**Problem Statement:**

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

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

**Loading the Data from Google Drive:**

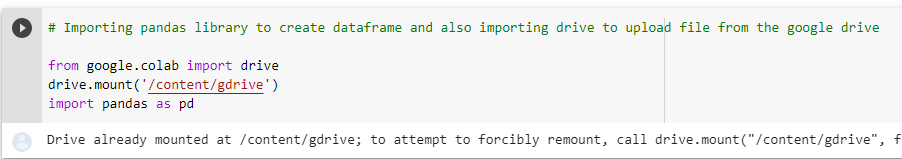
As downloading directly from drive and loading into google colab missing some rows directly loaded files from google drive.

**Code**:

from google.colab import drive

drive.mount('/content/gdrive')

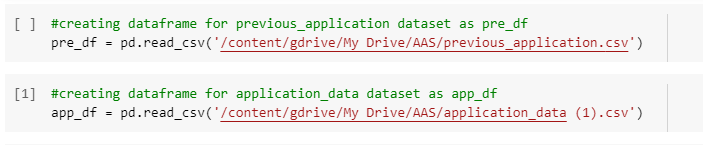
import pandas as pd



Reading files from drive:

pre\_df = pd.read\_csv('/content/gdrive/My Drive/AAS/previous\_application.csv')

app\_df = pd.read\_csv('/content/gdrive/My Drive/AAS/application\_data (1).csv')



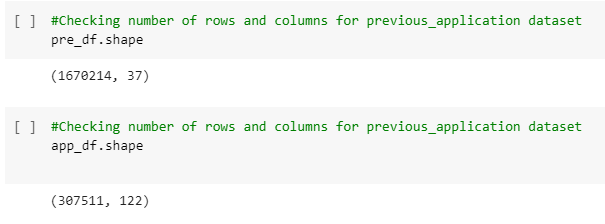
**Checking the Dataset Information:**

**Desc:**Checking the shape of dataframes created which gives number of rows and numbers of columns

**Code**:

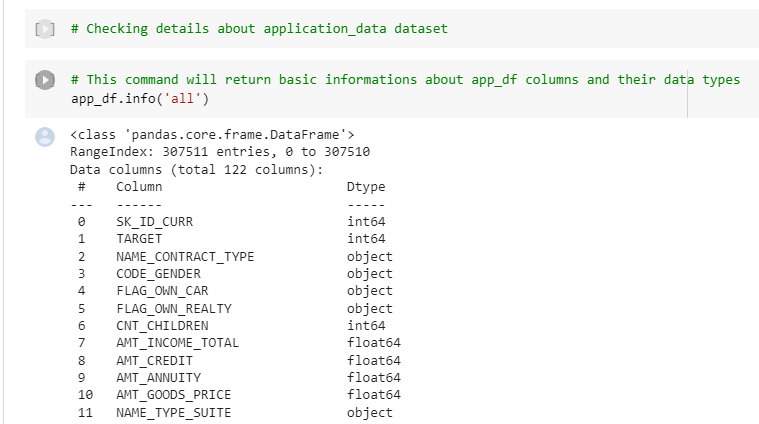
pre\_df.shape

app\_df.shape



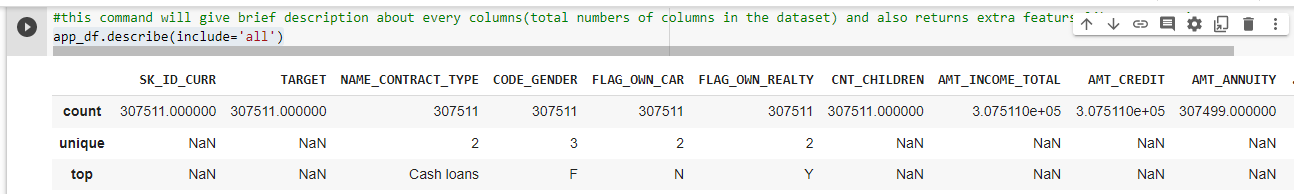
**Desc:**Checking the app\_df information and details.

**Code:**app\_df.info('all')



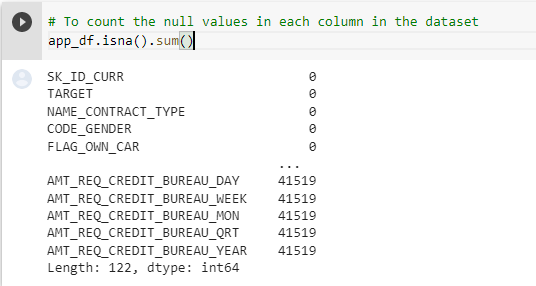
**Desc:**Describing the app\_df dataframe which displays count,unique values,mean,median etc details.

**Code:**app\_df.describe(include='all')



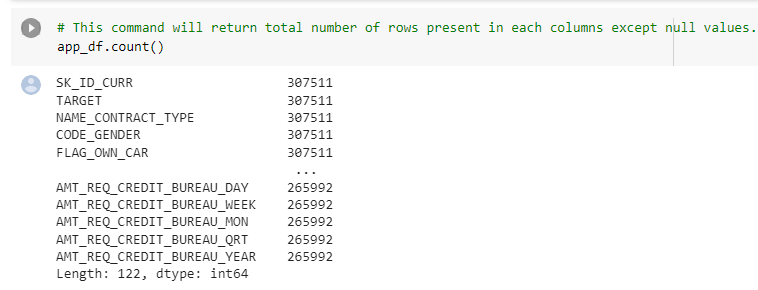
**Desc:**Count the null values column wise in our dataset

**Code:**app\_df.isna().sum()



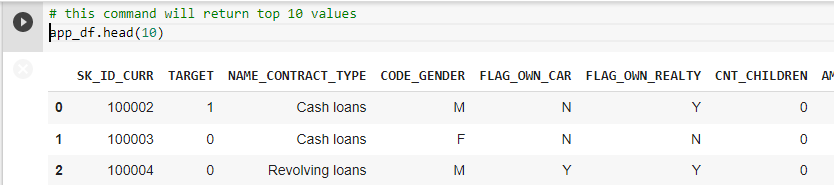
**Desc:**Returning total number of rows present in each column except null values

**Code:**app\_df.count()



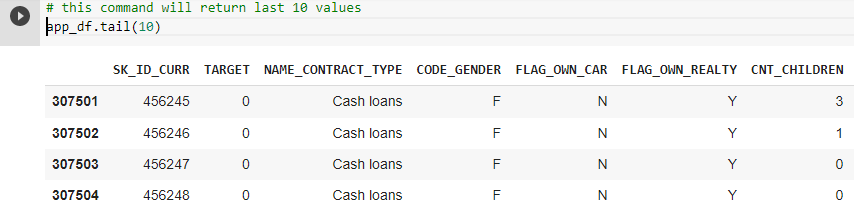
**Desc:**Return top 10 columns of dataset

**Code**:app\_df.head(10)



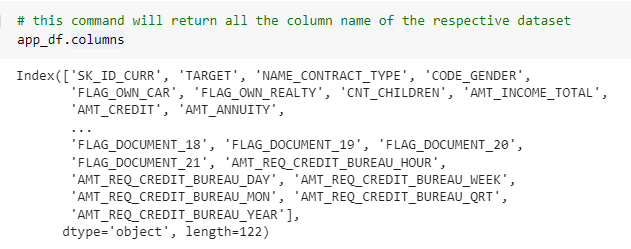
**Desc:**Return bottom 10 columns of dataset

**Code:**app\_df.tail(10)



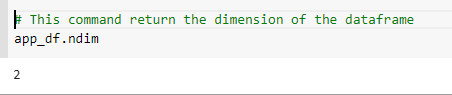
**Desc:**Return all column names available in dataset

**Code:**app\_df.columns



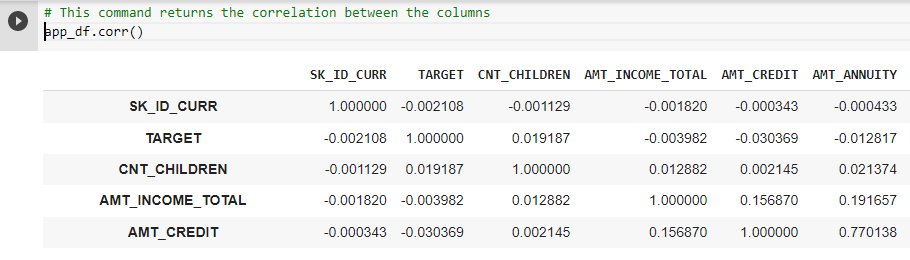
**Desc:**Checking dimensions of dataset

**Code**:app\_df.ndim



**Desc**:calculate correlation between all columns in dataset

**Code:**app\_df.corr()



**Cleaning the Data:**

**Desc:**Dropping flag columns from dataset as they are not useful in data analysis.

**Code:**newapp\_df=app\_df.drop(['FLAG\_MOBIL', 'FLAG\_EMP\_PHONE', 'FLAG\_WORK\_PHONE', 'FLAG\_CONT\_MOBILE',

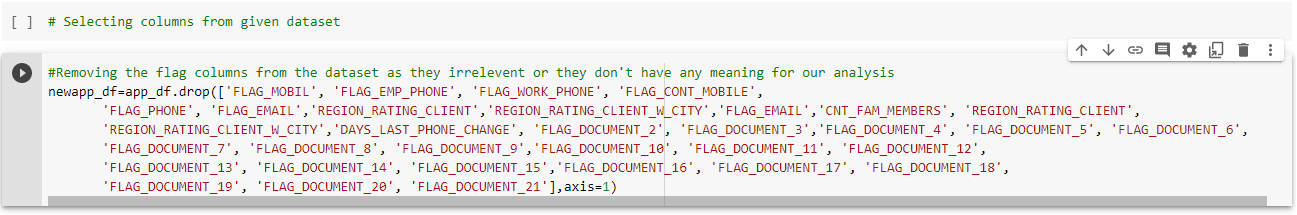
'FLAG\_PHONE', 'FLAG\_EMAIL','REGION\_RATING\_CLIENT','REGION\_RATING\_CLIENT\_W\_CITY','FLAG\_EMAIL','CNT\_FAM\_MEMBERS', 'REGION\_RATING\_CLIENT',

'REGION\_RATING\_CLIENT\_W\_CITY','DAYS\_LAST\_PHONE\_CHANGE', 'FLAG\_DOCUMENT\_2', 'FLAG\_DOCUMENT\_3','FLAG\_DOCUMENT\_4', 'FLAG\_DOCUMENT\_5', 'FLAG\_DOCUMENT\_6',

'FLAG\_DOCUMENT\_7', 'FLAG\_DOCUMENT\_8', 'FLAG\_DOCUMENT\_9','FLAG\_DOCUMENT\_10', 'FLAG\_DOCUMENT\_11', 'FLAG\_DOCUMENT\_12',

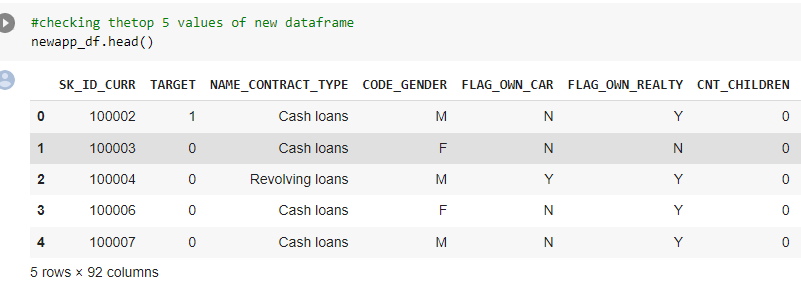
'FLAG\_DOCUMENT\_13', 'FLAG\_DOCUMENT\_14', 'FLAG\_DOCUMENT\_15','FLAG\_DOCUMENT\_16', 'FLAG\_DOCUMENT\_17', 'FLAG\_DOCUMENT\_18',

'FLAG\_DOCUMENT\_19', 'FLAG\_DOCUMENT\_20', 'FLAG\_DOCUMENT\_21'],axis=1)



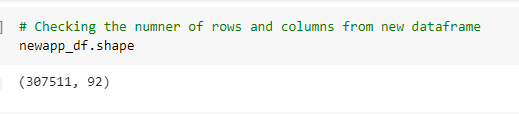
**Desc:**Checking top 5 rows of new dataset after dropping flag columns.

**Code:**newapp\_df.head()



**Desc:**Checking the shape after dropping

**Code:**newapp\_df.shape



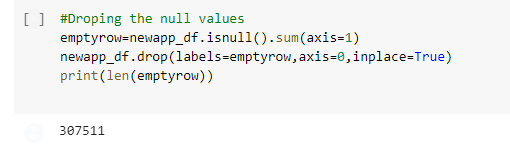
**Desc:**Identifying the rows which are having null values and dropping them from the dataset.

**Code:**

emptyrow=newapp\_df.isnull().sum(axis=1)

newapp\_df.drop(labels=emptyrow,axis=0,inplace=True)

print(len(emptyrow))



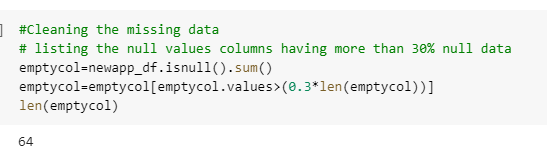
**Desc:**Identifying null values column wise which are having more than 30% null data and dropping those columns from dataset.

**Code:**

emptycol=newapp\_df.isnull().sum()

emptycol=emptycol[emptycol.values>(0.3\*len(emptycol))]

len(emptycol)

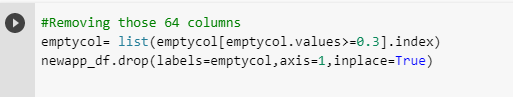


**Desc:**Storing the index of rows which are having more than 30% null data and dropping them from dataset.

**Code:**

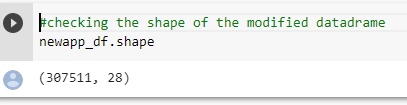
emptycol= list(emptycol[emptycol.values>=0.3].index)

newapp\_df.drop(labels=emptycol,axis=1,inplace=True)



**Desc:**Checking the shape of dataset after dropping

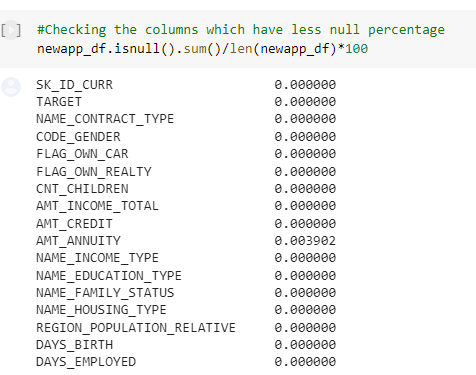
**Code:**newapp\_df.shape



**Desc:**Checking the columns which have less null percentage

**Code:**

newapp\_df.isnull().sum()/len(newapp\_df)\*100



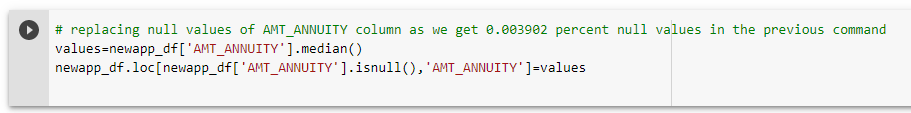
**Observation:**Since this column is having an outlier which is very large it will be inappropriate to fill those missing values with mean, Hence Median comes to rescue for this and we will fill those missing banks with median value.

**Desc:**Replacing null values of AMT\_ANNUITY with values.

**Code:**

values=newapp\_df['AMT\_ANNUITY'].median()

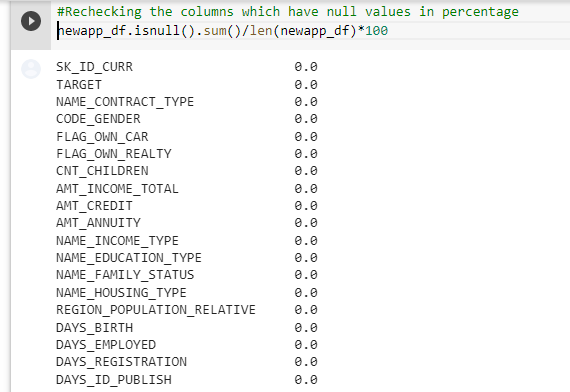
newapp\_df.loc[newapp\_df['AMT\_ANNUITY'].isnull(),'AMT\_ANNUITY']=values



**Desc:**After replacing checking the columns which have null values in percentage.

**Code:**

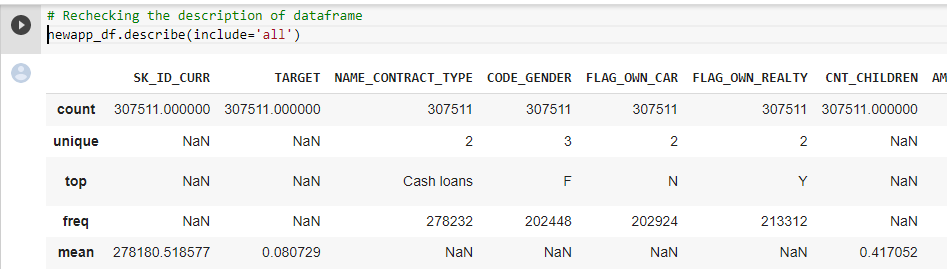
newapp\_df.isnull().sum()/len(newapp\_df)\*100



**Desc:** Rechecking the description of dataframe

**Code:**

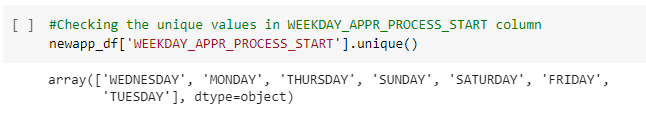
newapp\_df.describe(include='all')



**Desc:** Finding the unique values in WEEKDAY\_APPR\_PROCESS\_START column.

**Code:**

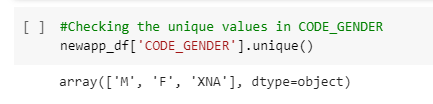
newapp\_df['WEEKDAY\_APPR\_PROCESS\_START'].unique()



**Desc:**Finding the unique values in CODE\_GENDER column

**Code:**

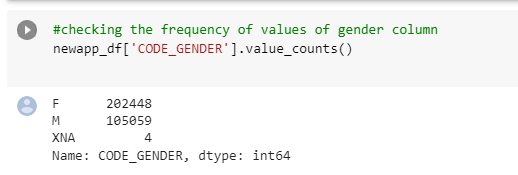
newapp\_df['CODE\_GENDER'].unique()



**Desc:**Checking the frequency of values of CODE\_GENDER column

**Code:**

newapp\_df['CODE\_GENDER'].value\_counts()

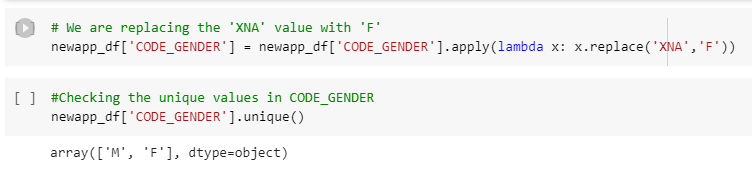


**Desc:**Replacing the value of ‘XNA’ to ‘F’ in CODE\_GENDER column and checking.

**Code:**

newapp\_df['CODE\_GENDER'] = newapp\_df['CODE\_GENDER'].apply(lambda x: x.replace('XNA','F'))

newapp\_df['CODE\_GENDER'].unique()

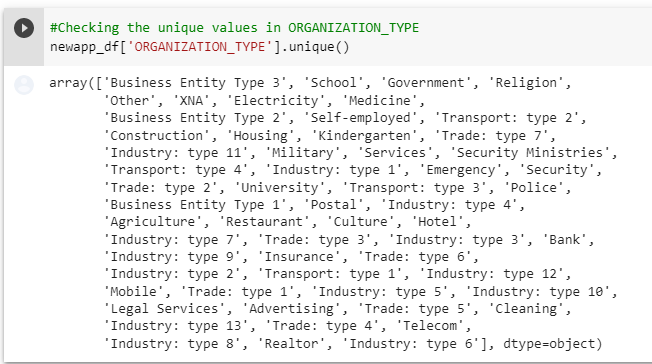


**Observation**:So, for column 'ORGANIZATION\_TYPE', we have total count of 307511 rows of which 55374 rows are having 'XNA' values. Which means 18% of the column is having this values. Hence if we drop the rows of total 55374, will not have any major impact on our dataset.

**Desc:**Finding unique values in ORGANIZATION\_TYPE column

**Code:**

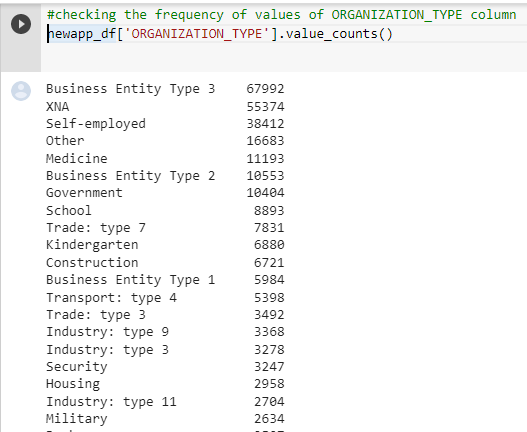
newapp\_df['ORGANIZATION\_TYPE'].unique()



**Desc:** Checking the frequency of values of ORGANIZATION\_TYPE column

**Code:**

newapp\_df['ORGANIZATION\_TYPE'].value\_counts()

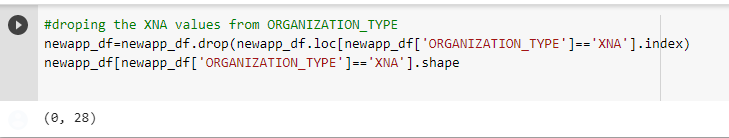


**Desc:** Dropping the XNA values from ORGANIZATION\_TYPE column.

**Code:**

newapp\_df=newapp\_df.drop(newapp\_df.loc[newapp\_df['ORGANIZATION\_TYPE']=='XNA'].index)

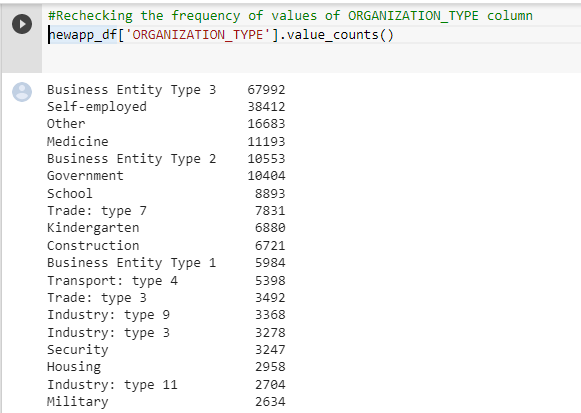
newapp\_df[newapp\_df['ORGANIZATION\_TYPE']=='XNA'].shape



**Desc:** Checking the frequency of values of ORGANIZATION\_TYPE column.

**Code:**

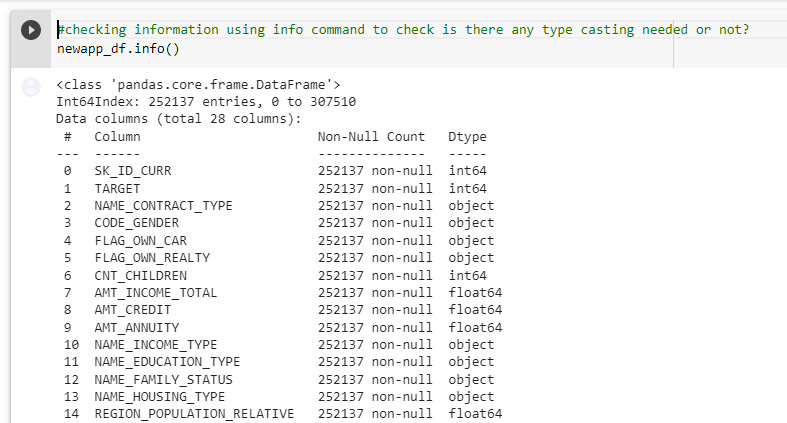
newapp\_df['ORGANIZATION\_TYPE'].value\_counts()



**Desc:** Checking the columns info to find type casting needed or not

**Code:**

newapp\_df.info()



**Desc:**Casting all variable into numeric in the dataset

**Code:**

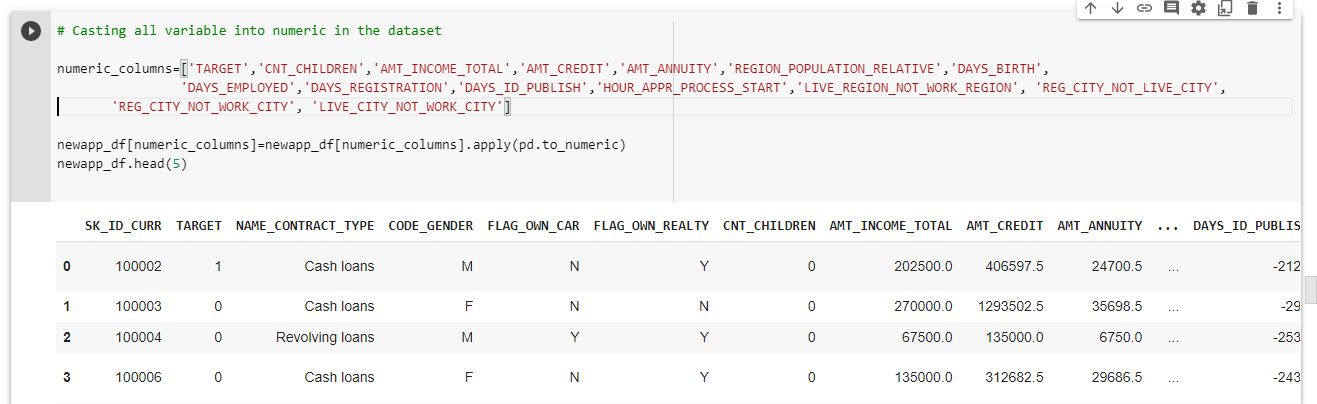
numeric\_columns=['TARGET','CNT\_CHILDREN','AMT\_INCOME\_TOTAL','AMT\_CREDIT','AMT\_ANNUITY','REGION\_POPULATION\_RELATIVE','DAYS\_BIRTH',

'DAYS\_EMPLOYED','DAYS\_REGISTRATION','DAYS\_ID\_PUBLISH','HOUR\_APPR\_PROCESS\_START','LIVE\_REGION\_NOT\_WORK\_REGION', 'REG\_CITY\_NOT\_LIVE\_CITY',

'REG\_CITY\_NOT\_WORK\_CITY', 'LIVE\_CITY\_NOT\_WORK\_CITY']

newapp\_df[numeric\_columns]=newapp\_df[numeric\_columns].apply(pd.to\_numeric)

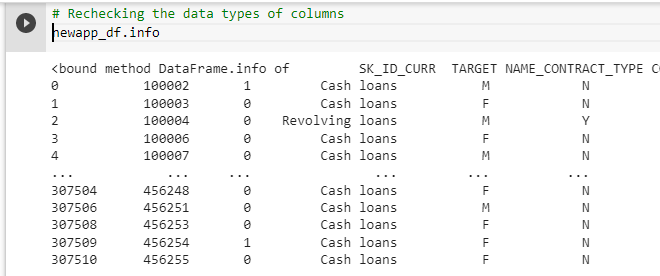
newapp\_df.head(5)

****

**Desc:** Rechecking the data types of column.

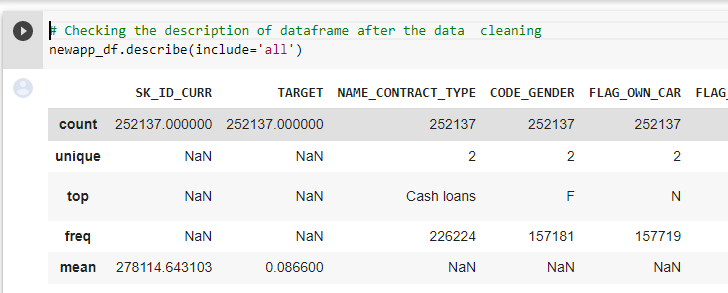
**Code:**

newapp\_df.info

****

**Desc:** Checking the dataframe after data cleaning

**Code:**

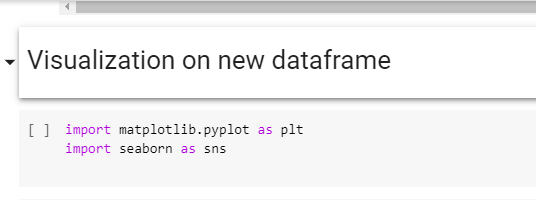
****

**Visualization of Data:**

**Importing Libraries:**

import matplotlib.pyplot as plt

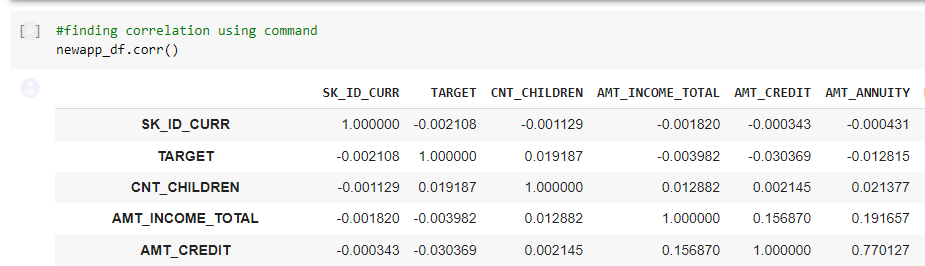
import seaborn as sns



**Desc:**Finding correlation

**Code:**

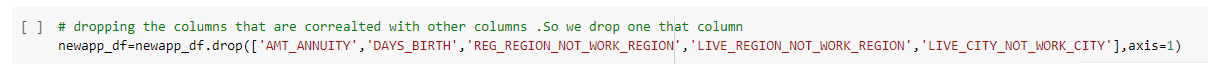
newapp\_df.corr()

****

**Desc:** Dropping one of the columns in a pair which are highly correlated.

**Code:**

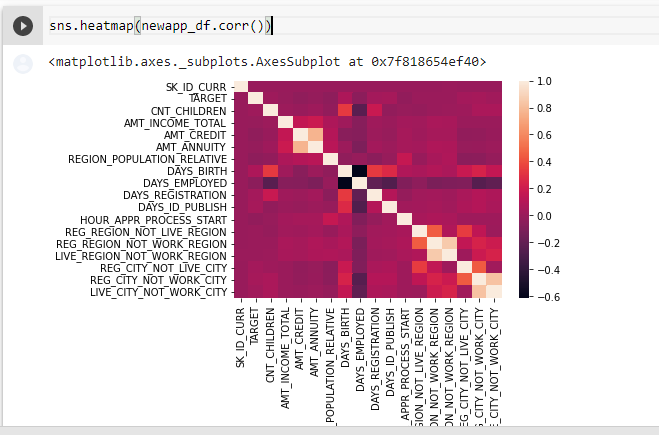
newapp\_df=newapp\_df.drop(['AMT\_ANNUITY','DAYS\_BIRTH','REG\_REGION\_NOT\_WORK\_REGION','LIVE\_REGION\_NOT\_WORK\_REGION','LIVE\_CITY\_NOT\_WORK\_CITY'],axis=1)



**Desc:** Visualization of correlation

**Code:**

sns.heatmap(newapp\_df.corr())



**Desc:** Creating bins,slots and adding a new column into dataframe with bins as label.

**Code:**

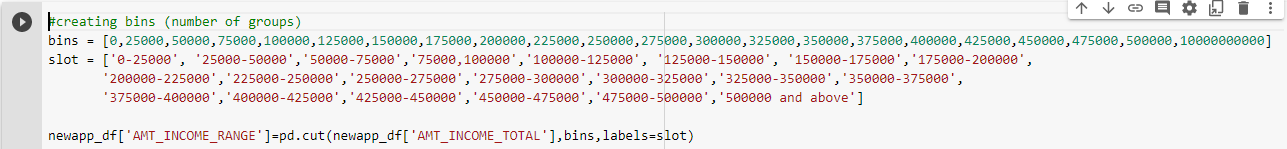
bins = [0,25000,50000,75000,100000,125000,150000,175000,200000,225000,250000,275000,300000,325000,350000,375000,400000,425000,450000,475000,500000,10000000000]

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']

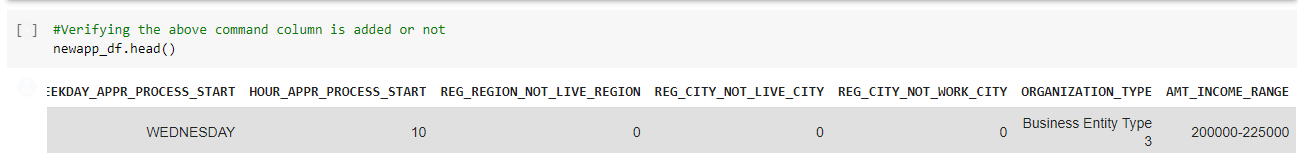
newapp\_df['AMT\_INCOME\_RANGE']=pd.cut(newapp\_df['AMT\_INCOME\_TOTAL'],bins,labels=slot)

****

**Desc:**  Checking whether the new column added in dataset

**Code:**

newapp\_df.head()

****

**Desc:** Creating bins for credit amount and adding new column into dataset.

**Code:**

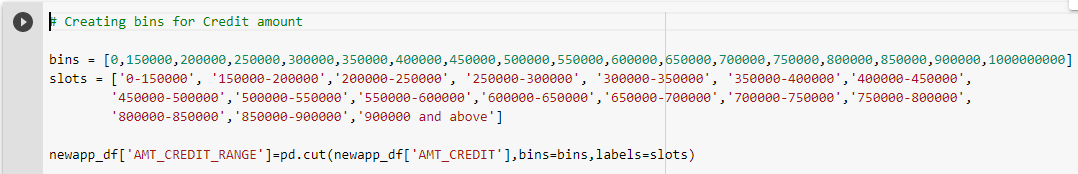
bins = [0,150000,200000,250000,300000,350000,400000,450000,500000,550000,600000,650000,700000,750000,800000,850000,900000,1000000000]

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']

newapp\_df['AMT\_CREDIT\_RANGE']=pd.cut(newapp\_df['AMT\_CREDIT'],bins=bins,labels=slots)

****

**Desc:** Checking whether new column added into dataframe.

**Code:**

newapp\_df.head()

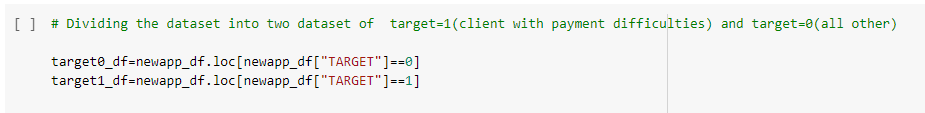
****

**Desc:**Dividing dataset into two dataset of target=1 and target=0.

**Code:**

target0\_df=newapp\_df.loc[newapp\_df["TARGET"]==0]

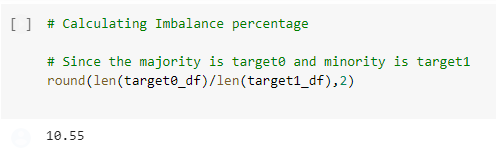
target1\_df=newapp\_df.loc[newapp\_df["TARGET"]==1]

****

**Desc:** Calculating imbalance percentage between target 0 and target 1

**Code:**

round(len(target0\_df)/len(target1\_df),2)

****

**Univariate analysis for categories :-**

Here, we are doing Categorical Univariate Analysis in logarithmic scale for target=0 where clients have no payment difficulties.

**Desc:**

**Code:**

def uniplot(newapp\_df,col,title,hue =None):

sns.set\_style('whitegrid')

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(newapp\_df[col].unique()) + 7 + 4\*len(temp.unique())

fig.set\_size\_inches(width , 8)

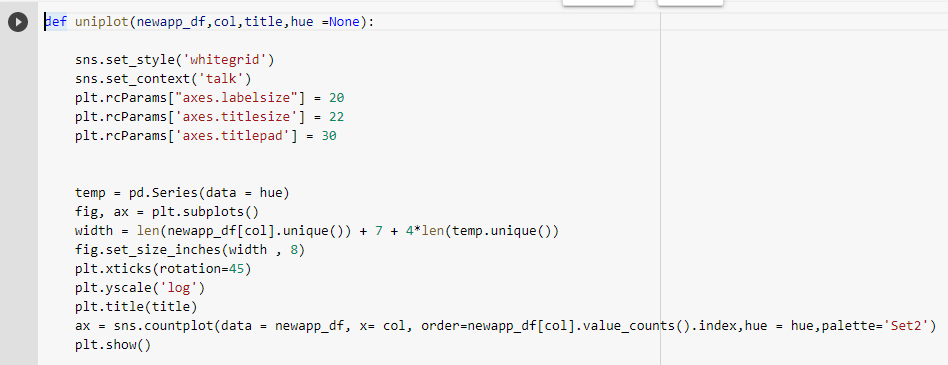
plt.xticks(rotation=45)

plt.yscale('log')

plt.title(title)

ax = sns.countplot(data = newapp\_df, x= col, order=newapp\_df[col].value\_counts().index,hue = hue,palette='Set2')

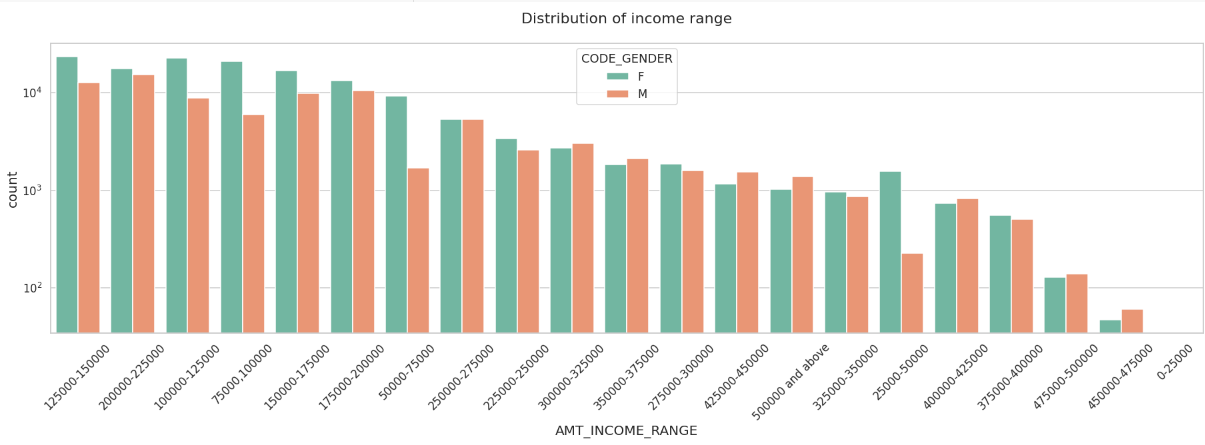
plt.show()

****

**Desc:** Plotting for income range

**Code:**

**uniplot(target0\_df,col='AMT\_INCOME\_RANGE',title='Distribution of income range',hue='CODE\_GENDER')**

****

**Observation:**Female counts are higher than male.

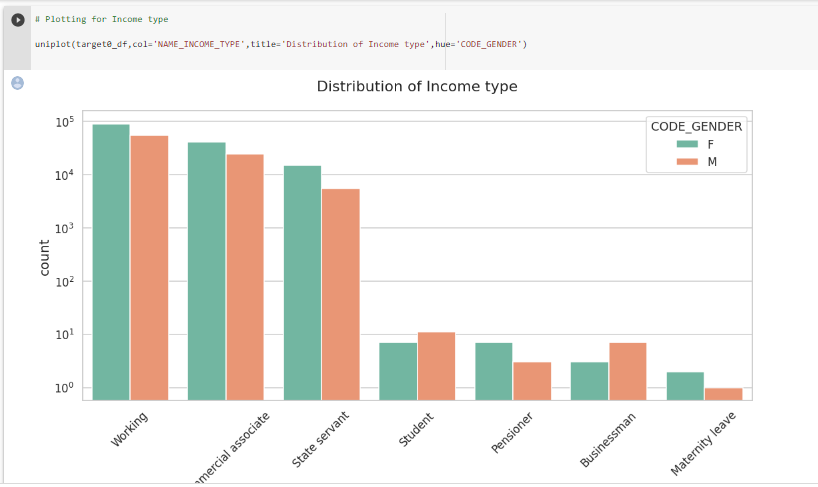
Income range from 100000 to 200000 is having more number of credits.

This graph show that females are more than male in having credits for that range.Very less count for income range 400000 and above

**Desc:** Plotting for income type.

**Code:**

uniplot(target0\_df,col='NAME\_INCOME\_TYPE',title='Distribution of Income type',hue='CODE\_GENDER')

****

**Observation:**For income type ‘working’, ’commercial associate’, and ‘State Servant’ the number of credits are higher than others.

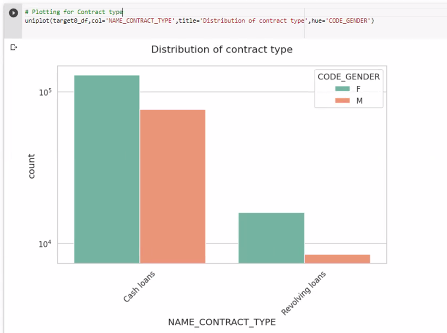
For this Females are having more number of credits than male.

Less number of credits for income type ‘student’ ,’pensioner’, ‘Businessman’ and ‘Maternity leave

**Desc:** Plotting for Contract type

**Code:**

uniplot(target0\_df,col='NAME\_CONTRACT\_TYPE',title='Distribution of contract type',hue='CODE\_GENDER')



**Observation:**For contract type ‘cash loans’ having higher number of credits than ‘Revolving loans’ contract type.

For this also Female is leading for applying credits.

**Desc:**Plotting for Organization type in logarithmic scale

**Code:**

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

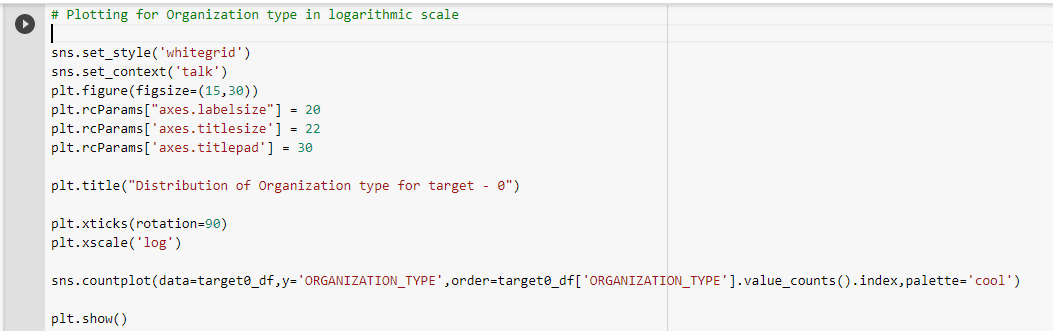
plt.title("Distribution of Organization type for target - 0")

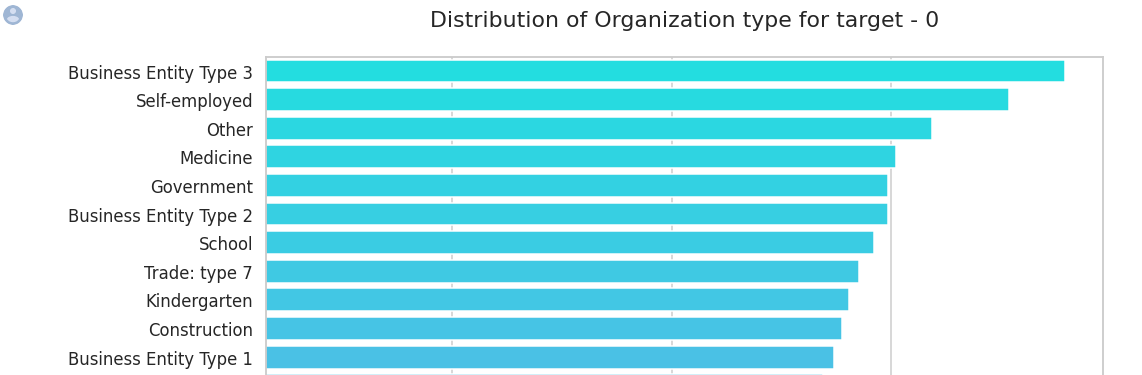
plt.xticks(rotation=90)

plt.xscale('log')

sns.countplot(data=target0\_df,y='ORGANIZATION\_TYPE',order=target0\_df['ORGANIZATION\_TYPE'].value\_counts().index,palette='cool')

plt.show()







Desc:

Code:

target0\_corr=target0\_df.iloc[0:,2:]

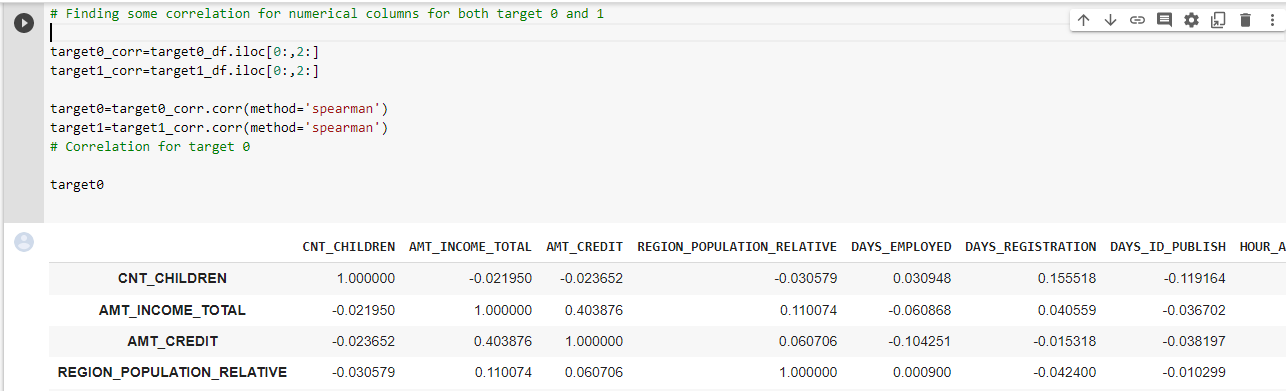
target1\_corr=target1\_df.iloc[0:,2:]

target0=target0\_corr.corr(method='spearman')

target1=target1\_corr.corr(method='spearman')

# Correlation for target 0

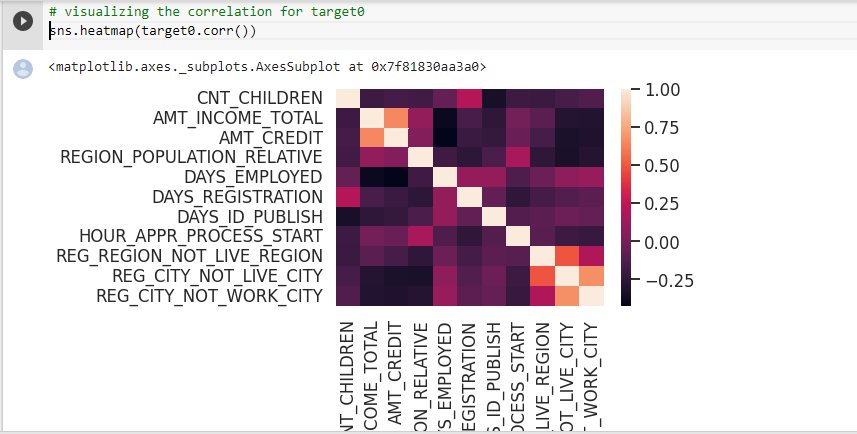
target0



**Desc:**Visualizing the correlation for target0

**Code:**

sns.heatmap(target0.corr())



**Observation:**

As we can see from above correlation heatmap, There are number of observation we can point out for TARGET 0:

Credit amount is inversely proportional to the date of birth, which means Credit amount is higher for low age and vice-versa.

Credit amount is inversely proportional to the number of children client have, means Credit amount is higher for less children count client have and vice-versa.

Income amount is inversely proportional to the number of children client have, means more income for less children client have and vice-versa.

less children client have in densely populated area.

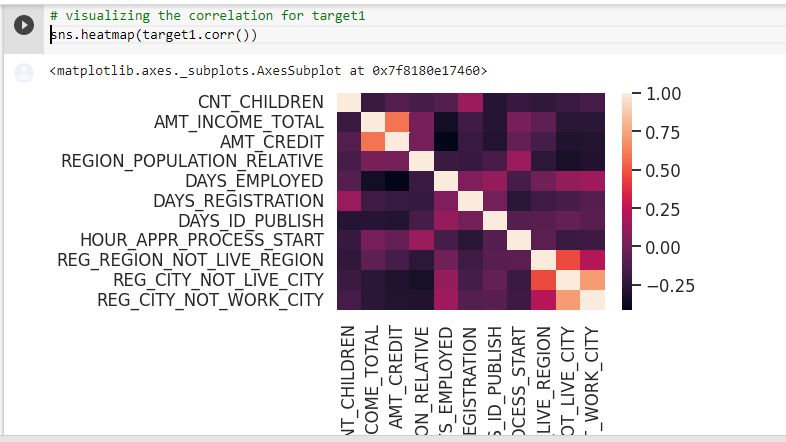
Credit amount is higher to densely populated area.

The income is also higher in densely populated area.

**Desc:**visualizing the correlation for target1

**Code:**

sns.heatmap(target1.corr())

****

**Observation:Target 1:**

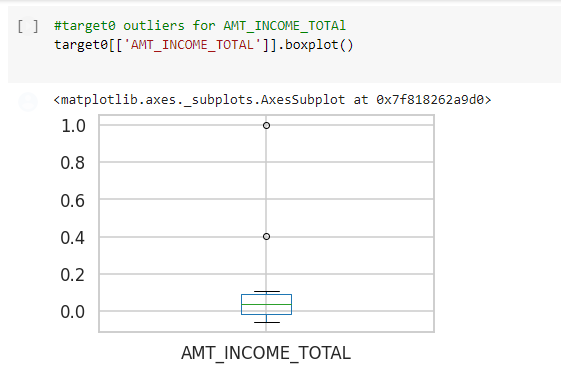
The client's permanent address does not match contact address are having less children and vice-versa

the client's permanent address does not match work address are having less children and vice-versa

**Desc:**target0 outliers for AMT\_INCOME\_TOTAl

**Code:**

target0[['AMT\_INCOME\_TOTAL']].boxplot()



**Observation**:Few points can be concluded from the graph above.

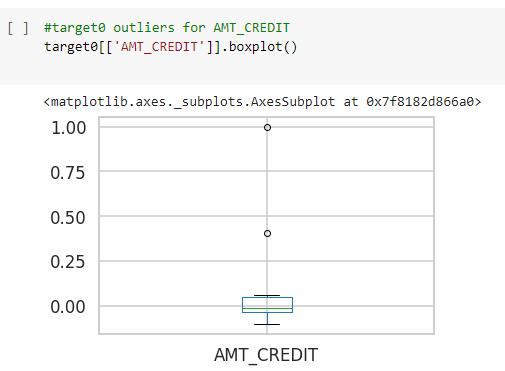
Some outliers are noticed in income amount.

The third quartiles is very slim for income amount.

**Desc:**target0 outliers for AMT\_CREDIT

**Code:**

target0[['AMT\_CREDIT']].boxplot()

****

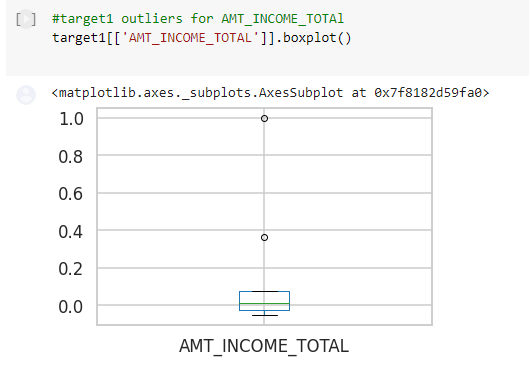
**Observation:**Some outliers are noticed in credit amount.

The first quartile is bigger than the third quartile for credit amount which means most of the credits of clients are present in the first quartile.

**Desc:**

**Code:**

target1[['AMT\_INCOME\_TOTAL']].boxplot()



**Observation:**Few points can be concluded from the graph above.

Some outliers are noticed in income amount.

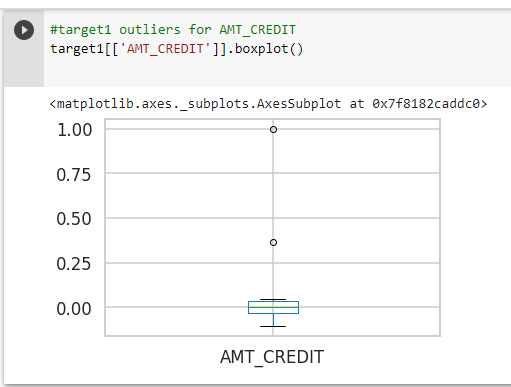
The third quartiles is very slim for income amount.

Most of the clients of income are present in first quartile.

**Desc:**target1 outliers for AMT\_CREDIT

**Code:**

target1[['AMT\_CREDIT']].boxplot()



**Observation:**

Few points can be concluded from the graph above.

Some outliers are noticed in credit amount.

The first quartile is bigger than third quartile for credit amount which means most of the credits of clients are present in the first quartile.

# **Bivariate Analysis:** For Continues & Continues, Categorical & Categorical, Continues & Categorical

**Desc:** Target 0-Box plotting for credit amount.

**Code:**

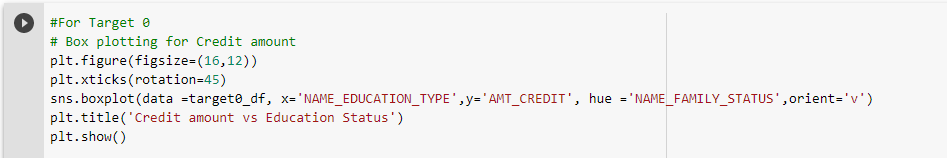
plt.figure(figsize=(16,12))

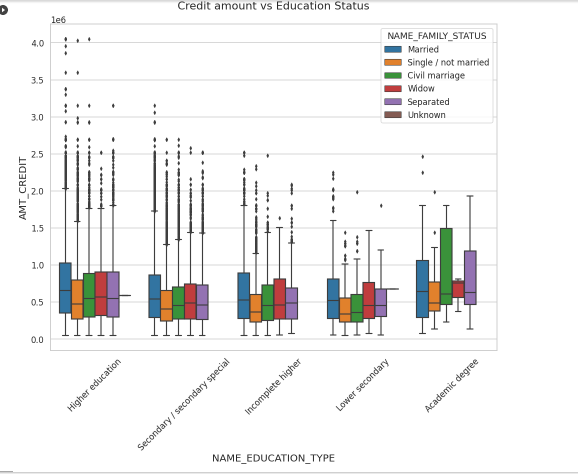
plt.xticks(rotation=45)

sns.boxplot(data =target0\_df, x='NAME\_EDUCATION\_TYPE',y='AMT\_CREDIT', hue ='NAME\_FAMILY\_STATUS',orient='v')

plt.title('Credit amount vs Education Status')

plt.show()





Observation:From the above box plot we can conclude that Family status of 'civil marriage', 'marriage' and 'separated' of Academic degree education are having higher number of credits than others. Also, higher education of family status of 'marriage', 'single' and 'civil marriage' are having more outliers. Civil marriage for Academic degree is having most of the credits in the third quartile.

**Desc:** Target 0: Box plotting for income amount

**Code:**

plt.figure(figsize=(16,12))

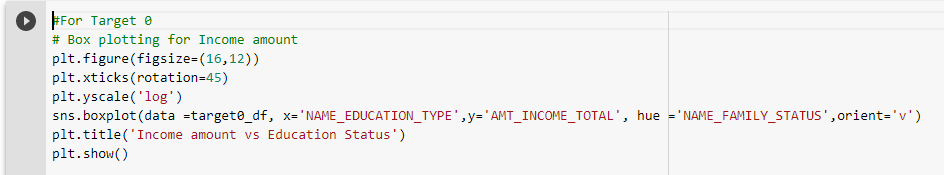
plt.xticks(rotation=45)

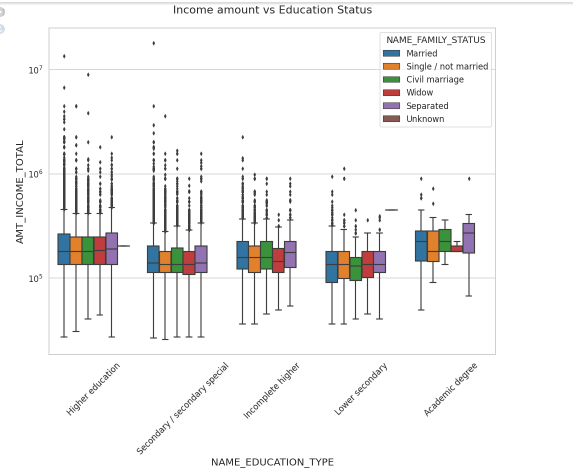
plt.yscale('log')

sns.boxplot(data =target0\_df, x='NAME\_EDUCATION\_TYPE',y='AMT\_INCOME\_TOTAL', hue ='NAME\_FAMILY\_STATUS',orient='v')

plt.title('Income amount vs Education Status')

plt.show()





**Observation:**From above boxplot for Education type 'Higher education' the income amount is mostly equal with family status. It does contain many outliers. Less outlier are having for Academic degree but there income amount is little higher that Higher education. Lower secondary of civil marriage family status are have less income amount than others.

**Desc:**Target 1: Box plotting for credit amount.

**Code:**

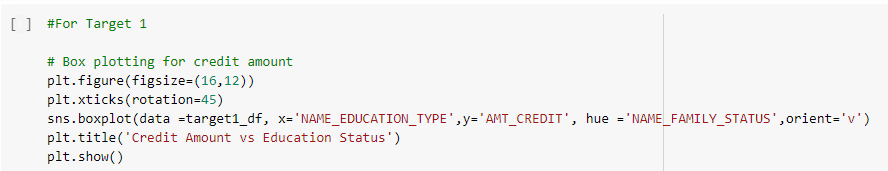
plt.figure(figsize=(16,12))

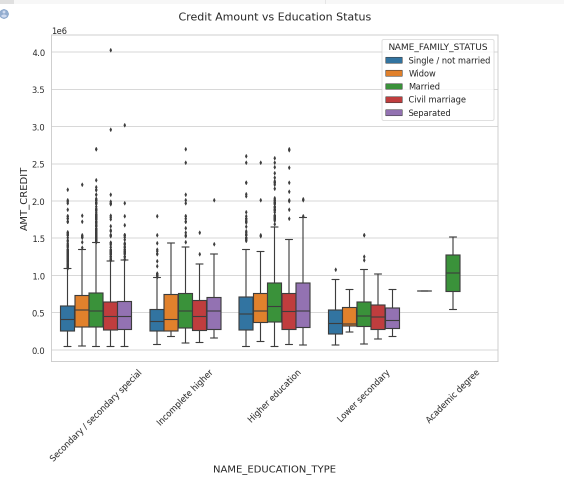
plt.xticks(rotation=45)

sns.boxplot(data =target1\_df, x='NAME\_EDUCATION\_TYPE',y='AMT\_CREDIT', hue ='NAME\_FAMILY\_STATUS',orient='v')

plt.title('Credit Amount vs Education Status')

plt.show()





**Observation:**Less outlier are having for Academic degree but there income amount is little higher that Higher education.

**Desc:** Target 1-box plotting for income amount.

**Code:**

plt.figure(figsize=(16,12))

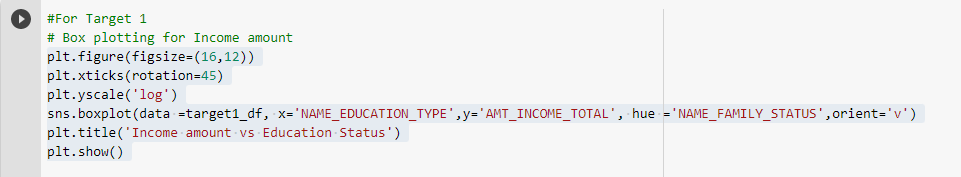
plt.xticks(rotation=45)

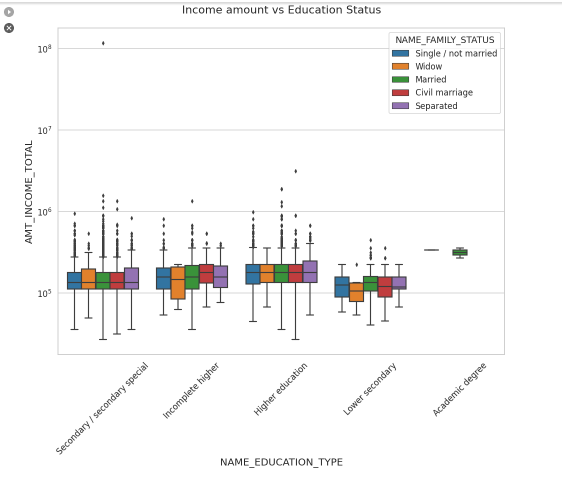
plt.yscale('log')

sns.boxplot(data =target1\_df, x='NAME\_EDUCATION\_TYPE',y='AMT\_INCOME\_TOTAL', hue ='NAME\_FAMILY\_STATUS',orient='v')

plt.title('Income amount vs Education Status')

plt.show()





**Observation**:Less outlier are having for Academic degree but they are having the income amount is little higher that Higher

**Dataset2:**

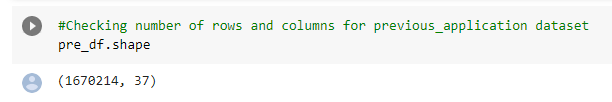
Loading dataset

pre\_df = pd.read\_csv('/content/gdrive/My Drive/AAS/previous\_application.csv')



**Desc:** Checking number of rows, number of columns in dataset.

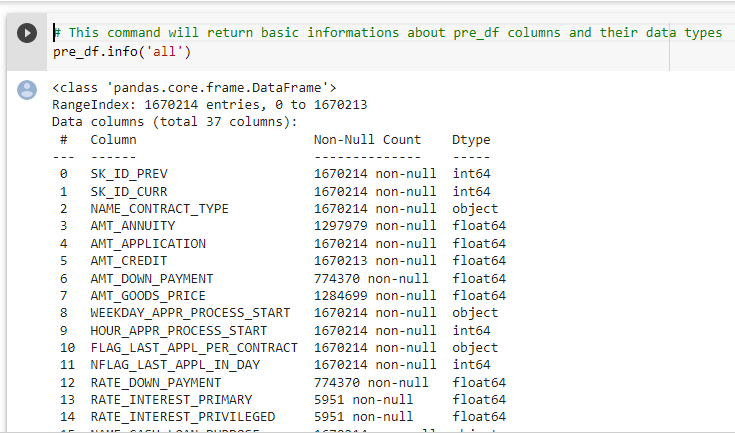
**Code:**pre\_df.shape



**Desc:** Checking the basic info and data types of columns

**Code:**

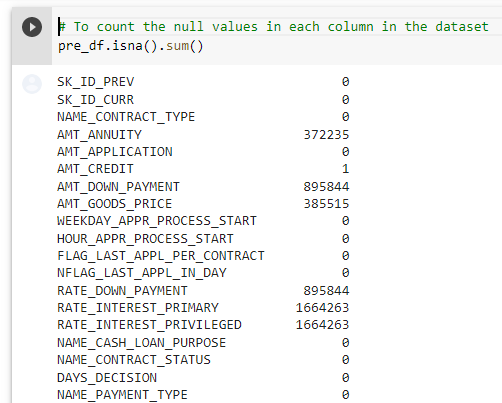
pre\_df.info('all')

****

**Desc:** Counting null values column wise.

**Code:**

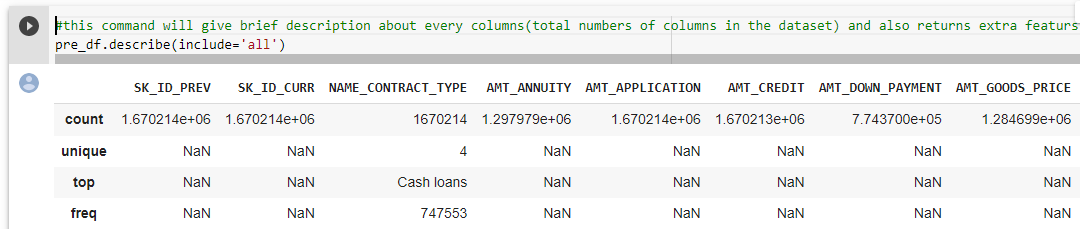
pre\_df.isna().sum()

****

**Desc:** Observing brief description info of dataset

**Code:**

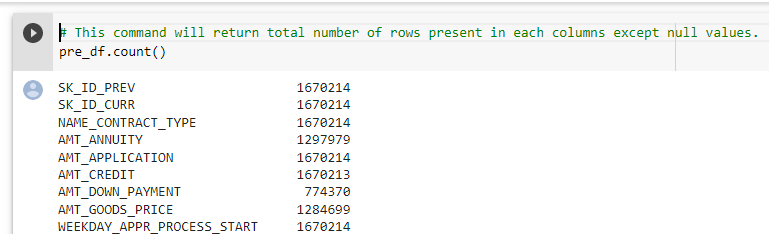
pre\_df.describe(include='all')

****

**Desc:** Returns no of rows in each column

**Code:**

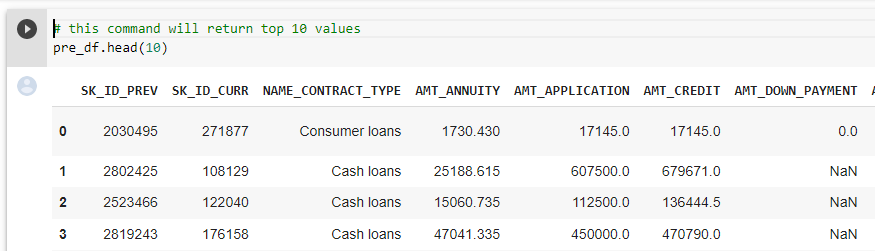
pre\_df.count()

****

**Desc:** Return top 10 rows in dataset.

**Code:**

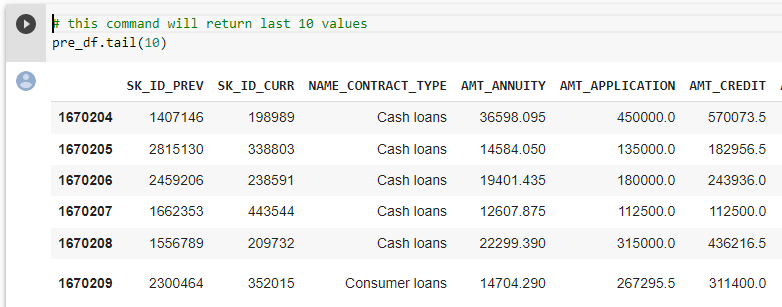
pre\_df.head(10)

****

**Desc:** Return bottom 10 rows in dataset.

**Code:**

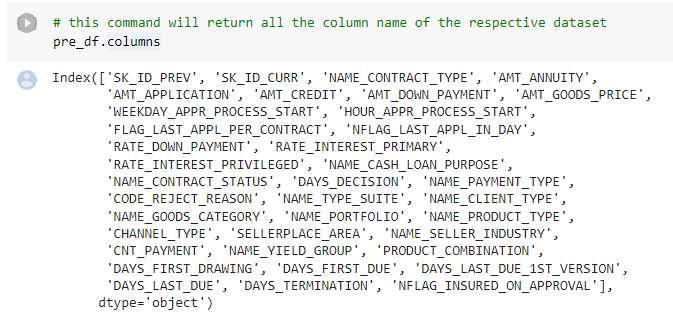
pre\_df.tail(10)

****

**Desc:** Returns columns of dataset.

**Code:**

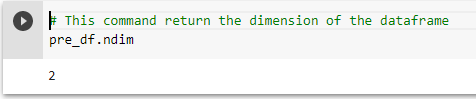
pre\_df.columns

****

**Desc:** Return the dimension of datafarme.

**Code:**

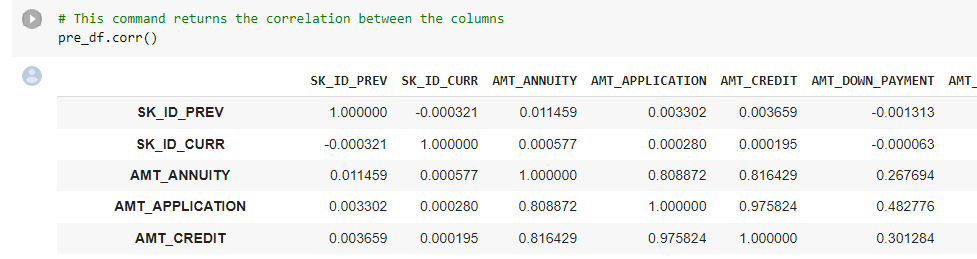
pre\_df.ndim

****

**Desc:** Displays correlation between columns in dataset.

**Code:**

pre\_df.corr()

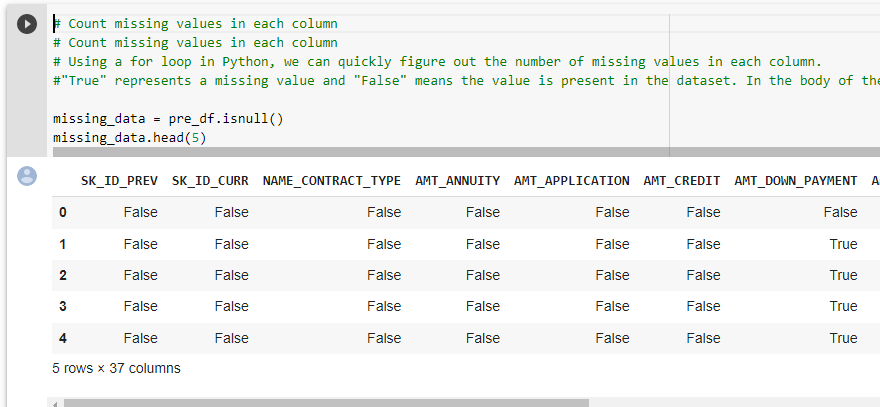
****

**Desc:** Displays whether data present in column or not by True and False.

**Code:**

missing\_data = pre\_df.isnull()

missing\_data.head(5)

****

**Desc:** Finding null values using for loop in python.

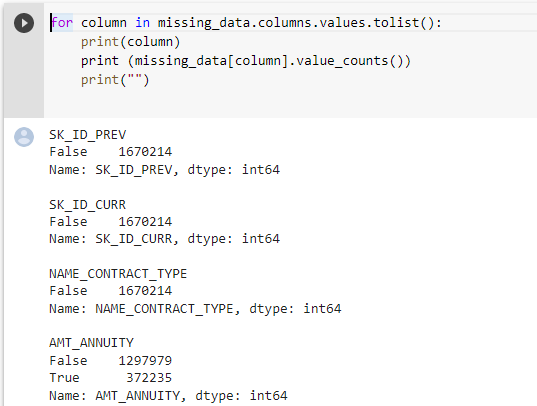
**Code:**

for column in missing\_data.columns.values.tolist():

print(column)

print (missing\_data[column].value\_counts())

print("")

****

**Desc:** Cleaning the missing data.

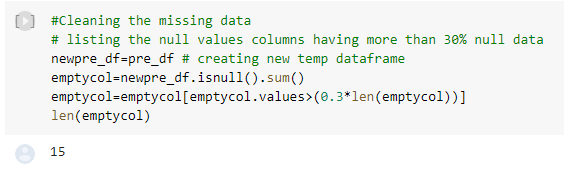
**Code:**

newpre\_df=pre\_df # creating new temp dataframe

emptycol=newpre\_df.isnull().sum()

emptycol=emptycol[emptycol.values>(0.3\*len(emptycol))]

len(emptycol)

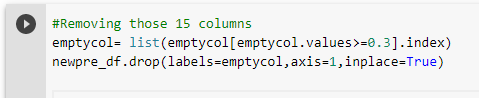
****

**Desc:** Dropping columns having null values.

**Code:**

emptycol= list(emptycol[emptycol.values>=0.3].index)

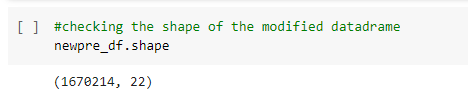
newpre\_df.drop(labels=emptycol,axis=1,inplace=True)

****

**Desc:** Return shape of data frame after dropping

**Code:**

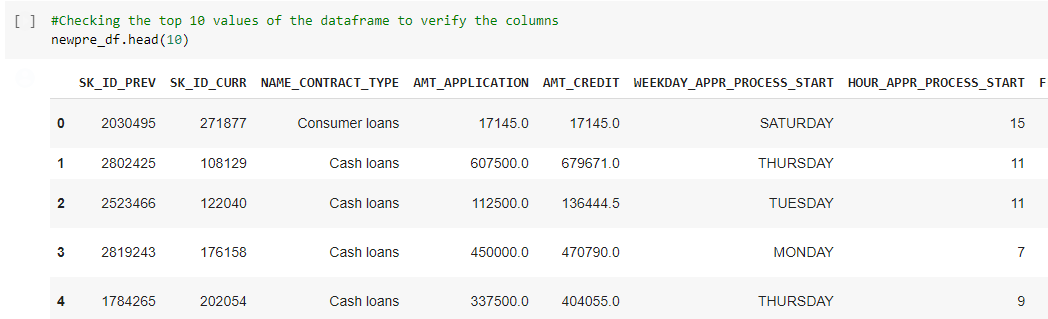
newpre\_df.shape

****

**Desc:** Return top 10 rows

**Code:**

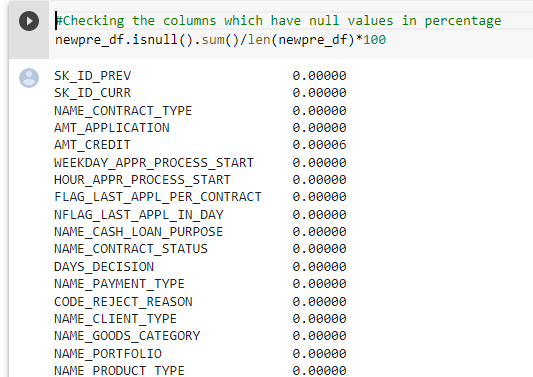
newpre\_df.head(10)

****

**Desc:** Columns having null values in percentage

**Code:**

newpre\_df.isnull().sum()/len(newpre\_df)\*100

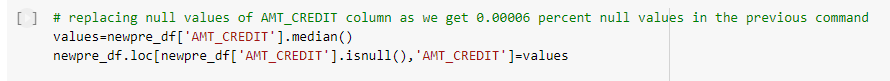
****

**Desc:** Replacing null values of AMT\_CREDIT

**Code:**

values=newpre\_df['AMT\_CREDIT'].median()

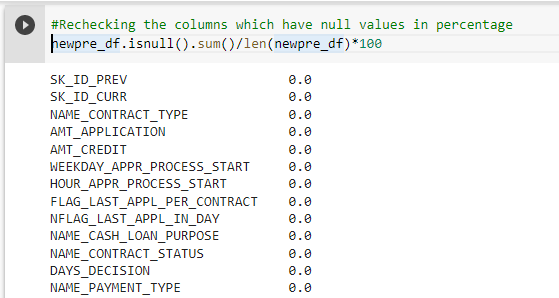
newpre\_df.loc[newpre\_df['AMT\_CREDIT'].isnull(),'AMT\_CREDIT']=values

****

**Desc:**Rechecking the columns which have null values in percentage

**Code:**

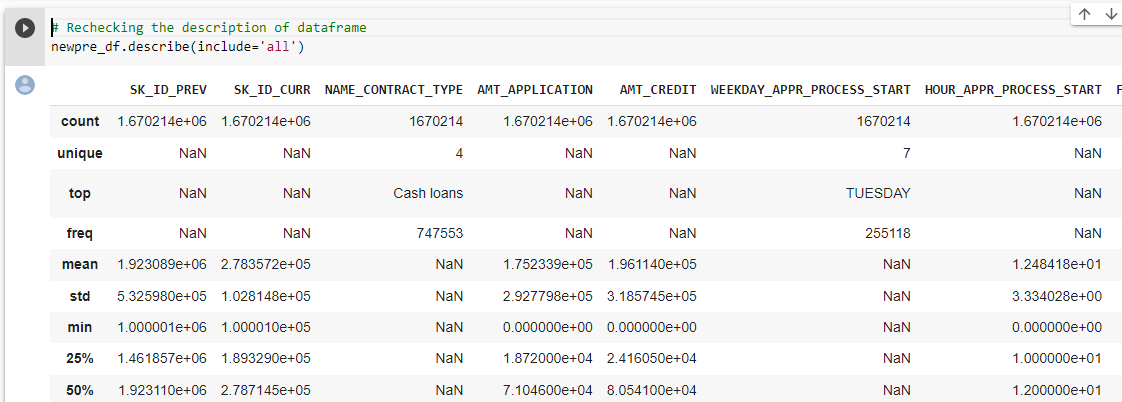
newpre\_df.isnull().sum()/len(newpre\_df)\*100

****

**Desc:** Description of dataframe.

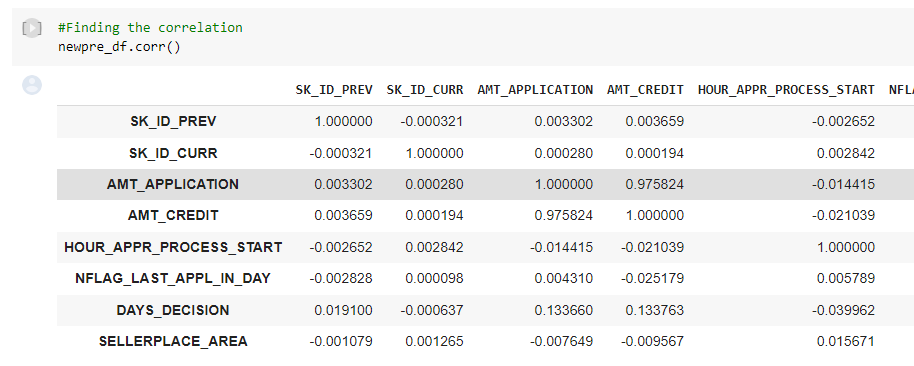
**Code:**

newpre\_df.describe(include='all')

****

**Desc:** Finding the correlation between columns.

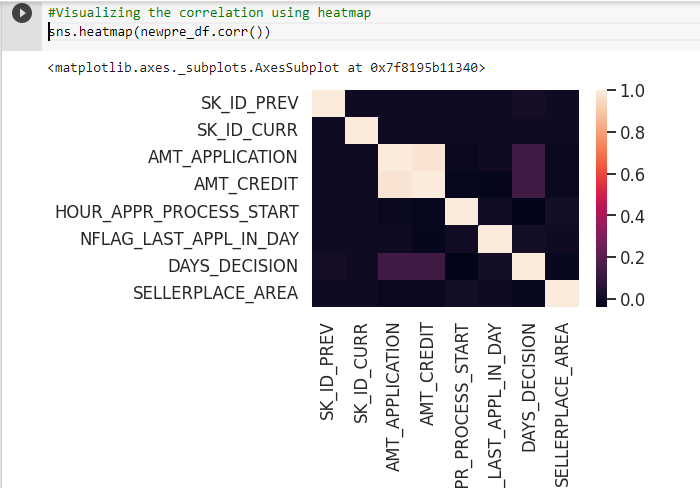
**Code:**newpre\_df.corr()

****

**Desc:** Visualizing correlation using heatmap.

**Code:**

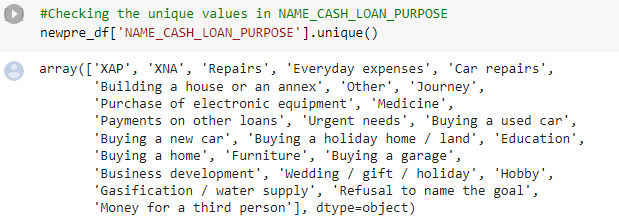
sns.heatmap(newpre\_df.corr())

****

**Desc:** Finding the uniques values in NAME\_CASH\_LOAN\_PURPOSE column

**Code:**

newpre\_df['NAME\_CASH\_LOAN\_PURPOSE'].unique()

****

**Desc**:Dropping the XNA and XPA values from NAME\_CASH\_LOAN\_PURPOSE

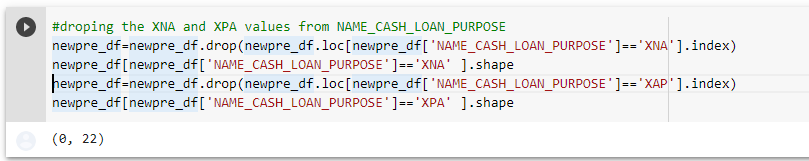
**Code:**

newpre\_df=newpre\_df.drop(newpre\_df.loc[newpre\_df['NAME\_CASH\_LOAN\_PURPOSE']=='XNA'].index)

newpre\_df[newpre\_df['NAME\_CASH\_LOAN\_PURPOSE']=='XNA' ].shape

newpre\_df=newpre\_df.drop(newpre\_df.loc[newpre\_df['NAME\_CASH\_LOAN\_PURPOSE']=='XAP'].index)

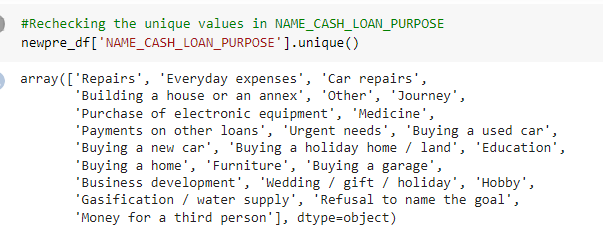
newpre\_df[newpre\_df['NAME\_CASH\_LOAN\_PURPOSE']=='XPA' ].shape

****

**Desc:**Rechecking the unique values in NAME\_CASH\_LOAN\_PURPOSE

**Code:**

newpre\_df['NAME\_CASH\_LOAN\_PURPOSE'].unique()

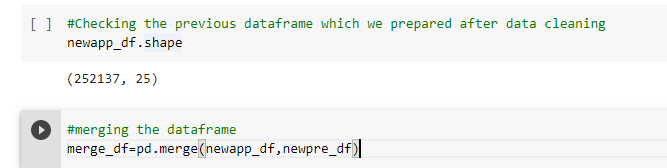
****

**Desc:** Checking shape of dataset and merging two dataframes.

**Code:**

newapp\_df.shape

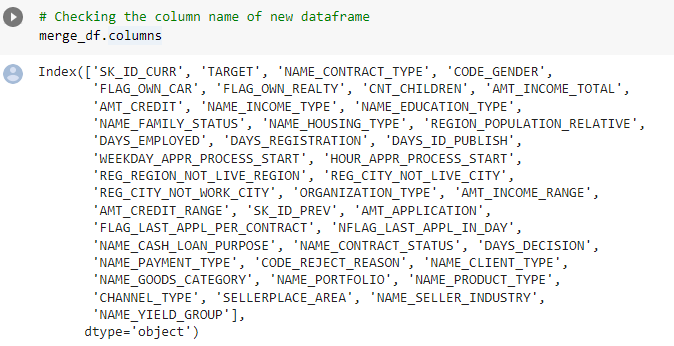
merge\_df=pd.merge(newapp\_df,newpre\_df)

****

**Desc:** Checking the columns of new dataframe.

**Code:**

merge\_df.columns

****

**Desc:**

**Code:**

merge\_df = merge\_df.rename({'NAME\_CONTRACT\_TYPE\_' : 'NAME\_CONTRACT\_TYPE','AMT\_CREDIT\_':'AMT\_CREDIT','AMT\_ANNUITY\_':'AMT\_ANNUITY',

'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',

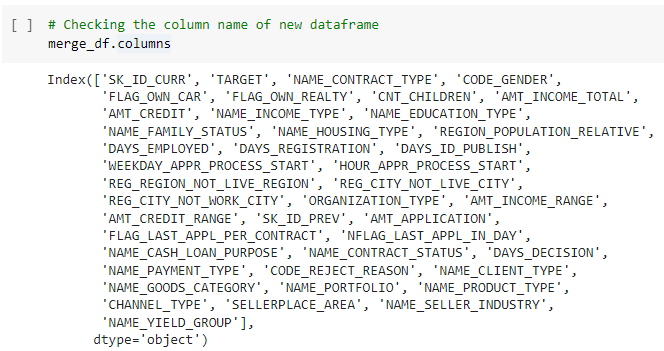
'HOUR\_APPR\_PROCESS\_STARTx':'HOUR\_APPR\_PROCESS\_START\_PREV'}, axis=1)

****

**Desc:**Checking the column name of new dataframe

**Code:**

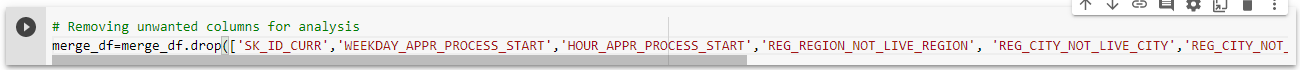
merge\_df.columns

****

**Desc:**Checking columns after dropping unnecessary columns

**Code:**

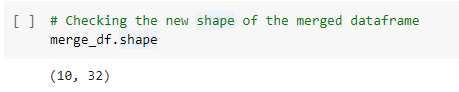
merge\_df=merge\_df.drop(['SK\_ID\_CURR','WEEKDAY\_APPR\_PROCESS\_START','HOUR\_APPR\_PROCESS\_START','REG\_REGION\_NOT\_LIVE\_REGION', 'REG\_CITY\_NOT\_LIVE\_CITY','REG\_CITY\_NOT\_WORK\_CITY', 'ORGANIZATION\_TYPE', 'AMT\_INCOME\_RANGE','FLAG\_LAST\_APPL\_PER\_CONTRACT', 'NFLAG\_LAST\_APPL\_IN\_DAY'],axis=1)

****

**Desc:** Checking the shape of dataframe.

**Code:**

merge\_df.shape

****

# Univariate Analysis on merged Dataframe

**Desc:** Distribution of contract status in logarithmic scale

**Code:**

sns.set\_style('whitegrid')

sns.set\_context('talk')

#fig = plt.figure(figsize=(50,20))

plt.figure(figsize=(5,10))

plt.rcParams["axes.labelsize"] = 20

plt.rcParams['axes.titlesize'] = 22

plt.rcParams['axes.titlepad'] = 30

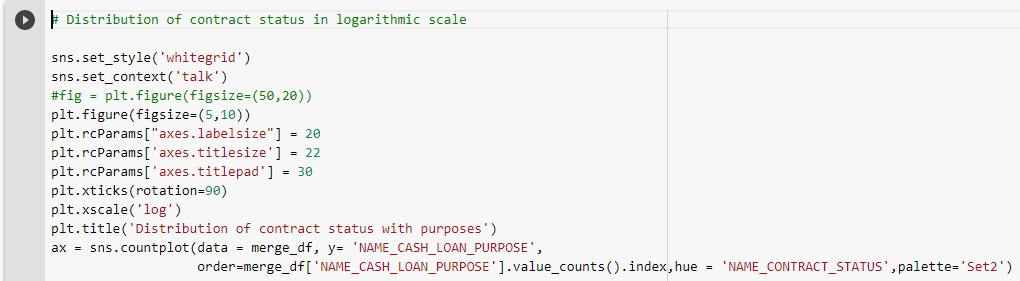
plt.xticks(rotation=90)

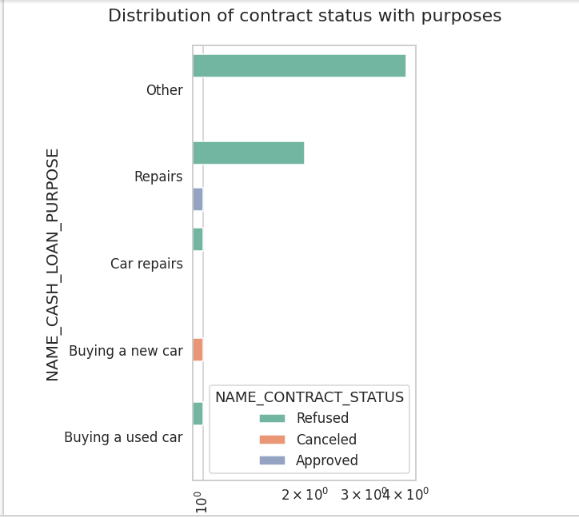
plt.xscale('log')

plt.title('Distribution of contract status with purposes')

ax = sns.countplot(data = merge\_df, y= 'NAME\_CASH\_LOAN\_PURPOSE',

order=merge\_df['NAME\_CASH\_LOAN\_PURPOSE'].value\_counts().index,hue = 'NAME\_CONTRACT\_STATUS',palette='Set2')

****

****

**Desc:**Distribution of contract status

**Code:**

sns.set\_style('whitegrid')

sns.set\_context('talk')

plt.figure(figsize=(5,8))

plt.rcParams["axes.labelsize"] = 20

plt.rcParams['axes.titlesize'] = 22

plt.rcParams['axes.titlepad'] = 30

plt.xticks(rotation=90)

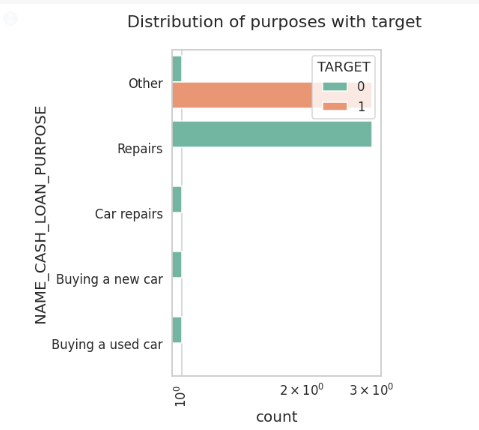
plt.xscale('log')

plt.title('Distribution of purposes with target ')

ax = sns.countplot(data = merge\_df, y= 'NAME\_CASH\_LOAN\_PURPOSE',

order=merge\_df['NAME\_CASH\_LOAN\_PURPOSE'].value\_counts().index,hue = 'TARGET',palette='Set2')

****

****

**Observation:**

Most rejection of loans came from purpose 'repairs'.

For education purposes we have equal number of approves and rejection

Paying other loans and buying a new car is having significantly higher rejection than approval.

**Bivariate Analysis**

**Desc:**Box plotting for Credit amount in logarithmic scale

**Code:**

plt.figure(figsize=(4,5))

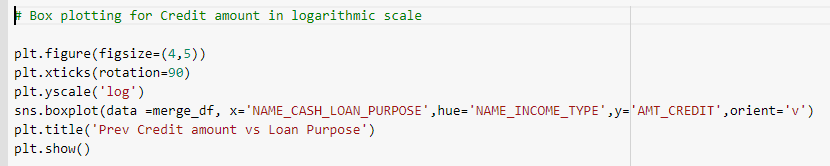
plt.xticks(rotation=90)

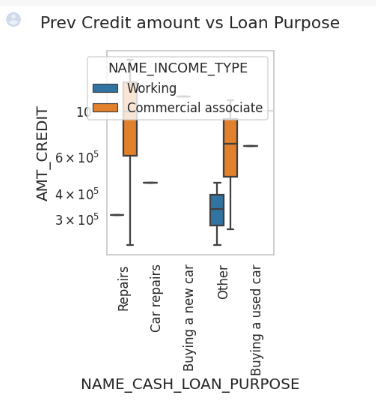
plt.yscale('log')

sns.boxplot(data =merge\_df, x='NAME\_CASH\_LOAN\_PURPOSE',hue='NAME\_INCOME\_TYPE',y='AMT\_CREDIT',orient='v')

plt.title('Prev Credit amount vs Loan Purpose')

plt.show()



****

**Observation:**Loan purposes with 'Repairs' are facing more difficulties in payment on time.There are few places where loan payment is significantly higher than facing difficulties. They are 'Buying a garage', 'Business development', 'Buying land','Buying a new car' and 'Education'

Hence we can focus on these purposes for which the client is having minimal payment difficulties.

**Desc:**Box plotting for Credit amount prev vs Housing type in logarithmic scale

**Code:**

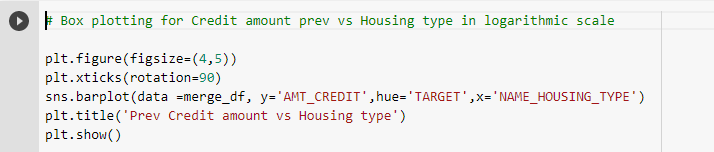
plt.figure(figsize=(4,5))

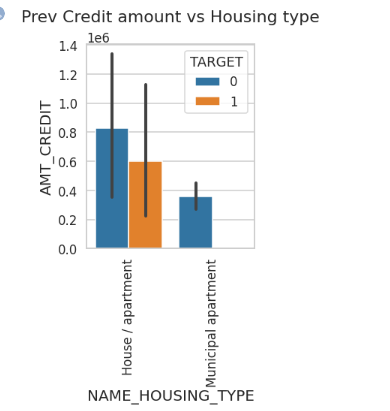
plt.xticks(rotation=90)

sns.barplot(data =merge\_df, y='AMT\_CREDIT',hue='TARGET',x='NAME\_HOUSING\_TYPE')

plt.title('Prev Credit amount vs Housing type')

plt.show()

****

****

**Observation:**Here for Housing type, office apartment is having higher credit of target 0 and co-op apartment is having higher credit of target 1. So, we can conclude that banks should avoid giving loans to the housing type of co-op apartment as they are having difficulties in payment. Banks can focus mostly on housing type with parents or House\apartment or municipal apartment for successful payments.

**CONCLUSION**

1. Banks should focus more on contract type ‘Student’ ,’pensioner’ and ‘Businessman’ with housing ‘type other than ‘Co-op apartment’ for successful payments.

2. Banks should focus less on income type ‘Working’ as they are having the most number of unsuccessful payments.

3. Also with loan purposes ‘Repair’ is having a higher number of unsuccessful payments on time.

4. Get as much as clients from housing type ‘With parents’ as they are having the least number of unsuccessful payments.

**Model Building:**

Now,all the data is numeric, so we can create a logistic regression model.

The logistic regression model provides an appropriate statistical treatment of these correlations.

If we prepare a model based on this dataset , we can predict that loan is Accepted or not by passing the parameter values.