**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:**

1. **If the applicant is likely to repay the loan, then not approving the loan results in a loss of business to the company**
2. **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.**

**Step 1: Loading the Data from Google Drive**

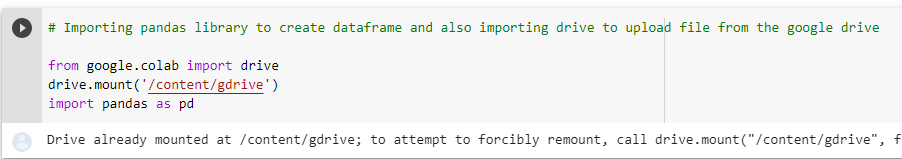
As downloading directly from drive and loading into google colab missing some rows directly loaded files from google drive.

**Code**:

from google.colab import drive

drive.mount('/content/gdrive')

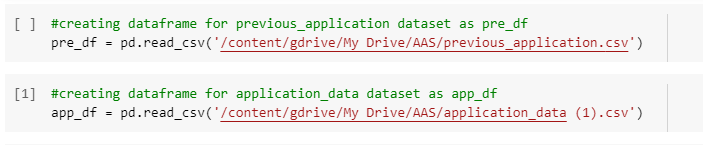
import pandas as pd



Step 2: 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')



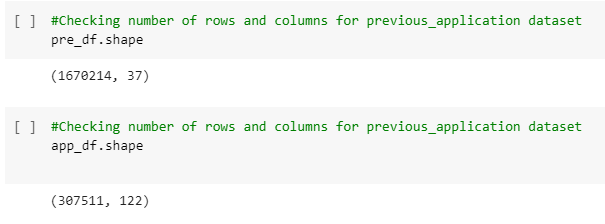
Step 3: Checking the Dataset Shape

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

**Code**:

pre\_df.shape

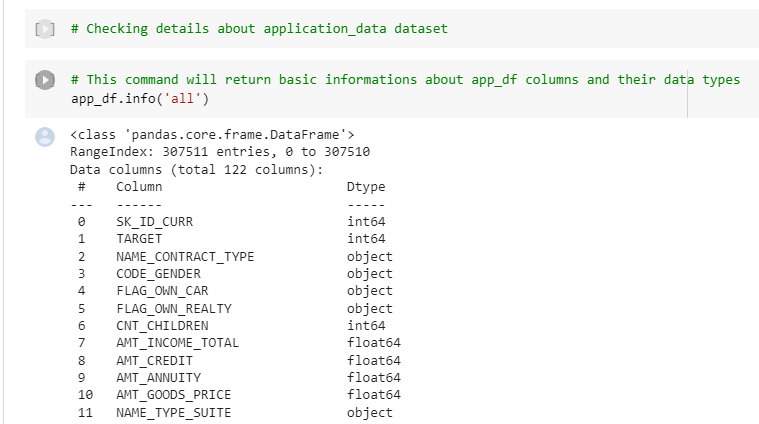
app\_df.shape



Step 4: Checking Dataset info including index, Column Name, Data types

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

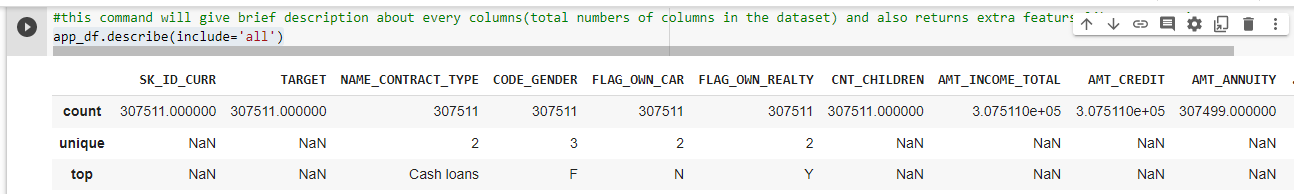
**Code:**app\_df.info('all')



**Step 5:** Checking the description of the data in the Data Frame for each column

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

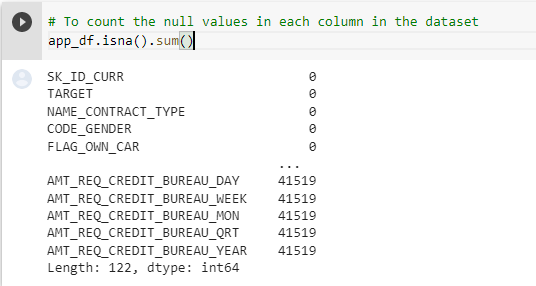
**Code:**app\_df.describe(include='all')



Step 6: Counting Null Value in Dataset according to columns

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

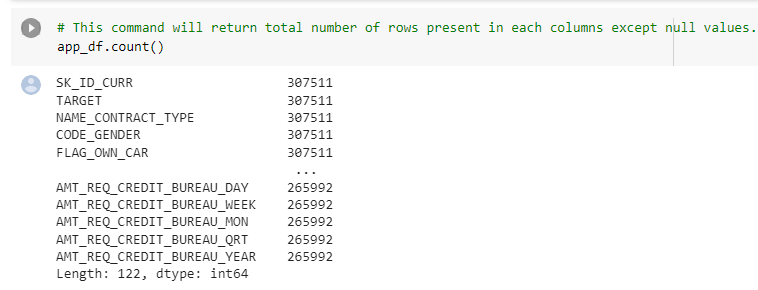
**Code:**app\_df.isna().sum()



Step 7: Checking the No. of rows containing Present data Except Null Value

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

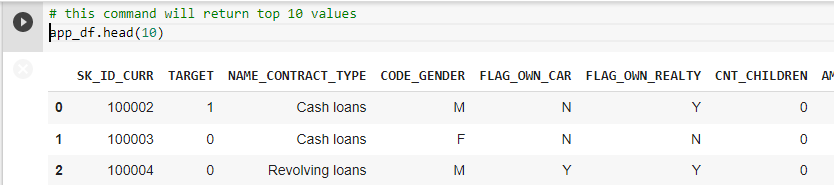
**Code:**app\_df.count()



Step 8: Checking top 10 Record of the data frame

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

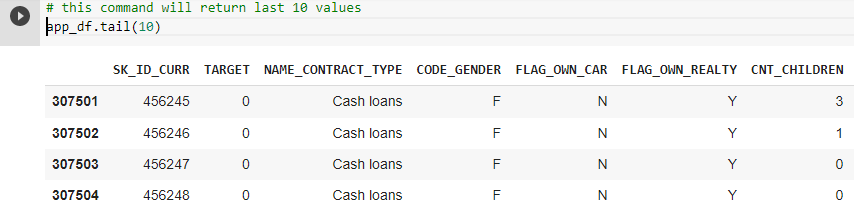
**Code**:app\_df.head(10)



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

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

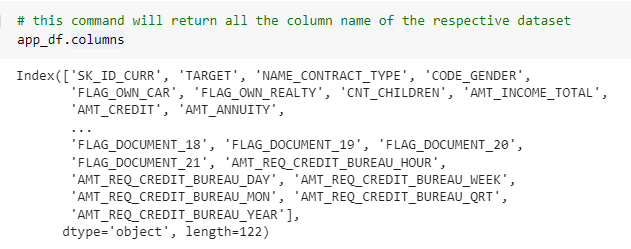
Step 9: Checking last 10 Record of the data frame



Step 10: Checking Columns Name available in data frame

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

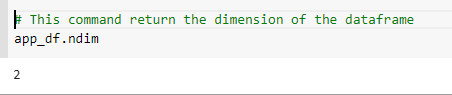
**Code:**app\_df.columns



Step 11: Checking Data Frame Dimension

**Desc:**Checking dimensions of dataset

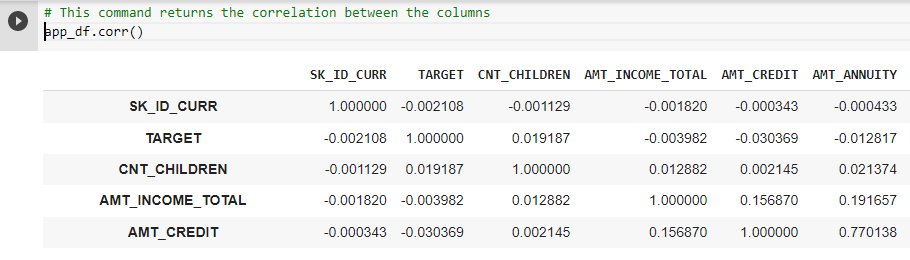
**Code**:app\_df.ndim



Step 12: Checking the Data Frame Correlation between columns

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

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



**Cleaning the Data**

Step 13: Dropping the unwanted Columns using drop command

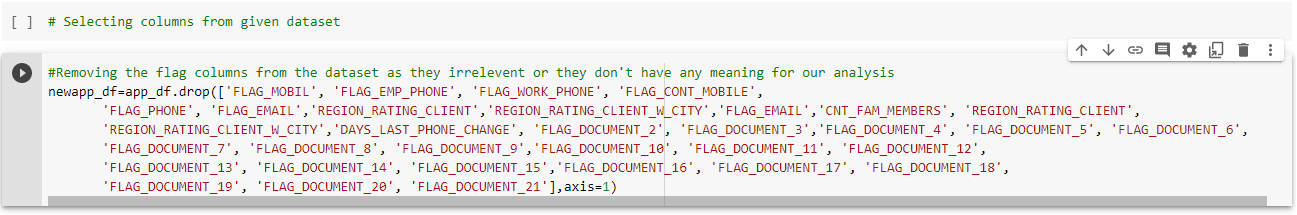
**Desc:**Dropping flag columns from dataset as they are not useful in data analysis.

**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',

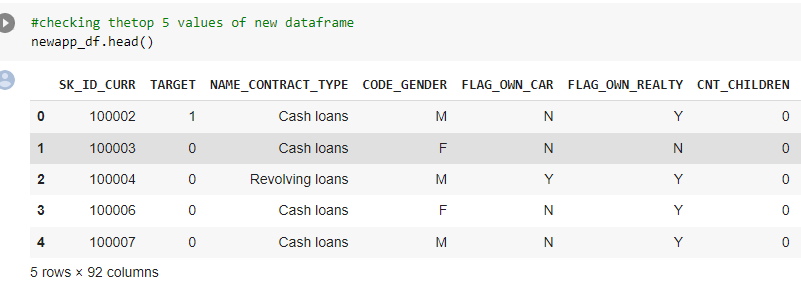
'FLAG\_DOCUMENT\_3','FLAG\_DOCUMENT\_4', 'FLAG\_DOCUMENT\_5', 'FLAG\_DOCUMENT\_6', 'FLAG\_DOCUMENT\_7', 'FLAG\_DOCUMENT\_8', 'FLAG\_DOCUMENT\_9','FLAG\_DOCUMENT\_10', 'FLAG\_DOCUMENT\_11', 'FLAG\_DOCUMENT\_12','FLAG\_DOCUMENT\_13', 'FLAG\_DOCUMENT\_14', 'FLAG\_DOCUMENT\_15','FLAG\_DOCUMENT\_16', 'FLAG\_DOCUMENT\_17', 'FLAG\_DOCUMENT\_18','FLAG\_DOCUMENT\_19', 'FLAG\_DOCUMENT\_20', 'FLAG\_DOCUMENT\_21'],axis=1)



Step 14: Rechecking the data frame record

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

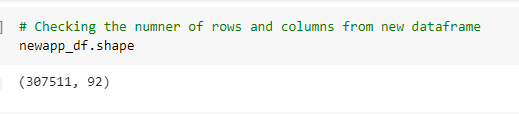
**Code:**newapp\_df.head()



Step 15: Rechecking Shape after dropping unwanted Columns

**Desc:**Checking the shape after dropping

**Code:**newapp\_df.shape



Step 16: Checking the null value in each column of a Data Frame who having more than 30 percent of total null data

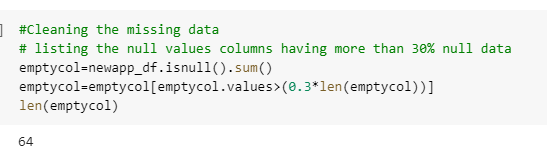
**Desc:** Identifying null values column wise which are having more than 30% null data and dropping those columns from dataset.

**Code:**

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

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

len(emptycol)



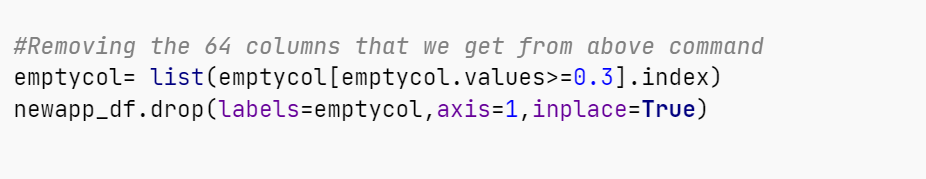
Step 17: Dropping the null value in data frame

**Desc:** Dropping the null value in data frame whose column has more than 30 % percent null Value

**Code:**

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

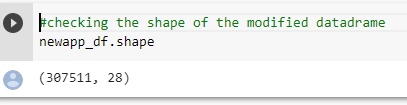
newapp\_df.drop(labels=emptycol,axis=1,inplace=True)

****

Step 18: Checking the shape of DataFrame

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

**Code:**newapp\_df.shape

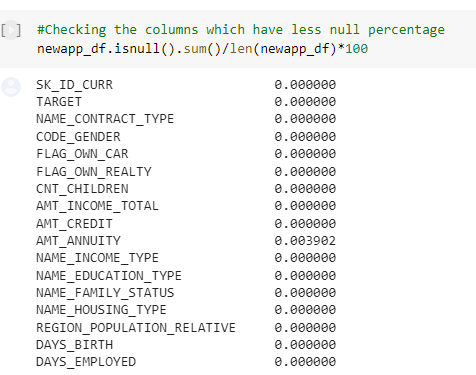


**Step 19: Checking the null percentage in DataFrame in each of the column**

**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. |

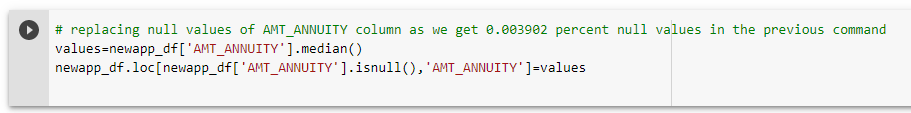
Step 20: Replacing the null Value with median of the column

**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

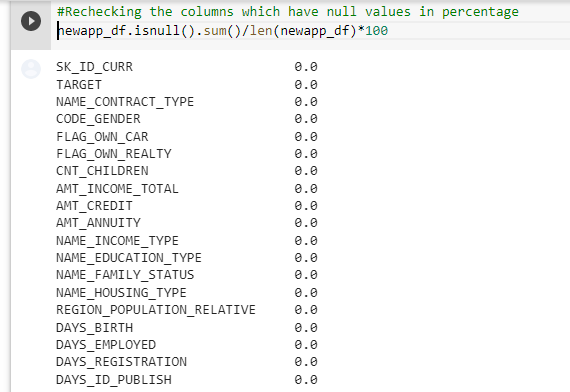


Step 21: After filling the null value rechecking the null percentage in each column

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

**Code:**

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

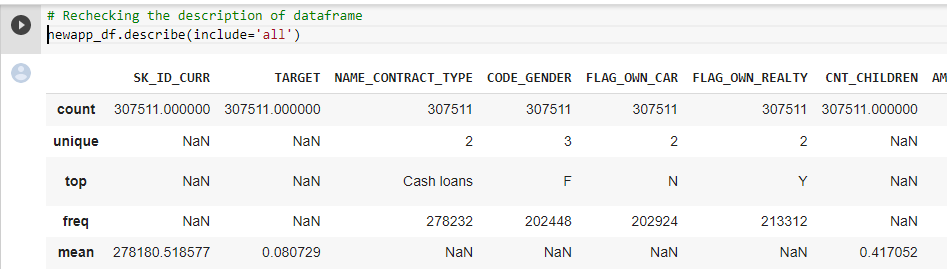


Step 22: Checking Description of DataFrame

**Desc:** Rechecking the description of dataframe

**Code:**

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

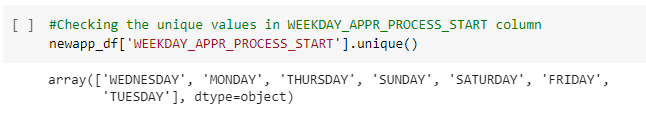


Step 23: Finding the unique values in Column

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

**Code:**

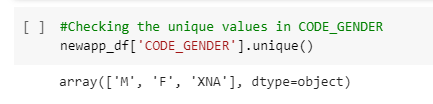
newapp\_df['WEEKDAY\_APPR\_PROCESS\_START'].unique()



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

**Code:**

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

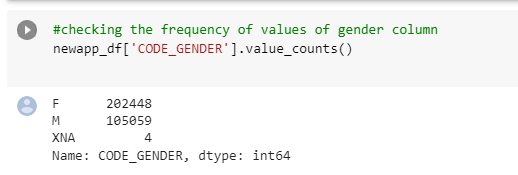


Step 24: Checking the Frequent (Repeated value in Column)

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

**Code:**

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



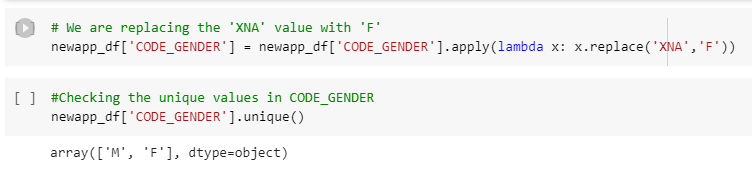
Step 25: Replacing the frequent Value (Repeated value in Column)

**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()



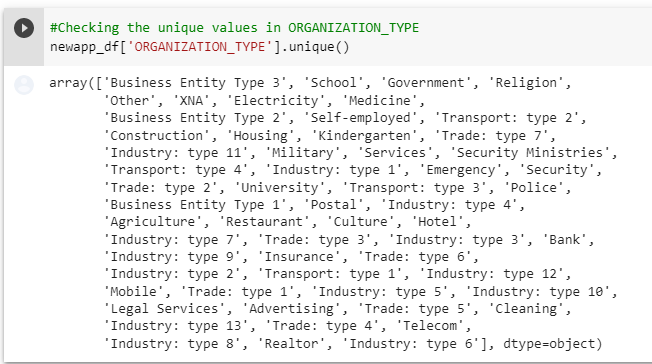
|  |
| --- |
| **Observation**: So, for column ' CODE\_GENDER ', we have total count of 4 rows of which are having 'XNA' values. Hence, we are replacing the ‘XNA’ with ‘F’ because ‘F’ is more frequently repeated. So, it has high probability chances that this 4 are ‘F’. |

**Step 26: Checking unique Value of column**

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

**Code:**

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

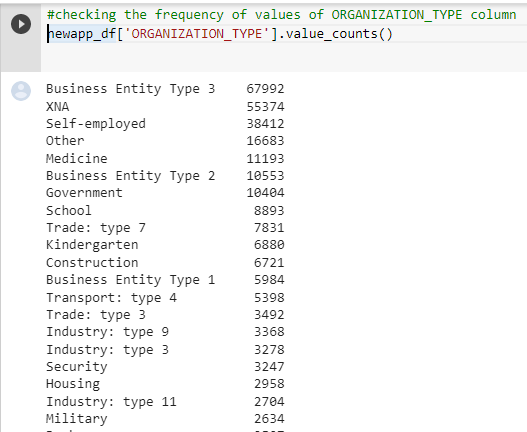


Step 27: Checking the Frequent(Repeated value in Column)

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

**Code:**

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



|  |
| --- |
| **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 these values. Hence if we drop the rows of total 55374, will not have any major impact on our dataset. |

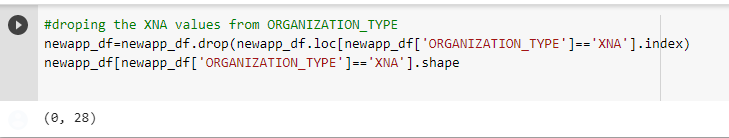
Step 28: Dropping the unwanted value from column

**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

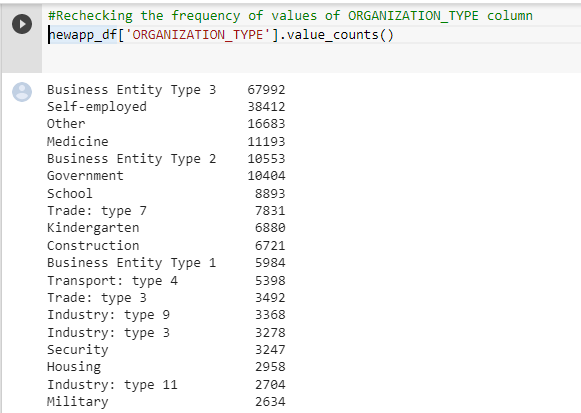


Step 29: Rechecking the most

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

**Code:**

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

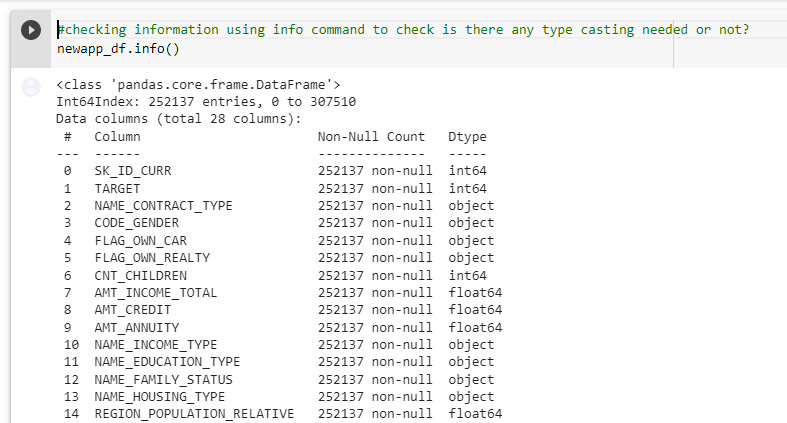


**Step 30: Checking Data Frame information**

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

**Code:**

newapp\_df.info()



Step 31: Typecasting the variable/Column in data frame to numeric

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

**Code:**

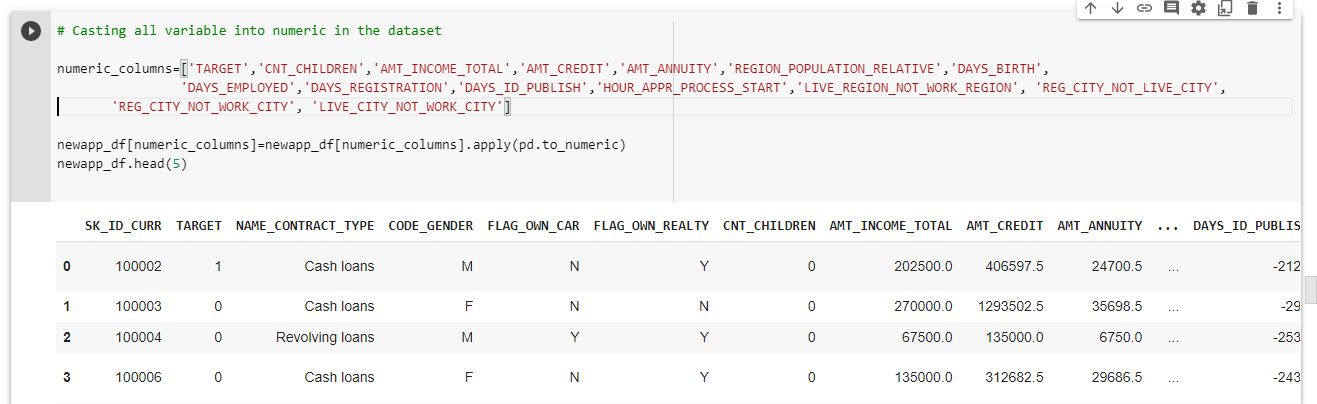
numeric\_columns=['TARGET','CNT\_CHILDREN','AMT\_INCOME\_TOTAL','AMT\_CREDIT','AMT\_ANNUITY','REGION\_POPULATION\_RELATIVE','DAYS\_BIRTH',

'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)

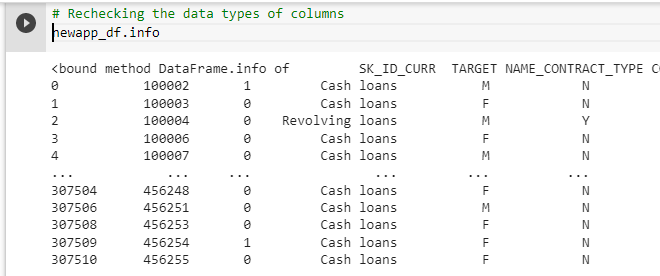
****

**Step 32: Rechecking the data type of each column in Data Frame using info method**

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

**Code:**

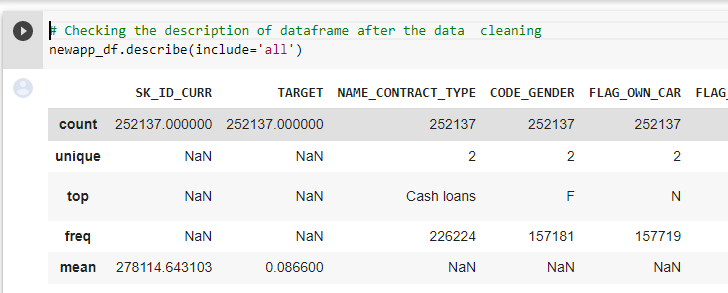
newapp\_df.info

****

**Step 33: Checking the Description of data using description method**

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

**Code:**

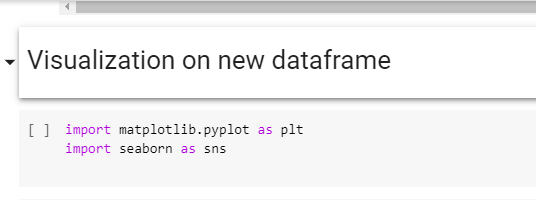
****

**Visualization of Data:**

**Importing Libraries:**

import matplotlib.pyplot as plt

import seaborn as sns

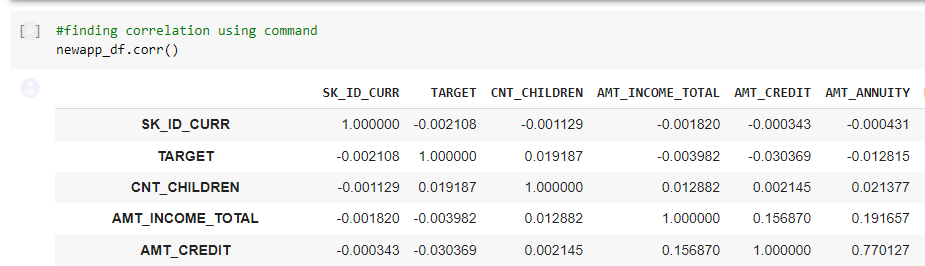


**Step 34: Finding Correlation between Columns/Variables (30 % null value cleaned)**

**Desc:**Finding correlation

**Code:**

newapp\_df.corr()

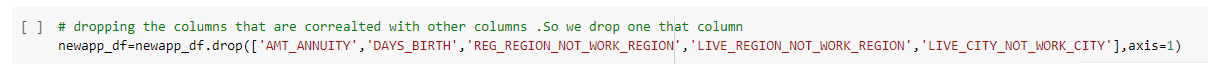
****

**Step 35: Dropping the Correlated columns/Variables**

**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)

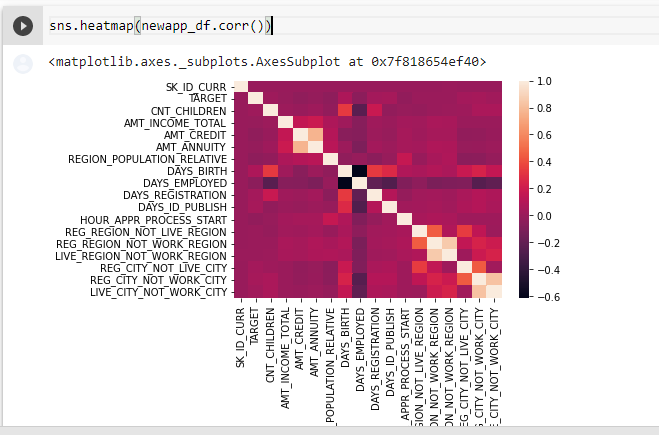


**Step 36: Visualizing the Correlation between columns/Variables using heatmap**

**Desc:** Visualization of correlation

**Code:**

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



**Step 37: Creating bin (no. of groups) and slot (range between two values) on column AMT\_INCOME \_RANGE**

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

**Code:**

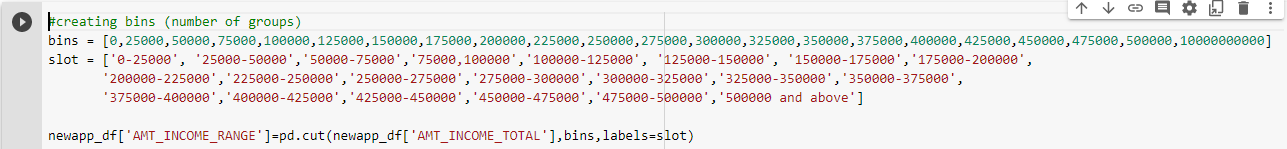
bins = [0,25000,50000,75000,100000,125000,150000,175000,200000,225000,250000,275000,300000,325000,350000,375000,400000,425000,450000,475000,500000,10000000000]

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)

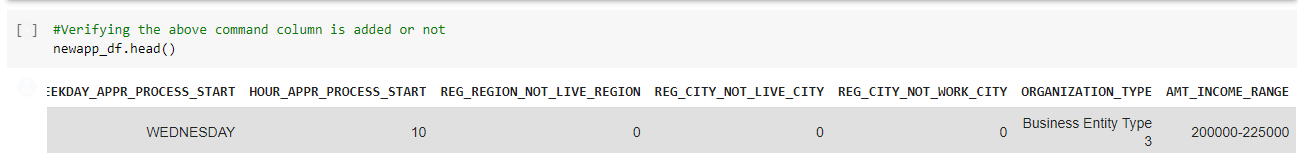
****

**Step 38: Checking new column** 'AMT\_INCOME\_RANGE' in Data Frame

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

**Code:**

newapp\_df.head()

****

**Step 39: Creating bins and slot on column** 'AMT\_CREDIT'

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

**Code:**

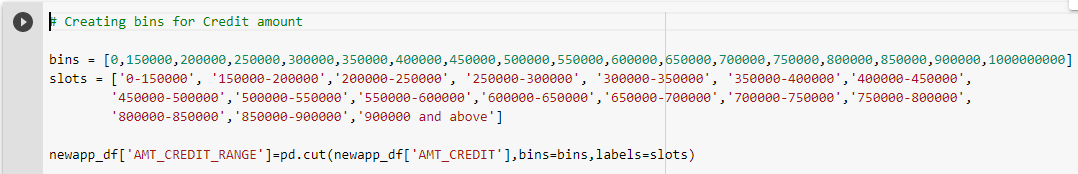
bins = [0,150000,200000,250000,300000,350000,400000,450000,500000,550000,600000,650000,700000,750000,800000,850000,900000,1000000000]

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)

****

**Step 40:** Checking new column 'AMT\_INCOME\_RANGE' in Data Frame

**Desc:** Checking whether new column added into Data Frame

**Code:**

newapp\_df.head()

****

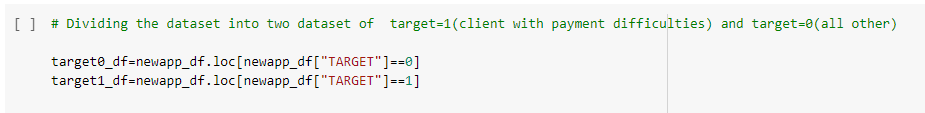
**Step 41: Dividing the Data Frame into two different Data Frame**

**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]

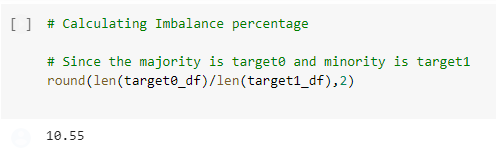
****

**Step 42: Calculating the imbalance percentage for two newly created Data Frame**

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

**Code:**

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

****

|  |
| --- |
| **Observation :**  **summarize the class distribution in percentage of the training dataset** |

**Step 43 :Univariate analysis for categories**

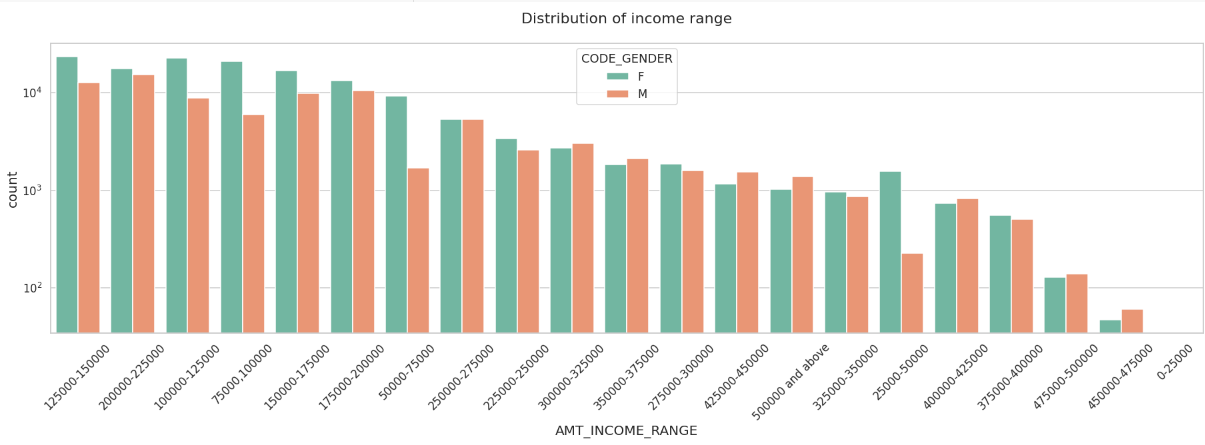
|  |
| --- |
| Univariate analysis is the simplest form of analyzing data. Uni means one, so in other words the data has only one variable. Univariate data requires to analyze each variable separately. |

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

**Step 43 -Part1: Plotting graph for AMT\_INCOME RANGE in Target 0 Data Frame**

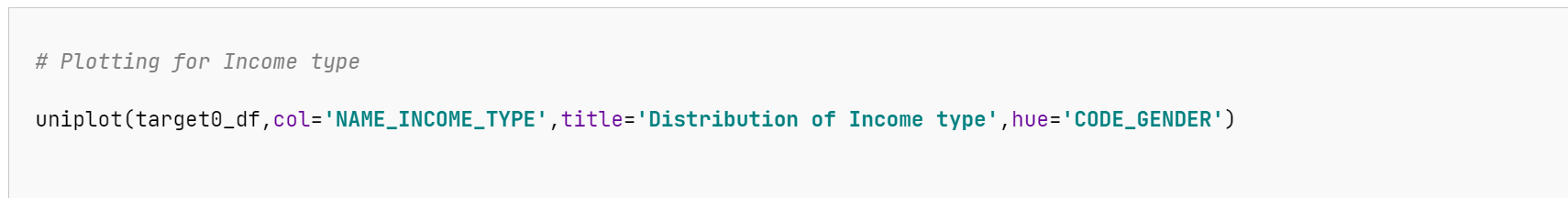
**Code:**

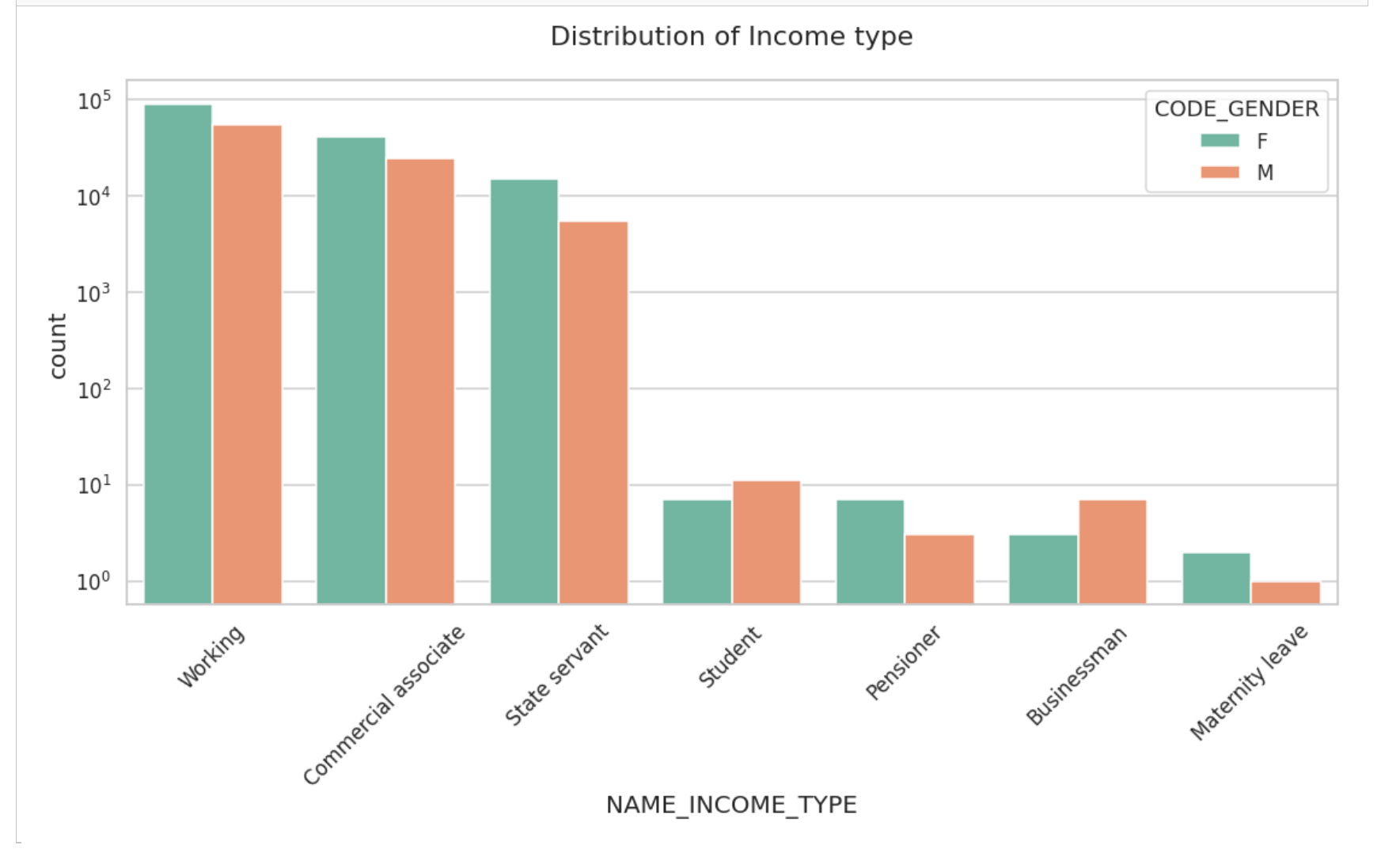
****

****

|  |
| --- |
| **Observation:**   1. Female counts are higher than male. 2. Income ranges from 100000 to 200000 is having more number of credits. 3. This graph show that females are more than male in having credits for that range. 4. Very less count for income range 400000 and above |

**Step 43-2:Plotting Graph for** 'NAME\_INCOME\_TYPE' in Target 0 DataFrame

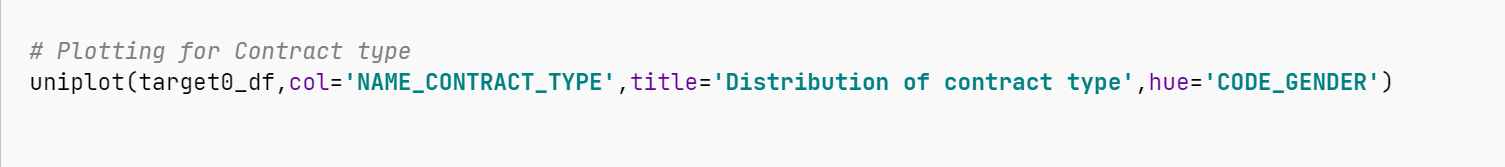
**Code:** ****

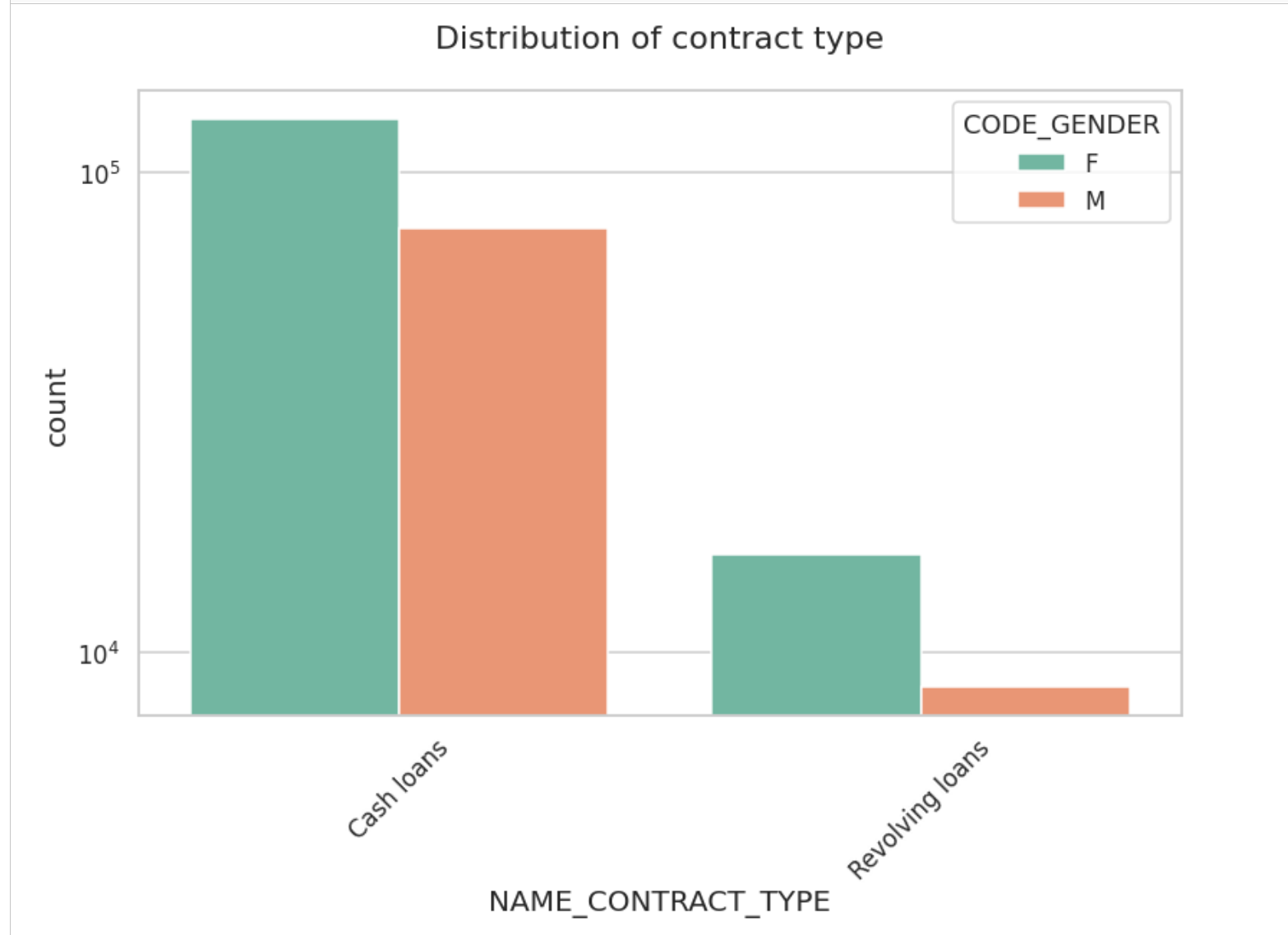
****

|  |
| --- |
| **Observation:**  For income type ‘working’, ’commercial associate’, and ‘State Servant’ the number of credits is 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’. |

Step 43- 3: Plotting Graph for Column 'NAME\_CONTRACT\_TYPE' in Target 0 Data Frame

**Code:**

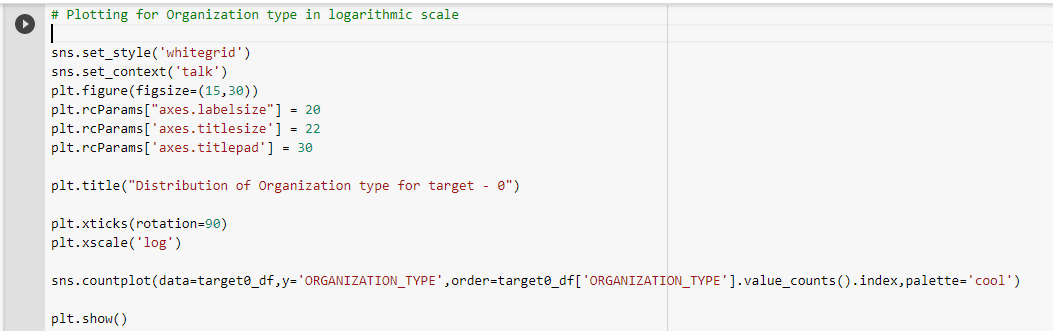
****

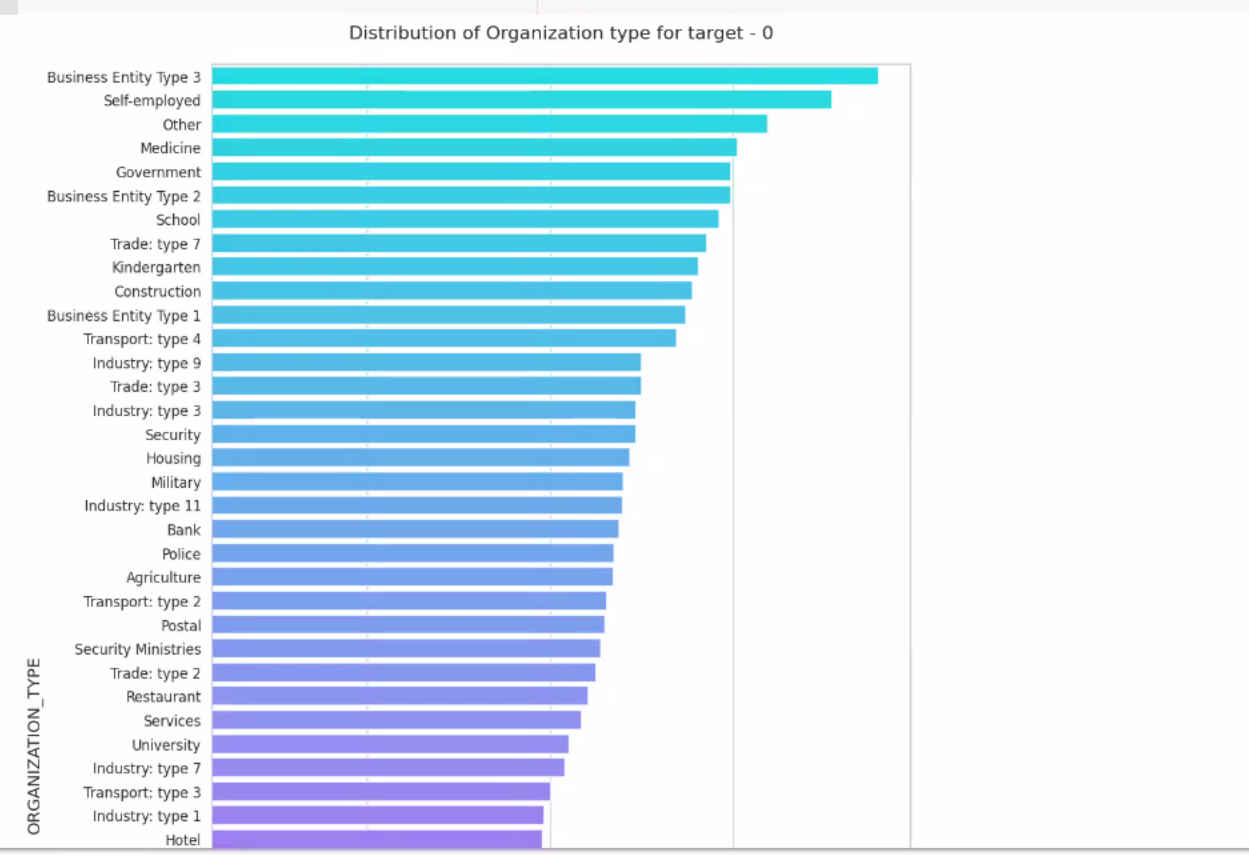


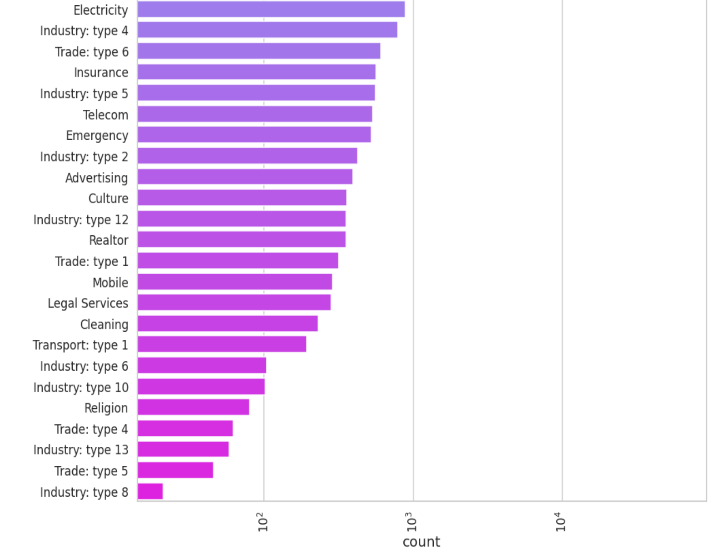
|  |
| --- |
| **Observation:**  For contract type ‘cash loans’ having higher number of credits than ‘Revolving loans’ contract type.  For this also Female is leading for applying credits. |

**Step 44:** Plotting for Organization type in logarithmic scale

**Code:**







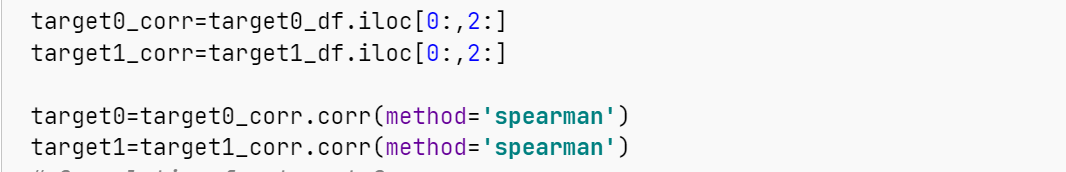
|  |
| --- |
| Observation:   1. Clients which have applied for credits are from most of the organization type ‘Business entity Type 3’, ‘Self-employed’, ‘Other’ , ‘Medicine’ and ‘Government’. 2. Less clients are from Industry type 8, type 6, type 10, religion and trade type 5, type 4. |

Step 46: Finding some correlation for numerical columns for both target 0 and 1

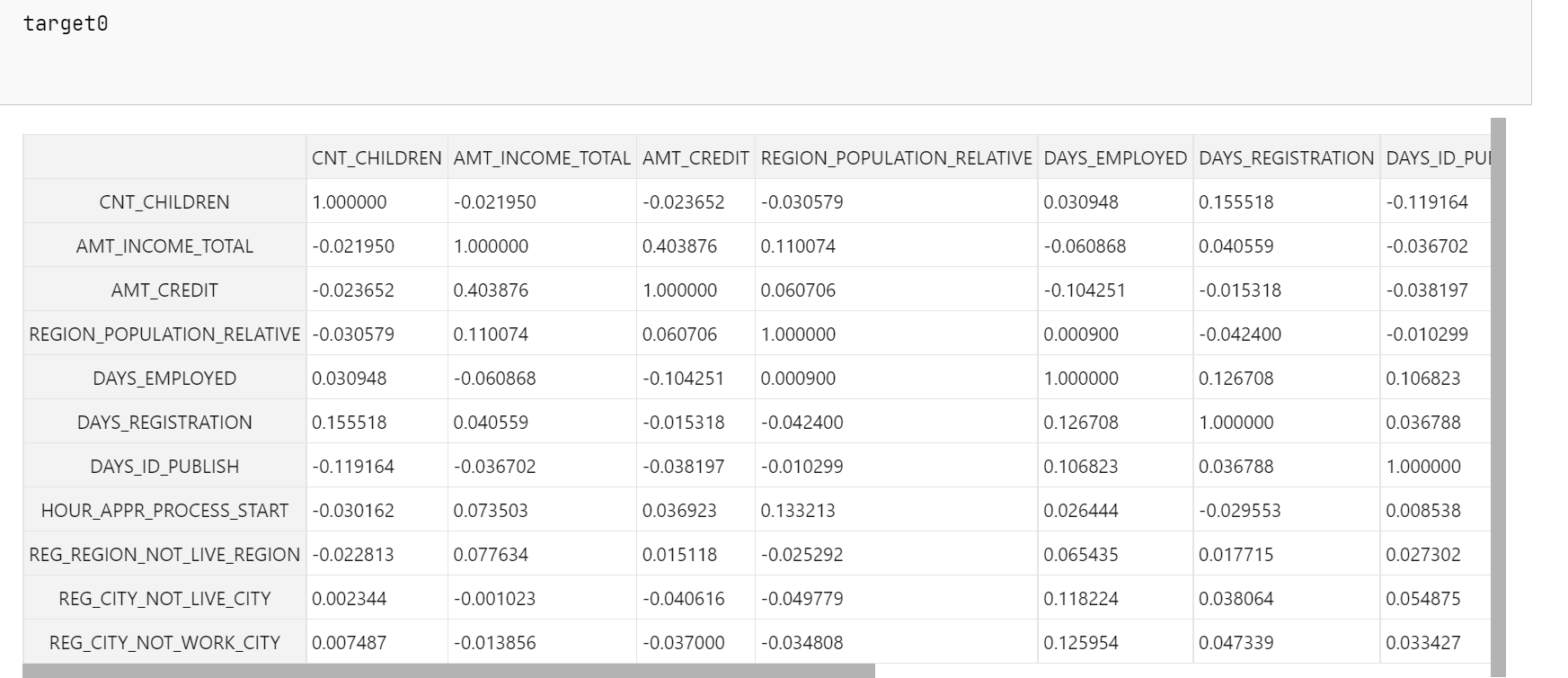
Note :

|  |
| --- |
| 1. Spearman rank correlation: Spearman rank correlation is a non-parametric test that is used **to measure the degree of association between two variables**. 2. Spearman's correlation ranges in value from **-1 to 1**, with values near 1 indicating similarity in ranks for the two variables and values near -1 indicating ranks are dissimilar for the two variables. |

Code:

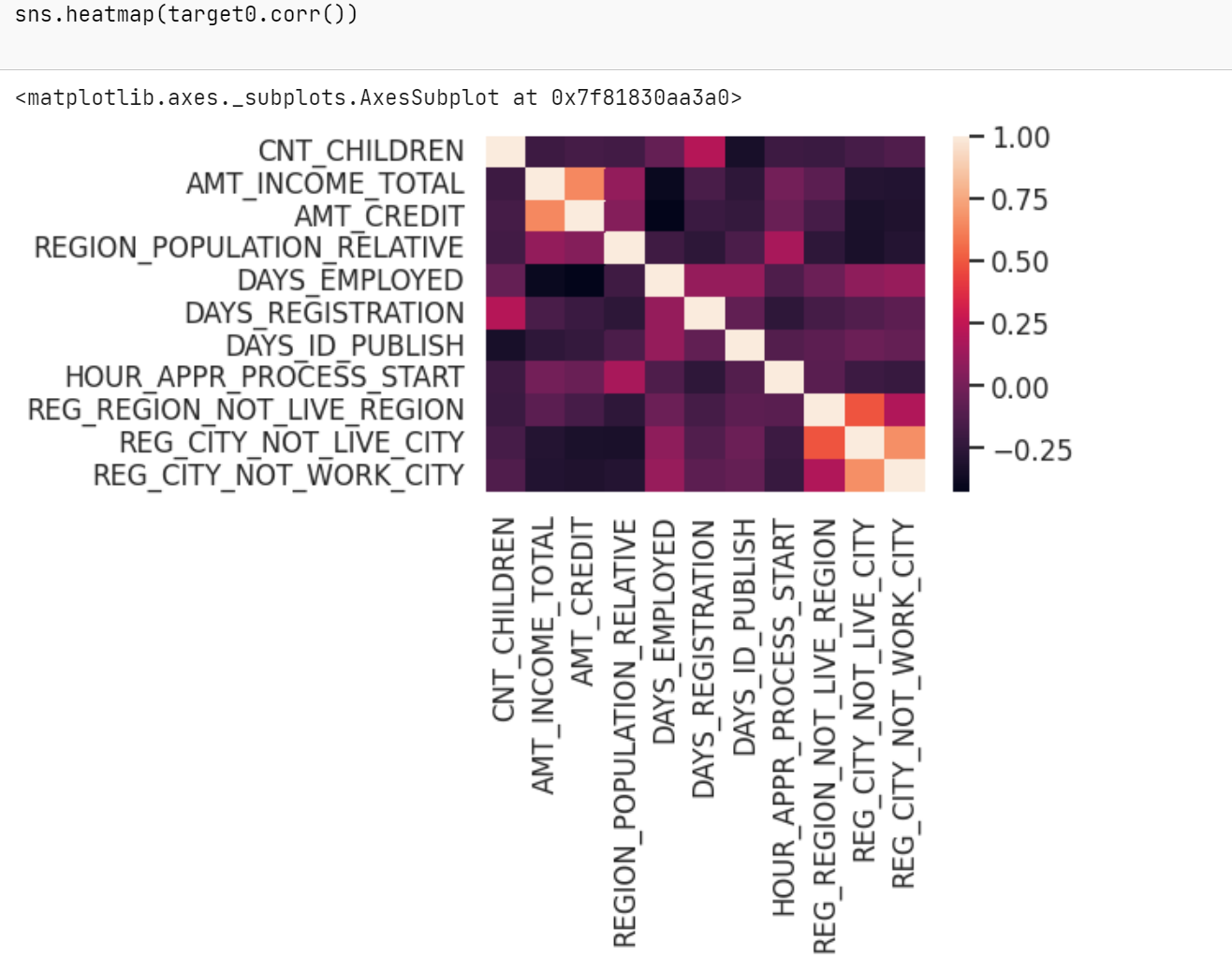


# Correlation for target 0



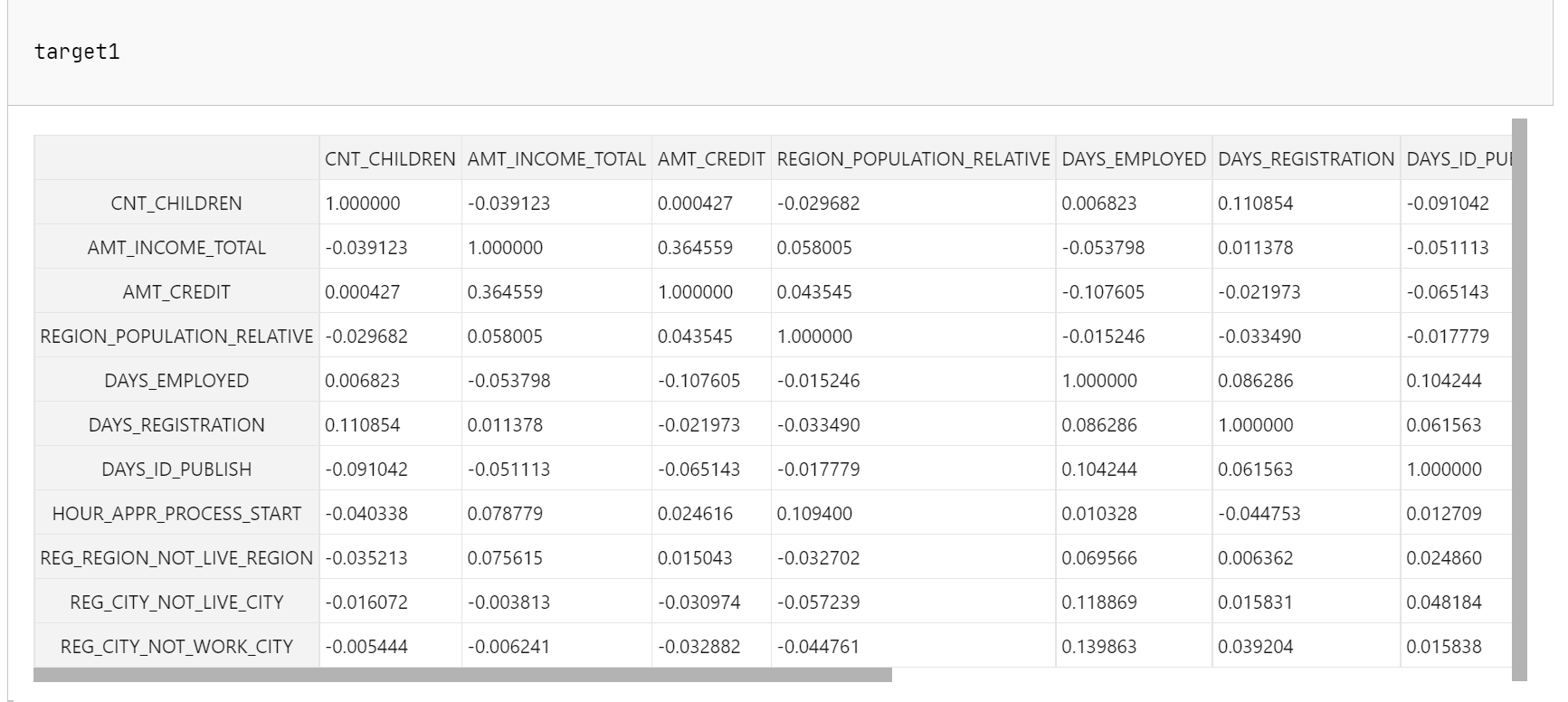
**Step 46 -1:**Visualizing the correlation for target0

**Code:**



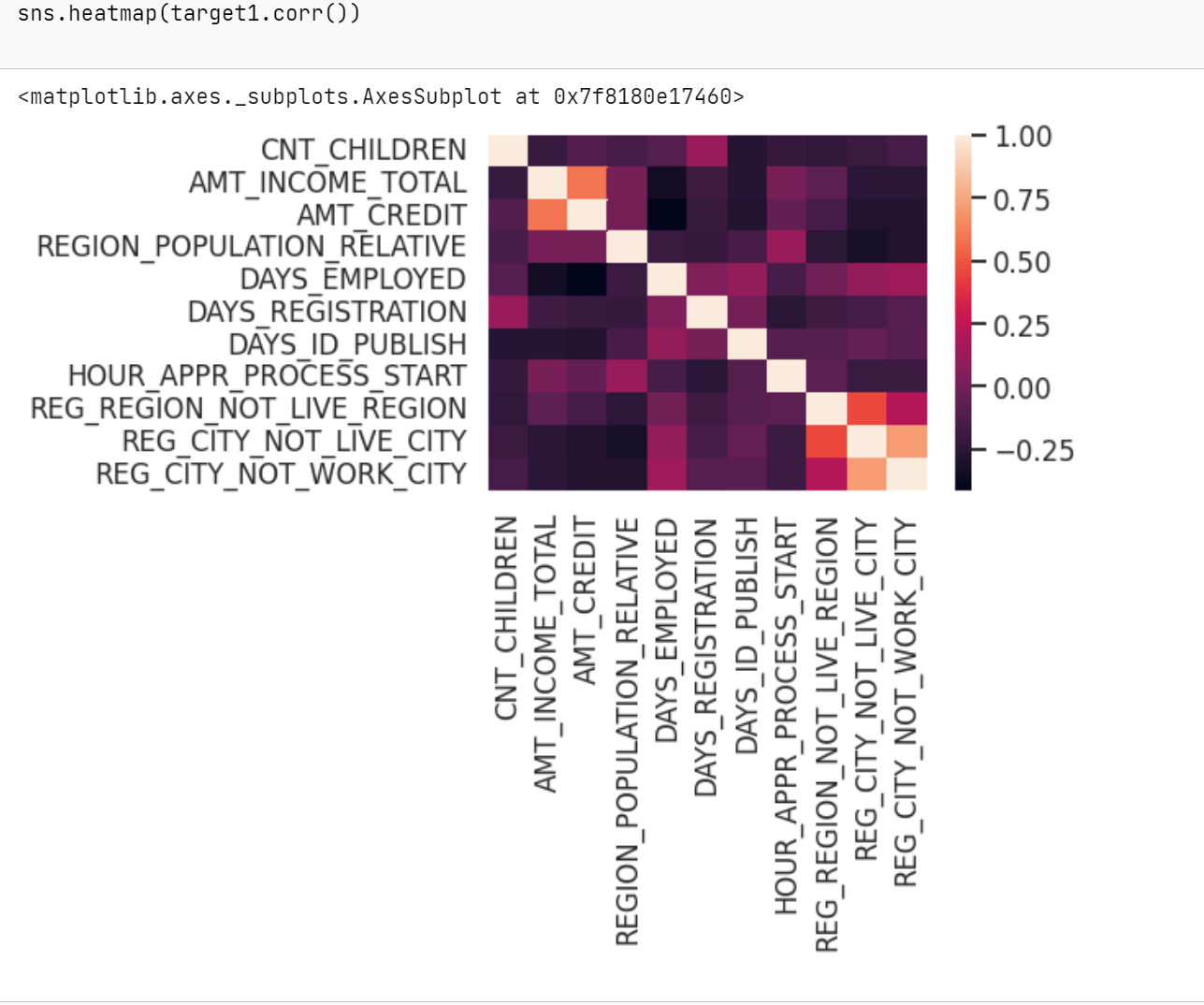
|  |
| --- |
| **Observation:**   1. Credit amount is inversely proportional to the date of birth, which means Credit amount is higher for low age and vice-versa. 2. Credit amount is inversely proportional to the number of children client have, means Credit amount is higher for less children count client have and vice-versa. 3. Income amount is inversely proportional to the number of children client have, means more income for less children client have and vice-versa. 4. less children client have in densely populated area. 5. Credit amount is higher to densely populated area. 6. The income is also higher in densely populated area. |

# Correlation for target 1



**Step 46-2:**visualizing the correlation for target1

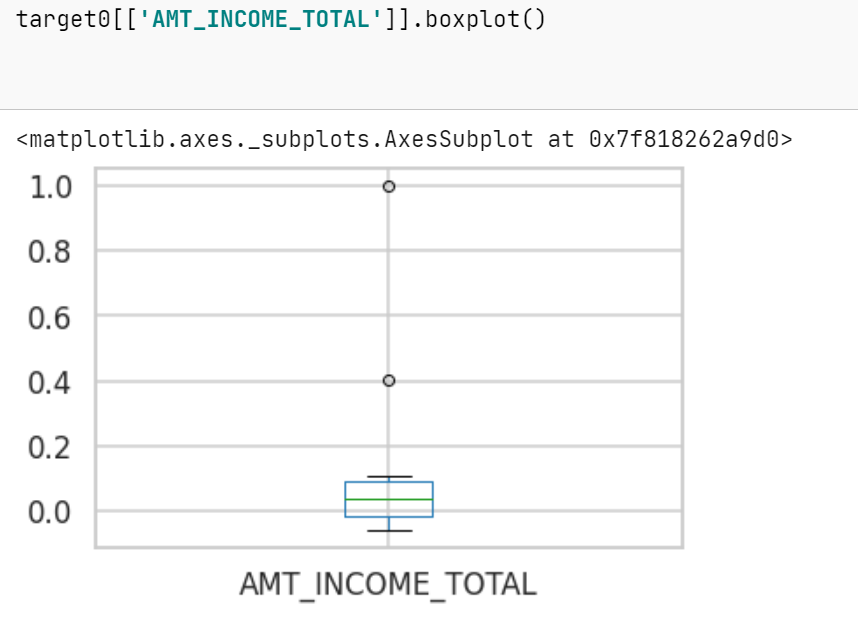
**Code:**



|  |
| --- |
| **Observation:**   1. The client's permanent address does not match contact address are having less children and vice-versa. 2. the client's permanent address does not match work address are having less children and vice-versa. |

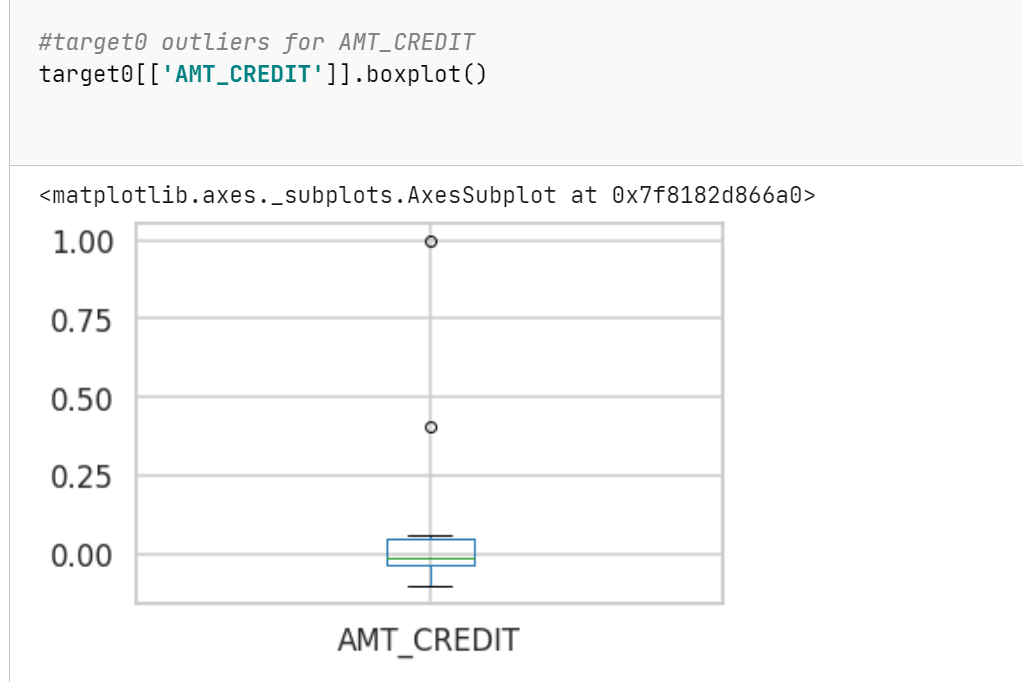
**Step 47 -1: For Target 0 - Finding any outliers**

**Desc:**target0 outliers for Column ‘AMT\_INCOME\_TOTAL’ in Data Frame Target 0



|  |
| --- |
| **Observation**:   1. Some outliers are noticed in income amount. 2. The third quartiles is very slim for income amount. |

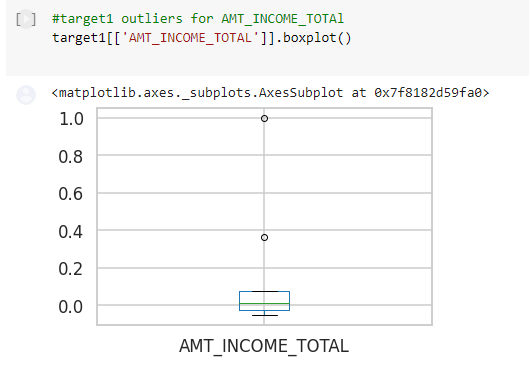
**Step 47-2:** target0 outliers for column ‘AMT\_CREDIT’ in Data Frame Target 0

****

|  |
| --- |
| **Observations:**   1. Some outliers are noticed in credit amount. 2. 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. |

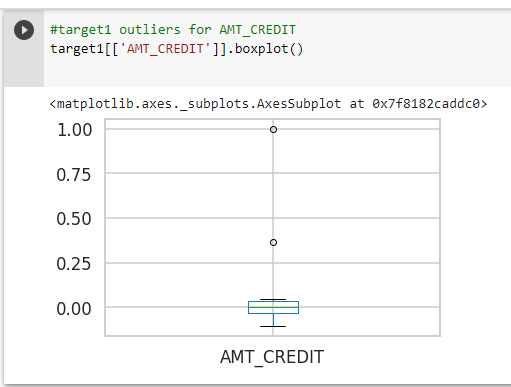
**Step 48 -1: For Target 0 - Finding any outliers**

**Desc:**target0 outliers for Column ‘AMT\_INCOME\_TOTAL’ in Data Frame Target 1



|  |
| --- |
| **Observation:**   1. Some outliers are noticed in income amount 2. The third quartiles is very slim for income amount. 3. Most of the clients of income are present in first quartile. |

**Step 48-2:** target0 outliers for column ‘AMT\_CREDIT’ in Data Frame Target 1

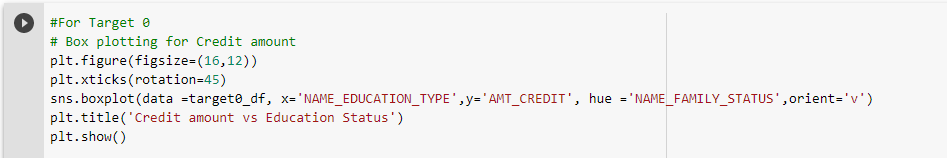


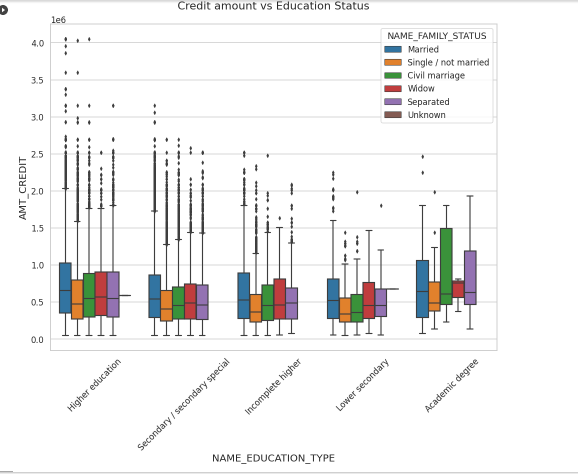
|  |
| --- |
| **Observation:**   1. Some outliers are noticed in credit amount. 2. The first quartile is bigger than third quartile for credit amount which means most of the credits of clients are present in the first quartile. |

**Bivariate Analysis** for Continues & Continues, Categorical & Categorical, Continues & Categorical

**Step 49-1:** Box plotting for **'Credit amount vs Education Status'** in target 0 Data Frame

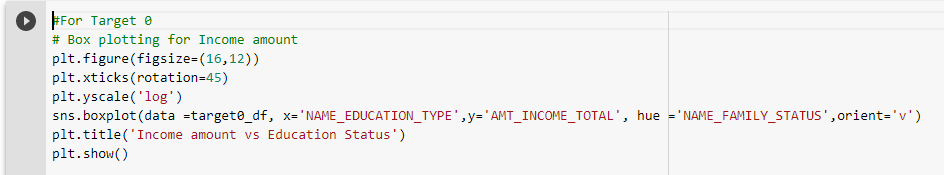
**Code:**

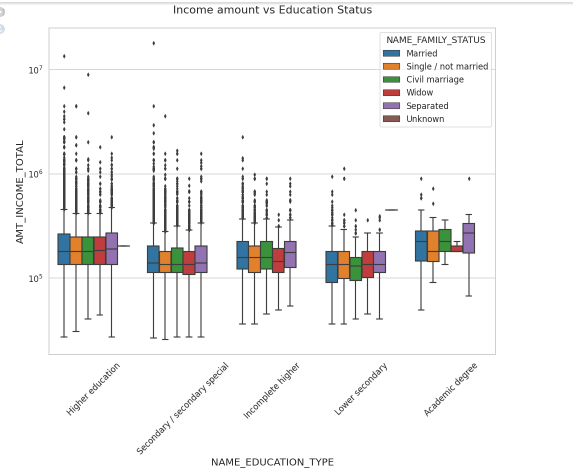




|  |
| --- |
| Observation:   1. From the above box plot we can conclude that Family status of 'civil marriage', 'marriage' and 'separated' of Academic degree education are having higher number of credits than others. 2. Also, higher education of family status of 'marriage', 'single' and 'civil marriage' are having more outliers. 3. Civil marriage for Academic degree is having most of the credits in the third quartile. |

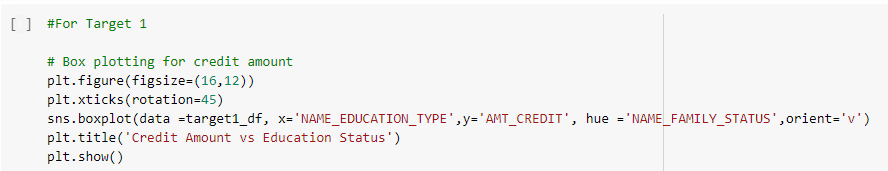
**Step 49-2:** Box plotting for 'Income amount vs Education Status' in target 0 Data Frame

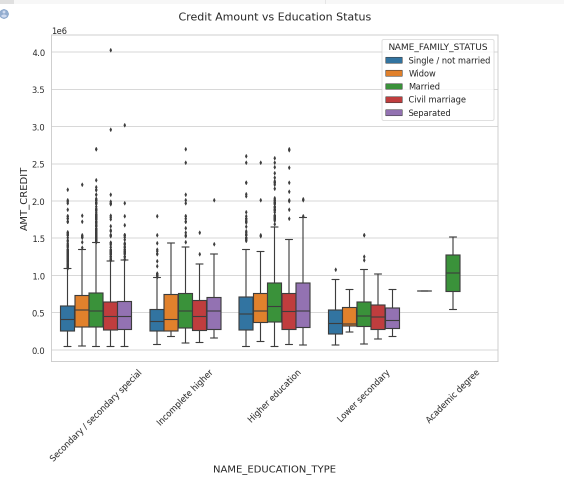




|  |
| --- |
| **Observation:**   1. From above boxplot for Education type 'Higher education' the income amount is mostly equal with family status. It does contain many outliers. 2. Less outlier are having for Academic degree but there income amount is little higher that Higher education. 3. Lower secondary of civil marriage family status are have less income amount than others. |

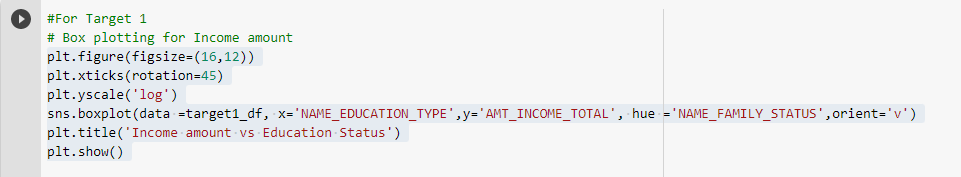
**Step 50-1:** Box plotting for **'Credit amount vs Education Status'** in target 1 Data Frame

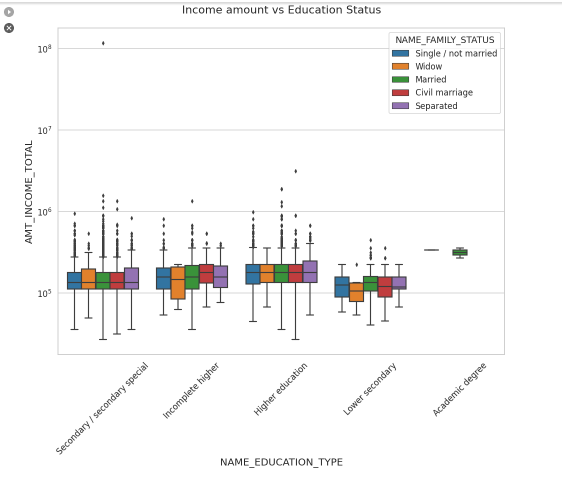




|  |
| --- |
| **Observation:**  Less outlier are having for Academic degree but there income amount is little higher that Higher education. |

**Step 50-2:** Box plotting for 'Income amount vs Education Status' in target 1 Data Frame





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

Data set 2:

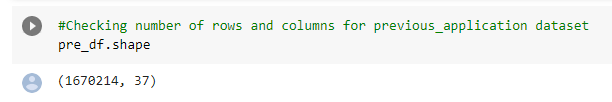
Step 1:Loading dataset

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



**Step 2:** Checking number of rows, number of columns in dataset.

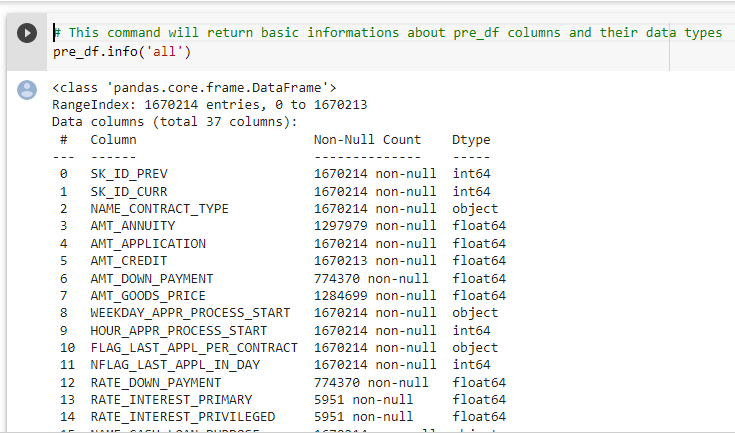
**Code:**pre\_df.shape



**Step 3:** Checking the basic info and data types of columns

**Code:**

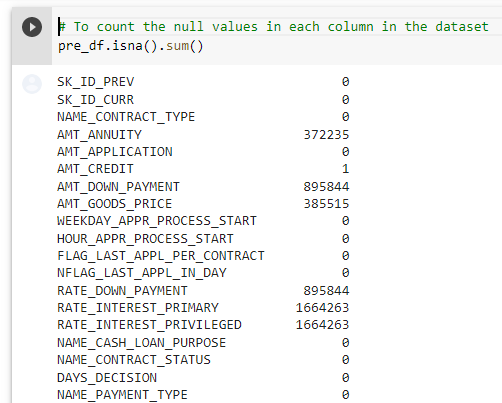
pre\_df.info('all')

****

**Step 4:**Counting null values column wise.

**Code:**

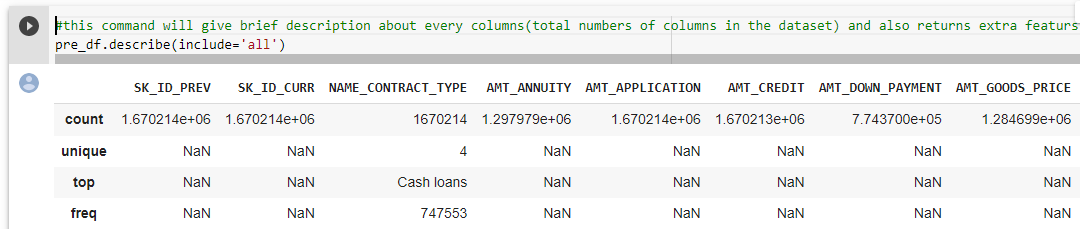
pre\_df.isna().sum()

****

**Step 5:** Observing brief description info of dataset

**Code:**

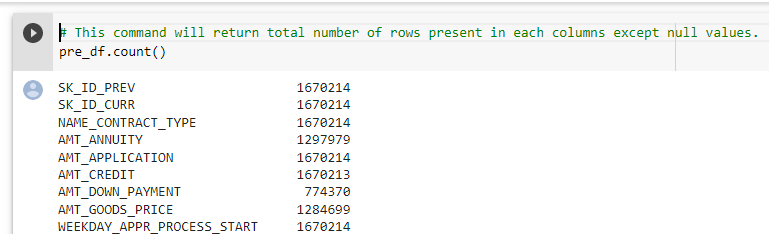
pre\_df.describe(include='all')

****

**Step 6:** Returns no of rows in each column

**Code:**

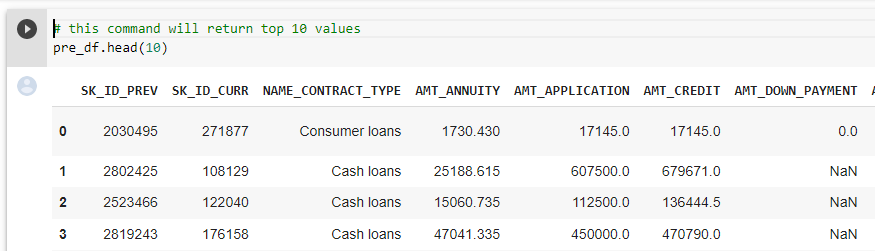
pre\_df.count()

****

**Step 7:** Return first 10 records in dataset.

**Code:**

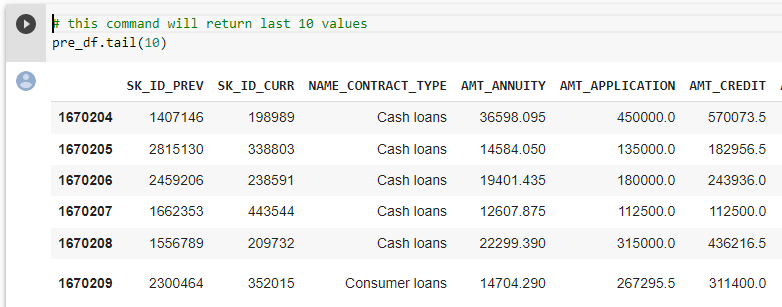
pre\_df.head(10)

****

**Step 8:** Return bottom 10 records in dataset.

**Code:**

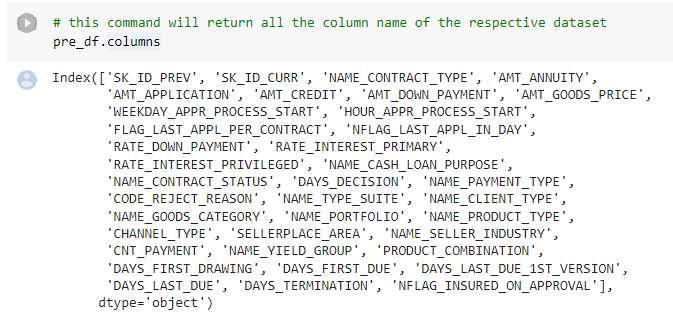
pre\_df.tail(10)

****

**Step 9:** Returns columns of dataset.

**Code:**

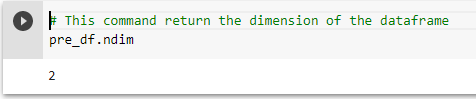
pre\_df.columns

****

**Step 10:** Return the dimension of dataset.

**Code:**

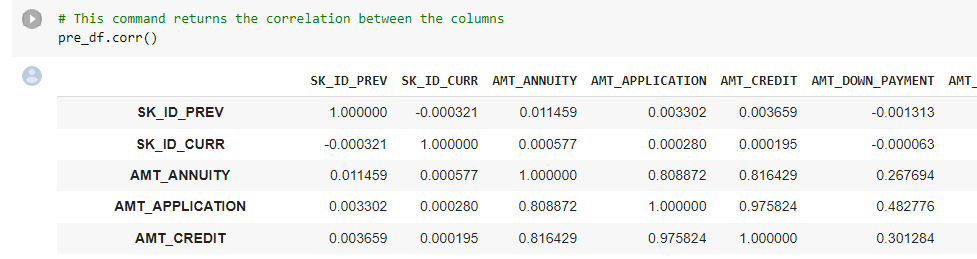
pre\_df.ndim

****

**Step 11:** Displays correlation between columns in dataset.

**Code:**

pre\_df.corr()

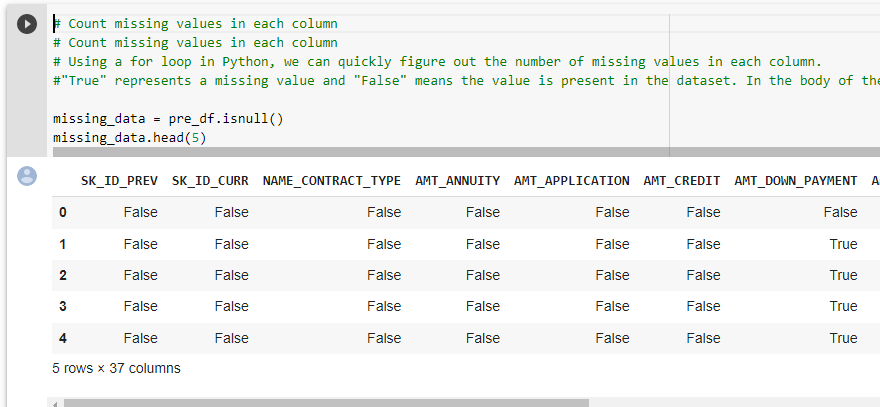
****

**Step 12:** Displays whether data present in column or not by True and False.

**Code:**

missing\_data = pre\_df.isnull()

missing\_data.head(5)

****

**Step 13:** Finding null values using for loop in python.

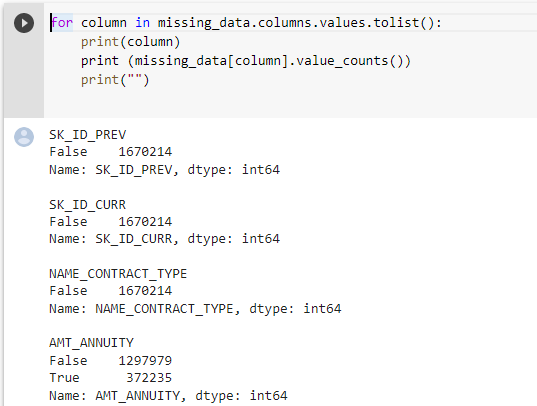
**Code:**

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

print(column)

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

print("")

****

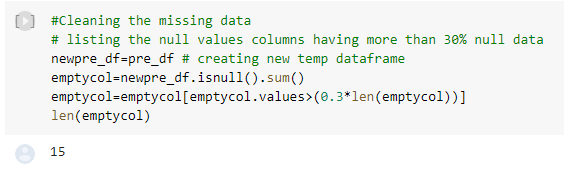
**Step 14:** Checking the null value in each column of a Data Frame who having more than 30 percent of total null data

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

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

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

len(emptycol)

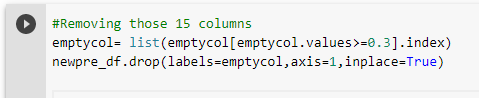
****

**Steps 15:** Dropping the null value in each column of a Data Frame who having more than 30 percent of total null data

**Code:**

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

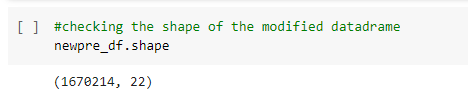
newpre\_df.drop(labels=emptycol,axis=1,inplace=True)

****

**Steps 16:** Return shape of data frame after dropping

**Code:**

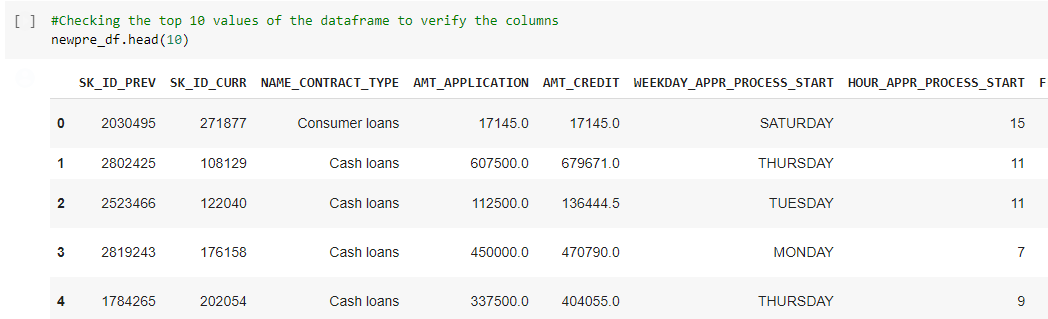
newpre\_df.shape

****

**Steps 17:** Return top 10 records

**Code:**

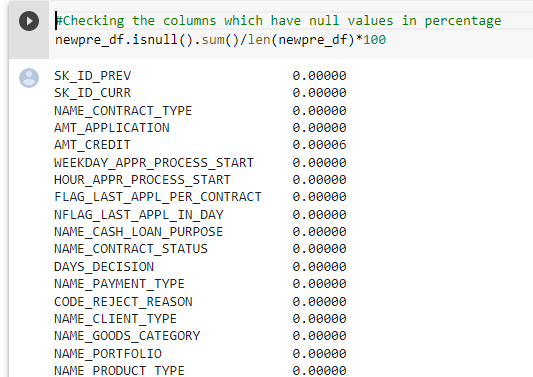
newpre\_df.head(10)

****

**Steps 18:** Columns having null values in percentage

**Code:**

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

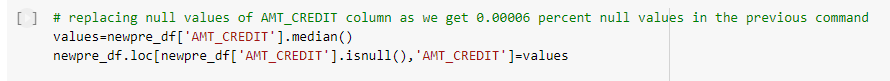
****

**Steps 19:** Replacing null values of AMT\_CREDIT

**Code:**

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

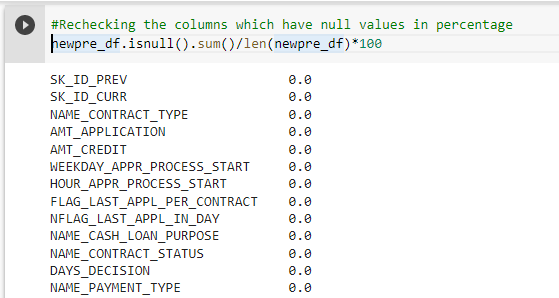
newpre\_df.loc[newpre\_df['AMT\_CREDIT'].isnull(),'AMT\_CREDIT']=values

****

**Steps 20:** Rechecking the columns which have null values in percentage

**Code:**

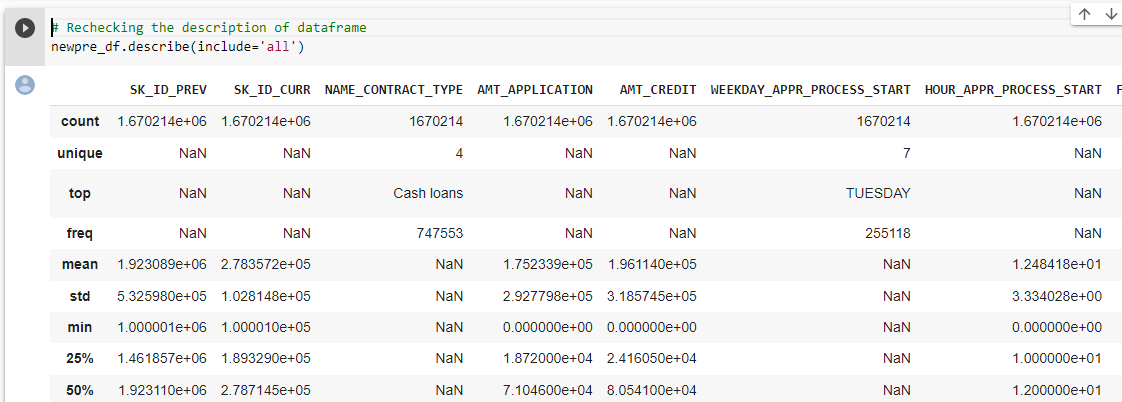
newpre\_df.isnull().sum()/len(newpre\_df)\*100

****

**Steps 21:** Description of data frame

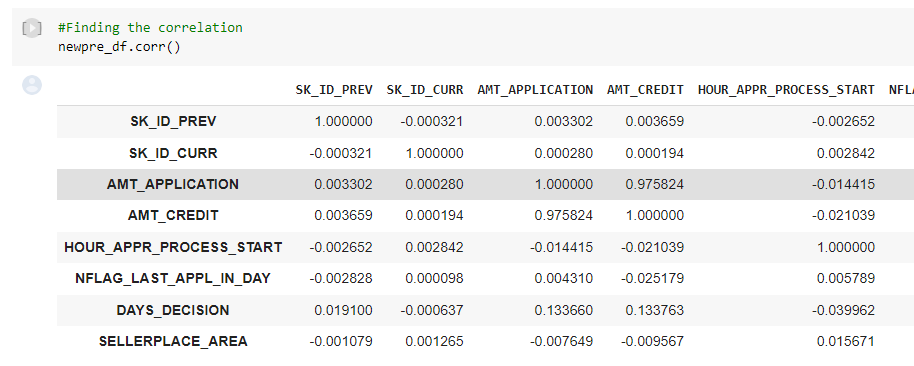
**Code:**

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

****

**Steps 22:** Finding the correlation between columns.

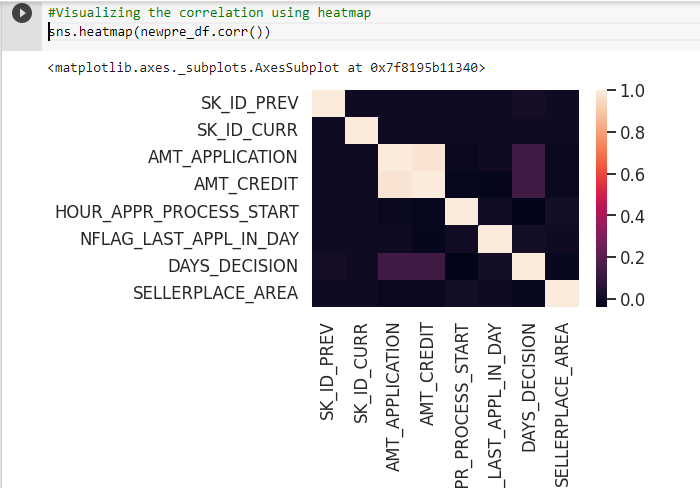
**Code:**newpre\_df.corr()

****

**Steps 23:** Visualizing correlation using heatmap.

**Code:**

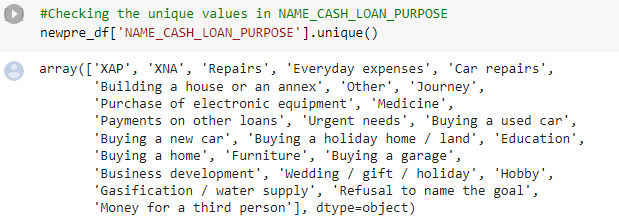
sns.heatmap(newpre\_df.corr())

****

**Steps 24:** Finding the uniques values in NAME\_CASH\_LOAN\_PURPOSE column

**Code:**

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

****

**Steps 25:** Dropping the XNA and XPA values from NAME\_CASH\_LOAN\_PURPOSE

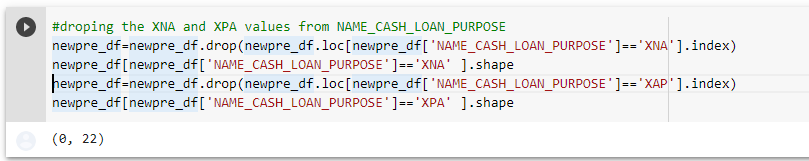
**Code:**

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)

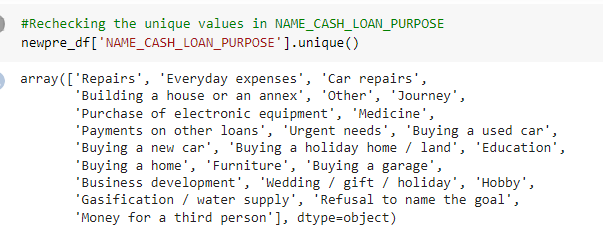
newpre\_df[newpre\_df['NAME\_CASH\_LOAN\_PURPOSE']=='XPA' ].shape

****

**Steps 26:** Rechecking the unique values in NAME\_CASH\_LOAN\_PURPOSE

**Code:**

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

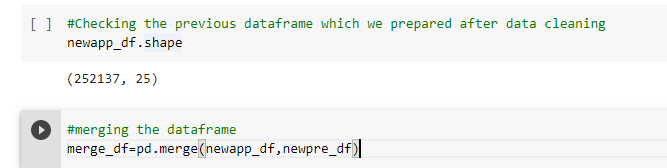
****

**Steps 27:**Checking shape of dataset and merging two dataframes.

**Code:**

newapp\_df.shape

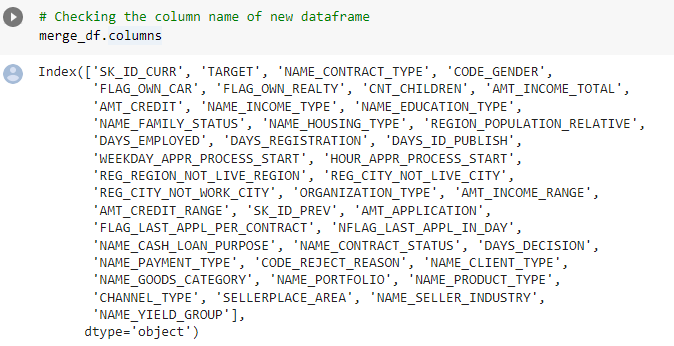
merge\_df=pd.merge(newapp\_df,newpre\_df)

****

**Steps 28:** Checking the columns of new dataframe.

**Code:**

merge\_df.columns

****

**Steps 29:** Checking the columns of new data frame.

**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',

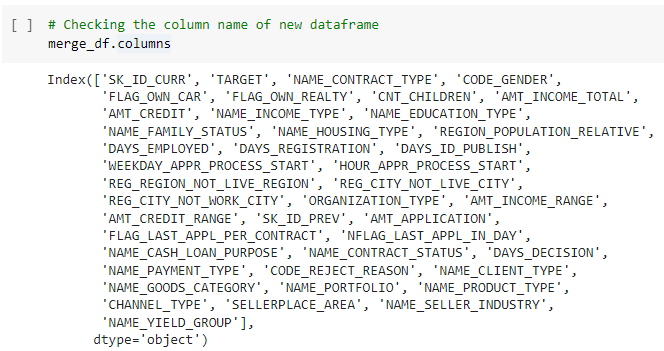
'HOUR\_APPR\_PROCESS\_START\_':'HOUR\_APPR\_PROCESS\_START','NAME\_CONTRACT\_TYPEx':'NAME\_CONTRACT\_TYPE\_PREV', 'AMT\_CREDITx':'AMT\_CREDIT\_PREV','AMT\_ANNUITYx':'AMT\_ANNUITY\_PREV', 'WEEKDAY\_APPR\_PROCESS\_STARTx':'WEEKDAY\_APPR\_PROCESS\_START\_PREV',

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

****

**Steps 30:** Checking the column of new data frame.

merge\_df.columns

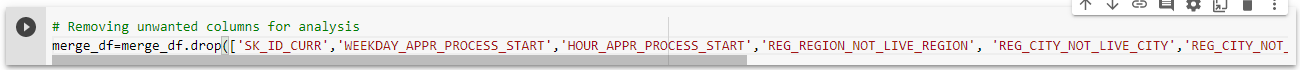
****

**Steps 31:** Checking the columns of new data frame.

**:**Checking columns after dropping unnecessary columns

**Code:**

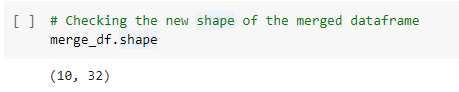
merge\_df=merge\_df.drop(['SK\_ID\_CURR','WEEKDAY\_APPR\_PROCESS\_START','HOUR\_APPR\_PROCESS\_START','REG\_REGION\_NOT\_LIVE\_REGION', 'REG\_CITY\_NOT\_LIVE\_CITY','REG\_CITY\_NOT\_WORK\_CITY', 'ORGANIZATION\_TYPE', 'AMT\_INCOME\_RANGE','FLAG\_LAST\_APPL\_PER\_CONTRACT', 'NFLAG\_LAST\_APPL\_IN\_DAY'],axis=1)

****

**Steps 32:**  Checking the shape of data frame.

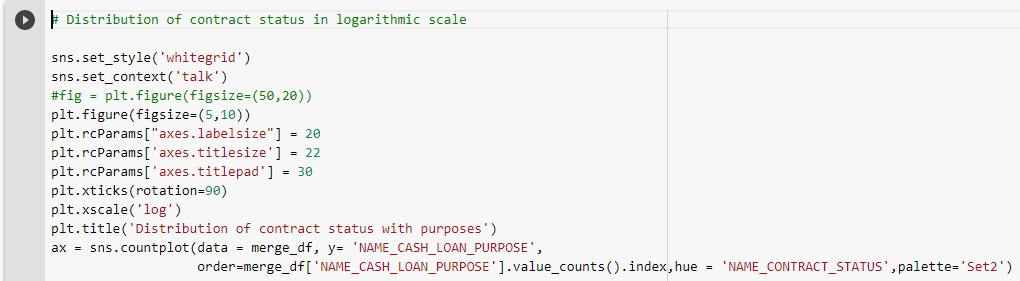
**Code:**

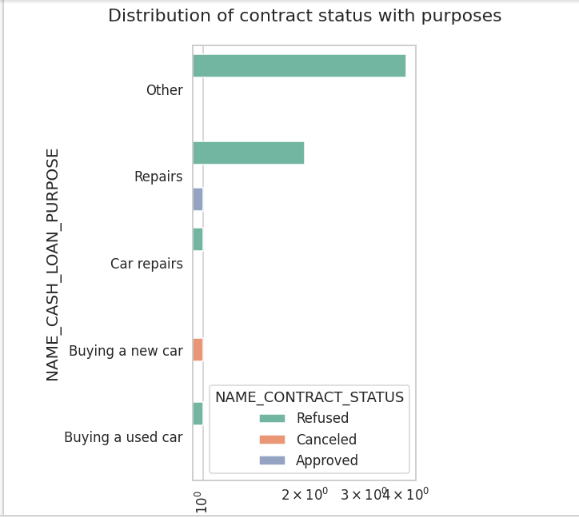
merge\_df.shape

****

Univariate Analysis on merged Data frame

**Step 33:**Distribution of Column ‘’NAME\_CASH\_LOAN\_PURPOSE’’ status in logarithmic scale

****

****

|  |
| --- |
| Observations:   1. Most rejection of loans came from purpose 'repairs'. 2. For education purposes we have equal number of approves and rejection 3. other loans and buying a new car is having significant higher rejection than approves. |

Step 34: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)

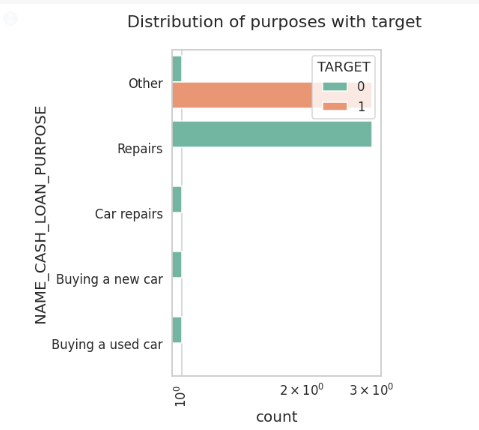
plt.xscale('log')

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:**   1. Loan purposes with 'Repairs' are facing more difficulties in payment on time. 2. 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 for minimal payment difficulties. |

**Bivariate Analysis**

**Step 35:** Box plotting for Credit amount in logarithmic scale

**Code:**

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

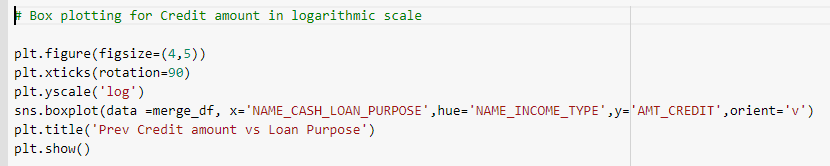
plt.xticks(rotation=90)

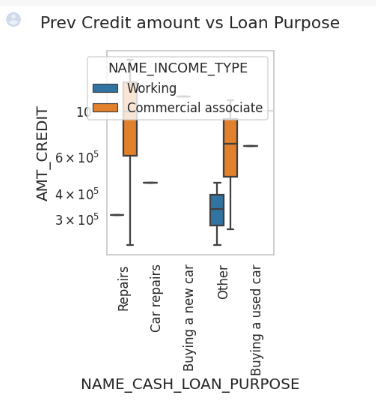
plt.yscale('log')

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:**   1. The credit amount of Loan purposes like 'Buying a home’, ‘Buying a land’,’ Buying a new car' and ‘Building a house' is higher. 2. Income type of state servants have a significant amount of credit applied 3. Money for third person or a Hobby is having less credits applied for. |

**Steps 36:**

Box plotting for Credit amount vs Housing type in logarithmic scale

**Code:**

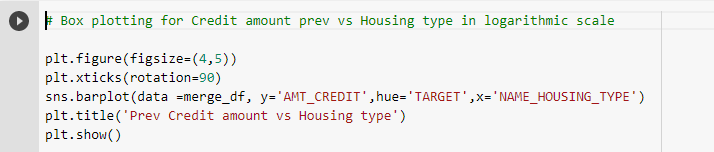
plt.figure(figsize=(4,5))

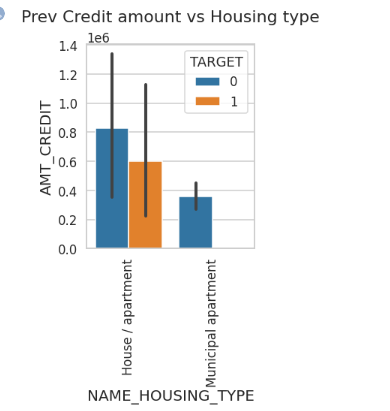
plt.xticks(rotation=90)

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:**   1. 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. 2. 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. |

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