Mitigating Loan Default Risks

As a data analyst in the finance industry, the primary goal is to mitigate loan default risks and optimize the loan approval process for urban customers. By understanding loan default patterns, the aim is to minimize financial losses while ensuring capable applicants are not rejected.

by Niraj Pal

Data Preprocessing and Cleaning

1 *Data Exploration*

1. Loading the data:

Formula: app_data = pd.read_csv('/content/drive/MyDrive/Colab Notebooks/Bank loan case study project./application_data.csv')

prev_data = pd.read_csv('/content/drive/MyDrive/Colab Notebooks/Bank loan case study project./previous_application.csv')

2. Checking the shape, structure and other info of data:

Formulas:

```
app_data.shape
```

```
(49999, 122)
```

```
prev_data.shape
```

```
(49999, 37)
```

```
[ ] app_data.head()
```

	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_T
0	100002	1	Cash loans	M	N	Y	0	202500.0	406597.5	24700.5	351000.0	Unac
1	100003	0	Cash loans	F	N	N	0	270000.0	1293502.5	35698.5	1129500.0	
2	100004	0	Revolving loans	M	Y	Y	0	67500.0	135000.0	6750.0	135000.0	Unac
3	100006	0	Cash loans	F	N	Y	0	135000.0	312682.5	29686.5	297000.0	Unac
4	100007	0	Cash loans	M	N	Y	0	121500.0	513000.0	21865.5	513000.0	Unac

```
[ ] prev_data.head()
```

	SK_ID_PREV	SK_ID_CURR	NAME_CONTRACT_TYPE	AMT_ANNUITY	AMT_APPLICATION	AMT_CREDIT	AMT_DOWN_PAYMENT	AMT_GOODS_PRICE	WEEKDAY_APPR_PROCESS_START	HOUR_APPR_PROC
0	2030495	271877	Consumer loans	1730.430	17145.0	17145.0	0.0	17145.0	SATURDAY	
1	2802425	108129	Cash loans	25188.615	607500.0	679671.0	NaN	607500.0	THURSDAY	
2	2523466	122040	Cash loans	15060.735	112500.0	136444.5	NaN	112500.0	TUESDAY	
3	2819243	176158	Cash loans	47041.335	450000.0	470790.0	NaN	450000.0	MONDAY	
4	1784265	202054	Cash loans	31924.395	337500.0	404055.0	NaN	337500.0	THURSDAY	

```
app_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 49999 entries, 0 to 49998
Columns: 122 entries, SK_ID_CURR to AMT_REQ_CREDIT_BUREAU_YEAR
dtypes: float64(64), int64(42), object(16)
memory usage: 46.5+ MB
```

2 *Data cleaning*

Removing the columns that have more the 30% null values.

- following are the steps and formulas:

1. Finding out the columns that have more then 30% null values.

Formula: null_data = app_data.isnull().sum()/49999*100

2. Storing the columns Index into different data Frame.

Formula: drop_col = null_data[null_data >= 30].index

drop_col

```
Index(['OWN_CAR_AGE', 'OCCUPATION_TYPE', 'EXT_SOURCE_1', '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'],
      dtype='object')
```

3. Removing the columns

Formula: app_data_nonull = app_data.drop(columns= drop_col)

Checking the shape of data to ensure that the columns has been removed.

```
app_data_nonull.shape
```

```
(49999, 72)
```

4. Removing the Columns that we do no require for this analysis.

Formula:

app_data_nonull.drop(columns = ['FLAG_MOBIL', 'FLAG_EMP_PHONE', 'FLAG_WORK_PHONE', 'FLAG_CONT_MOBILE', 'FLAG_PHONE', 'FLAG_EMAIL','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','EXT_SOURCE_2', 'EXT_SOURCE_3', 'OBS_30_CNT_SOCIAL_CIRCLE', 'DEF_30_CNT_SOCIAL_CIRCLE', 'OBS_60_CNT_SOCIAL_CIRCLE', 'DEF_60_CNT_SOCIAL_CIRCLE','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'], inplace = True)

Ensuring that columns has been removed.

```
app_data_nonull.shape
```

```
(49999, 34)
```

Repeat the following steps for prev_data as well to clean that data.

Dealing with the columns that have null values less the 30%.

1. Finding out the null values:

Formula: app_data_nonull.isnull().sum().sort_values(ascending = True)

```
DAYS_LAST_PHONE_CHANGE            1
AMT_ANNUITY                       1
CNT_FAM_MEMBERS                   1
AMT_GOODS_PRICE                  38
NAME_TYPE_SUITE                 192
AMT_REQ_CREDIT_BUREAU_HOUR     6734
AMT_REQ_CREDIT_BUREAU_DAY      6734
AMT_REQ_CREDIT_BUREAU_WEEK     6734
AMT_REQ_CREDIT_BUREAU_MON      6734
AMT_REQ_CREDIT_BUREAU_QRT      6734
AMT_REQ_CREDIT_BUREAU_YEAR     6734
dtype: int64
```

2. Columns that have null values less the 1 or 2 % we can replace those null values with mean, median or mode and the columns that have more than that we will drop those rows.

- for columns DAY_LAST_PHONE_CHARGE, AMT_ANNUITY, CNT_FAM_MEMBER, AMT_GOODS_PRICE AND NAME_TYPE_SUITE we will replace the null values with mean, median or mode and for rest of the columns we will drop the rows.

Formula:

```
## dealing with DAYS_LAST_PHONE_CHANGE
app_data_nonull['DAYS_LAST_PHONE_CHANGE'].describe()
```

```
count    49998.000000
mean       964.296172
std        829.485574
min          0.000000
25%        270.000000
50%        755.000000
75%       1573.000000
max       4002.000000
Name: DAYS_LAST_PHONE_CHANGE, dtype: float64
```

```
# filling the null with mean
app_data_nonull['DAYS_LAST_PHONE_CHANGE'].fillna(964, inplace = True)
```

```
## dealing with AMT_ANNUITY
app_data_nonull['AMT_ANNUITY'].describe()
```

```
count     49998.000000
mean      27107.377355
std       14562.944435
min        2052.000000
25%       16456.500000
50%       24939.000000
75%       34596.000000
max      258025.500000
Name: AMT_ANNUITY, dtype: float64
```

```
app_data_nonull['AMT_ANNUITY'].median()
```

```
24939.0
```

```
# replacing null with median
app_data_nonull['AMT_ANNUITY'].fillna(24939.0, inplace = True)
```

```
## dealing with CNT_FAM_MEMBERS
app_data_nonull['CNT_FAM_MEMBERS'].describe()
```

```
count    49998.000000
mean         2.158946
std          0.911332
min          1.000000
25%          2.000000
50%          2.000000
75%          3.000000
max         13.000000
Name: CNT_FAM_MEMBERS, dtype: float64
```

```
app_data_nonull['CNT_FAM_MEMBERS'].mode()
```

```
0    2.0
Name: CNT_FAM_MEMBERS, dtype: float64
```

```
# replacing the null with mode
app_data_nonull['CNT_FAM_MEMBERS'].fillna(2, inplace = True)
```

```
## dealing with AMT_GOODS_PRICE
app_data_nonull['AMT_GOODS_PRICE'].describe()
```

```
count    4.996100e+04
mean     5.390600e+05
std      3.698533e+05
min      4.500000e+04
25%      2.385000e+05
50%      4.500000e+05
75%      6.795000e+05
max      4.050000e+06
Name: AMT_GOODS_PRICE, dtype: float64
```

```
# replacing null with mean
m = app_data_nonull['AMT_GOODS_PRICE'].mean()
app_data_nonull['AMT_GOODS_PRICE'].fillna(m, inplace = True)
```

```
# Dealing with NAME_TYPE_SUITE
app_data_nonull['NAME_TYPE_SUITE'].head()
```

```
0    Unaccompanied
1           Family
2    Unaccompanied
3    Unaccompanied
4    Unaccompanied
Name: NAME_TYPE_SUITE, dtype: object
```

```
app_data_nonull['NAME_TYPE_SUITE'].mode()
```

```
0    Unaccompanied
Name: NAME_TYPE_SUITE, dtype: object
```

```
# Replacing null with mode(Unaccompanied)
app_data_nonull['NAME_TYPE_SUITE'].fillna('Unaccompanied', inplace = True)
```

- Removing rows for rest of the columns:

```
# droping rows which consist NA
app_data_nonull.dropna(inplace = True)
```

Repeat the following steps for prev_data as well to clean that data.

Identifying Outliers in the Dataset

<u>Using a Box Plot to detect outliers.</u>

** Creating a function to detect outliers using box plot.*

def box_plot(df, col):

df.boxplot(column = [col])

plt.grid(False)

plt.show()

** Creating a function to detect outliers.*

def outliers(df, col):

sorted(df[col])

Q1 = df[col].quantile(0.25)

Q3 = df[col].quantile(0.75)

IQR = Q3 - Q1

Lower_bound = Q1 - 1.5 * IQR

Upper_bound = Q3 + 1.5 * IQR

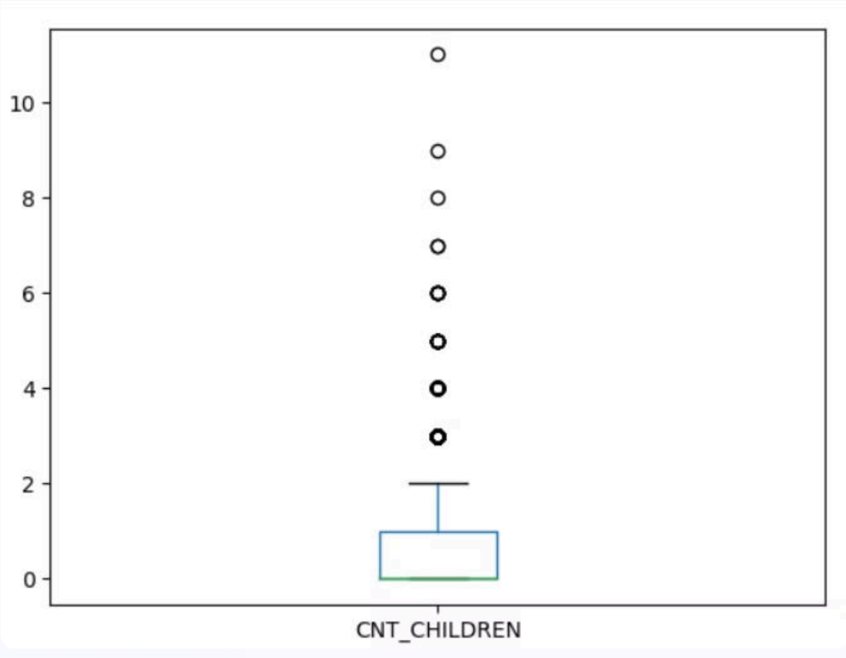
df['OL'] = np.where(df[col]< Lower_bound, 0,

np.where(df[col]> Upper_bound,0,1))

return df['OL']

Output:

```
1 # creating a box plot of all numarical columns to detect outliers
2 box_plot(app_data_nonull,'CNT_CHILDREN')
```



- As we can see that there are outliers present in the columns using Box Plot
- Now we will see the count of outliers by calling 2 function.

```
]  1 # Finding out the number of Outlier present in CNT_CHILDREN
   2 count_OL = app_data_nonull[outliers(app_data_nonull,'CNT_CHILDREN') == 0]
   3 count_OL['OL'].head()

   91     0
   92     0
   144    0
   180    0
   182    0
   Name: OL, dtype: int64

]  1 count_OL['OL'].count()

   629
```

- we have total 629 outliers present in the CNT_CHILDREN Columns.
- If you want u can remove the outliers using python, drop function.
- But, for now we will this as it is.

By calling the same function that we have created we have to find out outliers for all the columns that contain integer or float values.

•

Identifying the data imbalance

* Finding out the percentage of loan approved and denied.

Formula:

count = app_data_nonull['TARGET'].count()

perc_1 = len(app_data_nonull[app_data_nonull['TARGET']==1])/count*100

perc_0 = len(app_data_nonull[app_data_nonull['TARGET']==0])/count*100

print('0 = ', round((perc_0),2),'%')

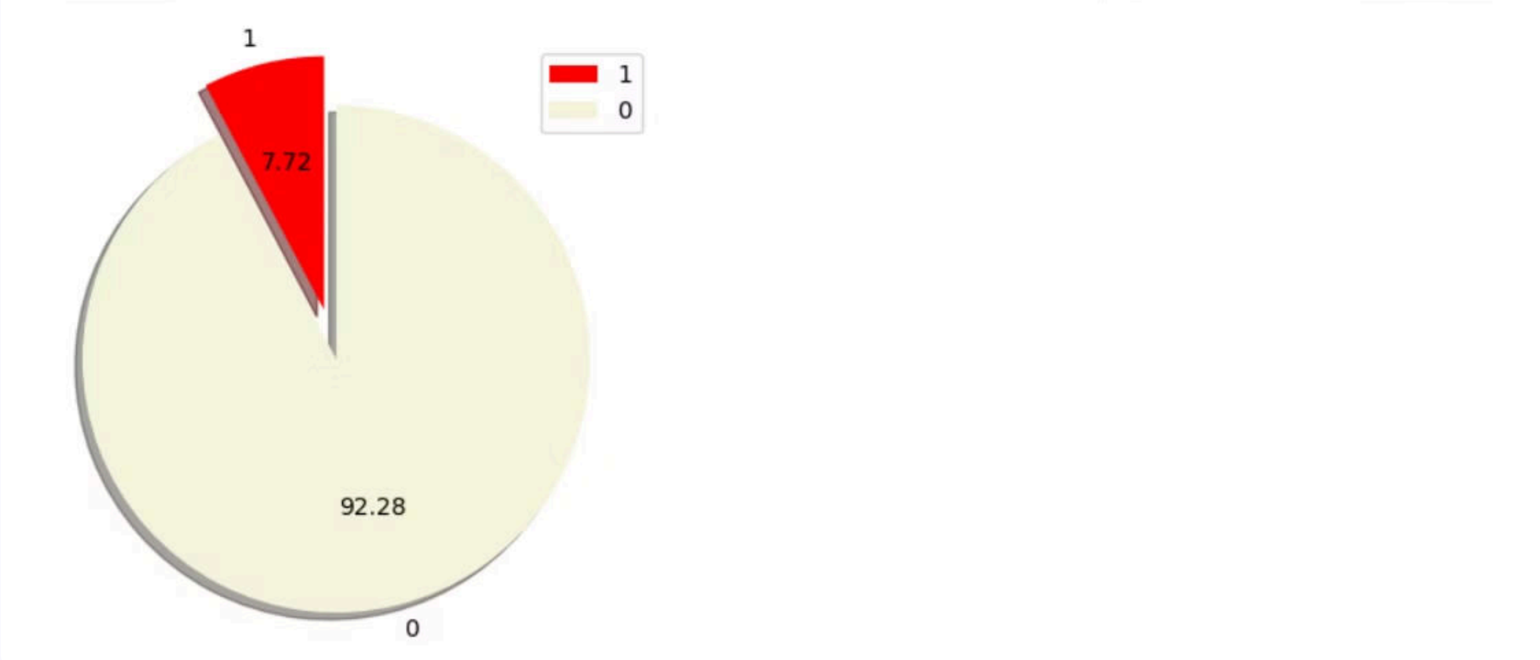
print('1 =' ,round((perc_1),2),'%')

```
0 =  92.28 %
1 = 7.72 %
```

0 stand for approved and 1 stand for denied.

* Creating a pie diagram to show the percentage.

```
1 pie = (perc_1,perc_0)
2 Label = ['1','0']
3 colour = ['r','beige']
4 plt.pie(pie, labels = Label, explode = [0.2,0.0], shadow = True , colors = colour,startangle=90, autopct='%.2f' )
5 plt.legend()
6 plt.show()
```



* Finding out the Imbalance Ratio

```
1 imbalance_ratio = len(app_data_nonull[app_data_nonull['TARGET']==0])/len(app_data_nonull[app_data_nonull['TARGET']==1])
2 "Imbalance_Ratio : 1:{:.2f}".format(imbalance_ratio)

'Imbalance_Ratio : 1:11.95'
```

Findings: I found out that on every loan application which are getting denied we have approximately 12 loans application which are getting accepted.

- **The data set exhibits data imbalance because the no. of loans application that are getting accepted far exceeds the no. of application that are getting rejected.**
- **this can pose a challenge to the ML algorithm because they may become biased towards the majority data and can have difficulties in accurately predicting the minority data.**

Univariate, Segmented Univariate, and Bivariate Analysis with Excel

- Before starting the analysis in Excel, I have downloaded the cleaned data from python.

Formula: app_data_nonull.to_csv('Application_data.csv')

- To perform *Univariate, Segmented Univariate, and Bivariate Analysis I have created a Bins for the columns CNT_FAM_MEMBER and AMT_INCOME_ TOTAL*
- Using CNT_FAM_MEMBER we will try to identify the pattern does size of family affect the ability of person to repay the loan.
- Using the AMT_INCOME_TOTAL we will try to identify that does income affect the loan repayment.

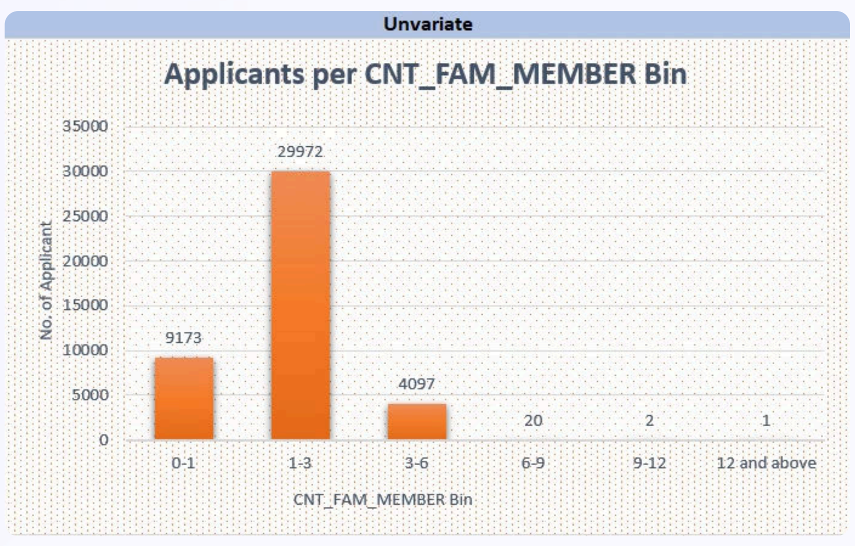
BIN OUTPUT:

CNT_FAM_MEMBER Bin	Applicants	TARGET(0)	TARGET(1)
0-1	9173	8446	727
1-3	29972	27723	2249
3-6	4097	3738	359
6-9	20	17	3
9-12	2	1	1
12 and above	1	0	1

AMT_INCOME_TOTAL Bin	Average of AMT_CREDIT
0 - 27K	10757.25
27k - 198K	23594.65
198K - 369K	34579.93
369K - 540K	44485.13
540K - 711K	48997.53
711K - 882K	59856.42
882K - 1.05M	57836.63
1.05M - 1.22M	55113.75
1.22M - 1.40M	42635.5
1.40M - 1.74M	45000
1.74M and Above	53929.69

- Conducted univariate analysis using Excel functions like COUNT, AVERAGE, and MEDIAN to understand the distribution of Applicant per Bin of CNT_FAM_MEMBER

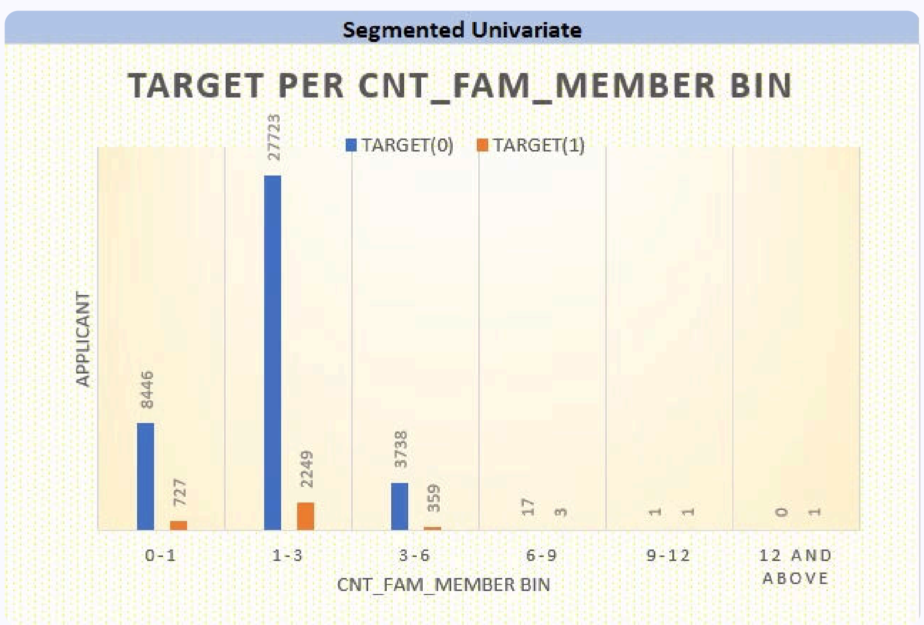
Output:



Finding: I found out that an individual with the family size range in between 1 to 3 are applying for a loan are more than the other ranges.

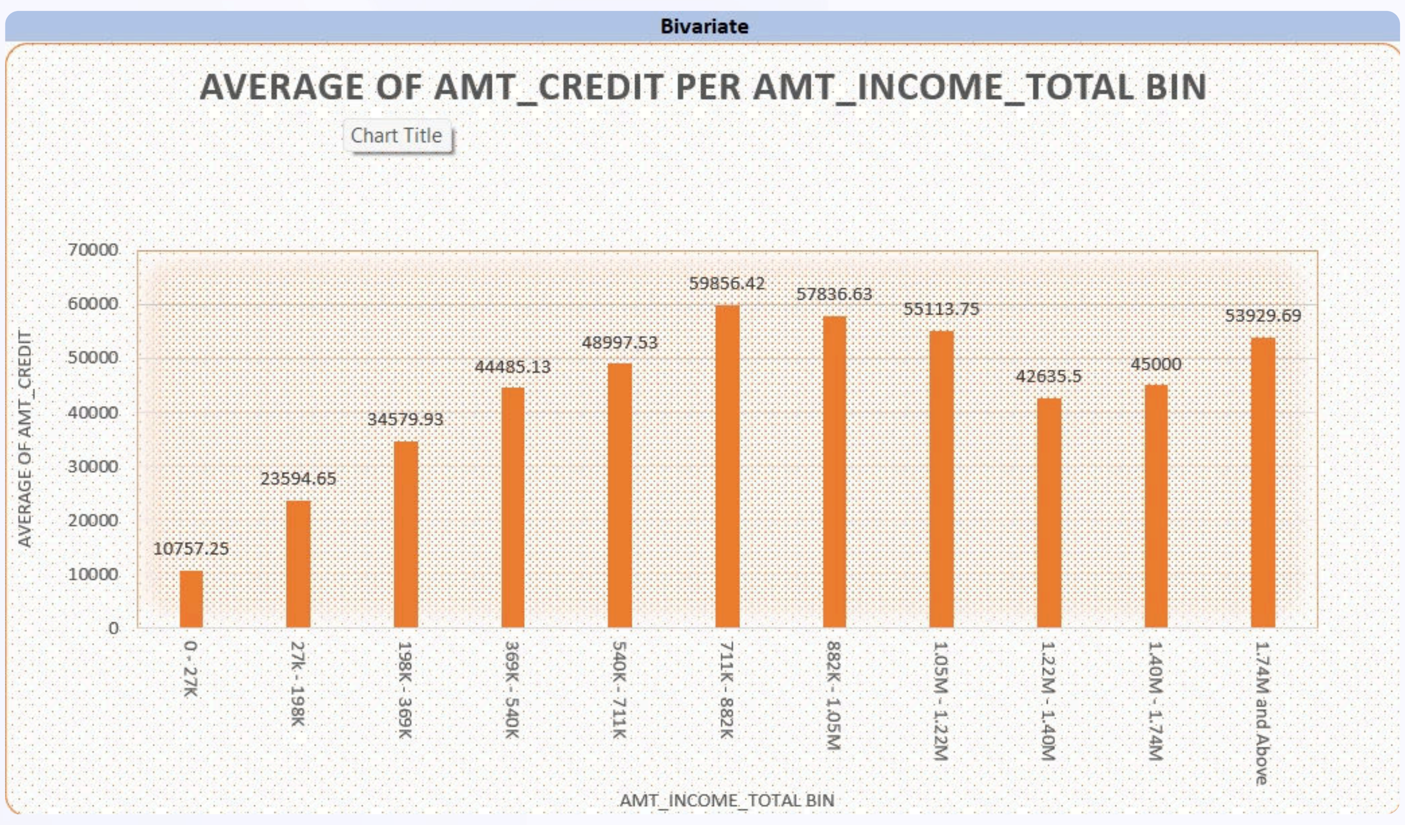
- Performed segmented univariate analysis using Excel features like filters and pivot tables to compare the loan approved and denied per CNT_FAM_MEMBER Bin.

Output:



Finding: Most loans are approved for an individual with the family size range in between 1 to 3 and most loans are denied for an individual with the same size of family range.

- Conducted bivariate analysis using pivot tables in Excel to explore relationships between Average of AMT_CREDIT and AMT_INCOME_TOTAL Bin.
- **Output:**



Finding: The individual with the income ranges in between 711k to 882k are get the highest amount credit.

Correlation Analysis

1 Correlation Identification

Identify top correlations between variables and loan default for each segment.

Calculated correlation coefficients between variables and the target variable within each segment using Excel functions like CORREL.

Output:

Correlation of Applicant with payment made on time

	CNT_FAM_MEMBER	AMT_INCOME_TOTAL	AMT_CREDIT	AMT_ANNUITY	REGION_POPULATION_RELATIVE	DAYS_EMPLOYED	REGION_RATING_CLIENT
CNT_FAM_MEMBER	1.00	0.04	0.06	0.08	-0.02	-0.24	0.02
AMT_INCOME_TOTAL	0.04	1.00	0.37	0.45	0.18	-0.16	-0.21
AMT_CREDIT	0.06	0.37	1.00	0.77	0.10	-0.08	-0.10
AMT_ANNUITY	0.08	0.45	0.77	1.00	0.12	-0.11	-0.13
REGION_POPULATION_RELATIVE	-0.02	0.18	0.10	0.12	1.00	-0.00	-0.54
DAYS_EMPLOYED	-0.24	-0.16	-0.08	-0.11	-0.00	1.00	0.04
REGION_RATING_CLIENT	0.02	-0.21	-0.10	-0.13	-0.54	0.04	1.00

Correlation of Applicant with payment Difficulties

	CNT_FAM_MEMBER	AMT_INCOME_TOTAL	AMT_CREDIT	AMT_ANNUITY	REGION_POPULATION_RELATIVE	DAYS_EMPLOYED	REGION_RATING_CLIENT
CNT_FAM_MEMBER	1.00	0.01	0.06	0.08	-0.03	-0.18	0.07
AMT_INCOME_TOTAL	0.01	1.00	0.01	0.02	-0.01	-0.01	-0.01
AMT_CREDIT	0.06	0.01	1.00	0.74	0.08	-0.00	-0.05
AMT_ANNUITY	0.08	0.02	0.74	1.00	0.07	-0.09	-0.05
REGION_POPULATION_RELATIVE	-0.03	-0.01	0.08	0.07	1.00	0.00	-0.42
DAYS_EMPLOYED	-0.18	-0.01	-0.00	-0.09	0.00	1.00	-0.01
REGION_RATING_CLIENT	0.07	-0.01	-0.05	-0.05	-0.42	-0.01	1.00

Insights from EDA and Data Analysis

Valuable Insights

Gain valuable insights into the factors influencing loan default using a combination of Python and Excel for analysis.

Data Interpretation

In-depth data interpretation highlighted the importance of a hybrid approach combining different tools and techniques.

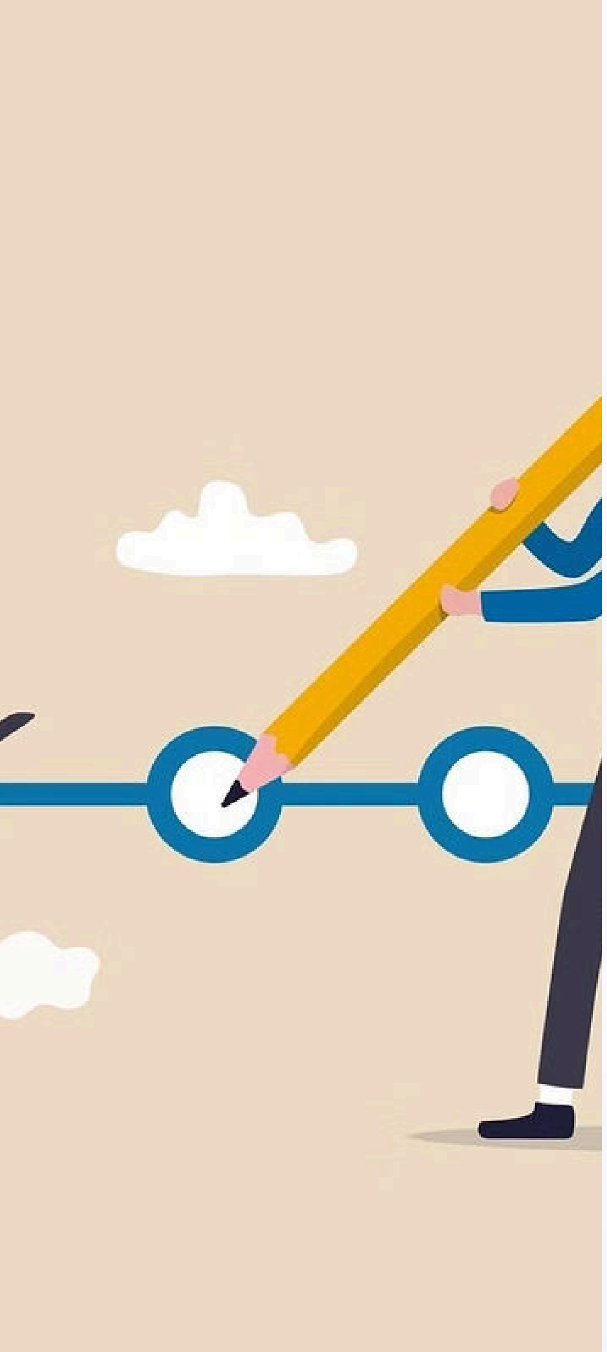
Technology Stack Utilized

Python Usage

Leverage Python for EDA, data cleaning, outlier identification, and data imbalance detection.

Excel Tools

Utilize Excel for univariate, segmented univariate, bivariate analysis, and identifying top correlations.



Result and Impact

1 **Project Outcome**

The project successfully addressed the key objectives of identifying patterns related to loan default risk.

2 **Insightful Decisions**

Valuable insights inform decision-making processes, enabling more informed loan approval and risk assessment strategies.

The Importance of Data Analysis

2

Effective Analysis

By leveraging Python for advanced data analysis and Excel for data visualization, valuable insights were gained.

1

Hybrid Approach

The project highlighted the importance of a hybrid approach combining different tools and techniques for analyzing complex datasets.

Linkes

- [Python Source code]
- [Excel File]
- [Video Presentation]