPULATION_RELATIVE     0
DAYS_BIRTH                     0
DAYS_EMPLOYED                  0
DAYS_REGISTRATION              0
DAYS_ID_PUBLISH                0
FLAG_MOBIL                     0
```

```
FLAG_EMP_PHONE                  0
FLAG_WORK_PHONE                 0
FLAG_CONT_MOBILE                0
FLAG_PHONE                      0
FLAG_EMAIL                      0
CNT_FAM_MEMBERS                 0
REGION_RATING_CLIENT            0
REGION_RATING_CLIENT_W_CITY     0
WEEKDAY_APPR_PROCESS_START_x    0
HOUR_APPR_PROCESS_START_x       0
REG_REGION_NOT_LIVE_REGION      0
REG_REGION_NOT_WORK_REGION      0
LIVE_REGION_NOT_WORK_REGION     0
REG_CITY_NOT_LIVE_CITY          0
REG_CITY_NOT_WORK_CITY          0
LIVE_CITY_NOT_WORK_CITY         0
ORGANIZATION_TYPE               0
OBS_30_CNT_SOCIAL_CIRCLE        3146
DEF_30_CNT_SOCIAL_CIRCLE        3146
OBS_60_CNT_SOCIAL_CIRCLE        3146
DEF_60_CNT_SOCIAL_CIRCLE        3146
DAYS_LAST_PHONE_CHANGE          0
FLAG_DOCUMENT_2                 0
FLAG_DOCUMENT_3                 0
FLAG_DOCUMENT_4                 0
FLAG_DOCUMENT_5                 0
FLAG_DOCUMENT_6                 0
FLAG_DOCUMENT_7                 0
FLAG_DOCUMENT_8                 0
FLAG_DOCUMENT_9                 0
FLAG_DOCUMENT_10                0
FLAG_DOCUMENT_11                0
FLAG_DOCUMENT_12                0
FLAG_DOCUMENT_13                0
FLAG_DOCUMENT_14                0
FLAG_DOCUMENT_15                0
FLAG_DOCUMENT_16                0

round(100*(mergeddf.isnull().sum()/len(mergeddf.index)), 2)
      _     _     _
FLAG_DOCUMENT_2                 0.00
FLAG_DOCUMENT_3                 0.00
FLAG_DOCUMENT_4                 0.00
```

FLAG_DOCUMENT_5	0.00
FLAG_DOCUMENT_6	0.00
FLAG_DOCUMENT_7	0.00
FLAG_DOCUMENT_8	0.00
FLAG_DOCUMENT_9	0.00
FLAG_DOCUMENT_10	0.00
FLAG_DOCUMENT_11	0.00
FLAG_DOCUMENT_12	0.00
FLAG_DOCUMENT_13	0.00
FLAG_DOCUMENT_14	0.00
FLAG_DOCUMENT_15	0.00
FLAG_DOCUMENT_16	0.00
FLAG_DOCUMENT_17	0.00
FLAG_DOCUMENT_18	0.00
FLAG_DOCUMENT_19	0.00
FLAG_DOCUMENT_20	0.00
FLAG_DOCUMENT_21	0.00
AMT_REQ_CREDIT_BUREAU_HOUR	11.57
AMT_REQ_CREDIT_BUREAU_DAY	11.57
AMT_REQ_CREDIT_BUREAU_WEEK	11.57
AMT_REQ_CREDIT_BUREAU_MON	11.57
AMT_REQ_CREDIT_BUREAU_QRT	11.57
AMT_REQ_CREDIT_BUREAU_YEAR	11.57
SK_ID_PREV	0.00
NAME_CONTRACT_TYPE_y	0.00
AMT_ANNUITY_y	21.73
AMT_APPLICATION	0.00
AMT_CREDIT_y	0.00
AMT_GOODS_PRICE_y	22.60
WEEKDAY_APPR_PROCESS_START_y	0.00
HOUR_APPR_PROCESS_START_y	0.00
FLAG_LAST_APPL_PER_CONTRACT	0.00
NFLAG_LAST_APPL_IN_DAY	0.00
NAME_CASH_LOAN_PURPOSE	0.00
NAME_CONTRACT_STATUS	0.00
DAYS_DECISION	0.00
NAME_PAYMENT_TYPE	0.00
CODE_REJECT_REASON	0.00
NAME_TYPE_SUITE_y	49.14
NAME_CLIENT_TYPE	0.00
NAME_GOODS_CATEGORY	0.00
NAME_PORTFOLIO	0.00
NAME_PRODUCT_TYPE	0.00

```
CHANNEL_TYPE                  0.00
SELLERPLACE_AREA              0.00
NAME_SELLER_INDUSTRY          0.00
CNT_PAYMENT                  21.73
NAME_YIELD_GROUP              0.00
PRODUCT_COMBINATION           0.02
DAYS_FIRST_DRAWING           39.69
DAYS_FIRST_DUE               39.69
DAYS_LAST_DUE_1ST_VERSION    39.69
DAYS_LAST_DUE                39.69
DAYS_TERMINATION             39.69
NFLAG_INSURED_ON_APPROVAL    39.69
dtype: float64
```

```
mergeddf.head()
```

	SK_ID_CURR	TARGET	NAME_CONTRACT_TYPE_x	CODE_GENDER	FLAG_OWN_CAR	FLAG_OWN_REALTY	CNT_CHILDREN	AMT_INCOME_TOTAL	AMT_CREDI
0	100002	1	Cash loans	M	N	Y	0	202500.0	40659
1	100003	0	Cash loans	F	N	N	0	270000.0	12935(
2	100003	0	Cash loans	F	N	N	0	270000.0	12935(
3	100003	0	Cash loans	F	N	N	0	270000.0	12935(
4	100004	0	Revolving loans	M	Y	Y	0	67500.0	1350(

```
#dropping SK_ID_CURR since it all unique values

mergeddf.drop(['SK_ID_CURR'], 1, inplace = True)


mergeddf.head()
```

```
mergeddf.head()
```

	TARGET	NAME_CONTRACT_TYPE_x	CODE_GENDER	FLAG_OWN_CAR	FLAG_OWN_REALTY	CNT_CHILDREN	AMT_INCOME_TOTAL	AMT_CREDIT_x	AMT_ANN
0	1	Cash loans	M	N	Y	0	202500.0	406597.5	2
1	0	Cash loans	F	N	N	0	270000.0	1293502.5	3
2	0	Cash loans	F	N	N	0	270000.0	1293502.5	3
3	0	Cash loans	F	N	N	0	270000.0	1293502.5	3
4	0	Revolving loans	M	Y	Y	0	67500.0	135000.0	

Now we will take care of null values in each column one by one.

```
round(100*(mergeddf.isnull().sum()/len(mergeddf.index)), 2)
```

```
TARGET                        0.00
NAME_CONTRACT_TYPE_x          0.00
CODE_GENDER                   0.00
FLAG_OWN_CAR                  0.00
FLAG_OWN_REALTY               0.00
CNT_CHILDREN                  0.00
AMT_INCOME_TOTAL              0.00
AMT_CREDIT_x                  0.00
AMT_ANNUITY_x                 0.01
AMT_GOODS_PRICE_x             0.09
NAME_TYPE_SUITE_x             0.25
NAME_INCOME_TYPE              0.00
NAME_EDUCATION_TYPE           0.00
NAME_FAMILY_STATUS            0.00
NAME_HOUSING_TYPE             0.00
REGION_POPULATION_RELATIVE    0.00
DAYS_BIRTH                    0.00
```

DAYS_EMPLOYED	0.00
DAYS_REGISTRATION	0.00
DAYS_ID_PUBLISH	0.00
FLAG_MOBIL	0.00
FLAG_EMP_PHONE	0.00
FLAG_WORK_PHONE	0.00
FLAG_CONT_MOBILE	0.00
FLAG_PHONE	0.00
FLAG_EMAIL	0.00
CNT_FAM_MEMBERS	0.00
REGION_RATING_CLIENT	0.00
REGION_RATING_CLIENT_W_CITY	0.00
WEEKDAY_APPR_PROCESS_START_x	0.00
HOUR_APPR_PROCESS_START_x	0.00
REG_REGION_NOT_LIVE_REGION	0.00
REG_REGION_NOT_WORK_REGION	0.00
LIVE_REGION_NOT_WORK_REGION	0.00
REG_CITY_NOT_LIVE_CITY	0.00
REG_CITY_NOT_WORK_CITY	0.00
LIVE_CITY_NOT_WORK_CITY	0.00
ORGANIZATION_TYPE	0.00
OBS_30_CNT_SOCIAL_CIRCLE	0.22
DEF_30_CNT_SOCIAL_CIRCLE	0.22
OBS_60_CNT_SOCIAL_CIRCLE	0.22
DEF_60_CNT_SOCIAL_CIRCLE	0.22
DAYS_LAST_PHONE_CHANGE	0.00
FLAG_DOCUMENT_2	0.00
FLAG_DOCUMENT_3	0.00
FLAG_DOCUMENT_4	0.00
FLAG_DOCUMENT_5	0.00
FLAG_DOCUMENT_6	0.00
FLAG_DOCUMENT_7	0.00
FLAG_DOCUMENT_8	0.00
FLAG_DOCUMENT_9	0.00
FLAG_DOCUMENT_10	0.00
FLAG_DOCUMENT_11	0.00
FLAG_DOCUMENT_12	0.00
FLAG_DOCUMENT_13	0.00
FLAG_DOCUMENT_14	0.00
FLAG_DOCUMENT_15	0.00
FLAG_DOCUMENT_16	0.00
FLAG_DOCUMENT_17	0.00

```
enq_cs =['AMT_REQ_CREDIT_BUREAU_DAY', 'AMT_REQ_CREDIT_BUREAU_HOUR',
       'AMT_REQ_CREDIT_BUREAU_MON', 'AMT_REQ_CREDIT_BUREAU_QRT',
       'AMT_REQ_CREDIT_BUREAU_WEEK', 'AMT_REQ_CREDIT_BUREAU_YEAR']
for i in enq_cs:
    mergeddf[i] = mergeddf[i].fillna(0)


amt_cs = ["AMT_ANNUITY_y","AMT_GOODS_PRICE_y"]
for i in amt_cs:
    mergeddf[i] = mergeddf[i].fillna(mergeddf[i].mean())



cols = ["DAYS_FIRST_DRAWING","DAYS_FIRST_DUE","DAYS_LAST_DUE_1ST_VERSION",
        "DAYS_LAST_DUE","DAYS_TERMINATION",'CNT_PAYMENT']
for i in cols :
    mergeddf[i]  = mergeddf[i].fillna(mergeddf[i].median())


cols = ["NAME_TYPE_SUITE_y","NFLAG_INSURED_ON_APPROVAL"]
for i in cols :
    mergeddf[i]  = mergeddf[i].fillna(mergeddf[i].mode()[0])


# Rest missing values are under 1.5% so we can drop these rows.
mergeddf.dropna(inplace = True)


round(100*(mergeddf.isnull().sum()/len(mergeddf.index)), 2)
```

```
TARGET                        0.0
NAME_CONTRACT_TYPE_x          0.0
CODE_GENDER                   0.0
FLAG_OWN_CAR                  0.0
FLAG_OWN_REALTY               0.0
CNT_CHILDREN                  0.0
AMT_INCOME_TOTAL              0.0
AMT_CREDIT_x                  0.0
AMT_ANNUITY_x                 0.0
AMT_GOODS_PRICE_x             0.0
NAME_TYPE_SUITE_x             0.0
```

```
NAME_INCOME_TYPE                0.0
NAME_EDUCATION_TYPE             0.0
NAME_FAMILY_STATUS              0.0
NAME_HOUSING_TYPE               0.0
REGION_POPULATION_RELATIVE      0.0
DAYS_BIRTH                      0.0
DAYS_EMPLOYED                   0.0
DAYS_REGISTRATION               0.0
DAYS_ID_PUBLISH                 0.0
FLAG_MOBIL                      0.0
FLAG_EMP_PHONE                  0.0
FLAG_WORK_PHONE                 0.0
FLAG_CONT_MOBILE                0.0
FLAG_PHONE                      0.0
FLAG_EMAIL                      0.0
CNT_FAM_MEMBERS                 0.0
REGION_RATING_CLIENT            0.0
REGION_RATING_CLIENT_W_CITY     0.0
WEEKDAY_APPR_PROCESS_START_x    0.0
HOUR_APPR_PROCESS_START_x       0.0
REG_REGION_NOT_LIVE_REGION      0.0
REG_REGION_NOT_WORK_REGION      0.0
LIVE_REGION_NOT_WORK_REGION     0.0
REG_CITY_NOT_LIVE_CITY          0.0
REG_CITY_NOT_WORK_CITY          0.0
LIVE_CITY_NOT_WORK_CITY         0.0
ORGANIZATION_TYPE               0.0
OBS_30_CNT_SOCIAL_CIRCLE        0.0
DEF_30_CNT_SOCIAL_CIRCLE        0.0
OBS_60_CNT_SOCIAL_CIRCLE        0.0
DEF_60_CNT_SOCIAL_CIRCLE        0.0
DAYS_LAST_PHONE_CHANGE          0.0
FLAG_DOCUMENT_2                 0.0
FLAG_DOCUMENT_3                 0.0
FLAG_DOCUMENT_4                 0.0
FLAG_DOCUMENT_5                 0.0
FLAG_DOCUMENT_6                 0.0
FLAG_DOCUMENT_7                 0.0
FLAG_DOCUMENT_8                 0.0
FLAG_DOCUMENT_9                 0.0
FLAG_DOCUMENT_10                0.0
FLAG_DOCUMENT_11                0.0
FLAG_DOCUMENT_12                0.0
```

```
FLAG_DOCUMENT_13                0.0
FLAG_DOCUMENT_14                0.0
FLAG_DOCUMENT_15                0.0
FLAG_DOCUMENT_16                0.0
FLAG_DOCUMENT_17                0.0
```

```
mergeddf.isnull().sum()
```

```
TARGET                          0
NAME_CONTRACT_TYPE_x            0
CODE_GENDER                     0
FLAG_OWN_CAR                    0
FLAG_OWN_REALTY                 0
CNT_CHILDREN                    0
AMT_INCOME_TOTAL                0
AMT_CREDIT_x                    0
AMT_ANNUITY_x                   0
AMT_GOODS_PRICE_x               0
NAME_TYPE_SUITE_x               0
NAME_INCOME_TYPE                0
NAME_EDUCATION_TYPE             0
NAME_FAMILY_STATUS              0
NAME_HOUSING_TYPE               0
REGION_POPULATION_RELATIVE      0
DAYS_BIRTH                      0
DAYS_EMPLOYED                   0
DAYS_REGISTRATION               0
DAYS_ID_PUBLISH                 0
FLAG_MOBIL                      0
FLAG_EMP_PHONE                  0
FLAG_WORK_PHONE                 0
FLAG_CONT_MOBILE                0
FLAG_PHONE                      0
FLAG_EMAIL                      0
CNT_FAM_MEMBERS                 0
REGION_RATING_CLIENT            0
REGION_RATING_CLIENT_W_CITY     0
WEEKDAY_APPR_PROCESS_START_x    0
HOUR_APPR_PROCESS_START_x       0
REG_REGION_NOT_LIVE_REGION      0
REG_REGION_NOT_WORK_REGION      0
LIVE_REGION_NOT_WORK_REGION     0
```

```
REG_CITY_NOT_LIVE_CITY          0
REG_CITY_NOT_WORK_CITY          0
LIVE_CITY_NOT_WORK_CITY         0
ORGANIZATION_TYPE               0
OBS_30_CNT_SOCIAL_CIRCLE        0
DEF_30_CNT_SOCIAL_CIRCLE        0
OBS_60_CNT_SOCIAL_CIRCLE        0
DEF_60_CNT_SOCIAL_CIRCLE        0
DAYS_LAST_PHONE_CHANGE          0
FLAG_DOCUMENT_2                 0
FLAG_DOCUMENT_3                 0
FLAG_DOCUMENT_4                 0
FLAG_DOCUMENT_5                 0
FLAG_DOCUMENT_6                 0
FLAG_DOCUMENT_7                 0
FLAG_DOCUMENT_8                 0
FLAG_DOCUMENT_9                 0
FLAG_DOCUMENT_10                0
FLAG_DOCUMENT_11                0
FLAG_DOCUMENT_12                0
FLAG_DOCUMENT_13                0
FLAG_DOCUMENT_14                0
FLAG_DOCUMENT_15                0
FLAG_DOCUMENT_16                0
FLAG_DOCUMENT_17                0
```

```
mergeddf.head()
```

	TARGET	NAME_CONTRACT_TYPE_x	CODE_GENDER	FLAG_OWN_CAR	FLAG_OWN_REALTY	CNT_CHILDREN	AMT_INCOME_TOTAL	AMT_CREDIT_x	AMT_ANN
0	1	Cash loans	M	N	Y	0	202500.0	406597.5	1

Data Preparation

Converting some binary variables (Y/N) to 1/0

```
# List of variables to map

varlist =  ['FLAG_OWN_CAR','FLAG_OWN_REALTY','FLAG_LAST_APPL_PER_CONTRACT']

# Defining the map function
def binary_map(x):
    return x.map({'Y': 1, "N": 0})

# Applying the function to the housing list
mergeddf[varlist] = mergeddf[varlist].apply(binary_map)
mergeddf.head()
```

```
#dropping SK_ID_PREV since non required technical field

mergeddf.drop(['SK_ID_PREV'], 1, inplace = True)
mergeddf.head()
```

	TARGET	NAME_CONTRACT_TYPE_x	CODE_GENDER	FLAG_OWN_CAR	FLAG_OWN_REALTY	CNT_CHILDREN	AMT_INCOME_TOTAL	AMT_CREDIT_x	AMT_ANN
0	1	Cash loans	M	0	1	0	202500.0	406597.5	2
1	0	Cash loans	F	0	0	0	270000.0	1293502.5	3
2	0	Cash loans	F	0	0	0	270000.0	1293502.5	3
3	0	Cash loans	F	0	0	0	270000.0	1293502.5	3
4	0	Revolving loans	M	1	1	0	67500.0	135000.0	

```
mergeddf[['FLAG_OWN_CAR','FLAG_OWN_REALTY','FLAG_MOBIL',
 'FLAG_EMP_PHONE','FLAG_WORK_PHONE','FLAG_CONT_MOBILE',
 'FLAG_PHONE','FLAG_EMAIL','REGION_RATING_CLIENT',
 'REGION_RATING_CLIENT_W_CITY','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','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',
 'NFLAG_INSURED_ON_APPROVAL']]= mergeddf[['FLAG_OWN_CAR','FLAG_OWN_REALTY','FLAG_MOBIL',
                                         'FLAG_EMP_PHONE','FLAG_WORK_PHONE','FLAG_CONT_MOBILE',
                                         'FLAG_PHONE','FLAG_EMAIL','REGION_RATING_CLIENT',
                                         'REGION_RATING_CLIENT_W_CITY','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','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',
                                        'NFLAG_INSURED_ON_APPROVAL']].astype('category')


obj_dtypes = [i for i in mergeddf.select_dtypes(include=np.object).columns if i not in ["type"] ]
num_dtypes = [i for i in mergeddf.select_dtypes(include = np.number).columns if i not in [ 'TARGET']]


num_dtypes

     ['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',
      'CNT_FAM_MEMBERS',
      'HOUR_APPR_PROCESS_START_x',
      'LIVE_CITY_NOT_WORK_CITY',
      'OBS_30_CNT_SOCIAL_CIRCLE',
      'DEF_30_CNT_SOCIAL_CIRCLE',
      'OBS_60_CNT_SOCIAL_CIRCLE',
      'DEF_60_CNT_SOCIAL_CIRCLE',
      'DAYS_LAST_PHONE_CHANGE',
      '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',
      'AMT_ANNUITY_y',
      'AMT_APPLICATION',
```

```
       'AMT_CREDIT_y',
       'AMT_GOODS_PRICE_y',
       'HOUR_APPR_PROCESS_START_y',
       'FLAG_LAST_APPL_PER_CONTRACT',
       'NFLAG_LAST_APPL_IN_DAY',
       'DAYS_DECISION',
       'SELLERPLACE_AREA',
       'CNT_PAYMENT',
       'DAYS_FIRST_DRAWING',
       'DAYS_FIRST_DUE',
       'DAYS_LAST_DUE_1ST_VERSION',
       'DAYS_LAST_DUE',
       'DAYS_TERMINATION']


obj_dtypes

     ['NAME_CONTRACT_TYPE_x',
      'CODE_GENDER',
      'NAME_TYPE_SUITE_x',
      'NAME_INCOME_TYPE',
      'NAME_EDUCATION_TYPE',
      'NAME_FAMILY_STATUS',
      'NAME_HOUSING_TYPE',
      'WEEKDAY_APPR_PROCESS_START_x',
      'ORGANIZATION_TYPE',
      'NAME_CONTRACT_TYPE_y',
      'WEEKDAY_APPR_PROCESS_START_y',
      'NAME_CASH_LOAN_PURPOSE',
      'NAME_CONTRACT_STATUS',
      'NAME_PAYMENT_TYPE',
      'CODE_REJECT_REASON',
      'NAME_TYPE_SUITE_y',
      'NAME_CLIENT_TYPE',
      'NAME_GOODS_CATEGORY',
      'NAME_PORTFOLIO',
      'NAME_PRODUCT_TYPE',
      'CHANNEL_TYPE',
      'NAME_SELLER_INDUSTRY',
      'NAME_YIELD_GROUP',
      'PRODUCT_COMBINATION']
```

```
# Creating a dummy variable for some of the categorical variables and dropping the first one.
dummy1 = pd.get_dummies(mergeddf[['FLAG_OWN_CAR','FLAG_OWN_REALTY','FLAG_MOBIL',
                                  'FLAG_EMP_PHONE','FLAG_WORK_PHONE','FLAG_CONT_MOBILE',
                                  'FLAG_PHONE','FLAG_EMAIL','REGION_RATING_CLIENT',
                                  'REGION_RATING_CLIENT_W_CITY','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','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',
                                  'NFLAG_INSURED_ON_APPROVAL','NAME_CONTRACT_TYPE_x','CODE_GENDER',
                                  'NAME_TYPE_SUITE_x','NAME_INCOME_TYPE','NAME_EDUCATION_TYPE','NAME_FAMILY_STATUS',
                                  'NAME_HOUSING_TYPE','WEEKDAY_APPR_PROCESS_START_x','ORGANIZATION_TYPE','NAME_CONTRACT_TYPE_y',
                                  'WEEKDAY_APPR_PROCESS_START_y','NAME_CASH_LOAN_PURPOSE','NAME_CONTRACT_STATUS','NAME_PAYMENT_TYPE',
                                  'CODE_REJECT_REASON','NAME_TYPE_SUITE_y','NAME_CLIENT_TYPE','NAME_GOODS_CATEGORY',
                                  'NAME_PORTFOLIO','NAME_PRODUCT_TYPE','CHANNEL_TYPE','NAME_SELLER_INDUSTRY',
                                  'NAME_YIELD_GROUP','PRODUCT_COMBINATION']], drop_first=True)
dummy1.head()
```

	FLAG_OWN_CAR_1	FLAG_OWN_REALTY_1	FLAG_EMP_PHONE_1	FLAG_WORK_PHONE_1	FLAG_CONT_MOBILE_1	FLAG_PHONE_1	FLAG_EMAIL_1	REGION_
0	0	1	1	0	1	1	0	
1	0	0	1	0	1	1	0	
2	0	0	1	0	1	1	0	
3	0	0	1	0	1	1	0	
4	1	1	1	1	1	1	0	

```
# Adding the results to the master dataframe
mergeddf = pd.concat([mergeddf, dummy1], axis=1)
mergeddf.head()
```

	TARGET	NAME_CONTRACT_TYPE_x	CODE_GENDER	FLAG_OWN_CAR	FLAG_OWN_REALTY	CNT_CHILDREN	AMT_INCOME_TOTAL	AMT_CREDIT_x	AMT_ANN
0	1	Cash loans	M	0	1	0	202500.0	406597.5	2
1	0	Cash loans	F	0	0	0	270000.0	1293502.5	3
2	0	Cash loans	F	0	0	0	270000.0	1293502.5	3
3	0	Cash loans	F	0	0	0	270000.0	1293502.5	3
4	0	Revolving loans	M	1	1	0	67500.0	135000.0	

```
mergeddf = mergeddf.drop(['FLAG_OWN_CAR','FLAG_OWN_REALTY','FLAG_MOBIL',
                          'FLAG_EMP_PHONE','FLAG_WORK_PHONE','FLAG_CONT_MOBILE',
                          'FLAG_PHONE','FLAG_EMAIL','REGION_RATING_CLIENT',
                          'REGION_RATING_CLIENT_W_CITY','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','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',
                          'NFLAG_INSURED_ON_APPROVAL','NAME_CONTRACT_TYPE_x','CODE_GENDER',
                          'NAME_TYPE_SUITE_x','NAME_INCOME_TYPE','NAME_EDUCATION_TYPE','NAME_FAMILY_STATUS',
                          'NAME_HOUSING_TYPE','WEEKDAY_APPR_PROCESS_START_x','ORGANIZATION_TYPE','NAME_CONTRACT_TYPE_y',
                          'WEEKDAY_APPR_PROCESS_START_y','NAME_CASH_LOAN_PURPOSE','NAME_CONTRACT_STATUS','NAME_PAYMENT_TYPE',
                          'CODE_REJECT_REASON','NAME_TYPE_SUITE_y','NAME_CLIENT_TYPE','NAME_GOODS_CATEGORY',
                          'NAME_PORTFOLIO','NAME_PRODUCT_TYPE','CHANNEL_TYPE','NAME_SELLER_INDUSTRY',
                          'NAME_YIELD_GROUP','PRODUCT_COMBINATION'], axis = 1)
mergeddf.head()
```

	TARGET	CNT_CHILDREN	AMT_INCOME_TOTAL	AMT_CREDIT_x	AMT_ANNUITY_x	AMT_GOODS_PRICE_x	REGION_POPULATION_RELATIVE	DAYS_BIRTH
0	1	0	202500.0	406597.5	24700.5	351000.0	0.018801	9461
1	0	0	270000.0	1293502.5	35698.5	1129500.0	0.003541	16765
2	0	0	270000.0	1293502.5	35698.5	1129500.0	0.003541	16765
3	0	0	270000.0	1293502.5	35698.5	1129500.0	0.003541	16765
4	0	0	67500.0	135000.0	6750.0	135000.0	0.010032	19046

```
mergeddf.shape
```

```
(1406625, 292)
```

```
mergeddfs=mergeddf.sample(n = 7000)
```

```
from sklearn.model_selection import train_test_split

# Putting feature variable to X
X = mergeddfs.drop(['TARGET'], axis=1)
```

```
X.head()
```

	CNT_CHILDREN	AMT_INCOME_TOTAL	AMT_CREDIT_x	AMT_ANNUITY_x	AMT_GOODS_PRICE_x	REGION_POPULATION_RELATIVE	DAYS_BIRTH	DA

```
X.shape
```

```
(7000, 291)
```

```
# Putting response variable to y
y = mergeddfs['TARGET']
```

```
y.head()
```

```
475640    0
439791    0
652144    1
717300    0
246545    0
Name: TARGET, dtype: int64
```

```
# Splitting the data into train and test
X_train, X_test, y_train, y_test = train_test_split(X, y, train_size=0.7, random_state=70)
```

```
X_train.head()
```

	CNT_CHILDREN	AMT_INCOME_TOTAL	AMT_CREDIT_x	AMT_ANNUITY_x	AMT_GOODS_PRICE_x	REGION_POPULATION_RELATIVE	DAYS_BIRTH	D

```
X_train.shape
```

```
(4900, 291)
```

```
X_test.head()
```

	CNT_CHILDREN	AMT_INCOME_TOTAL	AMT_CREDIT_x	AMT_ANNUITY_x	AMT_GOODS_PRICE_x	REGION_POPULATION_RELATIVE	DAYS_BIRTH	D
856561	0	126000.0	450000.0	23107.5	450000.0	0.046220	16228	
569831	0	369000.0	497520.0	36054.0	450000.0	0.018850	16981	
1122558	1	180000.0	781695.0	25344.0	652500.0	0.006671	17023	
1232806	0	139500.0	284400.0	13963.5	225000.0	0.007114	14795	
231290	1	76500.0	1185120.0	39298.5	900000.0	0.018029	11055	

```
X_test.shape
```

```
(2100, 291)
```

```
y_train.head()
```

```
1074420    0
723991     0
209411     0
1140628    0
1262058    0
Name: TARGET, dtype: int64
```

```
y_train.shape
```

```
(4900,)
```

```
y_test.head()
```

```
856561     0
569831     0
1122558    0
1232806    0
231290     0
Name: TARGET, dtype: int64
```

```
y_test.shape
```

```
(2100,)
```

Feature Scaling

```
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()

X_train[['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','CNT_FAM_MEMBERS',
         'HOUR_APPR_PROCESS_START_x','LIVE_CITY_NOT_WORK_CITY','OBS_30_CNT_SOCIAL_CIRCLE',
         'DEF_30_CNT_SOCIAL_CIRCLE','OBS_60_CNT_SOCIAL_CIRCLE','DEF_60_CNT_SOCIAL_CIRCLE',
         'DAYS_LAST_PHONE_CHANGE','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','AMT_ANNUITY_y','AMT_APPLICATION','AMT_CREDIT_y',
         'AMT_GOODS_PRICE_y','HOUR_APPR_PROCESS_START_y','FLAG_LAST_APPL_PER_CONTRACT',
         'NFLAG_LAST_APPL_IN_DAY','DAYS_DECISION','SELLERPLACE_AREA','CNT_PAYMENT',
         'DAYS_FIRST_DRAWING','DAYS_FIRST_DUE','DAYS_LAST_DUE_1ST_VERSION','DAYS_LAST_DUE',
         'DAYS_TERMINATION']] = scaler.fit_transform(X_train[['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',
                                                    'CNT_FAM_MEMBERS','HOUR_APPR_PROCESS_START_x','LIVE_CITY_NOT_WORK_CITY',
                                                    'OBS_30_CNT_SOCIAL_CIRCLE','DEF_30_CNT_SOCIAL_CIRCLE','OBS_60_CNT_SOCIAL_
                                                    'DEF_60_CNT_SOCIAL_CIRCLE','DAYS_LAST_PHONE_CHANGE','AMT_REQ_CREDIT_BUREA
                                                    'AMT_REQ_CREDIT_BUREAU_DAY','AMT_REQ_CREDIT_BUREAU_WEEK','AMT_REQ_CREDIT_
                                                    'AMT_REQ_CREDIT_BUREAU_QRT','AMT_REQ_CREDIT_BUREAU_YEAR','AMT_ANNUITY_y',
                                                    'AMT_CREDIT_y','AMT_GOODS_PRICE_y','HOUR_APPR_PROCESS_START_y','FLAG_LAST
                                                    'NFLAG_LAST_APPL_IN_DAY','DAYS_DECISION','SELLERPLACE_AREA','CNT_PAYMENT'
                                                    'DAYS_FIRST_DUE','DAYS_LAST_DUE_1ST_VERSION','DAYS_LAST_DUE','DAYS_TERMIN
```

```
X_train.head()
```

	CNT_CHILDREN	AMT_INCOME_TOTAL	AMT_CREDIT_x	AMT_ANNUITY_x	AMT_GOODS_PRICE_x	REGION_POPULATION_RELATIVE	DAYS_BIRTH	D
1074420	-0.562809	-0.013994	-0.443019	-0.358424	-0.477542	0.580838	0.345483	
723991	-0.562809	1.951656	2.616287	3.994344	3.037350	-0.303311	1.481749	
209411	0.805130	-0.856415	0.551554	-0.077339	0.397962	-1.242339	-0.601869	
1140628	-0.562809	-0.856415	-0.509675	-0.918325	-0.387416	0.580838	1.203008	
1262058	-0.562809	0.313615	-0.768213	-0.887201	-0.670668	-0.717543	1.078916	

```
X_test[['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','CNT_FAM_MEMBERS',
        'HOUR_APPR_PROCESS_START_x','LIVE_CITY_NOT_WORK_CITY','OBS_30_CNT_SOCIAL_CIRCLE',
        'DEF_30_CNT_SOCIAL_CIRCLE','OBS_60_CNT_SOCIAL_CIRCLE','DEF_60_CNT_SOCIAL_CIRCLE',
        'DAYS_LAST_PHONE_CHANGE','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','AMT_ANNUITY_y','AMT_APPLICATION','AMT_CREDIT_y',
        'AMT_GOODS_PRICE_y','HOUR_APPR_PROCESS_START_y','FLAG_LAST_APPL_PER_CONTRACT',
        'NFLAG_LAST_APPL_IN_DAY','DAYS_DECISION','SELLERPLACE_AREA','CNT_PAYMENT',
        'DAYS_FIRST_DRAWING','DAYS_FIRST_DUE','DAYS_LAST_DUE_1ST_VERSION','DAYS_LAST_DUE',
        'DAYS_TERMINATION']] = scaler.transform(X_test[['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',
'CNT_FAM_MEMBERS','HOUR_APPR_PROCESS_START_x','LIVE_CITY_NOT_WORK_CITY',
'OBS_30_CNT_SOCIAL_CIRCLE','DEF_30_CNT_SOCIAL_CIRCLE','OBS_60_CNT_SOCIAL_
'DEF_60_CNT_SOCIAL_CIRCLE','DAYS_LAST_PHONE_CHANGE','AMT_REQ_CREDIT_BUREA
'AMT_REQ_CREDIT_BUREAU_DAY','AMT_REQ_CREDIT_BUREAU_WEEK','AMT_REQ_CREDIT_
'AMT_REQ_CREDIT_BUREAU_QRT','AMT_REQ_CREDIT_BUREAU_YEAR','AMT_ANNUITY_y',
'AMT_CREDIT_y','AMT_GOODS_PRICE_y','HOUR_APPR_PROCESS_START_y','FLAG_LAST
'NFLAG_LAST_APPL_IN_DAY','DAYS_DECISION','SELLERPLACE_AREA','CNT_PAYMENT'
'DAYS_FIRST_DUE','DAYS_LAST_DUE_1ST_VERSION','DAYS_LAST_DUE','DAYS_TERMIN
```

```
X_test.head()
```

	CNT_CHILDREN	AMT_INCOME_TOTAL	AMT_CREDIT_x	AMT_ANNUITY_x	AMT_GOODS_PRICE_x	REGION_POPULATION_RELATIVE	DAYS_BIRTH	D
856561	-0.562809	-0.482005	-0.356902	-0.276076	-0.220041	1.868948	-0.029337	
569831	-0.562809	2.045258	-0.232804	0.656660	-0.220041	-0.139115	0.144992	
1122558	0.805130	0.079609	0.509318	-0.114946	0.359337	-1.032655	0.154716	
1232806	-0.562809	-0.341602	-0.789366	-0.934859	-0.863794	-1.000154	-0.361096	
231290	0.805130	-0.996818	1.562862	0.890411	1.067465	-0.199350	-1.226955	

```
# Checking the Converted Rate
Target = round((sum(mergeddf['TARGET'])/len(mergeddf['TARGET'].index))*100,2)
print("We have almost {} %  Converted rate after successful data manipulation".format(Target))
```

```
We have almost 8.66 %  Converted rate after successful data manipulation
```

Model Building

Decision Tree

It is a type of supervised learning algorithm that is mostly used for classification problems. Surprisingly, it works for both categorical and continuous dependent variables. In this algorithm, we split the population into two or more homogeneous sets. This is done based on most significant attributes/ independent variables to make as distinct groups as possible.

```
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score
model = DecisionTreeClassifier()
```

```
# fit the model with the training data
model.fit(X_train,y_train)
```

```
     DecisionTreeClassifier()
```

```
# predict the target on the train dataset
predict_train = model.predict(X_train)
predict_train
```

```
     array([0, 0, 0, ..., 0, 0, 0])
```

```
trainaccuracy = accuracy_score(y_train,predict_train)
print('accuracy_score on train dataset : ', trainaccuracy)
```

```
     accuracy_score on train dataset :  1.0
```

VIF

```
# Check for the VIF values of the feature variables.
```

```
from statsmodels.stats.outliers_influence import variance_inflation_factor
# Create a dataframe that will contain the names of all the feature variables and their respective VIFs
vif = pd.DataFrame()
vif['Features'] = X_train.columns
vif['VIF'] = [variance_inflation_factor(X_train.values, i) for i in range(X_train.shape[1])]
vif['VIF'] = round(vif['VIF'], 2)
vif = vif.sort_values(by = "VIF", ascending = False)
vif.tail()
```

	Features	VIF
222	NAME_GOODS_CATEGORY_Animals	NaN
230	NAME_GOODS_CATEGORY_Education	NaN
231	NAME_GOODS_CATEGORY_Fitness	NaN
246	NAME_GOODS_CATEGORY_Weapon	NaN
269	NAME_SELLER_INDUSTRY_Tourism	NaN

```
features_to_remove = vif.loc[vif['VIF'] >= 4.99,'Features'].values
features_to_remove = list(features_to_remove)
print(features_to_remove)
```

```
['CNT_CHILDREN', 'CODE_REJECT_REASON_SYSTEM', 'NAME_GOODS_CATEGORY_Insurance', 'NAME_PORTFOLIO_Cars', 'NAME_GOODS_CATEGORY_Office
```

```
X_train = X_train.drop(columns=features_to_remove, axis = 1)
X_train.head()
```

	AMT_INCOME_TOTAL	AMT_ANNUITY_x	REGION_POPULATION_RELATIVE	DAYS_BIRTH	DAYS_REGISTRATION	DAYS_ID_PUBLISH	HOUR_APPR_PR
1074420	-0.013994	-0.358424	0.580838	0.345483	-1.202882	-1.098868	

```
X_test = X_test.drop(columns=features_to_remove, axis = 1)
X_test.head()
```

	AMT_INCOME_TOTAL	AMT_ANNUITY_x	REGION_POPULATION_RELATIVE	DAYS_BIRTH	DAYS_REGISTRATION	DAYS_ID_PUBLISH	HOUR_APPR_PR
856561	-0.482005	-0.276076	1.868948	-0.029337	0.359641	0.606410	
569831	2.045258	0.656660	-0.139115	0.144992	0.719019	-1.670816	
1122558	0.079609	-0.114946	-1.032655	0.154716	0.660008	-1.673458	
1232806	-0.341602	-0.934859	-1.000154	-0.361096	-0.879022	0.591220	
231290	-0.996818	0.890411	-0.199350	-1.226955	-0.189073	-0.855162	

```
# Check for the VIF values of the feature variables.
from statsmodels.stats.outliers_influence import variance_inflation_factor
# Create a dataframe that will contain the names of all the feature variables and their respective VIFs
vif = pd.DataFrame()
vif['Features'] = X_train.columns
vif['VIF'] = [variance_inflation_factor(X_train.values, i) for i in range(X_train.shape[1])]
vif['VIF'] = round(vif['VIF'], 2)
vif = vif.sort_values(by = "VIF", ascending = False)
vif
```

	Features	VIF
24	FLAG_CONT_MOBILE_1	26.07
129	NAME_CLIENT_TYPE_Repeater	6.29
22	FLAG_OWN_REALTY_1	4.01
7	DEF_30_CNT_SOCIAL_CIRCLE	3.60
8	DEF_60_CNT_SOCIAL_CIRCLE	3.59
58	NAME_FAMILY_STATUS_Married	3.09
57	NAME_INCOME_TYPE_Working	2.68
100	WEEKDAY_APPR_PROCESS_START_y_WEDNESDAY	2.20
99	WEEKDAY_APPR_PROCESS_START_y_TUESDAY	2.14
95	WEEKDAY_APPR_PROCESS_START_y_MONDAY	2.14
64	WEEKDAY_APPR_PROCESS_START_x_TUESDAY	2.14
123	NAME_PAYMENT_TYPE_XNA	2.13
65	WEEKDAY_APPR_PROCESS_START_x_WEDNESDAY	2.11
98	WEEKDAY_APPR_PROCESS_START_y_THURSDAY	2.08
63	WEEKDAY_APPR_PROCESS_START_x_THURSDAY	2.04
96	WEEKDAY_APPR_PROCESS_START_y_SATURDAY	2.02
60	WEEKDAY_APPR_PROCESS_START_x_MONDAY	2.02
21	FLAG_OWN_CAR_1	1.91
47	CODE_GENDER_M	1.89
17	FLAG_LAST_APPL_PER_CONTRACT	1.89
18	NFLAG_LAST_APPL_IN_DAY	1.88
97	WEEKDAY_APPR_PROCESS_START_y_SUNDAY	1.72

61	WEEKDAY_APPR_PROCESS_START_x_SATURDAY	1.71
128	NAME_CLIENT_TYPE_Refreshed	1.64
25	FLAG_PHONE_1	1.62
23	FLAG_WORK_PHONE_1	1.54
3	DAYS_BIRTH	1.53
46	NFLAG_INSURED_ON_APPROVAL_1.0	1.50
0	AMT_INCOME_TOTAL	1.37
62	WEEKDAY_APPR_PROCESS_START_x_SUNDAY	1.35
19	DAYS_DECISION	1.35
16	HOUR_APPR_PROCESS_START_y	1.28
1	AMT_ANNUITY_x	1.28
54	NAME_INCOME_TYPE_State servant	1.27
28	REG_CITY_NOT_LIVE_CITY_1	1.27
6	HOUR_APPR_PROCESS_START_x	1.25
4	DAYS_REGISTRATION	1.18
27	REG_REGION_NOT_LIVE_REGION_1	1.16
26	FLAG_EMAIL_1	1.14
2	REGION_POPULATION_RELATIVE	1.13
15	AMT_REQ_CREDIT_BUREAU_YEAR	1.13
136	CHANNEL_TYPE_Contact center	1.13
127	NAME_TYPE_SUITE_y_Spouse, partner	1.11
135	CHANNEL_TYPE_Channel of corporate sales	1.11
5	DAYS_ID_PUBLISH	1.11
9	DAYS_LAST_PHONE_CHANGE	1.11

35	FLAG_DOCUMENT_11_1	1.09
52	NAME_TYPE_SUITE_x_Spouse, partner	1.09
104	NAME_CASH_LOAN_PURPOSE_Buying a home	1.08
40	FLAG_DOCUMENT_16_1	1.08
125	NAME_TYPE_SUITE_y_Other_A	1.07
37	FLAG_DOCUMENT_13_1	1.07
14	AMT_REQ_CREDIT_BUREAU_QRT	1.06
124	NAME_TYPE_SUITE_y_Group of people	1.06
13	AMT_REQ_CREDIT_BUREAU_MON	1.06
126	NAME_TYPE_SUITE_y_Other_B	1.05
10	AMT_REQ_CREDIT_BUREAU_HOUR	1.05
11	AMT_REQ_CREDIT_BUREAU_DAY	1.05
119	NAME_CASH_LOAN_PURPOSE_Urgent needs	1.05
20	SELLERPLACE_AREA	1.05
59	NAME_HOUSING_TYPE_Office apartment	1.04
86	ORGANIZATION_TYPE_Services	1.04
88	ORGANIZATION_TYPE_Trade: type 1	1.04
92	ORGANIZATION_TYPE_Transport: type 1	1.04
33	FLAG_DOCUMENT_9_1	1.04
130	NAME_CLIENT_TYPE_XNA	1.04
44	FLAG_DOCUMENT_20_1	1.04
85	ORGANIZATION_TYPE_Religion	1.04
106	NAME_CASH_LOAN_PURPOSE_Buying a used car	1.03
108	NAME_CASH_LOAN_PURPOSE_Education	1.03

93	ORGANIZATION_TYPE_Transport: type 3	1.03
109	NAME_CASH_LOAN_PURPOSE_Everyday expenses	1.03
83	ORGANIZATION_TYPE_Mobile	1.03
69	ORGANIZATION_TYPE_Emergency	1.03
75	ORGANIZATION_TYPE_Industry: type 2	1.03
71	ORGANIZATION_TYPE_Industry: type 1	1.03
31	FLAG_DOCUMENT_5_1	1.03
42	FLAG_DOCUMENT_18_1	1.03
70	ORGANIZATION_TYPE_Hotel	1.03
51	NAME_TYPE_SUITE_x_Other_B	1.03
67	ORGANIZATION_TYPE_Culture	1.02
107	NAME_CASH_LOAN_PURPOSE_Car repairs	1.02
110	NAME_CASH_LOAN_PURPOSE_Furniture	1.02
114	NAME_CASH_LOAN_PURPOSE_Medicine	1.02
122	NAME_PAYMENT_TYPE_Non-cash from your account	1.02
117	NAME_CASH_LOAN_PURPOSE_Purchase of electronic equipment	1.02
50	NAME_TYPE_SUITE_x_Other_A	1.02
49	NAME_TYPE_SUITE_x_Group of people	1.02
12	AMT_REQ_CREDIT_BUREAU_WEEK	1.02
137	NAME_SELLER_INDUSTRY_Jewelry	1.02
68	ORGANIZATION_TYPE_Electricity	1.02
138	NAME_SELLER_INDUSTRY_MLM partners	1.02
77	ORGANIZATION_TYPE_Industry: type 5	1.02

79	ORGANIZATION_TYPE_Industry: type 7	1.02
76	ORGANIZATION_TYPE_Industry: type 4	1.02
94	ORGANIZATION_TYPE_University	1.02
103	NAME_CASH_LOAN_PURPOSE_Buying a holiday home / land	1.01
81	ORGANIZATION_TYPE_Insurance	1.01
87	ORGANIZATION_TYPE_Telecom	1.01
32	FLAG_DOCUMENT_7_1	1.01
38	FLAG_DOCUMENT_14_1	1.01
39	FLAG_DOCUMENT_15_1	1.01
41	FLAG_DOCUMENT_17_1	1.01
89	ORGANIZATION_TYPE_Trade: type 4	1.01
43	FLAG_DOCUMENT_19_1	1.01
91	ORGANIZATION_TYPE_Trade: type 6	1.01
78	ORGANIZATION_TYPE_Industry: type 6	1.01
120	NAME_CASH_LOAN_PURPOSE_Wedding / gift / holiday	1.01
121	NAME_PAYMENT_TYPE_Cashless from the account of the employer	1.01
102	NAME_CASH_LOAN_PURPOSE_Buying a garage	1.01
116	NAME_CASH_LOAN_PURPOSE_Payments on other loans	1.01
74	ORGANIZATION_TYPE_Industry: type 13	1.01
113	NAME_CASH_LOAN_PURPOSE_Journey	1.01
111	NAME_CASH_LOAN_PURPOSE_Gasification / water supply	1.01
73	ORGANIZATION_TYPE_Industry: type 12	1.01
66	ORGANIZATION_TYPE_Cleaning	1.01
82	ORGANIZATION_TYPE_Legal Services	1.01

72	ORGANIZATION_TYPE_Industry: type 10	1.01
101	NAME_CASH_LOAN_PURPOSE_Business development	1.01
105	NAME_CASH_LOAN_PURPOSE_Buying a new car	1.01
84	ORGANIZATION_TYPE_Realtor	1.01
29	FLAG_DOCUMENT_2_1	NaN
30	FLAG_DOCUMENT_4_1	NaN
34	FLAG_DOCUMENT_10_1	NaN
36	FLAG_DOCUMENT_12_1	NaN
45	FLAG_DOCUMENT_21_1	NaN
48	CODE_GENDER_XNA	NaN
53	NAME_INCOME_TYPE_Maternity leave	NaN
55	NAME_INCOME_TYPE_Student	NaN
56	NAME_INCOME_TYPE_Unemployed	NaN
80	ORGANIZATION_TYPE_Industry: type 8	NaN
90	ORGANIZATION_TYPE_Trade: type 5	NaN
112	NAME_CASH_LOAN_PURPOSE_Hobby	NaN

The NaN, in if in case, is interpretted as no correlation between the two variables.

118	NAME_CASH_LOAN_PURPOSE_Refusal to name the goal	NaN

```
features_to_remove = vif.loc[vif['VIF'] >= 4.99,'Features'].values
features_to_remove = list(features_to_remove)
print(features_to_remove)
```

```
['FLAG_CONT_MOBILE_1', 'NAME_CLIENT_TYPE_Repeater']
```

```
X_train = X_train.drop(columns=features_to_remove, axis = 1)
X_train.head()
```

	AMT_INCOME_TOTAL	AMT_ANNUITY_x	REGION_POPULATION_RELATIVE	DAYS_BIRTH	DAYS_REGISTRATION	DAYS_ID_PUBLISH	HOUR_APPR_PR
1074420	-0.013994	-0.358424	0.580838	0.345483	-1.202882	-1.098868	
723991	1.951656	3.994344	-0.303311	1.481749	1.108321	1.322336	
209411	-0.856415	-0.077339	-1.242339	-0.601869	0.596523	1.329601	
1140628	-0.856415	-0.918325	0.580838	1.203008	0.218967	0.785391	
1262058	0.313615	-0.887201	-0.717543	1.078916	-1.158134	0.984847	

```
X_test = X_test.drop(columns=features_to_remove, axis = 1)
X_test.head()
```

	AMT_INCOME_TOTAL	AMT_ANNUITY_x	REGION_POPULATION_RELATIVE	DAYS_BIRTH	DAYS_REGISTRATION	DAYS_ID_PUBLISH	HOUR_APPR_PR
856561	-0.482005	-0.276076	1.868948	-0.029337	0.359641	0.606410	
569831	2.045258	0.656660	-0.139115	0.144992	0.719019	-1.670816	
1122558	0.079609	-0.114946	-1.032655	0.154716	0.660008	-1.673458	
1232806	-0.341602	-0.934859	-1.000154	-0.361096	-0.879022	0.591220	
231290	-0.996818	0.890411	-0.199350	-1.226955	-0.189073	-0.855162	

```
# Check for the VIF values of the feature variables.
from statsmodels.stats.outliers_influence import variance_inflation_factor
# Create a dataframe that will contain the names of all the feature variables and their respective VIFs
vif = pd.DataFrame()
vif['Features'] = X_train.columns
vif['VIF'] = [variance_inflation_factor(X_train.values, i) for i in range(X_train.shape[1])]
vif['VIF'] = round(vif['VIF'], 2)
```

```
vif = vif.sort_values(by = "VIF", ascending = False)
vif
```

	Features	VIF
7	DEF_30_CNT_SOCIAL_CIRCLE	3.60
8	DEF_60_CNT_SOCIAL_CIRCLE	3.59
22	FLAG_OWN_REALTY_1	3.41
57	NAME_FAMILY_STATUS_Married	2.90
56	NAME_INCOME_TYPE_Working	2.53
122	NAME_PAYMENT_TYPE_XNA	2.01
17	FLAG_LAST_APPL_PER_CONTRACT	1.89
18	NFLAG_LAST_APPL_IN_DAY	1.88
46	CODE_GENDER_M	1.88
21	FLAG_OWN_CAR_1	1.88
63	WEEKDAY_APPR_PROCESS_START_x_TUESDAY	1.84
99	WEEKDAY_APPR_PROCESS_START_y_WEDNESDAY	1.83
98	WEEKDAY_APPR_PROCESS_START_y_TUESDAY	1.81
94	WEEKDAY_APPR_PROCESS_START_y_MONDAY	1.79
64	WEEKDAY_APPR_PROCESS_START_x_WEDNESDAY	1.78
97	WEEKDAY_APPR_PROCESS_START_y_THURSDAY	1.74
62	WEEKDAY_APPR_PROCESS_START_x_THURSDAY	1.73
95	WEEKDAY_APPR_PROCESS_START_y_SATURDAY	1.73
59	WEEKDAY_APPR_PROCESS_START_x_MONDAY	1.72
24	FLAG_PHONE_1	1.61
23	FLAG_WORK_PHONE_1	1.53
60	WEEKDAY_APPR_PROCESS_START_x_SATURDAY	1.53

3	DAYS_BIRTH	1.51
96	WEEKDAY_APPR_PROCESS_START_y_SUNDAY	1.50
45	NFLAG_INSURED_ON_APPROVAL_1.0	1.45
0	AMT_INCOME_TOTAL	1.37
1	AMT_ANNUITY_x	1.28
16	HOUR_APPR_PROCESS_START_y	1.27
27	REG_CITY_NOT_LIVE_CITY_1	1.26
19	DAYS_DECISION	1.25
6	HOUR_APPR_PROCESS_START_x	1.24
61	WEEKDAY_APPR_PROCESS_START_x_SUNDAY	1.24
53	NAME_INCOME_TYPE_State servant	1.24
4	DAYS_REGISTRATION	1.18
26	REG_REGION_NOT_LIVE_REGION_1	1.16
127	NAME_CLIENT_TYPE_Refreshed	1.14
25	FLAG_EMAIL_1	1.13
134	CHANNEL_TYPE_Contact center	1.13
2	REGION_POPULATION_RELATIVE	1.13
126	NAME_TYPE_SUITE_y_Spouse, partner	1.11
15	AMT_REQ_CREDIT_BUREAU_YEAR	1.11
133	CHANNEL_TYPE_Channel of corporate sales	1.11
5	DAYS_ID_PUBLISH	1.11
34	FLAG_DOCUMENT_11_1	1.09
9	DAYS_LAST_PHONE_CHANGE	1.09
51	NAME_TYPE_SUITE_x_Spouse, partner	1.09

103	NAME_CASH_LOAN_PURPOSE_Buying a home	1.08
39	FLAG_DOCUMENT_16_1	1.07
124	NAME_TYPE_SUITE_y_Other_A	1.07
36	FLAG_DOCUMENT_13_1	1.07
13	AMT_REQ_CREDIT_BUREAU_MON	1.06
14	AMT_REQ_CREDIT_BUREAU_QRT	1.06
10	AMT_REQ_CREDIT_BUREAU_HOUR	1.05
118	NAME_CASH_LOAN_PURPOSE_Urgent needs	1.05
11	AMT_REQ_CREDIT_BUREAU_DAY	1.05
123	NAME_TYPE_SUITE_y_Group of people	1.05
20	SELLERPLACE_AREA	1.05
58	NAME_HOUSING_TYPE_Office apartment	1.04
84	ORGANIZATION_TYPE_Religion	1.04
32	FLAG_DOCUMENT_9_1	1.04
91	ORGANIZATION_TYPE_Transport: type 1	1.04
125	NAME_TYPE_SUITE_y_Other_B	1.04
43	FLAG_DOCUMENT_20_1	1.04
87	ORGANIZATION_TYPE_Trade: type 1	1.04
85	ORGANIZATION_TYPE_Services	1.03
108	NAME_CASH_LOAN_PURPOSE_Everyday expenses	1.03
92	ORGANIZATION_TYPE_Transport: type 3	1.03
105	NAME_CASH_LOAN_PURPOSE_Buying a used car	1.03
82	ORGANIZATION_TYPE_Mobile	1.03
128	NAME_CLIENT_TYPE_XNA	1.03

68	ORGANIZATION_TYPE_Emergency	1.03
69	ORGANIZATION_TYPE_Hotel	1.03
70	ORGANIZATION_TYPE_Industry: type 1	1.03
30	FLAG_DOCUMENT_5_1	1.03
50	NAME_TYPE_SUITE_x_Other_B	1.03
74	ORGANIZATION_TYPE_Industry: type 2	1.03
116	NAME_CASH_LOAN_PURPOSE_Purchase of electronic equipment	1.02
66	ORGANIZATION_TYPE_Culture	1.02
107	NAME_CASH_LOAN_PURPOSE_Education	1.02
109	NAME_CASH_LOAN_PURPOSE_Furniture	1.02
106	NAME_CASH_LOAN_PURPOSE_Car repairs	1.02
67	ORGANIZATION_TYPE_Electricity	1.02
121	NAME_PAYMENT_TYPE_Non-cash from your account	1.02
48	NAME_TYPE_SUITE_x_Group of people	1.02
41	FLAG_DOCUMENT_18_1	1.02
12	AMT_REQ_CREDIT_BUREAU_WEEK	1.02
135	NAME_SELLER_INDUSTRY_Jewelry	1.02
49	NAME_TYPE_SUITE_x_Other_A	1.02
136	NAME_SELLER_INDUSTRY_MLM partners	1.02
78	ORGANIZATION_TYPE_Industry: type 7	1.02
93	ORGANIZATION_TYPE_University	1.02
75	ORGANIZATION_TYPE_Industry: type 4	1.02
100	NAME_CASH_LOAN_PURPOSE_Business development	1.01

119	NAME_CASH_LOAN_PURPOSE_Wedding / gift / holiday	1.01
83	ORGANIZATION_TYPE_Realtor	1.01
77	ORGANIZATION_TYPE_Industry: type 6	1.01
31	FLAG_DOCUMENT_7_1	1.01
37	FLAG_DOCUMENT_14_1	1.01
38	FLAG_DOCUMENT_15_1	1.01
40	FLAG_DOCUMENT_17_1	1.01
86	ORGANIZATION_TYPE_Telecom	1.01
42	FLAG_DOCUMENT_19_1	1.01
76	ORGANIZATION_TYPE_Industry: type 5	1.01
88	ORGANIZATION_TYPE_Trade: type 4	1.01
120	NAME_PAYMENT_TYPE_Cashless from the account of the employer	1.01
90	ORGANIZATION_TYPE_Trade: type 6	1.01
101	NAME_CASH_LOAN_PURPOSE_Buying a garage	1.01
115	NAME_CASH_LOAN_PURPOSE_Payments on other loans	1.01
113	NAME_CASH_LOAN_PURPOSE_Medicine	1.01
112	NAME_CASH_LOAN_PURPOSE_Journey	1.01
110	NAME_CASH_LOAN_PURPOSE_Gasification / water supply	1.01
73	ORGANIZATION_TYPE_Industry: type 13	1.01
72	ORGANIZATION_TYPE_Industry: type 12	1.01

```
# fit the model with the training data
model.fit(X_train,y_train)
```

```
DecisionTreeClassifier()
```

```
# predict the target on the train dataset
```

```
# predict the target on the train dataset
predict_train = model.predict(X_train)
predict_train
```

```
array([0, 0, 0, ..., 0, 0, 0])
```

```
trainaccuracy = accuracy_score(y_train,predict_train)
print('accuracy_score on train dataset : ', trainaccuracy)
```

```
accuracy_score on train dataset :  1.0
```

44 FLAG_DOCUMENT_21 1 NaN

```
from sklearn import metrics
# Confusion matrix
confusion = metrics.confusion_matrix(y_train, predict_train )
print(confusion)
```

```
[[4456    0]
 [   0  444]]
```

79 ORGANIZATION_TYPE Industry: type 8 NaN

```
TP = confusion[1,1] # true positive
TN = confusion[0,0] # true negatives
FP = confusion[0,1] # false positives
FN = confusion[1,0] # false negatives
```

```
# Let's see the sensitivity of our model
trainsensitivity= TP / float(TP+FN)
trainsensitivity
```

```
1.0
```

```
# Let us calculate specificity
trainspecificity= TN / float(TN+FP)
trainspecificity
```

```
1.0
```

```
# Calculate false postive rate - predicting Defaulted when customer does not have Defaulted
print(FP/ float(TN+FP))
```

```
0.0
```

```
# Positive predictive value
print (TP / float(TP+FP))
```

```
1.0
```

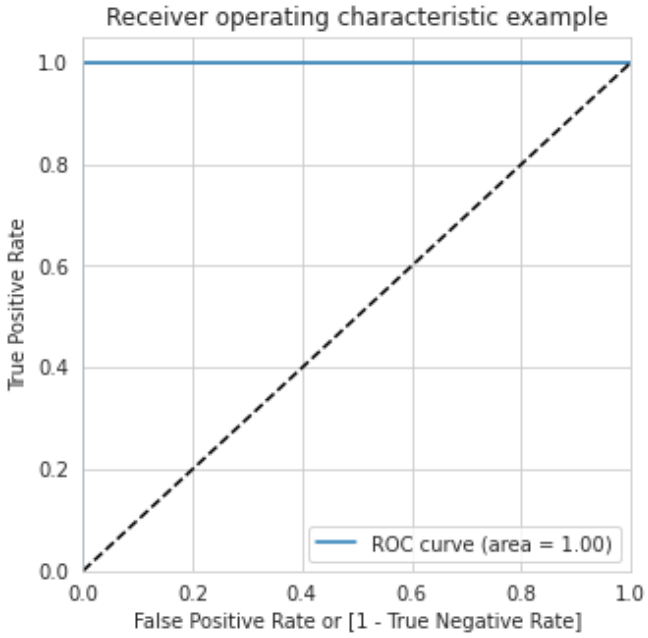
```
# Negative predictive value
print(TN / float(TN+ FN))
```

```
1.0
```

Plotting the ROC Curve

```
def draw_roc( actual, probs ):
    fpr, tpr, thresholds = metrics.roc_curve( actual, probs,
                                              drop_intermediate = False )
    auc_score = metrics.roc_auc_score( actual, probs )
    plt.figure(figsize=(5, 5))
    plt.plot( fpr, tpr, label='ROC curve (area = %0.2f)' % auc_score )
    plt.plot([0, 1], [0, 1], 'k--')
    plt.xlim([0.0, 1.0])
    plt.ylim([0.0, 1.05])
    plt.xlabel('False Positive Rate or [1 - True Negative Rate]')
    plt.ylabel('True Positive Rate')
    plt.title('Receiver operating characteristic example')
    plt.legend(loc="lower right")
    plt.show()

    return None
```

```
draw_roc(y_train,predict_train)
```



Precision and Recall

```
#Using sklearn utilities for the same


from sklearn.metrics import precision_score, recall_score
precision_score(y_train,predict_train)
```

```
1.0
```

```
recall_score(y_train,predict_train)
```

```
1.0
```

Making predictions on the test set

```
# predict the target on the test dataset
predict_test = model.predict(X_test)
print('Target on test data\n\n',predict_test)
```

```
     Target on test data

      [0 0 0 ... 0 0 0]
```

```
confusion2 = metrics.confusion_matrix(y_test, predict_test )
print(confusion2)
```

```
     [[1703  196]
      [ 169   32]]
```

```
TP = confusion2[1,1] # true positive
TN = confusion2[0,0] # true negatives
FP = confusion2[0,1] # false positives
FN = confusion2[1,0] # false negatives
```

```
# Let's check the overall accuracy.
testaccuracy= accuracy_score(y_test,predict_test)
testaccuracy
```

```
     0.8261904761904761
```

```
# Let's see the sensitivity of our lmodel
testsensitivity=TP / float(TP+FN)
testsensitivity
```

```
     0.15920398009950248
```

```
# Let us calculate specificity
testspecificity= TN / float(TN+FP)
testspecificity
```

```
0.8967877830437072
```

Final Observation:

```
# Let us compare the values obtained for Train & Test:
print("Train Data Accuracy    :{} %".format(round((trainaccuracy*100),2)))
print("Train Data Sensitivity :{} %".format(round((trainsensitivity*100),2)))
print("Train Data Specificity :{} %".format(round((trainspecificity*100),2)))
print("Test Data Accuracy     :{} %".format(round((testaccuracy*100),2)))
print("Test Data Sensitivity  :{} %".format(round((testsensitivity*100),2)))
print("Test Data Specificity  :{} %".format(round((testspecificity*100),2)))
```

```
Train Data Accuracy    :100.0 %
Train Data Sensitivity :100.0 %
Train Data Specificity :100.0 %
Test Data Accuracy     :82.62 %
Test Data Sensitivity  :15.92 %
Test Data Specificity  :89.68 %
```