Bank Loan Case Analysis

BY

Veeravalli Naga Manohar

Project Description :-

- The project is about bank loan application data where we got to work with the data of loan applications which consists of the income of the client, no.of.people in the client's family, their education..etc.
- So here got to deal with risk analytics where we analyze the risk of giving loans to certain clients.
- By analyzing the client details we need to derive the conclusion that we need to provide the loan to that particular client.
- Using the analytical skills we have we need to analyze the data and provide useful insights like the correlation between the data, cleaning the data, and finding the outliers..etc.

Approach :-

- My approach for this project would be first of all analyzing the given data and going through the data types present in them.
- Going through each column of data like what that column's are about and what is the relevance of data to our analysis.
- Writing some relevant data points and checking the errors and null values in the data.
- Loading the required data into the jupyter notebook using the pandas' library.
- Studying the data or going the insights of the data using different functions in Python.

Tech - Stack Used :-

1. Python

- The one of major tech stack I have used in this project is Python programming language where the most of program is written in it.
- The execution of commands which are written in this programming language generally python is the most preferred language for analyzing huge data.
- This project can be done in excel itself but due to the data's high no.of columns and row, we are using the Python program for the analysis.

2. Jupyter NoteBook

- Jupyter Notebook is the very best web-based IDE which can be really useful for data science and data analysis projects.
- It is very much easy to analyze the data using jupyter notebook because of its interface and useability.
- Plotting the data in jupyter notebook is also very easy and convenient by using simple commands.

3. Excel

- Excel is an excellent GUI tool for data exploration and data manipulation.
- Excel is very widely used for data crunching and visualization of data easily.
- In this project, we have used excel for data exploration and going through the values of data.

Insights:-

- This project is about the bank loan application data where we need to go through the data of clients of the bank and we need to analyze this data to get useful and meaningful insights from it.
- There are various things we gain by completing this project like the correlation between the data, outliers present in the data, various analyses of the data like univariate analysis, bivariate analysis..etc
- By doing the visualization of data we get to know more about the data that we have like the income of clients, their total credit, and the type of house their live in.

Data Cleaning

In [1]:
```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

In [2]:
```
df = pd.read_csv('application_data.csv')
df1 = pd.read_csv('previous_application.csv')
```

In [4]:
```
data = pd.merge(df, df1, on='SK_ID_CURR', how='inner')
data
```

- Importing the libraries required for the data analysis so here I have used pandas, seaborn and matplotlib libraries.
- Pandas is widely used for data frames analysis and manipulating data frames.
- Matplotlib and Seaborn are widely used for plotting and visualizations.
- Assigning the data frame to a particular variable as **df** and **df1** by using a command called **pd.read_csv("file path").**
- Creating the new variable called **data** which consists of two data frames where we merged the two data frames by using a command called **pd. merge(dataframe1,dataframe2,on=" which column",how=" joint type")**
- So **data** variable will consist of two data frame's data.

Out[4]:

	SK_ID_CURR	TARGET	NAME_CONTRACT_TYPE_x	CODE_GENDER	FLAG_OWN_CAR	FLAG_OWN_REALTY	CNT_CHILDREN	AMT_INCOME_TOTAL	A
0	100002	1	Cash loans	M	N	Y	0	202500.0	
1	100003	0	Cash loans	F	N	N	0	270000.0	
2	100003	0	Cash loans	F	N	N	0	270000.0	
3	100003	0	Cash loans	F	N	N	0	270000.0	
4	100004	0	Revolving loans	M	Y	Y	0	67500.0	
...	...	...	...	...	...	...	...	...	
1413696	456255	0	Cash loans	F	N	N	0	157500.0	
1413697	456255	0	Cash loans	F	N	N	0	157500.0	
1413698	456255	0	Cash loans	F	N	N	0	157500.0	
1413699	456255	0	Cash loans	F	N	N	0	157500.0	
1413700	456255	0	Cash loans	F	N	N	0	157500.0	

1413701 rows × 158 columns

- So the criteria which I have chosen for cleaning the data is i am going to remove the column which contains the **null values %** greater than **50%.**
- Because those columns don't add any value to our analysis due to null values the analysis will be led in the wrong direction.

- The below code is used for the cleaning of the data.

```
In [5]: #calculate the percentage of null values in each column
columns_null = data.isnull().mean()*100
#saving the columns names which consists of null values percentage greater than 50%
drop_columns = columns_null[columns_null>50].index
#dropping the columns those null values percentage is more then 50%
data.drop(drop_columns,axis=1,inplace=True)
data
```

- Where the **columns_null** variable is used for saving the columns null_values % and we have used the **data.isnull().mean()*100** commands for finding the null_values % for every column in the dataset called **data**.
- After finding the null values in each column we got to remove the columns which are having the null_values % greater than 50%.
- So here we have assigned the variable called **drop_columns** where we are storing the columns which are null_values % is greater than 50% by using the command **columns_null[columns_null>50].index**.
- Where we take the variable called **columns_null** which consists of columns and their % null_values to each column in the data.
- Columns that null_value % is greater than 50% get stored in the variable called **drop_columns.**
- So finally by using the command called **data. drop(drop_columns,axis=1,inplace=True),** we remove the columns in which null_value % is greater than 50%.

1	100003	0	Cash loans	F	N	N	0	270000.0
2	100003	0	Cash loans	F	N	N	0	270000.0
3	100003	0	Cash loans	F	N	N	0	270000.0
4	100004	0	Revolving loans	M	Y	Y	0	67500.0
...	...	...	...	...	...	...	...	...
1413696	456255	0	Cash loans	F	N	N	0	157500.0
1413697	456255	0	Cash loans	F	N	N	0	157500.0
1413698	456255	0	Cash loans	F	N	N	0	157500.0
1413699	456255	0	Cash loans	F	N	N	0	157500.0
1413700	456255	0	Cash loans	F	N	N	0	157500.0

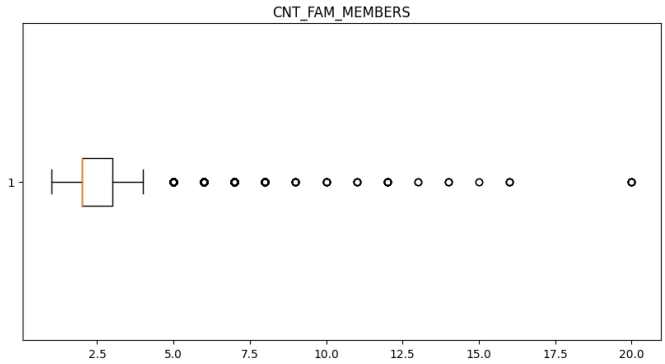
1413701 rows × 120 columns

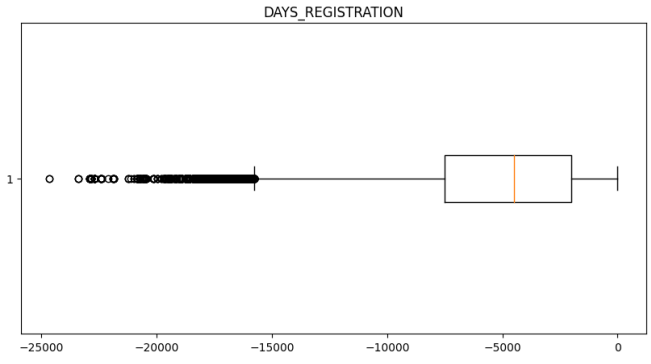
Outliers

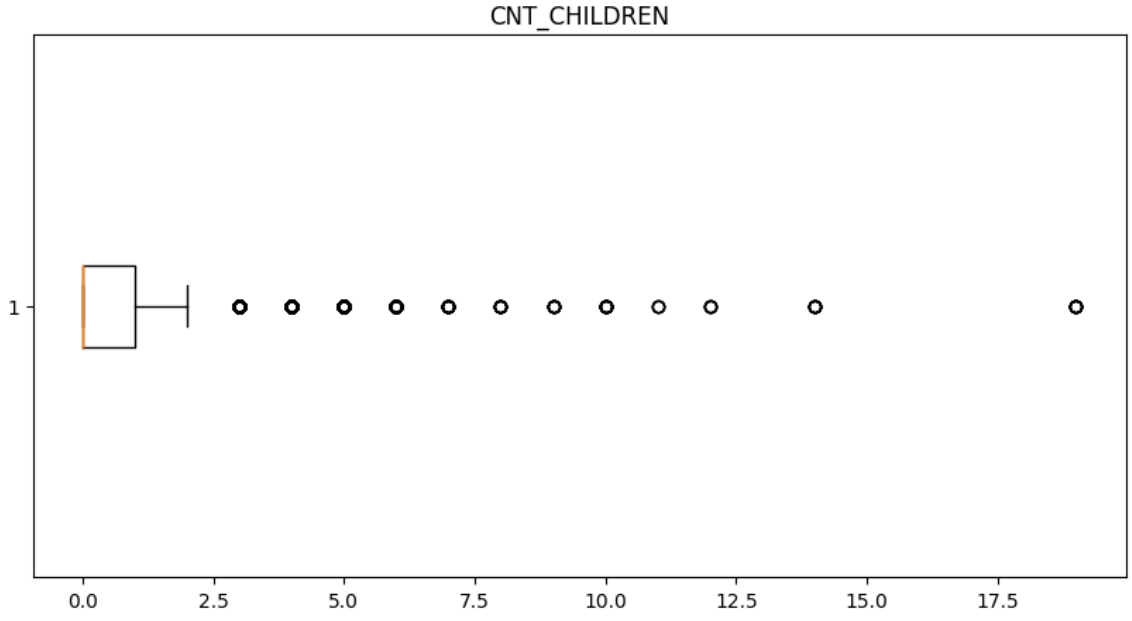
- Outliers are generally data points that are significantly different from the majority of data points those data points occur due to some unusual events.
- Due to the outlier's the data the analysis may be skewed due to the outliers in the data while finding the mean of the data sometimes, the results will be overestimated due to the outliers present in them.
- For finding the outlier's we can different methods like by using the z-score and quartile in this analysis (or) just plotting the values we can get the outliers in the data.
- So in this project, we are selecting the specific columns in which we will find the outliers.
- As we see in the above code we have selected a few column's in the dataset where we are finding the outlier's in them.

```
#columns for those we going to find  the outlier's in them
columns = ['CNT_FAM_MEMBERS','DAYS_EMPLOYED','DAYS_REGISTRATION','AMT_INCOME_TOTAL','AMT_ANNUITY_x','AMT_CREDIT_x','CNT_CHILDREN']

for col in columns:
    plt.figure(figsize=(10,5))
    plt.boxplot(data[col], vert=False)
    plt.title(col)
    plt.show()
```

- We have saved those column's in the variable called **columns.**
- By using the **for** loop we have plotted the outliers in data for that specific column using boxplots.
- The outliers are plotted in the form of box plots for that specific column so that visually we can see the outliers in the data.


CNT_FAM_MEMBERS
1
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0


DAYS_REGISTRATION
1
−25000
−20000
−15000
−10000
−5000
0

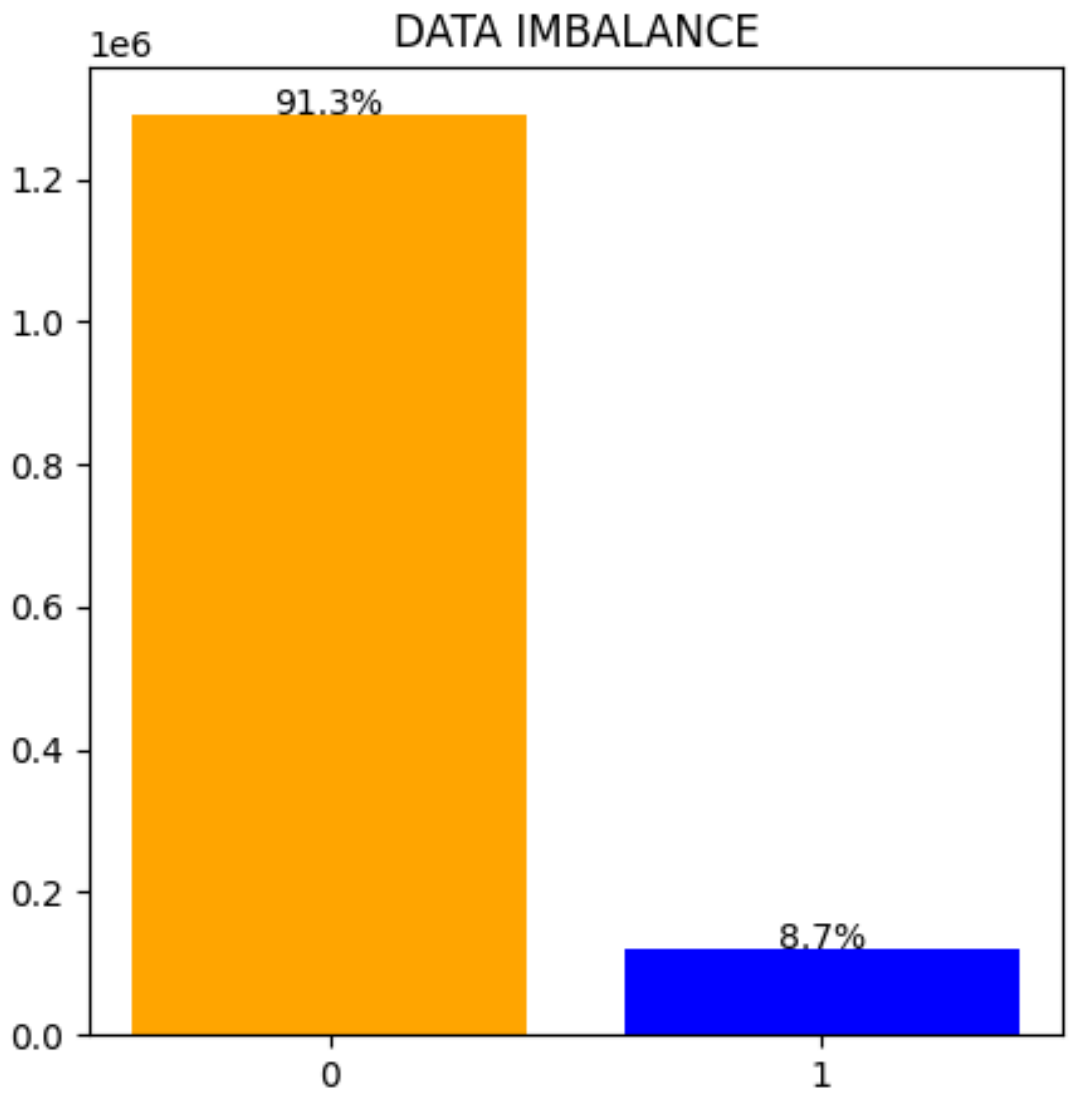

CNT_CHILDREN
1
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5

Data Imbalance

- Data imbalance is also known as class imbalance it is defined as where no.of observation's (or) instances belonging to one specific class are more where as the no.of observation's (or) instances belonging to another specific class are less This observation in the data is called as data imbalance.
- When it comes to our case we have a lot of clients with other cases and fewer clients with payment difficulties.
- Clients with other cases are represented with **0** and clients with payment difficulties are represented with **1** under the column of **TARGET.**

```
#target counts(counts no.of target's rows present in the dataset)
counts = data['TARGET'].value_counts()
colors = ['orange', 'blue']
plt.figure(figsize=(5,5))
plt.bar(counts.index,counts.values,color=colors)
plt.title("DATA IMBALANCE")
plt.xticks(counts.index)


total = sum(counts)
for i, count in enumerate(counts):
    percentage = count / total * 100
    plt.text(i, count + 100, f'{percentage:.1f}%', ha='center')

plt.show()

#for finding the ratio data imbalance between the target[0] and target[1]

print('Ratio of data imbalance:', counts[0]/counts[1])
```

- The above code is used for finding the data imbalance in our data so we can see that we have used the **counts** variable for saving the count of TARGET values in them.
- For plotting we have used the matplotlib library functions and those functions are **plt. figure()** which is used for deciding the plot length.
- **plt. bar(x-axis, y-axis, color)** is used for which data to shown on the plot of the x-axis, y-axis, and color in the plot's in our case we have used the **counts. index[0,1]** as our **x-axis** and **counts. values** are the **y-axis.**
- **plt. title()** to place the title for the plot and for the loop used for showing the percentage on the bars of the plot.




```
Ratio of data imbalance: 10.553620464203988
```

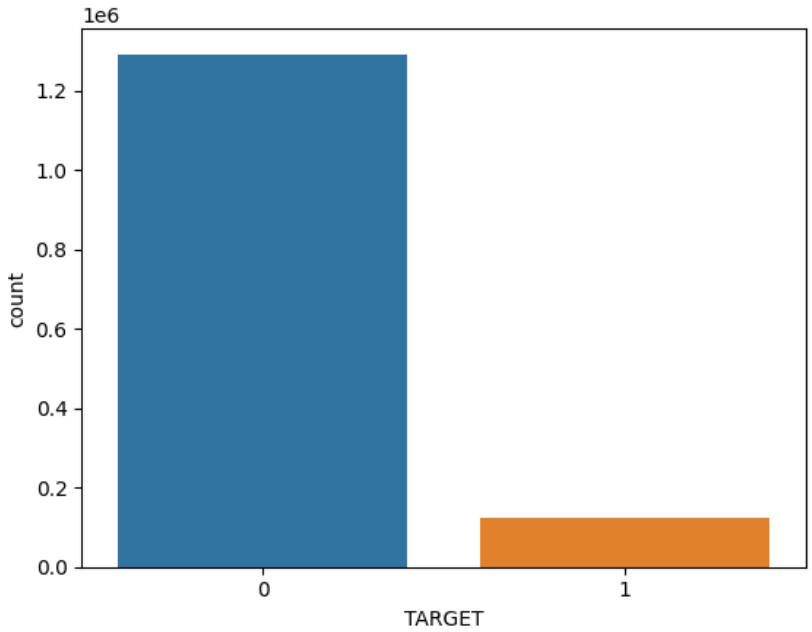
- The above plot shows the data imbalance in our data with respective to the column **TARGET.**
- Where we clearly see that clients with other cases are 10 times more than the clients with payment difficulties.

Univariate analysis

- It is defined as an analysis in which statistical analysis of a single variable at a time without considering its relation with any other variables.
- The univariate analysis explores the properties and behavior of a single variable without considering its relationship with other variables.
- This analysis includes the measure of central tendencies such as mean. median and mode.
- Univariate analysis is commonly used in many fields of social sciences, marketing,finance, and healthcare industries.

```
# Conduct univariate analysis on the target variable
sns.countplot(x='TARGET', data=data)
data.describe()
```

- Here we have performed the analysis on the **TARGET** column of data where we have plotted the data of the **TARGET** column only for the analysis by using the seaborn library function we have plotted the data column called **TARGET.**



Segmented univariate analysis

- Segmented univariate analysis is defined as statistical analysis performed on one single variable by considering the subgroups within the data.
- In our case, we are performing the analysis on the **total income of the client** by considering the subset called **TARGET.**

```
#  income bins for plotting
income_bins = [(25000, 50000), (50000, 75000), (75000, 100000), (100000, 125000), (125000, 150000), (150000, 175000),
               (175000, 200000), (200000, 225000), (225000, 250000), (250000, 275000), (275000, 300000),
               (300000, 325000), (325000, 350000), (350000, 375000), (375000, 400000), (400000, 425000),
               (425000, 450000), (450000, 475000), (475000, 500000)]
```

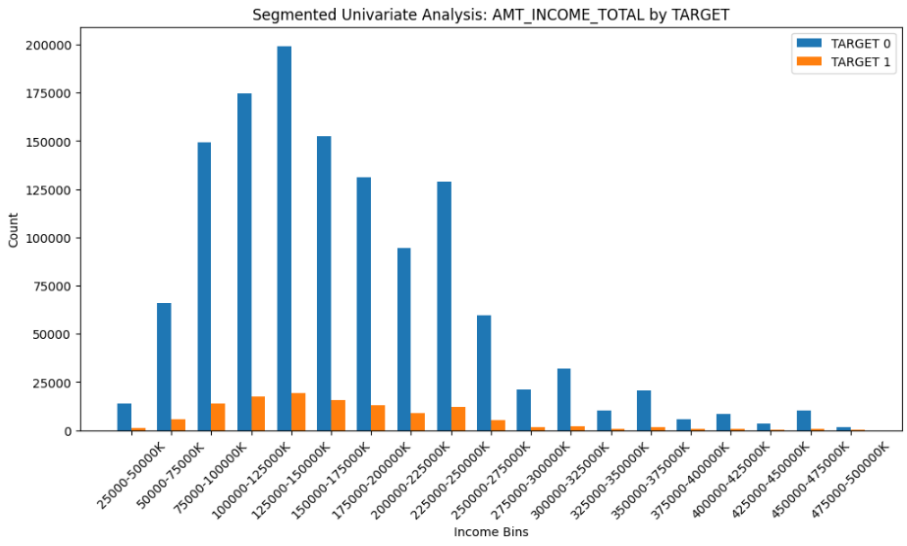
- First of all, we are deciding the income range of clients starting from 25k to 500k with a difference of 25k and storing them in a variable called **income_bins.**

```
# lists to store the counts for each income bin and target group
target_0_counts = []
target_1_counts = []
bin_labels = []

# this loop iterate throught each income bin
for bin_start, bin_end in income_bins:
    # sort  the data based on the income bin
    filtered_data = data[(data['AMT_INCOME_TOTAL'] >= bin_start) & (data['AMT_INCOME_TOTAL'] < bin_end)]

    # Calculating the counts for each TARGET either 0 (or) 1
    target_0_count = filtered_data[filtered_data['TARGET'] == 0].shape[0]
    target_1_count = filtered_data[filtered_data['TARGET'] == 1].shape[0]

    # Store the counts and bin labels
    target_0_counts.append(target_0_count)
    target_1_counts.append(target_1_count)
    bin_labels.append(f'{bin_start}-{bin_end}K')
```

- The above shows that there are three lists that are storing the three different data those lists are **target_0_counts** which stores the no.of 0 and are present in **TARGET**, **target_1_counts** which stores the no.of 1 and are present in **TARGET,** and **bin_labels** which stores the no.of target's present in that particular **income_bin** range.
- By using the **for** loop we have sorted the data which we need for plotting and saved those data in the list called **bin_labels**.

```
# width of each bar for plotting
bar_width = 0.35

positions = np.arange(len(bin_labels))


# Plot of the clustered column chart
plt.figure(figsize=(12, 6))
plt.bar(positions, target_0_counts, width=bar_width, label='TARGET 0')
plt.bar(positions + bar_width, target_1_counts, width=bar_width, label='TARGET 1')
plt.xlabel('Income Bins')
plt.ylabel('Count')
plt.title('Segmented Univariate Analysis: AMT_INCOME_TOTAL by TARGET')
plt.xticks(positions + bar_width / 2, bin_labels, rotation=45)
plt.legend()
plt.show()
```

- When it comes to the plotting we have decided the bar width of the plot by assigning the bar width to the variable called **bar_width.**
- After that, we used a library called **numpy** to arrange the **bin_labels** list in ascending order, and after that, we used the **matplotlib** library function to plot the data in clustered column style.



- By performing this analysis we got to know that the majority of clients draw the income of 1lac - 2lac.
- Another insight we got to know from it is that very few clients with payment difficulties[1] draw an income of above 5lac.

Bivariate analysis

- Bivariate analysis is defined as the relationship (or) association between the two variables in the dataset.
- It is the study that involves identifying the pattern (or) behavior between the two variables and how they are related to each other.
- Finding the correlation between them whether positive or negative and this analysis is widely used in various fields like marketing, finance, and healthcare.

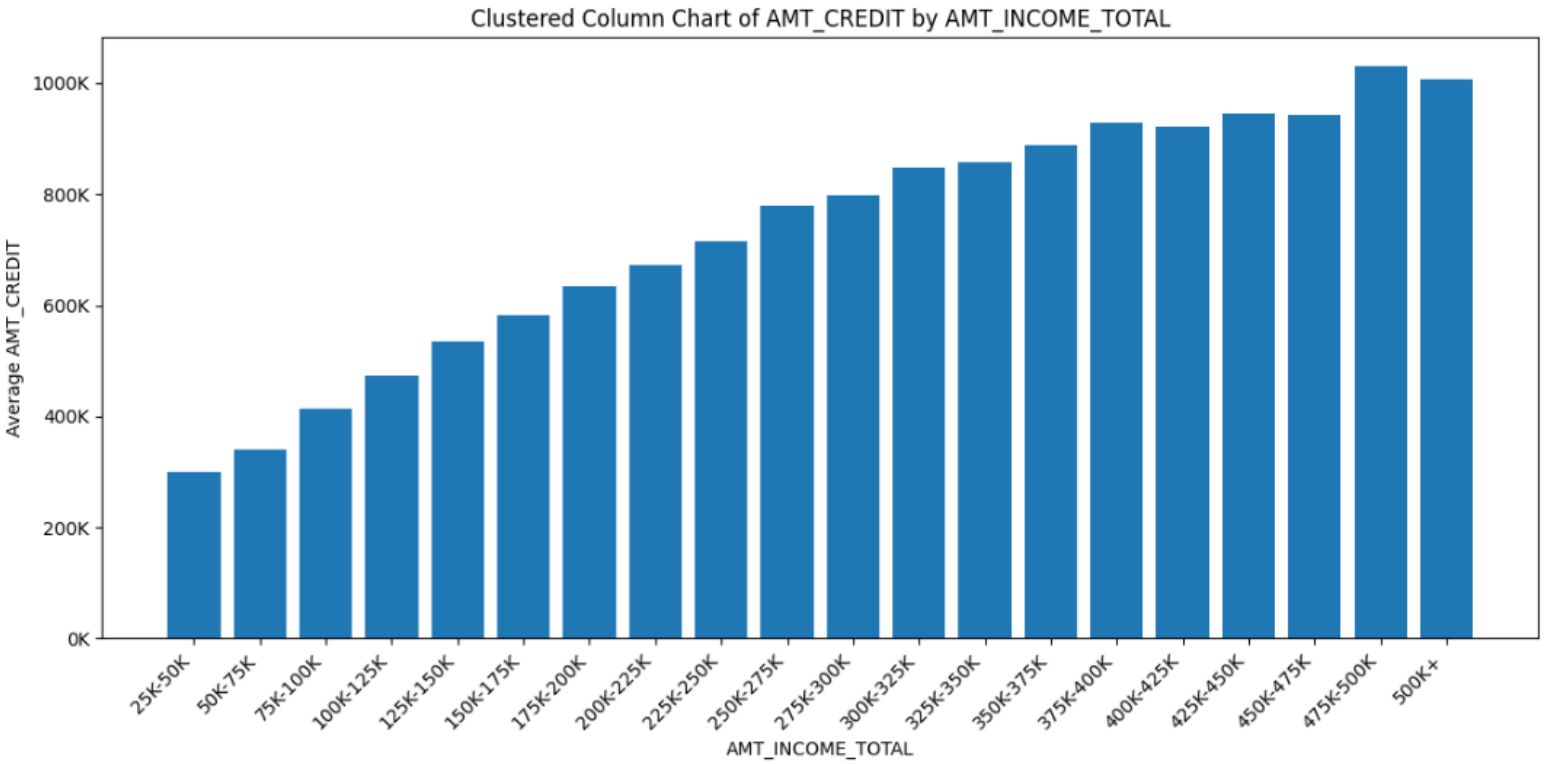
```
# Create bins for 'AMT_INCOME_TOTAL'
bin_labels = ['25K-50K', '50K-75K', '75K-100K', '100K-125K', '125K-150K', '150K-175K', '175K-200K', '200K-225K', '225K-250K', '250K-275K', '275K-300K', '300K-325K', '325K-350K', '350K-375K', '
bin_edges = [25000, 50000, 75000, 100000, 125000, 150000, 175000, 200000, 225000, 250000, 275000, 300000, 325000, 350000, 375000, 400000, 425000, 450000, 475000, 500000, float('inf')]
data['INCOME_BIN'] = pd.cut(data['AMT_INCOME_TOTAL'], bins=bin_edges, labels=bin_labels, right=False)
```

- The above code shows the **bin_labels** where we store the income ranges of the clients starting from 25k to 500k with a difference of 25k for each.
- By using the panda's library function **pd. cut()** we have categorized the data according to the bin's stored values in the data frame's column called **INCOME_BIN.**

```
# Calculate average 'AMT_CREDIT' for each income bin
grouped_data = data.groupby('INCOME_BIN')['AMT_CREDIT_x'].mean()

# Plotting the clustered column chart
plt.figure(figsize=(12, 6))
x = np.arange(len(bin_labels))
plt.bar(x, grouped_data, width=0.8)
plt.xlabel('AMT_INCOME_TOTAL')
plt.ylabel('Average AMT_CREDIT')
plt.title('Clustered Column Chart of AMT_CREDIT by AMT_INCOME_TOTAL')
plt.xticks(x, bin_labels, rotation=45, ha='right')

# Set y-axis ticks and labels
y_ticks = np.arange(0, grouped_data.max() + 1, 200000)
y_labels = [f'{round(val/1000):.0f}K' for val in y_ticks]
plt.yticks(y_ticks, y_labels)

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

- **Grouped_data** in the above code is used for storing the y-axis data i.e. credit provided to the client by the company where it is grouped by the **INCOME_BIN** an average of those credits according to the income bin we have provided.

- The below code is used for the plotting of the data where we have used the matplotlib functions to plot the data.
- Where we mention the axis data and other remaining data that needed to be plotted on the visualization of bivariate analysis.



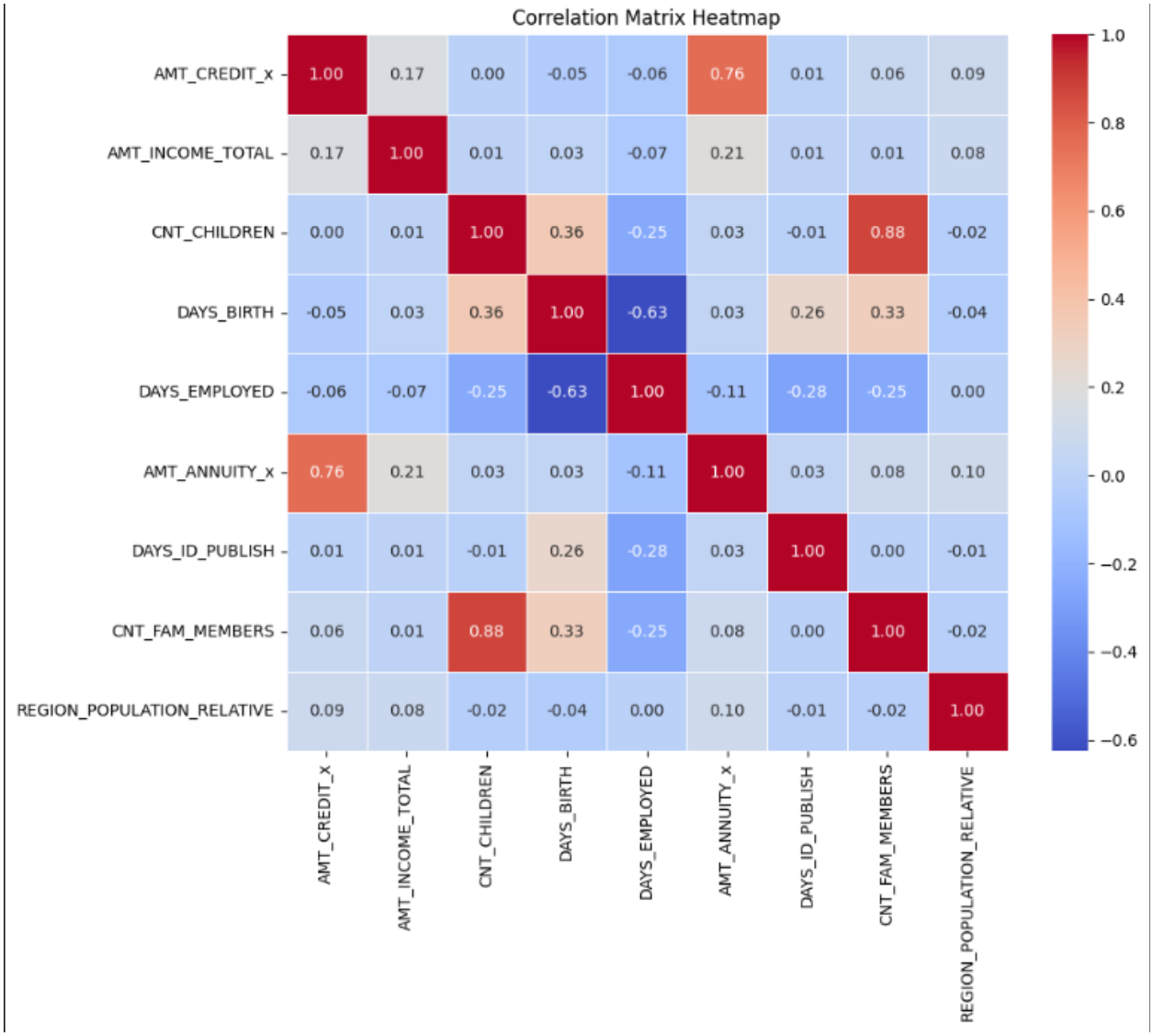
- From the above chart, we clearly understand that the credit provided by the company is directly related to the income of the client.
- Client's with higher income are getting higher credit from the company it is clearly visible that with an increase in the income of clients, the credit provided by the company also increases.
- So we can conclude that if the client has a higher income then the higher the credit client can get from the company so this is the conclusion that we derived from the bivariate analysis of the data between two different variables available in the data.

Correlation

- It is defined as it is a statistical measure that quantifies the relation (or) between two (or) more variables present in the data it also indicates the degree of the correlation between that variables to each other.
- Correlation ranges from -1 to 1 and values closer to -1 (or) 1 indicate a stronger correlation and values closer to 0 indicate a weaker correlation.
- There are three types of correlations are
- 1. **A positive correlation** is a positive value of correlation where one variable increases and another variable also increases.
- 2. **A negative correlation** is a negative value of correlation where one variable increases and another variable also decreases.
- 3.**No correlation** is where the correlation value is closer to 0 which indicates that the two variables are less related to each other (or) tend to impact each other.

```
# columns for correlation analysis
columns_of_interest = ['AMT_CREDIT_x', 'AMT_INCOME_TOTAL', 'CNT_CHILDREN','DAYS_BIRTH','DAYS_EMPLOYED','AMT_ANNUITY_x','DAYS_ID_PUBLISH','CNT_FAM_MEMBERS','REGION_POPULATION_RELATIVE']

#  data with the selected columns
selected_data = data[columns_of_interest]

# Calculating the correlation matrix
correlation_matrix = selected_data.corr()


# Ploting the heatmap for correlation data
plt.figure(figsize=(10, 8))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', fmt=".2f", linewidths=0.5)
plt.title('Correlation Matrix Heatmap')
plt.show()
```

- The above code is used for finding the correlation between the data of a few selected columns in the data.
- Where we have **columns_of_interest** variable to save the name of the columns in the data which columns we will be using to find the correlation between them.
- Fetching those columns' data and saving that data in a data frame called **selected_data** and saving that correlation data in a variable called **correlation_matrix.**
- Where we have used the **corr()** function to find the correlation between the data and for plotting that data in the form of a heatmap we have seaborn library functions to plot.
- Where for plotting the heatmap we have used the **sns. heatmap()** function of seaborn library.



- The above heatmap of the data is specified in the data from the data we can clearly observe that the amount of credit(**AMT_CREDIT_x**) and amount of annuity(**AMT_ANNUITY_x**) have the strongest correlation among them the correlation value is **0.76**.
- Another stronger correlation we can observe is between the no.of family members(**CNT_FAM_MEMBERS**) and no.of children(**CNT_CHILDREN**) but we can conclude that it is correct because the in the family members children are also included.
- Weaker correlation in the data we can observe is the correlation between the days of birth(**DAYS_BIRTH**) and days of employed(**DAYS_EMPLOYED**) the correlation value is -0.63.
- So these are conclusions we can derive from that heatmap.

Visualization

- Visualization is a great method to analyze the data and representation of data to understand it in a simple form.
- Visualization can be done in many methods to it and a lot more plot types in it plot the data and understand it a better way.
- So here are plotting the two data types present in the dataset are
- 1. Categorical Data
- 2.Numerical Data

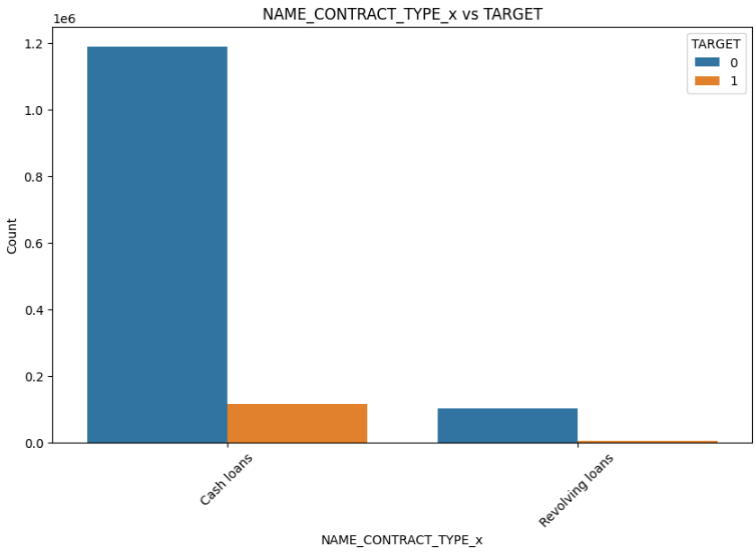
1. Categorical Data

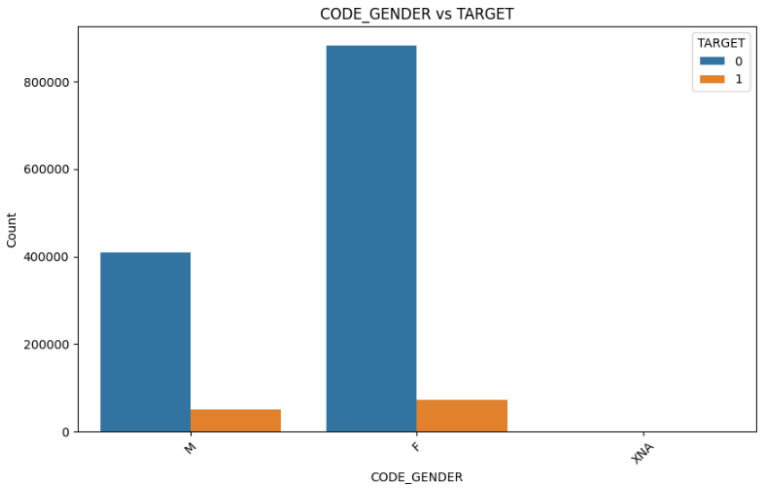
- It is defined as data that consists of a limited no.of categories in them and basically those data are non-numerical data so there will be segregation of data based on non-numerical's it is also called qualitative data.
- For the visual representation of categorical data we need to count the occurrence of those non-numerical values in the dataset and plot them.

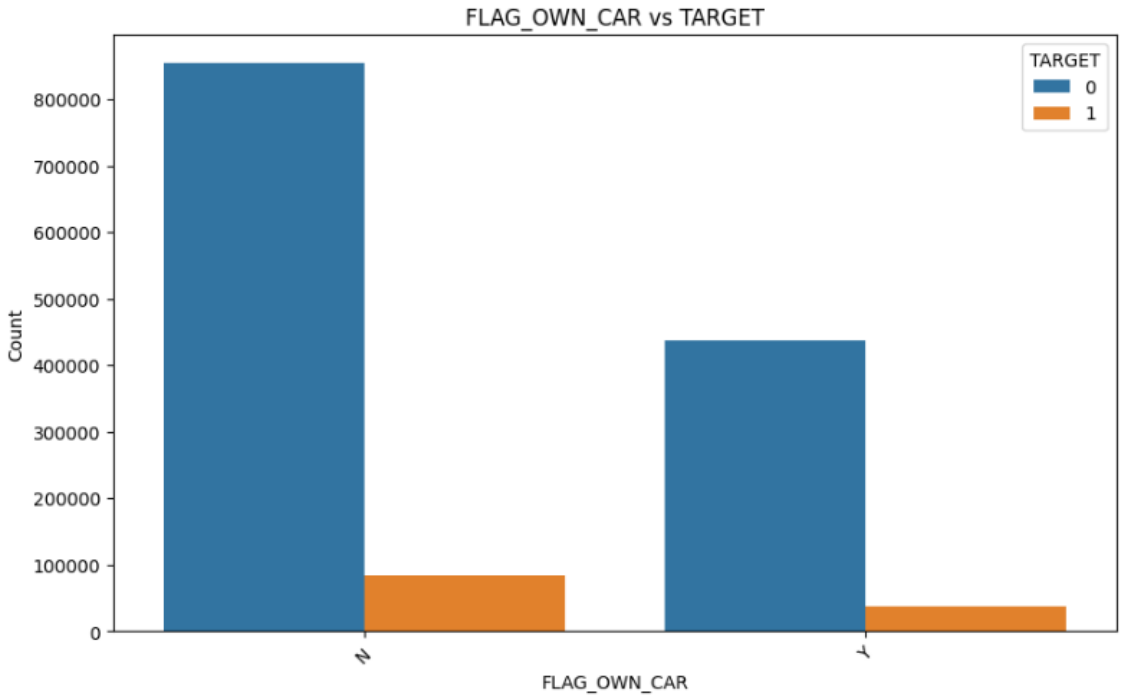
```
# Finding the categorical variables
categorical_vars = data.select_dtypes(include=['object']).columns.tolist()

# Print the columns with categorical variables
print("Categorical Variables:")
print(categorical_vars)

# plotting the categorical variable column through the looping using the for loop
for var in categorical_vars:
    # Create a countplot
    plt.figure(figsize=(10, 6))
    sns.countplot(x=var, hue='TARGET', data=data)
    plt.title(f'{var} vs TARGET')
    plt.xlabel(var)
    plt.ylabel('Count')
    plt.legend(title='TARGET', loc='upper right')
    plt.xticks(rotation=45)
    plt.show()
```

- The above code is used for the visual representation of the categorical data in the dataset where we have a **categorical_vars** variable to store the categorical data for finding the categorical columns we have used the panda's function where we select the columns those values are type is **object** and that panda's function is **data.select_dtypes().**

- For plotting all those columns we have used the **for loop** to iterate through each of those columns and count the categories present in them and by using the matplotlib library function we have plotted the data.





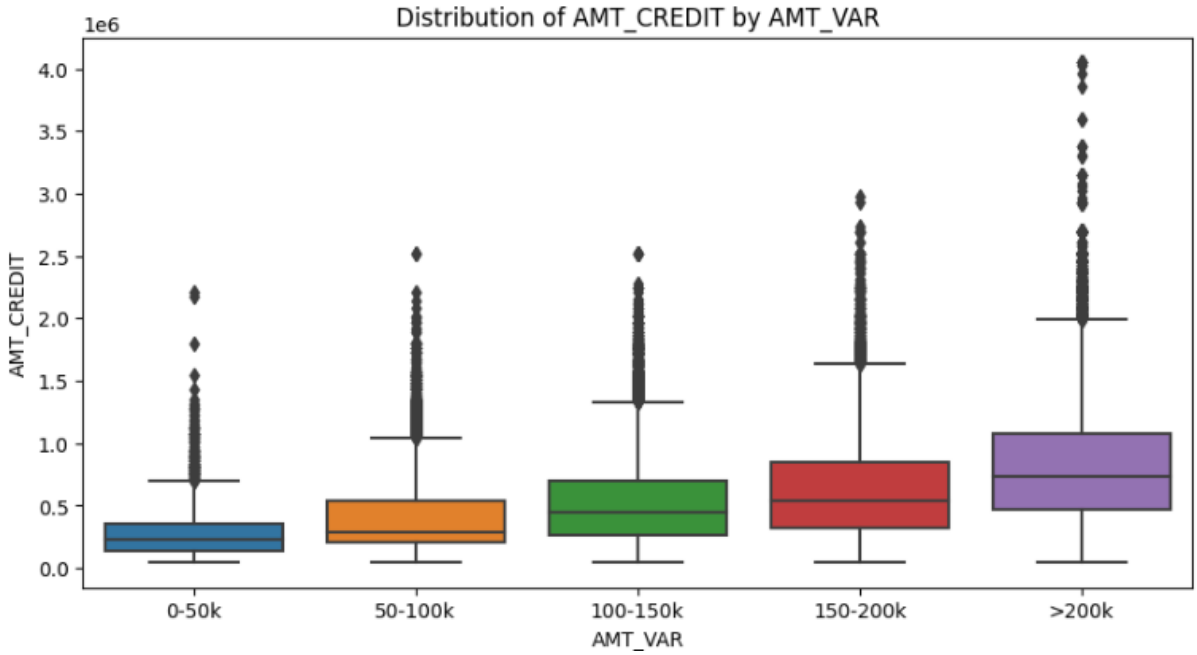


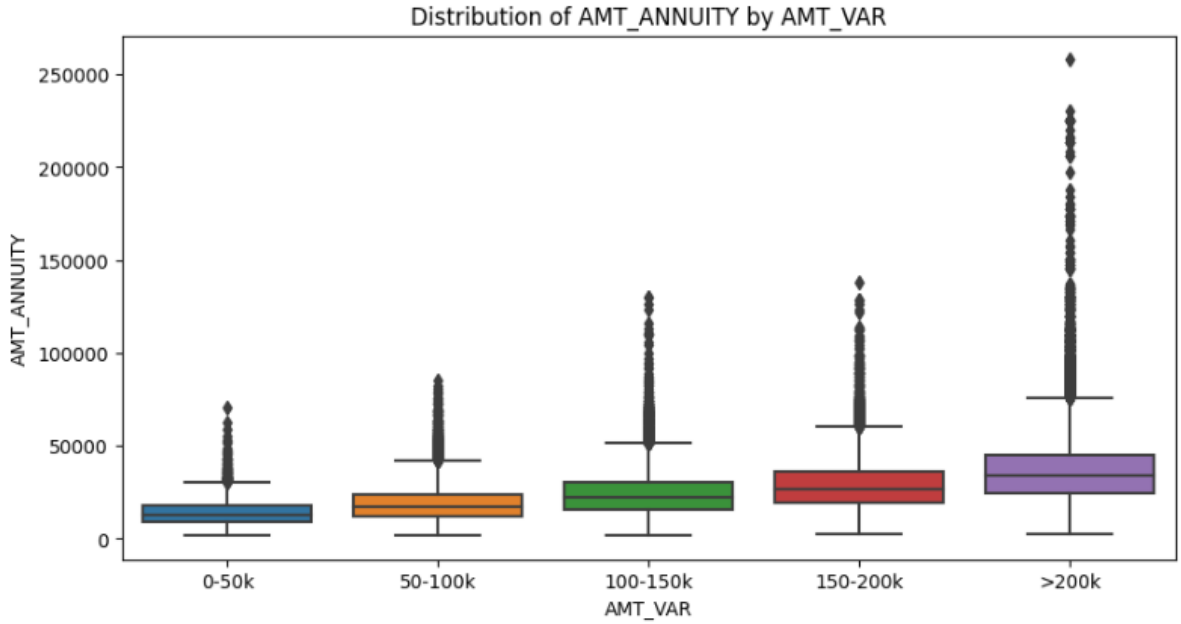
2. Numerical Data

- Numerical data is also called quantitative data where the data is in numerical form and can undergo mathematical operations such as addition, subtraction, multiplication, and division.
- Numerical data can be visually represented easily because of those numerical data consists of numerical values in them.

```
#  selected numerical variables
numerical_vars = ['AMT_INCOME_TOTAL', 'AMT_CREDIT', 'AMT_ANNUITY', 'AMT_GOODS_PRICE']

df['AMT_VAR'] = pd.cut(df['AMT_INCOME_TOTAL'], bins=[0, 50000, 100000, 150000, 200000, np.inf], labels=['0-50k', '50-100k', '100-150k', '150-200k', '>200k'])

#plotting for numerical variables using the for loop
for var in numerical_vars:
    plt.figure(figsize=(10, 5))
    sns.boxplot(x='AMT_VAR', y=var, data=df)
    plt.title('Distribution of ' + var + ' by AMT_VAR')
    plt.xlabel('AMT_VAR')
    plt.ylabel(var)
    plt.show()
```

- The above code shows the visual representation of some selected numerical columns of the dataset where we have **numerical_vars** variable to store the columns which consists of numerical value.
- We can perform analysis on no.of numerical columns present in the data but we have only selected only a few columns because there is no point in performing the analysis on all columns.
- For plotting the numerical data on specified columns we arranged the x-axis for all column's data to be plotted on the y-axis.
- Where we have created a data frame with a column called **AMT_VAR** and using the **for loop** we have iterated through the columns of numerical and plotted each plot separately for each column in the form of the **boxplot**.


Distribution of AMT_CREDIT by AMT_VAR
1e6
4.0
3.5
3.0
2.5
2.0
1.5
1.0
0.5
0.0
AMT_CREDIT
0-50k
50-100k
100-150k
150-200k
>200k
AMT_VAR


Distribution of AMT_ANNUITY by AMT_VAR
250000
200000
150000
100000
50000
0
AMT_ANNUITY
0-50k
50-100k
100-150k
150-200k
>200k
AMT_VAR

Conclusion

- The conclusions that we can derive from this analysis are :
- There is a data imbalance in this data with regard to the clients with payment difficulties and clients with other cases.
- The majority of clients own the reality property but very few of them own the car.
- Most of their education is secondary, most of them are married, most of them are working as laborers, and sales staff, and most of their loans are consumer loans and cash loans.
- Clients with lower education levels or who are unemployed are more likely to have payment difficulties.
- Clients who are divorced or single are more likely to have payment difficulties.
- Clients who are younger or have lower incomes are more likely to have payment difficulties.

THANK YOU