1.Case Study Introduction

This case study aims to give you an idea of applying EDA in a real business scenario. You will also develop a basic understanding of risk analytics in banking and financial services and understand how data is used to minimize the risk of losing money while lending to customers.

1.1 Business Understanding

The loan providing companies find it hard to give loans to the people due to their insufficient or non-existent credit history. Because of that, some consumers use it to their advantage by becoming a defaulter. Suppose you work for a consumer finance company which specializes in lending various types of loans to urban customers. You have to use EDA to analyze the patterns present in the data. This will ensure that the applicants capable of repaying the loan are not rejected. 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.The data given below contains the information about the loan application at the time of applying for the loan. It contains two types of scenarios:

   • The client with payment difficulties: he/she had late payment more than X days on at least one of the first Y installments of the loan in our sample

   • All other cases: All other cases when the payment is paid on time.

   When a client applies for a loan, there are four types of decisions that could be taken by the client/company):

   • Approved: The Company has approved loan Application

   • Cancelled: The client cancelled the application sometime during approval. Either the client changed her/his mind about the loan or in some cases due to a higher risk of the client he received worse pricing which he did not want.

• Refused: The company had rejected the loan (because the client does not meet their requirements etc.).

• Unused offer: Loan has been cancelled by the client but on different stages of the process.

1.2 Business Objectives

This case study aims to identify patterns which indicate if a client has difficulty paying their instalments which may be used for taking actions such as denying the loan, reducing the amount of loan, lending (to risky applicants) at a higher interest rate, etc. This will ensure that the consumers capable of repaying the loan are not rejected. Identification of such applicants using EDA is the aim of this case study. In other words, the company wants to understand the driving factors (or driver variables) behind loan default, i.e. the variables which are strong indicators of default. The company can utilise this knowledge for its portfolio and risk assessment.

1.3 Data Understanding

Our dataset has 3 files as explained below:

1. application_data.csv' contains all the information of the client at the time of application. The data is about whether a client has payment difficulties.
2. previous_application.csv' contains information about the client's previous loan data. It contains the data whether the previous application had been Approved, Cancelled, Refused or Unused offer.
3. columns_description.csv' is data dictionary which describes the meaning of the variables.

2. Data Exploration

2.1 Read the data file

We will first read the data file which includes importing the required libraries, loading the dataset and viewing the data file.

```
# import the required libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```

```
# filter out the warnings
import warnings
warnings.filterwarnings('ignore')
```

```
# setting maximum rows & columns display size to 200 for better visibility of c
pd.set_option('display.max_rows', 200)
pd.set_option('display.max_columns', 200)
```

```
# load dataset
app_df =  pd.read_csv('application_data.csv')
```

```
# viewing first 5 rows
app_df.head(5)
```

```
#viewing last 5 rows
app_df.tail(5)
```

2.2 Inspect the data frame

Next we will check the various attributes like shape (rows and cols) & datatypes

```
app_df.shape
```

```
(307511, 122)
```

```
app_df.columns.values
```

```
array(['SK_ID_CURR', 'TARGET', 'NAME_CONTRACT_TYPE', 'CODE_GENDER',
       'FLAG_OWN_CAR', 'FLAG_OWN_REALTY', 'CNT_CHILDREN',
       'AMT_INCOME_TOTAL', 'AMT_CREDIT', 'AMT_ANNUITY', 'AMT_GOODS_PRICE',
       'NAME_TYPE_SUITE', 'NAME_INCOME_TYPE', 'NAME_EDUCATION_TYPE',
       'NAME_FAMILY_STATUS', 'NAME_HOUSING_TYPE',
       'REGION_POPULATION_RELATIVE', 'DAYS_BIRTH', 'DAYS_EMPLOYED',
       'DAYS_REGISTRATION', 'DAYS_ID_PUBLISH', 'OWN_CAR_AGE',
       'FLAG_MOBIL', 'FLAG_EMP_PHONE', 'FLAG_WORK_PHONE',
       'FLAG_CONT_MOBILE', 'FLAG_PHONE', 'FLAG_EMAIL', 'OCCUPATION_TYPE',
       'CNT_FAM_MEMBERS', 'REGION_RATING_CLIENT',
       'REGION_RATING_CLIENT_W_CITY', 'WEEKDAY_APPR_PROCESS_START',
       'HOUR_APPR_PROCESS_START', 'REG_REGION_NOT_LIVE_REGION',
       'REG_REGION_NOT_WORK_REGION', 'LIVE_REGION_NOT_WORK_REGION',
       'REG_CITY_NOT_LIVE_CITY', 'REG_CITY_NOT_WORK_CITY',
       'LIVE_CITY_NOT_WORK_CITY', 'ORGANIZATION_TYPE', 'EXT_SOURCE_1',
       'EXT_SOURCE_2', 'EXT_SOURCE_3', 'APARTMENTS_AVG',
       'BASEMENTAREA_AVG', 'YEARS_BEGINEXPLUATATION_AVG',
       'YEARS_BUILD_AVG', 'COMMONAREA_AVG', 'ELEVATORS_AVG',
       'ENTRANCES_AVG', 'FLOORSMAX_AVG', 'FLOORSMIN_AVG', 'LANDAREA_AVG',
       'LIVINGAPARTMENTS_AVG', 'LIVINGAREA_AVG',
       'NONLIVINGAPARTMENTS_AVG', 'NONLIVINGAREA_AVG', 'APARTMENTS_MODE',
       'BASEMENTAREA_MODE', 'YEARS_BEGINEXPLUATATION_MODE',
       'YEARS_BUILD_MODE', 'COMMONAREA_MODE', 'ELEVATORS_MODE',
       'ENTRANCES_MODE', 'FLOORSMAX_MODE', 'FLOORSMIN_MODE',
       'LANDAREA_MODE', 'LIVINGAPARTMENTS_MODE', 'LIVINGAREA_MODE',
       'NONLIVINGAPARTMENTS_MODE', 'NONLIVINGAREA_MODE',
       'APARTMENTS_MEDI', 'BASEMENTAREA_MEDI',
       'YEARS_BEGINEXPLUATATION_MEDI', 'YEARS_BUILD_MEDI',
       'COMMONAREA_MEDI', 'ELEVATORS_MEDI', 'ENTRANCES_MEDI',
       'FLOORSMAX_MEDI', 'FLOORSMIN_MEDI', 'LANDAREA_MEDI',
       'LIVINGAPARTMENTS_MEDI', 'LIVINGAREA_MEDI',
       'NONLIVINGAPARTMENTS_MEDI', 'NONLIVINGAREA_MEDI',
       'FONDKAPREMONT_MODE', 'HOUSETYPE_MODE', 'TOTALAREA_MODE',
       'WALLSMATERIAL_MODE', 'EMERGENCYSTATE_MODE',
       'OBS_30_CNT_SOCIAL_CIRCLE', 'DEF_30_CNT_SOCIAL_CIRCLE',
       'OBS_60_CNT_SOCIAL_CIRCLE', 'DEF_60_CNT_SOCIAL_CIRCLE',
       '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',
       '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'],
      dtype=object)
```

```python
# checking the data types of all the columns
app_df.dtypes
```

```
SK_ID_CURR                       int64
TARGET                           int64
NAME_CONTRACT_TYPE              object
CODE_GENDER                     object
FLAG_OWN_CAR                    object
FLAG_OWN_REALTY                 object
CNT_CHILDREN                     int64
AMT_INCOME_TOTAL               float64
AMT_CREDIT                     float64
AMT_ANNUITY                    float64
AMT_GOODS_PRICE                float64
NAME_TYPE_SUITE                 object
NAME_INCOME_TYPE                object
```

```
NAME_EDUCATION_TYPE              object
NAME_FAMILY_STATUS               object
NAME_HOUSING_TYPE                object
REGION_POPULATION_RELATIVE      float64
DAYS_BIRTH                        int64
DAYS_EMPLOYED                     int64
DAYS_REGISTRATION               float64
DAYS_ID_PUBLISH                   int64
OWN_CAR_AGE                     float64
FLAG_MOBIL                        int64
FLAG_EMP_PHONE                    int64
FLAG_WORK_PHONE                   int64
FLAG_CONT_MOBILE                  int64
FLAG_PHONE                        int64
FLAG_EMAIL                        int64
OCCUPATION_TYPE                  object
CNT_FAM_MEMBERS                 float64
REGION_RATING_CLIENT              int64
REGION_RATING_CLIENT_W_CITY       int64
WEEKDAY_APPR_PROCESS_START       object
HOUR_APPR_PROCESS_START           int64
REG_REGION_NOT_LIVE_REGION        int64
REG_REGION_NOT_WORK_REGION        int64
LIVE_REGION_NOT_WORK_REGION       int64
REG_CITY_NOT_LIVE_CITY            int64
REG_CITY_NOT_WORK_CITY            int64
LIVE_CITY_NOT_WORK_CITY           int64
ORGANIZATION_TYPE                object
EXT_SOURCE_1                    float64
EXT_SOURCE_2                    float64
EXT_SOURCE_3                    float64
APARTMENTS_AVG                  float64
BASEMENTAREA_AVG                float64
YEARS_BEGINEXPLUATATION_AVG     float64
YEARS_BUILD_AVG                 float64
COMMONAREA_AVG                  float64
ELEVATORS_AVG                   float64
ENTRANCES_AVG                   float64
FLOORSMAX_AVG                   float64
FLOORSMIN_AVG                   float64
LANDAREA_AVG                    float64
LIVINGAPARTMENTS_AVG            float64
LIVINGAREA_AVG                  float64
NONLIVINGAPARTMENTS_AVG         float64
```

```
# check the descriptive statistics of numerical variables
#convert to float for better visibility
pd.options.display.float_format = '{:,.2f}'.format
app_df.describe()
```

- Around 8% of clients defaulted (TARGET = 1) → data is imbalanced.

- Average annual income ≈ 168k, but some extremely high values (outliers).
- Average loan amount ≈ 600k; several high-value loans above 3M.
- Typical applicant age ≈ 44 years old.
- Majority have 0–1 child; very few with >5.

```
app_df['TARGET'].value_counts()
```

```
TARGET
0    282686
1     24825
Name: count, dtype: int64
```

```
app_df['TARGET'].value_counts()/len(app_df) * 100
```

```
TARGET
0   91.93
1    8.07
Name: count, dtype: float64
```

```
#set figure size
plt.figure(figsize=(4,3))
sns.countplot(data=app_df, x='TARGET')
plt.title('Count of TARGET variable per value')
plt.xlabel('Target Variable')
plt.ylabel('Count')
plt.show()
```

```
#concise summary of the dataframe to view data types of all columns & null valu
#to display all columns use verbose = true
app_df.info(verbose=True, memory_usage=True)
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 307511 entries, 0 to 307510
Data columns (total 122 columns):
 #    Column                        Dtype
---   ------                        -----
 0    SK_ID_CURR                    int64
 1    TARGET                        int64
 2    NAME_CONTRACT_TYPE            object
 3    CODE_GENDER                   object
 4    FLAG_OWN_CAR                  object
 5    FLAG_OWN_REALTY               object
 6    CNT_CHILDREN                  int64
 7    AMT_INCOME_TOTAL              float64
 8    AMT_CREDIT                    float64
 9    AMT_ANNUITY                   float64
 10   AMT_GOODS_PRICE               float64
 11   NAME_TYPE_SUITE               object
 12   NAME_INCOME_TYPE              object
 13   NAME_EDUCATION_TYPE           object
 14   NAME_FAMILY_STATUS            object
 15   NAME_HOUSING_TYPE             object
 16   REGION_POPULATION_RELATIVE    float64
 17   DAYS_BIRTH                    int64
 18   DAYS_EMPLOYED                 int64
 19   DAYS_REGISTRATION             float64
 20   DAYS_ID_PUBLISH               int64
 21   OWN_CAR_AGE                   float64
 22   FLAG_MOBIL                    int64
 23   FLAG_EMP_PHONE                int64
 24   FLAG_WORK_PHONE               int64
 25   FLAG_CONT_MOBILE              int64
 26   FLAG_PHONE                    int64
 27   FLAG_EMAIL                    int64
 28   OCCUPATION_TYPE               object
 29   CNT_FAM_MEMBERS               float64
 30   REGION_RATING_CLIENT          int64
 31   REGION_RATING_CLIENT_W_CITY   int64
 32   WEEKDAY_APPR_PROCESS_START    object
 33   HOUR_APPR_PROCESS_START       int64
 34   REG_REGION_NOT_LIVE_REGION    int64
 35   REG_REGION_NOT_WORK_REGION    int64
 36   LIVE_REGION_NOT_WORK_REGION   int64
 37   REG_CITY_NOT_LIVE_CITY        int64
 38   REG_CITY_NOT_WORK_CITY        int64
 39   LIVE_CITY_NOT_WORK_CITY       int64
 40   ORGANIZATION_TYPE             object
 41   EXT_SOURCE_1                  float64
 42   EXT_SOURCE_2                  float64
 43   EXT_SOURCE_3                  float64
 44   APARTMENTS_AVG                float64
 45   BASEMENTAREA_AVG              float64
 46   YEARS_BEGINEXPLUATATION_AVG   float64
 47   YEARS_BUILD_AVG               float64
 48   COMMONAREA_AVG                float64
 49   ELEVATORS_AVG                 float64
 50   ENTRANCES_AVG                 float64
 51   FLOORSMAX_AVG                 float64
 52   FLOORSMIN_AVG                 float64
```

```
#Count null values per column
null_count = app_df.isnull().sum()
#Total number of rows
total_count = app_df.shape[0]
# Calculate percentage of nulls and round to 2 decimal places
null_pct = ((null_count / total_count) * 100).round(2)
#Filter only columns with missing values (makes plot cleaner)
null_pct = null_pct[null_pct > 0].sort_values(ascending=False)

#Plot using pointplot
plt.figure(figsize=(16,5))
sns.pointplot(x=null_pct.index, y=null_pct.values)
plt.xticks(rotation=90)
plt.xlabel("Columns")
plt.ylabel("Missing Values (%)")
plt.title("Percentage of Null Values per Column")
plt.show()
```

3. Data Cleaning

3.1 Missing Value Imputation

```
##create a copy of base data for manipulation & processing
new_app_df = app_df.copy()
```

```
#check the new dataframe
new_app_df.head(5)
```

```
#check for missing value in percentage
null_pct.sort_values(ascending=True)
```

```
AMT_GOODS_PRICE                  0.09
EXT_SOURCE_2                     0.21
DEF_30_CNT_SOCIAL_CIRCLE         0.33
OBS_30_CNT_SOCIAL_CIRCLE         0.33
OBS_60_CNT_SOCIAL_CIRCLE         0.33
DEF_60_CNT_SOCIAL_CIRCLE         0.33
NAME_TYPE_SUITE                  0.42
AMT_REQ_CREDIT_BUREAU_MON       13.50
AMT_REQ_CREDIT_BUREAU_WEEK      13.50
AMT_REQ_CREDIT_BUREAU_YEAR      13.50
AMT_REQ_CREDIT_BUREAU_HOUR      13.50
AMT_REQ_CREDIT_BUREAU_QRT       13.50
AMT_REQ_CREDIT_BUREAU_DAY       13.50
EXT_SOURCE_3                    19.83
OCCUPATION_TYPE                 31.35
EMERGENCYSTATE_MODE             47.40
TOTALAREA_MODE                  48.27
YEARS_BEGINEXPLUATATION_MEDI    48.78
YEARS_BEGINEXPLUATATION_AVG     48.78
YEARS_BEGINEXPLUATATION_MODE    48.78
FLOORSMAX_MEDI                  49.76
FLOORSMAX_MODE                  49.76
FLOORSMAX_AVG                   49.76
HOUSETYPE_MODE                  50.18
LIVINGAREA_MODE                 50.19
LIVINGAREA_MEDI                 50.19
```

```
LIVINGAREA_AVG                  50.19
ENTRANCES_MODE                  50.35
ENTRANCES_MEDI                  50.35
ENTRANCES_AVG                   50.35
APARTMENTS_AVG                  50.75
APARTMENTS_MODE                 50.75
APARTMENTS_MEDI                 50.75
WALLSMATERIAL_MODE              50.84
ELEVATORS_AVG                   53.30
ELEVATORS_MODE                  53.30
ELEVATORS_MEDI                  53.30
NONLIVINGAREA_AVG               55.18
NONLIVINGAREA_MODE              55.18
NONLIVINGAREA_MEDI              55.18
EXT_SOURCE_1                    56.38
BASEMENTAREA_MEDI               58.52
BASEMENTAREA_AVG                58.52
BASEMENTAREA_MODE               58.52
LANDAREA_MODE                   59.38
LANDAREA_AVG                    59.38
LANDAREA_MEDI                   59.38
OWN_CAR_AGE                     65.99
YEARS_BUILD_MODE                66.50
YEARS_BUILD_AVG                 66.50
YEARS_BUILD_MEDI                66.50
FLOORSMIN_AVG                   67.85
FLOORSMIN_MODE                  67.85
FLOORSMIN_MEDI                  67.85
LIVINGAPARTMENTS_MODE           68.35
LIVINGAPARTMENTS_AVG            68.35
LIVINGAPARTMENTS_MEDI           68.35
FONDKAPREMONT_MODE              68.39
```

```
# get ONLY column names with >50% missing
cols_50 = null_pct[null_pct >= 50].index.tolist()
print("Columns with >50% missing values:")
print(cols_50)
```

```
Columns with >50% missing values:
['COMMONAREA_AVG', 'COMMONAREA_MODE', 'COMMONAREA_MEDI', 'NONLIVINGAPARTMENTS_AV
```

- Since these columns have more than 50% null values, it is wise to drop them because when there is too little real data (e.x > 50% values missing), there is a higher biasness risk.

```
new_app_df.drop(cols_50, axis = 1, inplace = True)
new_app_df.shape
```

```
(307511, 81)
```

- We will also drop columns with more than 40% null values as they will also introduce biasness to our dataset

```
# get the column names with >40% missing
```

```
cols_between_40_and_50 = null_pct[(null_pct > 40) & (null_pct < 50)].index.toli
cols_between_40_and_50
```

```
['FLOORSMAX_AVG',
 'FLOORSMAX_MEDI',
 'FLOORSMAX_MODE',
 'YEARS_BEGINEXPLUATATION_AVG',
 'YEARS_BEGINEXPLUATATION_MODE',
 'YEARS_BEGINEXPLUATATION_MEDI',
 'TOTALAREA_MODE',
 'EMERGENCYSTATE_MODE']
```

```
new_app_df.drop(cols_between_40_and_50, axis=1, inplace=True)
new_app_df.shape
```

```
(307511, 114)
```

- As OCCUPATION_TYPE is a categorical variable and is of object type and since the missing value percentage is high (31.35%) we could NOT take its mode value to fill the missing ones because that will simply make the data biased. So, it would be safe to rather create a new type 'Unknown' to fill the missing values.For the rest of categorical columns, we will fill them with mode.

```
cat_cols = new_app_df.select_dtypes(include='object').columns

for col in cat_cols:
    if col in ['OCCUPATION_TYPE']:
        new_app_df[col].fillna('Unknown', inplace=True)
    else:
        new_app_df[col].fillna(new_app_df[col].mode()[0], inplace=True)
```

```
C:\Users\HP\AppData\Local\Temp\ipykernel_10208\2814705985.py:7: FutureWarning: A
The behavior will change in pandas 3.0. This inplace method will never work beca

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.met


  new_app_df[col].fillna(new_app_df[col].mode()[0], inplace=True)
C:\Users\HP\AppData\Local\Temp\ipykernel_10208\2814705985.py:5: FutureWarning: A
The behavior will change in pandas 3.0. This inplace method will never work beca

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.met
```

```
  new_app_df[col].fillna('Unknown', inplace=True)
```

- For the rest of the numerical columns, we will impute them with median.

```
num_cols = new_app_df.select_dtypes(include=['int64', 'float64']).columns
for col in num_cols:
    new_app_df[col].fillna(new_app_df[col].median(), inplace=True)
```

```
C:\Users\HP\AppData\Local\Temp\ipykernel_10208\3182007245.py:3: FutureWarning: A
The behavior will change in pandas 3.0. This inplace method will never work beca

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.met


  new_app_df[col].fillna(new_app_df[col].median(), inplace=True)
```

```
#sanity check to see if there are any null columns left
null_col_check = new_app_df.columns.isna().sum()
null_col_check
```

```
np.int64(0)
```

3.2 Data Correction

- Firstly, we will convert the columns FLAG_OWN_CAR & FLAG_OWN_REALTY to int

```
# fix FLAG columns (Y/N → 1/0)
flag_text_cols = ['FLAG_OWN_CAR', 'FLAG_OWN_REALTY']
for col in flag_text_cols:
    new_app_df[col] = new_app_df[col].map({'Y': 1, 'N': 0}).astype('int64')
```

- We will also change these columns as count is usually in int
    - CNT_FAM_MEMBERS
    - OBS_30_CNT_SOCIAL_CIRCLE, DEF_30_CNT_SOCIAL_CIRCLE
    - OBS_60_CNT_SOCIAL_CIRCLE, DEF_60_CNT_SOCIAL_CIRCLE
    - 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

```
count_cols = [
    'CNT_FAM_MEMBERS',
    'OBS_30_CNT_SOCIAL_CIRCLE', 'DEF_30_CNT_SOCIAL_CIRCLE',
    'OBS_60_CNT_SOCIAL_CIRCLE', 'DEF_60_CNT_SOCIAL_CIRCLE',
    '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 count_cols:
    new_app_df[col] = new_app_df[col].astype('int64')
```

- We will also change these columns to int as days are usually in int
- DAYS_BIRTH, DAYS_EMPLOYED, DAYS_REGISTRATION, DAYS_ID_PUBLISH, DAYS_LAST_PHONE_CHANGE

```
days_cols = ['DAYS_BIRTH','DAYS_EMPLOYED','DAYS_REGISTRATION','DAYS_ID_PUBLISH'

for col in days_cols:
    new_app_df[col] = new_app_df[col].astype('int64')
```

3.3 Data Standardization

- There are some columns that are negative in value but should be positive as a negative value wouldn't make sense in such columns. We need to convert them to positive using their absolute values.Those columns are listed below.
    - DAYS_BIRTH, DAYS_EMPLOYED, DAYS_REGISTRATION, DAYS_ID_PUBLISH, DAYS_LAST_PHONE_CHANGE

```
pos_days_cols = ['DAYS_BIRTH','DAYS_EMPLOYED','DAYS_REGISTRATION','DAYS_ID_PUBL
# concvert to absolute values
for col in pos_days_cols:
    new_app_df[col]= new_app_df[col].abs()
```

- We should also convert columns DAYS_BIRTH & DAYS_EMPLOYED to AGE_YEARS & YEARS_EMPLOYED for better readability

```
new_app_df['AGE_YEARS']= (new_app_df['DAYS_BIRTH'] / 365.25).round().astype('ir
new_app_df['AGE_YEARS']
```

```
0         26
1         46
2         52
```

```
3         52
4         55
          ..
307506    26
307507    57
307508    41
307509    33
307510    46
Name: AGE_YEARS, Length: 307511, dtype: int64
```

```
new_app_df['YEARS_EMPLOYED'] = (new_app_df['DAYS_EMPLOYED'] / 365.25).round().a
new_app_df['YEARS_EMPLOYED']
```

```
0            2
1            3
2            1
3            8
4            8
          ...
307506       1
307507    1000
307508      22
307509      13
307510       3
Name: YEARS_EMPLOYED, Length: 307511, dtype: int64
```

```
#drop DAYS_BIRTH to avoid redundancy
new_app_df.drop(columns=['DAYS_BIRTH'], inplace=True)
new_app_df.drop(columns=['DAYS_EMPLOYED'], inplace=True)
```

- We will also convert 1000 (365243 in days) which is a placeholder -> nan in DAYS_EMPLOYED
- This is because 365243 is used to represent specific flag to indicate applicant is unemployed
- Since we had converted the DAYS_EMPLOYED to YEARS_EMPLOYED, we will now convert 1000 to nan

```
new_app_df['YEARS_EMPLOYED'].replace(1000, np.nan, inplace=True)
```

```
C:\Users\HP\AppData\Local\Temp\ipykernel_10208\209312331.py:1: FutureWarning: A
The behavior will change in pandas 3.0. This inplace method will never work beca

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.met


  new_app_df['YEARS_EMPLOYED'].replace(1000, np.nan, inplace=True)
```

- We will also convert CODE_GENDER 'XNA' to nan as such gender doesn't exist

```
new_app_df['CODE_GENDER'].value_counts()
```

```
CODE_GENDER
F      202448
M      105059
XNA         4
Name: count, dtype: int64
```

```
new_app_df['CODE_GENDER'] = new_app_df['CODE_GENDER'].replace('XNA',np.nan)
new_app_df['CODE_GENDER'].value_counts()
```

```
CODE_GENDER
F    202448
M    105059
Name: count, dtype: int64
```

3.4 Outlier Analysis

- Next we are going to detect and possibly remove outliers from our dataset. Outliers are values that are unusually high or low compared to the rest of the data. Below are the columns we will perform outlier analysis on
  - AMT_INCOME_TOTAL, AMT_CREDIT, AMT_ANNUITY, AMT_GOODS_PRICE, AGE_YEARS, YEARS_EMPLOYED, CNT_CHILDREN, CNT_FAM_MEMBERS, EXT_SOURCE_2, EXT_SOURCE_3
  - Firstly we will perform IQR Rule followed by visualizations using boxlpots for each column.

```
outlier_cols = [
    'AMT_INCOME_TOTAL', 'AMT_CREDIT', 'AMT_ANNUITY', 'AMT_GOODS_PRICE',
    'AGE_YEARS', 'YEARS_EMPLOYED', 'CNT_CHILDREN', 'CNT_FAM_MEMBERS',
    'EXT_SOURCE_2', 'EXT_SOURCE_3'
]

def find_outliers(df, col):
    Q1 = df[col].quantile(0.25)
    Q3 = df[col].quantile(0.75)
    IQR = Q3 - Q1
    lower = Q1 - 1.5 * IQR
    upper = Q3 + 1.5 * IQR
    outliers = df[(df[col] < lower) | (df[col] > upper)]
    return len(outliers), lower, upper

# Summary
outlier_summary = []
```

```
for col in outlier_cols:
    n_out, low, up = find_outliers(new_app_df, col)
    pct = n_out / len(new_app_df) * 100
    outlier_summary.append({
        'Column': col,
        'Outliers': n_out,
        '%': round(pct, 2),
        'Lower': round(low, 2),
        'Upper': round(up, 2)
    })

summary_df = pd.DataFrame(outlier_summary)
print(summary_df)
```

```
             Column  Outliers     %       Lower       Upper
0  AMT_INCOME_TOTAL     14035  4.56   -22500.00   337500.00
1        AMT_CREDIT      6562  2.13 -537975.00  1616625.00
2       AMT_ANNUITY      7504  2.44   -10584.00    61704.00
3   AMT_GOODS_PRICE     14728  4.79 -423000.00  1341000.00
4         AGE_YEARS         0  0.00        4.00       84.00
5    YEARS_EMPLOYED     13457  4.38       -8.50       19.50
6      CNT_CHILDREN      4272  1.39       -1.50        2.50
7   CNT_FAM_MEMBERS      4007  1.30        0.50        4.50
8      EXT_SOURCE_2         0  0.00       -0.01        1.07
9      EXT_SOURCE_3      4313  1.40        0.09        0.97
```

```
fig, axes = plt.subplots(2, 5, figsize=(18, 8))
axes = axes.ravel()

for i, col in enumerate(outlier_cols):
    sns.boxplot(data=new_app_df, y=col, ax=axes[i], color='lightblue')
    axes[i].set_title(col)
    axes[i].grid(True, alpha=0.3)

plt.tight_layout()
plt.show()
```

- Next we will perform capping which is used to handle extreme outliers in the numerical columns.
- In our dataset, we will perform capping by replacing all values that fall above the upper limit with the upper limit value itself.
- We decided to cap the outliers instead of removing them to ensure data integrity & to take into account for extreme values.
- We acknowlege that extreme values are real values that provide valuable insight in our analysis.

```
#capping
# List of columns to cap
cap_cols = [
    'AMT_INCOME_TOTAL', 'AMT_CREDIT', 'AMT_ANNUITY', 'AMT_GOODS_PRICE',
    'CNT_CHILDREN', 'CNT_FAM_MEMBERS','YEARS_EMPLOYED'
]

# Cap using IQR upper bound
for col in cap_cols:
    Q1 = new_app_df[col].quantile(0.25)
    Q3 = new_app_df[col].quantile(0.75)
    IQR = Q3 - Q1
    upper = Q3 + 1.5 * IQR
    new_app_df[col] = new_app_df[col].clip(upper=upper)
```

```
import seaborn as sns
import matplotlib.pyplot as plt

fig, axes = plt.subplots(2, 5, figsize=(18, 8))
axes = axes.ravel()

for i, col in enumerate(outlier_cols):
    sns.boxplot(data=new_app_df, y=col, ax=axes[i], color='lightblue')
    axes[i].set_title(col)
    axes[i].grid(True, alpha=0.3)

plt.tight_layout()
plt.show()
```

- From the boxplot, it can be seen that no more outliers exist as they are capped at their upper quantile values

3.5 Feature Binning

- We will also perform feature binning on some of the numerical columns to make sure that we can later used those binned columns for visualization of the default risk rate plot.
- These are the columns we will perform binning on:
  - AMT_CREDIT
  - AMT_ANNUITY
  - AMT_INCOME_TOTAL
  - AMT_GOODS_PRICE

  - AGE_YEARS
  - YEARS_EMPLOYED
  - EXT_SOURCE_2
  - EXT_SOURCE_3

AMT_CREDIT

```
# Define the number of quantiles/bins (e.g., 4 quartiles)
N_BINS = 4

# Create the new binned column
new_app_df['CREDIT_BIN_FREQ'] = pd.qcut(
    new_app_df['AMT_CREDIT'],
    q=N_BINS,
    labels=['Q1_Low', 'Q2_MidLow', 'Q3_MidHigh', 'Q4_High'],
    duplicates='drop' # Handles cases where values are identical at the cutoff
)

new_app_df['CREDIT_BIN_FREQ'].value_counts().sort_index()
```

```
CREDIT_BIN_FREQ
Q1_Low        78421
Q2_MidLow     75428
Q3_MidHigh    77786
Q4_High       75876
Name: count, dtype: int64
```

AMT_ANNUITY

```
# Define the number of quantiles/bins (e.g., 4 quartiles)
N_BINS = 4

# Create the new binned column
new_app_df['ANNUITY_BIN_FREQ'] = pd.qcut(
    new_app_df['AMT_ANNUITY'],
    q=N_BINS,
    labels=['Q1_Low', 'Q2_MidLow', 'Q3_MidHigh', 'Q4_High'],
    duplicates='drop' # Handles cases where values are identical at the cutoff
)

new_app_df['ANNUITY_BIN_FREQ'].value_counts().sort_index()
```

```
ANNUITY_BIN_FREQ
Q1_Low        76893
Q2_MidLow     76892
Q3_MidHigh    76962
```

```
Q4_High       76764
Name: count, dtype: int64
```

AMT_INCOME_TOTAL

```
# Define the number of quantiles/bins (e.g., 4 quartiles)
N_BINS = 4

# Create the new binned column
new_app_df['INCOME_BIN_FREQ'] = pd.qcut(
    new_app_df['AMT_INCOME_TOTAL'],
    q=N_BINS,
    labels=['Q1_Low', 'Q2_MidLow', 'Q3_MidHigh', 'Q4_High'],
    duplicates='drop' # Handles cases where values are identical at the cutoff
)

new_app_df['INCOME_BIN_FREQ'].value_counts().sort_index()
```

```
INCOME_BIN_FREQ
Q1_Low        100578
Q2_MidLow      53182
Q3_MidHigh     82213
Q4_High        71538
Name: count, dtype: int64
```

AMT_GOODS_PRICE

```
# Define the number of quantiles/bins (e.g., 4 quartiles)
N_BINS = 4

# Create the new binned column
new_app_df['GOODS_PRICE_BIN_FREQ'] = pd.qcut(
    new_app_df['AMT_GOODS_PRICE'],
    q=N_BINS,
    labels=['Q1_Low', 'Q2_MidLow', 'Q3_MidHigh', 'Q4_High'],
    duplicates='drop' # Handles cases where values are identical at the cutoff
)

new_app_df['GOODS_PRICE_BIN_FREQ'].value_counts().sort_index()
```

```
GOODS_PRICE_BIN_FREQ
Q1_Low        79877
Q2_MidLow     78380
Q3_MidHigh    73720
Q4_High       75534
Name: count, dtype: int64
```

AGE

```
# Define the number of quantiles/bins (e.g., 4 quartiles)
N_BINS = 4

# Create the new binned column for Age
new_app_df['AGE_BIN_FREQ'] = pd.qcut(
    new_app_df['AGE_YEARS'],
    q=N_BINS,
    labels=['Q1_Youngest', 'Q2_Young', 'Q3_MidAge', 'Q4_Oldest'],
    duplicates='drop' # Handles identical values at cutoffs
)
new_app_df['AGE_BIN_FREQ'].value_counts().sort_index()
```

```
AGE_BIN_FREQ
Q1_Youngest    80804
Q2_Young       76153
Q3_MidAge      78242
Q4_Oldest      72312
Name: count, dtype: int64
```

YEARS_EMPLOYED

```
# Define the number of quantiles/bins (e.g., 4 quartiles)
N_BINS = 4

# Create the new binned column for Years Employed
new_app_df['EMPLOYMENT_BIN_FREQ'] = pd.qcut(
    new_app_df['YEARS_EMPLOYED'],
    q=N_BINS,
    labels=['Q1_Shortest', 'Q2_Short', 'Q3_Long', 'Q4_Longest'],
    duplicates='drop'
)
new_app_df['EMPLOYMENT_BIN_FREQ'].value_counts().sort_index()
```

```
EMPLOYMENT_BIN_FREQ
Q1_Shortest    75163
Q2_Short       69813
Q3_Long        51804
Q4_Longest     55357
Name: count, dtype: int64
```

- We will first convert the EMPLOYMENT_BIN_FREQ into an object

```
# 1. FIX: Convert the column from 'category' type to generic 'object' (string)
# This removes the fixed category list constraint, allowing the new string to b
new_app_df['EMPLOYMENT_BIN_FREQ'] = new_app_df['EMPLOYMENT_BIN_FREQ'].astype(ob
print(new_app_df['EMPLOYMENT_BIN_FREQ'].isna().sum())
```

```
55374
```

- We will then add a new category which will contain the 'No Employment' values

```
new_app_df['EMPLOYMENT_BIN_FREQ'] = new_app_df['EMPLOYMENT_BIN_FREQ'].fillna('N
```

```
new_app_df['EMPLOYMENT_BIN_FREQ'].isna().sum()
```

```
np.int64(0)
```

- We will use np.where since the round method gave incorrect results.
- Using np.where, we can make sure that values that are 2.5 are converted to 3 accurately.

CNT_CHILDREN

```
new_app_df['CNT_CHILDREN'] = np.where(
    (new_app_df['CNT_CHILDREN'] > 2.49) & (new_app_df['CNT_CHILDREN'] < 2.51),
    3.0,
    new_app_df['CNT_CHILDREN']
)

# Enforce integer type now that the fix is applied
new_app_df['CNT_CHILDREN'] = new_app_df['CNT_CHILDREN'].astype(int)

print("--- Data Integrity Check (After Fixing 2.5 -> 3) ---")
print("Unique Counts in the MODIFIED original column:")
print(new_app_df['CNT_CHILDREN'].value_counts().sort_index())

##--- B. CREATE STRING TIER (FOR REPORTING/EDA) ---
##This converts the placeholder 3 into the readable string '3+ Children'.

CAP_VALUE = 3

new_app_df['CNT_CHILDREN_OBJ'] = np.where(
    new_app_df['CNT_CHILDREN'] == CAP_VALUE,
    f'{CAP_VALUE}+', # Labels the placeholder 3 as the string '3+ Children'
    new_app_df['CNT_CHILDREN'].astype(str) # Converts 0, 1, 2 to '0', '1', '2'
)
print(new_app_df['CNT_CHILDREN_OBJ'].value_counts().sort_index())
```

```
--- Data Integrity Check (After Fixing 2.5 -> 3) ---
Unique Counts in the MODIFIED original column:
```

```
CNT_CHILDREN
0    215371
1     61119
2     26749
3      4272
Name: count, dtype: int64
CNT_CHILDREN_OBJ
0     215371
1      61119
2      26749
3+      4272
Name: count, dtype: int64
```

CNT_FAM_MEMBERS

```
# 1. FIX CORRUPTION (4.5 -> 5.0)
# We assume a similar corruption exists for family members (e.g., 4.5) and fix
new_app_df['CNT_FAM_MEMBERS'] = np.where(
    (new_app_df['CNT_FAM_MEMBERS'] > 4.49) & (new_app_df['CNT_FAM_MEMBERS'] < 4
    5.0, # Fix 4.5 to 5.0
    new_app_df['CNT_FAM_MEMBERS']
)
new_app_df['CNT_FAM_MEMBERS'] = new_app_df['CNT_FAM_MEMBERS'].astype(int)

CAP_MEMBERS = 5

# 2. CREATE FINAL STRING TIER (Capping Logic)
# We cap all counts >= 5 into the '5+ Members' group.
new_app_df['CNT_FAM_MEMBERS_OBJ'] = np.where(
    new_app_df['CNT_FAM_MEMBERS'] >= CAP_MEMBERS,
    f'{CAP_MEMBERS}+', # Labels 5, 6, 7, etc. as '5+ Members'
    new_app_df['CNT_FAM_MEMBERS'].astype(str) # Converts 1, 2, 3, 4 to strings
)

new_app_df['CNT_FAM_MEMBERS_OBJ'].value_counts().sort_index()
```

```
CNT_FAM_MEMBERS_OBJ
1      67847
2     158359
3      52601
4      24697
5+      4007
Name: count, dtype: int64
```

EXT_SOURCE_2

```
# Define the number of quantiles/bins (e.g., 4 quartiles)
N_BINS = 4
```

```
# Create the new binned column for Age
new_app_df['EXT_SOURCE_2_BIN_FREQ'] = pd.qcut(
    new_app_df['EXT_SOURCE_2'],
    q=N_BINS,
    # Optional: name your categories
    labels=['Q1_Low', 'Q2_MidLow', 'Q3_MidHigh', 'Q4_High'],
    duplicates='drop' # Handles identical values at cutoffs
)
new_app_df['EXT_SOURCE_2_BIN_FREQ'].value_counts().sort_index()
```

```
EXT_SOURCE_2_BIN_FREQ
Q1_Low        76879
Q2_MidLow     77207
Q3_MidHigh    76548
Q4_High       76877
Name: count, dtype: int64
```

EXT_SOURCE_3

```
# Define the number of quantiles/bins (e.g., 4 quartiles)
N_BINS = 4

# Create the new binned column for Age
new_app_df['EXT_SOURCE_3_BIN_FREQ'] = pd.qcut(
    new_app_df['EXT_SOURCE_3'],
    q=N_BINS,
    labels=['Q1_Low', 'Q2_MidLow', 'Q3_MidHigh', 'Q4_High'],
    duplicates='drop' # Handles identical values at cutoffs
)
new_app_df['EXT_SOURCE_3_BIN_FREQ'].value_counts().sort_index()
```

```
EXT_SOURCE_3_BIN_FREQ
Q1_Low         77497
Q2_MidLow     106845
Q3_MidHigh     47086
Q4_High        76083
Name: count, dtype: int64
```

4. Data Analysis

- Next we will perform segmented univariate analysis using segmented univariate, bivariate & multivariate analysis.

4.1 Segmented Univariate Analysis

Univariate Categorical Analysis

- We will perform this analysis on the categorial columns below:
    - NAME_CONTRACT_TYPE, CODE_GENDER, OCCUPATION_TYPE, NAME_INCOME_TYPE, NAME_EDUCATION_TYPE, NAME_FAMILY_STATUS, NAME_HOUSING_TYPE, AGE_GROUP,
    - In order to determine the % of the group that is most likely to default, we will use the default risk plot which utilises the formula below: Default Rate (Risk) = Count of Target (1) / (Count of Target (1) + Count of Target (0))

```python
#function to plot univariate categorical variables
def cat_plot(df, col_x):
    plt.figure(figsize=(10,5))
    # create the count plot
    ax = sns.countplot(data=df, x=col_x, hue='TARGET')

    # create the title
    plt.title(f"Target Distribution by {col_x}")
    plt.xticks(rotation=45, ha='right')
    #show the plot
    plt.show()
```

```python
# 1. Calculate the Default Risk Rate (Mean of TARGET) for each category
# The mean of a binary column (0 or 1) is the proportion of 1s (defaulters)

def univariate_decision_support_risk_plot(df, col_x):
   default_rate_pct = df.groupby(col_x)['TARGET'].mean()

   # 2. Convert to percentage and round to 2 decimal places
   default_rate_pct = (default_rate_pct * 100).round(2)

   # 3. Sort the categories by risk rate (highest risk first)
   default_rate_pct = default_rate_pct.sort_values(ascending=False)

   # 4. Prepare data for plotting (reset index to make columns)
   plot_data = default_rate_pct.reset_index()
   plot_data.columns = [col_x, 'Default Risk Rate (%)']

   # Plot using pointplot (as requested)
   plt.figure(figsize=(10, 5))
   sns.barplot(x=col_x, y='Default Risk Rate (%)', data=plot_data)
   plt.xticks(rotation=45, ha='right')
   plt.xlabel(col_x)
   plt.ylabel("Default Risk Rate (%)")
```

```
        plt.title(f"Default Risk Rate by {col_x}")
        plt.show()
```

NAME_CONTRACT_TYPE

```
cat_plot(new_app_df,'NAME_CONTRACT_TYPE')
```

- The number of deefaulters & those paying on time is higher for cash loans as compared to revolving loans.

```
univariate_decision_support_risk_plot(new_app_df,'NAME_CONTRACT_TYPE')
```

- Revolving loans appear to have a lower default risk rate as compared to cash loans.

CODE_GENDER

```
cat_plot(new_app_df,'CODE_GENDER')
```

- The number of defaulters & those paying on time is higher for females as compared to males.

```
univariate_decision_support_risk_plot(new_app_df,'CODE_GENDER')
```

- Males appear to have a higher default risk rate as compared to females.

OCCUPATION_TYPE

```
cat_plot(new_app_df,'OCCUPATION_TYPE')
```

- The number of defaulters and those paying on time is the highest for unknown.

```
univariate_decision_support_risk_plot(new_app_df,'OCCUPATION_TYPE')
```

- Low skill Laborers appear to have the highest default risk rate.
- Accountants have the lowest default risk rate.

NAME_INCOME_TYPE

```
cat_plot(new_app_df,'NAME_INCOME_TYPE')
```

- Working people have the highest defaulters and those paying on time.

```
univariate_decision_support_risk_plot(new_app_df,'NAME_INCOME_TYPE')
```

- People with maternity leave appears to have highest default risk rate followed closely by unemployed.
- Businessman and student seem to have a non-existant default risk rate.
- Pensioners have the lowest default risk rate.

NAME_EDUCATION_TYPE

```
cat_plot(new_app_df,'NAME_EDUCATION_TYPE')
```

- The number of defaulters and those paying on time is the highest for secondary/secondary special.

```
univariate_decision_support_risk_plot(new_app_df,'NAME_EDUCATION_TYPE')
```

- People with lower secondary appear to have the highest default risk rate.
- People with academic degree appear to have the lowest default risk rate.

NAME_FAMILY_STATUS

```
cat_plot(new_app_df,'NAME_FAMILY_STATUS')
```

- The number of defaulters and those paying on time is the highest for married.

```
univariate_decision_support_risk_plot(new_app_df,'NAME_FAMILY_STATUS')
```

- People in civil marriage appear to have the highest default risk rate followed very closely by single/not married.
- The default risk rate for unknown is non-existant.

NAME_HOUSING_TYPE

```
cat_plot(new_app_df,'NAME_HOUSING_TYPE')
```

- Those staying in house/apartment has the highest number of defaulters and those paying on time.

```
univariate_decision_support_risk_plot(new_app_df,'NAME_HOUSING_TYPE')
```

- People with rented apartment have the highest default risk rate.
- People with office apartment has the lowest default risk rate.

CNT_CHILDREN

```
cat_plot(new_app_df,'CNT_CHILDREN_OBJ')
```

- As the number of children increases, the number of defaulters and repayers on time decreases.

```
univariate_decision_support_risk_plot(new_app_df,'CNT_CHILDREN_OBJ')
```

- The default risk rate is the highest for 3+ children.
- The default risk rate is the lowest for 0 children.

CNT_FAM_MEMBERS

```
cat_plot(new_app_df,'CNT_FAM_MEMBERS_OBJ')
```

- The number of defaulters & payment on time is the highest for 2 family members.

```
univariate_decision_support_risk_plot(new_app_df,'CNT_FAM_MEMBERS_OBJ')
```

- The default risk rate is the highest for 5+ members.
- The default risk rate is the lowest for 2 members.

FLAG_OWN_CAR

```
cat_plot(new_app_df,'FLAG_OWN_CAR')
```

- The number of defaulters and payment on time is the highest for those not owning a car.

```
univariate_decision_support_risk_plot(new_app_df,'FLAG_OWN_CAR')
```

- The default risk rate is higher for those not owning a car.

FLAG_OWN_REALTY

```
cat_plot(new_app_df,'FLAG_OWN_REALTY')
```

- The number of defaulters and payment on time is higher for those owning an asset.

```
univariate_decision_support_risk_plot(new_app_df,'FLAG_OWN_REALTY')
```

- The default risk rate is higher for those not owning an asset.

REGION_RATING_CLIENT

```
cat_plot(new_app_df,'REGION_RATING_CLIENT')
```

- Region rated as 2 has the highest defaulters and payment on time.

```
univariate_decision_support_risk_plot(new_app_df,'REGION_RATING_CLIENT')
```

- The default risk rate is highest for the region rated as 3.
- The default risk rate is the lowest for the region rated as 1.

Univariate Numerical Analysis

- We will perform this analysis on the numerical columns below:
  - CNT_CHILDREN, AMT_INCOME_TOTAL, AMT_CREDIT, AMT_ANNUITY, AMT_GOODS_PRICE, CNT_FAM_MEMBERS, AGE_YEARS, YEARS_EMPLOYED, FLAG_OWN_CAR, FLAG_OWN_REALTY
  - The higher the blue peak relative to its red peak, the higher the risk density & vice versa.
  - Once we find the risk density, we will plot the default risk rate graph.

```
df0 = new_app_df[new_app_df['TARGET'] == 0]
df1 = new_app_df[new_app_df['TARGET'] == 1]
```

```
# function to plot numerical variables
def cont_plot(df, col_x):
    plt.figure(figsize=(10,6))

    # 1. Use plt.gca() to get the current Axes object for the first plot
    #    and assign it to 'ax'.
    ax = sns.kdeplot(
        df0[col_x],
        label="Target 0 (No Default)",
        color="Red",
        fill=True
    )

    # 2. Plot the second distribution, referencing the same axes object 'ax'
    #    to ensure they are overlaid correctly.
    ax = sns.kdeplot(
        df1[col_x],
        label="Target 1 (Default)",
        color="Blue",
        ax=ax,
        fill=True
    )

    # 3. Create the title and labels
    plt.title(f"Target Distribution by {col_x}")
    plt.xlabel(col_x)
    plt.ylabel("Density")
    plt.legend()

    # 4. Show the plot
    plt.show()
```

AMT_INCOME_TOTAL

```
cont_plot(new_app_df,'AMT_INCOME_TOTAL')
```

- People with income range around 50000 has the highest risk density.

```
univariate_decision_support_risk_plot(new_app_df,'INCOME_BIN_FREQ')
```

- The default risk rate is the highest for Q2_MidLow.
- The default risk rate is the lowest for Q4_High.

AMT_CREDIT

```
cont_plot(new_app_df,'AMT_CREDIT')
```

- People with AMT_CREDIT around 0.5 has the highest risk density.
- People with AMT_CREDIT around 1.6 has the lowest risk density.

```
univariate_decision_support_risk_plot(new_app_df,'CREDIT_BIN_FREQ')
```

- The highest default risk rate is the Q2_MidLow.
- The lowest default risk rate is th Q4_High.

AMT_ANNUITY

```
cont_plot(new_app_df,'AMT_ANNUITY')
```

- People with AMT_ANNUITY between 18000 to 40000 have the highest risk density.
- People with AMT_ANNUITY >60000 have the lowest risk density.

```
univariate_decision_support_risk_plot(new_app_df,'ANNUITY_BIN_FREQ')
```

- Q3_MidHigh has the highest default risk rate.
- Q1_Low has the lowest default risk rate.

AMT_GOODS_PRICE

```
cont_plot(new_app_df,'AMT_GOODS_PRICE')
```

- People with AMT_GOODS_PRICE between 0.4 to 0.6 have the highest risk density.

```
univariate_decision_support_risk_plot(new_app_df,'GOODS_PRICE_BIN_FREQ')
```

- Q2_MidLow has the highest default risk rate.
- Q4_High has the lowest default risk rate.

AGE_YEARS

```
cont_plot(new_app_df,'AGE_YEARS')
```

- People with age between 30 to 40 have the highest risk density.

```
univariate_decision_support_risk_plot(new_app_df,'AGE_BIN_FREQ')
```

- Q1_Youngest has the highest default risk rate.
- Q4_Oldest has the lowest default risk rate.

YEARS_EMPLOYED

```
cont_plot(new_app_df,'YEARS_EMPLOYED')
```

- People employed between 0 to 5 years have the highest risk density.

```
univariate_decision_support_risk_plot(new_app_df,'EMPLOYMENT_BIN_FREQ')
```

- Q1_Shortest has the highest default risk rate.
- Q4_Longest has the lowest default risk rate.

EXT_SOURCE_2

```
cont_plot(new_app_df,'EXT_SOURCE_2')
```

- The risk density is highest between 0.0 to 0.5.

```
univariate_decision_support_risk_plot(new_app_df,'EXT_SOURCE_2_BIN_FREQ')
```

- The default risk rate is the highest for Q1_Low.
- The default risk rate is the lowest for Q4_High.

EXT_SOURCE_3

```
cont_plot(new_app_df,'EXT_SOURCE_3')
```

- The risk density is the highest between 0.0 to 0.5.

```
univariate_decision_support_risk_plot(new_app_df,'EXT_SOURCE_3_BIN_FREQ')
```

- The default risk rate is the highest for Q1_Low.
- The default risk rate is the lowest for Q4_High.

Univariate Numerical Discrete Analysis

4.2 Bivariate Analysis

- We will perform 3 types of bivariate analysis.
  - Categorical-categorical columns
  - Categorical-continuous columns

- Continuos-continuos columns

Categorical-categorical columns

- These are the pairings used
    - NAME_EDUCATION_TYPE - NAME_INCOME_TYPE
    - NAME_FAMILY_STATUS - CODE_GENDER
    - NAME_HOUSING_TYPE - NAME_CONTRACT_TYPE
    - NAME_INCOME_TYPE - OCCUPATION_TYPE
    - FLAG_OWN_CAR - CNT_CHILDREN

```
df0 = new_app_df[new_app_df['TARGET']==0]
df1 = new_app_df[new_app_df['TARGET']==1]
```

```
# function to plot categorical,categorical
def cat_cat_plot(df,col_x,col_hue):

    num_hues = df[col_hue].nunique(dropna=False)

    # 2. Generate the dynamic, high-contrast palette
    if num_hues <= 10:
        # Use 'Set1' for up to 9 categories for maximum contrast
        dynamic_palette = 'Set1'
    elif num_hues <= 20:
        # Use 'tab20' for up to 20 categories
        dynamic_palette = sns.color_palette("tab20", n_colors=num_hues)
    elif num_hues > 20:
        # If there are more than 20, we use a different qualitative map like 'h
        # WARNING: If num_hues > 20, consider re-binning var2 as visualization
        dynamic_palette = sns.color_palette("hls", n_colors=num_hues)


    plt.figure(figsize=(15,8))
    # --- Subplot 1: DEFAULTERS (df1) ---
    plt.subplot(1, 2, 1)
    # Pass the column names as strings (var1, var2) to sns.countplot
    ax1 = sns.countplot(data=df1, x=col_x, hue=col_hue, palette=dynamic_palette

    # Pass the column names as strings for the title and labels
    plt.title(f"Distribution of {col_x} by {col_hue} for DEFAULTERS")
    plt.xlabel(col_x)
    plt.ylabel("Count")
    plt.xticks(rotation=45, ha='right')
    plt.legend(title=col_hue) # Added legend

    # --- Subplot 2: NON-DEFAULTERS (df0) ---
    plt.subplot(1, 2, 2)
    ax2 = sns.countplot(data=df0, x=col_x, hue=col_hue, palette=dynamic_palette

    plt.title(f"Distribution of {col_x} by {col_hue} for Others")
    plt.xlabel(col_x)
    # Set ylabel to blank or remove it, as it's the same as the first plot's Y-
    plt.ylabel("Count")
    plt.xticks(rotation=45, ha='right')
    plt.legend(title=col_hue) # Added legend

    # Crucial for preventing subplot titles and labels from overlapping
    plt.tight_layout()
    plt.show()
```

```
def bivariate_decision_support_risk_plot(df, col_x, col_hue):
    # 1. Group by both columns and calculate the mean of TARGET (the Default Ra
    default_rate_pct = df.groupby([col_x, col_hue])['TARGET'].mean()
    default_rate_pct = (default_rate_pct * 100).round(2)

    plot_data = default_rate_pct.reset_index()
    plot_data.columns = [col_x, col_hue, 'Default Risk Rate (%)']

    num_hues = df[col_hue].nunique(dropna=False)

    # 2. Generate the dynamic, high-contrast palette
    if num_hues <= 10:
        # Use 'Set1' for up to 9 categories for maximum contrast
        dynamic_palette = 'Set1'
    elif num_hues <= 20:
        # Use 'tab20' for up to 20 categories
        dynamic_palette = sns.color_palette("tab20", n_colors=num_hues)
    elif num_hues > 20:
        # If there are more than 20, we use a different qualitative map like 'h
        # WARNING: If num_hues > 20, consider re-binning var2 as visualization
        dynamic_palette = sns.color_palette("hls", n_colors=num_hues)


    plt.figure(figsize=(15,8))

    # 3. Plot the data using the default column names:
    ax = sns.barplot(
        data=plot_data,
        x=col_x,
        y='Default Risk Rate (%)',
        hue=col_hue,
        palette=dynamic_palette
    )

    plt.title(f"Default Risk Rate (%) for {col_x} by {col_hue}")
    plt.xlabel(col_x)
```

```
        plt.ylabel("Default Rate Percentage")
        # You must manually rename the legend title if you don't use .columns
        plt.legend(title=col_hue)
        plt.xticks(rotation=45, ha='right')
        plt.tight_layout()
        plt.show()
```

NAME_EDUCATION_TYPE - NAME_INCOME_TYPE

```
cat_cat_plot(new_app_df,'NAME_EDUCATION_TYPE', 'NAME_INCOME_TYPE')
```

- People with secondary/secondary special has the highest number of defaulters and payment on time.

```
bivariate_decision_support_risk_plot(new_app_df,'NAME_EDUCATION_TYPE', 'NAME_IN
```

- Maternity leave income type within secondary/secondary special have the highest default risk rate.
- People with academic degree have a very low default risk rate.

NAME_FAMILY_STATUS - CODE_GENDER

```
cat_cat_plot(new_app_df,'NAME_FAMILY_STATUS','CODE_GENDER')
```

- In all categories other than Single/not married, there are more females than males who have defaulted.
- Females have made payment on time for all categories.

```
bivariate_decision_support_risk_plot(new_app_df, 'NAME_FAMILY_STATUS', 'CODE_GE
```

- Males have a higher default risk rate across all family status.
- Among all females within each family status group, widow has the lowest default risk rate.

NAME_HOUSING_TYPE - NAME_CONTRACT_STATUS

```
cat_cat_plot(new_app_df,'NAME_HOUSING_TYPE','NAME_CONTRACT_TYPE')
```

- Cash loans have higher number of defaulters & payment on time than revolving loans.
- Cash loans for house/apartment has the highest number of defaulters & payment on time.

```
bivariate_decision_support_risk_plot(new_app_df,'NAME_HOUSING_TYPE', 'NAME_CONT
```

- Cash loans have a higher default risk rate compared to revolving loans for all housing types.
- Cash loans for rented apartment has the highest default risk rate followed closely by cash loans with parents.
- Revolving loans for municipal apartment has the lowest default risk rate.

NAME_INCOME_TYPE - OCCUPATION_TYPE

```
cat_cat_plot(new_app_df,'NAME_INCOME_TYPE','OCCUPATION_TYPE')
```

- Working laborers have the highest number of defaulters.
- Pensioners have the highest number of people paying on time.

```
bivariate_decision_support_risk_plot(new_app_df,'NAME_INCOME_TYPE', 'OCCUPATION
```

- Maternity leave laborers has the highest default risk rate followed by maternity leave core staff.
- Pensioners have among the lowest default rates, highest likelihood of paying on time.
- The default rate for businessman & student is non-existant.

FLAG_OWN_CAR - CNT_CHILDREN

```
cat_cat_plot(new_app_df, 'FLAG_OWN_CAR','CNT_CHILDREN_OBJ')
```

- The number of defaulters and payment on time is the highest for those with 0 children for both graphs.
- Those with 3 children has the lowest defaulters for both car categories.

```
bivariate_decision_support_risk_plot(new_app_df,'FLAG_OWN_CAR', 'CNT_CHILDREN_C
```

- The default risk rate is the the highest for those with 3+ children regardless of whether the person owns a car or not.
- The default risk rate is the lowest for those with 0 children regardless of whether the person owns a car or not.

FLAG_OWN_REALTY - CNT_FAM_MEMBERS

```
cat_cat_plot(new_app_df, 'FLAG_OWN_REALTY','CNT_FAM_MEMBERS_OBJ')
```

- The number of defaulters and payment on time is the highest for 2 family members regardless if the person owns an asset or not.

```
bivariate_decision_support_risk_plot(new_app_df,'FLAG_OWN_REALTY','CNT_FAM_MEME
```

- The default risk rate is the highest for 5+ family members regardless of whether the person own an asset or not.
- The default risk rate is the lowest for 2 family members regardless of whether the person own an asset or not.

Categorical-continuos columns

- NAME_INCOME_TYPE - AMT_CREDIT
- NAME_EDUCATION_TYPE - AMT_ANNUITY
- OCCUPATION_TYPE - AMT_INCOME_TOTAL
- NAME_HOUSING_TYPE - AMT_GOODS_PRICE
- FLAG_OWN_REALTY - EXT_SOURCE_2

```
# function to plot categorical,contuinious
def cat_cont_plot(df, var1, var2):

    plt.figure(figsize=(15,8))

    # --- Subplot 1: DEFAULTERS (df1) ---
    plt.subplot(1, 2, 1)
    # Pass the column names as strings (var1, var2) to sns.countplot
    ax1 = sns.boxplot(data=df1, x=var1, y=var2, showfliers=False)
    # Pass the column names as strings for the title and labels
    plt.title(f"Distribution of {var1} by {var2} for Defaulters")
    plt.xlabel(var1)
    plt.xticks(rotation=45, ha='right')
    plt.ylabel(var2)

    # --- Subplot 2: NON-DEFAULTERS (df0) ---
    plt.subplot(1, 2, 2)
    ax2 = sns.boxplot(data=df0, x=var1, y=var2, showfliers=False)
    plt.title(f"Distribution of {var1} by {var2} for Others")
    plt.xlabel(var1)
    plt.xticks(rotation=45, ha='right')
    plt.ylabel(var2)

    # Crucial for preventing subplot titles and labels from overlapping
    plt.tight_layout()
    plt.show()
```

NAME_INCOME_TYPE - AMT_CREDIT

```
cat_cont_plot(new_app_df,'NAME_INCOME_TYPE','AMT_CREDIT')
```

- The median of maternity leave and unemployed is higher for defaulters as compared to others.
- The median of businessman is the highest among all repayment groups

```
bivariate_decision_support_risk_plot(new_app_df,'NAME_INCOME_TYPE','CREDIT_BIN_
```

- Unemployed income type within the Q2_MidLow has the highest default risk rate.
- The default rate for businessman is non-existant.
- Pensioners have the lowest default rate almost all income categories.

NAME_EDUCATION_TYPE - AMT_ANNUITY

```
cat_cont_plot(new_app_df,'NAME_EDUCATION_TYPE','AMT_ANNUITY')
```

- The median is somewhat similar across almost all education types.
- The median of academic degree is higher for defaulters as compared to others.

```
bivariate_decision_support_risk_plot(new_app_df,'NAME_EDUCATION_TYPE','ANNUITY_
```

- Lower secondary with Q3_MidHigh seems to have the highest default risk rate.
- Academic degree seems to have lowest default risk rate for the columns visible while some of the columns are non-existant.
- Lower secondary also has the highest default risk rate across all the annuity categories.

OCCUPATION_TYPE - AMT_INCOME_TOTAL

```
cat_cont_plot(new_app_df,'OCCUPATION_TYPE','AMT_INCOME_TOTAL')
```

- The median is somewhat similar between most of the occupation types.
- All boxes (IQR) overlap each other.
- The median of IT Staff for defaulters is lower as compared to in others.
- The median of HR staff is higher for defaulters as compared to others.

```
bivariate_decision_support_risk_plot(new_app_df,'OCCUPATION_TYPE','INCOME_BIN_F
```

- Low-skill laborers within Q1_Low has the highest default risk rate.
- Low-skill laborers also have the highest default risk rate for each categories.
- Accoutants have the lowest default risk rate for each of the categories.

HOUSING_TYPE - AMT_GOODS_PRICE

```
cat_cont_plot(new_app_df,'NAME_HOUSING_TYPE','AMT_GOODS_PRICE')
```

- The median of most housing types is somewhat similar.
- The boxes (IQR) overlap each other.
- The median of Co-op apartment in defaulters is slightly lower as compared to in others.

```
bivariate_decision_support_risk_plot(new_app_df,'NAME_HOUSING_TYPE','GOODS_PRIC
```

- Rented apartment within Q2_MidLow has the highest default risk rate followed very closely by staying with parents within Q2_MidLow.
- Rented apartment has the highest default risk rate across all goods price categories.
- Office apartment has the lowest default risk rate across almost all goods price categories.

FLAG_OWN_REALTY - EXT_SOURCE_2

```
cat_cont_plot(new_app_df,'FLAG_OWN_REALTY','EXT_SOURCE_2')
```

- The median of FLAG_OWN_REALTY in defaulters is lower as compared to the median in others.

```
bivariate_decision_support_risk_plot(new_app_df,'FLAG_OWN_REALTY','EXT_SOURCE_2
```

- The default risk rate is the highest for Q1_Low regardless of whether the person owns an asset or does not.
- The default risk rate is the lowest for Q4_High in both cases.

CNT_FAM_MEMBERS - EXT_SOURCE_3

```
cat_cont_plot(new_app_df,'CNT_FAM_MEMBERS_OBJ','EXT_SOURCE_3')
```

- The median for cnt_fam_members is lower for defautlers compared to others.

```
bivariate_decision_support_risk_plot(new_app_df,'CNT_FAM_MEMBERS_OBJ','EXT_SOUR
```

- 5+ members within Q1_Low has the highest default risk rate.
- Those within Q4_High has the lowest default risk rate regardless of the number of family members.

Multivariate Analysis

- We will perform 2 types of multivariate analysis:
  - Categorical-categorical-categorical columns
  - Categorical-categorical-continous columns

Categorical-categorical-categorical

```
def cat_cat_cat_plot(df, col_x, col_hue, col_facet, target_col='TARGET'):
    # The sns.catplot command handles the segmentation by Target and the third

    num_hues = df[col_hue].nunique(dropna=False)

    # 2. Generate the dynamic, high-contrast palette based on hue column
    if num_hues <= 10:
        dynamic_palette = 'Set1'
    elif num_hues <= 20:
        dynamic_palette = sns.color_palette("tab20", n_colors=num_hues)
    else:
        # Use 'hls' for a larger number, but note visualization will be clutter
        dynamic_palette = sns.color_palette("hls", n_colors=num_hues)

    # 2. Use Seaborn's catplot for efficient faceting and grouping
    g = sns.catplot(
        data=df, # Use the entire DataFrame (no need for df0/df1 split)
        x=col_x,
        hue=col_hue,
        col=target_col,  # Separates Defaulters (1) vs Non-Defaulters (0) side-
        row=col_facet,  # CREATES A NEW ROW OF PLOTS FOR EACH UNIQUE VALUE OF T
        kind="count",
        aspect=2.0,  # Stretches the plot horizontally
        height=4,
        palette=dynamic_palette,
        sharey=True,  # All plots share the same Y-axis scale (Count)
        legend_out=True
    )

    # 3. Clean up the plot appearance
    g.set_titles(row_template='{row_name} Group', col_template='Target = {col_r
    g.set_axis_labels(col_x, "Count of Applications")

     # FIX: Explicitly iterate over the axes to set x-tick labels using the unc
    for ax in g.axes.flat:
        # CRITICAL FIX: Force Matplotlib to draw the X-axis labels on this spec
        ax.tick_params(labelbottom=True)

        # FIX for AttributeError: 'list' object has no attribute 'size'. Use le
        if len(ax.get_xticks()) > 0:
            # Get the existing labels (the category names)
            labels = [t.get_text() for t in ax.get_xticklabels()]
            # Re-set them with the rotation and alignment
            ax.set_xticklabels(labels, rotation=45, ha='right')

     # g.add_legend(title=col_hue, bbox_to_anchor=(1.02, 0.5), loc='center left

    plt.suptitle(
        f"Multivariate Count Analysis by {col_x}, {col_hue}, and {col_facet}",
        y=1.02, # Adjust title position
        fontsize=16
    )

    # FIX: Increase bottom margin to ensure rotated X-axis labels are not cut c
    g.fig.subplots_adjust(bottom=0.25, right=0.85, hspace=0.7)
    plt.show()
```

```
    def multivariate_decision_support_risk_plot(df, col_x, col_hue, col_facet, targ
        # 1. Group by all three columns and calculate the mean of TARGET (the Defau
        default_rate_pct = df.groupby([col_x, col_hue, col_facet])[target_col].mear
        # Convert mean (e.g., 0.08) to percentage (e.g., 8.0) and reset index for p
        plot_data = (default_rate_pct * 100).round(2).reset_index()
        plot_data.columns = [col_x, col_hue, col_facet, 'Default Risk Rate (%)']

        num_hues = df[col_hue].nunique(dropna=False)

        # 2. Generate the dynamic, high-contrast palette based on hue column
        if num_hues <= 10:
            dynamic_palette = 'Set1'
        elif num_hues <= 20:
            dynamic_palette = sns.color_palette("tab20", n_colors=num_hues)
        else:
            # Use 'hls' for a larger number, but note visualization will be clutter
            dynamic_palette = sns.color_palette("hls", n_colors=num_hues)

        # 3. Use catplot to create a faceted bar plot
        g = sns.catplot(
            data=plot_data,
            x=col_x,
            y='Default Risk Rate (%)', # Plot the pre-calculated percentage
            hue=col_hue,
            col=col_facet,  # The third variable creates the facets (sub-plots)
            kind="bar",
            col_wrap=2,
            aspect=1.5,
            height=6,
            palette=dynamic_palette,
            legend_out=True
        )

        # 4. Clean up the plot appearance
        g.set_axis_labels(col_x, "Default Risk Rate Percentage")
        g.set_titles(col_template='{col_name} Group')

         # FIX: Explicitly iterate over the axes to set x-tick labels using the unc
        for ax in g.axes.flat:
            # CRITICAL FIX: Force Matplotlib to draw the X-axis labels on this spec
            ax.tick_params(labelbottom=True)

            # FIX for AttributeError: 'list' object has no attribute 'size'. Use le
            if len(ax.get_xticks()) > 0:
                # Get the existing labels (the category names)
                labels = [t.get_text() for t in ax.get_xticklabels()]
                # Re-set them with the rotation and alignment
                ax.set_xticklabels(labels, rotation=45, ha='right')

        # g.add_legend(title=col_hue, bbox_to_anchor=(1.02, 0.5), loc='center left'

        plt.suptitle(
            f"Multivariate Default Rate (%) by {col_x}, {col_hue}, and {col_facet}'
            y=1.02,
            fontsize=16
        )
        # 6. Keep increased bottom margin and vertical spacing (hspace) between row
        g.fig.subplots_adjust(bottom=0.30, right=0.85, hspace=0.75)
        plt.show()
```

NAME_INCOME_TYPE - FLAG_OWN_REALTY - NAME_EDUCATION_TYPE

```
cat_cat_cat_plot(new_app_df,'NAME_INCOME_TYPE','FLAG_OWN_REALTY','NAME_EDUCATIC
```

- For the secondary/secondary special group, the number of payment on time are higher for those owning an asset as compared to those not owning an asset.
- For the secondary/secondary special group, the number of defaulters are higher for those owning an asset as compared to those not owning an asset.

```
multivariate_decision_support_risk_plot(new_app_df,'NAME_INCOME_TYPE','FLAG_OWN
```

- For incomplete higher group, the default risk rate is the highest for unemployed & not owning an asset.
- For incomplete higher group, the default risk rate is higher for those not owning an asset across all income categories.
- For secondary/secondary special group, the default risk rate is the highest for maternity leave & owning an asset.
- For secondary/secondary special group, unemployed & owning an asset has a higher default risk rate as compared to not owning an asset.
- The default risk rate for businessman is non-existant across all income categories.

CNT_CHILDREN - FLAG_OWN_CAR - NAME_EDUCATION_TYPE

```
cat_cat_cat_plot(new_app_df,'CNT_CHILDREN_OBJ','FLAG_OWN_CAR','NAME_EDUCATION_T
```

- For secondary/secondary special, the number of payment on time & defaulters is the highest for 0 children.

- For higher education, the number of defaulters and payment on time is the highest for 0 children.

```
multivariate_decision_support_risk_plot(new_app_df,'CNT_CHILDREN_OBJ','FLAG_OWN
```

- For academic degree, the default risk rate is higher for 1 children & not owning a car as compared to owning a car.
- For higher education & incomplete higher group, the default risk rate is higher for not owning a car as compared to owning a car for all children categories.
- For lower secondary group, the default risk rate is higher for owning a car as compared to not owning a car for 1 & 3+ children groups.
- For secondary/secondary special group, the default risk rate is higher for not owning a car as compared to owning a car for all children groups.

Categorical - cateogrical - continuos

```
def cat_cat_cont_plot(df, col_x, col_hue, col_y, target_col='TARGET'):
    # 1. Use catplot with kind="box" and set 'col' to target_col for faceting
    g = sns.catplot(
        data=df,
        x=col_x,
        y=col_y,
        hue=col_hue,
        col=target_col,    # CRITICAL CHANGE: Facet by TARGET (0 and 1)
        kind="box",
        col_wrap=2,        # Ensures the two TARGET plots sit side-by-side
        aspect=1.5,
        height=6,
        legend_out=True,
        sharey=True        # Ensure both plots share the same Y-axis scale
    )

    # 2. Clean up the plot appearance
    g.set_axis_labels(col_x, f"Distribution of {col_y}")
    g.set_titles(col_template='Target = {col_name}') # Label facets as Target =

    # Set X-tick labels
    for ax in g.axes.flat:
        ax.tick_params(labelbottom=True)
        if len(ax.get_xticks()) > 0:
            labels = [t.get_text() for t in ax.get_xticklabels()]
            ax.set_xticklabels(labels, rotation=45, ha='right')
    # Add legend and title
    #g.add_legend(title=col_hue, bbox_to_anchor=(1.02, 0.5), loc='center left')
    plt.suptitle(
        f"Distribution of {col_y} by {col_x}, {col_hue}, and {target_col}",
        y=1.02,
        fontsize=16
    )
    g.fig.subplots_adjust(bottom=0.30, right=0.85)
    plt.show()
```

NAME_INCOME_TYPE - FLAG_OWN_REALTY - AMT_CREDIT

```
cat_cat_cont_plot(new_app_df,'NAME_INCOME_TYPE','FLAG_OWN_REALTY','AMT_CREDIT')
```

- The median for unemployed regardless if they own an asset or not is higher as compared to unemployed.
- The median for maternity leave & owning an asset for payment on time is lower compared to those defaulting.

```
multivariate_decision_support_risk_plot(new_app_df,'NAME_INCOME_TYPE','FLAG_OWN
```

- For Q1_Low Group, the default risk rate for working & not owning an asset is higher as compared to owning an asset.
- For Q2_MidLow Group, the default risk rate for unemployed, state servant, maternity leave & commercial associate while owning an asset is higher as compared to not owning an asset.
- For Q3_MidHigh Group, the default risk rate for unemployed & owning an asset is higher as compared to not owning an asset.
- For Q4_High Group, the default risk rate for commercial associate & working while owning an asset is higher as compared to not owning an asset.
- The default risk rate for businessman is non-existant across all groups.

NAME_EDUCATION_TYPE - FLAG_OWN_CAR - AMT_ANNUITY

```
cat_cat_cont_plot(new_app_df,'NAME_EDUCATION_TYPE','FLAG_OWN_CAR','AMT_ANNUITY'
```

- The median is somewhat similar.

```
multivariate_decision_support_risk_plot(new_app_df,'NAME_EDUCATION_TYPE','FLAG_
```

- For Q1_Low Group, the default risk rate is higher for higher education. incomplete higher, lower secondary & secondary/secondary special while not owning a car.
- For Q2_MidLow Group, the default risk rate is higher for higher education,incomplete higher,secondary/secondary special while not owning a car as compared to owning a car.
- For Q3_MidHigh Group, the default risk rate is higher for higher education,incomplete higher,lower secondary, secondary/secondary special while not owning a car compared to owning a car.
- For Q4_High Group, the default risk rate is higher for academic degree,higher education,lower secondary, secondary/secondary special while not owning a car compared to owning a car.

Continous-continuos columns

- The below colums are considered.
  - AMT_CREDIT
  - AMT_ANNUITY
  - AMT_GOODS_PRICE
  - AMT_INCOME_TOTAL

```
#defining function for continious-continous columns
def cont_cont_plot(df, var1, var2):

    plt.figure(figsize=(15,8))

    # --- Subplot 1: DEFAULTERS (df1) ---
    plt.subplot(1, 2, 1)
    # Pass the column names as strings (var1, var2) to sns.countplot
    ax1 = sns.scatterplot(data=df1, x=var1, y=var2)
    # Pass the column names as strings for the title and labels
    plt.title(f"Correlation of {var1} by {var2} for Defaulters")
    plt.xlabel(var1)
    plt.ylabel(var2)

    # --- Subplot 2: NON-DEFAULTERS (df0) ---
    plt.subplot(1, 2, 2)
    ax2 = sns.scatterplot(data=df0, x=var1, y=var2)
    plt.title(f"Correlation of {var1} by {var2} for Others")
    plt.xlabel(var1)
    plt.ylabel(var2)

    # Crucial for preventing subplot titles and labels from overlapping
    plt.tight_layout()
    plt.show()
```

```
#function to calc correlation
def calc_correlation(df,col1,col2):
    # Correlation for Repayers (T=0)
    corr_repayers = df0[col1].corr(df0[col2])
    # Correlation for Defaulters (T=1)
    corr_defaulters = df1[col1].corr(df1[col2])

    print(f"--- Correlation for {col1} vs. {col2} ---")
    print(f"Repayers (T=0) Correlation: {corr_repayers:.4f}")
    print(f"Defaulters (T=1) Correlation: {corr_defaulters:.4f}")

    return corr_repayers, corr_defaulters
```

```
cont_cont_plot(new_app_df,'AMT_CREDIT','AMT_ANNUITY')
```

```
rep_corr, def_corr = calc_correlation(new_app_df,'AMT_CREDIT', 'AMT_ANNUITY')
```

- In both graphs, AMT_REDIT is strongly positively correlated to AMT_ANNUITY.

```
cont_cont_plot(new_app_df,'AMT_CREDIT','AMT_GOODS_PRICE')
```

```
rep_corr, def_corr = calc_correlation(new_app_df,'AMT_CREDIT', 'AMT_GOODS_PRICE
```

- In both graphs, AMT_CREDIT is strongly correlated to AMT_GOODS_PRICE.

```
cont_cont_plot(new_app_df,'AMT_CREDIT','AMT_INCOME_TOTAL')
```

- There is no correlation between AMT_CREDIT & AMT_INCOME_TOTAL in both graphs.

4.3 Correlation

```
# split data by target
df0 = new_app_df[new_app_df['TARGET'] == 0]
df1 = new_app_df[new_app_df['TARGET'] == 1]

#compute corr
corr0 = df0.corr(numeric_only=True)
corr1 = df1.corr(numeric_only=True)

def get_top_corr(corr, n=10):
    # take upper triangle only (no duplicates)
    c = corr.where(np.triu(np.ones(corr.shape), k=1).astype(bool))
    c = c.stack().sort_values(ascending=False)
    return c.head(n)

top10_target1 = get_top_corr(corr1, 10)
top10_target0 = get_top_corr(corr0, 10)

print("\nTOP 10 CORRELATIONS — Target = 1")
print(top10_target1)

print("\nTOP 10 CORRELATIONS — Target = 0")
print(top10_target0)
```

```
TOP 10 CORRELATIONS — Target = 1
OBS_30_CNT_SOCIAL_CIRCLE  OBS_60_CNT_SOCIAL_CIRCLE    0.998270
BASEMENTAREA_AVG          BASEMENTAREA_MEDI           0.998252
COMMONAREA_AVG            COMMONAREA_MEDI             0.998204
NONLIVINGAPARTMENTS_AVG   NONLIVINGAPARTMENTS_MEDI    0.998116
YEARS_BUILD_AVG           YEARS_BUILD_MEDI            0.998115
FLOORSMIN_AVG             FLOORSMIN_MEDI              0.997826
LIVINGAPARTMENTS_AVG      LIVINGAPARTMENTS_MEDI       0.997721
NONLIVINGAPARTMENTS_MODE  NONLIVINGAPARTMENTS_MEDI    0.997076
ENTRANCES_AVG             ENTRANCES_MEDI              0.996697
LIVINGAREA_AVG            LIVINGAREA_MEDI             0.996050
dtype: float64

TOP 10 CORRELATIONS — Target = 0
YEARS_BUILD_AVG           YEARS_BUILD_MEDI            0.998523
OBS_30_CNT_SOCIAL_CIRCLE  OBS_60_CNT_SOCIAL_CIRCLE    0.998510
FLOORSMIN_AVG             FLOORSMIN_MEDI              0.997244
ENTRANCES_AVG             ENTRANCES_MEDI              0.996913
ELEVATORS_AVG             ELEVATORS_MEDI              0.996747
COMMONAREA_AVG            COMMONAREA_MEDI             0.996119
LIVINGAREA_AVG            LIVINGAREA_MEDI             0.995763
APARTMENTS_AVG            APARTMENTS_MEDI             0.995321
BASEMENTAREA_AVG          BASEMENTAREA_MEDI           0.994160
LIVINGAPARTMENTS_AVG      LIVINGAPARTMENTS_MEDI       0.993889
```

```
dtype: float64
```

5. PREVIOUS_APPLICATION

5.1 Read the Data File

```
# load dataset
prev_df = pd.read_csv('previous_application.csv')
```

```
#read first 5 rows
prev_df.head(5)
```

```
#read last 5 rows
prev_df.tail(5)
```

5.2 Inspect the Data Frame

```
prev_df.shape
```

```
prev_df.columns.values
```

```
prev_df.dtypes
```

```
pd.options.display.float_format = '{:,.2f}'.format
prev_df.describe()
```

- AMT_ANNUITY, AMT_APPLICATION, AMT_CREDIT, AMT_DOWN_PAYMENT & AMT_GOODS_PRICE have outliers.

5.3 Data Imbalance Checking

```
#Count null values per column
null_count = prev_df.isnull().sum()
#Total number of rows
total_count = prev_df.shape[0]
# Calculate percentage of nulls per column and round to 2 decimal places
null_pct2 = ((null_count / total_count) * 100).round(2)
```

```
#Filter only columns with missing values (makes plot cleaner)
null_pct2 = null_pct2[null_pct2 > 0].sort_values(ascending=False)

#Plot using pointplot
plt.figure(figsize=(16,5))
sns.pointplot(x=null_pct2.index, y=null_pct2.values)
plt.xticks(rotation=90)
plt.xlabel("Columns")
plt.ylabel("Missing Values (%)")
plt.title("Percentage of Null Values per Column")
plt.show()
```

6.Data Cleaning

6.1 Missing Value Imputation

```
# create a copy of base data
new_prev_df = prev_df.copy()
```

```
new_prev_df.head(5)
```

```
#check for missing values in percentage for each col
null_pct2.sort_values(ascending=True)
```

```
#get column names with more than 50% missing
cols_50 = null_pct2[null_pct2 >= 50].index.tolist()
cols_50
```

```
['RATE_INTEREST_PRIMARY',
 'RATE_INTEREST_PRIVILEGED',
 'AMT_DOWN_PAYMENT',
 'RATE_DOWN_PAYMENT']
```

```
#remove the colums with more than 50% missing
```

```
new_prev_df.drop(cols_50, axis=1, inplace=True)
new_prev_df.shape
```

```
(1670214, 33)
```

- We will also drop NAME_TYPE_SUITE since it has null % value close to 50%.

```
new_prev_df.drop('NAME_TYPE_SUITE', axis=1, inplace=True)
new_prev_df.shape
```

```
(1670214, 32)
```

```
new_prev_df['NFLAG_INSURED_ON_APPROVAL'].value_counts()
```

- We will impute the null values of NFLAG_INSURED_ON_APPROVAL with NA eventhough the null & is very close to 50%. NA is chosen because missingness in this column is likely non-random(representing 'Not Applicable').
- First we will convert its data type to object.

```
new_prev_df['NFLAG_INSURED_ON_APPROVAL'].astype(object)
```

```
0          0.00
1          1.00
2          1.00
3          1.00
4           NaN
            ...
1670209    0.00
1670210    0.00
1670211    0.00
1670212    1.00
1670213    0.00
Name: NFLAG_INSURED_ON_APPROVAL, Length: 1670214, dtype: object
```

```
#fill null values with NA (third category)
new_prev_df['NFLAG_INSURED_ON_APPROVAL'].fillna('NA', inplace=True)
```

```
# check to see any null valus remaining
new_prev_df['NFLAG_INSURED_ON_APPROVAL'].isna().sum()
```

```
np.int64(0)
```

- We will impute PRODUCT_COMBINATION with mode since it is a categorical data

```
new_prev_df['PRODUCT_COMBINATION'].fillna(new_prev_df['PRODUCT_COMBINATION'].mc
```

```
#check to see any null values remaining
new_prev_df['PRODUCT_COMBINATION'].isna().sum()
```

```
np.int64(0)
```

- Next, we will get a list of columns with values of 365243.

```
PLACEHOLDER_CODE = 365243

# 1. Create a boolean Series: True if the column contains the placeholder
#    (Using .any() checks if ANY value in the column is equal to the code)
cols_with_placeholder = (new_prev_df == PLACEHOLDER_CODE).any()

# 2. Filter the index to get the list of column names
placeholder_cols = cols_with_placeholder[cols_with_placeholder].index.tolist()
placeholder_cols
```

```
['SK_ID_CURR',
 'DAYS_FIRST_DRAWING',
 'DAYS_FIRST_DUE',
 'DAYS_LAST_DUE_1ST_VERSION',
 'DAYS_LAST_DUE',
 'DAYS_TERMINATION']
```

- We will also convert 365243 which represents a missing value to NAN in DAYS_FIRST_DUE ,DAYS_TERMINATION , DAYS_FIRST_DRAWING, DAYS_FIRST_DUE, DAYS_LAST_DUE_1ST_VERSION,DAYS_LAST_DUE

```
cols_to_convert = ['DAYS_FIRST_DRAWING','DAYS_FIRST_DUE','DAYS_LAST_DUE_1ST_VER

for col in cols_to_convert:
    # Action: Modifies the column directly and returns None
    new_prev_df[col].replace(365243, np.nan, inplace=True)
```

- We will impute AMT_ANNUITY, CNT_PAYMENT, AMT_GOODS_PRICE, DAYS_FIRST_DUE, DAYS_TERMINATION, DAYS_FIRST_DRAWING, DAYS_LAST_DUE_1ST_VERSION & DAYS_LAST_DUE with the median since they are numerical columns.

```
num_cols = ['AMT_ANNUITY', 'CNT_PAYMENT', 'AMT_GOODS_PRICE', 'DAYS_FIRST_DUE',
            'DAYS_TERMINATION', 'DAYS_FIRST_DRAWING','DAYS_LAST_DUE_1ST_VERSION

for col in num_cols:
    # Ensure you are calculating the median *excluding* the NaNs
    median_val = new_prev_df[col].median()
    new_prev_df[col].fillna(median_val, inplace=True)
```

```
#check entire dataframe for null values
new_prev_df.isnull().sum()
```

```
SK_ID_PREV                     0
SK_ID_CURR                     0
```

```
NAME_CONTRACT_TYPE            0
AMT_ANNUITY                   0
AMT_APPLICATION               0
AMT_CREDIT                    1
AMT_GOODS_PRICE               0
WEEKDAY_APPR_PROCESS_START    0
HOUR_APPR_PROCESS_START       0
FLAG_LAST_APPL_PER_CONTRACT   0
NFLAG_LAST_APPL_IN_DAY        0
NAME_CASH_LOAN_PURPOSE        0
NAME_CONTRACT_STATUS          0
DAYS_DECISION                 0
NAME_PAYMENT_TYPE             0
CODE_REJECT_REASON            0
NAME_CLIENT_TYPE              0
NAME_GOODS_CATEGORY           0
NAME_PORTFOLIO                0
NAME_PRODUCT_TYPE             0
CHANNEL_TYPE                  0
SELLERPLACE_AREA              0
NAME_SELLER_INDUSTRY          0
CNT_PAYMENT                   0
NAME_YIELD_GROUP              0
PRODUCT_COMBINATION           0
DAYS_FIRST_DRAWING            0
DAYS_FIRST_DUE                0
DAYS_LAST_DUE_1ST_VERSION     0
DAYS_LAST_DUE                 0
DAYS_TERMINATION              0
NFLAG_INSURED_ON_APPROVAL     0
dtype: int64
```

- AMT_CREDIT has a null value, so we will impute it with median.

```
new_prev_df['AMT_CREDIT'].fillna(new_prev_df['AMT_CREDIT'].median(), inplace=Tr
```

```
#recheck entire dataframe for null values
new_prev_df.isnull().sum()
```

```
SK_ID_PREV                    0
SK_ID_CURR                    0
NAME_CONTRACT_TYPE            0
AMT_ANNUITY                   0
AMT_APPLICATION               0
AMT_CREDIT                    0
AMT_GOODS_PRICE               0
WEEKDAY_APPR_PROCESS_START    0
HOUR_APPR_PROCESS_START       0
FLAG_LAST_APPL_PER_CONTRACT   0
NFLAG_LAST_APPL_IN_DAY        0
NAME_CASH_LOAN_PURPOSE        0
NAME_CONTRACT_STATUS          0
DAYS_DECISION                 0
```

```
NAME_PAYMENT_TYPE            0
CODE_REJECT_REASON           0
NAME_CLIENT_TYPE             0
NAME_GOODS_CATEGORY          0
NAME_PORTFOLIO               0
NAME_PRODUCT_TYPE            0
CHANNEL_TYPE                 0
SELLERPLACE_AREA             0
NAME_SELLER_INDUSTRY         0
CNT_PAYMENT                  0
NAME_YIELD_GROUP             0
PRODUCT_COMBINATION          0
DAYS_FIRST_DRAWING           0
DAYS_FIRST_DUE               0
DAYS_LAST_DUE_1ST_VERSION    0
DAYS_LAST_DUE                0
DAYS_TERMINATION             0
NFLAG_INSURED_ON_APPROVAL    0
dtype: int64
```

6.2 Data Type Correction

```
count_cols = ['DAYS_FIRST_DRAWING','DAYS_FIRST_DUE','DAYS_LAST_DUE_1ST_VERSION'

for col in count_cols:
    new_prev_df[col] = new_prev_df[col].astype('int64')
```

6.3 Data Standardization

- There are some columns having negative values. We will convert them to positive using their absolute values.
- The columns are DAYS_FIRST_DRAWING, DAYS_FIRST_DUE, DAYS_LAST_DUE_1ST_VERSION, DAYS_LAST_DUE, DAYS_TERMINATION, DAYS_DECISION.

```
new_prev_df.describe()
```

```
pos_days_cols = ['DAYS_FIRST_DRAWING', 'DAYS_FIRST_DUE', 'DAYS_LAST_DUE_1ST_VEF
# concvert to absolute values
for col in pos_days_cols:
    new_prev_df[col]= new_prev_df[col].abs()
```

6.4 Outlier Analysis

- These are the columns we will perform outlier analysis on: AMT_ANNUITY, AMT_APPLICATION, AMT_CREDIT, AMT_GOODS_PRICE
- Firstly we will perform IQR Rule followed by visualizations using boxlpots for each column.

```
outlier_cols = ['AMT_ANNUITY','AMT_APPLICATION','AMT_CREDIT','AMT_GOODS_PRICE']

def find_outliers(df, col):
    Q1 = df[col].quantile(0.25)
    Q3 = df[col].quantile(0.75)
    IQR = Q3 - Q1
    lower = Q1 - 1.5 * IQR
    upper = Q3 + 1.5 * IQR
    outliers = df[(df[col] < lower) | (df[col] > upper)]
    return len(outliers), lower, upper

# Summary
outlier_summary = []
for col in outlier_cols:
    n_out, low, up = find_outliers(new_prev_df, col)
    pct = n_out / len(new_prev_df) * 100
    outlier_summary.append({
        'Column': col,
        'Outliers': n_out,
        '%': round(pct, 2),
        'Lower': round(low, 2),
        'Upper': round(up, 2)
    })

summary_df = pd.DataFrame(outlier_summary)
print(summary_df)
```

```
            Column  Outliers     %        Lower      Upper
0      AMT_ANNUITY    162620  9.74    -6,368.30  30,739.42
1  AMT_APPLICATION    208019 12.45 -223,740.00 422,820.00
2       AMT_CREDIT    179989 10.78 -264,226.50 504,805.50
3  AMT_GOODS_PRICE    236035 14.13 -101,857.50 349,762.50
```

```
fig, axes = plt.subplots(2, 2, figsize=(8,6))
axes = axes.ravel()

for i, col in enumerate(outlier_cols):
    sns.boxplot(data=new_prev_df, y=col, ax=axes[i], color='lightblue')
    axes[i].set_title(col)
    axes[i].grid(True, alpha=0.3)

plt.tight_layout()
plt.show()
```

- We will also perform capping on AMT_ANNUITY, AMT_APPLICATION, AMT_CREDIT & AMT_GOODS_PRICE

```
#capping
# List of columns to cap
cap_cols = [
    'AMT_ANNUITY','AMT_APPLICATION','AMT_CREDIT','AMT_GOODS_PRICE'
]

# Cap using IQR upper bound
for col in cap_cols:
    Q1 = new_prev_df[col].quantile(0.25)
    Q3 = new_prev_df[col].quantile(0.75)
    IQR = Q3 - Q1
    upper = Q3 + 1.5 * IQR
    new_prev_df[col] = new_prev_df[col].clip(upper=upper)
```

```
fig, axes = plt.subplots(2, 2, figsize=(8,6))
axes = axes.ravel()

for i, col in enumerate(outlier_cols):
    sns.boxplot(data=new_prev_df, y=col, ax=axes[i], color='lightblue')
    axes[i].set_title(col)
    axes[i].grid(True, alpha=0.3)

plt.tight_layout()
plt.show()
```

7.MERGING DATA SETS

7.1 Merging the Data Sets

- We will perform Data Flattening. Its purpose is to summarize all of an applicant's past loan applications into a single row of features, making the rich historical data usable in the main application table.

```
### NOTE: THIS TAKES ABOUT 20 MINS TO RUN

import pandas as pd
import numpy as np

import pandas as pd
import numpy as np

# ============================================
# 1. Columns you specifically want
# ============================================

prev_keep = [
    'SK_ID_CURR',
    'AMT_ANNUITY',
    'AMT_APPLICATION',
    'AMT_CREDIT',
    'AMT_GOODS_PRICE',
    'CNT_PAYMENT',
    'DAYS_DECISION',
    'SELLERPLACE_AREA',
    'NAME_CONTRACT_STATUS',
    'CODE_REJECT_REASON',
    'NAME_CASH_LOAN_PURPOSE',
    'PRODUCT_COMBINATION',
    'NFLAG_INSURED_ON_APPROVAL'
]

prev_small = new_prev_df[prev_keep].copy()

# ============================================
# 2. NUMERIC aggregation → median
# ============================================

num_cols = [
    'AMT_ANNUITY',
    'AMT_APPLICATION',
    'AMT_CREDIT',
    'AMT_GOODS_PRICE',
    'CNT_PAYMENT',
    'DAYS_DECISION',
    'SELLERPLACE_AREA'
]

prev_num_agg = (
    prev_small.groupby('SK_ID_CURR')[num_cols]
    .median()
    .reset_index()
)

# ============================================
# 3. CATEGORICAL aggregation → mode
# ============================================

cat_cols = [
    'NAME_CONTRACT_STATUS',
    'CODE_REJECT_REASON',
    'NAME_CASH_LOAN_PURPOSE',
    'PRODUCT_COMBINATION',
    'NFLAG_INSURED_ON_APPROVAL'
]

def mode_or_unknown(x):
    m = x.mode()
    return m.iloc[0] if not m.empty else 'UNKNOWN'

prev_cat_agg = (
    prev_small.groupby('SK_ID_CURR')[cat_cols]
    .agg(mode_or_unknown)
    .reset_index()
)

# ============================================
# 4. Combine numeric + categorical
# ============================================

prev_agg = prev_num_agg.merge(prev_cat_agg, on='SK_ID_CURR', how='left')

# ============================================
# 5. Rename columns to avoid collisions
# ============================================

prev_agg.rename(columns=lambda c: f"PREV_{c}" if c != 'SK_ID_CURR' else c,
                inplace=True)

# ============================================
# 6. Merge with application data
# ============================================

merged_df = new_app_df.merge(prev_agg, on='SK_ID_CURR', how='left')

print("Final merged_df shape:", merged_df.shape)
```

```
Final merged_df shape: (307511, 94)
```

```
# check the records to see the new merged df
merged_df.head(5)
```

```
#Count null values per column
null_count = merged_df.isnull().sum()
#Total number of rows
total_count = merged_df.shape[0]
# Calculate percentage of nulls and round to 2 decimal places
null_pct3 = ((null_count / total_count) * 100).round(2)
#Filter only columns with missing values (makes plot cleaner)
null_pct3 = null_pct3[null_pct3 > 0].sort_values(ascending=False)
```

```
#Plot using pointplot
plt.figure(figsize=(16,5))
sns.pointplot(x=null_pct3.index, y=null_pct3.values)
plt.xticks(rotation=90)
plt.xlabel("Columns")
plt.ylabel("Missing Values (%)")
plt.title("Percentage of Null Values per Column")
plt.show()
```

```
#check for missing value in percentage
null_pct3.sort_values(ascending=True)
```

- There are many 'PREV' columns with NAN.These nan appear because the customer has no previous loans. We will use 0 for these numerical columns.

```
prev_cols = ['PREV_DAYS_DECISION','PREV_AMT_CREDIT','PREV_AMT_APPLICATION','PRE

for col in prev_cols:
    merged_df[col].fillna(0, inplace=True)
```

```
merged_df['YEARS_EMPLOYED'].fillna(merged_df['YEARS_EMPLOYED'].median(),inplace
```

```
cat_col = ['PREV_NAME_CONTRACT_STATUS','PREV_NFLAG_INSURED_ON_APPROVAL','PREV_C

for col in cat_col:
    merged_df[col].fillna('NA', inplace=True)
```

```
#check for remaining missing values
missing_count = merged_df.isnull().sum()
missing_data = missing_count[missing_count > 0]
missing_data
```

```
CODE_GENDER    4
dtype: int64
```

```
merged_df['CODE_GENDER'].fillna(merged_df['CODE_GENDER'].mode()[0],inplace=True
```

```
#check again
missing_count = merged_df.isnull().sum()
missing_data = missing_count[missing_count > 0]
missing_data
```

```
Series([], dtype: int64)
```

- We will also perform feature binning for the numerical columns that we will use for bivariate analysis, categorical-continous columns. This is to make our visualization cleaner.

PREV_ANNUITY

```
# Define the number of quantiles/bins (e.g., 4 quartiles)
N_BINS = 4

# Create the new binned column
merged_df['PREV_ANNUITY_BIN_FREQ'] = pd.qcut(
    merged_df['PREV_AMT_ANNUITY'],
    q=N_BINS,
    # Optional: name your categories (e.g., Low, Medium, High)
    labels=['Q1_Low', 'Q2_MidLow', 'Q3_MidHigh', 'Q4_High'],
    duplicates='drop' # Handles cases where values are identical at the cutoff
)

# Check the results: notice the counts are much closer to each other
merged_df['PREV_ANNUITY_BIN_FREQ'].value_counts().sort_index()
```

```
PREV_ANNUITY_BIN_FREQ
Q1_Low         76878
Q2_MidLow     125809
Q3_MidHigh     27946
Q4_High        76878
Name: count, dtype: int64
```

PREV_APPLICATION

```
# Define the number of quantiles/bins (e.g., 4 quartiles)
N_BINS = 4

# Create the new binned column
merged_df['PREV_APPLICATION_BIN_FREQ'] = pd.qcut(
    merged_df['PREV_AMT_APPLICATION'],
    q=N_BINS,
    # Optional: name your categories (e.g., Low, Medium, High)
    labels=['Q1_Low', 'Q2_MidLow', 'Q3_MidHigh', 'Q4_High'],
    duplicates='drop' # Handles cases where values are identical at the cutoff
)

# Check the results: notice the counts are much closer to each other
merged_df['PREV_APPLICATION_BIN_FREQ'].value_counts().sort_index()
```

```
PREV_APPLICATION_BIN_FREQ
Q1_Low        76889
```

```
Q2_MidLow     76875
Q3_MidHigh    76871
Q4_High       76876
Name: count, dtype: int64
```

PREV_CREDIT

```
# Define the number of quantiles/bins (e.g., 4 quartiles)
N_BINS = 4

# Create the new binned column
merged_df['PREV_CREDIT_BIN_FREQ'] = pd.qcut(
    merged_df['PREV_AMT_CREDIT'],
    q=N_BINS,
    # Optional: name your categories (e.g., Low, Medium, High)
    labels=['Q1_Low', 'Q2_MidLow', 'Q3_MidHigh', 'Q4_High'],
    duplicates='drop' # Handles cases where values are identical at the cutoff
)

# Check the results: notice the counts are much closer to each other
merged_df['PREV_CREDIT_BIN_FREQ'].value_counts().sort_index()
```

```
PREV_CREDIT_BIN_FREQ
Q1_Low        76884
Q2_MidLow     76873
Q3_MidHigh    76876
Q4_High       76878
Name: count, dtype: int64
```

```
unique_count = merged_df['PREV_CNT_PAYMENT'].nunique()
print(f"Unique values in PREV_CNT_PAYMENT: {unique_count}")
```

PREV_DAYS_DECISION

```
# Create the new binned column using pd.qcut
# This splits the data into 4 equal-sized groups based on the count of previous
N_BINS = 4

merged_df['PREV_DAYS_DECISION_BIN_FREQ'] = pd.qcut(
    merged_df['PREV_DAYS_DECISION'],
    q=N_BINS, # Splits the data into 4 bins (quartiles)
    # 4 bins require exactly 4 labels
    labels=['Q1_Low', 'Q2_MidLow', 'Q3_MidHigh', 'Q4_High'],
    duplicates='drop'
)

merged_df['PREV_DAYS_DECISION_BIN_FREQ'].value_counts().sort_index()
```

```
[Binned Feature Result]
PREV_DAYS_DECISION_BIN_FREQ
Q1_Low        77099
Q2_MidLow     76673
Q3_MidHigh    76914
Q4_High       76825
Name: count, dtype: int64
```

PREV_GOODS_PRICE

```
# Create the new binned column using pd.qcut
# This splits the data into 4 equal-sized groups based on the count of previous
N_BINS = 4

merged_df['PREV_GOODS_PRICE_BIN_FREQ'] = pd.qcut(
    merged_df['PREV_AMT_GOODS_PRICE'],
    q=N_BINS, # Splits the data into 4 bins (quartiles)
    # 4 bins require exactly 4 labels
    labels=['Q1_Low', 'Q2_MidLow', 'Q3_MidHigh', 'Q4_High'],
    duplicates='drop'
)

merged_df['PREV_GOODS_PRICE_BIN_FREQ'].value_counts().sort_index()
```

```
[Binned Feature Result]
PREV_GOODS_PRICE_BIN_FREQ
Q1_Low         76878
Q2_MidLow     127866
Q3_MidHigh     25896
Q4_High        76871
Name: count, dtype: int64
```

PREV_PAYMENT

```
N_BINS = 4

# --- SIMPLIFIED SOLUTION: Using pd.cut for fixed-width binning ---
# Since pd.qcut was unstable, pd.cut is used to segment the feature based on va
# This operation is stable and simple, avoiding all the previous errors.

merged_df['PREV_PAYMENT_BIN_FREQ'] = pd.cut(
    merged_df['PREV_CNT_PAYMENT'],
    bins=N_BINS, # Automatically calculates 4 equally spaced bins between min a
    labels=['Q1_Low', 'Q2_MidLow', 'Q3_MidHigh', 'Q4_High'],
    include_lowest=True # Ensures the minimum value is included in the first bi
)
```

```
# Check the results
merged_df['PREV_PAYMENT_BIN_FREQ'].value_counts().sort_index()
```

```
[Binned Feature Result (pd.cut)]
PREV_PAYMENT_BIN_FREQ
Q1_Low        277787
Q2_MidLow      26932
Q3_MidHigh      2063
Q4_High          729
Name: count, dtype: int64
```

7.2 Segmented Univariate Analysis

Univariate Categorical Analysis

- We will perform this analysis on these columns
  - PREV_NAME_CONTRACT_STATUS, PREV_NFLAG_INSURED_ON_APPROVAL, PREV_CODE_REJECT_REASON, PREV_NAME_CASH_LOAN_PURPOSE, PREV_PRODUCT_COMBINATION

PREV_NAME_CONTRACT_STATUS

```
cat_plot(merged_df,'PREV_NAME_CONTRACT_STATUS')
```

- Approved contract has the highest defaulters & payment on time.

```
univariate_decision_support_risk_plot(merged_df,'PREV_NAME_CONTRACT_STATUS')
```

- Refused previous loans have the highest default risk rate followed by cancelled previous loans.
- NA (no previous loans) has the lowest default risk rate followed by approved loans.

PREV_NFLAG_INSURED_ON_APPROVAL

```
cat_plot(merged_df,'PREV_NFLAG_INSURED_ON_APPROVAL')
```

- People with no previous loan status has the highest defaulters and those paying on time.

```
univariate_decision_support_risk_plot(merged_df,'PREV_NFLAG_INSURED_ON_APPROVAL
```

- People with no previous loans status have the highest default risk rate.
- People with 1 previous loan status have the lowest default rate.

PREV_CODE_REJECT_REASON

```
cat_plot(merged_df,'PREV_CODE_REJECT_REASON')
```

- The number of defaulters and those paying on time is the highest for XAP.

```
univariate_decision_support_risk_plot(merged_df,'PREV_CODE_REJECT_REASON')
```

- SCOFR has the highest default risk rate.
- NA has the lowest default risk rate.

PREV_NAME_CASH_LOAN_PURPOSE

```
cat_plot(merged_df,'PREV_NAME_CASH_LOAN_PURPOSE')
```

- XAP has the highest number of defaulters and those paying on time.

```
univariate_decision_support_risk_plot(merged_df,'PREV_NAME_CASH_LOAN_PURPOSE')
```

- Cash loan for water supply has the highest default risk rate.
- The default rate for refusal to name the goal, hobby & money for a third person is non-existant.
- The default rate is the lowest for business development.

PREV_PRODUCT_COMBINATION

```
cat_plot(merged_df,'PREV_PRODUCT_COMBINATION')
```

- The number of defaulters & those paying on time is the highest for POS household with interest.

```
univariate_decision_support_risk_plot(merged_df,'PREV_PRODUCT_COMBINATION')
```

- The highest default risk rate is for Cash X-Sell: high.
- The lowest default risk rate is for POS industry wihtout interest.

Univariate Numerical Analysis

- We will perform this analysis on these columns:
  - PREV_AMT_ANNUITY, PREV_AMT_APPLICATION, PREV_AMT_CREDIT, PREV_AMT_GOODS_PRICE , PREV_CNT_PAYMENT, PREV_DAYS_DECISION, PREV_SELLERPLACE_AREA

```
df0 = merged_df[merged_df['TARGET'] == 0]
df1 = merged_df[merged_df['TARGET'] == 1]
```

PREV_AMT_ANNUITY

```
cont_plot(merged_df,'PREV_AMT_ANNUITY')
```

- The risk density is the highest among those with previous amount annuity between 0 and 5000.
- The risk density is the lowest among those with high previous amount annuity between 10000 and 15000.

```
univariate_decision_support_risk_plot(merged_df,'PREV_ANNUITY_BIN_FREQ')
```

- Q2_MidLow has the highest default risk rate.
- Q4_High has the lowest default risk rate.

PREV_AMT_APPLICATION

```
cont_plot(merged_df,'PREV_AMT_APPLICATION')
```

- The risk density is the highest between 0 and 50000.
- The risk density is the lowest for previous amount application > 400000.

```
univariate_decision_support_risk_plot(merged_df,'PREV_APPLICATION_BIN_FREQ')
```

- Q1_Low has the highest default risk rate.
- Q4_High has the lowest default risk rate.

PREV_AMT_CREDIT

```
cont_plot(merged_df,'PREV_AMT_CREDIT')
```

- The risk density is the highest between 0 and 100000.
- The risk dnesity is the lowest for previous amount credit > 500000.

```
univariate_decision_support_risk_plot(merged_df,'PREV_CREDIT_BIN_FREQ')
```

- Q1_Low has the highest default risk rate.
- Q3_MidHigh has the lowest default risk rate.

```
cont_plot(merged_df,'PREV_AMT_GOODS_PRICE')
```

- The highest risk density is around 50000.
- The lowest risk density is between 100000 & 150000.

```
univariate_decision_support_risk_plot(merged_df,'PREV_GOODS_PRICE_BIN_FREQ')
```

- The default risk rate is the highest for Q2_MidLow.
- The default risk rate is the lowest for Q3_MidHigh.

PREV_CNT_PAYMENT

```
cont_plot(merged_df,'PREV_CNT_PAYMENT')
```

- Client with around 12 payment has the lowest risk density.

```
univariate_decision_support_risk_plot(merged_df,'PREV_PAYMENT_BIN_FREQ')
```

- Q3_MidHigh has the highest default risk rate.
- Q1_Low has the lowest default risk rate.

PREV_DAYS_DECISION

```
cont_plot(merged_df,'PREV_DAYS_DECISION')
```

- Around 300 days, the risk density is the highest.
- People who recently made decision has the lowest risk density.

```
univariate_decision_support_risk_plot(merged_df,'PREV_DAYS_DECISION_BIN_FREQ')
```

- Q1_Low has the highest default risk rate.
- Q4_High has the lowest default risk rate.

SELLERPLACE_AREA

```
cont_plot(merged_df,'PREV_SELLERPLACE_AREA')
```

- Due to nature of reading, prev_sellerplace area is not a good indicator for default risk analysis

7.3 Bivariate Analysis

Categorical-categorical columns

- These are the pairings used:
  - PREV_NAME_CONTRACT_STATUS - NAME_HOUSING_TYPE
  - PREV_NFLAG_INSURED_ON_APPROVAL - OCCUPATION_TYPE
  - PREV_CODE_REJECT_REASON - CODE_GENDER
  - PREV_NAME_CASH_LOAN_PURPOSE - NAME_FAMILY_STATUS

```
df0 = merged_df[merged_df['TARGET']==0]
df1 = merged_df[merged_df['TARGET']==1]
```

PREV_NAME_CONTRACT_STATUS - NAME_HOUSING_TYPE

```
cat_cat_plot(merged_df,'PREV_NAME_CONTRACT_STATUS','NAME_HOUSING_TYPE')
```

- The number of defaulters & others is the highest for those with approved house/apartment across all contract status in both graphs.

```
bivariate_decision_support_risk_plot(merged_df,'PREV_NAME_CONTRACT_STATUS','NAM
```

- The highest default risk rate is among those who stays in a rented apartment & has their previous contract refused.
- Among the different contract status, those in a rented apartment has the highest default risk rate.

PREV_NFLAG_INSURED_ON_APPROVAL- OCCUPATION_TYPE

```
cat_cat_plot(merged_df,'PREV_NFLAG_INSURED_ON_APPROVAL','OCCUPATION_TYPE')
```

- Unkown occupation type seems to have the highest number of defautlers & payment on time across all insurance approval types.

```
bivariate_decision_support_risk_plot(merged_df,'PREV_NFLAG_INSURED_ON_APPROVAL'
```

- Low skill laborers have the highest default rate across all insurance approval flags.
- Accountants hav the lowest default rate across all insurance approval flags.

PREV_CODE_REJECT_REASON - CODE_GENDER

```
cat_cat_plot(merged_df,'PREV_CODE_REJECT_REASON','CODE_GENDER')
```

- Females have the higher number of defaulters & payment on time as compared to males for all different codes.

```
bivariate_decision_support_risk_plot(merged_df,'PREV_CODE_REJECT_REASON','CODE_
```

- Males have the higher default rate across all codes except for SYSTEM & XNA.
- For code SYSTEM, the default rate between males & females are the same.

PREV_NAME_CASH_LOAN_PURPOSE - NAME_FAMILY_STATUS

```
cat_cat_plot(merged_df,'PREV_NAME_CASH_LOAN_PURPOSE','NAME_FAMILY_STATUS')
```

- Married has the highest number of defaulters and payment on time.

```
bivariate_decision_support_risk_plot(merged_df,'PREV_NAME_CASH_LOAN_PURPOSE','N
```

- Civil marriage has the highest default risk rate for most of the cash loan purposes.
- Default rate for hobby and refusal to name the goal is non-existant.

Categorical-continuos columns

- These are the columns we will perform our analysis on:
  - PREV_AMT_ANNUITY - NAME_INCOME_TYPE
  - PREV_AMT_APPLICATION - OCCUPATION_TYPE
  - PREV_AMT_CREDIT - NAME_FAMILY_STATUS
  - PREV_CNT_PAYMENT - FLAG_OWN_CAR

PREV_AMT_ANNUITY - NAME_INCOME_TYPE

```
cat_cont_plot(merged_df,'PREV_AMT_ANNUITY','NAME_INCOME_TYPE')
```

- The box (IQR) for Maternity Leave is located much higher than any other category, with a median PREV_AMT_ANNUITY around 20000.
- The Maternity leave category is barely visible, showing a near-total lack of volume in the non-defaulting population.

- The Businessman category is virtually absent from the Defaulter plot and barely visible in the Non-Defaulter plot.

```
bivariate_decision_support_risk_plot(merged_df,'PREV_ANNUITY_BIN_FREQ','NAME_IN
```

- Unemployed income type within Q3_MidHigh or Q4_High has the highest default risk rate.
- For most of the income types, Q4_High seems to have the lowest default risk rate.

PREV_AMT_APPLICATION - OCCUPATION_TYPE

```
cat_cont_plot(merged_df,'PREV_AMT_APPLICATION','OCCUPATION_TYPE')
```

- The boxs (IQR) overlap each other while the median are about the same in both graphs.

```
bivariate_decision_support_risk_plot(merged_df,'PREV_APPLICATION_BIN_FREQ','NAM
```

- Q2_MidLow Unemployed within Q2_MidLow, Maternity Leave within Q3_MidHigh or Q4_High has the highest default risk rate.
- The default risk rate for businessman is non-existant.
- Pensions have the lowest default risk rate across all application categories.

PREV_AMT_CREDIT - NAME_FAMILY_STATUS

```
cat_cont_plot(merged_df,'PREV_AMT_CREDIT','NAME_FAMILY_STATUS')
```

- The boxes (IQR) overlap each other while the median are about the same in both graphs for most of the occupation types.
- The unknown category is virtually absent from the others graph.

```
bivariate_decision_support_risk_plot(merged_df,'PREV_CREDIT_BIN_FREQ','NAME_FAM
```

- For almost all credit categories, civil marriage has the highest default risk rate.
- Civil marriage within Q1_Low has the highest default risk rate.
- The default risk rate for unknown is non-existant.
- Widow has the lowest default risk rate across all credit categories.

Multivariate Analysis

- We will perform categorical-categorical-categorical types of multivariate analysis:
  - Categorical-categorical-categorical
  - Categorical-categorical-continous

Categorical - cateogorical - cateogorical

PREV_NAME_CONTRACT_STATUS - NAME_HOUSING_TYPE - NAME_INCOME_TYPE

```
cat_cat_cat_plot(merged_df,'PREV_NAME_CONTRACT_STATUS','NAME_HOUSING_TYPE','NAM
```

- Working group,state servant group,commercial associate & pensioner group with their previou contract status approved has the highest number of payment on time & defaulters

```
multivariate_decision_support_risk_plot(merged_df,'PREV_NAME_CONTRACT_STATUS','
```

- For the Commercial Associate group, the Rented Apartment housing type shows the highest default risk rate within the categories of Approved, Cancelled, and NA previous contract status
- For the Commerical Associate group, the NA contract status shows the lowest default risk rate within all the contract status types.

- For pensioner group, the NA contract status shows the highest default risk within all the contract status types.
- For the state servent group, the house/apartment housing type with unused offer shows the highest default risk rate within all contract stauts types.
- For maternity leave, the house/apartment housing type with approved contract status shows the highest defualt risk rate across all graphs.
- Similar to maternity leave, municipal apartment housing type with approved contract status & house/apartment with canceled status shows the highest defualt risk rate across all graphs.
- The default risk rate for businessman group is non-existant.

PREV_NAME_CASH_LOAN_PURPOSE - NAME_FAMILY_STATUS-FLAG_OWN_REALTY

```
cat_cat_cat_plot(merged_df,'PREV_NAME_CASH_LOAN_PURPOSE','NAME_FAMILY_STATUS','
```

- For those owing & not owning an asset, married family status shows the highest number of payment on tine across the XAP, XNA & NA cash loan purpose categories.

```
multivariate_decision_support_risk_plot(merged_df,'PREV_NAME_CASH_LOAN_PURPOSE'
```

- For those not owning an asset, civil marriage housing type within the buying a holiday home/land category shows the highest default risk rate.
- For those owning an asset, separated civil marriage housing type within the car repairs category shows the highest default risk rate.

PREV_AMT_ANNUITY - NAME_INCOME_TYPE - FLAG_OWN_REALTY

```
cat_cat_cat_plot(merged_df,'PREV_ANNUITY_BIN_FREQ','NAME_INCOME_TYPE','FLAG_OWN
```

- For the groups owning & not owning an asset, working category has the highest payment on time within all annuity categories.

```
multivariate_decision_support_risk_plot(merged_df,'PREV_ANNUITY_BIN_FREQ','NAME
```

- For those not owning an asset. unemployed income type within Q2_MidLow & Q3_MidHigh has the highest default risk rate.
- For those owning an asset, unemployed income type within Q2_MidLow & Q3_MidHigh has the highest default risk rate.

PREV_AMT_CREDIT - NAME_FAMILY_STATUS - FLAG_OWN_CAR

```
cat_cat_cat_plot(merged_df,'PREV_CREDIT_BIN_FREQ','NAME_FAMILY_STATUS','FLAG_OW
```

- For those not owning & owning a car, married within the Q4_High category shows the highest number of payment on time.

```
multivariate_decision_support_risk_plot(merged_df,'PREV_CREDIT_BIN_FREQ','NAME_
```

- For those not owning an asset, civil marriage category within Q1_Low shows the highest default rate.
- For those owning & not owning an asset, widow shows the the lowest default risk rate within each credit category.
- For those owning an asset, civil marriage within the Q1_Low category shows the highest default risk rate.

Continous-continuos columns

```
- PREV_AMT_CREDIT - PREV_AMT_GOODS_PRICE
```

PREV_AMT_CREDIT - PREV_AMT_GOODS_PRICE

```
cont_cont_plot(merged_df,'PREV_AMT_CREDIT','PREV_AMT_GOODS_PRICE')
```

```
rep_corr, def_corr = calc_correlation(merged_df,'PREV_AMT_CREDIT', 'PREV_AMT_GC
```

```
--- Correlation for PREV_AMT_CREDIT vs. PREV_AMT_GOODS_PRICE ---
Repayers (T=0) Correlation: 0.8949
Defaulters (T=1) Correlation: 0.8801
```

- PREV_AMT_CREDIT is strongly positively correlated to PREV_AMT_GOODS_PRICE.

PREV_AMT_CREDIT - AMT_APPLICATION

7.4 Correlation

```
# split data by target
df0 = merged_df[merged_df['TARGET'] == 0]
df1 = merged_df[merged_df['TARGET'] == 1]

#compute corr
corr0 = df0.corr(numeric_only=True)
corr1 = df1.corr(numeric_only=True)

def get_top_corr(corr, n=10):
    # take upper triangle only (no duplicates)
    c = corr.where(np.triu(np.ones(corr.shape), k=1).astype(bool))
    c = c.stack().sort_values(ascending=False)
    return c.head(n)

top10_target1 = get_top_corr(corr1, 10)
top10_target0 = get_top_corr(corr0, 10)

print("\nTOP 10 CORRELATIONS — Target = 1")
print(top10_target1)

print("\nTOP 10 CORRELATIONS — Target = 0")
print(top10_target0)
```

```

TOP 10 CORRELATIONS — Target = 1
OBS_30_CNT_SOCIAL_CIRCLE    OBS_60_CNT_SOCIAL_CIRCLE      1.00
AMT_CREDIT                  AMT_GOODS_PRICE               0.98
REGION_RATING_CLIENT        REGION_RATING_CLIENT_W_CITY   0.96
PREV_AMT_APPLICATION        PREV_AMT_CREDIT               0.94
                            PREV_AMT_GOODS_PRICE          0.91
CNT_CHILDREN                CNT_FAM_MEMBERS               0.88
PREV_AMT_CREDIT             PREV_AMT_GOODS_PRICE          0.88
DEF_30_CNT_SOCIAL_CIRCLE    DEF_60_CNT_SOCIAL_CIRCLE      0.87
REG_REGION_NOT_WORK_REGION  LIVE_REGION_NOT_WORK_REGION   0.85
PREV_AMT_ANNUITY            PREV_AMT_GOODS_PRICE          0.84
```

```
dtype: float64

TOP 10 CORRELATIONS — Target = 0
OBS_30_CNT_SOCIAL_CIRCLE    OBS_60_CNT_SOCIAL_CIRCLE      1.00
AMT_CREDIT                  AMT_GOODS_PRICE               0.99
REGION_RATING_CLIENT        REGION_RATING_CLIENT_W_CITY   0.95
PREV_AMT_APPLICATION        PREV_AMT_CREDIT               0.94
                            PREV_AMT_GOODS_PRICE          0.92
PREV_AMT_CREDIT             PREV_AMT_GOODS_PRICE          0.89
CNT_CHILDREN                CNT_FAM_MEMBERS               0.88
REG_REGION_NOT_WORK_REGION  LIVE_REGION_NOT_WORK_REGION   0.86
DEF_30_CNT_SOCIAL_CIRCLE    DEF_60_CNT_SOCIAL_CIRCLE      0.86
PREV_AMT_ANNUITY            PREV_AMT_GOODS_PRICE          0.85
dtype: float64
```

7.5 Conclusion

From the above analysis, we have gathered the below insights:

Part A: Strategic Overview & Research Insights

i) Research Questions

- How do occupational stability and life-stage transitions (like maternity or unemployment) interact with educational
  backgrounds to determine default risk?
    - The research indicates that stable professions such as Accountants and Pensioners act as a hedge against risk across all income groups. Conversely, life transitions like maternity leave or unemployment create severe risk spikes when coupled with lower education levels (Secondary/Secondary Special), suggesting that educational attainment is a primary resilience factor during income volatility.
- To what extent does "Civil Marriage" status combined with specific loan purposes (real estate/car repairs) signal financial distress compared to traditional marital statuses?
    - Findings suggest that "Civil Marriage" is a significant risk proxy, particularly when applicants are in the lowest income quartiles (Q1_Low) or attempting to fund specific lifestyle assets like holiday homes or car repairs. This identifies a behavioral trend where non-traditional marital structures may correlate with less conservative financial planning in specific credit categories.
- Does asset ownership effectively mitigate the risk of a "Refused" or "Cancelled" previous contract history?

    - The data shows that for unemployed individuals or those in rented apartments, previous contract status remains a dominant predictor. Even when owning an asset, certain groups (unemployed in Q2/Q3 annuity categories) remain highly likely to default, proving that asset ownership is not a universal solution for poor credit history or employment instability.

ii) Interesting Insights

- The "Widow" Reliability Factor: Among all female applicants, widows consistently demonstrate the lowest default risk rate within every credit category, regardless of asset ownership. This points to a highly disciplined financial demographic.
- Office Apartment Advantage: Applicants residing in office apartments show a consistently lower likelihood of default across all goods price groups, serving as a stronger positive indicator than general home ownership.
- The Commercial Associate Exception: While Academic Degree holders are generally the safest borrowers, those working as "Commercial Associates" deviate from this trend and show a higher likelihood of defaulting, highlighting role-specific volatility even among the highly educated.

Part B: Specific Findings

1. Employment Status and Occupational Risk Profiles

- This section details the default likelihood based on the applicant's current employment status, occupation type, and income category interactions.

  High Default Risk (Employment & Low Skill)

    - Maternity leave income type specifically with secondary/secondary special education are very likely to default.
    - Maternity leave laborers are more likely to default.
    - Low-skill laborers within Q1_Low income group are more likely to default.
    - Low skill laborers with NA nflag insured on approval are more likely to default.
    - Those unemployed within the Q2_MidLow credit group are more likely to default.
    - Those in Q2_MidLow credit group while being unemployed are more likely to default.
    - Those unemployed within Q3_MidHigh or Q4_High previous annuity group are more likely to default.
    - Those who are unemployed belonging to Q2_MidLow, maternity leave within Q3_MidHigh & Q4_High previous application are more likely to default.

  - Across all education types, those who are unemployed & not owning an asset while those who are on maternity leave & owning an asset are more likely to default.
  - For those owning an asset, unemployed income type within Q2_MidLow annuity category or Q3_MidHigh are more likely to default.

  Low Default Risk (Stable Occupations)

  - Pensioners are less likely to default.
  - Businessman & students are the least likely to default, indicating a very high likelihood of paying on time.
  - Pensioners are more likely to pay on time in almost every credit groups.
  - Accountants consistently show a higher likelihood of paying on time across all income groups.
  - State servent within Q4_High are less likely to default.
  - Pensioners are less likely to default within each previous application groups.

2. Education Level and Financial Group Interactions

- This section focuses on how education level influences default risk, particularly when combined with annuity groups.

  High Default Risk (Lower Education)

  - Those with lower secondary education wthin the Q3_MidHigh annuity group are more likely to default.
  - Those with lower secondary education regardless of whether they own an asset or have a certain number of children are more likely to default.
  - Those with lower secondary education belonging to Q3_MidLow or Q3_MidHigh annuity group are more likely to default.
  - Those with academic degree specifically who are commercial associates & working are likely to default while others being least likely.

  Low Default Risk (Higher Education)

  - Academic degree regardless of the annuity group consistently shows a high likelihood of paying on time.
  - Most of the people with academic degree & 2 or 3+ children are most likely to repay on time.
  - Those with academic degree either belonging to Q1_Low annuity group or Q3_MidHigh group are least likely to default.

3. Demographics, Family Status, and Gender

- This theme groups findings related to gender, family size, marital status, and related risk categories.

  High Default Risk (Demographics & Family Structure)

  - Regardless of the family status, males are more likely to default as compared to females.
  - Those with 3+ children are more likely to default while those with 0 children are less likely to default regardless of whether these two groups own a car or not.
  - Similarly, we found that those with 5+ family members are more likely to default while those with 2 members are less likely to default regardless of whether they own an asset or not.
  - Males with SCOFR code are more likely to default while females with VERIF less likely to default.
  - Civil marriage with buying a home land are more likely to default.
  - Civil marriage belonging to Q1_Low are more likely to default.
  - For those not owning an asset, civil marriage housing type within the buying a holiday home/land category are more likely to default.
  - For those not owning an asset, civil marriage category within Q1_Low shows the highest default rate.
  - For those owning an asset, separated civil marriage housing type within the car repairs category are more likely to default.
  - For those owning an asset, civil marriage within the Q1_Low category shows the highest default risk rate.

  Low Default Risk (Demographics)

  - Among all female applicants, those who are widows are less likely to default compared to every other family status, indicating a higher likelihood of paying on time.
  - For those owning & not owning an asset, widow shows the the lowest default risk rate within each credit category.

4. Loan Characteristics and Housing/Collateral

- This theme focuses on loan type (Cash vs. Revolving) and the applicant's housing situation.

  High Default Risk (Loan & Housing)

  - Regardless of the housing type, people with cash loans are more likely to default as compared to revolving loans.

- Specifically, we found that cash loans for apartment has the highest likelihood of defaulting.
- Those in rented apartment within Q2_MidLow goods price group are more likely to default.
- Those who stay in a rented apartment & has their previous contract refused are more likely to default.
- Among all the previous contract status, those in a rented apartment are more likely to default.

Low Default Risk (Loan & Housing)

- Revolving loans for municipal apartment are less likely to default.
- Those in office apartment consistently show that they are less likely to default across all goods price group.

5. External Scores and Multi-Factor Scenarios

- This theme covers complex interactions involving external credit scores and highly specific combinations of employment, assets, and previous application history.
    - Those in Q1_Low external source 2 group are more likely to default while those in Q4_High are less likely to default regardless of whether they own an asset or not.
    - Maternity group with their previous contract status approved & staying in a house/apartment are more likely to default.
    - Those unemployed approved contract staying in municipal apartment are more likely to default.
    - Unemployed group staying in house/apartment with contract cancelled are more likely to default.

Credit Risk: Consolidated Action Policy Guide

- This policy guide consolidates all identified default risk factors into mandatory actions (Decline, Review, or Approve) grouped by theme for simplified operational application.
- Employment Status and Occupational Risk Profiles

  A. Mandatory Decline (High Default Risk)

    - Unemployment/Temporary Leave: Automatically reject all applications from unemployed individuals.
    - Specific Maternity Cases: Automatically reject applicants who are currently on maternity leave AND have Secondary or Secondary Special education.

- Low-Skill Financial Risk: Automatically reject low-skill laborers who fall within the Q1_Low income group.

B. Mandatory Review and Mitigation (Conditional Check)

- Low-Skill Labor & Insurability: Require mandatory collateral or a financial guarantee for any laborer on maternity leave. Additionally, require mandatory insurance for low-skill laborers with an NA nflag.
- Asset-Holding Unemployed: If an unemployed applicant does own an asset (specifically in Q2_MidLow annuity or Q3_MidHigh groups), the loan must be 100% secured with that asset as collateral, requiring a manual underwriter review.

C. Instant Approval (Low Default Risk)

- Stable Profiles: Fast-track and offer the most favorable interest rates and terms to all Pensioners, Businessmen, Students, and Accountants.
- High-Tier Government Servants: Instantly approve State Servants who are in the Q4_High credit group.

- Education Level and Financial Group Interactions

A. Mandatory Decline (High Default Risk)

- Lower Secondary Education: Automatically reject all applicants with Lower Secondary Education, especially if they are in the Q3_MidLow or ' Q3_MidHigh annuity groups, regardless of asset ownership or number of children.

B. Mandatory Review and Mitigation (Conditional Check)

- Commercial Academic Associates: Require a manual review for all Commercial Associates who have an Academic Degree and are currently working.

C. Instant Approval (Low Default Risk)

- Academic Degree Holders: Offer premium terms, fast-track processing, and favorable pricing to all applicants with an Academic Degree, especially those with 2 or more children.

- Demographics, Family Status, and Gender

A. Mandatory Decline (High Default Risk)

- Civil Marriage & Financial Risk: Automatically reject applicants in a Civil Marriage if they are in the Q1_Low credit group (regardless of asset ownership) or if the loan is intended for real estate (buying a home land or holiday home).

B. Mandatory Review and Mitigation (Conditional Check)

- Gender and Risk: All male applicants must be subject to a higher Debt-to-Income (DTI) threshold compared to female applicants, especially those with the SCOFR code.
- Family Size: Require a co-signer or mandatory manual review for applicants with 3 or more children, or 5 or more family members.
- Separated Applicants: Require a co-signer for separated civil marriage applicants who own an asset and are applying for a car repairs loan.

C. Instant Approval (Low Default Risk)

- Widow Status: Fast-track and offer the best rates to all female applicants who are widows, as they consistently show the lowest default risk.
- Family Size: Offer favorable terms and a rate discount to applicants with zero or two family members.

- Loan Characteristics and Housing/Collateral

  A. Mandatory Decline (High Default Risk)

  - Cash Loan/Housing Specifics: Reject all Cash Loan applications intended for an apartment.
  - Rented Apartment Risk: Reject any applicant living in a Rented Apartment who has a previous contract refusal or is in the Q2_MidLow goods price group.

  B. Mandatory Review and Mitigation (Conditional Check)

  - Cash Loan Premium: All cash loan applications must carry a substantial risk premium or higher collateral requirement compared to revolving loans.

  C. Instant Approval (Low Default Risk)