1.0 Common commands

Following actions are performed in this section:

- **Google drive is mounted**
- **Required packages are is installed and updated**
- **Required libraries are imported**
- **Custom functions (for dataframe optimisation, group plot and pie chart) are written**

1.1 Mount drive

In [ ]:

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount
("/content/drive", force_remount=True).
```

> **All the relevant files can be accessed through the following link:**
> https://drive.google.com/drive/folders/1evFZRwFWh4zkR9CiT46llB9PlaXFLfLA?usp=sharing

1.2 Install packages

pyod is installed and updated

In [ ]:

```
'''In this code cell, all the relevant packages are installed and updated'''

!pip install pyod
!pip install --upgrade pyod
```

```
Requirement already satisfied: pyod in /usr/local/lib/python3.7/dist-packages (0.9.5)
Requirement already satisfied: matplotlib in /usr/local/lib/python3.7/dist-packages (from
pyod) (3.2.2)
Requirement already satisfied: numba>=0.35 in /usr/local/lib/python3.7/dist-packages (fro
m pyod) (0.51.2)
Requirement already satisfied: scipy>=1.3.1 in /usr/local/lib/python3.7/dist-packages (fr
om pyod) (1.4.1)
Requirement already satisfied: statsmodels in /usr/local/lib/python3.7/dist-packages (fro
m pyod) (0.10.2)
Requirement already satisfied: six in /usr/local/lib/python3.7/dist-packages (from pyod)
(1.15.0)
Requirement already satisfied: joblib in /usr/local/lib/python3.7/dist-packages (from pyo
d) (1.1.0)
Requirement already satisfied: scikit-learn>=0.20.0 in /usr/local/lib/python3.7/dist-pack
ages (from pyod) (1.0.1)
Requirement already satisfied: numpy>=1.13 in /usr/local/lib/python3.7/dist-packages (fro
m pyod) (1.19.5)
Requirement already satisfied: llvmlite<0.35,>=0.34.0.dev0 in /usr/local/lib/python3.7/di
st-packages (from numba>=0.35->pyod) (0.34.0)
Requirement already satisfied: setuptools in /usr/local/lib/python3.7/dist-packages (from
numba>=0.35->pyod) (57.4.0)
Requirement already satisfied: threadpoolctl>=2.0.0 in /usr/local/lib/python3.7/dist-pack
ages (from scikit-learn>=0.20.0->pyod) (3.0.0)
Requirement already satisfied: pyparsing!=2.0.4,!=2.1.2,!=2.1.6,>=2.0.1 in /usr/local/lib
/python3.7/dist-packages (from matplotlib->pyod) (3.0.6)
Requirement already satisfied: kiwisolver>=1.0.1 in /usr/local/lib/python3.7/dist-package
s (from matplotlib->pyod) (1.3.2)
```

```
s (from matplotlib->pyod) (1.3.2)
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.7/dist-packages (fr
om matplotlib->pyod) (0.11.0)
Requirement already satisfied: python-dateutil>=2.1 in /usr/local/lib/python3.7/dist-pack
ages (from matplotlib->pyod) (2.8.2)
Requirement already satisfied: patsy>=0.4.0 in /usr/local/lib/python3.7/dist-packages (fr
om statsmodels->pyod) (0.5.2)
Requirement already satisfied: pandas>=0.19 in /usr/local/lib/python3.7/dist-packages (fr
om statsmodels->pyod) (1.1.5)
Requirement already satisfied: pytz>=2017.2 in /usr/local/lib/python3.7/dist-packages (fr
om pandas>=0.19->statsmodels->pyod) (2018.9)
Requirement already satisfied: pyod in /usr/local/lib/python3.7/dist-packages (0.9.5)
Requirement already satisfied: matplotlib in /usr/local/lib/python3.7/dist-packages (from
pyod) (3.2.2)
Requirement already satisfied: statsmodels in /usr/local/lib/python3.7/dist-packages (fro
m pyod) (0.10.2)
Requirement already satisfied: six in /usr/local/lib/python3.7/dist-packages (from pyod)
 (1.15.0)
Requirement already satisfied: numpy>=1.13 in /usr/local/lib/python3.7/dist-packages (fro
m pyod) (1.19.5)
Requirement already satisfied: scikit-learn>=0.20.0 in /usr/local/lib/python3.7/dist-pack
ages (from pyod) (1.0.1)
Requirement already satisfied: joblib in /usr/local/lib/python3.7/dist-packages (from pyo
d) (1.1.0)
Requirement already satisfied: scipy>=1.3.1 in /usr/local/lib/python3.7/dist-packages (fr
om pyod) (1.4.1)
Requirement already satisfied: numba>=0.35 in /usr/local/lib/python3.7/dist-packages (fro
m pyod) (0.51.2)
Requirement already satisfied: llvmlite<0.35,>=0.34.0.dev0 in /usr/local/lib/python3.7/di
st-packages (from numba>=0.35->pyod) (0.34.0)
Requirement already satisfied: setuptools in /usr/local/lib/python3.7/dist-packages (from
numba>=0.35->pyod) (57.4.0)
Requirement already satisfied: threadpoolctl>=2.0.0 in /usr/local/lib/python3.7/dist-pack
ages (from scikit-learn>=0.20.0->pyod) (3.0.0)
Requirement already satisfied: kiwisolver>=1.0.1 in /usr/local/lib/python3.7/dist-package
s (from matplotlib->pyod) (1.3.2)
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.7/dist-packages (fr
om matplotlib->pyod) (0.11.0)
Requirement already satisfied: python-dateutil>=2.1 in /usr/local/lib/python3.7/dist-pack
ages (from matplotlib->pyod) (2.8.2)
Requirement already satisfied: pyparsing!=2.0.4,!=2.1.2,!=2.1.6,>=2.0.1 in /usr/local/lib
/python3.7/dist-packages (from matplotlib->pyod) (3.0.6)
Requirement already satisfied: patsy>=0.4.0 in /usr/local/lib/python3.7/dist-packages (fr
om statsmodels->pyod) (0.5.2)
Requirement already satisfied: pandas>=0.19 in /usr/local/lib/python3.7/dist-packages (fr
om statsmodels->pyod) (1.1.5)
Requirement already satisfied: pytz>=2017.2 in /usr/local/lib/python3.7/dist-packages (fr
om pandas>=0.19->statsmodels->pyod) (2018.9)
```

1.3 Import libraries

Some libraries imported here may not be used. They were imported while trying different approach.

In [ ]:

```python
'''In this code cell, all the relevant libraries are imported'''

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from matplotlib_venn import venn3
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.impute import SimpleImputer
from google.colab import files
from pyod.models.cblof import CBLOF
from pyod.models.hbos import HBOS
from sklearn.ensemble import ExtraTreesClassifier
from sklearn.feature_selection import SelectFromModel
```

```
from sklearn.manifold import TSNE
from xgboost import XGBClassifier
from sklearn.ensemble import GradientBoostingClassifier
```

1.4 Define custom functions

Dataframe optimiser

In [ ]:

```
'''In this code cell a function called dataframe_optimizer is defined. This will be used to
reduce space consumption by dataframes.'''

#Credit - https://www.kaggle.com/rinnqd/reduce-memory-usage and
#https://www.analyticsvidhya.com/blog/2021/04/how-to-reduce-memory-usage-in-python-pandas/
def dataframe_optimizer(df):
  '''This is a dataframe optimizer'''
  start_mem=np.round(df.memory_usage().sum()/1024**2,2)
  for col in df.columns:
    col_type=df[col].dtype
    if col_type!=object:
      c_min=df[col].min()
      c_max=df[col].max()
      if str(col_type)[:3]=='int':
        if c_min>np.iinfo(np.int8).min and c_max<np.iinfo(np.int8).max:
            df[col]=df[col].astype(np.int8)
        elif c_min>np.iinfo(np.int16).min and c_max<np.iinfo(np.int16).max:
            df[col]=df[col].astype(np.int16)
        elif c_min>np.iinfo(np.int32).min and c_max<np.iinfo(np.int32).max:
            df[col]=df[col].astype(np.int32)
        elif c_min>np.iinfo(np.int64).min and c_max<np.iinfo(np.int64).max:
            df[col]=df[col].astype(np.int64)
      else:
        if c_min>np.finfo(np.float16).min and c_max<np.finfo(np.float16).max:
            df[col]=df[col].astype(np.float16)
        elif c_min>np.finfo(np.float32).min and c_max<np.finfo(np.float32).max:
            df[col]=df[col].astype(np.float32)
        else:
            df[col]=df[col].astype(np.float64)
  end_mem=np.round(df.memory_usage().sum()/1024**2,2)
  return df
```

Function for group plot

In [ ]:

```
'''In this code cell, a function to plot group plot is defined'''

def group_plot(size, y_axis_0, y_axis_1, classes, x_label, y_label, legend, title):
  '''This is a custom function to draw group plot'''
  x = np.arange(len(y_axis_0))
  width = 0.4
  plt.figure(figsize=size)
  plt.bar(x-0.2, y_axis_0, width, color='green')
  plt.bar(x+0.2, y_axis_1, width, color='red')
  plt.xticks(x, classes, rotation=90)
  plt.xlabel(x_label)
  plt.ylabel(y_label)
  plt.legend(legend)
  plt.title(title)
  plt.show()
```

Function for pie chart

In [ ]:

```
'''In this code cell, a function to plot pie chart is defined'''

def pie_chart(size, category, labels, title):
  '''This is a custom function to draw pie chart'''
  plt.figure(figsize = size)
  plt.pie(category, autopct='%1.2f%%')
  plt.legend(labels)
  plt.title(title)
  plt.show()
```

2.0 Data set level analysis

After importing relavant data, following analysis are performed in this section:

- **Analysis on common items across different data sets**
- **Analysis based on missing values**

2.1 Import relevant data

In [ ]:

```
'''In this code cell, data is read.'''

#Read application_train.csv
application_train = pd.read_csv('/content/drive/MyDrive/AI_ML_Project/Data/application_tr
ain.csv')

#Read application_test.csv
application_test = pd.read_csv('/content/drive/MyDrive/AI_ML_Project/Data/application_tes
t.csv')

#Read bureau.csv
bureau = pd.read_csv('/content/drive/MyDrive/AI_ML_Project/Data/bureau.csv')

#Read previous_application.csv
previous_application = pd.read_csv('/content/drive/MyDrive/AI_ML_Project/Data/previous_ap
plication.csv')
```

2.2 Analysis on common items across different data sets

In [ ]:

```
'''In this code cell, count of applicants and set opration is performed for applicants fr
om
application_train, bureau and previous_application.'''

application_train_id = application_train['SK_ID_CURR'].to_numpy()
print('Total application in application_train: ', len(application_train_id))
bureau_id = bureau['SK_ID_CURR'].to_numpy()
print('Total unique application in bureau: ', len(set(bureau_id)))
previous_application_id = previous_application['SK_ID_CURR'].to_numpy()
print('Total unique application in previous: ', len(set(previous_application_id)))

print('-'*100)

application_train_not_in_bureau = np.array(list(set(application_train_id) - set(bureau_id
)))
print('Number of appl ication in application_train but not in bureau: ', len(application_
train_not_in_bureau))
print('Percentage of application not in bureau: ', (len(application_train_not_in_bureau)/
len(application_train_id))*100)
application_train_not_in_bureau_columns = ((len(set(application_train.columns) - set(bure
au.columns)))/len(set(application_train.columns)))*100
print('Percentage of columns from application that are not in bureau: ', application_trai
n_not_in_bureau_columns)
```

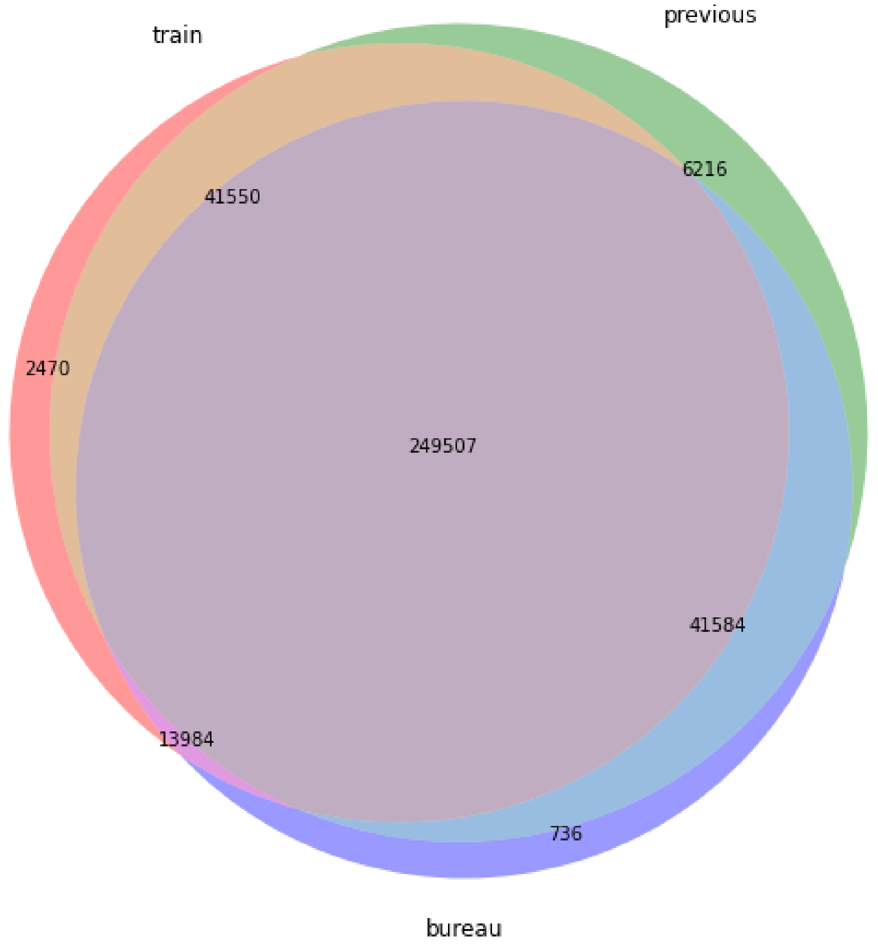
```
print('-'*100)

application_train_not_in_previous = np.array(list(set(application_train_id) - set(previous_application_id)))
print('Number of application in application_train but not in previous: ', len(application_train_not_in_previous))
print('Percentage of application not in previous: ', (len(application_train_not_in_previous)/len(application_train_id))*100)
application_train_not_in_previous_columns = ((len(set(application_train.columns) - set(previous_application.columns)))/len(set(application_train.columns)))*100
print('Percentage of columns from application that are not in previous: ', application_train_not_in_previous_columns)

print('-'*100)

application_train_not_in_bureau_not_in_previous = np.array(list(set(application_train_id) - set(bureau_id) - set(previous_application_id)))
print('Number of application in application_train but neither in bureau nor in previous: ', len(application_train_not_in_bureau_not_in_previous))
print('Percentage of application in application_train but neither in bureau nor in previous: ', (len(application_train_not_in_bureau_not_in_previous)/len(application_train_id))*100)
application_train_in_bureau_in_previous = np.array(list(set(application_train_id) & set(bureau_id) & set(previous_application_id)))
print('Number of application from application_train both in bureau and in previous: ', len(application_train_in_bureau_in_previous))
print('Percentage of application from application_train both in bureau and in previous: ', (len(application_train_in_bureau_in_previous)/len(application_train_id))*100)
```

```
Total application in application_train:  307511
Total unique application in bureau:  305811
Total unique application in previous:  338857
----------------------------------------------------------------------------------------------------
Number of appl ication in application_train but not in bureau:  44020
Percentage of application not in bureau:  14.314935075493235
Percentage of columns from application that are not in bureau:  98.36065573770492
----------------------------------------------------------------------------------------------------
Number of application in application_train but not in previous:  16454
Percentage of application not in previous:  5.350702901684818
Percentage of columns from application that are not in previous:  93.44262295081968
----------------------------------------------------------------------------------------------------
Number of application in application_train but neither in bureau nor in previous:  2470
Percentage of application in application_train but neither in bureau nor in previous:  0.8032232993291297
Number of application from application_train both in bureau and in previous:  249507
Percentage of application from application_train both in bureau and in previous:  81.13758532215107
```

In [ ]:

```
'''In this code cell, venn diagram is plotted for unique entries in application_train, bureau and previous_application'''

train_only = len(set(application_train_id) - set(bureau_id) - set(previous_application_id))
previous_only = len(set(previous_application_id) - set(application_train_id) - set(bureau_id))
train_previous_only = len(set(application_train_id) & set(previous_application_id) - set(bureau_id))
bureau_only = len(set(bureau_id) - set(application_train_id) - set(previous_application_id))
train_bureau_only = len(set(application_train_id) & set(bureau_id) - set(previous_application_id))
previous_bureau_only = len(set(previous_application_id) & set(bureau_id) - set(application_train_id))
all = len(set(application_train_id) & set(previous_application_id) & set(bureau_id))
plt.subplots(figsize = (10,10))
venn3(subsets=(train_only, previous_only, train_previous_only, bureau_only, train_bureau_
```

```
only, previous_bureau_only, all), set_labels=['train', 'previous', 'bureau'])
plt.show()
```



In [ ]:

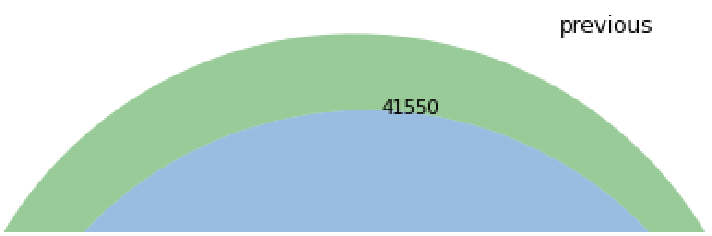
```
'''In this code cell, count of applicants and set opration is performed for applicants fr
om
application_test, bureau and previous_application.'''

application_test_id = application_test['SK_ID_CURR'].to_numpy()
print('Total application in application_test: ', len(application_test_id))
bureau_id = bureau['SK_ID_CURR'].to_numpy()
print('Total unique application in bureau: ', len(set(bureau_id)))
previous_application_id = previous_application['SK_ID_CURR'].to_numpy()
print('Total unique application in previous: ', len(set(previous_application_id)))

print('-'*100)

application_test_not_in_bureau = np.array(list(set(application_test_id) - set(bureau_id))
)
print('Number of application in application_test but not in bureau: ', len(application_te
st_not_in_bureau))
print('Percentage of application not in bureau: ', (len(application_test_not_in_bureau)/l
en(application_test_id))*100)
application_test_not_in_bureau_columns = ((len(set(application_test.columns) - set(bureau
.columns)))/len(set(application_test.columns)))*100
print('Percentage of columns from application that are not in bureau: ', application_test
_not_in_bureau_columns)

print('-'*100)

application_test_not_in_previous = np.array(list(set(application_test_id) - set(previous_
application_id)))
print('Number of application in application_test but not in previous: ', len(application_
test_not_in_bureau))
print('Percentage of application not in previous: ', (len(application_test_not_in_previou
s)/len(application_test_id))*100)
```

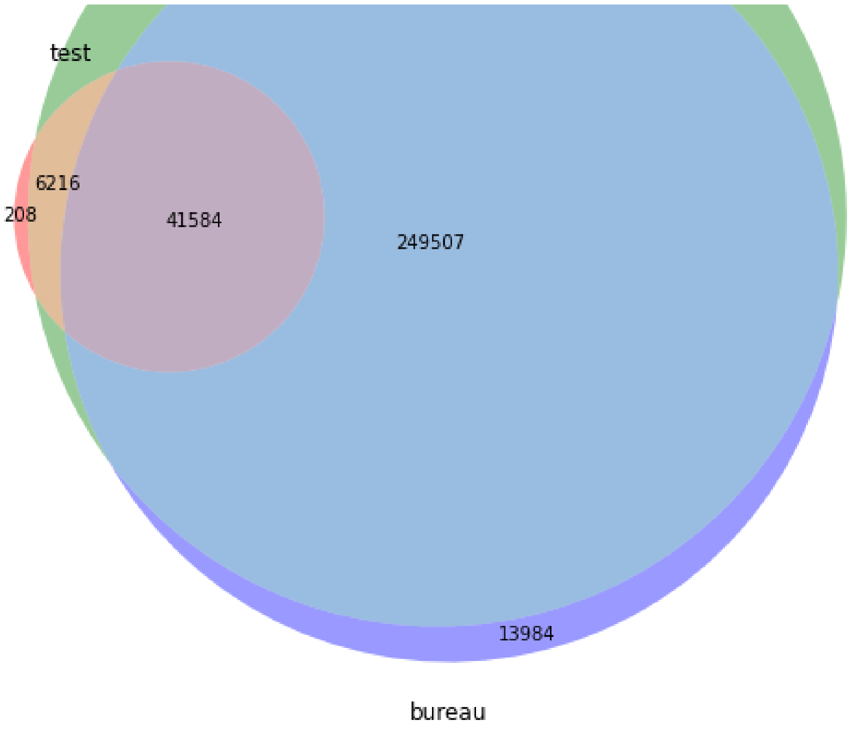
```
application_test_not_in_previous_columns = ((len(set(application_test.columns) - set(prev
ious_application.columns)))/len(set(application_test.columns)))*100
print('Percentage of columns from application that are not in previous: ', application_te
st_not_in_previous_columns)

print('-'*100)

application_test_not_in_bureau_not_in_previous = np.array(list(set(application_test_id) -
set(bureau_id) - set(previous_application_id)))
print('Number of application in application_test but neither in bureau nor in previous: '
, len(application_test_not_in_bureau_not_in_previous))
print('Percentage of application neither in bureau nor in previous: ', (len(application_t
est_not_in_bureau_not_in_previous)/len(application_test_id))*100)
application_test_in_bureau_in_previous = np.array(list(set(application_test_id) & set(bur
eau_id) & set(previous_application_id)))
print('Number of application in application_test both in bureau and in previous: ', len(a
pplication_test_in_bureau_in_previous))
print('Percentage of application in application_test both in bureau and in previous: ', (
len(application_test_in_bureau_in_previous)/len(application_test_id))*100)
```

```
Total application in application_test:  48744
Total unique application in bureau:  305811
Total unique application in previous:  338857
--------------------------------------------------------------------------------------
-------------
Number of application in application_test but not in bureau:  6424
Percentage of application not in bureau:  13.179057935335631
Percentage of columns from application that are not in bureau:  98.34710743801654
--------------------------------------------------------------------------------------
-------------
Number of application in application_test but not in previous:  6424
Percentage of application not in previous:  1.9366486131626455
Percentage of columns from application that are not in previous:  93.38842975206612
--------------------------------------------------------------------------------------
-------------
Number of application in application_test but neither in bureau nor in previous:  208
Percentage of application neither in bureau nor in previous:  0.4267191859510914
Number of application in application_test both in bureau and in previous:  41584
Percentage of application in application_test both in bureau and in previous:  85.3110126
374528
```

In [ ]:

```
'''In this code cell, venn diagram is plotted for unique entries in application_test, bur
eau and previous_application'''

test_only = len(set(application_test_id) - set(bureau_id) - set(previous_application_id))
previous_only = len(set(previous_application_id) - set(application_test_id) - set(bureau_
id))
test_previous_only = len(set(application_test_id) & set(previous_application_id) - set(bu
reau_id))
bureau_only = len(set(bureau_id) - set(application_test_id) - set(previous_application_id
))
test_bureau_only = len(set(application_test_id) & set(bureau_id) - set(previous_applicati
on_id))
previous_bureau_only = len(set(previous_application_id) & set(bureau_id) - set(applicatio
n_test_id))
all = len(set(application_test_id) & set(previous_application_id) & set(bureau_id))
plt.figure(figsize = (10,10))
venn3(subsets=(test_only, previous_only, test_previous_only, bureau_only, test_bureau_onl
y, previous_bureau_only, all), set_labels=['test', 'previous', 'bureau'])
plt.show()
```





> **It is observed that 14.31% of applications from application_train are not available in bureau. 5.35% of applications from application_train are not available in previous_application. Overall 81% of applications from application_train are available either in bureau or in previous_application. It is observed that 13.18% of applications from application_test are not available in bureau. 1.94% of applications from application_test are not available in previous_application. Overall 85.31% of applications from application_test are available either in bureau or in previous_application. It is decided that features/columns from bureau and previous_application shall be added to application_train and application_test as a part of feature engineering to get more features.**

2.3 Analysis based on missing values

In [ ]:

```
'''In this code cell, missing values shall be counted for each column in application_trai
n.'''

pd.set_option('max_rows', 400)
pd.set_option('max_colwidth', 400)
column_null_percentage = (application_train.isnull().sum()/application_train.shape[0])*1
00
print('Percentage of emply cell in each column:')
print(column_null_percentage)
column_null = [column for column in list(application_train.columns) if (application_trai
n[column].isnull().sum()/application_train.shape[0])*100 > 0]
print('Number of columns with null value: ', len(column_null))
print('Columns with null value: ', column_null)
column_no_null = [column for column in list(application_train.columns) if (application_t
rain[column].isnull().sum()/application_train.shape[0])*100 == 0]
print('Number of columns without null value: ', len(column_no_null))
print('Columns without null value: ', column_no_null)
```

```
Percentage of emply cell in each column:
SK_ID_CURR                    0.000000
TARGET                        0.000000
NAME_CONTRACT_TYPE            0.000000
CODE_GENDER                   0.000000
FLAG_OWN_CAR                  0.000000
FLAG_OWN_REALTY               0.000000
CNT_CHILDREN                  0.000000
AMT_INCOME_TOTAL              0.000000
AMT_CREDIT                    0.000000
```

```
AMT_CREDIT                      0.000000
AMT_ANNUITY                     0.003902
AMT_GOODS_PRICE                 0.090403
NAME_TYPE_SUITE                 0.420148
NAME_INCOME_TYPE                0.000000
NAME_EDUCATION_TYPE             0.000000
NAME_FAMILY_STATUS              0.000000
NAME_HOUSING_TYPE               0.000000
REGION_POPULATION_RELATIVE      0.000000
DAYS_BIRTH                      0.000000
DAYS_EMPLOYED                   0.000000
DAYS_REGISTRATION               0.000000
DAYS_ID_PUBLISH                 0.000000
OWN_CAR_AGE                    65.990810
FLAG_MOBIL                      0.000000
FLAG_EMP_PHONE                  0.000000
FLAG_WORK_PHONE                 0.000000
FLAG_CONT_MOBILE                0.000000
FLAG_PHONE                      0.000000
FLAG_EMAIL                      0.000000
OCCUPATION_TYPE                31.345545
CNT_FAM_MEMBERS                 0.000650
REGION_RATING_CLIENT            0.000000
REGION_RATING_CLIENT_W_CITY     0.000000
WEEKDAY_APPR_PROCESS_START      0.000000
HOUR_APPR_PROCESS_START         0.000000
REG_REGION_NOT_LIVE_REGION      0.000000
REG_REGION_NOT_WORK_REGION      0.000000
LIVE_REGION_NOT_WORK_REGION     0.000000
REG_CITY_NOT_LIVE_CITY          0.000000
REG_CITY_NOT_WORK_CITY          0.000000
LIVE_CITY_NOT_WORK_CITY         0.000000
ORGANIZATION_TYPE               0.000000
EXT_SOURCE_1                   56.381073
EXT_SOURCE_2                    0.214626
EXT_SOURCE_3                   19.825307
APARTMENTS_AVG                 50.749729
BASEMENTAREA_AVG               58.515956
YEARS_BEGINEXPLUATATION_AVG    48.781019
YEARS_BUILD_AVG                66.497784
COMMONAREA_AVG                 69.872297
ELEVATORS_AVG                  53.295980
ENTRANCES_AVG                  50.348768
FLOORSMAX_AVG                  49.760822
FLOORSMIN_AVG                  67.848630
LANDAREA_AVG                   59.376738
LIVINGAPARTMENTS_AVG           68.354953
LIVINGAREA_AVG                 50.193326
NONLIVINGAPARTMENTS_AVG        69.432963
NONLIVINGAREA_AVG              55.179164
APARTMENTS_MODE                50.749729
BASEMENTAREA_MODE              58.515956
YEARS_BEGINEXPLUATATION_MODE   48.781019
YEARS_BUILD_MODE               66.497784
COMMONAREA_MODE                69.872297
ELEVATORS_MODE                 53.295980
ENTRANCES_MODE                 50.348768
FLOORSMAX_MODE                 49.760822
FLOORSMIN_MODE                 67.848630
LANDAREA_MODE                  59.376738
LIVINGAPARTMENTS_MODE          68.354953
LIVINGAREA_MODE                50.193326
NONLIVINGAPARTMENTS_MODE       69.432963
NONLIVINGAREA_MODE             55.179164
APARTMENTS_MEDI                50.749729
BASEMENTAREA_MEDI              58.515956
YEARS_BEGINEXPLUATATION_MEDI   48.781019
YEARS_BUILD_MEDI               66.497784
COMMONAREA_MEDI                69.872297
ELEVATORS_MEDI                 53.295980
ENTRANCES_MEDI                 50.348768
FLOORSMAX_MEDI                 49.760822
FLOORSMIN_MEDI                 67.848630
```

```
FLOORSMIN_MEDI                67.848630
LANDAREA_MEDI                 59.376738
LIVINGAPARTMENTS_MEDI         68.354953
LIVINGAREA_MEDI               50.193326
NONLIVINGAPARTMENTS_MEDI      69.432963
NONLIVINGAREA_MEDI            55.179164
FONDKAPREMONT_MODE            68.386172
HOUSETYPE_MODE                50.176091
TOTALAREA_MODE                48.268517
WALLSMATERIAL_MODE            50.840783
EMERGENCYSTATE_MODE           47.398304
OBS_30_CNT_SOCIAL_CIRCLE       0.332021
DEF_30_CNT_SOCIAL_CIRCLE       0.332021
OBS_60_CNT_SOCIAL_CIRCLE       0.332021
DEF_60_CNT_SOCIAL_CIRCLE       0.332021
DAYS_LAST_PHONE_CHANGE         0.000325
FLAG_DOCUMENT_2                0.000000
FLAG_DOCUMENT_3                0.000000
FLAG_DOCUMENT_4                0.000000
FLAG_DOCUMENT_5                0.000000
FLAG_DOCUMENT_6                0.000000
FLAG_DOCUMENT_7                0.000000
FLAG_DOCUMENT_8                0.000000
FLAG_DOCUMENT_9                0.000000
FLAG_DOCUMENT_10               0.000000
FLAG_DOCUMENT_11               0.000000
FLAG_DOCUMENT_12               0.000000
FLAG_DOCUMENT_13               0.000000
FLAG_DOCUMENT_14               0.000000
FLAG_DOCUMENT_15               0.000000
FLAG_DOCUMENT_16               0.000000
FLAG_DOCUMENT_17               0.000000
FLAG_DOCUMENT_18               0.000000
FLAG_DOCUMENT_19               0.000000
FLAG_DOCUMENT_20               0.000000
FLAG_DOCUMENT_21               0.000000
AMT_REQ_CREDIT_BUREAU_HOUR    13.501631
AMT_REQ_CREDIT_BUREAU_DAY     13.501631
AMT_REQ_CREDIT_BUREAU_WEEK    13.501631
AMT_REQ_CREDIT_BUREAU_MON     13.501631
AMT_REQ_CREDIT_BUREAU_QRT     13.501631
AMT_REQ_CREDIT_BUREAU_YEAR    13.501631
dtype: float64
Number of columns with null value:  67
Columns with null value:  ['AMT_ANNUITY', 'AMT_GOODS_PRICE', 'NAME_TYPE_SUITE', 'OWN_CAR_
AGE', 'OCCUPATION_TYPE', 'CNT_FAM_MEMBERS', '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', 'NONL
IVINGAREA_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', 'NONLIV
INGAPARTMENTS_MODE', 'NONLIVINGAREA_MODE', 'APARTMENTS_MEDI', 'BASEMENTAREA_MEDI', 'YEARS
_BEGINEXPLUATATION_MEDI', 'YEARS_BUILD_MEDI', 'COMMONAREA_MEDI', 'ELEVATORS_MEDI', 'ENTRA
NCES_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', 'OBS_3
0_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_B
UREAU_QRT', 'AMT_REQ_CREDIT_BUREAU_YEAR']
Number of columns without null value:  55
Columns without null value:  ['SK_ID_CURR', 'TARGET', 'NAME_CONTRACT_TYPE', 'CODE_GENDER'
, 'FLAG_OWN_CAR', 'FLAG_OWN_REALTY', 'CNT_CHILDREN', 'AMT_INCOME_TOTAL', 'AMT_CREDIT', 'N
AME_INCOME_TYPE', 'NAME_EDUCATION_TYPE', 'NAME_FAMILY_STATUS', 'NAME_HOUSING_TYPE', 'REGI
ON_POPULATION_RELATIVE', 'DAYS_BIRTH', 'DAYS_EMPLOYED', 'DAYS_REGISTRATION', 'DAYS_ID_PUB
LISH', 'FLAG_MOBIL', 'FLAG_EMP_PHONE', 'FLAG_WORK_PHONE', 'FLAG_CONT_MOBILE', 'FLAG_PHONE
', 'FLAG_EMAIL', 'REGION_RATING_CLIENT', 'REGION_RATING_CLIENT_W_CITY', 'WEEKDAY_APPR_PRO
CESS_START', 'HOUR_APPR_PROCESS_START', 'REG_REGION_NOT_LIVE_REGION', 'REG_REGION_NOT_WOR
K_REGION', 'LIVE_REGION_NOT_WORK_REGION', 'REG_CITY_NOT_LIVE_CITY', 'REG_CITY_NOT_WORK_CI
TY', 'LIVE_CITY_NOT_WORK_CITY', 'ORGANIZATION_TYPE', 'FLAG_DOCUMENT_2', 'FLAG_DOCUMENT_3'
, 'FLAG_DOCUMENT_4', 'FLAG_DOCUMENT_5', 'FLAG_DOCUMENT_6', 'FLAG_DOCUMENT_7', 'FLAG_DOCUM
ENT_8', 'FLAG_DOCUMENT_9', 'FLAG_DOCUMENT_10', 'FLAG_DOCUMENT_11', 'FLAG_DOCUMENT_12', 'F
```

```
ENT_8', 'FLAG_DOCUMENT_9', 'FLAG_DOCUMENT_10', 'FLAG_DOCUMENT_11', 'FLAG_DOCUMENT_12', 'F
LAG_DOCUMENT_13', 'FLAG_DOCUMENT_14', 'FLAG_DOCUMENT_15', 'FLAG_DOCUMENT_16', 'FLAG_DOCUM
ENT_17', 'FLAG_DOCUMENT_18', 'FLAG_DOCUMENT_19', 'FLAG_DOCUMENT_20', 'FLAG_DOCUMENT_21']
```

In [ ]:

```
pd.set_option('max_rows', 400)
pd.set_option('max_colwidth', 400)
column_null_percentage = (application_test.isnull().sum()/application_test.shape[0])*100
print('Percentage of emply cell in each column:')
print(column_null_percentage)
column_null = [column for column in list(application_test.columns) if (application_test[
column].isnull().sum()/application_test.shape[0])*100 > 0]
print('Number of columns with null value: ', len(column_null))
print('Columns with null value: ', column_null)
column_no_null = [column for column in list(application_test.columns) if (application_te
st[column].isnull().sum()/application_test.shape[0])*100 == 0]
print('Number of columns without null value: ', len(column_no_null))
print('Columns without null value: ', column_no_null)
```

```
Percentage of emply cell in each column:
SK_ID_CURR                     0.000000
NAME_CONTRACT_TYPE             0.000000
CODE_GENDER                    0.000000
FLAG_OWN_CAR                   0.000000
FLAG_OWN_REALTY                0.000000
CNT_CHILDREN                   0.000000
AMT_INCOME_TOTAL               0.000000
AMT_CREDIT                     0.000000
AMT_ANNUITY                    0.049237
AMT_GOODS_PRICE                0.000000
NAME_TYPE_SUITE                1.868948
NAME_INCOME_TYPE               0.000000
NAME_EDUCATION_TYPE            0.000000
NAME_FAMILY_STATUS             0.000000
NAME_HOUSING_TYPE              0.000000
REGION_POPULATION_RELATIVE     0.000000
DAYS_BIRTH                     0.000000
DAYS_EMPLOYED                  0.000000
DAYS_REGISTRATION              0.000000
DAYS_ID_PUBLISH                0.000000
OWN_CAR_AGE                   66.289184
FLAG_MOBIL                     0.000000
FLAG_EMP_PHONE                 0.000000
FLAG_WORK_PHONE                0.000000
FLAG_CONT_MOBILE               0.000000
FLAG_PHONE                     0.000000
FLAG_EMAIL                     0.000000
OCCUPATION_TYPE               32.014197
CNT_FAM_MEMBERS                0.000000
REGION_RATING_CLIENT           0.000000
REGION_RATING_CLIENT_W_CITY    0.000000
WEEKDAY_APPR_PROCESS_START     0.000000
HOUR_APPR_PROCESS_START        0.000000
REG_REGION_NOT_LIVE_REGION     0.000000
REG_REGION_NOT_WORK_REGION     0.000000
LIVE_REGION_NOT_WORK_REGION    0.000000
REG_CITY_NOT_LIVE_CITY         0.000000
REG_CITY_NOT_WORK_CITY         0.000000
LIVE_CITY_NOT_WORK_CITY        0.000000
ORGANIZATION_TYPE              0.000000
EXT_SOURCE_1                  42.122107
EXT_SOURCE_2                   0.016412
EXT_SOURCE_3                  17.782701
APARTMENTS_AVG                49.005006
BASEMENTAREA_AVG              56.706466
YEARS_BEGINEXPLUATATION_AVG   46.889874
YEARS_BUILD_AVG               65.275726
COMMONAREA_AVG                68.716150
ELEVATORS_AVG                 51.676104
ENTRANCES_AVG                 48.373133
FLOORSMAX_AVG                 47.843837
FLOORSMIN_AVG                 66.605121
```

```
FLOORSMIN_AVG                   66.605121
LANDAREA_AVG                    57.964057
LIVINGAPARTMENTS_AVG            67.249302
LIVINGAREA_AVG                  48.317742
NONLIVINGAPARTMENTS_AVG         68.412523
NONLIVINGAREA_AVG               53.512227
APARTMENTS_MODE                 49.005006
BASEMENTAREA_MODE               56.706466
YEARS_BEGINEXPLUATATION_MODE    46.889874
YEARS_BUILD_MODE                65.275726
COMMONAREA_MODE                 68.716150
ELEVATORS_MODE                  51.676104
ENTRANCES_MODE                  48.373133
FLOORSMAX_MODE                  47.843837
FLOORSMIN_MODE                  66.605121
LANDAREA_MODE                   57.964057
LIVINGAPARTMENTS_MODE           67.249302
LIVINGAREA_MODE                 48.317742
NONLIVINGAPARTMENTS_MODE        68.412523
NONLIVINGAREA_MODE              53.512227
APARTMENTS_MEDI                 49.005006
BASEMENTAREA_MEDI               56.706466
YEARS_BEGINEXPLUATATION_MEDI    46.889874
YEARS_BUILD_MEDI                65.275726
COMMONAREA_MEDI                 68.716150
ELEVATORS_MEDI                  51.676104
ENTRANCES_MEDI                  48.373133
FLOORSMAX_MEDI                  47.843837
FLOORSMIN_MEDI                  66.605121
LANDAREA_MEDI                   57.964057
LIVINGAPARTMENTS_MEDI           67.249302
LIVINGAREA_MEDI                 48.317742
NONLIVINGAPARTMENTS_MEDI        68.412523
NONLIVINGAREA_MEDI              53.512227
FONDKAPREMONT_MODE              67.284179
HOUSETYPE_MODE                  48.455194
TOTALAREA_MODE                  46.413918
WALLSMATERIAL_MODE              49.017315
EMERGENCYSTATE_MODE             45.562531
OBS_30_CNT_SOCIAL_CIRCLE         0.059495
DEF_30_CNT_SOCIAL_CIRCLE         0.059495
OBS_60_CNT_SOCIAL_CIRCLE         0.059495
DEF_60_CNT_SOCIAL_CIRCLE         0.059495
DAYS_LAST_PHONE_CHANGE           0.000000
FLAG_DOCUMENT_2                  0.000000
FLAG_DOCUMENT_3                  0.000000
FLAG_DOCUMENT_4                  0.000000
FLAG_DOCUMENT_5                  0.000000
FLAG_DOCUMENT_6                  0.000000
FLAG_DOCUMENT_7                  0.000000
FLAG_DOCUMENT_8                  0.000000
FLAG_DOCUMENT_9                  0.000000
FLAG_DOCUMENT_10                 0.000000
FLAG_DOCUMENT_11                 0.000000
FLAG_DOCUMENT_12                 0.000000
FLAG_DOCUMENT_13                 0.000000
FLAG_DOCUMENT_14                 0.000000
FLAG_DOCUMENT_15                 0.000000
FLAG_DOCUMENT_16                 0.000000
FLAG_DOCUMENT_17                 0.000000
FLAG_DOCUMENT_18                 0.000000
FLAG_DOCUMENT_19                 0.000000
FLAG_DOCUMENT_20                 0.000000
FLAG_DOCUMENT_21                 0.000000
AMT_REQ_CREDIT_BUREAU_HOUR      12.409732
AMT_REQ_CREDIT_BUREAU_DAY       12.409732
AMT_REQ_CREDIT_BUREAU_WEEK      12.409732
AMT_REQ_CREDIT_BUREAU_MON       12.409732
AMT_REQ_CREDIT_BUREAU_QRT       12.409732
AMT_REQ_CREDIT_BUREAU_YEAR      12.409732
dtype: float64
Number of columns with null value:  64
Columns with null value:  ['AMT_ANNUITY', 'NAME_TYPE_SUITE', 'OWN_CAR_AGE', 'OCCUPATION_T
```

```
Columns with null value:  ['AMT_ANNUITY', 'NAME_TYPE_SUITE', 'OWN_CAR_AGE', 'OCCUPATION_T
YPE', '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', 'BAS
EMENTAREA_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_MOD
E', 'APARTMENTS_MEDI', 'BASEMENTAREA_MEDI', 'YEARS_BEGINEXPLUATATION_MEDI', 'YEARS_BUILD_
MEDI', 'COMMONAREA_MEDI', 'ELEVATORS_MEDI', 'ENTRANCES_MEDI', 'FLOORSMAX_MEDI', 'FLOORSMI
N_MEDI', 'LANDAREA_MEDI', 'LIVINGAPARTMENTS_MEDI', 'LIVINGAREA_MEDI', 'NONLIVINGAPARTMENT
S_MEDI', 'NONLIVINGAREA_MEDI', 'FONDKAPREMONT_MODE', 'HOUSETYPE_MODE', 'TOTALAREA_MODE',
'WALLSMATERIAL_MODE', 'EMERGENCYSTATE_MODE', 'OBS_30_CNT_SOCIAL_CIRCLE', 'DEF_30_CNT_SOCI
AL_CIRCLE', 'OBS_60_CNT_SOCIAL_CIRCLE', 'DEF_60_CNT_SOCIAL_CIRCLE', 'AMT_REQ_CREDIT_BUREA
U_HOUR', 'AMT_REQ_CREDIT_BUREAU_DAY', 'AMT_REQ_CREDIT_BUREAU_WEEK', 'AMT_REQ_CREDIT_BUREA
U_MON', 'AMT_REQ_CREDIT_BUREAU_QRT', 'AMT_REQ_CREDIT_BUREAU_YEAR']
Number of columns without null value:  57
Columns without null value:  ['SK_ID_CURR', 'NAME_CONTRACT_TYPE', 'CODE_GENDER', 'FLAG_OW
N_CAR', 'FLAG_OWN_REALTY', 'CNT_CHILDREN', 'AMT_INCOME_TOTAL', 'AMT_CREDIT', 'AMT_GOODS_P
RICE', 'NAME_INCOME_TYPE', 'NAME_EDUCATION_TYPE', 'NAME_FAMILY_STATUS', 'NAME_HOUSING_TYP
E', 'REGION_POPULATION_RELATIVE', 'DAYS_BIRTH', 'DAYS_EMPLOYED', 'DAYS_REGISTRATION', 'DA
YS_ID_PUBLISH', 'FLAG_MOBIL', 'FLAG_EMP_PHONE', 'FLAG_WORK_PHONE', 'FLAG_CONT_MOBILE', 'F
LAG_PHONE', 'FLAG_EMAIL', 'CNT_FAM_MEMBERS', 'REGION_RATING_CLIENT', 'REGION_RATING_CLIEN
T_W_CITY', 'WEEKDAY_APPR_PROCESS_START', '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', 'ORGANIZATION_TYPE', 'DAYS_LA
ST_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_DO
CUMENT_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']
```

> **It is observed that 67 columns of application_train and 64 columns of application_test have null values. In some of the columns more than 50% of the values are missing.**

3.0 Univariate and multivariate analysis

The following analysis are performed in this section:

- **Analysis of default**
- **Analysis based on family status**
- **Analysis based on contract type**
- **Analysis based on gender**
- **Analysis based on income**
- **Analysis based on income type**
- **Analysis based on education type**
- **Analysis based on occupation type**
- **Analysis based on day of the week**

3.1 Analysis of default

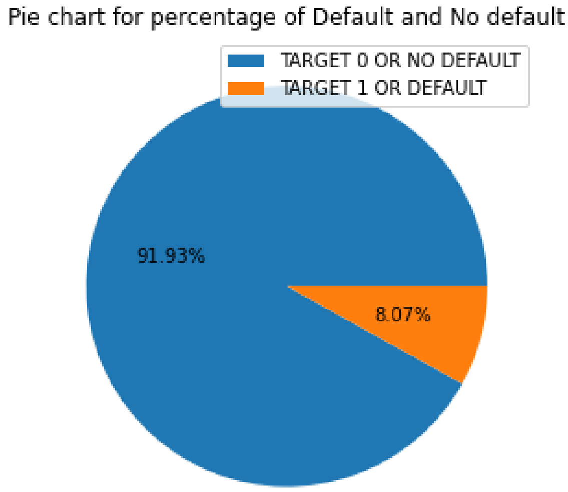
In [ ]:

```
'''This code cell gives us the percentage of loans where default has been observed and pe
rcentage of loan where no default has been observed'''

application_train_group = application_train[['SK_ID_CURR', 'TARGET']].fillna('MISSING')
category_count = (application_train_group.groupby(by = ['TARGET']).count()/application_t
rain['TARGET'].count())*100
category_name = np.unique(application_train_group['TARGET'])
#Print a table of different categories and their count
print('Table of percentage of Default represented by 1 and no default represented by 0')
print('\033[1m' + "Category".ljust(30) + "Percentage".ljust(30) + '\033[0m')
```

```
for a, b in zip(list(category_name), np.array(category_count)):
  print(str(a).ljust(30) + str('%.2f'%b).ljust(30))
pie_chart((5,5), category_count, ["TARGET 0 OR NO DEFAULT", "TARGET 1 OR DEFAULT"], 'Pie chart for percentage of Default and No default')
```

```
Table of percentage of Default represented by 1 and no default represented by 0
Category                      Percentage
0                             91.93
1                             8.07
```

```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:6: MatplotlibDeprecationWarning: Non-1D inputs to pie() are currently squeeze()d, but this behavior is deprecated since 3.1 and will be removed in 3.3; pass a 1D array instead.
```



> **An analysis of default indicates that 8.07% of applicants in application_train are defaulters. From a business perspective, this percentage needs to be reduced. From a machine learning perspective, the dataset is imbalanced.**

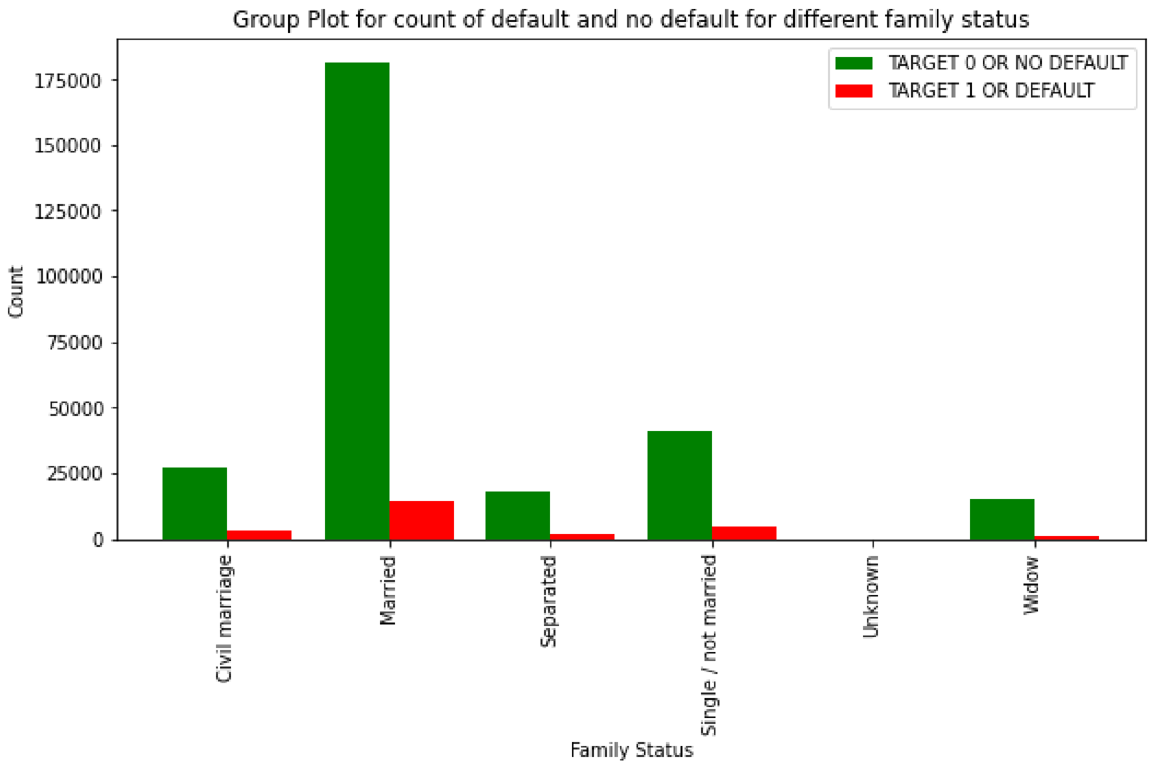
3.2 Analysis based on family status

In [ ]:

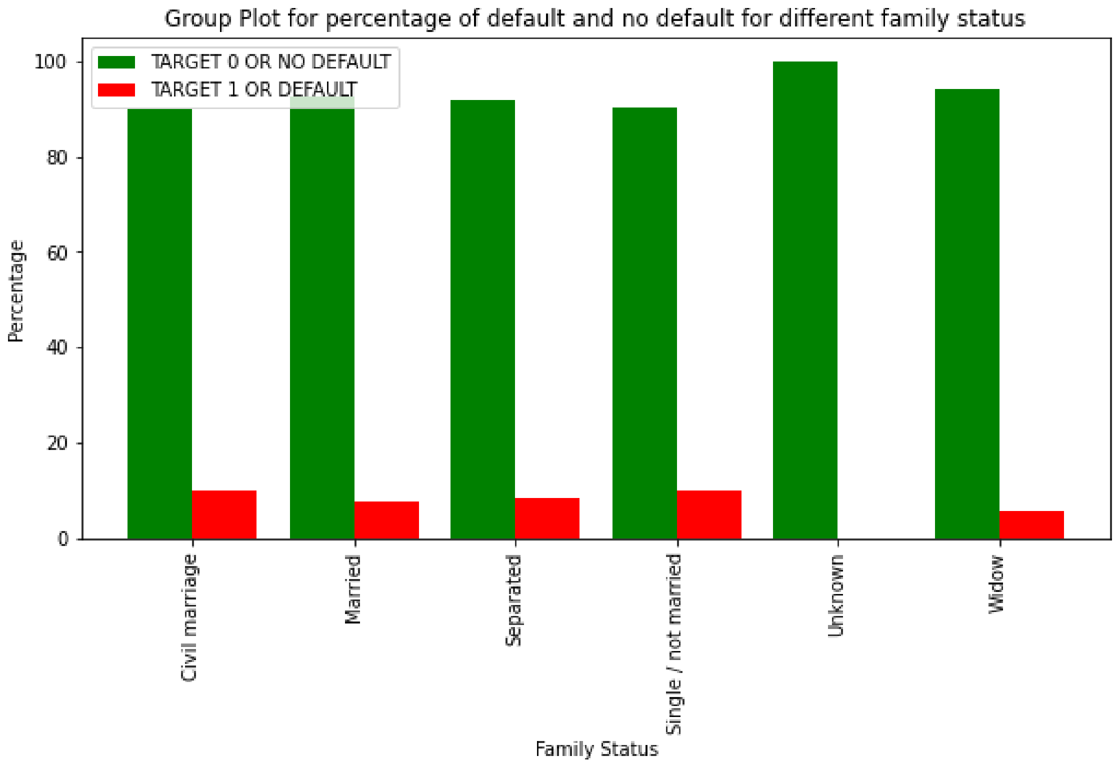
```
'''This code cell does plots graph plot for distribution (count and percentage) of default
and no default applicants with different family status'''

#Calculate different categories and their count
y_axis = application_train.pivot_table(index='NAME_FAMILY_STATUS', columns='TARGET', values='SK_ID_CURR', fill_value=0, aggfunc='count').unstack()
classes = np.unique(application_train['NAME_FAMILY_STATUS'])
#Print a table of different categories and their count
print('Table of count for diffent category')
print("Category".ljust(30) + "Count of Target 0/No Default".ljust(35) + "Count of Target 1/Default".ljust(30))
for a, b, c in zip(classes, list(y_axis[0]), list(y_axis[1])):
  print(a.ljust(30) + str(b).ljust(35) + str(c).ljust(30))
#Call group_plot function to plot group plot
group_plot((10,5), y_axis[0], y_axis[1], classes, "Family Status", "Count", ["TARGET 0 OR NO DEFAULT", "TARGET 1 OR DEFAULT"],
           "Group Plot for count of default and no default for different family status")

print("-"*100)

#Calculate percentage of different category
y_axis_0_percentage = [(y_axis[0][i]/(y_axis[0][i] + y_axis[1][i]))*100 for i in range(len(y_axis[0]))]
y_axis_1_percentage = [(y_axis[1][i]/(y_axis[0][i] + y_axis[1][i]))*100 for i in range(l
```

```
en(y_axis[0]))]
#Print a table of different categories and their percentage
print('Table of percentage for diffent category')
print('\033[1m' + "Category".ljust(30) + "% of Target 0/No Default".ljust(35) + "% of Ta
rget 1/Default".ljust(30) + '\033[0m')
for a, b, c in zip(classes, list(y_axis_0_percentage), list(y_axis_1_percentage)):
  print(a.ljust(30) + str('%.2f'%b).ljust(35) + str('%.2f'%c).ljust(30))
#Call group_plot function to plot group plot
group_plot((10,5), y_axis_0_percentage, y_axis_1_percentage, classes, "Family Status", "
Percentage", ["TARGET 0 OR NO DEFAULT", "TARGET 1 OR DEFAULT"],
           "Group Plot for percentage of default and no default for different family stat
us")
```

```
Table of count for diffent category
Category                      Count of Target 0/No Default       Count of Target 1/Defau
lt
Civil marriage                26814                              2961

Married                       181582                             14850

Separated                     18150                              1620

Single / not married          40987                              4457

Unknown                       2                                  0

Widow                         15151                              937
```




```
-----------------------------------------------------------------------------------------
-------------
Table of percentage for diffent category
Category                      % of Target 0/No Default           % of Target 1/Default

Civil marriage                90.06                              9.94

Married                       92.44                              7.56

Separated                     91.81                              8.19

Single / not married          90.19                              9.81

Unknown                       100.00                             0.00

Widow                         94.18                              5.82
```

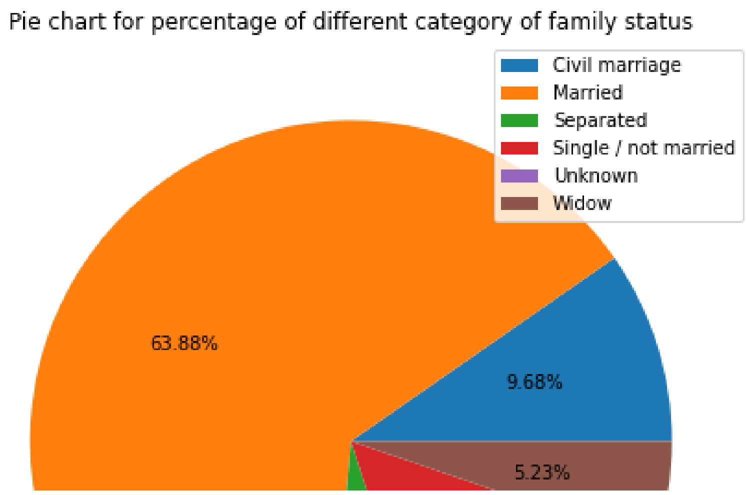


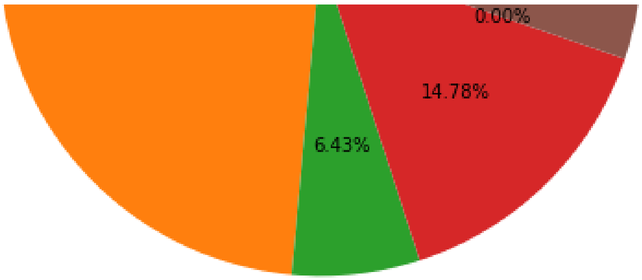
In [ ]:

```
'''In this code cell, pie chart is plotted for distribution of family status'''

application_train_group = application_train[['SK_ID_CURR', 'NAME_FAMILY_STATUS']].fillna
('MISSING')
category_count = (application_train_group.groupby(by = ['NAME_FAMILY_STATUS']).count()/a
pplication_train['NAME_FAMILY_STATUS'].count())*100
category_name = np.unique(application_train_group['NAME_FAMILY_STATUS'])
print('Table of percentage of different family status')
print('\033[1m' + "Category".ljust(30) + "Percentage".ljust(30) + '\033[0m')
for a, b in zip(list(category_name), np.array(category_count)):
  print(a.ljust(30) + str('%.2f'%b).ljust(30))
pie_chart((8,8), category_count, category_name, 'Pie chart for percentage of different ca
tegory of family status')
```

```
Table of percentage of different family status
Category                      Percentage
Civil marriage                9.68
Married                       63.88
Separated                     6.43
Single / not married          14.78
Unknown                       0.00
Widow                         5.23
```

```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:6: MatplotlibDeprecationWarn
ing: Non-1D inputs to pie() are currently squeeze()d, but this behavior is deprecated sin
ce 3.1 and will be removed in 3.3; pass a 1D array instead.
```





> **An analysis of family status of applicants and default indicates that maximum applicants are married and percentage of default is lower among married people. From a business point of view, married people should be further targeted for loans. Default among widows is also very low however, the percentage of widow applicants is low. This group can be targeted and further analysed till there is a substantial population in this group. Civil marriage and Single / not married categories are where focus should be to reduce the percentage of defaulters. The loan process may put an extra check for these groups.**

3.3 Analysis based on contract type

In [ ]:

```
'''This code cell does plots graph plot for distribution (count and percentage) of defaul
t
and no default applicants with different contract type'''

#Calculate different categories and their count
y_axis = application_train.pivot_table(index='NAME_CONTRACT_TYPE', columns='TARGET', val
ues='SK_ID_CURR', fill_value=0, aggfunc='count').unstack()
classes = np.unique(application_train['NAME_CONTRACT_TYPE'])
#Print a table of different categories and their count
print('Table of count for diffent category')
print('\033[1m' + "Category".ljust(30) + "Count of Target 0/No Default".ljust(35) + "Cou
nt of Target 1/Default".ljust(30) + '\033[0m')
for a, b, c in zip(classes, list(y_axis[0]), list(y_axis[1])):
  print(a.ljust(30) + str(b).ljust(35) + str(c).ljust(30))
#Call group_plot function to plot group plot
group_plot((5,5), y_axis[0], y_axis[1], classes, "Contract Type", "Count", ["TARGET 0 OR
NO DEFAULT", "TARGET 1 OR DEFAULT"],
           "Group Plot for count of default and no default for different types of loans")

print('-'*100)

y_axis_0_percentage = [(y_axis[0][i]/(y_axis[0][i] + y_axis[1][i]))*100 for i in range(l
en(y_axis[0]))]
y_axis_1_percentage = [(y_axis[1][i]/(y_axis[0][i] + y_axis[1][i]))*100 for i in range(l
en(y_axis[0]))]
#Print a table of different categories and their percentage
print('Table of percentage for diffent category')
print('\033[1m' + "Category".ljust(30) + "% of Target 0/No Default".ljust(35) + "% of Ta
rget 1/Default".ljust(30) + '\033[0m')
for a, b, c in zip(classes, list(y_axis_0_percentage), list(y_axis_1_percentage)):
  print(a.ljust(30) + str('%.2f'%b).ljust(35) + str('%.2f'%c).ljust(30))
#Call group_plot function to plot group plot
group_plot((5,5), y_axis_0_percentage, y_axis_1_percentage, classes, "Contract Type", "P
ercentage", ["TARGET 0 OR NO DEFAULT", "TARGET 1 OR DEFAULT"],
           "Group Plot for percentage of default and no default for different types of lo
ans")
```

```
Table of count for diffent category
Category                          Count of Target 0/No Default        Count of Target 1/Defau
lt
Cash loans                        255011                              23221
```

Revolving loans 27675 1604

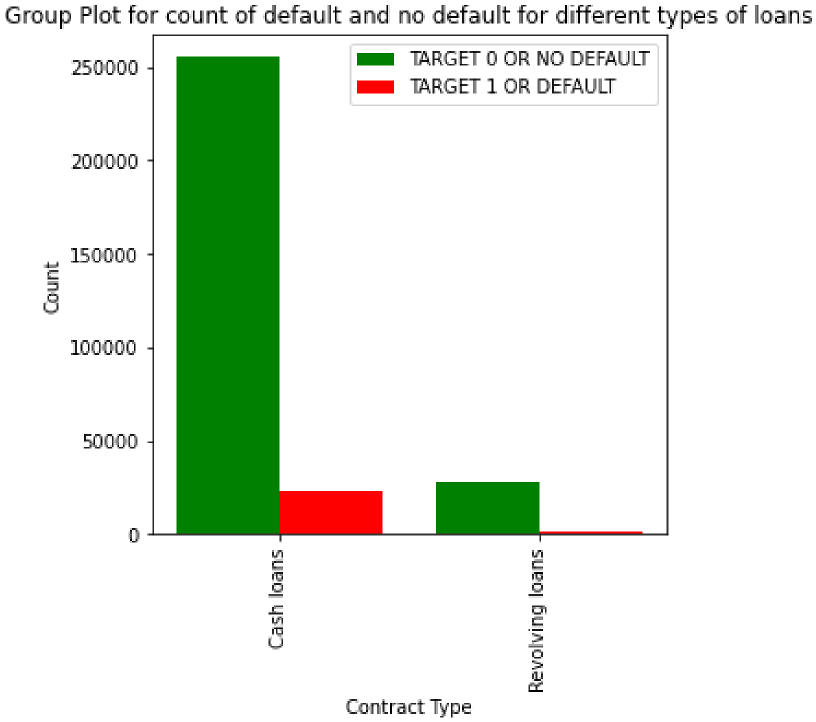
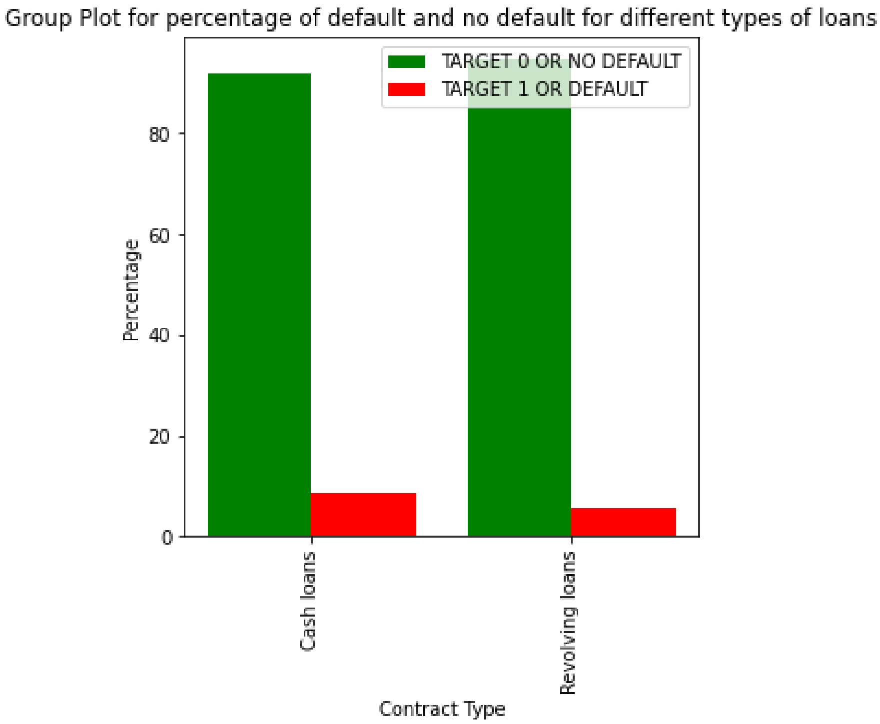


Table of percentage for diffent category

Category	% of Target 0/No Default	% of Target 1/Default
Cash loans	91.65	8.35
Revolving loans	94.52	5.48



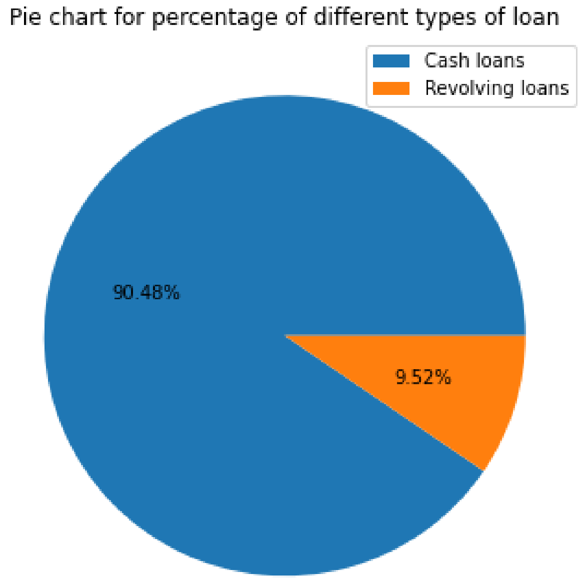
In [ ]:

```
'''In this code cell, pie chart is plotted for distribution of loan types'''

application_train_group = application_train[['SK_ID_CURR', 'NAME_CONTRACT_TYPE']].fillna
('MISSING')
category_count = (application_train_group.groupby(by = ['NAME_CONTRACT_TYPE']).count()/a
pplication_train['NAME_CONTRACT_TYPE'].count())*100
```

```
category_name = np.unique(application_train_group['NAME_CONTRACT_TYPE'])
print('Table of percentage of different types of loan')
print('\033[1m' + "Category".ljust(30) + "Percentage".ljust(30) + '\033[0m')
for a, b in zip(list(category_name), np.array(category_count)):
  print(a.ljust(30) + str('%.2f'%b).ljust(30))
pie_chart((6,6), category_count, category_name, 'Pie chart for percentage of different ty
pes of loan')
```

```
Table of percentage of different types of loan
Category                      Percentage
Cash loans                    90.48
Revolving loans               9.52
```

```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:6: MatplotlibDeprecationWarn
ing: Non-1D inputs to pie() are currently squeeze()d, but this behavior is deprecated sin
ce 3.1 and will be removed in 3.3; pass a 1D array instead.
```



> **An analysis of the type of loans and default indicates that the percentage of cash loans is much higher than revolving loans. The percentage of default is lower in case of revolving loans. Thrust should be laid on disbursing revolving loans, which generally have variable rates of interest.**

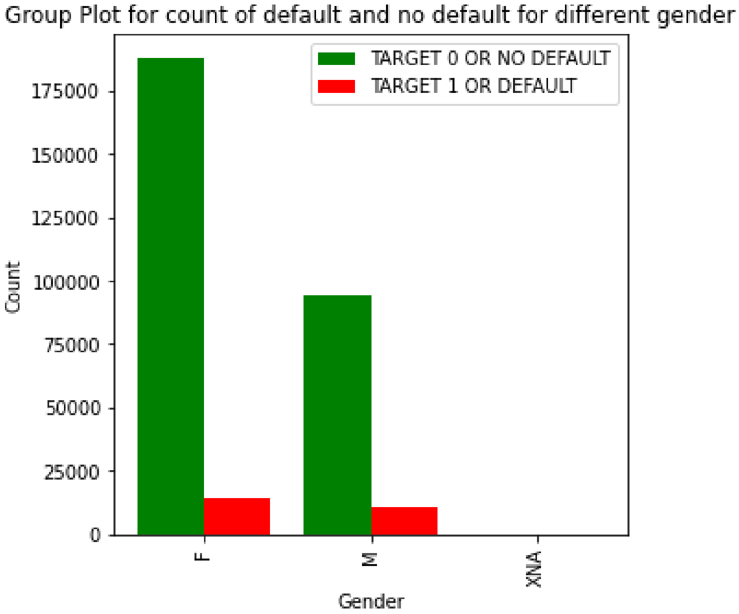
3.4 Analysis based on gender

In [ ]:

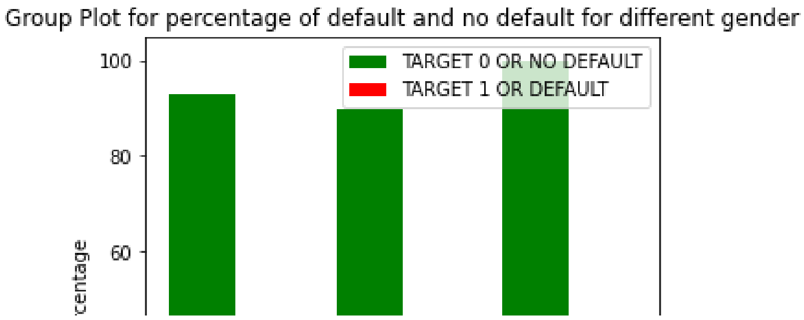
```
'''This code cell does plots graph plot for distribution (count and percentage) of defaul
t
and no default applicants for diggerent gender.'''

#Calculate different categories and their count
y_axis = application_train.pivot_table(index='CODE_GENDER', columns='TARGET', values='SK
_ID_CURR', fill_value=0, aggfunc='count').unstack()
classes = np.unique(application_train['CODE_GENDER'])
#Print a table of different categories and their count
print('Table of count for diffent category')
print("Category".ljust(30) + "Count of Target 0/No Default".ljust(35) + "Count of Target
1/Default".ljust(30))
for a, b, c in zip(classes, list(y_axis[0]), list(y_axis[1])):
  print(a.ljust(30) + str(b).ljust(35) + str(c).ljust(30))
#Call group_plot function to plot group plot
group_plot((5,5), y_axis[0], y_axis[1], classes, "Gender", "Count", ["TARGET 0 OR NO DEF
AULT", "TARGET 1 OR DEFAULT"],
           "Group Plot for count of default and no default for different gender")
```

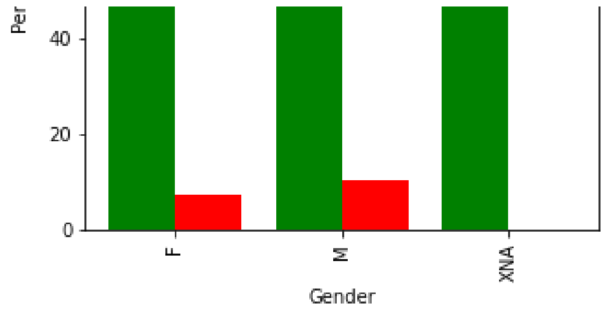
```
print('-'*100)

#Calculate percentage of different category
y_axis_0_percentage = [(y_axis[0][i]/(y_axis[0][i] + y_axis[1][i]))*100 for i in range(l
en(y_axis[0]))]
y_axis_1_percentage = [(y_axis[1][i]/(y_axis[0][i] + y_axis[1][i]))*100 for i in range(l
en(y_axis[0]))]
#Print a table of different categories and their percentage
print('Table of percentage for diffent category')
print('\033[1m' + "Category".ljust(30) + "% of Target 0/No Default".ljust(35) + "% of Ta
rget 1/Default".ljust(30) + '\033[0m')
for a, b, c in zip(classes, list(y_axis_0_percentage), list(y_axis_1_percentage)):
  print(a.ljust(30) + str('%.2f'%b).ljust(35) + str('%.2f'%c).ljust(30))
#Call group_plot function to plot group plot
group_plot((5,5), y_axis_0_percentage, y_axis_1_percentage, classes, "Gender", "Percenta
ge", ["TARGET 0 OR NO DEFAULT", "TARGET 1 OR DEFAULT"],
           "Group Plot for percentage of default and no default for different gender")
```

```
Table of count for diffent category
Category                       Count of Target 0/No Default       Count of Target 1/Defau
lt
F                              188278                             14170

M                              94404                              10655

XNA                            4                                  0
```




```
----------------------------------------------------------------------------------------
-------------
Table of percentage for diffent category
Category                       % of Target 0/No Default           % of Target 1/Default

F                              93.00                              7.00

M                              89.86                              10.14

XNA                            100.00                             0.00
```



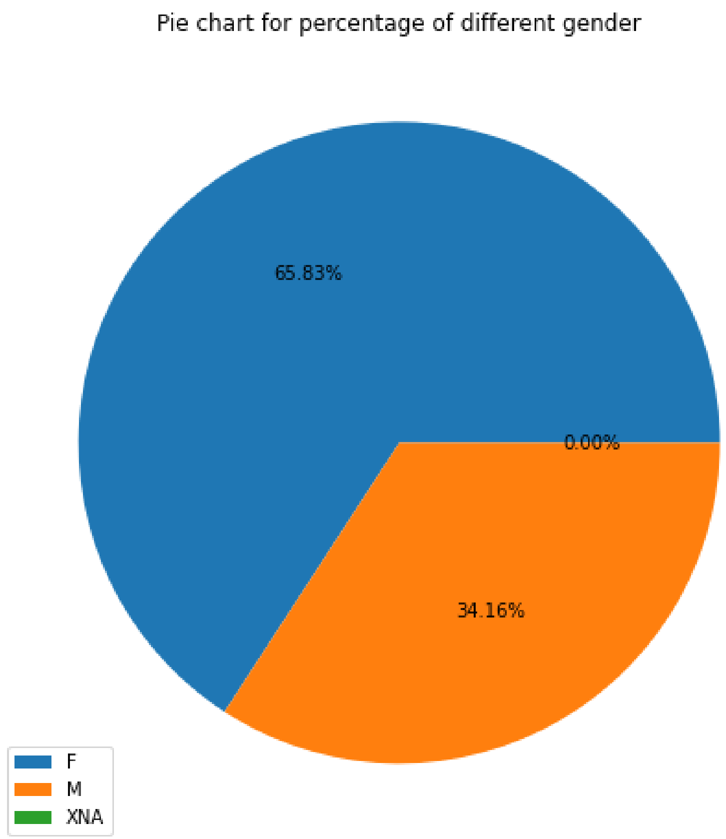


In [ ]:

```
'''In this code cell, pie chart is plotted for distribution of gender'''

application_train_group = application_train[['SK_ID_CURR', 'CODE_GENDER']].fillna('MISSI
NG')
category_count = (application_train_group.groupby(by = ['CODE_GENDER']).count()/applicat
ion_train['CODE_GENDER'].count())*100
category_name = np.unique(application_train_group['CODE_GENDER'])
print('Table of percentage of different gender')
print("Category".ljust(30) + "Percentage".ljust(30))
for a, b in zip(list(category_name), np.array(category_count)):
  print(a.ljust(30) + str('%.2f'%b).ljust(30))
pie_chart((8,8), category_count, category_name, 'Pie chart for percentage of different ge
nder')
```

```
Table of percentage of different gender
Category                      Percentage
F                             65.83
M                             34.16
XNA                           0.00
```

```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:6: MatplotlibDeprecationWarn
ing: Non-1D inputs to pie() are currently squeeze()d, but this behavior is deprecated sin
ce 3.1 and will be removed in 3.3; pass a 1D array instead.
```



> **An analysis based on gender and default indicates that the number of male applicants is double the number of female applicants. This result also has a social connotation apart from business impact. It can be concluded that more number of male are earning compared to females and a**

gender disparity exists. Policy makers will have the responsibility to remove this disparity or evaluate their steps taken earlier. As the percentage of female defaulters is lesser than male defaulters, this calls for promotion of loan among females. From this preliminary analysis, it seems that gender is an important parameter. However, the decision to retain or remove this parameter shall be taken after mathematical analysis of data in the feature selection section.

3.5 Analysis based on income

In [ ]:

```
'''In this code cell, data under column titled AMT_INCOME_TOTAL is grouped into 10 bins b
ased
upon 10 quantitle. A new column - income_bins - is created in application_train which ind
icates
respective bins for entries in AMT_INCOME_TOTAL'''

application_train['income_bins'] = pd.qcut(application_train['AMT_INCOME_TOTAL'], q=10)
```

In [ ]:

```
'''This code cell does plots graph plot for distribution (count and percentage) of defaul
t
and no default applicants with different income groups.'''

#Calculate different categories and their count
y_axis = application_train.pivot_table(index='income_bins', columns='TARGET', values='SK
_ID_CURR', fill_value=0, aggfunc='count').unstack()
classes = np.unique(application_train['income_bins'])
#Print a table of different categories and their count
print('Table of count for diffent category')
print("Category".ljust(30) + "Count of Target 0/No Default".ljust(35) + "Count of Target
1/Default".ljust(30))
for a, b, c in zip(classes, list(y_axis[0]), list(y_axis[1])):
  print(str(a).ljust(30) + str(b).ljust(35) + str(c).ljust(30))
#Call group_plot function to plot group plot
group_plot((15,5), y_axis[0], y_axis[1], classes, "Income group", "Count", ["TARGET 0 OR
NO DEFAULT", "TARGET 1 OR DEFAULT"],
           "Group Plot for count of default and no default for different income groups")

print('-'*100)

#Calculate percentage of different category
y_axis_0_percentage = [(y_axis[0][i]/(y_axis[0][i] + y_axis[1][i]))*100 for i in range(l
en(y_axis[0]))]
y_axis_1_percentage = [(y_axis[1][i]/(y_axis[0][i] + y_axis[1][i]))*100 for i in range(l
en(y_axis[0]))]
#Print a table of different categories and their percentage
print('Table of percentage for diffent category')
print("Category".ljust(30) + "% of Target 0/No Default".ljust(35) + "% of Target 1/Defau
lt".ljust(30))
for a, b, c in zip(classes, list(y_axis_0_percentage), list(y_axis_1_percentage)):
  print(str(a).ljust(30) + str('%.2f'%b).ljust(35) + str('%.2f'%c).ljust(30))
#Call group_plot function to plot group plot
group_plot((15,5), y_axis_0_percentage, y_axis_1_percentage, classes, "Income group", "P
ercentage", ["TARGET 0 OR NO DEFAULT", "TARGET 1 OR DEFAULT"],
           "Group Plot for percentage of default and no default for different income grou
ps")
```

```
Table of count for diffent category
Category                      Count of Target 0/No Default       Count of Target 1/Defau
lt
(25649.999, 81000.0]          30656                              2735

(81000.0, 99000.0]            27790                              2490

(99000.0, 112500.0]           33689                              3218

(112500.0, 135000.0]          44702                              4147
```

(135000.0, 147150.0]	3943	390
(147150.0, 162000.0]	28431	2689
(162000.0, 180000.0]	28118	2586
(180000.0, 225000.0]	41311	3498
(225000.0, 270000.0]	18552	1405
(270000.0, 117000000.0]	25494	1667

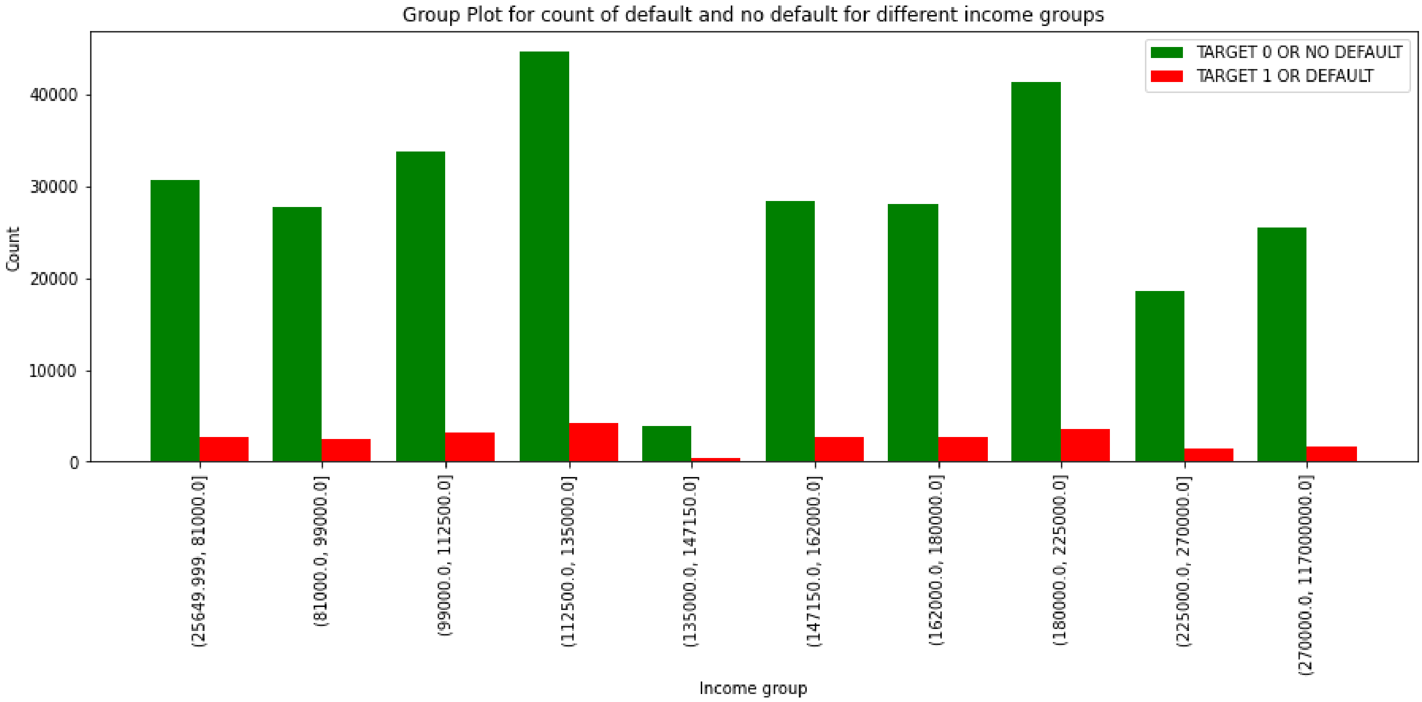
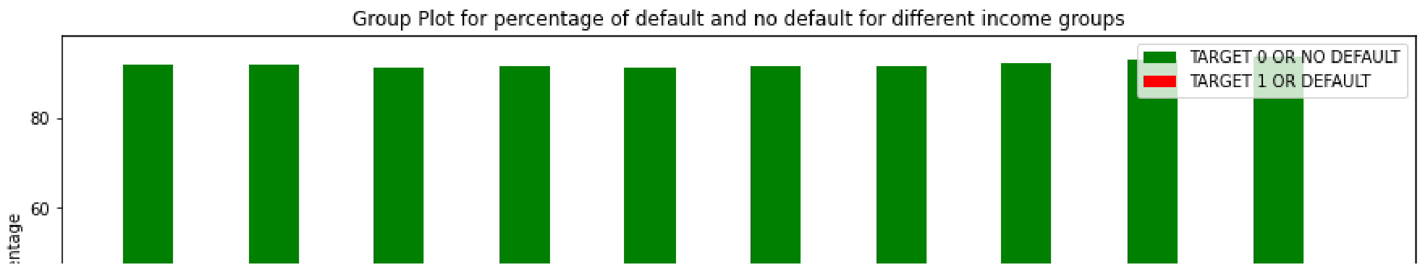
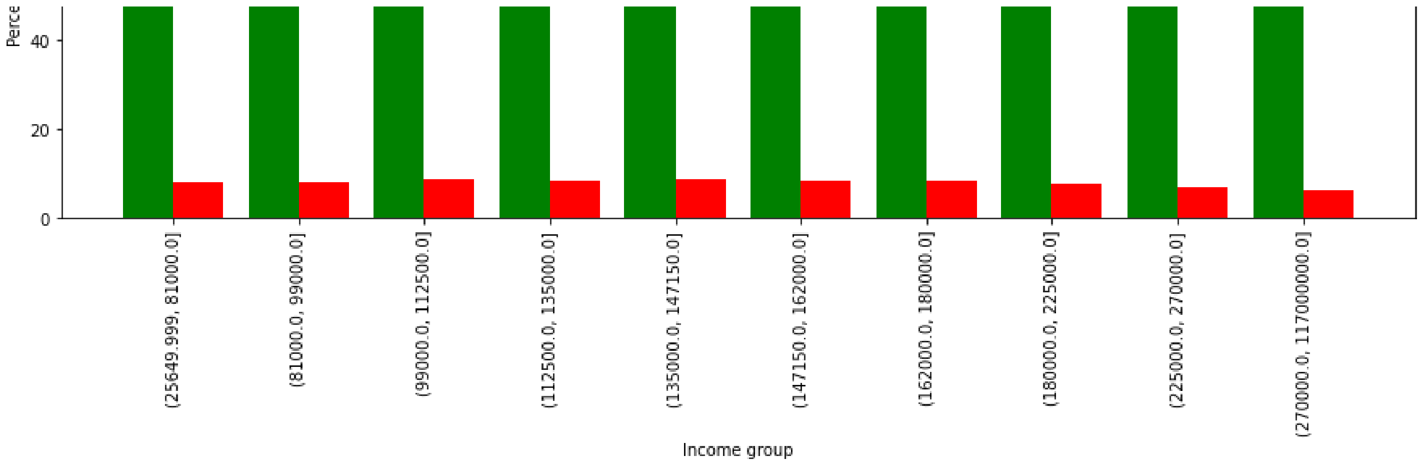


Table of percentage for diffent category

Category	% of Target 0/No Default	% of Target 1/Default
(25649.999, 81000.0]	91.81	8.19
(81000.0, 99000.0]	91.78	8.22
(99000.0, 112500.0]	91.28	8.72
(112500.0, 135000.0]	91.51	8.49
(135000.0, 147150.0]	91.00	9.00
(147150.0, 162000.0]	91.36	8.64
(162000.0, 180000.0]	91.58	8.42
(180000.0, 225000.0]	92.19	7.81
(225000.0, 270000.0]	92.96	7.04
(270000.0, 117000000.0]	93.86	6.14



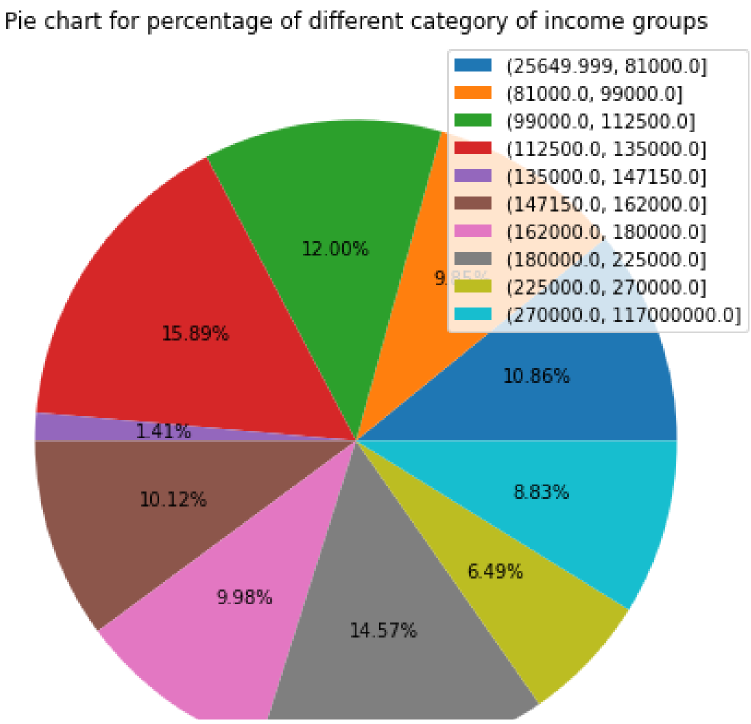


In [ ]:

```
'''In this code cell, pie chart is plotted for distribution of income groups'''

application_train_group = application_train[['SK_ID_CURR', 'income_bins']]
category_count = (application_train_group.groupby(by = ['income_bins']).count()/applicat
ion_train['income_bins'].count())*100
category_name = np.unique(application_train_group['income_bins'])
print('Table of percentage of different income groups')
print("Category".ljust(30) + "Percentage".ljust(30))
for a, b in zip(list(category_name), np.array(category_count)):
  print(str(a).ljust(30) + str('%.2f'%b).ljust(30))
pie_chart((8,8), category_count, category_name, 'Pie chart for percentage of different ca
tegory of income groups')
```

```
Table of percentage of different income groups
Category                      Percentage
(25649.999, 81000.0]          10.86
(81000.0, 99000.0]            9.85
(99000.0, 112500.0]           12.00
(112500.0, 135000.0]          15.89
(135000.0, 147150.0]          1.41
(147150.0, 162000.0]          10.12
(162000.0, 180000.0]          9.98
(180000.0, 225000.0]          14.57
(225000.0, 270000.0]          6.49
(270000.0, 117000000.0]       8.83
```

```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:6: MatplotlibDeprecationWarn
ing: Non-1D inputs to pie() are currently squeeze()d, but this behavior is deprecated sin
ce 3.1 and will be removed in 3.3; pass a 1D array instead.
```



> **There is not much significant disparity among different income groups except for 135000.0-147150.0 where loan applicants are very low. Loan applicants are more in lower income groups compared to higher income groups. Loan defaulters are also leeser in higher income groups compared to lower income groups. Two particular income groups with higher percentage of applicants are 112500.0-135000.0 and 180000.0-225000.0. This data also indicates a few inordinately high income applicants. These data points may be outliers which will be dealt separately in the outlier detection and removal section.**

3.6 Analysis based on income type

In [ ]:

```
'''This code cell does plots graph plot for distribution (count and percentage) of defaul
t
and no default applicants with different income type'''

#Calculate different categories and their count
y_axis = application_train.pivot_table(index='NAME_INCOME_TYPE', columns='TARGET', value
s='SK_ID_CURR', fill_value=0, aggfunc='count').unstack()
classes = np.unique(application_train['NAME_INCOME_TYPE'])
#Print a table of different categories and their count
print('Table of count for diffent category')
print("Category".ljust(30) + "Count of Target 0/No Default".ljust(35) + "Count of Target
1/Default".ljust(30))
for a, b, c in zip(classes, list(y_axis[0]), list(y_axis[1])):
  print(a.ljust(30) + str(b).ljust(35) + str(c).ljust(30))
#Call group_plot function to plot group plot
group_plot((10,5), y_axis[0], y_axis[1], classes, "Income type", "Count", ["TARGET 0 OR
NO DEFAULT", "TARGET 1 OR DEFAULT"],
           "Group Plot for count of default and no default for different income type")

print("-"*100)

#Calculate percentage of different category
y_axis_0_percentage = [(y_axis[0][i]/(y_axis[0][i] + y_axis[1][i]))*100 for i in range(l
en(y_axis[0]))]
y_axis_1_percentage = [(y_axis[1][i]/(y_axis[0][i] + y_axis[1][i]))*100 for i in range(l
en(y_axis[0]))]
#Print a table of different categories and their percentage
print('Table of percentage for diffent category')
print("Category".ljust(30) + "% of Target 0/No Default".ljust(35) + "% of Target 1/Defau
lt".ljust(30))
for a, b, c in zip(classes, list(y_axis_0_percentage), list(y_axis_1_percentage)):
  print(a.ljust(30) + str('%.2f'%b).ljust(35) + str('%.2f'%c).ljust(30))
#Call group_plot function to plot group plot
group_plot((10,5), y_axis_0_percentage, y_axis_1_percentage, classes, "Income type", "Pe
rcentage", ["TARGET 0 OR NO DEFAULT", "TARGET 1 OR DEFAULT"],
           "Group Plot for percentage of default and no default for different income type
")
```

```
Table of count for diffent category
Category                      Count of Target 0/No Default       Count of Target 1/Defau
lt
Businessman                   10                                 0

Commercial associate          66257                              5360

Maternity leave               3                                  2

Pensioner                     52380                              2982
```

State servant	20454	1249
Student	18	0
Unemployed	14	8
Working	143550	15224

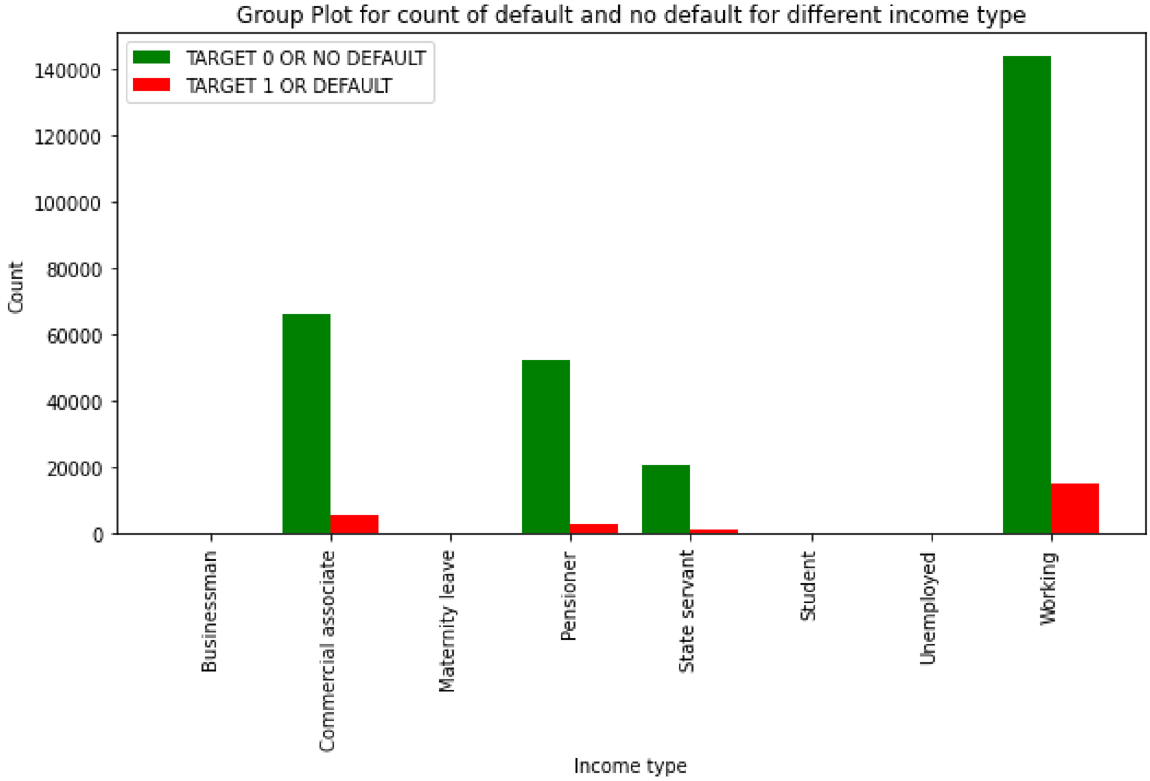
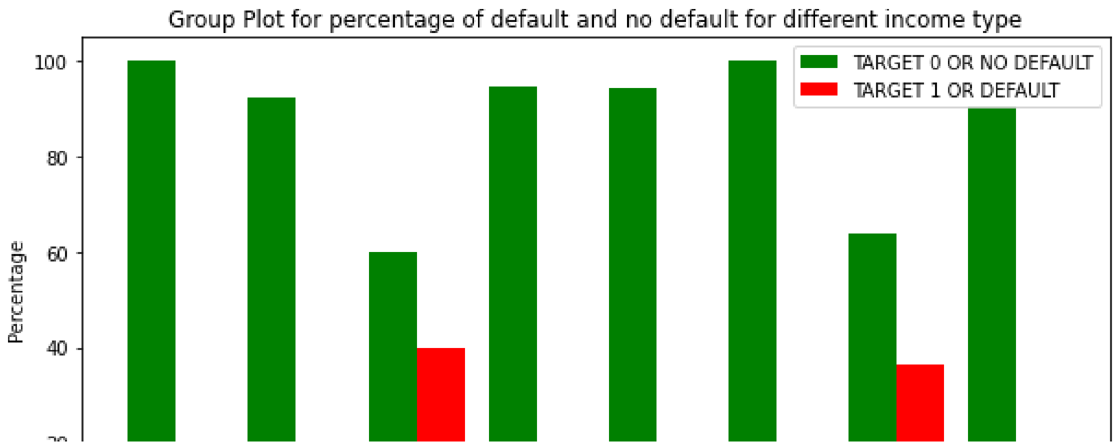
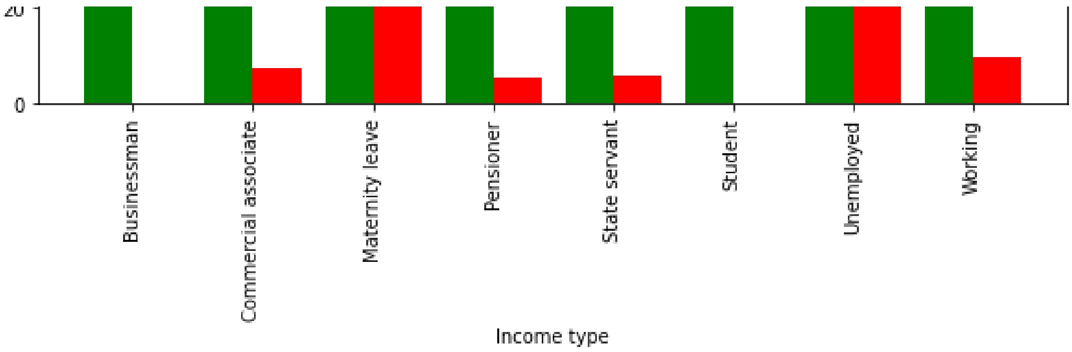


Table of percentage for diffent category

Category	% of Target 0/No Default	% of Target 1/Default
Businessman	100.00	0.00
Commercial associate	92.52	7.48
Maternity leave	60.00	40.00
Pensioner	94.61	5.39
State servant	94.25	5.75
Student	100.00	0.00
Unemployed	63.64	36.36
Working	90.41	9.59



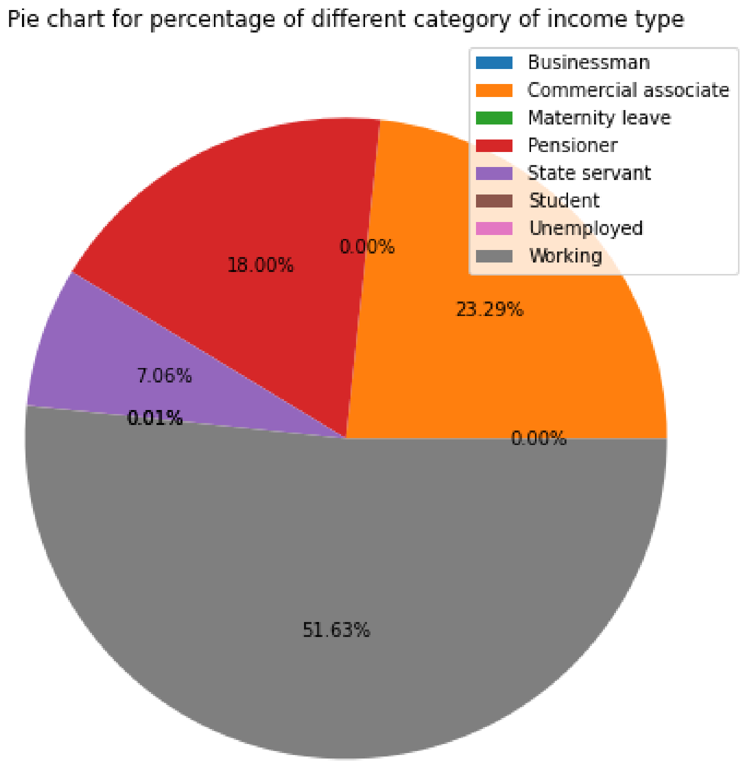


In [ ]:

```
'''In this code cell, pie chart is plotted for distribution of income type'''

application_train_group = application_train[['SK_ID_CURR', 'NAME_INCOME_TYPE']].fillna('
MISSING')
category_count = (application_train_group.groupby(by = ['NAME_INCOME_TYPE']).count()/app
lication_train['NAME_INCOME_TYPE'].count())*100
category_name = np.unique(application_train_group['NAME_INCOME_TYPE'])
print('Table of percentage of different income type')
print("Category".ljust(30) + "Percentage".ljust(30))
for a, b in zip(list(category_name), np.array(category_count)):
  print(a.ljust(30) + str('%.2f'%b).ljust(30))
pie_chart((8,8), category_count, category_name, 'Pie chart for percentage of different ca
tegory of income type')
```

```
Table of percentage of different income type
Category                      Percentage
Businessman                   0.00
Commercial associate          23.29
Maternity leave               0.00
Pensioner                     18.00
State servant                 7.06
Student                       0.01
Unemployed                    0.01
Working                       51.63
```

```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:6: MatplotlibDeprecationWarn
ing: Non-1D inputs to pie() are currently squeeze()d, but this behavior is deprecated sin
ce 3.1 and will be removed in 3.3; pass a 1D array instead.
```



> **An analysis based on income type indicates huge disparity among the groups with different income types. The default rate is very low among pensioners and government servants. They become a target group for promotion of loans and further evaluation of results. Groups with lower numbers of applicants can be targeted for loans and results shall be evaluated after some time. From this preliminary analysis, it seems that income type is an important parameter. However, the decision to retain or remove this parameter shall be taken after mathematical analysis of data in the feature selection section.**

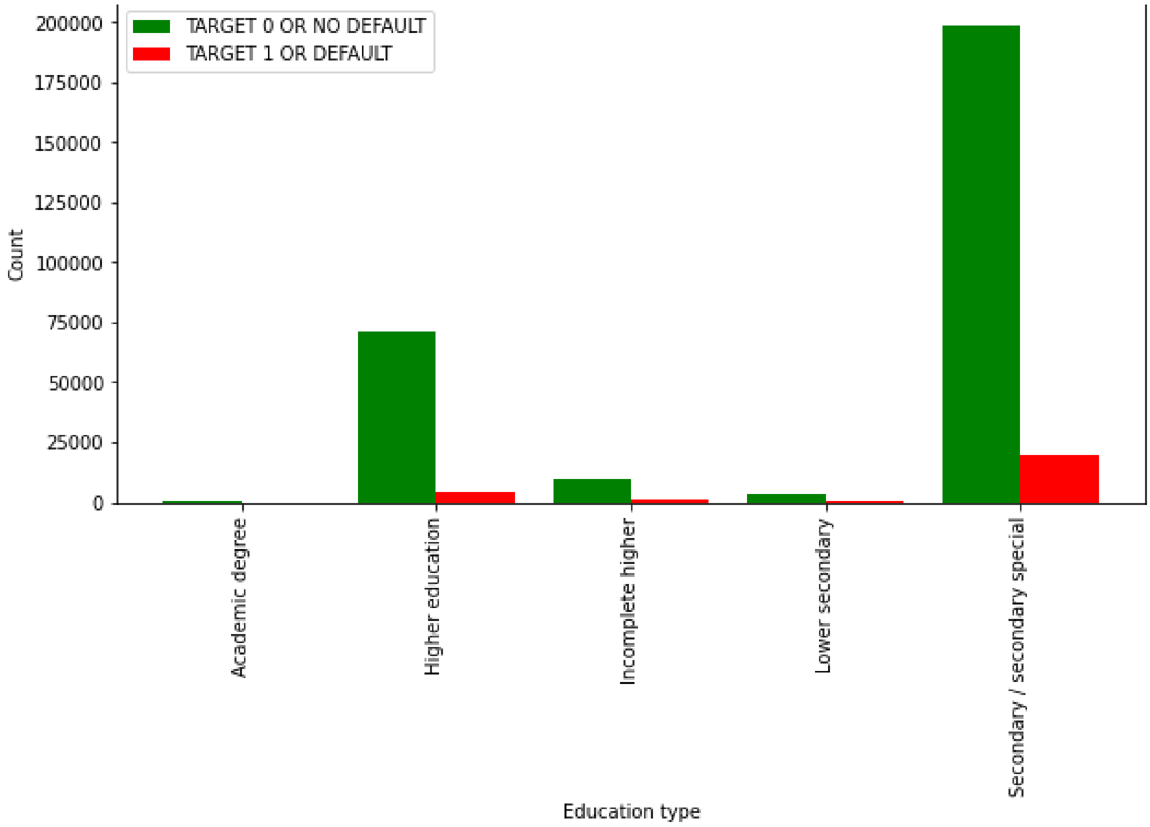
3.7 Analysis based on education type

In [ ]:

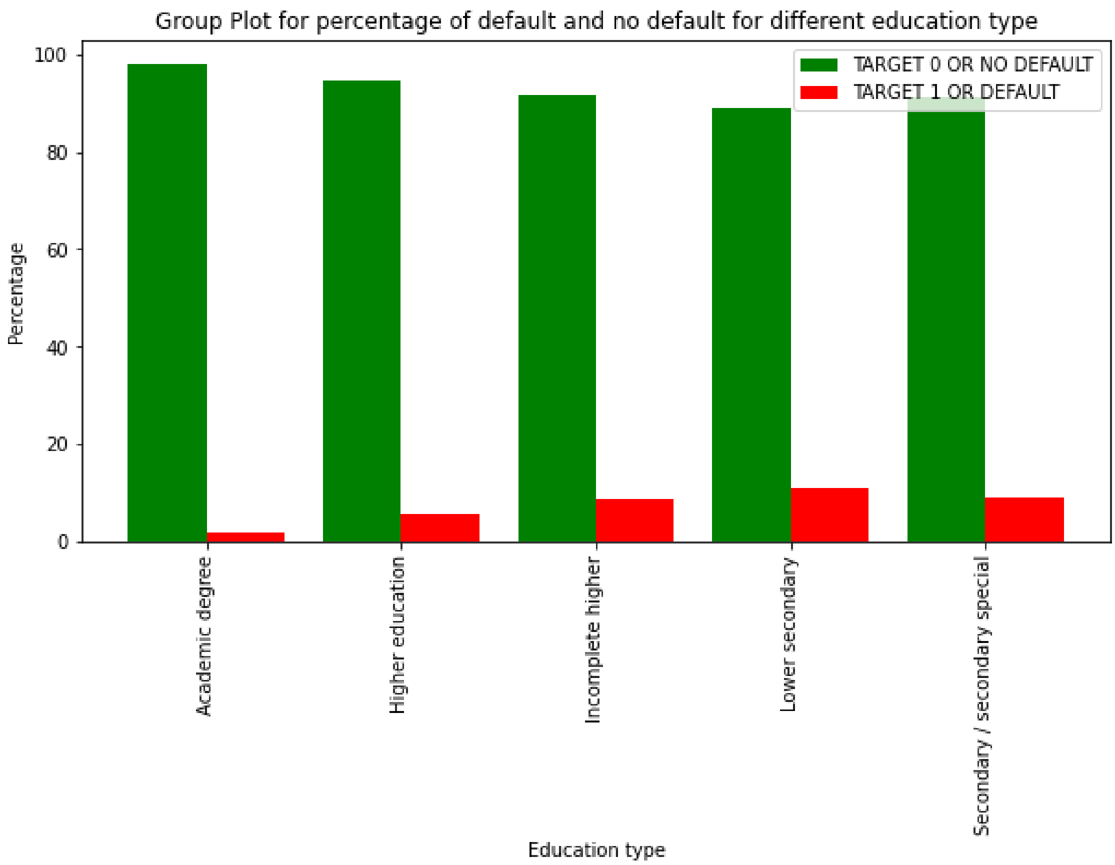
```
'''This code cell does plots graph plot for distribution (count and percentage) of defaul
t
and no default applicants with different education type'''

#Calculate different categories and their count
y_axis = application_train.pivot_table(index='NAME_EDUCATION_TYPE', columns='TARGET', va
lues='SK_ID_CURR', fill_value=0, aggfunc='count').unstack()
classes = np.unique(application_train['NAME_EDUCATION_TYPE'])
#Print a table of different categories and their count
print('Table of count for diffent category')
print("Category".ljust(30) + "Count of Target 0/No Default".ljust(35) + "Count of Target
1/Default".ljust(30))
for a, b, c in zip(classes, list(y_axis[0]), list(y_axis[1])):
  print(a.ljust(30) + str(b).ljust(35) + str(c).ljust(30))
#Call group_plot function to plot group plot
group_plot((10,5), y_axis[0], y_axis[1], classes, "Education type", "Count", ["TARGET 0
OR NO DEFAULT", "TARGET 1 OR DEFAULT"],
           "Group Plot for count of default and no default for different education type")

print("-"*100)

#Print a table of different categories and their percentage
y_axis_0_percentage = [(y_axis[0][i]/(y_axis[0][i] + y_axis[1][i]))*100 for i in range(l
en(y_axis[0]))]
y_axis_1_percentage = [(y_axis[1][i]/(y_axis[0][i] + y_axis[1][i]))*100 for i in range(l
en(y_axis[0]))]
#Print a table of different categories and their percentage
print('Table of percentage for diffent category')
print("Category".ljust(30) + "% of Target 0/No Default".ljust(35) + "% of Target 1/Defau
lt".ljust(30))
for a, b, c in zip(classes, list(y_axis_0_percentage), list(y_axis_1_percentage)):
  print(a.ljust(30) + str('%.2f'%b).ljust(35) + str('%.2f'%c).ljust(30))
#Call group_plot function to plot group plot
group_plot((10,5), y_axis_0_percentage, y_axis_1_percentage, classes, "Education type",
"Percentage", ["TARGET 0 OR NO DEFAULT", "TARGET 1 OR DEFAULT"],
           "Group Plot for percentage of default and no default for different education t
ype")
```

```
Table of count for diffent category
Category                      Count of Target 0/No Default       Count of Target 1/Defau
lt
Academic degree               161                                3

Higher education              70854                              4009

Incomplete higher             9405                               872

Lower secondary               3399                               417

Secondary / secondary special 198867                             19524
```

Group Plot for count of default and no default for different education type




```
---------------------------------------------------------------------------------------------
Table of percentage for diffent category
Category                      % of Target 0/No Default          % of Target 1/Default

Academic degree               98.17                             1.83

Higher education              94.64                             5.36

Incomplete higher             91.52                             8.48

Lower secondary               89.07                             10.93

Secondary / secondary special 91.06                             8.94
```

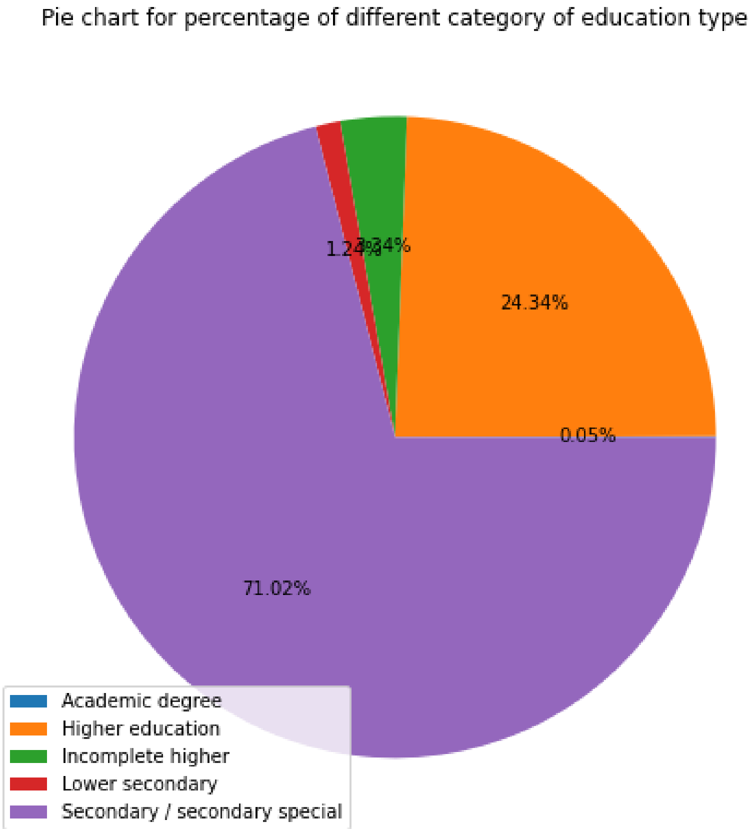


In [ ]:

```
'''In this code cell, pie chart is plotted for distribution of education type'''

application_train_group = application_train[['SK_ID_CURR', 'NAME_EDUCATION_TYPE']].filln
a('MISSING')
category_count = (application_train_group.groupby(by = ['NAME_EDUCATION_TYPE']).count()/
application_train['NAME_EDUCATION_TYPE'].count())*100
category_name = np.unique(application_train_group['NAME_EDUCATION_TYPE'])
print('Table of percentage of different education type')
print("Category".ljust(30) + "Percentage".ljust(30))
for a, b in zip(list(category_name), np.array(category_count)):
  print(a.ljust(30) + str('%.2f'%b).ljust(30))
pie_chart((8,8), category_count, category_name, 'Pie chart for percentage of different ca
tegory of education type')
```

```
Table of percentage of different education type
Category                      Percentage
Academic degree               0.05
Higher education              24.34
Incomplete higher             3.34
Lower secondary               1.24
Secondary / secondary special 71.02
```

```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:6: MatplotlibDeprecationWarn
ing: Non-1D inputs to pie() are currently squeeze()d, but this behavior is deprecated sin
ce 3.1 and will be removed in 3.3; pass a 1D array instead.
```



> **An analysis based on education type indicates huge disparity among the groups with different education types. It is observed that as the level of education increases, the percentage of default decreases. Education is in general correlated to income. Hence, this observation suggests that groups with higher levels of education should be targeted for loans. From this preliminary analysis, it seems that education is an important parameter. However, the decision to retain or remove this parameter shall be taken after mathematical analysis of data in the feature selection section.**

3.8 Analysis based on occupation type

In [ ]:

```
'''This code cell does plots graph plot for distribution (count and percentage) of defaul
t
and no default applicants with different occupation type'''

#Calculate different categories and their count
df = application_train[['SK_ID_CURR', 'TARGET', 'OCCUPATION_TYPE']].fillna('MISSING DATA
')
y_axis = df.pivot_table(index='OCCUPATION_TYPE', columns='TARGET', values='SK_ID_CURR',
fill_value=0, aggfunc='count').unstack()
classes = np.unique(df['OCCUPATION_TYPE'])
#Print a table of different categories and their count
print('Table of count for diffent category')
print("Category".ljust(30) + "Count of Target 0/No Default".ljust(35) + "Count of Target
1/Default".ljust(30))
for a, b, c in zip(classes, list(y_axis[0]), list(y_axis[1])):
  print(a.ljust(30) + str(b).ljust(35) + str(c).ljust(30))
#Call group_plot function to plot group plot
group_plot((15,5), y_axis[0], y_axis[1], classes, "Occupation type", "Count", ["TARGET 0
OR NO DEFAULT", "TARGET 1 OR DEFAULT"],
           "Group Plot for count of default and no default for different occupation type"
)

print("-"*100)

#Calculate percentage of different category
y_axis_0_percentage = [(y_axis[0][i]/(y_axis[0][i] + y_axis[1][i]))*100 for i in range(l
en(y_axis[0]))]
y_axis_1_percentage = [(y_axis[1][i]/(y_axis[0][i] + y_axis[1][i]))*100 for i in range(l
en(y_axis[0]))]
#Print a table of different categories and their percentage
print('Table of percentage for diffent category')
print("Category".ljust(30) + "% of Target 0/No Default".ljust(35) + "% of Target 1/Defau
lt".ljust(30))
for a, b, c in zip(classes, list(y_axis_0_percentage), list(y_axis_1_percentage)):
  print(a.ljust(30) + str('%.2f'%b).ljust(35) + str('%.2f'%c).ljust(30))
#Call group_plot function to plot group plot
group_plot((15,5), y_axis_0_percentage, y_axis_1_percentage, classes, "Occupation type",
"Percentage", ["TARGET 0 OR NO DEFAULT", "TARGET 1 OR DEFAULT"],
           "Group Plot for percentage of default and no default for different occupation
type")
```

```
Table of count for diffent category
Category                      Count of Target 0/No Default       Count of Target 1/Defau
lt
Accountants                   9339                               474

Cleaning staff                4206                               447

Cooking staff                 5325                               621

Core staff                    25832                              1738

Drivers                       16496                              2107

HR staff                      527                                36

High skill tech staff         10679                              701

IT staff                      492                                34

Laborers                      49348                              5838

Low-skill Laborers            1734                               359

MISSING DATA                  90113                              6278

Managers                      20043                              1328

Medicine staff                7965                               572
```

Private service staff	2477	175
Realty agents	692	59
Sales staff	29010	3092
Secretaries	1213	92
Security staff	5999	722
Waiters/barmen staff	1196	152

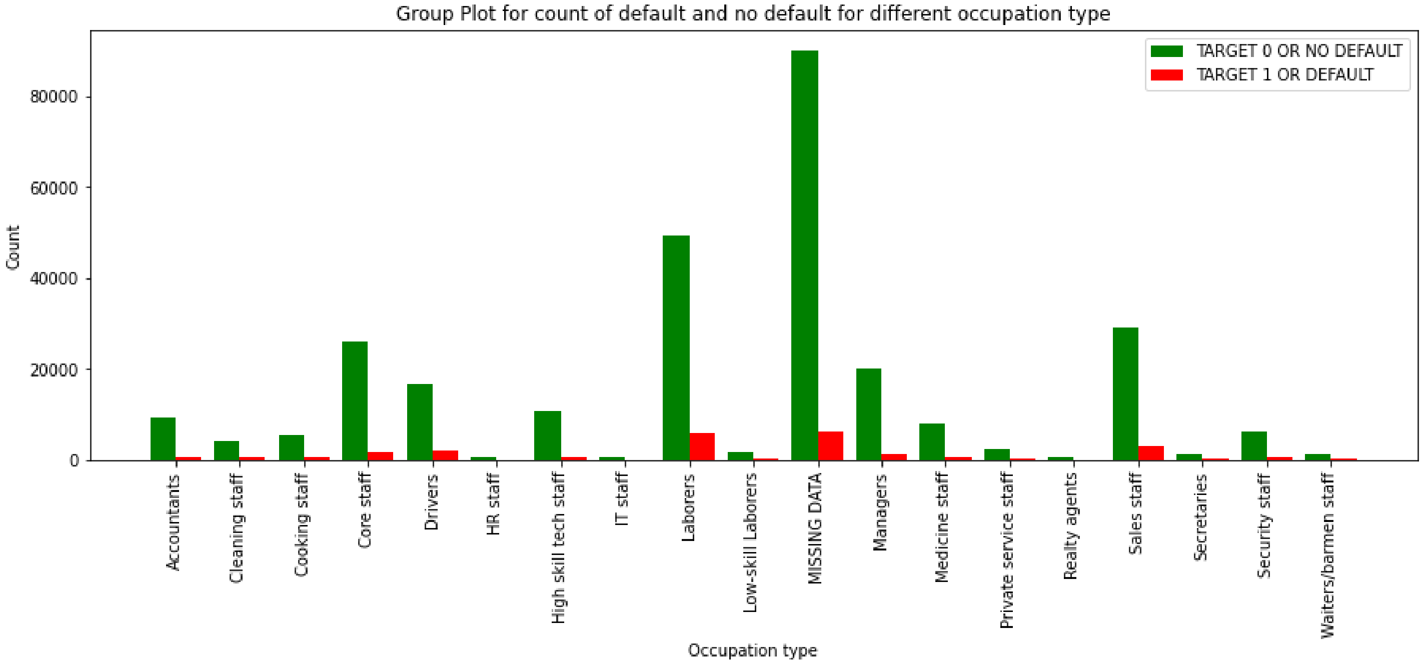
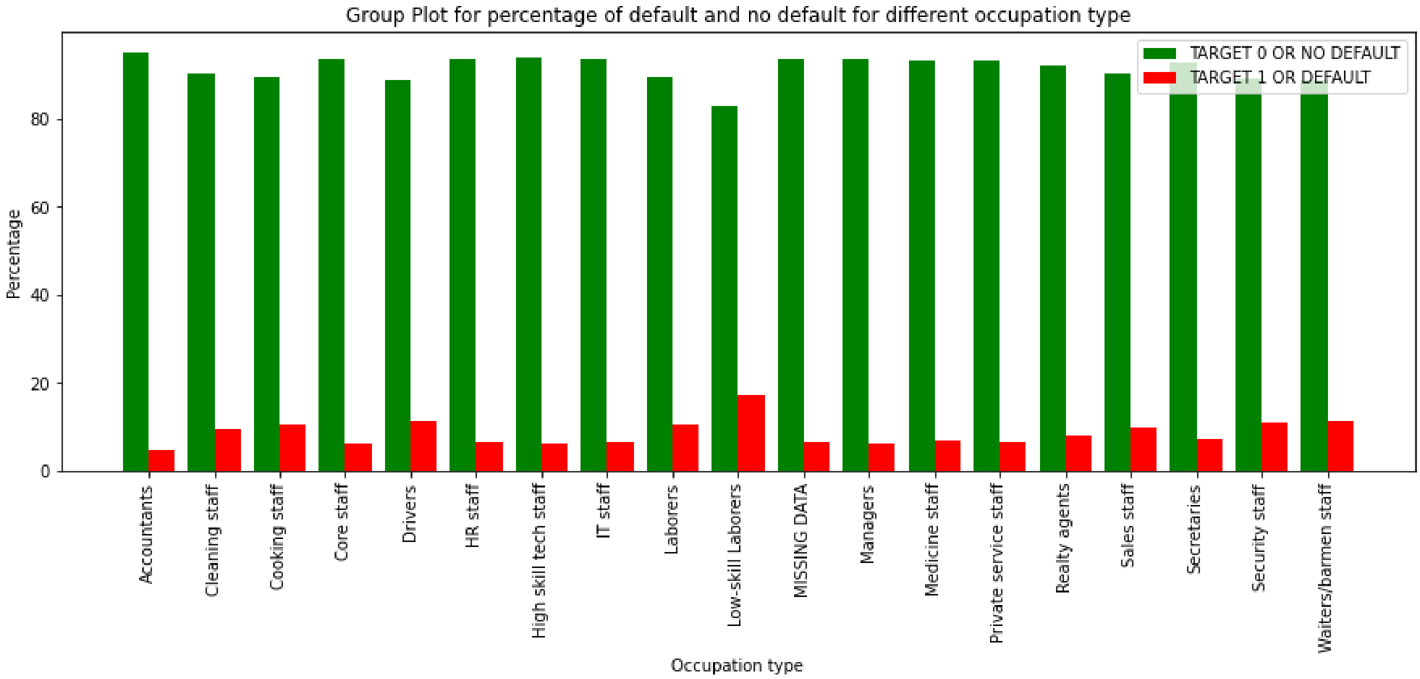


Table of percentage for diffent category

Category	% of Target 0/No Default	% of Target 1/Default
Accountants	95.17	4.83
Cleaning staff	90.39	9.61
Cooking staff	89.56	10.44
Core staff	93.70	6.30
Drivers	88.67	11.33
HR staff	93.61	6.39
High skill tech staff	93.84	6.16
IT staff	93.54	6.46
Laborers	89.42	10.58
Low-skill Laborers	82.85	17.15
MISSING DATA	93.49	6.51
Managers	93.79	6.21
Medicine staff	93.30	6.70
Private service staff	93.40	6.60
Realty agents	92.14	7.86
Sales staff	90.37	9.63

```
Sales staff                   90.37                                   9.63

Secretaries                   92.95                                   7.05

Security staff                89.26                                   10.74

Waiters/barmen staff          88.72                                   11.28
```



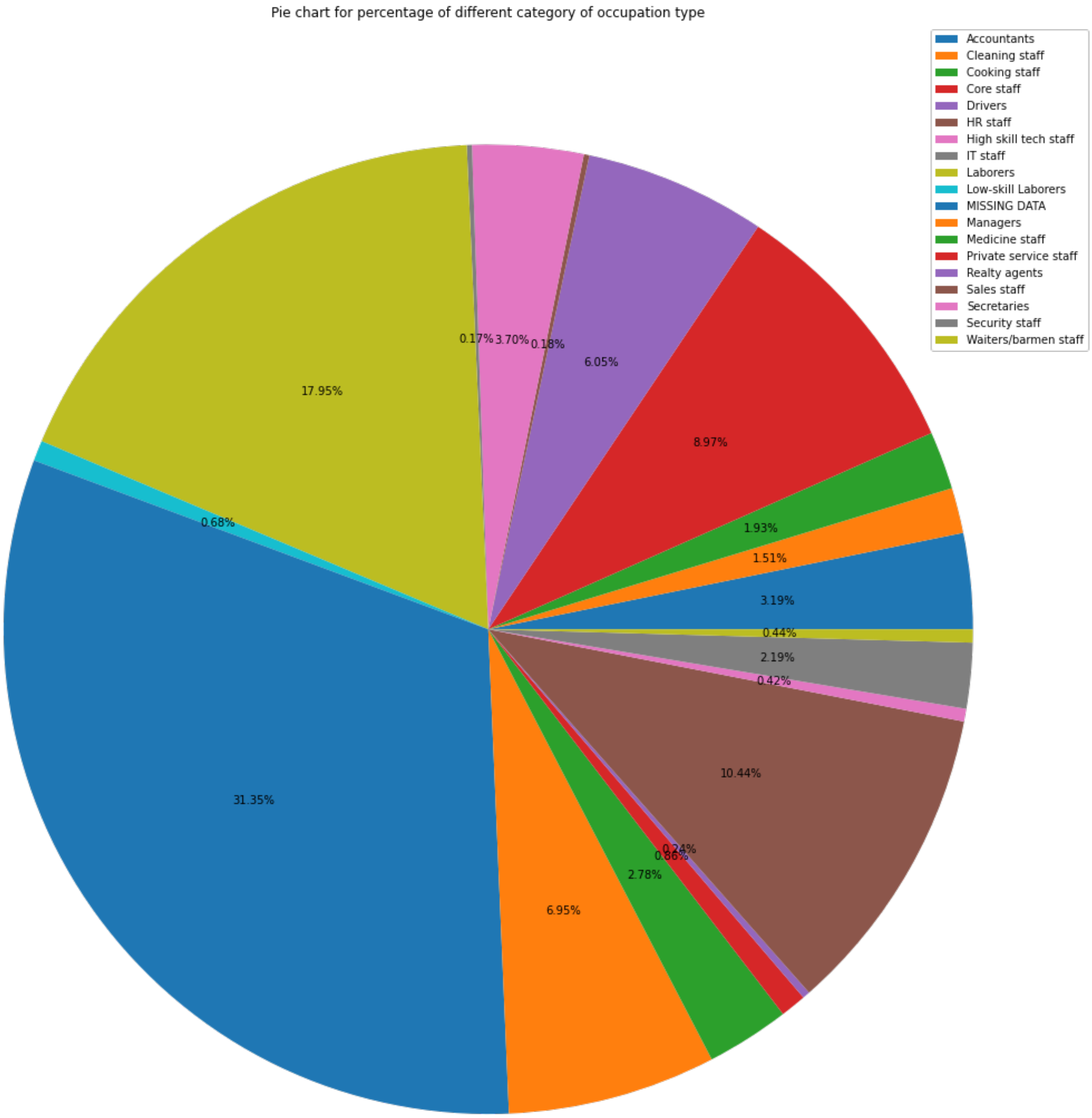
In [ ]:

```python
'''In this code cell, pie chart is plotted for distribution of occupation type'''

application_train_group = application_train[['SK_ID_CURR', 'OCCUPATION_TYPE']].fillna('MISSING DATA')
category_count = (application_train_group.groupby(by = ['OCCUPATION_TYPE']).count()/application_train['NAME_EDUCATION_TYPE'].count())*100
category_name = np.unique(application_train_group['OCCUPATION_TYPE'])
print('Table of percentage of different occupation type')
print("Category".ljust(30) + "Percentage".ljust(30))
for a, b in zip(list(category_name), np.array(category_count)):
  print(a.ljust(30) + str('%.2f'%b).ljust(30))
pie_chart((20,20), category_count, category_name, 'Pie chart for percentage of different category of occupation type')
```

```
Table of percentage of different occupation type
Category                      Percentage
Accountants                   3.19
Cleaning staff                1.51
Cooking staff                 1.93
Core staff                    8.97
Drivers                       6.05
HR staff                      0.18
High skill tech staff         3.70
IT staff                      0.17
Laborers                      17.95
Low-skill Laborers            0.68
MISSING DATA                  31.35
Managers                      6.95
Medicine staff                2.78
Private service staff         0.86
Realty agents                 0.24
Sales staff                   10.44
Secretaries                   0.42
Security staff                2.19
Waiters/barmen staff          0.44
```

```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:6: MatplotlibDeprecationWarning: Non-1D inputs to pie() are currently squeezed()d, but this behavior is deprecated since 3.1 and will be removed in 3.3; pass a 1D array instead.
```

```
ce 3.1 and will be removed in 3.3; pass a 1D array instead.
```



> **31.35% data is missing in case of occupation type. Among the available data, labourers constitute the highest percentage of defaulters. Groups with low percentage of default e.g., Accountants, Core staff, HR staff, High skill tech staff, IT staff etc., should be targeted for disbursing loans. Accountants have the lowest percentage of default and also low percentage of application. They form an important target group. From this preliminary analysis, it seems that occupation type is an important parameter. However, the decision to retain or remove this parameter shall be taken after mathematical analysis of data in the feature selection section.**

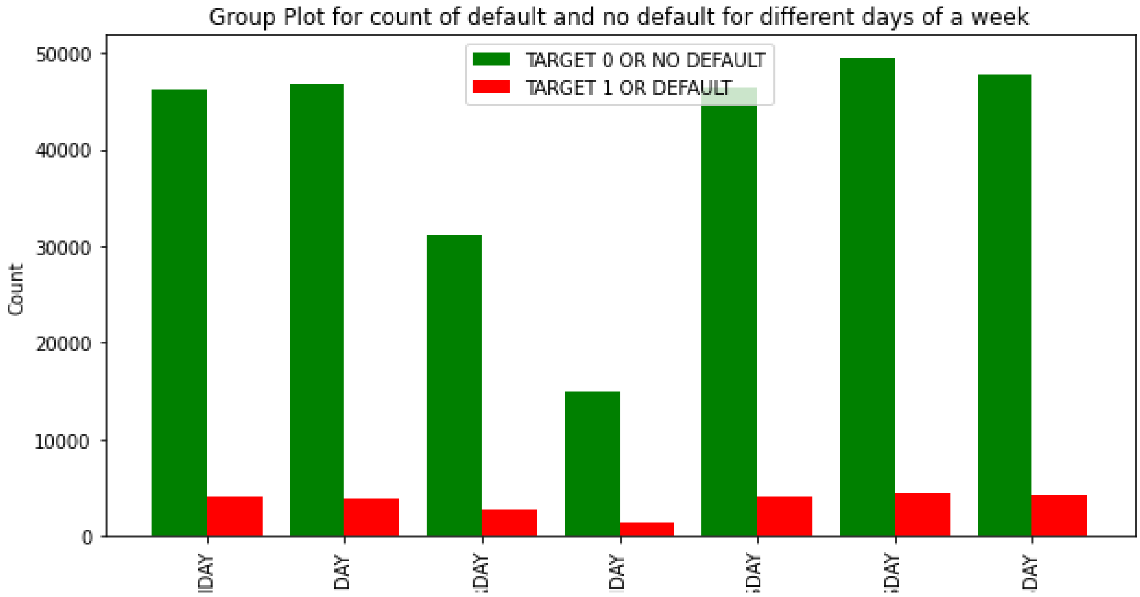
3.9 Analysis based on day of the week

In [ ]:

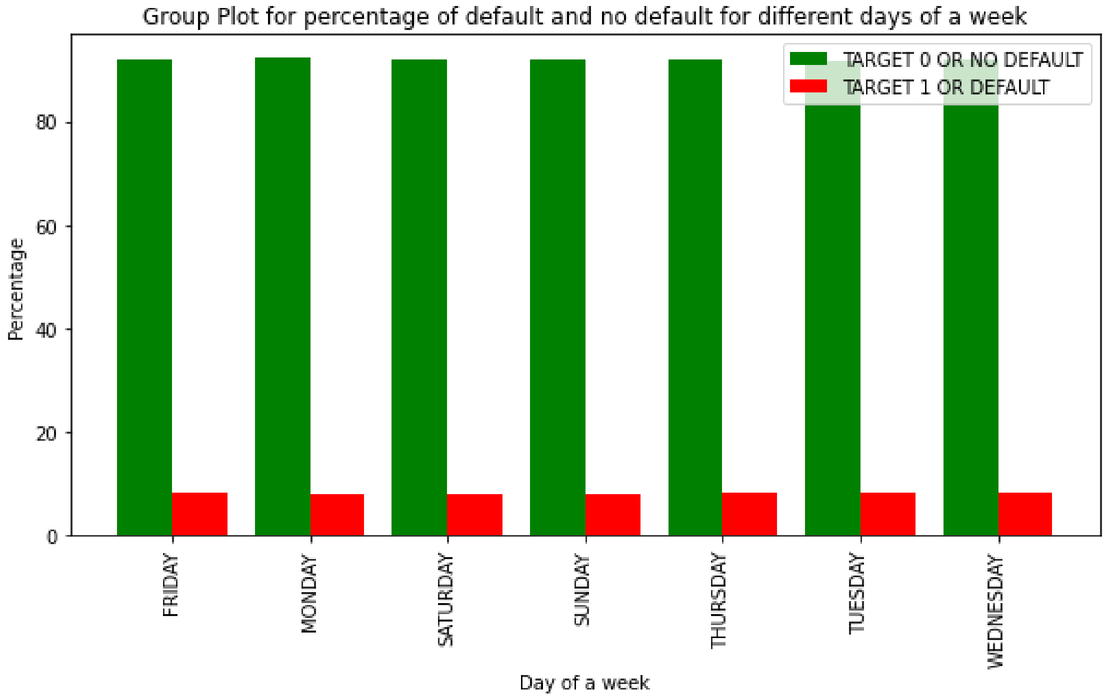
```
'''This code cell does plots graph plot for distribution (count and percentage) of defaul
t
and no default applicants based on day of application'''
```

```python
#Calculate different categories and their count
y_axis = application_train.pivot_table(index='WEEKDAY_APPR_PROCESS_START', columns='TARG
ET', values='SK_ID_CURR', fill_value=0, aggfunc='count').unstack()
classes = np.unique(application_train['WEEKDAY_APPR_PROCESS_START'])
#Print a table of different categories and their count
print('Table of count for diffent category')
print("Category".ljust(30) + "Count of Target 0/No Default".ljust(35) + "Count of Target
1/Default".ljust(30))
for a, b, c in zip(classes, list(y_axis[0]), list(y_axis[1])):
  print(a.ljust(30) + str(b).ljust(35) + str(c).ljust(30))
#Call group_plot function to plot group plot
group_plot((10,5), y_axis[0], y_axis[1], classes, "Day of week", "Count", ["TARGET 0 OR
NO DEFAULT", "TARGET 1 OR DEFAULT"],
           "Group Plot for count of default and no default for different days of a week")

print("-"*100)

#Calculate percentage of different category
y_axis_0_percentage = [(y_axis[0][i]/(y_axis[0][i] + y_axis[1][i]))*100 for i in range(l
en(y_axis[0]))]
y_axis_1_percentage = [(y_axis[1][i]/(y_axis[0][i] + y_axis[1][i]))*100 for i in range(l
en(y_axis[0]))]
#Print a table of different categories and their percentage
print('Table of percentage for diffent category')
print("Category".ljust(30) + "% of Target 0/No Default".ljust(35) + "% of Target 1/Defau
lt".ljust(30))
for a, b, c in zip(classes, list(y_axis_0_percentage), list(y_axis_1_percentage)):
  print(a.ljust(30) + str('%.2f'%b).ljust(35) + str('%.2f'%c).ljust(30))
#Call group_plot function to plot group plot
group_plot((10,5), y_axis_0_percentage, y_axis_1_percentage, classes, "Day of a week", "
Percentage", ["TARGET 0 OR NO DEFAULT", "TARGET 1 OR DEFAULT"],
           "Group Plot for percentage of default and no default for different days of a w
eek")
```

```
Table of count for diffent category
Category                      Count of Target 0/No Default       Count of Target 1/Defau
lt
FRIDAY                        46237                              4101

MONDAY                        46780                              3934

SATURDAY                      31182                              2670

SUNDAY                        14898                              1283

THURSDAY                      46493                              4098

TUESDAY                       49400                              4501

WEDNESDAY                     47696                              4238
```




```
---------------------------------------------------------------------------------------------------
Table of percentage for diffent category
Category                      % of Target 0/No Default          % of Target 1/Default

FRIDAY                        91.85                             8.15

MONDAY                        92.24                             7.76

SATURDAY                      92.11                             7.89

SUNDAY                        92.07                             7.93

THURSDAY                      91.90                             8.10

TUESDAY                       91.65                             8.35

WEDNESDAY                     91.84                             8.16
```



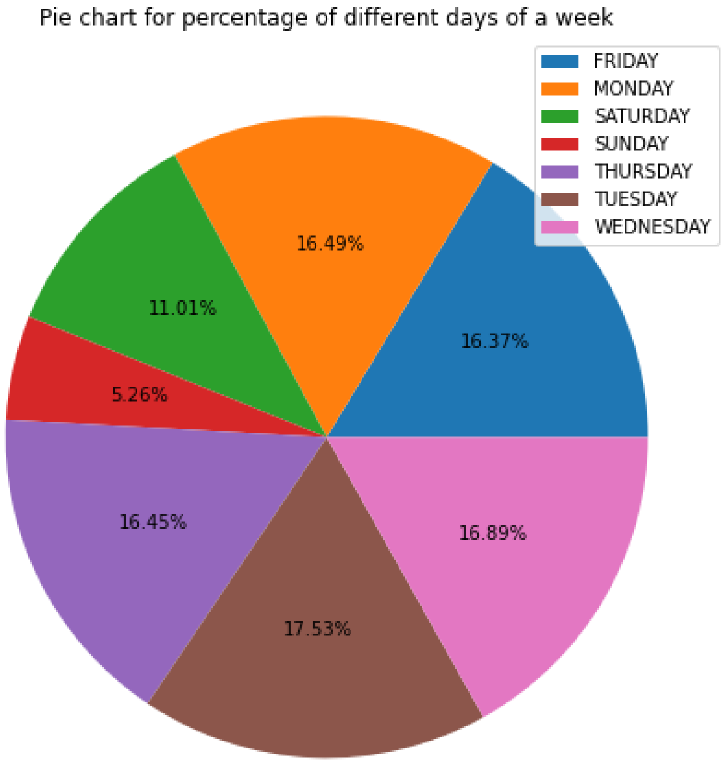
In [ ]:

```
'''In this code cell, pie chart is plotted for distribution of occupation type type'''

application_train_group = application_train[['SK_ID_CURR', 'WEEKDAY_APPR_PROCESS_START']]
.fillna('MISSING')
category_count = (application_train_group.groupby(by = ['WEEKDAY_APPR_PROCESS_START']).c
ount())
category_name = np.unique(application_train_group['WEEKDAY_APPR_PROCESS_START'])
print('Table of percentage of different days of a week')
print("Category".ljust(30) + "Percentage".ljust(30))
for a, b in zip(list(category_name), np.array(category_count)):
  print(a.ljust(30) + str('%.2f'%b).ljust(30))
pie_chart((8,8), category_count, category_name, 'Pie chart for percentage of different da
ys of a week')
```

```
Table of percentage of different days of a week
Category                      Percentage
FRIDAY                        50338.00
MONDAY                        50714.00
SATURDAY                      33852.00
SUNDAY                        16181.00
THURSDAY                      50591.00
TUESDAY                       53901.00
```

```
WEDNESDAY                        51934.00
```

```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:6: MatplotlibDeprecationWarning: Non-1D inputs to pie() are currently squeeze()d, but this behavior is deprecated since 3.1 and will be removed in 3.3; pass a 1D array instead.
```



> **An analysis based on day of the week indicates, most loan applications are done on weekdays. Number of loan applications decreases on Saturday and becomes very less on Sunday. This data is very useful for staff management as more staff is required on weekdays compared to weekends.**

4.0 Feature Engineering

Based upon the domain study in Phase-1 of this project, the following features shall be added to the list of parameters after importing relevant data:

- **Create 3 new ratio:**

1. **Debt-to-Income Ratio - This is the ratio of loan annuity (AMT_ANNUITY) and income (AMT_INCOME_TOTAL) of the applicants.**
2. **Loan-to-Value Ratio - This is the ratio of loan amount (AMT_CREDIT) and price of the goods for which loan is given (AMT_GOODS_PRICE) to the applicants.**
3. **Loan-to-Income Ratio - This is the ratio of loan amount (AMT_CREDIT) and income (AMT_INCOME_TOTAL) of the applicants.**

- **Features shall be added to application_train and application_test from bureau.**
- **Features shall be added to application_train and application_test from previous_application.**

4.1 Import data

In [ ]:

```
'''In this code cell data from all the csv files are imported'''
```

```
#Read application_train
application_train = dataframe_optimizer(pd.read_csv('/content/drive/MyDrive/AI_ML_Project
/Data/application_train.csv'))

#Read application_test
application_test = dataframe_optimizer(pd.read_csv('/content/drive/MyDrive/AI_ML_Project/
Data/application_test.csv'))

#Read bureau
bureau = dataframe_optimizer(pd.read_csv('/content/drive/MyDrive/AI_ML_Project/Data/burea
u.csv'))

#Read bureau_balance
bureau_balance = dataframe_optimizer(pd.read_csv('/content/drive/MyDrive/AI_ML_Project/Da
ta/bureau_balance.csv'))

#Read previous_application
previous_application = dataframe_optimizer(pd.read_csv('/content/drive/MyDrive/AI_ML_Proj
ect/Data/previous_application.csv'))

#Read installments_payments
#installments_payments = pd.read_csv('/content/drive/MyDrive/AI_ML_Project/Data/installme
nts_payments.csv')

#Read POS_CASH_balance
#POS_CASH_balance = pd.read_csv('/content/drive/MyDrive/AI_ML_Project/Data/POS_CASH_balan
ce.csv')

#Read credit_card_balance
#credit_card_balance = pd.read_csv('/content/drive/MyDrive/AI_ML_Project/Data/credit_card
_balance.csv')
```

4.2 Create 3 new ratios from existing columns of application_train and application_test

In [ ]:

```
'''In this code cell, 3 new columns/parameters are being added to application_train and a
pplication_test'''

#Add columns titled DEBT_INCOME_RATIO to application_train
application_train['DEBT_INCOME_RATIO'] = application_train['AMT_ANNUITY']/application_tra
in['AMT_INCOME_TOTAL']

#Add columns titled LOAN_VALUE_RATIO to application_train
application_train['LOAN_VALUE_RATIO'] = application_train['AMT_CREDIT']/application_trai
n['AMT_GOODS_PRICE']

#Add columns titled LOAN_INCOME_RATIO to application_train
application_train['LOAN_INCOME_RATIO'] = application_train['AMT_CREDIT']/application_tra
in['AMT_INCOME_TOTAL']

#Add columns titled DEBT_INCOME_RATIO to application_test
application_test['DEBT_INCOME_RATIO'] = application_test['AMT_ANNUITY']/application_test[
'AMT_INCOME_TOTAL']

#Add columns titled LOAN_VALUE_RATIO to application_test
application_test['LOAN_VALUE_RATIO'] = application_test['AMT_CREDIT']/application_test['
AMT_GOODS_PRICE']

#Add columns titled LOAN_INCOME_RATIO to application_test
application_test['LOAN_INCOME_RATIO'] = application_test['AMT_CREDIT']/application_test[
'AMT_INCOME_TOTAL']
```

In [ ]:

```
'''In this code cell columns with numerical and categorical features are put in separate
lists'''
```

```
numerical_column = list(application_train.drop(columns = ['SK_ID_CURR', 'TARGET']).selec
t_dtypes(exclude=object).columns)
categorical_column = list(application_train.drop(columns = ['TARGET']).select_dtypes(incl
ude=object).columns)
print(numerical_column)
print(categorical_column)
```

```
['CNT_CHILDREN', 'AMT_INCOME_TOTAL', 'AMT_CREDIT', 'AMT_ANNUITY', 'AMT_GOODS_PRICE', 'REG
ION_POPULATION_RELATIVE', 'DAYS_BIRTH', 'DAYS_EMPLOYED', 'DAYS_REGISTRATION', 'DAYS_ID_PU
BLISH', 'OWN_CAR_AGE', 'FLAG_MOBIL', 'FLAG_EMP_PHONE', 'FLAG_WORK_PHONE', 'FLAG_CONT_MOBI
LE', 'FLAG_PHONE', 'FLAG_EMAIL', 'CNT_FAM_MEMBERS', 'REGION_RATING_CLIENT', 'REGION_RATIN
G_CLIENT_W_CITY', 'HOUR_APPR_PROCESS_START', 'REG_REGION_NOT_LIVE_REGION', 'REG_REGION_NO
T_WORK_REGION', 'LIVE_REGION_NOT_WORK_REGION', 'REG_CITY_NOT_LIVE_CITY', 'REG_CITY_NOT_WO
RK_CITY', 'LIVE_CITY_NOT_WORK_CITY', 'EXT_SOURCE_1', 'EXT_SOURCE_2', 'EXT_SOURCE_3', 'APA
RTMENTS_AVG', 'BASEMENTAREA_AVG', 'YEARS_BEGINEXPLUATATION_AVG', 'YEARS_BUILD_AVG', 'COMM
ONAREA_AVG', 'ELEVATORS_AVG', 'ENTRANCES_AVG', 'FLOORSMAX_AVG', 'FLOORSMIN_AVG', 'LANDARE
A_AVG', 'LIVINGAPARTMENTS_AVG', 'LIVINGAREA_AVG', 'NONLIVINGAPARTMENTS_AVG', 'NONLIVINGAR
EA_AVG', 'APARTMENTS_MODE', 'BASEMENTAREA_MODE', 'YEARS_BEGINEXPLUATATION_MODE', 'YEARS_B
UILD_MODE', 'COMMONAREA_MODE', 'ELEVATORS_MODE', 'ENTRANCES_MODE', 'FLOORSMAX_MODE', 'FLO
ORSMIN_MODE', 'LANDAREA_MODE', 'LIVINGAPARTMENTS_MODE', 'LIVINGAREA_MODE', 'NONLIVINGAPAR
TMENTS_MODE', 'NONLIVINGAREA_MODE', 'APARTMENTS_MEDI', 'BASEMENTAREA_MEDI', 'YEARS_BEGINE
XPLUATATION_MEDI', 'YEARS_BUILD_MEDI', 'COMMONAREA_MEDI', 'ELEVATORS_MEDI', 'ENTRANCES_ME
DI', 'FLOORSMAX_MEDI', 'FLOORSMIN_MEDI', 'LANDAREA_MEDI', 'LIVINGAPARTMENTS_MEDI', 'LIVIN
GAREA_MEDI', 'NONLIVINGAPARTMENTS_MEDI', 'NONLIVINGAREA_MEDI', 'TOTALAREA_MODE', 'OBS_30_
CNT_SOCIAL_CIRCLE', 'DEF_30_CNT_SOCIAL_CIRCLE', 'OBS_60_CNT_SOCIAL_CIRCLE', 'DEF_60_CNT_S
OCIAL_CIRCLE', 'DAYS_LAST_PHONE_CHANGE', 'FLAG_DOCUMENT_2', 'FLAG_DOCUMENT_3', 'FLAG_DOCU
MENT_4', 'FLAG_DOCUMENT_5', 'FLAG_DOCUMENT_6', 'FLAG_DOCUMENT_7', 'FLAG_DOCUMENT_8', 'FLA
G_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', 'FL
AG_DOCUMENT_18', 'FLAG_DOCUMENT_19', 'FLAG_DOCUMENT_20', 'FLAG_DOCUMENT_21', 'AMT_REQ_CRE
DIT_BUREAU_HOUR', 'AMT_REQ_CREDIT_BUREAU_DAY', 'AMT_REQ_CREDIT_BUREAU_WEEK', 'AMT_REQ_CRE
DIT_BUREAU_MON', 'AMT_REQ_CREDIT_BUREAU_QRT', 'AMT_REQ_CREDIT_BUREAU_YEAR', 'DEBT_INCOME_
RATIO', 'LOAN_VALUE_RATIO', 'LOAN_INCOME_RATIO']
['NAME_CONTRACT_TYPE', 'CODE_GENDER', 'FLAG_OWN_CAR', 'FLAG_OWN_REALTY', 'NAME_TYPE_SUITE
', 'NAME_INCOME_TYPE', 'NAME_EDUCATION_TYPE', 'NAME_FAMILY_STATUS', 'NAME_HOUSING_TYPE',
'OCCUPATION_TYPE', 'WEEKDAY_APPR_PROCESS_START', 'ORGANIZATION_TYPE', 'FONDKAPREMONT_MODE
', 'HOUSETYPE_MODE', 'WALLSMATERIAL_MODE', 'EMERGENCYSTATE_MODE']
```

4.3 Merge bureau with application_train and application_test

In [ ]:

```
'''In this code cell columns common among application (application_train/application_test
)
and bureau are found'''

#Find common column between application and bureau and print
bureau_common = set(set(application_train.columns) & set(bureau.columns))
print(bureau_common)
```

```
{'SK_ID_CURR', 'AMT_ANNUITY'}
```

In [ ]:

```
'''In this code cell common columns are renamed in bureau and put in a dictionary'''

#Rename columns common in bureau except SK_ID_CURR as merge will be performed on SK_ID_CU
RR
bureau.rename(columns={"AMT_ANNUITY": "AMT_ANNUITY_BUREAU"}, inplace = True)
#Save the renamed column in a dictionary for future reference
bureau_renamed = {"AMT_ANNUITY": "AMT_ANNUITY_BUREAU"}
#Create a dataframe with numerical columns of bureau
bureau_numerical = bureau.select_dtypes(exclude=object)
#Create a dataframe with categorical columns of bureau
bureau_categorical = bureau.select_dtypes(include=object)
```

In [ ]:

```
'''In this code cell numerical columns of bureau are merged with application_train'''
```

```
#Merge numerical features from bureau to application_train
bureau_numerical_merge = bureau_numerical.groupby(by=['SK_ID_CURR']).mean().reset_index(
)
application_train_1 = application_train.merge(bureau_numerical_merge, on='SK_ID_CURR', h
ow='left', suffixes=('__left', ))
application_train_1.columns = application_train_1.columns.str.replace('__left', '')

#Merge categorical features from bureau to application_train
bureau_categorical_merge = pd.get_dummies(bureau_categorical)
bureau_categorical_merge['SK_ID_CURR'] = bureau['SK_ID_CURR']
bureau_categorical_merge = bureau_categorical_merge.groupby(by = ['SK_ID_CURR']).median(
)
application_train_1 = application_train_1.merge(bureau_categorical_merge, on='SK_ID_CURR'
, how='left')
application_train_1.update(application_train_1[bureau_categorical_merge.columns].fillna(0
))
application_train_1.columns = application_train_1.columns.str.replace('__left', '')

#Drop SK_ID_BUREAU
application_train_1 = application_train_1.drop(columns = ['SK_ID_BUREAU'])

#Shape of application and bureau data combined
print('The shape of application_train and bureau data merged: ', application_train_1.shap
e)
```

```
The shape of application_train and bureau data merged:  (307511, 160)
```

In [ ]:

```
'''In this code cell numerical columns of bureau are merged with application_test'''

#Merge numerical features from bureau to application_test
bureau_numerical_merge = bureau.groupby(by=['SK_ID_CURR']).mean().reset_index()
application_test_1 = application_test.merge(bureau_numerical_merge, on='SK_ID_CURR', how
='left', suffixes=('__left', ))
application_test_1.columns = application_test_1.columns.str.replace('__left', '')

#Merge categorical features from bureau to application_test
bureau_categorical_merge = pd.get_dummies(bureau_categorical)
bureau_categorical_merge['SK_ID_CURR'] = bureau['SK_ID_CURR']
bureau_categorical_merge = bureau_categorical_merge.groupby(by = ['SK_ID_CURR']).median(
)
application_test_1 = application_test_1.merge(bureau_categorical_merge, on='SK_ID_CURR',
how='left')
application_test_1.update(application_test_1[bureau_categorical_merge.columns].fillna(0))
application_test_1.columns = application_test_1.columns.str.replace('__left', '')

#Drop SK_ID_BUREAU
application_test_1 = application_test_1.drop(columns = ['SK_ID_BUREAU'])

#Shape of application and bureau data combined
print('The shape of application_test and bureau data merged: ', application_test_1.shape)
```

```
The shape of application_test and bureau data merged:  (48744, 159)
```

In [ ]:

```
'''In this code cell list of numerical and categorical columns are updated'''

#Update and print numerical and categorical columns
numerical_column = numerical_column + list(bureau_numerical.columns)
categorical_column = categorical_column + list(bureau_categorical_merge.columns)
numerical_column.remove("SK_ID_CURR")
numerical_column.remove("SK_ID_BUREAU")
print(numerical_column)
print(categorical_column)
```

```
['CNT_CHILDREN', 'AMT_INCOME_TOTAL', 'AMT_CREDIT', 'AMT_ANNUITY', 'AMT_GOODS_PRICE', 'REG
ION_POPULATION_RELATIVE', 'DAYS_BIRTH', 'DAYS_EMPLOYED', 'DAYS_REGISTRATION', 'DAYS_ID_PU
BLISH', 'OWN_CAR_AGE', 'FLAG_MOBIL', 'FLAG_EMP_PHONE', 'FLAG_WORK_PHONE', 'FLAG_CONT_MOBI
LE', 'FLAG_PHONE', 'FLAG_EMAIL', 'CNT_FAM_MEMBERS', 'REGION_RATING_CLIENT', 'REGION_RATIN
G_CLIENT_W_CITY', 'HOUR_APPR_PROCESS_START', 'REG_REGION_NOT_LIVE_REGION', 'REG_REGION_NO
```

```
T_WORK_REGION', 'LIVE_REGION_NOT_WORK_REGION', 'REG_CITY_NOT_LIVE_CITY', 'REG_CITY_NOT_WO
RK_CITY', 'LIVE_CITY_NOT_WORK_CITY', 'EXT_SOURCE_1', 'EXT_SOURCE_2', 'EXT_SOURCE_3', 'APA
RTMENTS_AVG', 'BASEMENTAREA_AVG', 'YEARS_BEGINEXPLUATATION_AVG', 'YEARS_BUILD_AVG', 'COMM
ONAREA_AVG', 'ELEVATORS_AVG', 'ENTRANCES_AVG', 'FLOORSMAX_AVG', 'FLOORSMIN_AVG', 'LANDARE
A_AVG', 'LIVINGAPARTMENTS_AVG', 'LIVINGAREA_AVG', 'NONLIVINGAPARTMENTS_AVG', 'NONLIVINGAR
EA_AVG', 'APARTMENTS_MODE', 'BASEMENTAREA_MODE', 'YEARS_BEGINEXPLUATATION_MODE', 'YEARS_B
UILD_MODE', 'COMMONAREA_MODE', 'ELEVATORS_MODE', 'ENTRANCES_MODE', 'FLOORSMAX_MODE', 'FLO
ORSMIN_MODE', 'LANDAREA_MODE', 'LIVINGAPARTMENTS_MODE', 'LIVINGAREA_MODE', 'NONLIVINGAPAR
TMENTS_MODE', 'NONLIVINGAREA_MODE', 'APARTMENTS_MEDI', 'BASEMENTAREA_MEDI', 'YEARS_BEGINE
XPLUATATION_MEDI', 'YEARS_BUILD_MEDI', 'COMMONAREA_MEDI', 'ELEVATORS_MEDI', 'ENTRANCES_ME
DI', 'FLOORSMAX_MEDI', 'FLOORSMIN_MEDI', 'LANDAREA_MEDI', 'LIVINGAPARTMENTS_MEDI', 'LIVIN
GAREA_MEDI', 'NONLIVINGAPARTMENTS_MEDI', 'NONLIVINGAREA_MEDI', 'TOTALAREA_MODE', 'OBS_30_
CNT_SOCIAL_CIRCLE', 'DEF_30_CNT_SOCIAL_CIRCLE', 'OBS_60_CNT_SOCIAL_CIRCLE', 'DEF_60_CNT_S
OCIAL_CIRCLE', 'DAYS_LAST_PHONE_CHANGE', 'FLAG_DOCUMENT_2', 'FLAG_DOCUMENT_3', 'FLAG_DOCU
MENT_4', 'FLAG_DOCUMENT_5', 'FLAG_DOCUMENT_6', 'FLAG_DOCUMENT_7', 'FLAG_DOCUMENT_8', 'FLA
G_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', 'FL
AG_DOCUMENT_18', 'FLAG_DOCUMENT_19', 'FLAG_DOCUMENT_20', 'FLAG_DOCUMENT_21', 'AMT_REQ_CRE
DIT_BUREAU_HOUR', 'AMT_REQ_CREDIT_BUREAU_DAY', 'AMT_REQ_CREDIT_BUREAU_WEEK', 'AMT_REQ_CRE
DIT_BUREAU_MON', 'AMT_REQ_CREDIT_BUREAU_QRT', 'AMT_REQ_CREDIT_BUREAU_YEAR', 'DEBT_INCOME_
RATIO', 'LOAN_VALUE_RATIO', 'LOAN_INCOME_RATIO', 'DAYS_CREDIT', 'CREDIT_DAY_OVERDUE', 'DA
YS_CREDIT_ENDDATE', 'DAYS_ENDDATE_FACT', 'AMT_CREDIT_MAX_OVERDUE', 'CNT_CREDIT_PROLONG',
'AMT_CREDIT_SUM', 'AMT_CREDIT_SUM_DEBT', 'AMT_CREDIT_SUM_LIMIT', 'AMT_CREDIT_SUM_OVERDUE'
, 'DAYS_CREDIT_UPDATE', 'AMT_ANNUITY_BUREAU']
['NAME_CONTRACT_TYPE', 'CODE_GENDER', 'FLAG_OWN_CAR', 'FLAG_OWN_REALTY', 'NAME_TYPE_SUITE
', 'NAME_INCOME_TYPE', 'NAME_EDUCATION_TYPE', 'NAME_FAMILY_STATUS', 'NAME_HOUSING_TYPE',
'OCCUPATION_TYPE', 'WEEKDAY_APPR_PROCESS_START', 'ORGANIZATION_TYPE', 'FONDKAPREMONT_MODE
', 'HOUSETYPE_MODE', 'WALLSMATERIAL_MODE', 'EMERGENCYSTATE_MODE', 'CREDIT_ACTIVE_Active',
'CREDIT_ACTIVE_Bad debt', 'CREDIT_ACTIVE_Closed', 'CREDIT_ACTIVE_Sold', 'CREDIT_CURRENCY_
currency 1', 'CREDIT_CURRENCY_currency 2', 'CREDIT_CURRENCY_currency 3', 'CREDIT_CURRENCY
_currency 4', 'CREDIT_TYPE_Another type of loan', 'CREDIT_TYPE_Car loan', 'CREDIT_TYPE_Ca
sh loan (non-earmarked)', 'CREDIT_TYPE_Consumer credit', 'CREDIT_TYPE_Credit card', 'CRED
IT_TYPE_Interbank credit', 'CREDIT_TYPE_Loan for business development', 'CREDIT_TYPE_Loan
for purchase of shares (margin lending)', 'CREDIT_TYPE_Loan for the purchase of equipment
', 'CREDIT_TYPE_Loan for working capital replenishment', 'CREDIT_TYPE_Microloan', 'CREDIT
_TYPE_Mobile operator loan', 'CREDIT_TYPE_Mortgage', 'CREDIT_TYPE_Real estate loan', 'CRE
DIT_TYPE_Unknown type of loan']
```

4.4 Merge previous_application with application_train_1 and application_test_1

In [ ]:

```
'''In this code cell columns common among application (application_train/application_test
)
and bureau are found'''

#Find common column between application and previous_application and print
previous_application_common = set(set(application_train.columns) & set(previous_applicati
on.columns))
print(previous_application_common)
```

```
{'SK_ID_CURR', 'AMT_CREDIT', 'AMT_GOODS_PRICE', 'HOUR_APPR_PROCESS_START', 'NAME_CONTRACT
_TYPE', 'NAME_TYPE_SUITE', 'AMT_ANNUITY', 'WEEKDAY_APPR_PROCESS_START'}
```

In [ ]:

```
'''In this code cell common columns are renamed in previous_application and put in a dict
ionary'''

#Rename common column in previous_application except SK_ID_CURR as merge will be performe
d on SK_ID_CURR
previous_application.rename(columns={"NAME_TYPE_SUITE": "NAME_TYPE_SUITE_PREVIOUS_APPLICA
TION", "AMT_GOODS_PRICE": "AMT_GOODS_PRICE_PREVIOUS_APPLICATION",
                           "WEEKDAY_APPR_PROCESS_START": "WEEKDAY_APPR_PROCESS_START_PREVIOU
S_APPLICATION", "NAME_CONTRACT_TYPE": "NAME_CONTRACT_TYPE_PREVIOUS_APPLICATION",
                           "AMT_CREDIT": "AMT_CREDIT_PREVIOUS_APPLICATION", "HOUR_APPR_PROCE
SS_START": "HOUR_APPR_PROCESS_START_PREVIOUS_APPLICATION",
                           "AMT_ANNUITY":"AMT_ANNUITY_PREVIOUS_APPLICATION"}, inplace = True
)
```

```
#Save the renamed column in a dictionary for future reference
previous_application_renamed = {"NAME_TYPE_SUITE": "NAME_TYPE_SUITE_PREVIOUS_APPLICATION"
, "AMT_GOODS_PRICE": "AMT_GOODS_PRICE_PREVIOUS_APPLICATION",
                      "WEEKDAY_APPR_PROCESS_START": "WEEKDAY_APPR_PROCESS_START_PREVIOU
S_APPLICATION", "NAME_CONTRACT_TYPE": "NAME_CONTRACT_TYPE_PREVIOUS_APPLICATION",
                      "AMT_CREDIT": "AMT_CREDIT_PREVIOUS_APPLICATION", "HOUR_APPR_PROCE
SS_START": "HOUR_APPR_PROCESS_START_PREVIOUS_APPLICATION",
                      "AMT_ANNUITY":"AMT_ANNUITY_PREVIOUS_APPLICATION"}
#Create a dataframe with numerical columns of previous_application
previous_application_numerical = previous_application.select_dtypes(exclude=object)
#Create a dataframe with categorical columns of previous_application
previous_application_categorical = previous_application.select_dtypes(include=object)
```

In [ ]:

```
'''In this code cell numerical columns of bureau are merged with application_train_1'''

#Merge numerical features from previous_application to application_train_1
previous_application_numerical_merge = previous_application_numerical.groupby(by=['SK_ID
_CURR']).mean().reset_index()
application_train_2 = application_train_1.merge(previous_application_numerical_merge, on=
'SK_ID_CURR', how='left', suffixes=('__left', ))
application_train_2.columns = application_train_2.columns.str.replace('__left', '')

#Merge categorical features from previous_application to application_train_1
previous_application_categorical_merge = pd.get_dummies(previous_application_categorical)
previous_application_categorical_merge['SK_ID_CURR'] = previous_application['SK_ID_CURR']
previous_application_categorical_merge = previous_application_categorical_merge.groupby(b
y = ['SK_ID_CURR']).median()
application_train_2 = application_train_2.merge(previous_application_categorical_merge, o
n='SK_ID_CURR', how='left', suffixes=('__left', ))
application_train_2.update(application_train_2[previous_application_categorical_merge.col
umns].fillna(0))
application_train_2.columns = application_train_2.columns.str.replace('__left', '')

#Drop SK_ID_PREV
application_train_2 = application_train_2.drop(columns = ['SK_ID_PREV'])

#Shape of application_train_1 and previous_application data combined
print('The shape of application_train_1 and previous_application data merged: ', applicat
ion_train_2.shape)
```

```
The shape of application_train_1 and previous_application data merged:  (307511, 322)
```

In [ ]:

```
'''In this code cell numerical columns of bureau are merged with application_test_1'''

#Merge numerical features from previous_application to application_test_1
previous_application_numerical_merge = previous_application_numerical.groupby(by=['SK_ID
_CURR']).mean().reset_index()
application_test_2 = application_test_1.merge(previous_application_numerical_merge, on='S
K_ID_CURR', how='left', suffixes=('__left', ))
application_test_2.columns = application_test_2.columns.str.replace('__left', '')

#Merge categorical features from previous_application to application_test_1
previous_application_categorical_merge = pd.get_dummies(previous_application_categorical)
previous_application_categorical_merge['SK_ID_CURR'] = previous_application['SK_ID_CURR']
previous_application_categorical_merge = previous_application_categorical_merge.groupby(b
y = ['SK_ID_CURR']).median()
application_test_2 = application_test_2.merge(previous_application_categorical_merge, on=
'SK_ID_CURR', how='left', suffixes=('__left', ))
application_test_2.update(application_test_2[previous_application_categorical_merge.colum
ns].fillna(0))
application_test_2.columns = application_test_2.columns.str.replace('__left', '')

#Drop SK_ID_PREV
application_test_2 = application_test_2.drop(columns = ['SK_ID_PREV'])

#Shape of application_test_1 and previous_application data combined
print('The shape of application_test_1 and previous_application data merged: ', applicati
```

```
on_test_2.shape)
```

```
The shape of application_test_1 and previous_application data merged:  (48744, 321)
```

In [ ]:

```
'''In this code cell list of numerical and categorical columns are updated'''

#Update and print numerical and categorical columns
numerical_column = numerical_column + list(previous_application_numerical.columns)
categorical_column = categorical_column + list(previous_application_categorical_merge.col
umns)
numerical_column.remove("SK_ID_CURR")
numerical_column.remove("SK_ID_PREV")
print(numerical_column)
print(categorical_column)
```

```
['CNT_CHILDREN', 'AMT_INCOME_TOTAL', 'AMT_CREDIT', 'AMT_ANNUITY', 'AMT_GOODS_PRICE', 'REG
ION_POPULATION_RELATIVE', 'DAYS_BIRTH', 'DAYS_EMPLOYED', 'DAYS_REGISTRATION', 'DAYS_ID_PU
BLISH', 'OWN_CAR_AGE', 'FLAG_MOBIL', 'FLAG_EMP_PHONE', 'FLAG_WORK_PHONE', 'FLAG_CONT_MOBI
LE', 'FLAG_PHONE', 'FLAG_EMAIL', 'CNT_FAM_MEMBERS', 'REGION_RATING_CLIENT', 'REGION_RATIN
G_CLIENT_W_CITY', 'HOUR_APPR_PROCESS_START', 'REG_REGION_NOT_LIVE_REGION', 'REG_REGION_NO
T_WORK_REGION', 'LIVE_REGION_NOT_WORK_REGION', 'REG_CITY_NOT_LIVE_CITY', 'REG_CITY_NOT_WO
RK_CITY', 'LIVE_CITY_NOT_WORK_CITY', 'EXT_SOURCE_1', 'EXT_SOURCE_2', 'EXT_SOURCE_3', 'APA
RTMENTS_AVG', 'BASEMENTAREA_AVG', 'YEARS_BEGINEXPLUATATION_AVG', 'YEARS_BUILD_AVG', 'COMM
ONAREA_AVG', 'ELEVATORS_AVG', 'ENTRANCES_AVG', 'FLOORSMAX_AVG', 'FLOORSMIN_AVG', 'LANDARE
A_AVG', 'LIVINGAPARTMENTS_AVG', 'LIVINGAREA_AVG', 'NONLIVINGAPARTMENTS_AVG', 'NONLIVINGAR
EA_AVG', 'APARTMENTS_MODE', 'BASEMENTAREA_MODE', 'YEARS_BEGINEXPLUATATION_MODE', 'YEARS_B
UILD_MODE', 'COMMONAREA_MODE', 'ELEVATORS_MODE', 'ENTRANCES_MODE', 'FLOORSMAX_MODE', 'FLO
ORSMIN_MODE', 'LANDAREA_MODE', 'LIVINGAPARTMENTS_MODE', 'LIVINGAREA_MODE', 'NONLIVINGAPAR
TMENTS_MODE', 'NONLIVINGAREA_MODE', 'APARTMENTS_MEDI', 'BASEMENTAREA_MEDI', 'YEARS_BEGINE
XPLUATATION_MEDI', 'YEARS_BUILD_MEDI', 'COMMONAREA_MEDI', 'ELEVATORS_MEDI', 'ENTRANCES_ME
DI', 'FLOORSMAX_MEDI', 'FLOORSMIN_MEDI', 'LANDAREA_MEDI', 'LIVINGAPARTMENTS_MEDI', 'LIVIN
GAREA_MEDI', 'NONLIVINGAPARTMENTS_MEDI', 'NONLIVINGAREA_MEDI', 'TOTALAREA_MODE', 'OBS_30_
CNT_SOCIAL_CIRCLE', 'DEF_30_CNT_SOCIAL_CIRCLE', 'OBS_60_CNT_SOCIAL_CIRCLE', 'DEF_60_CNT_S
OCIAL_CIRCLE', 'DAYS_LAST_PHONE_CHANGE', 'FLAG_DOCUMENT_2', 'FLAG_DOCUMENT_3', 'FLAG_DOCU
MENT_4', 'FLAG_DOCUMENT_5', 'FLAG_DOCUMENT_6', 'FLAG_DOCUMENT_7', 'FLAG_DOCUMENT_8', 'FLA
G_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', 'FL
AG_DOCUMENT_18', 'FLAG_DOCUMENT_19', 'FLAG_DOCUMENT_20', 'FLAG_DOCUMENT_21', 'AMT_REQ_CRE
DIT_BUREAU_HOUR', 'AMT_REQ_CREDIT_BUREAU_DAY', 'AMT_REQ_CREDIT_BUREAU_WEEK', 'AMT_REQ_CRE
DIT_BUREAU_MON', 'AMT_REQ_CREDIT_BUREAU_QRT', 'AMT_REQ_CREDIT_BUREAU_YEAR', 'DEBT_INCOME_
RATIO', 'LOAN_VALUE_RATIO', 'LOAN_INCOME_RATIO', 'DAYS_CREDIT', 'CREDIT_DAY_OVERDUE', 'DA
YS_CREDIT_ENDDATE', 'DAYS_ENDDATE_FACT', 'AMT_CREDIT_MAX_OVERDUE', 'CNT_CREDIT_PROLONG',
'AMT_CREDIT_SUM', 'AMT_CREDIT_SUM_DEBT', 'AMT_CREDIT_SUM_LIMIT', 'AMT_CREDIT_SUM_OVERDUE'
, 'DAYS_CREDIT_UPDATE', 'AMT_ANNUITY_BUREAU', 'AMT_ANNUITY_PREVIOUS_APPLICATION', 'AMT_AP
PLICATION', 'AMT_CREDIT_PREVIOUS_APPLICATION', 'AMT_DOWN_PAYMENT', 'AMT_GOODS_PRICE_PREVI
OUS_APPLICATION', 'HOUR_APPR_PROCESS_START_PREVIOUS_APPLICATION', 'NFLAG_LAST_APPL_IN_DAY
', 'RATE_DOWN_PAYMENT', 'RATE_INTEREST_PRIMARY', 'RATE_INTEREST_PRIVILEGED', 'DAYS_DECISI
ON', 'SELLERPLACE_AREA', 'CNT_PAYMENT', 'DAYS_FIRST_DRAWING', 'DAYS_FIRST_DUE', 'DAYS_LAS
T_DUE_1ST_VERSION', 'DAYS_LAST_DUE', 'DAYS_TERMINATION', 'NFLAG_INSURED_ON_APPROVAL']
['NAME_CONTRACT_TYPE', 'CODE_GENDER', 'FLAG_OWN_CAR', 'FLAG_OWN_REALTY', 'NAME_TYPE_SUITE
', 'NAME_INCOME_TYPE', 'NAME_EDUCATION_TYPE', 'NAME_FAMILY_STATUS', 'NAME_HOUSING_TYPE',
'OCCUPATION_TYPE', 'WEEKDAY_APPR_PROCESS_START', 'ORGANIZATION_TYPE', 'FONDKAPREMONT_MODE
', 'HOUSETYPE_MODE', 'WALLSMATERIAL_MODE', 'EMERGENCYSTATE_MODE', 'CREDIT_ACTIVE_Active',
'CREDIT_ACTIVE_Bad debt', 'CREDIT_ACTIVE_Closed', 'CREDIT_ACTIVE_Sold', 'CREDIT_CURRENCY_
currency 1', 'CREDIT_CURRENCY_currency 2', 'CREDIT_CURRENCY_currency 3', 'CREDIT_CURRENCY
_currency 4', 'CREDIT_TYPE_Another type of loan', 'CREDIT_TYPE_Car loan', 'CREDIT_TYPE_Ca
sh loan (non-earmarked)', 'CREDIT_TYPE_Consumer credit', 'CREDIT_TYPE_Credit card', 'CRED
IT_TYPE_Interbank credit', 'CREDIT_TYPE_Loan for business development', 'CREDIT_TYPE_Loan
 for purchase of shares (margin lending)', 'CREDIT_TYPE_Loan for the purchase of equipment
', 'CREDIT_TYPE_Loan for working capital replenishment', 'CREDIT_TYPE_Microloan', 'CREDIT
_TYPE_Mobile operator loan', 'CREDIT_TYPE_Mortgage', 'CREDIT_TYPE_Real estate loan', 'CRE
DIT_TYPE_Unknown type of loan', 'NAME_CONTRACT_TYPE_PREVIOUS_APPLICATION_Cash loans', 'NA
ME_CONTRACT_TYPE_PREVIOUS_APPLICATION_Consumer loans', 'NAME_CONTRACT_TYPE_PREVIOUS_APPLI
CATION_Revolving loans', 'NAME_CONTRACT_TYPE_PREVIOUS_APPLICATION_XNA', 'WEEKDAY_APPR_PRO
CESS_START_PREVIOUS_APPLICATION_FRIDAY', 'WEEKDAY_APPR_PROCESS_START_PREVIOUS_APPLICATION
_MONDAY', 'WEEKDAY_APPR_PROCESS_START_PREVIOUS_APPLICATION_SATURDAY', 'WEEKDAY_APPR_PROCE
SS_START_PREVIOUS_APPLICATION_SUNDAY', 'WEEKDAY_APPR_PROCESS_START_PREVIOUS_APPLICATION_T
HURSDAY', 'WEEKDAY_APPR_PROCESS_START_PREVIOUS_APPLICATION_TUESDAY', 'WEEKDAY_APPR_PROCES
S_START_PREVIOUS_APPLICATION_WEDNESDAY', 'FLAG_LAST_APPL_PER_CONTRACT_N', 'FLAG_LAST_APPL
_PER_CONTRACT_Y', 'NAME_CASH_LOAN_PURPOSE_Building a house or an annex', 'NAME_CASH_LOAN_
```

```
PURPOSE_Business development', 'NAME_CASH_LOAN_PURPOSE_Buying a garage', 'NAME_CASH_LOAN_
PURPOSE_Buying a holiday home / land', 'NAME_CASH_LOAN_PURPOSE_Buying a home', 'NAME_CASH
_LOAN_PURPOSE_Buying a new car', 'NAME_CASH_LOAN_PURPOSE_Buying a used car', 'NAME_CASH_L
OAN_PURPOSE_Car repairs', 'NAME_CASH_LOAN_PURPOSE_Education', 'NAME_CASH_LOAN_PURPOSE_Eve
ryday expenses', 'NAME_CASH_LOAN_PURPOSE_Furniture', 'NAME_CASH_LOAN_PURPOSE_Gasification
/ water supply', 'NAME_CASH_LOAN_PURPOSE_Hobby', 'NAME_CASH_LOAN_PURPOSE_Journey', 'NAME_
CASH_LOAN_PURPOSE_Medicine', 'NAME_CASH_LOAN_PURPOSE_Money for a third person', 'NAME_CAS
H_LOAN_PURPOSE_Other', 'NAME_CASH_LOAN_PURPOSE_Payments on other loans', 'NAME_CASH_LOAN_
PURPOSE_Purchase of electronic equipment', 'NAME_CASH_LOAN_PURPOSE_Refusal to name the go
al', 'NAME_CASH_LOAN_PURPOSE_Repairs', 'NAME_CASH_LOAN_PURPOSE_Urgent needs', 'NAME_CASH_
LOAN_PURPOSE_Wedding / gift / holiday', 'NAME_CASH_LOAN_PURPOSE_XAP', 'NAME_CASH_LOAN_PUR
POSE_XNA', 'NAME_CONTRACT_STATUS_Approved', 'NAME_CONTRACT_STATUS_Canceled', 'NAME_CONTRA
CT_STATUS_Refused', 'NAME_CONTRACT_STATUS_Unused offer', 'NAME_PAYMENT_TYPE_Cash through
the bank', 'NAME_PAYMENT_TYPE_Cashless from the account of the employer', 'NAME_PAYMENT_T
YPE_Non-cash from your account', 'NAME_PAYMENT_TYPE_XNA', 'CODE_REJECT_REASON_CLIENT', 'C
ODE_REJECT_REASON_HC', 'CODE_REJECT_REASON_LIMIT', 'CODE_REJECT_REASON_SCO', 'CODE_REJECT
_REASON_SCOFR', 'CODE_REJECT_REASON_SYSTEM', 'CODE_REJECT_REASON_VERIF', 'CODE_REJECT_REA
SON_XAP', 'CODE_REJECT_REASON_XNA', 'NAME_TYPE_SUITE_PREVIOUS_APPLICATION_Children', 'NAM
E_TYPE_SUITE_PREVIOUS_APPLICATION_Family', 'NAME_TYPE_SUITE_PREVIOUS_APPLICATION_Group of
people', 'NAME_TYPE_SUITE_PREVIOUS_APPLICATION_Other_A', 'NAME_TYPE_SUITE_PREVIOUS_APPLIC
ATION_Other_B', 'NAME_TYPE_SUITE_PREVIOUS_APPLICATION_Spouse, partner', 'NAME_TYPE_SUITE_
PREVIOUS_APPLICATION_Unaccompanied', 'NAME_CLIENT_TYPE_New', 'NAME_CLIENT_TYPE_Refreshed'
, 'NAME_CLIENT_TYPE_Repeater', 'NAME_CLIENT_TYPE_XNA', 'NAME_GOODS_CATEGORY_Additional Se
rvice', 'NAME_GOODS_CATEGORY_Animals', 'NAME_GOODS_CATEGORY_Audio/Video', 'NAME_GOODS_CAT
EGORY_Auto Accessories', 'NAME_GOODS_CATEGORY_Clothing and Accessories', 'NAME_GOODS_CATE
GORY_Computers', 'NAME_GOODS_CATEGORY_Construction Materials', 'NAME_GOODS_CATEGORY_Consu
mer Electronics', 'NAME_GOODS_CATEGORY_Direct Sales', 'NAME_GOODS_CATEGORY_Education', 'N
AME_GOODS_CATEGORY_Fitness', 'NAME_GOODS_CATEGORY_Furniture', 'NAME_GOODS_CATEGORY_Garden
ing', 'NAME_GOODS_CATEGORY_Homewares', 'NAME_GOODS_CATEGORY_House Construction', 'NAME_GO
ODS_CATEGORY_Insurance', 'NAME_GOODS_CATEGORY_Jewelry', 'NAME_GOODS_CATEGORY_Medical Supp
lies', 'NAME_GOODS_CATEGORY_Medicine', 'NAME_GOODS_CATEGORY_Mobile', 'NAME_GOODS_CATEGORY
_Office Appliances', 'NAME_GOODS_CATEGORY_Other', 'NAME_GOODS_CATEGORY_Photo / Cinema Equ
ipment', 'NAME_GOODS_CATEGORY_Sport and Leisure', 'NAME_GOODS_CATEGORY_Tourism', 'NAME_GO
ODS_CATEGORY_Vehicles', 'NAME_GOODS_CATEGORY_Weapon', 'NAME_GOODS_CATEGORY_XNA', 'NAME_PO
RTFOLIO_Cards', 'NAME_PORTFOLIO_Cars', 'NAME_PORTFOLIO_Cash', 'NAME_PORTFOLIO_POS', 'NAME
_PORTFOLIO_XNA', 'NAME_PRODUCT_TYPE_XNA', 'NAME_PRODUCT_TYPE_walk-in', 'NAME_PRODUCT_TYPE
_x-sell', 'CHANNEL_TYPE_AP+ (Cash loan)', 'CHANNEL_TYPE_Car dealer', 'CHANNEL_TYPE_Channe
l of corporate sales', 'CHANNEL_TYPE_Contact center', 'CHANNEL_TYPE_Country-wide', 'CHANN
EL_TYPE_Credit and cash offices', 'CHANNEL_TYPE_Regional / Local', 'CHANNEL_TYPE_Stone',
'NAME_SELLER_INDUSTRY_Auto technology', 'NAME_SELLER_INDUSTRY_Clothing', 'NAME_SELLER_IND
USTRY_Connectivity', 'NAME_SELLER_INDUSTRY_Construction', 'NAME_SELLER_INDUSTRY_Consumer
electronics', 'NAME_SELLER_INDUSTRY_Furniture', 'NAME_SELLER_INDUSTRY_Industry', 'NAME_SE
LLER_INDUSTRY_Jewelry', 'NAME_SELLER_INDUSTRY_MLM partners', 'NAME_SELLER_INDUSTRY_Touris
m', 'NAME_SELLER_INDUSTRY_XNA', 'NAME_YIELD_GROUP_XNA', 'NAME_YIELD_GROUP_high', 'NAME_YI
ELD_GROUP_low_action', 'NAME_YIELD_GROUP_low_normal', 'NAME_YIELD_GROUP_middle', 'PRODUCT
_COMBINATION_Card Street', 'PRODUCT_COMBINATION_Card X-Sell', 'PRODUCT_COMBINATION_Cash',
'PRODUCT_COMBINATION_Cash Street: high', 'PRODUCT_COMBINATION_Cash Street: low', 'PRODUCT
_COMBINATION_Cash Street: middle', 'PRODUCT_COMBINATION_Cash X-Sell: high', 'PRODUCT_COMB
INATION_Cash X-Sell: low', 'PRODUCT_COMBINATION_Cash X-Sell: middle', 'PRODUCT_COMBINATIO
N_POS household with interest', 'PRODUCT_COMBINATION_POS household without interest', 'PR
ODUCT_COMBINATION_POS industry with interest', 'PRODUCT_COMBINATION_POS industry without
interest', 'PRODUCT_COMBINATION_POS mobile with interest', 'PRODUCT_COMBINATION_POS mobil
e without interest', 'PRODUCT_COMBINATION_POS other with interest', 'PRODUCT_COMBINATION_
POS others without interest']
```

Define the final raw dataset

In [ ]:

```python
'''In this code cell final raw data is printed'''

#Define the final combined dataset
application_train_final = application_train_2
application_test_final = application_test_2

#Define target values
target = application_train_final["TARGET"]

#Print the shape of final combined dataset
print(application_train_final.shape)
print(application_test_final.shape)
```

```
#Find and print number of data points in application_train_final and application_test_fin
al
train_len = len(application_train_final)
test_len = len(application_test_final)
print("Number of data points in application_train_final: ", train_len)
print("Number of data points in application_test_final: ", test_len)
```

```
(307511, 322)
(48744, 321)
Number of data points in application_train_final:  307511
Number of data points in application_test_final:  48744
```

5.0 Data Preparation: One hot encoding, imputation and standard scaling

Following actions are performed in this section:

- **One hot encoding of categorical columns after vertical concatanation of application_train_final & application_test_final**
- **Split the one hot encoded data into application_train_final_ohe and application_test_final_ohe**
- **Imputation and standard scaling**
- **Create a restore point**

5.1 One hot encoding of categorical data after combining train and test data

In [ ]:

```
'''In this code cell application_train_final and application_test_final are vertically
concatanated for one hot encoding'''

#Combine application_train_final and application_test_final for one hot encoding
combined_train_test = pd.concat([application_train_final.drop(columns = ['TARGET']), appl
ication_test_final], axis = 0, ignore_index=True)
print(combined_train_test.shape)
```

```
(356255, 321)
```

In [ ]:

```
'''In this code cell one hot encoding is performed on above data'''

#Perform one hot encoding on final combined data data
ohe_column = combined_train_test.select_dtypes(include=object).columns
none_ohe_column = set(combined_train_test.columns) - set(ohe_column)

#Prepare datasets
ohe = pd.get_dummies(combined_train_test[ohe_column])
none_ohe = combined_train_test[none_ohe_column]

#Merge datasets
combined_train_test_ohe = pd.concat([ohe, none_ohe], axis = 1)

#Print the shape of combined_train_test_ohe
print(combined_train_test_ohe.shape)
```

```
(356255, 445)
```

In [ ]:

```
'''In this code cell list of categorical column in updated'''

#Update categorical_column
categorical_column = list(set(combined_train_test_ohe.columns) - set(['SK_ID_CURR']) - s
et(numerical_column))
```

In [ ]:

```
'''In this code cell one hot encoded data is split into application_train_final_ohe and
application_test_final_ohe'''

#Split into original train and test data
application_train_final_ohe = combined_train_test_ohe[0:train_len]
application_test_final_ohe = combined_train_test_ohe[train_len:train_len + test_len]

#Print the shape of one hot encoded data
print(application_train_final_ohe.shape)
print(application_test_final_ohe.shape)
```

```
(307511, 445)
(48744, 445)
```

5.2 Split the data obtained after one hot encoding of application_train - application_train_final_ohe - into train, validate and test data

In [ ]:

```
'''In this code cell application_train_final and target are split into training, validati
on and test datasets'''

X_train, X_validate_test, y_train, y_validate_test = train_test_split(application_train_
final_ohe.drop(columns = ['SK_ID_CURR']), target, stratify = target, test_size=0.3, rand
om_state=42)
X_validate, X_test, y_validate, y_test = train_test_split(X_validate_test, y_validate_te
st, stratify = y_validate_test, test_size=0.5, random_state=42)
print('Shape of X_train or training dataset: ', X_train.shape)
print('Shape of X_validate or validation dataset: ', X_validate.shape)
print('Shape of X_test or test dataset: ', X_test.shape)
```

```
Shape of X_train or training dataset:  (215257, 444)
Shape of X_validate or validation dataset:  (46127, 444)
Shape of X_test or test dataset:  (46127, 444)
```

5.3 Imputation and standard scaling

In [ ]:

```
'''In this code cell imputation and standard scaling of numerical data for X_train,
X_validate and X_test'''

#Imputation of numerical data
imputer_numerical = SimpleImputer(strategy='median')
X_train_numerical_impute = imputer_numerical.fit_transform(X_train[numerical_column])
X_validate_numerical_impute = imputer_numerical.transform(X_validate[numerical_column])
X_test_numerical_impute = imputer_numerical.transform(X_test[numerical_column])

#Standardization of numerical data
standard_numerical = StandardScaler()
X_train_numerical_impute_standard = standard_numerical.fit_transform(X_train_numerical_im
pute)
X_validate_numerical_impute_standard = standard_numerical.transform(X_validate_numerical_
impute)
X_test_numerical_impute_standard = standard_numerical.transform(X_test_numerical_impute)

#Convert above data to datframe
X_train_numerical = pd.DataFrame(X_train_numerical_impute_standard, columns=numerical_col
umn)
X_validate_numerical = pd.DataFrame(X_validate_numerical_impute_standard, columns=numeri
cal_column)
X_test_numerical = pd.DataFrame(X_test_numerical_impute_standard, columns=numerical_colu
mn)

#Print shape of above data
```

```
print(X_train_numerical.shape)
print(X_validate_numerical.shape)
print(X_test_numerical.shape)
```

```
(215257, 138)
(46127, 138)
(46127, 138)
```

In [ ]:

```
'''In this code cell imputation and standard scaling of numerical data for application_te
st_final_ohe'''

#Imputation of numerical data
imputer_numerical = SimpleImputer(strategy='median')
application_test_final_ohe_numerical_impute = imputer_numerical.fit_transform(application
_test_final_ohe[numerical_column])

#Standardization of numerical data
standard_numerical = StandardScaler()
application_test_final_ohe_numerical_impute_standard = standard_numerical.fit_transform(a
pplication_test_final_ohe_numerical_impute)

#Convert above data to datframe
application_test_final_ohe_numerical = pd.DataFrame(application_test_final_ohe_numerical_
impute_standard, columns=numerical_column)

#Print shape of above data
print(application_test_final_ohe_numerical.shape)
```

```
(48744, 138)
```

In [ ]:

```
'''In this code cell categorical data for merging with imputed and standardised numerical

data is extracted for X_train, X_validate and X_test'''

#Extract categorical data for merging with imputed and standardised numerical data
X_train_categorical = X_train[categorical_column]
X_validate_categorical = X_validate[categorical_column]
X_test_categorical = X_test[categorical_column]

#Print shape of above data
print(X_train_categorical.shape)
print(X_validate_categorical.shape)
print(X_test_categorical.shape)
```

```
(215257, 306)
(46127, 306)
(46127, 306)
```

In [ ]:

```
'''In this code cell categorical data for merging with imputed and standardised numerical
data is extracted'''

#Extract categorical data for merging with imputed and standardised numerical data for ap
plication_test_final_ohe
application_test_final_ohe_categorical = application_test_final_ohe[categorical_column]

#Print shape of above data
print(application_test_final_ohe_categorical.shape)
```

```
(48744, 306)
```

In [ ]:

```
'''In this code cell application_test_final_ohe_numerical and application_test_final_ohe_
categorical
are horizontally concatanated'''
```

```
#Combine application_test_final_ohe_numerical and application_test_final_ohe_categorical
application_test_final_ohe_combined = pd.concat([application_test_final_ohe_numerical.res
et_index(drop=True), application_test_final_ohe_categorical.reset_index(drop=True)], axis
= 1)
print(application_test_final_ohe_combined.shape)
```

```
(48744, 444)
```

In [ ]:

```
'''In this code cell numerical and calegorical data are combined to get prepared data'''

#Combine numerical and categorical data to get complete dataset
X_train_final = pd.concat([X_train_numerical.reset_index(drop=True), X_train_categorical
.reset_index(drop=True)], axis = 1)
X_validate_final = pd.concat([X_validate_numerical.reset_index(drop=True), X_validate_ca
tegorical.reset_index(drop=True)], axis = 1)
X_test_final = pd.concat([X_test_numerical.reset_index(drop=True), X_test_categorical.re
set_index(drop=True)], axis = 1)

#Print shape of complete dataset
print(X_train_final.shape)
print(X_validate_final.shape)
print(X_test_final.shape)
print(y_train.shape)
print(y_validate.shape)
print(y_test.shape)
```

```
(215257, 444)
(46127, 444)
(46127, 444)
(215257,)
(46127,)
(46127,)
```

In [ ]:

```
'''In this code cell two dataframes with names of columns which were originally
numerical and originally categorical are created.'''

numerical = {'COLUMN': numerical_column}
NUMERICAL_COLUMN = pd.DataFrame(numerical)
categorical = {'COLUMN': categorical_column}
CATEGORICAL_COLUMN = pd.DataFrame(categorical)
```

5.4 Create a restore point

Datasets generated at this point are saved as csv files and uploaded to google drive for future use. This is done because of RAM limitation on google colaboratory. Also it allows to run diffent sections independently.

In [ ]:

```
'''In this code cell datframes with names of original numerical and categorical
columns are saved as csv files for future reference. These csv files are uploaded
to google drive.'''

#Save the dataframes into CSV files for future use
NUMERICAL_COLUMN.to_csv('NUMERICAL_COLUMN.csv', index = False)
files.download("NUMERICAL_COLUMN.csv")
CATEGORICAL_COLUMN.to_csv('CATEGORICAL_COLUMN.csv', index = False)
files.download("CATEGORICAL_COLUMN.csv")
```

In [ ]:

```
'''In this code cell feature encoded data (data ready for further mathematical operations
)
are saved to csv file and downloaded.'''
```

```
#Save the dataframes into CSV files for future use
X_train_final.to_csv('X_train_final.csv', index = False)
files.download("X_train_final.csv")
X_validate_final.to_csv('X_validate_final.csv', index = False)
files.download("X_validate_final.csv")
X_test_final.to_csv('X_test_final.csv', index = False)
files.download("X_test_final.csv")
y_train.to_csv('y_train.csv', index = False)
files.download("y_train.csv")
y_validate.to_csv('y_validate.csv', index = False)
files.download("y_validate.csv")
y_test.to_csv('y_test.csv', index = False)
files.download("y_test.csv")
application_test_final_ohe_combined.to_csv('application_test_final_ohe_combined.csv', ind
ex = False)
files.download("application_test_final_ohe_combined.csv")
```

6.0 Outlier detection and removal

Following actions are performed in this section:

- **Required data is imported.**
- **Box plot for annual income is plotted before outlier removal.**
- **Outlier removal and detection is done.**
- **Box plot for annual income is plotted after outlier removal.**
- **A restore point is created.**

6.1 Import data

In [ ]:

```
'''In this code cell data from all the csv files are imported'''

#Read X_train_final
X_train_final = pd.read_csv('/content/drive/MyDrive/AI_ML_Project/Data/X_train_final.csv'
)

#Read X_validate_final
X_validate_final = pd.read_csv('/content/drive/MyDrive/AI_ML_Project/Data/X_validate_fina
l.csv')

#Read X_test_final
X_test_final = pd.read_csv('/content/drive/MyDrive/AI_ML_Project/Data/X_test_final.csv')

#Read y_train
y_train = pd.read_csv('/content/drive/MyDrive/AI_ML_Project/Data/y_train.csv')

#Read y_validate
y_validate = pd.read_csv('/content/drive/MyDrive/AI_ML_Project/Data/y_validate.csv')

#Read y_test
y_test = pd.read_csv('/content/drive/MyDrive/AI_ML_Project/Data/y_test.csv')

#Read application_test_final_ohe_combined
application_test_final_ohe_combined = pd.read_csv('/content/drive/MyDrive/AI_ML_Project/D
ata/application_test_final_ohe_combined.csv')
```

In [ ]:

```
'''In this code cell shapes of imported datatables are printed'''

#Print shapes of imported datasets
print(X_train_final.shape)
print(X_validate_final.shape)
print(X_test_final.shape)
print(y_train.shape)
print(y_validate.shape)
print(y_test.shape)
print(application_test_final_ohe_combined.shape)
```

```
(215257, 444)
(46127, 444)
(46127, 444)
(215257, 1)
(46127, 1)
(46127, 1)
(48744, 444)
```

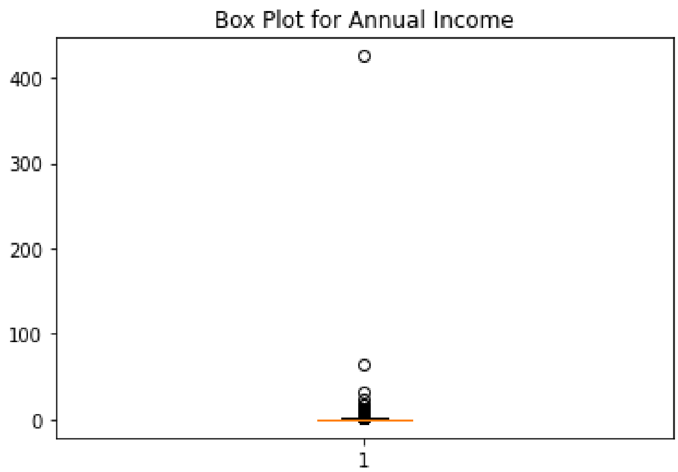
6.2 Box plot for annual income before outlier removal

In [ ]:

```
'''In this code cell box plot for AMT_INCOME_TOTAL of X_train_final is plotted before out
lier
detection and removal'''

fig1, ax1 = plt.subplots()
ax1.set_title('Box Plot for Annual Income')
ax1.boxplot(X_train_final['AMT_INCOME_TOTAL'])
```

Out[ ]:

```
{'boxes': [<matplotlib.lines.Line2D at 0x7fa36eafde50>],
 'caps': [<matplotlib.lines.Line2D at 0x7fa36eae8f10>,
  <matplotlib.lines.Line2D at 0x7fa36eaec490>],
 'fliers': [<matplotlib.lines.Line2D at 0x7fa36eaecf50>],
 'means': [],
 'medians': [<matplotlib.lines.Line2D at 0x7fa36eaeca10>],
 'whiskers': [<matplotlib.lines.Line2D at 0x7fa36eae8490>,
  <matplotlib.lines.Line2D at 0x7fa36eae89d0>]}
```



> **Based on the above plot, it is observed that there are a lot of outliers.**

6.3 LOF (Local Outlier Factor) based outlier detection and removal

In [ ]:

In [ ]:

```
'''In this code cell LOF based outlier detection model from pyod called CBLOF is defined
and fit upon X_train_final. Outliers are detected and a column named outlier is created
which indicated whether adatapoint is outlier or inlier.'''

#Define outlier detector and fit it to X_train_final with contamination = 0.05
clf = CBLOF(contamination=0.05, check_estimator=False, random_state=42)
clf.fit(X_train_final)
scores_pred = clf.decision_function(X_train_final) * -1

#Predict the datapoints as outlier or inlier
outlier_prediction = clf.predict(X_train_final)
inliers = len(outlier_prediction) - np.count_nonzero(outlier_prediction)
outliers = np.count_nonzero(outlier_prediction == 1)

#Create dataframe named X_train_final_outlier with all the columns of X_train_final and a
n
#additional column showing whether a datapoint is outlier or not
X_train_final_outlier = X_train_final.copy()
X_train_final_outlier['outlier'] = outlier_prediction.tolist()
```

In [ ]:

```
'''In this code cell new dataframes are created after removing outliers'''

#Create new training dataset named X_train_final_outlier_removed and new target values
#named y_train_outlier_removed with outliers removed
y_train_outlier = y_train.copy()
X_y_train_final_outlier = pd.concat([X_train_final_outlier, y_train_outlier], axis = 1)
X_y_train_final_outlier_removed = X_y_train_final_outlier[X_y_train_final_outlier['outlie
r'] != 1]
X_train_final_outlier_removed = X_y_train_final_outlier_removed.drop(columns = ['TARGET',
'outlier'])
y_train_final_outlier_removed = X_y_train_final_outlier_removed['TARGET']
```

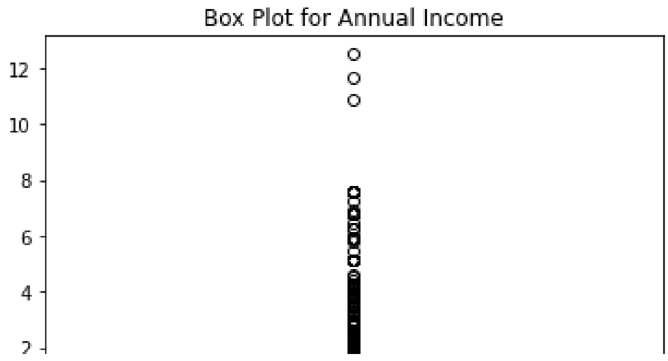
6.4 Box plot for annual income before outlier removal

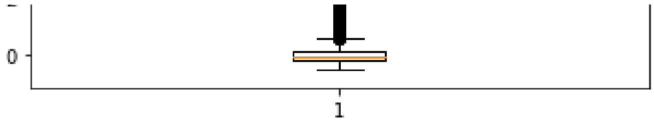
In [ ]:

```
'''In this code cell box plot for AMT_INCOME_TOTAL of X_train_final is plotted after outl
ier
detection and removal'''

#Plot box plot for AMT_INCOME_TOTAL of X_train_final_outlier_removed
fig1, ax1 = plt.subplots()
ax1.set_title('Box Plot for Annual Income')
ax1.boxplot(X_train_final_outlier_removed['AMT_INCOME_TOTAL'])
```

Out[ ]:

```
{'boxes': [<matplotlib.lines.Line2D at 0x7fa35f382fd0>],
 'caps': [<matplotlib.lines.Line2D at 0x7fa35f38f0d0>,
  <matplotlib.lines.Line2D at 0x7fa35f38f610>],
 'fliers': [<matplotlib.lines.Line2D at 0x7fa35f396110>],
 'means': [],
 'medians': [<matplotlib.lines.Line2D at 0x7fa35f38fb90>],
 'whiskers': [<matplotlib.lines.Line2D at 0x7fa35f38b610>,
  <matplotlib.lines.Line2D at 0x7fa35f38bb50>]}
```





> **It is observed that the box plot has improved significantly after outlier removal.**

In [ ]:

```
'''This code cell gives us the percentage of loans where default has been observed and
percentage of loan where no default has been observed'''

total_0 = len(X_y_train_final_outlier_removed[X_y_train_final_outlier_removed['TARGET'] =
= 0])
total_1 = len(X_y_train_final_outlier_removed[X_y_train_final_outlier_removed['TARGET'] =
= 1])
print("Percentage of TARGET 0: ", '%.2f'%(total_0*100/(total_0 + total_1)))
print("Percentage of TARGET 1: ", '%.2f'%(total_1*100/(total_0 + total_1)))
```

```
Percentage of TARGET 0:  91.82
Percentage of TARGET 1:  8.18
```

> **No significant change in percentage of Target 0 and Target 1 values is observed with removal of outliers.**

6.5 Create a restore point

Datasets generated at this point are saved as csv files and uploaded to google drive for future use. This is done because of RAM limitation on google colaboratory. Also it allows to run diffent sections independently.

In [ ]:

```
'''In this code cell final data sets with outliers removed are saved and downloaded as cs
v files.
These are uploaded to google drive for future access.'''

#Save the dataframes into CSV files for future use
X_train_final_outlier_removed.to_csv('X_train_final_outlier_removed.csv', index = False)
files.download("X_train_final_outlier_removed.csv")
y_train_final_outlier_removed.to_csv('y_train_final_outlier_removed.csv', index = False)
files.download("y_train_final_outlier_removed.csv")
```

7.0 Feature selection

Following actions are performed in this section:

- **Import relevant data and print their shape**
- **XGBoost based feature importance**
- **Gradient Boosting based feature importance**
- **Feature selection from top 225 features based on XGBoost and Gradient Boosting**
- **Create a restore point**

7.1 Import data and print shape

In [ ]:

```
'''In this code cell data from the required csv files are imported'''

#Read X_train_final_feature_selected
X_train_final_outlier_removed = pd.read_csv('/content/drive/MyDrive/AI_ML_Project/Data/X_
train_final_outlier_removed.csv')

#Read y_train_final_feature_selected
y_train_final_outlier_removed = pd.read_csv('/content/drive/MyDrive/AI_ML_Project/Data/y_
train_final_outlier_removed.csv')

#Read X_validate_final
X_validate_final = pd.read_csv('/content/drive/MyDrive/AI_ML_Project/Data/X_validate_fina
l.csv')

#Read y_validate
y_validate = pd.read_csv('/content/drive/MyDrive/AI_ML_Project/Data/y_validate.csv')

#Read X_test_final
X_test_final = pd.read_csv('/content/drive/MyDrive/AI_ML_Project/Data/X_test_final.csv')

#Read y_test
y_test = pd.read_csv('/content/drive/MyDrive/AI_ML_Project/Data/y_test.csv')

#Read application_test_final_ohe_combined
application_test_final_ohe_combined = pd.read_csv('/content/drive/MyDrive/AI_ML_Project/D
ata/application_test_final_ohe_combined.csv')

#Read NUMERICAL_COLUMN
numerical_col = pd.read_csv('/content/drive/MyDrive/AI_ML_Project/Data/NUMERICAL_COLUMN.c
sv')
```

In [ ]:

```
'''In this code cell shapes of datasets imported above are printed'''

print(X_train_final_outlier_removed.shape)
print(y_train_final_outlier_removed.shape)
print(X_validate_final.shape)
print(y_validate.shape)
print(X_test_final.shape)
print(y_test.shape)
print(application_test_final_ohe_combined.shape)
print(numerical_col.shape)
```

```
(204494, 444)
(204494, 1)
(46127, 444)
(46127, 1)
(46127, 444)
(46127, 1)
(48744, 444)
(138, 1)
```

7.2 XGBoost based feature slecetion

In [ ]:

```
'''In this code cell XGBoost based classifier is defined and fit on X_train_final_outlier
_removed
and y_train_final_outlier_removed'''

xgb = XGBClassifier(n_estimators=100, random_state=42)
xgb.fit(X_train_final_outlier_removed, y_train_final_outlier_removed)
```

```
/usr/local/lib/python3.7/dist-packages/sklearn/preprocessing/_label.py:98: DataConversion
Warning: A column-vector y was passed when a 1d array was expected. Please change the sha
pe of y to (n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)
/usr/local/lib/python3.7/dist-packages/sklearn/preprocessing/_label.py:133: DataConversio
nWarning: A column-vector y was passed when a 1d array was expected. Please change the sh
```

```
ape of y to (n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)
```

Out[ ]:

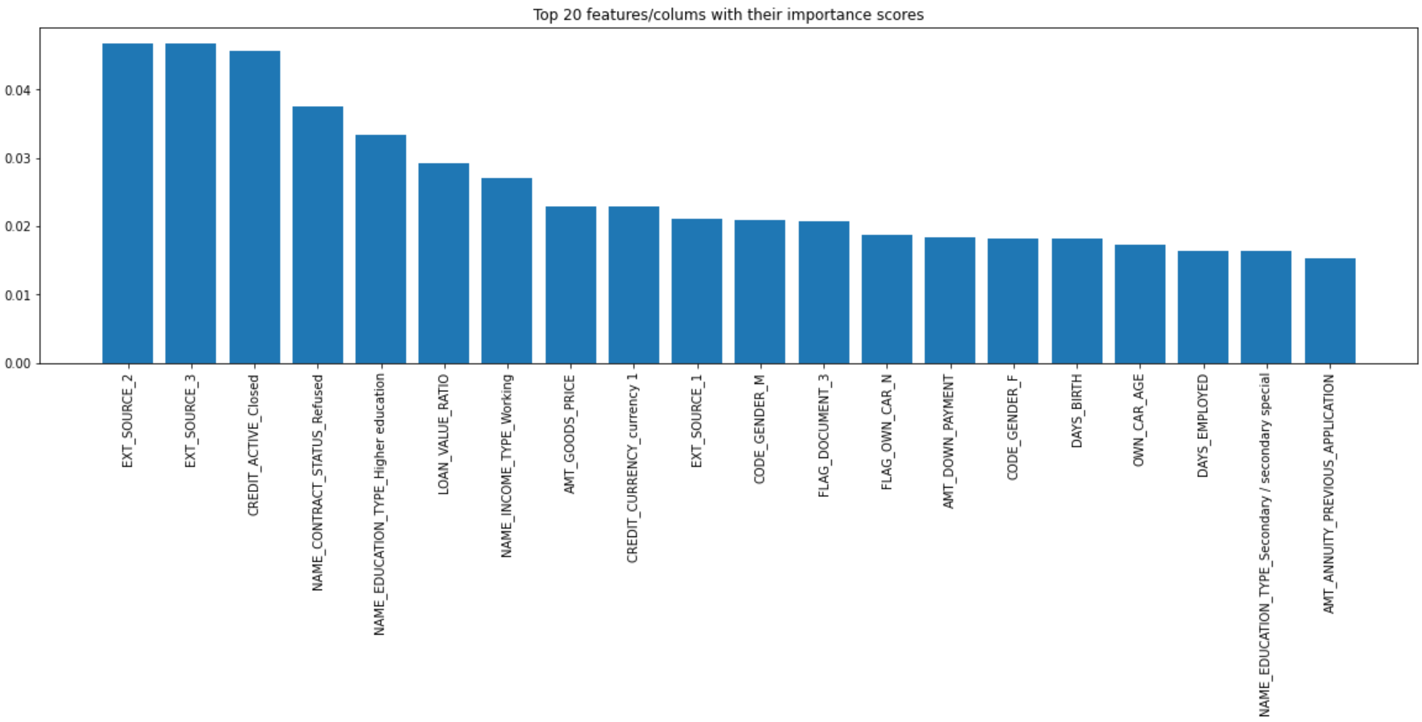
```
XGBClassifier(random_state=42)
```

In [ ]:

```
'''In this code cell dataframe consisting of columns and their importance is created.
This data frame is further sorted in descending order of feature importance.'''

#Create dataframe
data1 = {'COLUMN': X_train_final_outlier_removed.columns,
        'SCORE': xgb.feature_importances_}
Column_Score1 = pd.DataFrame(data1)

#Sort dataframe by decreasing importance scores
Column_Score_sorted1 = Column_Score1.sort_values(by='SCORE', ascending=False)
```

In [ ]:

```
'''In this code cell a graph plot of top 20 features and their importance is plotted.'''

column1 = Column_Score_sorted1['COLUMN'][0:20]
score1 = Column_Score_sorted1['SCORE'][0:20]

plt.figure(figsize =(20, 5))
plt.bar(column1, score1)
plt.xticks(rotation=90)
plt.title('Top 20 features/colums with their importance scores')
plt.show()
```



In [ ]:

```
'''In this code cell top 225 features are selected based on XGBoost'''

feature_selected_xgb = Column_Score_sorted1["COLUMN"][0:225]
```

7.3 Gradient Boosting based feature slecetion

In [ ]:

```
'''In this code cell Grradient Boosting based classifier is defined and fit on
X_train_final_outlier_removed and y_train_final_outlier_removed'''
```

```
gb = GradientBoostingClassifier(n_estimators=100)
gb.fit(X_train_final_outlier_removed, y_train_final_outlier_removed)
```

```
/usr/local/lib/python3.7/dist-packages/sklearn/ensemble/_gb.py:494: DataConversionWarning
: A column-vector y was passed when a 1d array was expected. Please change the shape of y
to (n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)
```

Out[ ]:

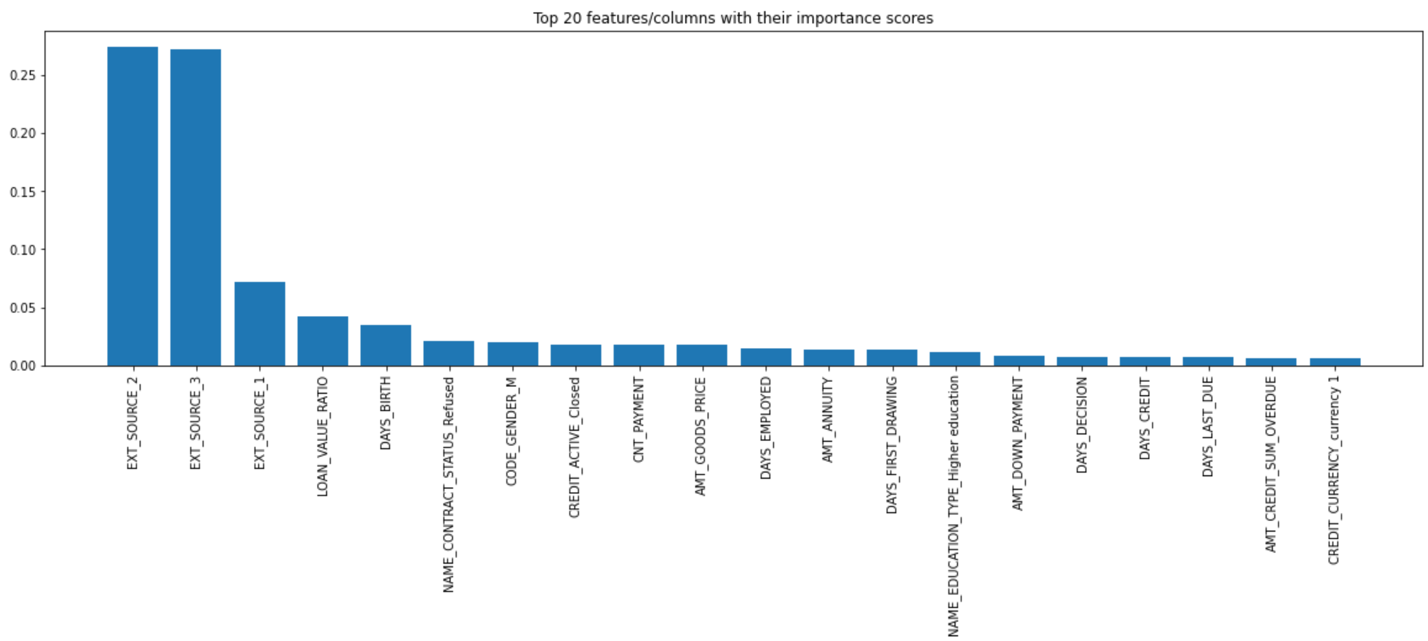
```
GradientBoostingClassifier()
```

In [ ]:

```
'''In this code cell dataframe consisting of columns and their importance is created.
This data frame is further sorted in descending order of feature importance.'''

#Create dataframe
data2 = {'COLUMN': X_train_final_outlier_removed.columns,
         'SCORE': gb.feature_importances_}
Column_Score2 = pd.DataFrame(data2)

#Sort dataframe by decreasing importance scores
Column_Score_sorted2 = Column_Score2.sort_values(by='SCORE', ascending=False)
```

In [ ]:

```
'''In this code cell a graph plot of top 20 features and their importance is plotted.'''

column2 = Column_Score_sorted2['COLUMN'][0:20]
score2 = Column_Score_sorted2['SCORE'][0:20]

plt.figure(figsize =(20, 5))
plt.bar(column2, score2)
plt.xticks(rotation=90)
plt.title('Top 20 features/columns with their importance scores')
plt.show()
```



In [ ]:

```
'''In this code cell top 225 features are selected based on GBDT'''

feature_selected_gbdt = Column_Score_sorted2["COLUMN"][0:225]
```

7.4 Feature selection

In [ ]:

```
'''In this code cell features common in top 225 features selected based on ExtraClassifie
r
and XGBoost are put in a list and its length is printed'''

feature_selected = list(set(feature_selected_gbdt) & set(feature_selected_xgb))
print(len(feature_selected))
```

```
176
```

In [ ]:

```
'''In this code cell selected features are printed as list'''

print(list(feature_selected))
```

```
['NAME_HOUSING_TYPE_Municipal apartment', 'ORGANIZATION_TYPE_Construction', 'REGION_RATIN
G_CLIENT_W_CITY', 'ORGANIZATION_TYPE_Trade: type 3', 'NAME_CASH_LOAN_PURPOSE_Buying a hom
e', 'NAME_FAMILY_STATUS_Married', 'ORGANIZATION_TYPE_Industry: type 13', 'ORGANIZATION_TY
PE_Industry: type 8', 'ORGANIZATION_TYPE_Transport: type 3', 'NAME_CLIENT_TYPE_Refreshed'
, 'DAYS_BIRTH', 'WEEKDAY_APPR_PROCESS_START_PREVIOUS_APPLICATION_SATURDAY', 'CREDIT_TYPE_
Microloan', 'YEARS_BEGINEXPLUATATION_AVG', 'FLAG_LAST_APPL_PER_CONTRACT_Y', 'HOUSETYPE_MO
DE_terraced house', 'NAME_GOODS_CATEGORY_Gardening', 'AMT_ANNUITY', 'ORGANIZATION_TYPE_Cu
lture', 'WEEKDAY_APPR_PROCESS_START_SATURDAY', 'NAME_HOUSING_TYPE_With parents', 'CODE_GE
NDER_XNA', 'CREDIT_ACTIVE_Sold', 'WEEKDAY_APPR_PROCESS_START_PREVIOUS_APPLICATION_MONDAY'
, 'AMT_REQ_CREDIT_BUREAU_QRT', 'PRODUCT_COMBINATION_POS household without interest', 'FLA
G_LAST_APPL_PER_CONTRACT_N', 'LOAN_INCOME_RATIO', 'AMT_CREDIT_MAX_OVERDUE', 'OCCUPATION_T
YPE_Private service staff', 'NAME_CASH_LOAN_PURPOSE_Purchase of electronic equipment', 'N
AME_TYPE_SUITE_PREVIOUS_APPLICATION_Children', 'AMT_GOODS_PRICE_PREVIOUS_APPLICATION', 'N
AME_CONTRACT_STATUS_Canceled', 'NAME_CASH_LOAN_PURPOSE_Building a house or an annex', 'DA
YS_LAST_DUE', 'NAME_GOODS_CATEGORY_XNA', 'NAME_CLIENT_TYPE_New', 'ORGANIZATION_TYPE_Indus
try: type 3', 'NAME_CASH_LOAN_PURPOSE_Furniture', 'CREDIT_CURRENCY_currency 2', 'AMT_APPL
ICATION', 'NAME_SELLER_INDUSTRY_Jewelry', 'WALLSMATERIAL_MODE_Panel', 'NAME_CONTRACT_TYPE
_PREVIOUS_APPLICATION_Revolving loans', 'NAME_INCOME_TYPE_State servant', 'AMT_CREDIT_PRE
VIOUS_APPLICATION', 'ORGANIZATION_TYPE_Telecom', 'FLAG_OWN_CAR_N', 'NAME_FAMILY_STATUS_Ci
vil marriage', 'CODE_REJECT_REASON_SCO', 'EXT_SOURCE_2', 'PRODUCT_COMBINATION_Cash Street
: low', 'NAME_INCOME_TYPE_Working', 'ORGANIZATION_TYPE_Trade: type 5', 'NAME_PAYMENT_TYPE
_Cashless from the account of the employer', 'NAME_EDUCATION_TYPE_Academic degree', 'ORGA
NIZATION_TYPE_Self-employed', 'FLAG_WORK_PHONE', 'WALLSMATERIAL_MODE_Others', 'NAME_TYPE_
SUITE_Group of people', 'NAME_GOODS_CATEGORY_Consumer Electronics', 'REG_CITY_NOT_LIVE_CI
TY', 'CHANNEL_TYPE_Car dealer', 'NAME_CASH_LOAN_PURPOSE_Urgent needs', 'NAME_CONTRACT_TYP
E_PREVIOUS_APPLICATION_Consumer loans', 'NAME_EDUCATION_TYPE_Secondary / secondary specia
l', 'NAME_GOODS_CATEGORY_Medicine', 'DAYS_REGISTRATION', 'CREDIT_TYPE_Real estate loan',
'ORGANIZATION_TYPE_Security', 'DAYS_LAST_PHONE_CHANGE', 'NAME_PRODUCT_TYPE_walk-in', 'AMT
_ANNUITY_PREVIOUS_APPLICATION', 'DAYS_DECISION', 'EXT_SOURCE_3', 'DAYS_FIRST_DUE', 'ORGAN
IZATION_TYPE_Trade: type 1', 'WEEKDAY_APPR_PROCESS_START_PREVIOUS_APPLICATION_SUNDAY', 'A
MT_CREDIT', 'PRODUCT_COMBINATION_POS industry with interest', 'NAME_SELLER_INDUSTRY_Touri
sm', 'PRODUCT_COMBINATION_Cash', 'NAME_CASH_LOAN_PURPOSE_Other', 'FLAG_DOCUMENT_3', 'OCCU
PATION_TYPE_Secretaries', 'NAME_GOODS_CATEGORY_Animals', 'AMT_CREDIT_SUM', 'NAME_TYPE_SUI
TE_PREVIOUS_APPLICATION_Family', 'NAME_CONTRACT_STATUS_Unused offer', 'NAME_EDUCATION_TYP
E_Higher education', 'CNT_CHILDREN', 'NAME_TYPE_SUITE_Other_B', 'CREDIT_TYPE_Mortgage', '
NONLIVINGAREA_AVG', 'CREDIT_CURRENCY_currency 1', 'ORGANIZATION_TYPE_Trade: type 6', 'DEF
_30_CNT_SOCIAL_CIRCLE', 'ORGANIZATION_TYPE_Industry: type 4', 'CNT_PAYMENT', 'NAME_CONTRA
CT_STATUS_Approved', 'NAME_TYPE_SUITE_Children', 'WEEKDAY_APPR_PROCESS_START_PREVIOUS_APP
LICATION_TUESDAY', 'AMT_DOWN_PAYMENT', 'WEEKDAY_APPR_PROCESS_START_MONDAY', 'NAME_FAMILY_
STATUS_Single / not married', 'NAME_HOUSING_TYPE_Rented apartment', 'FONDKAPREMONT_MODE_o
rg spec account', 'NAME_GOODS_CATEGORY_Tourism', 'NAME_CASH_LOAN_PURPOSE_XNA', 'DAYS_EMPL
OYED', 'NAME_CASH_LOAN_PURPOSE_Refusal to name the goal', 'NAME_CASH_LOAN_PURPOSE_Educati
on', 'SELLERPLACE_AREA', 'HOUR_APPR_PROCESS_START_PREVIOUS_APPLICATION', 'OCCUPATION_TYPE
_Core staff', 'NAME_GOODS_CATEGORY_Medical Supplies', 'NAME_SELLER_INDUSTRY_Industry', 'N
AME_INCOME_TYPE_Commercial associate', 'OCCUPATION_TYPE_Low-skill Laborers', 'NAME_CASH_L
OAN_PURPOSE_Hobby', 'PRODUCT_COMBINATION_Card Street', 'ORGANIZATION_TYPE_Industry: type
5', 'OCCUPATION_TYPE_Laborers', 'DEBT_INCOME_RATIO', 'ORGANIZATION_TYPE_Other', 'NAME_GOO
DS_CATEGORY_Insurance', 'NAME_TYPE_SUITE_Family', 'NAME_CONTRACT_TYPE_Cash loans', 'DEF_6
0_CNT_SOCIAL_CIRCLE', 'CREDIT_TYPE_Cash loan (non-earmarked)', 'ORGANIZATION_TYPE_Cleanin
g', 'NAME_HOUSING_TYPE_Co-op apartment', 'WEEKDAY_APPR_PROCESS_START_WEDNESDAY', 'NAME_TY
PE_SUITE_Unaccompanied', 'NAME_CASH_LOAN_PURPOSE_Medicine', 'NAME_SELLER_INDUSTRY_XNA', '
AMT_GOODS_PRICE', 'PRODUCT_COMBINATION_POS household with interest', 'PRODUCT_COMBINATION
_Card X-Sell', 'NAME_GOODS_CATEGORY_Audio/Video', 'REGION_POPULATION_RELATIVE', 'CREDIT_A
CTIVE_Closed', 'WEEKDAY_APPR_PROCESS_START_THURSDAY', 'NAME_CONTRACT_STATUS_Refused', 'DA
YS_ID_PUBLISH', 'NAME_CASH_LOAN_PURPOSE_XAP', 'ORGANIZATION_TYPE_Trade: type 2', 'NAME_PO
RTFOLIO_POS', 'NAME_YIELD_GROUP_high', 'CODE_REJECT_REASON_SCOFR', 'CODE_GENDER_M', 'NAME
_GOODS_CATEGORY_Photo / Cinema Equipment', 'LOAN_VALUE_RATIO', 'AMT_CREDIT_SUM_DEBT', 'CO
```

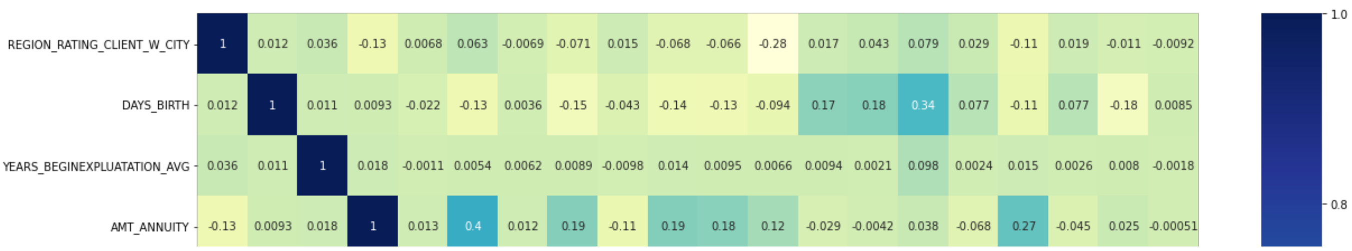
```
DE_GENDER_F', 'ORGANIZATION_TYPE_Agriculture', 'NAME_INCOME_TYPE_Unemployed', 'AMT_CREDIT
_SUM_OVERDUE', 'NAME_YIELD_GROUP_low_action', 'NAME_FAMILY_STATUS_Separated', 'NAME_EDUCA
TION_TYPE_Lower secondary', 'NAME_GOODS_CATEGORY_Computers', 'ORGANIZATION_TYPE_Universit
y', 'CREDIT_CURRENCY_currency 3', 'CREDIT_TYPE_Mobile operator loan', 'NAME_GOODS_CATEGOR
Y_Weapon', 'EXT_SOURCE_1', 'OWN_CAR_AGE', 'DAYS_FIRST_DRAWING', 'PRODUCT_COMBINATION_Cash
X-Sell: middle', 'CREDIT_TYPE_Loan for the purchase of equipment', 'PRODUCT_COMBINATION_P
OS industry without interest', 'DAYS_CREDIT_ENDDATE', 'FLAG_OWN_CAR_Y', 'DAYS_CREDIT']
```

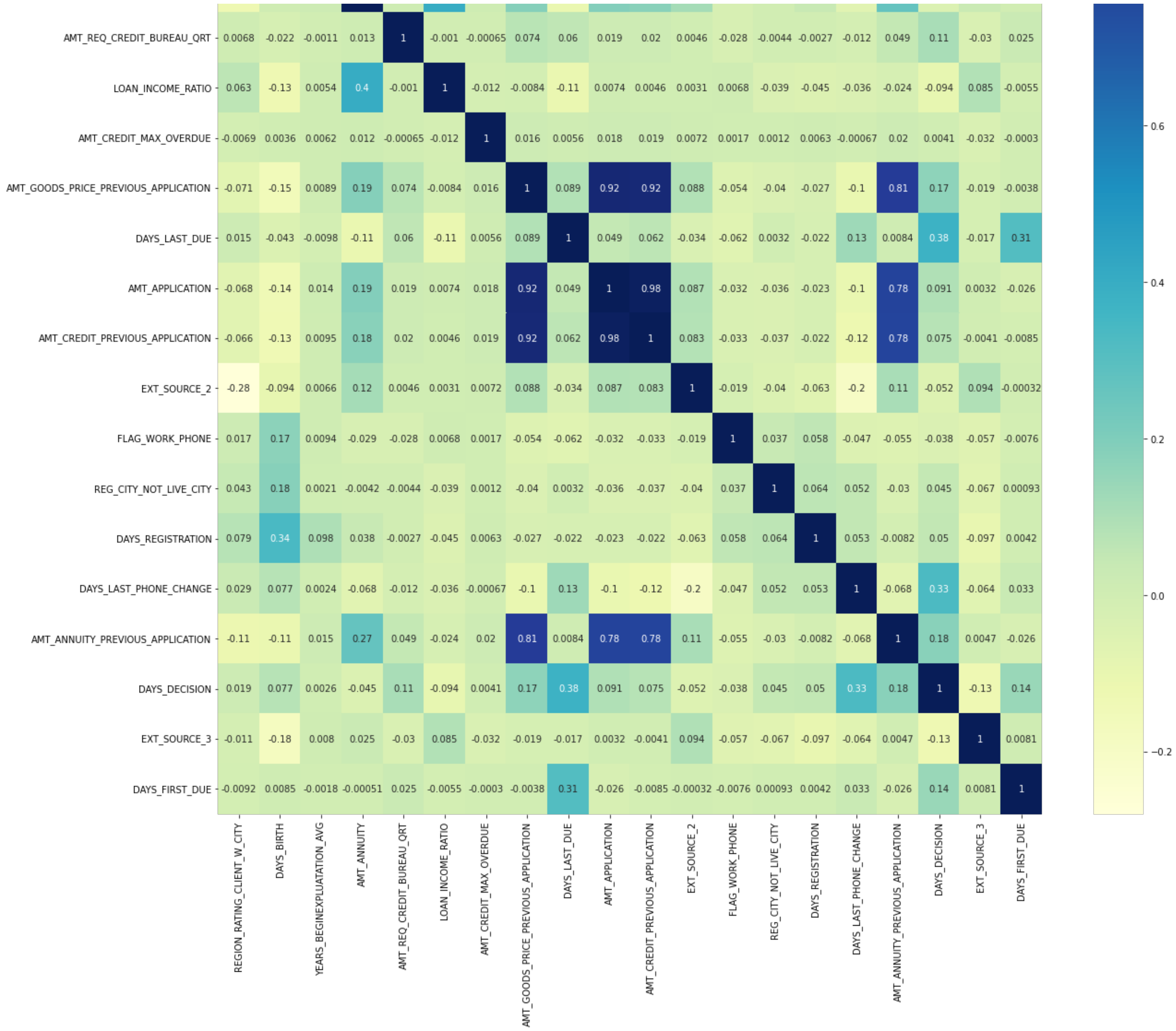
> **We observe that the additional features created - DEBT_INCOME_RATIO, LOAN_INCOME_RATIO, LOAN_VALUE_RATIO - during feature engineering figure in this list of selected features.**

In [ ]:

```
'''In this code cell heat map based on correlation matrix is plotted'''

#Get list of numerical columns in numerical
numerical = list(numerical_col["COLUMN"].values)

#A list consisting of selected columns which were originally numerical is created.
feature_corr = []
for feature in feature_selected:
  if feature in numerical:
    feature_corr.append(feature)

#Print correlation matrix
print(X_train_final_outlier_removed[feature_corr[0:20]].corr())

print('-'*100)

#Plot heat map based on correlation matrix
plt.figure(figsize =(20, 20))
dataplot = sns.heatmap(X_train_final_outlier_removed[feature_corr[0:20]].corr(), cmap="Yl
GnBu", annot=True)
plt.show()
```

```
                                  REGION_RATING_CLIENT_W_CITY  ...  DAYS_FIRST_DUE
REGION_RATING_CLIENT_W_CITY                          1.000000  ...       -0.009183
DAYS_BIRTH                                           0.011864  ...        0.008468
YEARS_BEGINEXPLUATATION_AVG                          0.035539  ...       -0.001765
AMT_ANNUITY                                         -0.129389  ...       -0.000515
AMT_REQ_CREDIT_BUREAU_QRT                            0.006809  ...        0.025291
LOAN_INCOME_RATIO                                    0.062781  ...       -0.005542
AMT_CREDIT_MAX_OVERDUE                              -0.006877  ...       -0.000301
AMT_GOODS_PRICE_PREVIOUS_APPLICATION                -0.070661  ...       -0.003838
DAYS_LAST_DUE                                        0.015467  ...        0.312531
AMT_APPLICATION                                     -0.067549  ...       -0.026487
AMT_CREDIT_PREVIOUS_APPLICATION                     -0.066185  ...       -0.008477
EXT_SOURCE_2                                        -0.280087  ...       -0.000319
FLAG_WORK_PHONE                                      0.016843  ...       -0.007560
REG_CITY_NOT_LIVE_CITY                               0.043462  ...        0.000928
DAYS_REGISTRATION                                    0.079369  ...        0.004220
DAYS_LAST_PHONE_CHANGE                               0.029036  ...        0.033257
AMT_ANNUITY_PREVIOUS_APPLICATION                    -0.112764  ...       -0.025851
DAYS_DECISION                                        0.019275  ...        0.140414
EXT_SOURCE_3                                        -0.011461  ...        0.008097
DAYS_FIRST_DUE                                      -0.009183  ...        1.000000

[20 rows x 20 columns]
----------------------------------------------------------------------------------------------------
```





It is observed that only a few features are strongly correlated to other/others. From this perspective also feature selection is correctly done.

In [ ]:

```
'''In this code cell data sets with selected featurs are saved'''

#Create dataframes X_validate_final_feature_selected, X_test_final_feature_selected
#and application_test_final_feature_selected with feature selected columns
X_train_final_feature_selected = X_train_final_outlier_removed[list(feature_selected)]
X_validate_final_feature_selected = X_validate_final[list(feature_selected)]
X_test_final_feature_selected = X_test_final[list(feature_selected)]
application_test_final_feature_selected = application_test_final_ohe_combined[list(featur
e_selected)]

#Copy y or target values for train, validate and test in variables with names similar to
respective X
y_train_final_feature_selected = y_train_final_outlier_removed
y_validate_final_feature_selected = y_validate
y_test_final_feature_selected = y_test
```

7.5 Create a restore point

Datasets generated at this point are saved as csv files and uploaded to google drive for future use. This is done because of RAM limitation on google colaboratory. Also it allows to run diffent sections independently.

In [ ]:

```
'''In this code cell final data sets with selected featurs are saved and downloaded as cs
v files. These are uploaded to google drive for future access.'''

#Save the dataframes into CSV files for future use
X_train_final_feature_selected.to_csv('X_train_final_feature_selected.csv', index = False
)
files.download("X_train_final_feature_selected.csv")
y_train_final_feature_selected.to_csv('y_train_final_feature_selected.csv', index = False
)
files.download("y_train_final_feature_selected.csv")
X_validate_final_feature_selected.to_csv('X_validate_final_feature_selected.csv', index =
False)
files.download("X_validate_final_feature_selected.csv")
y_validate_final_feature_selected.to_csv('y_validate_final_feature_selected.csv', index =
False)
files.download("y_validate_final_feature_selected.csv")
X_test_final_feature_selected.to_csv('X_test_final_feature_selected.csv', index = False)
files.download("X_test_final_feature_selected.csv")
y_test_final_feature_selected.to_csv('y_test_final_feature_selected.csv', index = False)
files.download("y_test_final_feature_selected.csv")
application_test_final_feature_selected.to_csv('application_test_final_feature_selected.c
sv', index = False)
files.download("application_test_final_feature_selected.csv")
```

8.0 TSNE

TSNE is performed and clusters are plotted with different values of perplexity and iterations. Finally two combinations and their plots are retained. Final output is summarised below:

- **Import data**
- **Perform TSNE with perpexity value of 30 and 1000 iterations**
- **Perform TSNE with perpexity value of 50 and 1000 iterations**

8.1 Import data

In [ ]:

```
'''In this code cell data from the required csv files are imported'''

#Read X_train_final_feature_selected
X_train_final_feature_selected = pd.read_csv('/content/drive/MyDrive/AI_ML_Project/Data/X
_train_final_feature_selected.csv')

#Read y_train_final_feature_selected
y_train_final_feature_selected = pd.read_csv('/content/drive/MyDrive/AI_ML_Project/Data/y
_train_final_feature_selected.csv')
```

In [ ]:

```
'''In this code cell shapes of datasets imported above are printed'''

print(X_train_final_feature_selected.shape)
print(y_train_final_feature_selected.shape)
```

(204494, 176)

(204494, 1)

8.2 Perform TSNE with perpexity value of 30 and 1000 iterations

In [ ]:

```
'''In this code cell TSNE is fit on X_train_final_feature_selected'''

#Fit X_train_final_feature_selected to TSNE
X_embedded = TSNE(n_components=2, perplexity=30, n_iter=1000, learning_rate='auto', init
='random').fit_transform(X_train_final_feature_selected)
print(X_embedded.shape)
```

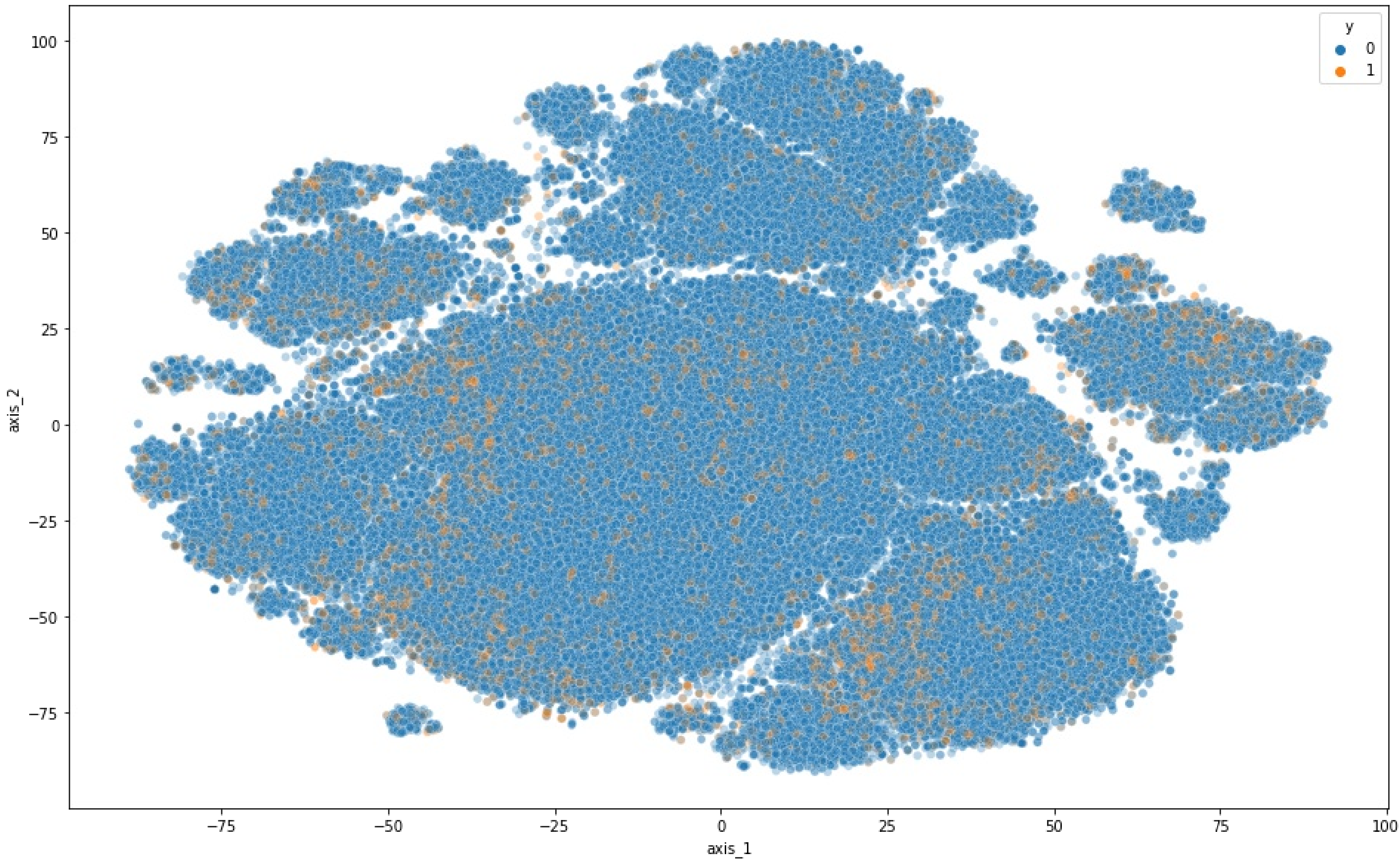
(204494, 2)

In [ ]:

```
'''In this code cell clusters are plotted'''

#Create dataframe with 2 dimensions obtained from TSNE and y/TARGET label
data = {"axis_1": X_embedded[:, 0],
        "axis_2": X_embedded[:, 1],
        "y": y_train_final_feature_selected.to_numpy().flatten()}
X_embedded_dataframe = pd.DataFrame(data)

#Plot scatter plot
plt.figure(figsize=(16,10))
sns.scatterplot(data=X_embedded_dataframe, x="axis_1", y="axis_2", hue="y", legend="full"
, alpha=0.3)
```

Out[ ]:

<matplotlib.axes._subplots.AxesSubplot at 0x7f105b1c0210>



8.3 Perform TSNE with perpexity value of 50 and 1000 iterations

In [ ]:

```
'''In this code cell TSNE is fit on X_train_final_feature_selected'''
```

```
#Fit X_train_final_feature_selected to TSNE
X_embedded = TSNE(n_components=2, perplexity=50, n_iter=1000, learning_rate='auto', init
='random').fit_transform(X_train_final_feature_selected)
print(X_embedded.shape)
```

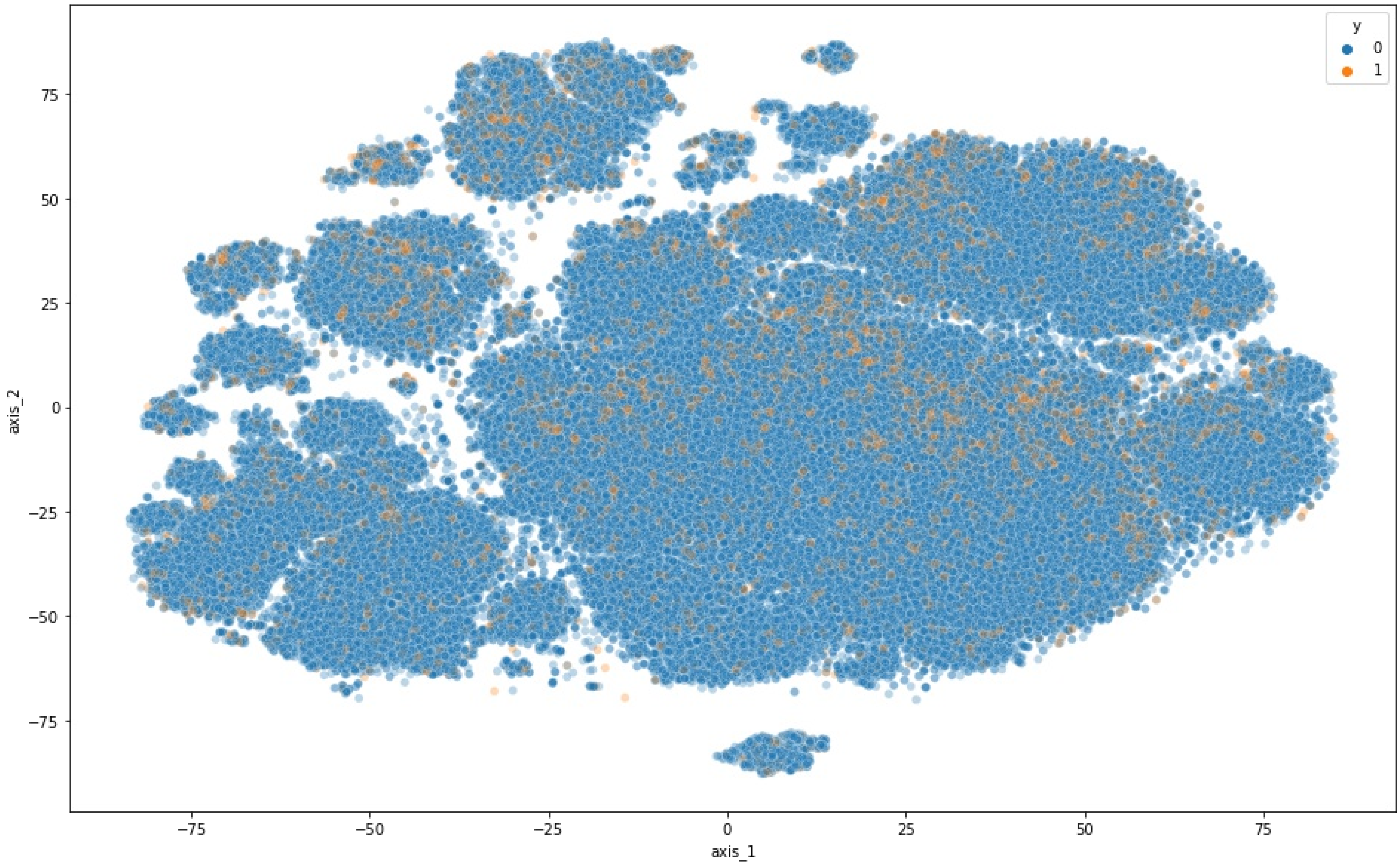
(204494, 2)

In [ ]:

```
'''In this code cell clusters are plotted'''

#Create dataframe with 2 dimensions obtained from TSNE and y/TARGET label
data = {'axis_1': X_embedded[:, 0],
        'axis_2': X_embedded[:, 1],
        'y': y_train_final_feature_selected.to_numpy().flatten()}
X_embedded_dataframe = pd.DataFrame(data)

#Plot scatter plot
plt.figure(figsize=(16,10))
sns.scatterplot(x="axis_1", y="axis_2", hue="y", data=X_embedded_dataframe, legend="full"
, alpha=0.3)
```

Out[ ]:

<matplotlib.axes._subplots.AxesSubplot at 0x7f105a9ad350>



> **Some clusters are obtained but the 2 target values are mixed. Separability based on target values shall be further evaluated after model training. Different models shall be tried.**