DATA CLEANER AND ANALYSER

USING PYTHON..

PROJECT REPORT

BY

NEELESHWAR – B522015

ABSTRACT :

Data cleaning, or data cleansing, is the process of identifying and correcting errors, inconsistencies, and inaccuracies in datasets. It ensures the data you work with is accurate, reliable, and ready for analysis, visualization, or further processing. It is a critical step in the data preparation pipeline.This is a self made python

Project aimed at automating data cleansing and analysis.

1.Python

Python is a high-level, interpreted, interactive and object-oriented scripting language. Python is designed to be highly readable. It uses English keywords frequently where as other languages use punctuation, and it has fewer syntactical constructions than other languages.

• Python is Interpreted − Python is processed at runtime by the interpreter. You do not need to compile your program before executing it. This is similar to PERL and PHP.

• Python is Interactive − You can actually sit at a Python prompt and interact with the interpreter directly to write your programs.

• Python is Object-Oriented − Python supports Object-Oriented style or technique of programming that encapsulates code within objects.

• Python is a Beginner's Language − Python is a great language for the beginner-level programmers and supports the development of a wide range of applications from simple text processing to WWW browsers to games. History of Python Python was developed by Guido van Rossum in the late eighties and early nineties at the National Research Institute for Mathematics and Computer Science in the Netherlands. Python is derived from many other languages, including ABC, Modula-3, C, C++, Algol-68, SmallTalk, and Unix shell and other scripting languages. Python is copyrighted. Like Perl, Python source code is now available under the GNU General Public License (GPL). Python is now maintained by a core development team at the institute, although Guido van Rossum still holds a vital role in directing its progress. Python Features Python's features include –

• Easy-to-learn − Python has few keywords, simple structure, and a clearly defined syntax. This allows the student to pick up the language quickly.

6 • Easy-to-read − Python code is more clearly defined and visible to the eyes. • Easy-to-maintain − Python's source code is fairly easy-tomaintain.

• A broad standard library − Python's bulk of the library is very portable and cross-platform compatible on UNIX, Windows, and Macintosh.

• Interactive Mode − Python has support for an interactive mode which allows interactive testing and debugging of snippets of code.

• Portable − Python can run on a wide variety of hardware platforms and has the same interface on all platforms.

• Extendable − You can add low-level modules to the Python interpreter. These modules enable programmers to add to or customize their tools to be more efficient.

• Databases − Python provides interfaces to all major commercial databases. • GUI Programming − Python supports GUI applications that can be created and ported to many system calls, libraries and windows systems, such as Windows MFC, Macintosh, and the X Window system of Unix. • Scalable − Python provides a better structure and support for large programs than shell scripting. Apart from the above-mentioned features, Python has a big list of good features, few are listed below − • It supports functional and structured programming methods as well as OOP.

• It can be used as a scripting language or can be compiled to bytecode for building large applications.

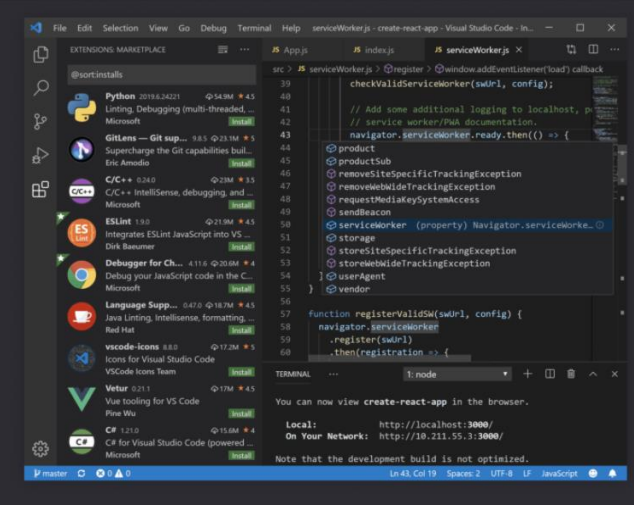
• It provides very high-level dynamic data types and supports dynamic type checking. • It supports automatic garbage collection. • It can be easily integrated with C, C++, COM, ActiveX, CORBA, and

Java.

3 Visual studio code

Visual Studio Code is a free source code editor, made by Microsoft for Windows, Linux and macOS.

Features include support for debugging syntax highlighting, intelligent code completion, snippets, code refactoring, and embedded Git. Users can change the theme, keyboard shortcuts, preferences, and install extensions that add additional functionality. The python extension in Visual Studio Code makes it an excellent video editor.



4.Source Code

Importing library

import warnings

warnings.filterwarnings("ignore")

#import liberaries

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

import seaborn as sns

#%matplotlib inline

pd.set\_option("display.max\_columns", None)

# Read Application

app\_data=pd.read\_csv("application\_data.csv")

app\_data.head()

# Data insepection on Apllication DataSet

app\_data.info()

# Data Quality check

### check for the percentage null values in Application Dataset

pd.set\_option("display.max\_rows",200)

app\_data.isnull().mean() \* 100

#conclusion: columns with null value more than 47% may give wrong insight hence will drop them

# Dropping columns with missing values greater than 47%

percentage = 47

threshold = int(((100-percentage)/100)\*app\_data.shape[0] + 1)

#app\_df = app\_data.dropna(axis=1, )

app\_df = app\_data.dropna(axis=1,thresh=threshold)

app\_df.head()

app\_df.shape

app\_df.isnull().mean() \* 100

# Imputing Missing value

# check the missing value in application datasetb before imputing

app\_df.info()

# OCCUPATION\_TYPE column has 31% missing values, since its a categorical column, imputing the missing values with a unknown or others values

app\_df.OCCUPATION\_TYPE.isnull().mean()\*100

app\_df.OCCUPATION\_TYPE.value\_counts(normalize=True)\*100

app\_df.OCCUPATION\_TYPE.fillna("Others",inplace=True)

app\_df.OCCUPATION\_TYPE.isnull().mean()\*100

app\_df.OCCUPATION\_TYPE.value\_counts(normalize=True)\*100

app\_df.EXT\_SOURCE\_3.isnull().mean()\*100

app\_df.EXT\_SOURCE\_3.value\_counts(normalize=True)\*100

app\_df.EXT\_SOURCE\_3.describe()

sns.boxplot(x=app\_df['EXT\_SOURCE\_3'])

plt.show()app\_df['EXT\_SOURCE\_3'].fillna(app\_df['EXT\_SOURCE\_3'].median(), inplace=True)

app\_df.EXT\_SOURCE\_3.isnull().mean()\*100

app\_df.EXT\_SOURCE\_3.value\_counts(normalize=True)\*100

null\_cols=list(app\_df.columns[app\_df.isna().any()])

len(null\_cols)

app\_df.isnull().mean()\*100

# - Handling Missing values in Columns with 13% null values

app\_df.AMT\_REQ\_CREDIT\_BUREAU\_HOUR.value\_counts(normalize=True)\*100

app\_df.AMT\_REQ\_CREDIT\_BUREAU\_DAY.value\_counts(normalize=True)\*100

cols = [ "AMT\_REQ\_CREDIT\_BUREAU\_HOUR",

"AMT\_REQ\_CREDIT\_BUREAU\_DAY",

"AMT\_REQ\_CREDIT\_BUREAU\_WEEK",

"AMT\_REQ\_CREDIT\_BUREAU\_MON", "AMT\_REQ\_CREDIT\_BUREAU\_QRT",

"AMT\_REQ\_CREDIT\_BUREAU\_YEAR"]

for col in cols:

app\_df[col].fillna(app\_df[col].mode()[0],inplace=True)

app\_df.isnull().mean()\*100

# Handling Missing values less than 1%

null\_cols=list(app\_df.columns[app\_df.isna().any()])

len(null\_cols)

app\_df.OBS\_30\_CNT\_SOCIAL\_CIRCLE.value\_counts(normalize=True)\*100

app\_df.EXT\_SOURCE\_2.value\_counts(normalize=True)\*100

app\_df.OBS\_30\_CNT\_SOCIAL\_CIRCLE.value\_counts(normalize=True)\*100

# -Conclusion:

# -for categorical Columns, impute the missing values with mode

# -for numerical columns, imputing missing vlaues with median

app\_df.NAME\_TYPE\_SUITE.fillna(app\_df.NAME\_TYPE\_SUITE .mode()[0], inplace=True)

app\_df.CNT\_FAM\_MEMBERS.fillna(app\_df.CNT\_FAM\_MEMBERS .mode()[0], inplace=True)

#Imputing Numerical Columns

app\_df.EXT\_SOURCE\_2.fillna(app\_df.EXT\_SOURCE\_2 .median(), inplace=True)

app\_df.AMT\_GOODS\_PRICE.fillna(app\_df.AMT\_GOODS\_PRICE .median(), inplace=True)

app\_df.AMT\_ANNUITY.fillna(app\_df.AMT\_ANNUITY .median(), inplace=True)

app\_df.OBS\_60\_CNT\_SOCIAL\_CIRCLE.fillna(app\_df.OBS\_60\_CNT\_SOCIAL\_CIRCLE .median(), inplace=True)

app\_df.DAYS\_LAST\_PHONE\_CHANGE.fillna(app\_df.DAYS\_LAST\_PHONE\_CHANGE .median(), inplace=True)

null\_cols = list(app\_df.columns[app\_df.isna().any()])

len(null\_cols)

app\_df.isnull().mean()\*100

# Convert Negative values to positive in days variable so that median is nit affected

app\_df.DAYS\_BIRTH=app\_df.DAYS\_BIRTH.apply(lambda x: abs(x))

app\_df.DAYS\_EMPLOYED=app\_df.DAYS\_EMPLOYED.apply(lambda x: abs(x))

app\_df.DAYS\_ID\_PUBLISH=app\_df.DAYS\_ID\_PUBLISH.apply(lambda x: abs(x))

app\_df.DAYS\_LAST\_PHONE\_CHANGE=app\_df.DAYS\_LAST\_PHONE\_CHANGE.apply(lambda x: abs(x))

app\_df.DAYS\_REGISTRATION=app\_df.DAYS\_REGISTRATION.apply(lambda x: abs(x))

# Diving Application Dataset with Target Variable as 0 and 1

tar\_0 = app\_df[app\_df.TARGET ==0]

tar\_1 = app\_df[app\_df.TARGET ==1]

app\_df.TARGET.value\_counts(normalize =True)\*100

# Univariate Analysis

cat\_cols=list(app\_df.columns[app\_df.dtypes == object])

num\_cols=list(app\_df.columns[app\_df.dtypes == np.int64])+ list(app\_df.columns[app\_df.dtypes == np.float64])

cat\_cols

num\_cols

for col in cat\_cols:

print(app\_df[col].value\_counts(normalize = True))

plt.figure(figsize=[5,5])

app\_df[col].value\_counts(normalize = True).plot.pie(labeldistance = None, autopct='%1.2f%%')

plt.legend()

## Plot on Numberical Columns

### Univariate Analysis on Columns with Target 0 to 1

for col in cat\_cols:

print(f"plot on {col} for Target o to 1")

plt.figure(figsize=[10,7])

plt.subplot(1,2,1)

tar\_0[col].value\_counts(normalize=True).plot.bar()

plt.title("Target 0")

plt.xlabel(col)

plt.ylabel("Density")

plt.subplot(1,2,2)

tar\_1[col].value\_counts(normalize = True).plot.bar()

plt.title("Target 1")

plt.xlabel(col)

plt.ylabel("Density")

plt.show()

print("\n-------------------------------------------------------------------------------------------------------\n")

### Pair Plot of Amount Columns for Targt 1

sns.pairplot(tar\_1[["AMT\_INCOME\_TOTAL","AMT\_CREDIT","AMT\_ANNUITY","AMT\_GOODS\_PRICE"]])

plt.show()

### Co-relation between Numerical Columns

corr\_data=app\_df[["AMT\_INCOME\_TOTAL","AMT\_CREDIT","AMT\_ANNUITY","AMT\_GOODS\_PRICE",

"YEARS\_BIRTH","YEARS\_ELPLOYED","YEARS\_REGISTRATION","YEARS\_ID\_PUBLISH","YEARS\_LAST\_PHONE\_CHANGE"]]

corr\_data.head()

corr\_data.corr()

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

sns.heatmap(corr\_data.corr(),annot=True,cmap="RdYlGn")

plt.show()

### Split the Numberical variable based on Traget 0 and 1 to find the co\_relation

corr\_data\_0=tar\_0[["AMT\_INCOME\_TOTAL","AMT\_CREDIT","AMT\_ANNUITY","AMT\_GOODS\_PRICE",

"YEARS\_BIRTH","YEARS\_ELPLOYED","YEARS\_REGISTRATION","YEARS\_ID\_PUBLISH","YEARS\_LAST\_PHONE\_CHANGE"]]

corr\_data\_0.head()

corr\_data\_1=tar\_1[["AMT\_INCOME\_TOTAL","AMT\_CREDIT","AMT\_ANNUITY","AMT\_GOODS\_PRICE",

"YEARS\_BIRTH","YEARS\_ELPLOYED","YEARS\_REGISTRATION","YEARS\_ID\_PUBLISH","YEARS\_LAST\_PHONE\_CHANGE"]]

corr\_data\_1.head()

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

sns.heatmap(corr\_data\_0.corr(),annot=True,cmap="RdYlGn")

plt.show()

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

sns.heatmap(corr\_data\_1.corr(),annot=True,cmap="RdYlGn")

plt.show()

### Impute Missing values

#### Check the dtype of missing values in Application dataset before imputing values

for col in papp\_df.columns:

if papp\_df[col].dtypes == np.int64 or papp\_df[col].dtypes == np.float64:

papp\_df[col]=papp\_df[col].apply(lambda x:abs(x))

### Validate if any null values present in dataset

null\_cols= list(papp\_df.columns[papp\_df.isna().any()])

len(null\_cols)

papp\_df.isnull().mean()\*100

### Binnig od continuous variable

#### Binning AMT\_CREDIT Column

papp\_df.AMT\_CREDIT.describe()

papp\_df["AMT\_CREDIT\_Category"]=pd.cut(papp\_df.AMT\_CREDIT, [0,200000,400000,600000,800000,1000000],

labels=["Very low Credit","Low Credit","Medium Credit","High Credit","Very High Credit"])

papp\_df["AMT\_CREDIT\_Category"].value\_counts(normalize=True).plot.bar()

plt.show()

papp\_df['AMT\_GOODS\_PRICE\_Category'] = pd.qcut(

papp\_df.AMT\_GOODS\_PRICE,q=[0, 0.25, 0.45, 0.65, 0.85, 1],

labels=["Very low Price", "Low Price", "Medium Price", "High Price", "Very High Price"]

)

papp\_df['AMT\_GOODS\_PRICE\_Category'].value\_counts(normalize=True).plot.pie(autopct='%1.2f%%')

plt.legend()

plt.show()

# Data Imbalance Check

### Dividing Application Dataset with NAME\_CONTRACT\_STATUS

approved = papp\_df[papp\_df.NAME\_CONTRACT\_STATUS == "Approved"]

cancelled = papp\_df[papp\_df.NAME\_CONTRACT\_STATUS == "Canceled"]

refused = papp\_df[papp\_df.NAME\_CONTRACT\_STATUS == "Refused"]

unused = papp\_df[papp\_df.NAME\_CONTRACT\_STATUS == "Unused offer"]

papp\_df.NAME\_CONTRACT\_STATUS.value\_counts(normalize =True)\*100

papp\_df.NAME\_CONTRACT\_STATUS.value\_counts(normalize=True).plot.pie(autopct='%1.2f%%')

plt.legend()

plt.show()

# Univariate Analysis

cat\_cols = list(papp\_df.columns[papp\_df.dtypes == object])

num\_cols = list(papp\_df.columns[papp\_df.dtypes == np.int64])+ list(papp\_df.columns[papp\_df.dtypes == np.float64])

cat\_cols

num\_cols

cat\_cols=["NAME\_CONTRACT\_TYPE","WEEKDAY\_APPR\_PROCESS\_START","NAME\_CONTRACT\_STATUS","NAME\_PAYMENT\_TYPE","NAME\_SELLER\_INDUSTRY","CHANNEL\_TYPE","NAME\_YIELD\_GROUP","PRODUCT\_COMBINATION"]

num\_cols=["HOUR\_APPR\_PROCESS\_START","DAYS\_DECISION","AMT\_ANNUITY","AMT\_APPLICATION","AMT\_CREDIT","AMT\_GOODS\_PRICE","CNT\_PAYMENT"]

plt.legend()

plt.show()

print("-----------------------------------------------------------------------------------")

### Bivariate and Multivariate Analysis

#### Bivariate Analysis between WEEKDAY\_APPR\_PROCESS\_START VS AMT\_APPLICATION

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

sns.barplot(x='WEEKDAY\_APPR\_PROCESS\_START',y='AMT\_APPLICATION',data=approved)

plt.title("Plot for Approved")

plt.show()

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

sns.barplot(x='WEEKDAY\_APPR\_PROCESS\_START',y='AMT\_APPLICATION',data=refused)

plt.title("Plot for refused")

plt.show()

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

sns.barplot(x='WEEKDAY\_APPR\_PROCESS\_START',y='AMT\_APPLICATION',data=unused)

plt.title("Plot for unused")

plt.show()

corr\_refused=refused[["DAYS\_DECISION","AMT\_ANNUITY","AMT\_APPLICATION","AMT\_CREDIT","AMT\_GOODS\_PRICE","CNT\_PAYMENT"]]

corr\_cancelled=cancelled[["DAYS\_DECISION","AMT\_ANNUITY","AMT\_APPLICATION","AMT\_CREDIT","AMT\_GOODS\_PRICE","CNT\_PAYMENT"]]

corr\_unused=unused[["DAYS\_DECISION","AMT\_ANNUITY","AMT\_APPLICATION","AMT\_CREDIT","AMT\_GOODS\_PRICE","CNT\_PAYMENT"]]

### Co-relation for Numerical columns for Approved

plt.figure(figsize=[10,10])

sns.heatmap(corr\_approved.corr(),annot=True,cmap="Blues")

plt.title("Heat Map plot for approved")

plt.show()

#### Co-relation for Numerical columns for Refused

plt.figure(figsize=[10,10])

sns.heatmap(corr\_refused.corr(),annot=True,cmap="Blues")

plt.title("Heat Map plot for Refused")

plt.show()

#### Co-relation for Numerical columns for Cancelled

plt.figure(figsize=[10,10])

sns.heatmap(corr\_cancelled.corr(),annot=True,cmap="Blues")

plt.title("Heat Map plot for Cancelled")

plt.show()

#### Filtering required columns for our Analysis

for col in merge\_df.columns:

if col.startswith("FLAG"):

merge\_df.drop(columns=col, axis=1,inplace=True)

merge\_df.shape

res1 = pd.pivot\_table(data=merge\_df, index=["NAME\_INCOME\_TYPE","NAME\_CLIENT\_TYPE"], columns=["NAME\_CONTRACT\_STATUS"],values="TARGET", aggfunc="mean")

plt.figure(figsize=[10,10])

sns.heatmap(res1,annot=True,cmap='BuPu')

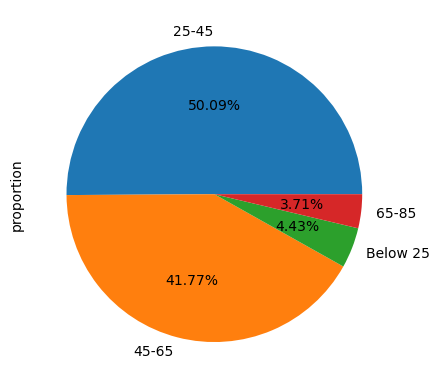
plt.show()

res2 = pd.pivot\_table(data=merge\_df, index=["CODE\_GENDER","NAME\_SELLER\_INDUSTRY"],

columns=["TARGET"], values="AMT\_GOODS\_PRICE\_x", aggfunc='sum')

sns.heatmap(res2,annot=True,cmap='BuPu') plt.figure(figsize=[12,15]sns.heatmap(res2,annot=True,cmap='BuPu') plt.show()

5 Output:



NAME\_CONTRACT\_TYPE

Cash loans 0.904787

Revolving loans 0.095213

Name: proportion, dtype: float64

CODE\_GENDER

F 0.658344

M 0.341643

XNA 0.000013

Name: proportion, dtype: float64

FLAG\_OWN\_CAR

N 0.659892

Y 0.340108

Name: proportion, dtype: float64

FLAG\_OWN\_REALTY

Y 0.693673

N 0.306327

Name: proportion, dtype: float64

NAME\_TYPE\_SUITE

Unaccompanied 0.812387

Family 0.130561

Spouse, partner 0.036974

Children 0.010624

Other\_B 0.005756

Other\_A 0.002816

Group of people 0.000881

Name: proportion, dtype: float64

NAME\_INCOME\_TYPE

Working 0.516320

Commercial associate 0.232892

Pensioner 0.180033

State servant 0.070576

Unemployed 0.000072

Student 0.000059

Businessman 0.000033

Maternity leave 0.000016

Name: proportion, dtype: float64

NAME\_EDUCATION\_TYPE

Secondary / secondary special 0.710189

Higher education 0.243448

Incomplete higher 0.033420

Lower secondary 0.012409

Academic degree 0.000533

Name: proportion, dtype: float64

NAME\_FAMILY\_STATUS

Married 0.638780

Single / not married 0.147780

Civil marriage 0.096826

Separated 0.064290

Widow 0.052317

Unknown 0.000007

Name: proportion, dtype: float64

NAME\_HOUSING\_TYPE

House / apartment 0.887344

With parents 0.048258

Municipal apartment 0.036366

Rented apartment 0.015873

Office apartment 0.008510

Co-op apartment 0.003649

Name: proportion, dtype: float64

OCCUPATION\_TYPE

Others 0.313455

Laborers 0.179460

Sales staff 0.104393

Core staff 0.089655

Managers 0.069497

Drivers 0.060495

High skill tech staff 0.037007

Accountants 0.031911

Medicine staff 0.027762

Security staff 0.021856

Restaurant 0.005889

Services 0.005122

University 0.004315

Industry: type 7 0.004250

Transport: type 3 0.003860

Industry: type 1 0.003379

Hotel 0.003141

Electricity 0.003089

Industry: type 4 0.002852

Trade: type 6 0.002052

Industry: type 5 0.001948

Insurance 0.001941

Telecom 0.001876

Emergency 0.001821

Industry: type 2 0.001489

Advertising 0.001395

Realtor 0.001288

Culture 0.001232

Industry: type 12 0.001200

Trade: type 1 0.001132

Mobile 0.001031

Legal Services 0.000992

Cleaning 0.000845

Transport: type 1 0.000654

Industry: type 6 0.000364

Industry: type 10 0.000354

Religion 0.000276

Industry: type 13 0.000218

Trade: type 4 0.000208

Trade: type 5 0.000159

Industry: type 8 0.000078

Name: proportion, dtype: float64

count 307511.000000

mean 278180.518577

std 102790.175348

min 100002.000000

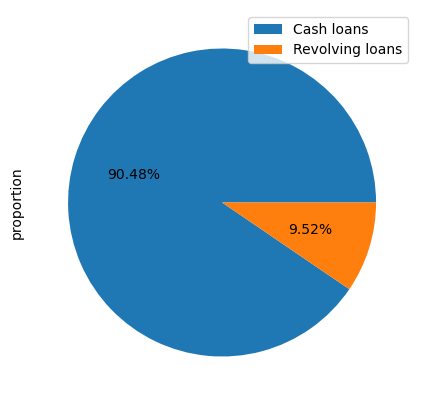
25% 189145.500000

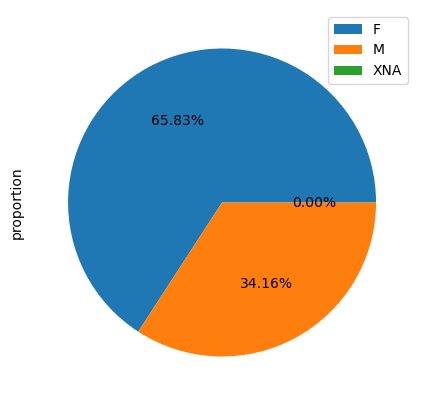
50% 278202.000000

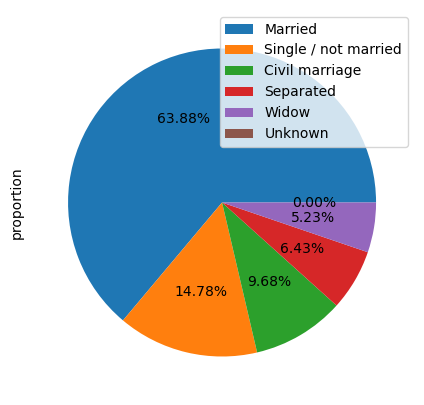
75% 367142.500000

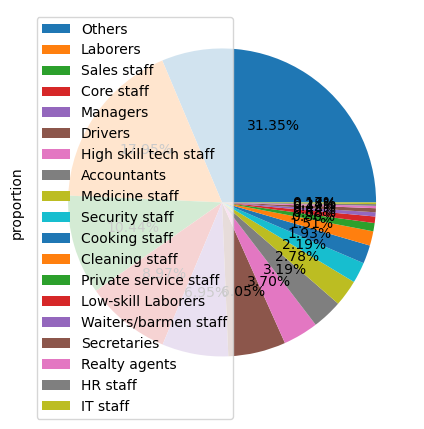
max 456255.000000

Name: SK\_ID\_CURR, dtype: float64









-----------------------------------------

count 307511.000000

mean 0.080729

std 0.272419

min 0.000000

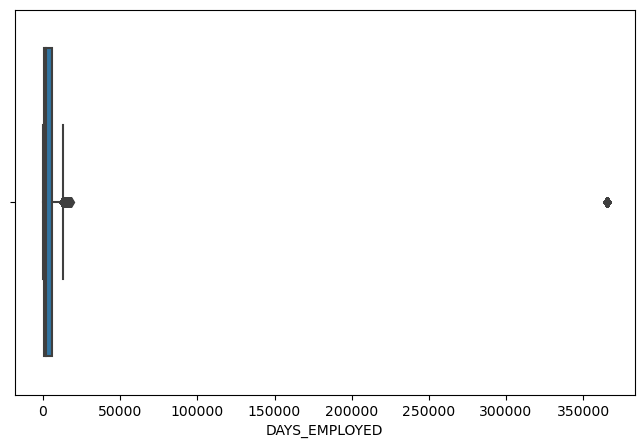
25% 0.000000

50% 0.000000

75% 0.000000

max 1.000000

Name: TARGET, dtype: float64



-----------------------------------------

count 307511.000000

mean 2994.202373

std 1509.450419

min 0.000000

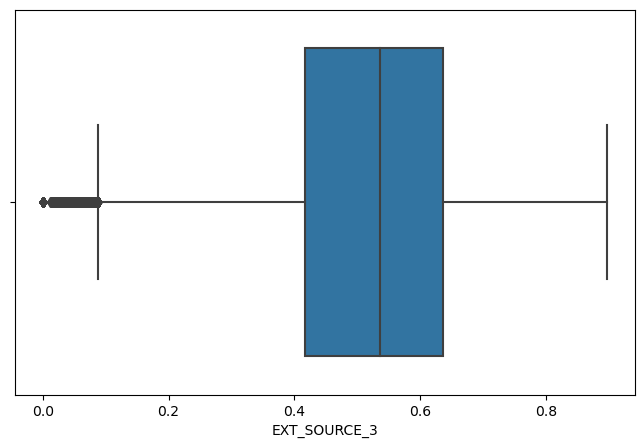
25% 1720.000000

50% 3254.000000

75% 4299.000000

max 7197.000000

Name: DAYS\_ID\_PUBLISH, dtype: float64



-----------------------------------------

count 307511.000000

mean 1.417523

std 2.398395

min 0.000000

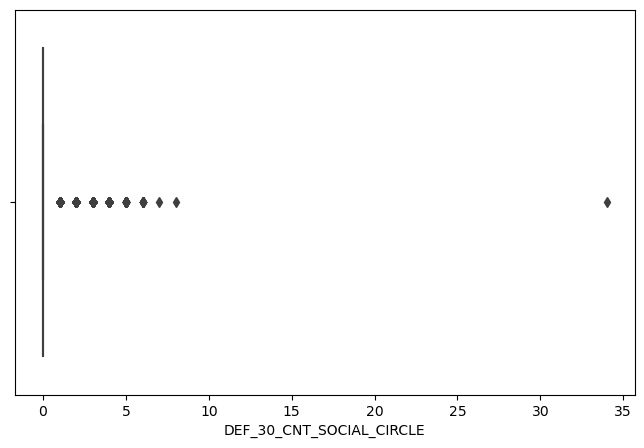
25% 0.000000

50% 0.000000

75% 2.000000

max 348.000000

Name: OBS\_30\_CNT\_SOCIAL\_CIRCLE, dtype: float64



-----------------------------------------

count 307511.000000

mean 1.400626

std 2.377224

min 0.000000

25% 0.000000

50% 0.000000

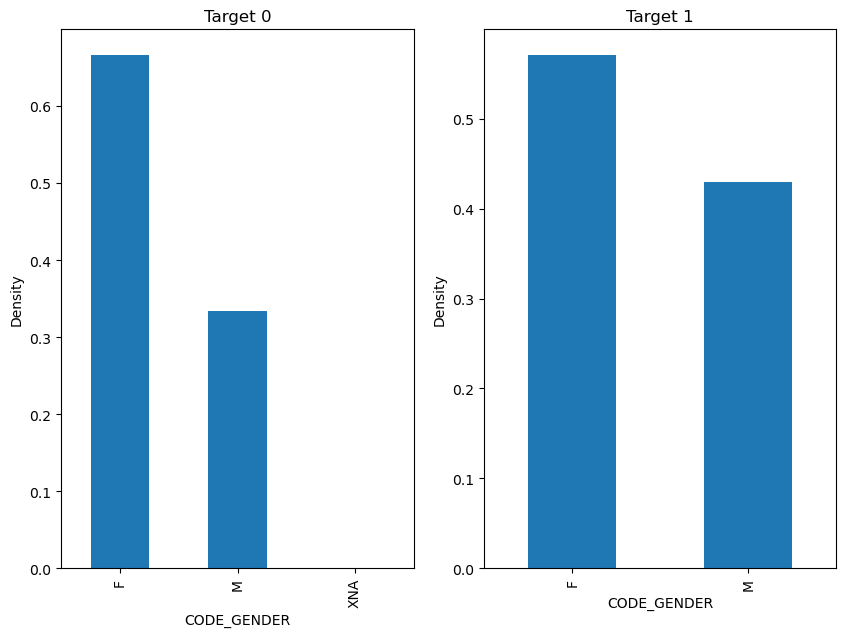
75% 2.000000

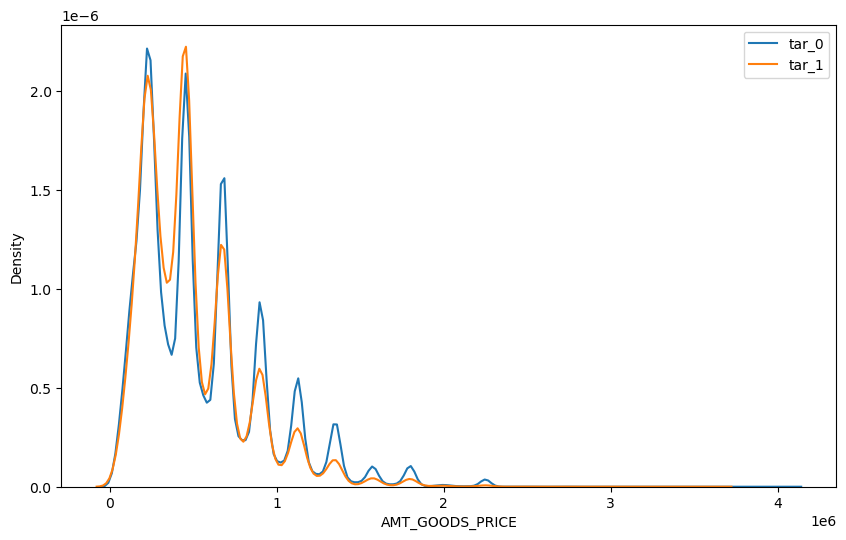
max 344.000000

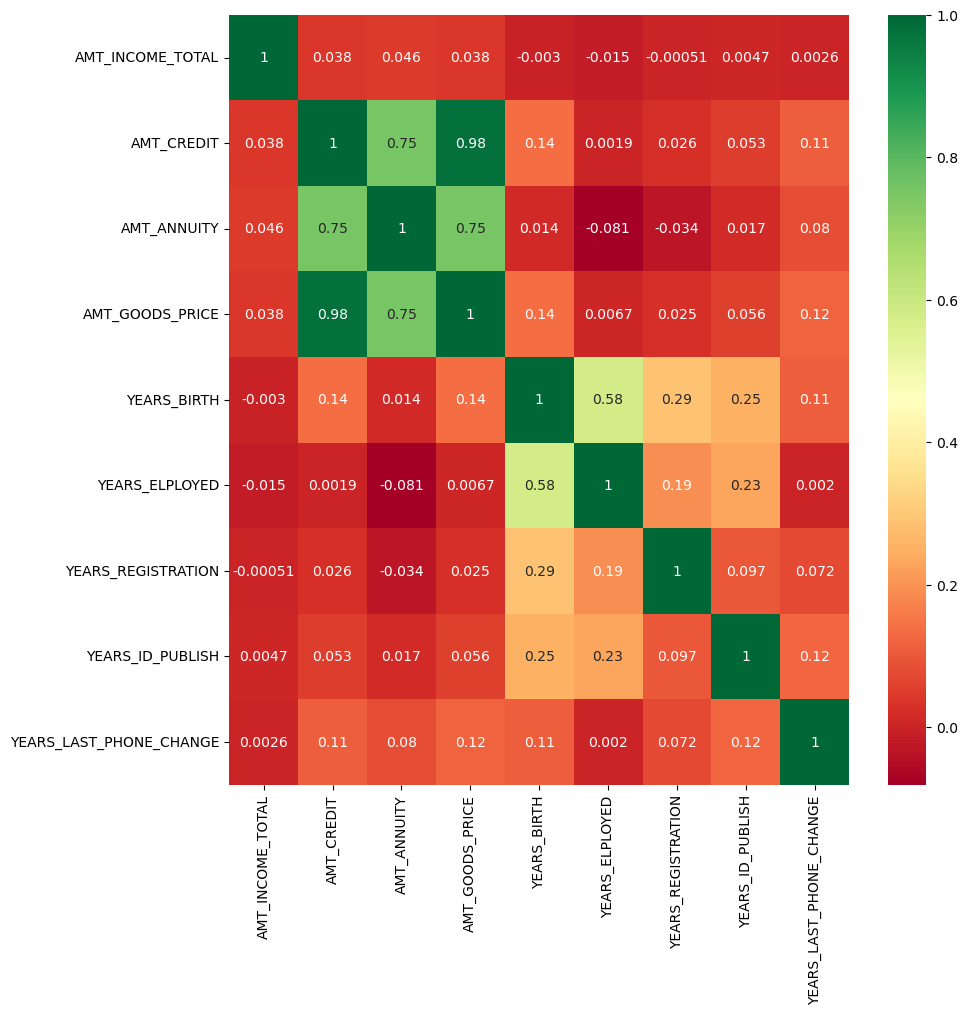
Name: OBS\_60\_CNT\_SOCIAL\_CIRCLE, dtype: float64

-------------------------------------------------------------------------------------------------------

plot on CODE\_GENDER for Target o to 1







**---------------------------------------------------------------------------**