```
import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
import os
for dirname, _, filenames in os.walk('/kaggle/input'):
    for filename in filenames:
        print(os.path.join(dirname, filename))
```

```
import warnings
warnings.filterwarnings('ignore')
```

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
%matplotlib inline
from scipy import stats
import re
import seaborn as sns
import matplotlib.ticker as ticker
```

Reading the data

```
# Reading the csv files

customer_data=pd.read_csv("/content/application_data.csv");
customer_prev_data=pd.read_csv("/content/previous_application.csv");
```

```
---------------------------------------------------------------------------
FileNotFoundError                         Traceback (most recent call last)
<ipython-input-6-a20914fc5930> in <cell line: 3>()
      1 # Reading the csv files
      2 
----> 3 customer_data=pd.read_csv("/content/application_data.csv");
      4 customer_prev_data=pd.read_csv("/content/previous_application.csv");

                                    6 frames
/usr/local/lib/python3.10/dist-packages/pandas/io/common.py in get_handle(path_or_buf, mode, encoding, compression, memory_map,
is_text, errors, storage_options)
    854         if ioargs.encoding and "b" not in ioargs.mode:
    855             # Encoding
--> 856             handle = open(
    857                 handle,
    858                 ioargs.mode,

FileNotFoundError: [Errno 2] No such file or directory: '/content/application_data.csv'
```

```
# Information about the application DataFrame

customer_data.info(verbose=True)
```

```
# Information about the previous application DataFrame

customer_prev_data.info(verbose=True)
```

Analyzing the missing data

```
# Checking null values in Customer data
null_sum= customer_data.isnull().sum()
perc_missing= pd.DataFrame({'Columns': null_sum.index, 'Percentage': null_sum.values/customer_data.shape[0]*100})
perc_missing
```

```
#Converting previous application missing value into percentages
null_prev_sum= customer_prev_data.isnull().sum()
perc_missing_prev= pd.DataFrame({'Columns': null_prev_sum.index, 'Percentage': null_prev_sum.values/customer_prev_data.shape[0]*100
```

```
perc_missing_prev
```

```
#Empty columns in the previous data
empty_cols= null_prev_sum[null_prev_sum>(0.3*len(null_prev_sum))]
len(empty_cols)
```

```
# Removing those 15 columns for previous application data

empty_cols = list(empty_cols[empty_cols.values>=0.3].index)
customer_prev_data.drop(labels=empty_cols,axis=1,inplace=True)

customer_prev_data.shape
```

```
# Dropping unnecessary values in NAME_CASH_LOAN_PURPOSE column for previous application data

customer_prev_data=customer_prev_data.drop(customer_prev_data[customer_prev_data['NAME_CASH_LOAN_PURPOSE']=='XNA'].index)
customer_prev_data=customer_prev_data.drop(customer_prev_data[customer_prev_data['NAME_CASH_LOAN_PURPOSE']=='XNA'].index)
customer_prev_data=customer_prev_data.drop(customer_prev_data[customer_prev_data['NAME_CASH_LOAN_PURPOSE']=='XAP'].index)

customer_prev_data.shape
```

```
# Columns with more than 50% missing values for the application data

missing_more_50 = perc_missing [perc_missing['Percentage']>=50]
missing_more_50
```

```
#Droppng the columnns with more than 50% missing values for application data

customer_data = customer_data.drop(columns=missing_more_50.Columns.to_list())
customer_data
```

```
# Removing some unnecessary columns for application data

customer_data = customer_data.drop(columns=['NAME_TYPE_SUITE','FLAG_DOCUMENT_2','FLAG_DOCUMENT_3','FLAG_DOCUMENT_4','FLAG_DOCUMENT_
customer_data
```

```
# Checking for the missing values less than 50 percent for application data

val_missing_50 = pd.DataFrame({'Columns':customer_data.isnull().sum().index,'Percentage':(customer_data.isnull().sum().values/custor
filteredCols = val_missing_50 [val_missing_50["Percentage"]>0]
filteredCols
```

```
#Plotting the graph for displaying the missing data percentage

y_vals = filteredCols.Columns.tolist()
x_vals = filteredCols.Percentage.tolist()

# Data to plot

plt.figure(figsize=[15,6])
plt.bar(range(len(y_vals)), x_vals, tick_label=y_vals)
plt.title("Before handling the missing application data \n", fontdict={'fontsize': 20, 'fontweight' : 20, 'color' : 'Green'})
plt.xticks(rotation=90)
plt.show()
```

Treating the Missing Data

```
#describing the AMT_ANNUITY column

customer_data.AMT_ANNUITY.describe()
```

```
# imputing the missing value in the AMT_ANNUITY column with median value

customer_data['AMT_ANNUITY'] = customer_data['AMT_ANNUITY'].fillna(customer_data['AMT_ANNUITY'].median())
```

```
#validating the missing value for column AMT_ANNUITY after replacing with median value

customer_data.AMT_ANNUITY.isnull().sum()
```

```
#describing the AMT_GOODS_PRICE column

customer_data.AMT_GOODS_PRICE.describe()
```

```
# imputing the missing value in the  AMT_GOODS_PRICE column with median value

customer_data['AMT_GOODS_PRICE'] = customer_data['AMT_GOODS_PRICE'].fillna( customer_data['AMT_GOODS_PRICE'].median())
```

```
#describing the CNT_FAM_MEMBERS column

customer_data.CNT_FAM_MEMBERS.describe()
```

```
#imputing the missing value in the CNT_FAM_MEMBERS column with median value

customer_data['CNT_FAM_MEMBERS'] = customer_data['CNT_FAM_MEMBERS'].fillna(customer_data['CNT_FAM_MEMBERS'].median())
```

```
#describing the OBS_30_CNT_SOCIAL_CIRCLE column

customer_data.OBS_30_CNT_SOCIAL_CIRCLE.describe()
```

```
#imputing the missing value in the CNT_FAM_MEMBERS column with median value
customer_data['OBS_30_CNT_SOCIAL_CIRCLE'] = customer_data['OBS_30_CNT_SOCIAL_CIRCLE'].fillna(customer_data['OBS_30_CNT_SOCIAL_CIRCLE
```

```
#describing the DEF_30_CNT_SOCIAL_CIRCLE column

customer_data.DEF_30_CNT_SOCIAL_CIRCLE.describe()
```

```
#imputng the missing value in the DEF_30_CNT_SOCIAL_CIRCLE column with median value

customer_data['DEF_30_CNT_SOCIAL_CIRCLE'] = customer_data['DEF_30_CNT_SOCIAL_CIRCLE'].fillna(customer_data['DEF_30_CNT_SOCIAL_CIRCLE
```

```
#describing the OBS_60_CNT_SOCIAL_CIRCLE column

customer_data.OBS_60_CNT_SOCIAL_CIRCLE.describe()
```

```
#imputing the missing value in the OBS_60_CNT_SOCIAL_CIRCLE column with median value

customer_data['OBS_60_CNT_SOCIAL_CIRCLE'] =customer_data['OBS_60_CNT_SOCIAL_CIRCLE'].fillna(customer_data['OBS_60_CNT_SOCIAL_CIRCLE

```

```
#describing the DEF_60_CNT_SOCIAL_CIRCLE column

customer_data.DEF_60_CNT_SOCIAL_CIRCLE.describe()
```

```
#imputng the missing value in the DEF_60_CNT_SOCIAL_CIRCLE column with median value

customer_data['DEF_60_CNT_SOCIAL_CIRCLE'] = customer_data['DEF_60_CNT_SOCIAL_CIRCLE'].fillna(customer_data['DEF_60_CNT_SOCIAL_CIRCLE
```

```
customer_data.AMT_REQ_CREDIT_BUREAU_HOUR.describe()
```

```
# imputing the missing value in the AMT_REQ_CREDIT_BUREAU_HOUR column with median value

customer_data['AMT_REQ_CREDIT_BUREAU_HOUR'] = customer_data['AMT_REQ_CREDIT_BUREAU_HOUR'].fillna(customer_data['AMT_REQ_CREDIT_BURE

```

```
#describing the AMT_REQ_CREDIT_BUREAU_DAY column

customer_data.AMT_REQ_CREDIT_BUREAU_DAY.describe()
```

```
# imputing the missing value in the AMT_REQ_CREDIT_BUREAU_DAY column with median value

customer_data['AMT_REQ_CREDIT_BUREAU_DAY'] = customer_data['AMT_REQ_CREDIT_BUREAU_DAY'].fillna( customer_data['AMT_REQ_CREDIT_BUREAU
```

```
# describing the AMT_REQ_CREDIT_BUREAU_WEEK column

customer_data.AMT_REQ_CREDIT_BUREAU_WEEK.describe()
```

```
# imputing the missing value in the AMT_REQ_CREDIT_BUREAU_WEEK column with median value

customer_data['AMT_REQ_CREDIT_BUREAU_WEEK'] = customer_data['AMT_REQ_CREDIT_BUREAU_WEEK'].fillna( customer_data['AMT_REQ_CREDIT_BURE
```

```
# describing the AMT_REQ_CREDIT_BUREAU_MON column

customer_data.AMT_REQ_CREDIT_BUREAU_MON.describe()
```

```python
# imputing the missing value in the AMT_REQ_CREDIT_BUREAU_MON column with median value

customer_data['AMT_REQ_CREDIT_BUREAU_MON'] = customer_data['AMT_REQ_CREDIT_BUREAU_MON'].fillna( customer_data['AMT_REQ_CREDIT_BUREAU
```

```python
# describing the AMT_REQ_CREDIT_BUREAU_YEAR column

customer_data.AMT_REQ_CREDIT_BUREAU_YEAR.describe()
```

```python
# imputing the missing value in the AMT_REQ_CREDIT_BUREAU_YEAR column with median value

customer_data['AMT_REQ_CREDIT_BUREAU_YEAR'] = customer_data['AMT_REQ_CREDIT_BUREAU_YEAR'].fillna( customer_data['AMT_REQ_CREDIT_BURE
```

```python
# describing the AMT_REQ_CREDIT_BUREAU_QRT column

customer_data.AMT_REQ_CREDIT_BUREAU_QRT.describe()
```

```python
# imputing the missing value in the AMT_REQ_CREDIT_BUREAU_QRT column with median value

customer_data['AMT_REQ_CREDIT_BUREAU_QRT'] = customer_data['AMT_REQ_CREDIT_BUREAU_QRT'].fillna( customer_data['AMT_REQ_CREDIT_BUREAU
```

```python
# describing the OCCUPATION_TYPE column

customer_data.OCCUPATION_TYPE.describe()
```

```python
# imputing the missing value in the OCCUPATION_TYPE column with Laborers.

customer_data['OCCUPATION_TYPE'] = customer_data['OCCUPATION_TYPE'].fillna('Laborers')
```

Standardizing the Column Values

```python
# Converting the negetive value columns to positive using abs() function

customer_data.DAYS_BIRTH = customer_data.DAYS_BIRTH.abs()                # Converting the Days_Birth column

customer_data.DAYS_EMPLOYED = customer_data.DAYS_EMPLOYED.abs()          # Convert the DAYS_EMPLOYED column

customer_data.DAYS_REGISTRATION = customer_data.DAYS_REGISTRATION.abs()  # Convert the DAYS_REGISTRATION column

customer_data.DAYS_ID_PUBLISH = customer_data.DAYS_ID_PUBLISH.abs()      # Convert the DAYS_ID_PUBLISH column
```

```python
#Calculating the Age of the Client from DAYS_BIRTH column

customer_data['Age_of_Client'] = (customer_data.DAYS_BIRTH / 365).round(2)
customer_data
```

```python
# Binnig the age column

bins = [0,30,40,50,60,100]
labels = ['<30','30-40','40-50','50-60','60+']
customer_data['AGE_RANGE'] = pd.cut(customer_data.Age_of_Client, bins=bins, labels=labels)
customer_data
```

```python
# Removing Age_Range column after making bins out of it

customer_data.drop(columns='Age_of_Client',inplace=True)
```

```python
# Value Counts for CODE_GENDER column

customer_data.CODE_GENDER.value_counts()
```

Since, Female is having the majority and only 4 rows are having NA values, we can update those columns with Gender 'F' as there will be no impact on the dataset.

```
# Creating bins for income amount

bins = [0,25000,50000,75000,100000,125000,150000,175000,200000,225000,250000,275000,300000,325000,350000,375000,400000,425000,45000
slot = ['0-25000', '25000-50000','50000-75000','75000,100000','100000-125000', '125000-150000', '150000-175000','175000-200000',
       '200000-225000','225000-250000','250000-275000','275000-300000','300000-325000','325000-350000','350000-375000',
       '375000-400000','400000-425000','425000-450000','450000-475000','475000-500000','500000 and above']

customer_data['AMT_INCOME_RANGE']=pd.cut(customer_data['AMT_INCOME_TOTAL'],bins,labels=slot)
```

```
# Creating bins for Credit amount

bins = [0,150000,200000,250000,300000,350000,400000,450000,500000,550000,600000,650000,700000,750000,800000,850000,900000,100000000
slots = ['0-150000', '150000-200000','200000-250000', '250000-300000', '300000-350000', '350000-400000','400000-450000',
        '450000-500000','500000-550000','550000-600000','600000-650000','650000-700000','700000-750000','750000-800000',
        '800000-850000','850000-900000','900000 and above']

customer_data['AMT_CREDIT_RANGE']=pd.cut(customer_data['AMT_CREDIT'],bins=bins,labels=slots)
```

```
# Updating the CODE_GENDER column

customer_data.loc[customer_data['CODE_GENDER']=='XNA','CODE_GENDER']='F'
customer_data['CODE_GENDER'].value_counts()
```

```
# Value counts for ORGANIZATION_TYPE column

customer_data.ORGANIZATION_TYPE.value_counts()
```

Since there are huge number of 'XNA' values, so either we can drop or leave them as it is. No affect will occur on further analysis

Handling Outliers

Plotting Graphs to Check for Outlier Values

AMT_INCOME_TOTAL Variable

```
#Box Plot for AMT_INCOME_TOTAL Variable

fig = plt.figure(figsize=(22,6))
ax = sns.boxplot(data=customer_data.AMT_INCOME_TOTAL, orient='h')
ax.set_xscale('linear')
ax.xaxis.set_major_locator(ticker.MultipleLocator(2000000))
ax.set_title('Income Total Amount',fontsize=45)
plt.xticks(rotation=90)
plt.show()
```

```
# Describing the AMT_INCOME_TOTAL column for max value

customer_data.AMT_INCOME_TOTAL.describe()
```

As the max value is 1.17 X 10^8 which is infact a really high value but this is often a legitimate value as income can vary from people to people. Therefore, no operation is required in this

AMT_CREDIT Variable

```
# Box Plot for AMT_CREDIT Variable

fig = plt.figure(figsize=(22,6))
ax = sns.boxplot(data=customer_data.AMT_CREDIT,orient='h',color='green')
ax.set_xscale('linear')
ax.xaxis.set_major_locator(ticker.MultipleLocator(100000))
ax.set_title('Credit Amount',fontsize=45)
plt.show()
```

```
# Describing the AMT_CREDIT column

customer_data.AMT_CREDIT.describe()
```

The outliers value can handled by converting into bins .

AMT_ANNUITY Variable

```
# Box Plot for AMT_ANNUITY Variable

fig = plt.figure(figsize=(22,6))
ax = sns.boxplot(data =customer_data.AMT_ANNUITY,orient='h')
ax.set_xscale('linear')
ax.xaxis.set_major_locator(ticker.MultipleLocator(10000))
ax.set_title('Annuity Amount',fontsize=45)
plt.xticks(rotation=90)
plt.show()
```

```
# Describing the AMT_ANNUITY column

customer_data.AMT_ANNUITY.describe()
```

Amount Annuity column has some outliers value and handled it by converting into bins.

AMT_GOODS_PRICE Variable

```
# Box Plot for AMT_GOODS_PRICE Variable

fig = plt.figure(figsize=(22,6))
ax = sns.boxplot(data=customer_data.AMT_GOODS_PRICE,orient='h',color='pink')
ax.set_xscale('linear')
ax.xaxis.set_major_locator(ticker.MultipleLocator(100000))
ax.set_title('Goods Amount',fontsize=45)
plt.xticks(rotation = 90)
plt.show()
```

```
  # Describing the AMT_GOODS_PRICE column

customer_data.AMT_GOODS_PRICE.describe()
```

No action should be taken on this column because prices can vary from product to product

DAYS_BIRTH Variable

```
# Box Plot for DAYS_BIRTH Variable

fig = plt.figure(figsize=(19,4))
ax = sns.boxplot(data=customer_data.DAYS_BIRTH, orient='h',color='brown')
ax.set_xscale('linear')
ax.xaxis.set_major_locator(ticker.MultipleLocator(1000))
ax.set_title('Age',fontsize=45)
plt.show()
```

```
# Box Plot for DAYS_EMPLOYED Variable

fig = plt.figure(figsize=(20,5))
ax = sns.boxplot(customer_data.DAYS_EMPLOYED, orient='h', color='orange')
ax.set_title('Days Employed',fontsize=45)
ax.xaxis.set_major_locator(ticker.MultipleLocator(10000))
plt.xticks(rotation=90)
plt.show()
```

```
# Describing the DAYS_EMPLOYED column

customer_data['DAYS_EMPLOYED'].describe()
```

Maximum value is 365243 which we need to divide from 365 to get no. of years

```
# Converting to years

years = 365243 / 365
years
```

On dividing the outlier value by days in a year we get 1000 years approximately which is not possible and therefore it's an invalid data. We can correct this data by replacing it to the last closest percentile value or value approximately close to it.

```
# Treating the outlier value of DAYS_EMPLOYED column

customer_data = customer_data [customer_data.DAYS_EMPLOYED<np.nanpercentile(customer_data['DAYS_EMPLOYED'], 99)]

```

DAYS_ID_PUBLISH Variable

```
#  Box Plot for DAYS_ID_PUBLISH Variable

fig = plt.figure(figsize=(15,3))
ax = sns.boxplot(customer_data.DAYS_ID_PUBLISH,orient='h',color='lavender')
ax.set_title('Days Id Publish',fontsize=45)
ax.xaxis.set_major_locator(ticker.MultipleLocator(1000))
plt.xticks(rotation=90)
plt.show()
```

No outliers in the DAYS_ID_PUBLISH column so we can leave it as it is.

ANALYSIS APPLICATION DATAFRAME

Data Imbalance Check¶

```
# Checking the TARGET column for imbalance data in percentage

customer_data.TARGET.value_counts(normalize=True)*100
```

```
# Plotting the graphs

fig, (ax1,ax2) = plt.subplots(1,2,figsize =(20,8))

ax = sns.countplot(customer_data.TARGET,ax=ax1)

ax1.set_title('TARGET',fontsize=20)

plt.setp(ax1.xaxis.get_majorticklabels(),fontsize=18)

ax2 = plt.pie(x=customer_data.TARGET.value_counts(normalize=True),autopct='%.2f',textprops={'fontsize':15},shadow=True,labels=['No P

plt.title('Distribution of the Target Variable',fontsize=20)

plt.show()
```

```
# Check the Imbalance Percentage

print('Imbalance Percentage is : %.2f'%(customer_data.TARGET.value_counts(normalize=True)[0]/customer_data.TARGET.value_counts(norma
```

```
# We can divide the dataset to two into different dataframes i.e. target=0 (client with no payment difficulties) & target=1 (client

target0 = customer_data.loc[customer_data["TARGET"]==0]
target1 = customer_data.loc[customer_data["TARGET"]==1]
```

Univariate Analysis

Categorical Univariate Analysis for target = 0 and target = 1

Unordered Categorical Data

```python
# function for countplot

def graph_uni(col):
    plt.figure(figsize=(17,6))
    plt.style.use('bmh')
    plt.subplot(1, 2, 1)
    sns.countplot(col, data=target0)
    plt.title('Distribution of '+ '%s' %col +' for target=0', fontsize=15)
    plt.xlabel(col , fontsize=15)
    plt.xticks(rotation=90)
    plt.ylabel('Number of cases for non-defaulters')

    plt.subplot(1, 2, 2)
    sns.countplot(col, data=target1)
    plt.title('Distribution of '+ '%s' %col +' for target=1', fontsize=14)
    plt.xlabel(col , fontsize=15)
    plt.xticks(rotation=90)
    plt.ylabel('Number of cases for defaulters')
    plt.show()
```

Inference : Working category has high number of defaulters

Inference : We can see the revolving loans has less distribution in number of cases for defaulters compared to non-defaulters.

Inference : People living with parents and in apartments show high number of default as compared to non defaulters. The reason is that living with parents so less income and living in municipal apartments so more cash flow in apartment rent.

Inference : Single/not married category are showing high number of defaulters.

Ordered Categorical Data

Inference : Income Range of 125000-150000 is maximum for both Loan Repayers & Loan Defaulters

Inference : Income Range of 125000-150000 is maximum for both Loan Repayers & Loan Defaulters

Inference : Amount Credit Range value of 900000 and above is the highest ranked in both scenarios

Numerical - Categorical

```python
def graph(customer_data, col, title, hue =None):

    sns.set_style("darkgrid")
    sns.set_context('talk')
    plt.rcParams["axes.labelsize"] = 20
    plt.rcParams['axes.titlesize'] = 22
    plt.rcParams['axes.titlepad'] = 30


    temp = pd.Series(data = hue)
    fig, ax = plt.subplots()
    width = len(customer_data[col].unique()) + 6.5 + 5*len(temp.unique())
    fig.set_size_inches(width , 8)
    plt.xticks(rotation=45)
    plt.yscale('log')
    plt.title(title)

    ax = sns.countplot(data = customer_data, x= col, order=customer_data[col].value_counts().index, hue = hue, palette='pastel')
    plt.show()
```

```python
# Countplot for Income Range Type Vs Gender for target = 0

graph(target0 ,col='AMT_INCOME_RANGE',title='Distribution of Income Range Vs CODE_GENDER for Non-Defaulters', hue='CODE_GENDER')
```

Inference : Female numbers are mostly high in every proportion and also female credit are mostly higher than males(Loan Repayers)

```python
# Countplot for Income Range Type Vs Gender for target = 1

graph(target1 ,col='AMT_INCOME_RANGE',title='Distribution of Income Range Vs Gender for Defaulters', hue='CODE_GENDER')
```

Inference : Male numbers are mostly high in every proportion and also Male credit are mostly higher than Females(Loan Defaults)

```
# Countplot for Income Type Vs Contract type for target = 0

graph(target0 ,col='NAME_INCOME_TYPE',title='Distribution of income range', hue='NAME_CONTRACT_TYPE')
```

Inference : Working class has more number of Cash Loans

```
# Countplot for Income Type Type Vs Contract type for target = 1

graph(target1 ,col='NAME_INCOME_TYPE',title='Distribution of income range', hue='NAME_CONTRACT_TYPE')
```

Inference : Working class has more number of Cash Loans

```
# function for countplot

plt.figure(figsize=(15,30))
sns.set_style('darkgrid')
plt.rcParams["axes.labelsize"] = 21
plt.rcParams['axes.titlesize'] = 23
plt.rcParams['axes.titlepad'] = 30
plt.xscale('log')

plt.title("Distribution of Organization type for target - 0")

sns.countplot(data=target0 ,y='ORGANIZATION_TYPE',order=target0['ORGANIZATION_TYPE'].value_counts().index, palette="icefire")
plt.show()
```

Inference : ‘Business entity Type 3’ and ‘Self employed’ hold the most credits and less credit to industry trade: type5 and Industry: type 8(target = 0)

```
# Graph for ORGANIZATION TYPE

plt.figure(figsize=(15,30))
sns.set_style('darkgrid')
plt.rcParams["axes.labelsize"] = 21
plt.rcParams['axes.titlesize'] = 23
plt.rcParams['axes.titlepad'] = 30
plt.xscale('log')

plt.title("Distribution of Organization type for target - 1")

sns.countplot(data=target1 ,y='ORGANIZATION_TYPE',order=target0['ORGANIZATION_TYPE'].value_counts().index, palette="icefire")

plt.show()
```

Inference : ‘Business entity Type 3’ and ‘Self employed’ hold the most credits and less credit to industry trade: type5 and Industry: type 8 (target = 1)

Bivariate Analysis

Numerical - Numerical Analysis

```
# Creating the function for the numeric bivariate analysis

def graph_numeric (a,b):
    sns.set_style(style='ticks')
    fig=plt.figure(figsize=(16,7))

    sns.scatterplot(data = target0, y = b, x = a,  label='Loan Repayers',    color='steelblue')
    sns.scatterplot(data = target1, y = b, x = a,  label='Loan Defaulters',  color='hotpink')

    plt.title(f'{a} vs {b}',fontsize=21)
    plt.show()
```

```
# graph for AMT_ANNUITY vs AMT_GOODS_PRICE

graph_numeric ('AMT_ANNUITY','AMT_GOODS_PRICE')
```

Inference : Correlation between Amount Annuity & Amount Goods Price is pretty moderates but they are not thoroughly correlated because there are above par values for both the columns. . Amount Annuity having values less than 70000 are prone to be in default while values above 70000 are tend to decrease in defaulters.

```
# graph for AMT_ANNUITY vs AMT_CREDIT

graph_numeric('AMT_ANNUITY','AMT_CREDIT')
```

Inference :

There is fair correlation between the two columns Amount Annuity & Amount Credit. There's a decrease in defaulters when Amount Annuity increases. Most of the defaulters are having AMT_ANNUITY values less than 60000.

```
# graph for AMT_GOODS_PRICE vs AMT_CREDIT

graph_numeric('AMT_GOODS_PRICE','AMT_CREDIT')

```

Inference : Amount Good Price & Amount Credit have a strong correlation as clients having goods price and will repay their loans.

Correlation or Multivariate analysis

TARGET 0 : Loan Repayer

```
# Dropping the columns which are not required for target 0

cols_drop = ['SK_ID_CURR','AMT_REQ_CREDIT_BUREAU_HOUR','AMT_REQ_CREDIT_BUREAU_DAY', 'AMT_REQ_CREDIT_BUREAU_WEEK','AMT_REQ_CREDIT_BU
target0 = target0.drop(columns=cols_drop)
```

```
# Creating the correlation matrix for the Loan defaulter data frame

corr_t0 = target0.corr().abs().round(3)
corr_t0
```

```
# Visualising the correlation data of target 0 using heatmaps

fig = plt.figure(figsize=(20,15))

sns.heatmap(data=corr_t0 ,linewidths=.5,center=0.1,cmap='gist_earth',annot= True)

plt.show()
```

```
# Unstacking the TARGET_0 variable

c = corr_t0 .abs()
s = c.unstack()
```

```
# Finding top 10 correlation among the people with no payment issues and displaying it.

target_0_corr = s[s.index.get_level_values(0)!= s.index.get_level_values(1)].sort_values(ascending=False,kind='quicksort').drop_dup

top_10_target0 = pd.DataFrame(target_0_corr)

top_10_target0 = top_10_target0.reset_index().rename(columns={'level_0':'Var1','level_1':'Var2',0:'Correlation'}).dropna()

top_10_target0.head(10)
```

TARGET 1 : Loan Defaulter

```
# Dropping the columns which are not required for target1

target1 = target1.drop(columns=cols_drop)
```

```
# Creating the correlation matrix for the Loan defaulter dataframe

corr_t1 = target1.corr().abs().round(3)
corr_t1
```

```
# PLotting the heatmap for displaying the correlation for target 1

fig = plt.figure(figsize=(20,15))

sns.heatmap(data=corr_t1,annot=True,cmap='PuOr',linewidths=0.5,center=0.1)

plt.show()
```

```
#unstacking the correlation of target1 variable
c1 = corr_t1

s1 = c1.unstack()
```

```
# Displaying Top 10 Correlations from target_1 : Loan Defaulter data frame

target_1_corr = s1[s1.index.get_level_values(0)!= s1.index.get_level_values(1)].sort_values(ascending=False,kind='quicksort').drop_

top_10_target1  = pd.DataFrame(target_1_corr)

top_10_target1 = top_10_target1.reset_index().rename(columns={'level_0':'Var1','level_1':'Var2',0:'Correlation'}).dropna()

top_10_target1.head(10)
```

Insights from Correlation

For target variable there is no correlation as there are empty space in the graph and NAN in the tables while comparing with other variables.

Credit amount is highly correlated with amount of goods price which is slightly different from target 0 i.e Loan Repayers.

The correlation is strong between family member and children counts, although the correlation increases for the defaulters.

The loan annuity correlation with credit amount and also with goods price has slightly reduced in defaulters(0.748) when compared to repayers(0.777)

We can also see that repayers have high correlation in number of days employed(0.62) when compared to defaulters(0.58).

Days_birth and number of children correlation has reduced to 0.256 in defaulters when compared to 0.336 in repayers.

Merging the Dataset

```
# Merging the Application Dataset with Previous Appliaction Dataset

merge_df = pd.merge(left=customer_data,right=customer_prev_data,how='inner',on='SK_ID_CURR',suffixes='_x')
```

```
# Renaming the column names after merging the data

rename_merge_df = merge_df.rename({'NAME_CONTRACT_TYPE_' : 'NAME_CONTRACT_TYPE','AMT_CREDIT_':'AMT_CREDIT','AMT_ANNUITY_':'AMT_ANNU
                         'WEEKDAY_APPR_PROCESS_START_' : 'WEEKDAY_APPR_PROCESS_START',
                         'HOUR_APPR_PROCESS_START_':'HOUR_APPR_PROCESS_START','NAME_CONTRACT_TYPEx':'NAME_CONTRACT_TYPE_PREV',
                         'AMT_CREDITx':'AMT_CREDIT_PREV','AMT_ANNUITYx':'AMT_ANNUITY_PREV',
                         'WEEKDAY_APPR_PROCESS_STARTx':'WEEKDAY_APPR_PROCESS_START_PREV',
                         'HOUR_APPR_PROCESS_STARTx':'HOUR_APPR_PROCESS_START_PREV'}, axis=1)
```

```
# Removing unwanted columns for analysis of merged data

rename_merge_df.drop(['SK_ID_CURR','WEEKDAY_APPR_PROCESS_START', 'HOUR_APPR_PROCESS_START','WEEKDAY_APPR_PROCESS_START_PREV','HOUR_
```

Univariate analysis on Merged Dataset

```
# Distribution of Cash Loans Vs Contract Purpose

sns.set_style('whitegrid')
sns.set_context('talk')

plt.figure(figsize=(15,30))
plt.rcParams["axes.labelsize"] = 20
plt.rcParams['axes.titlesize'] = 22
plt.rcParams['axes.titlepad'] = 30
plt.xticks(rotation=90)
plt.xscale('log')
plt.title('Distribution of Contract Status with Purposes')

ax = sns.countplot(data = rename_merge_df, y= 'NAME_CASH_LOAN_PURPOSE', order=rename_merge_df['NAME_CASH_LOAN_PURPOSE'].value_count

```

Inference : Repairs got most refused loans Education has similar outcomes for approval and rejection of loans Paying other loans and buying a new car is having significant higher rejection than approves.

```
# Distribution of contract status

sns.set_style('whitegrid')
sns.set_context('talk')

plt.figure(figsize=(15,30))
plt.rcParams["axes.labelsize"] = 20
plt.rcParams['axes.titlesize'] = 22
plt.rcParams['axes.titlepad'] = 30
plt.xticks(rotation=90)
plt.xscale('log')
plt.title('Distribution of purposes with target')

ax = sns.countplot(data = rename_merge_df, y= 'NAME_CASH_LOAN_PURPOSE', order=rename_merge_df['NAME_CASH_LOAN_PURPOSE'].value_count
```

Inference : Repairs are dealing with more difficulties in payment on time Buying a garage, Business development, Buying land, Buying a new car and Education having basically higher loan payment

```
# Bar plotting for Credit amount prev vs Housing type

plt.figure(figsize=(16,12))
plt.xticks(rotation=90)
sns.barplot(data = rename_merge_df, y='AMT_CREDIT_PREV',hue='TARGET',x='NAME_HOUSING_TYPE')
plt.title('Prev Credit amount vs Housing type')
```