ALFRED HARUN AMUKO PORTFOLLIO PROJECTS

This document comprises sample projects I have handled before for some clients I have worked with as a Data Analyst and a Machine Learning Engineer. I use my skills and expertise in Machine Learning, Data Analysis, Statistical modelling , Data Cleaning and validation , Data Visualization and reporting to ensure I provide high precision data solutions and help stakeholders make impactful data driven decisions, leading to business growth and development in all aspects. My stack majors on Python, SQL, Pyspark, Microsoft Power BI, Tableau, Looker, Qlikview ,Airflow.

In this cocument I have included a few projects I have worked on.

PROJECT 1

INDUSTRY: Fintech- Device Financing Company

Objective

The mission was to help a device financing company boost credit scoring accuracy, make smarter financial and operational decisions, and create a better customer experience—all powered by data analytics, machine learning, and AI**.** I got the privilege to handle the whole complete project from computing a credit score Machine learning to Business Intelligence **Analytics.**

Project Components & Key Achievements

1. Credit Scoring Model

We designed a model to assess customer creditworthiness to reduce loan defaults:

- How it Works: Using KYC and historical data, I identified key variables—like payment history and income stability—and built models using logistic regression and random forests.
- Impact: This model helped identify creditworthy customers, lowering non-performing loans (NPLs) and strengthening the loan portfolio**.**

2. Business Intelligence & Analytics Solutions

A suite of analytics insights to improve business performance:

- Credit Score Optimization: Fine-tuned scoring thresholds for better balance between credit risk and eligibility.
- Sales & Conversions: Analyzed customer journeys, pinpointing where conversions dropped, leading to higher approval rates.
- Collections & Default Analysis: Used predictive insights to reduce collection costs and improve recovery rates.
- Financial Forecasting: Created forecasting models for cash flow, sales, and profits, improving financial planning.

Outcome: Better decision-making in sales, collections, and financial forecasting helped boost revenue and reduce credit losses.

3. AI-Powered Chatbot with LLM

To streamline customer support, we created a chatbot using large language models:

- Capabilities: It could handle customer queries quickly, only escalating complex ones to agents.
- Outcome: This reduced response times, increased customer satisfaction, and improved agent productivity by 20%.

4. Speech-to-Text Automated Customer Recognition (ACR) Model

An innovative model to serve speech-impaired customers:

- Solution: A robust ACR model enabling seamless communication for speech-impaired customers.
- Outcome: This boosted inclusivity and contributed to a 15% increase in new sign-ups, enhancing the company's accessibility and brand presence.

5. Market Research for Credit Scoring

To adapt credit scoring to East African markets, we conducted thorough research:

- Approach: Used surveys, interviews, and regional data analysis to align our model with local financial behaviors.
- Outcome: Established region-specific scoring thresholds, balancing accessibility with financial security.

Visual Insights and Recommendations

A. Sales Analysis

- Goal: Optimize inventory and marketing based on sales trends.
- Visuals: Sales ,trend charts,heatmaps and funnel charts revealed high-demand periods and top product categories.
- Action: We recommended adjusting inventory and scaling back promotion for high-risk devices, leading to fewer defaults.

B. Customer Engagement Analysis

- Goal: Tailor marketing based on customer behavior.
- Visuals: FRM segmentation and interest heatmaps highlighted engagement hotspots and device preferences.
- Action: Launched targeted, geotargeted campaigns in high-engagement areas, boosting customer retention and conversion.

C. Non-Performing Loan (NPL) Analysis

- Goal: Reduce NPL rates by spotting default patterns.
- Visuals: Trend charts by device type and region helped pinpoint high-risk models.
- Action: Adjusted inventory and introduced risk-based loan approvals, reducing NPLs by an estimated 15%.

D. Credit Score & Limit Analysis

- Goal: Fine-tune credit limits based on risk.
- Visuals: Score distribution and recommended limits charts helped in setting effective credit limits.
- Action: Tailored credit limits based on customer risk profiles, further reducing defaults.

Marketing Campaign Optimization

- Goal: Boost campaign effectiveness through targeted insights.
- Visuals: Performance dashboards and segmentation maps guided campaign planning.
- Action: Focused campaigns on peak engagement periods, increasing ROI by 30%.

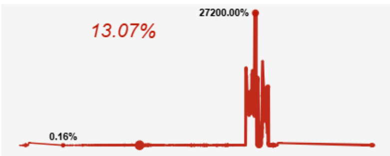
Key Results & Business Impact

- Enhanced Credit Risk Management: Loan default rates dropped by 30%, strengthening the loan portfolio.
- Sales Conversions: Approvals for device financing rose by 25%.
- Financial Planning: Forecasting models helped achieve precise financial planning and cash flow management.
- Customer Satisfaction & Inclusivity: The chatbot and ACR model improved satisfaction by 20% and broadened the customer base to include previously underserved groups.

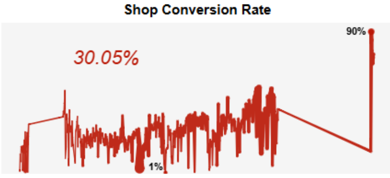
Technical Codes and Sample Charts and Dashboards (Used Python Programming for Creating Model and SQL for data analysis and data wrangling , connecting to databases, Tableau and Microsoft power BI for Data Visualization)

1,818,035	487,316	52.21%
Total Registered	Total Visited Shop	Qualification Rate

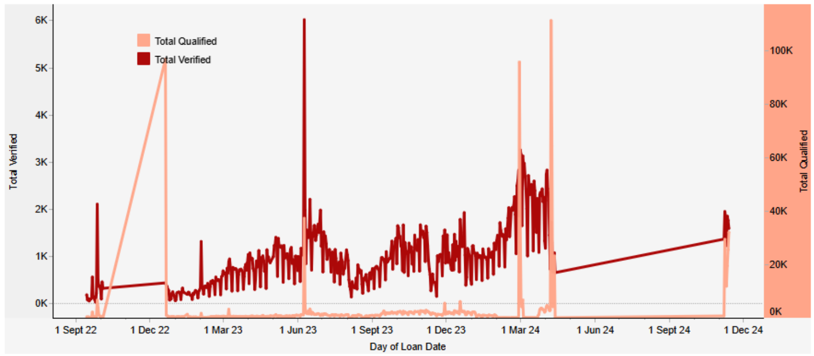
Global Conversion Rate



Shop Conversion Rate



Scored & Verified



Filters

Measure Names

- Loan count %
- Qualification Rate
- Total Verified %

- Loan Count
- Total Qualified
- Total Scored
- Total Verified

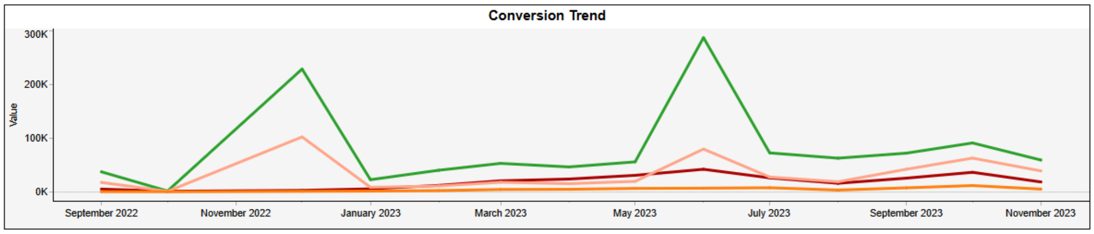
start date

01/09/2022

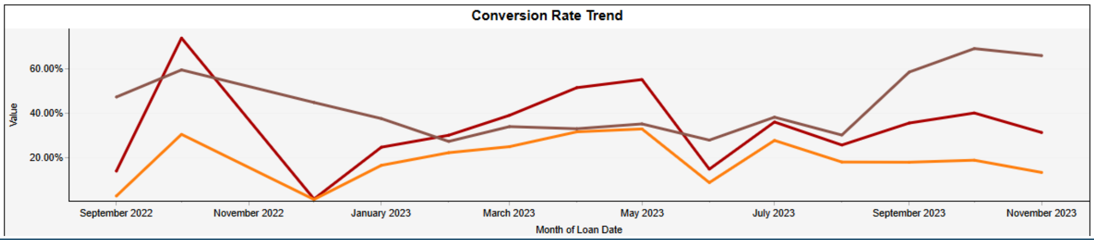
end date

21/11/2023

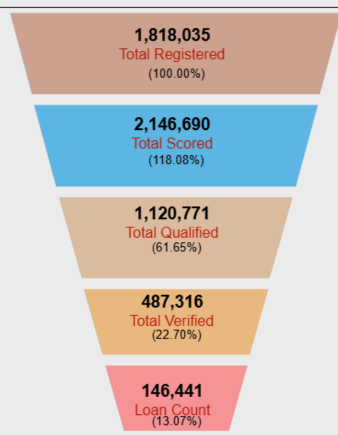
Conversion Trend

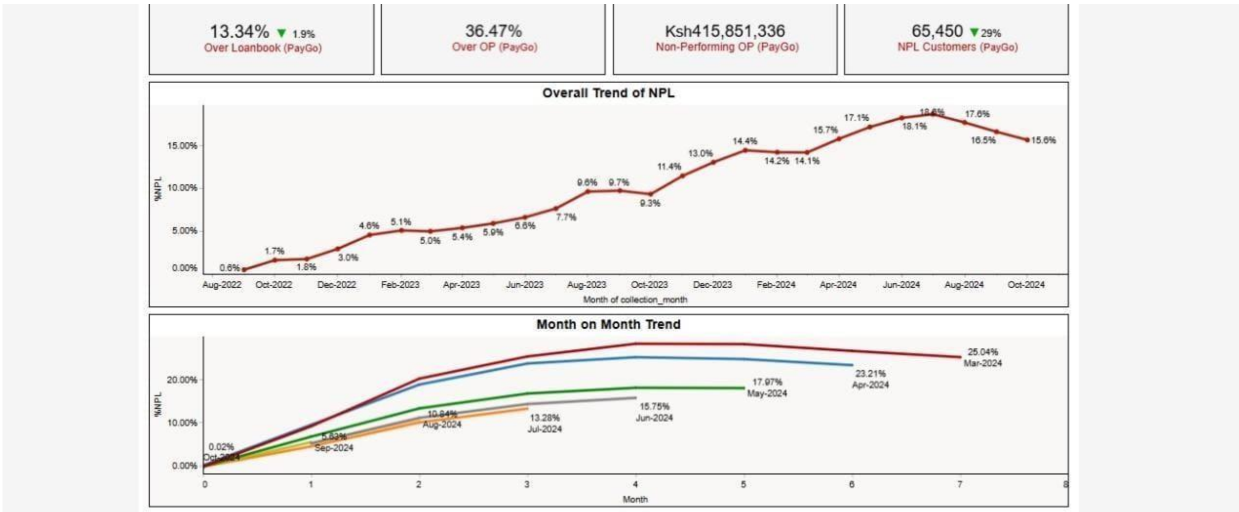


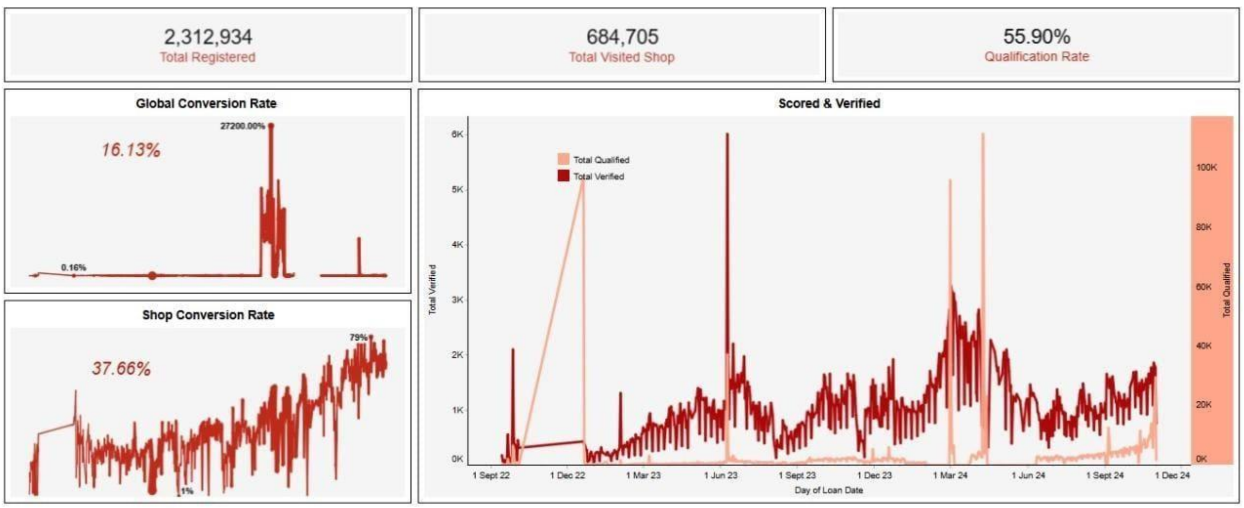
Conversion Rate Trend

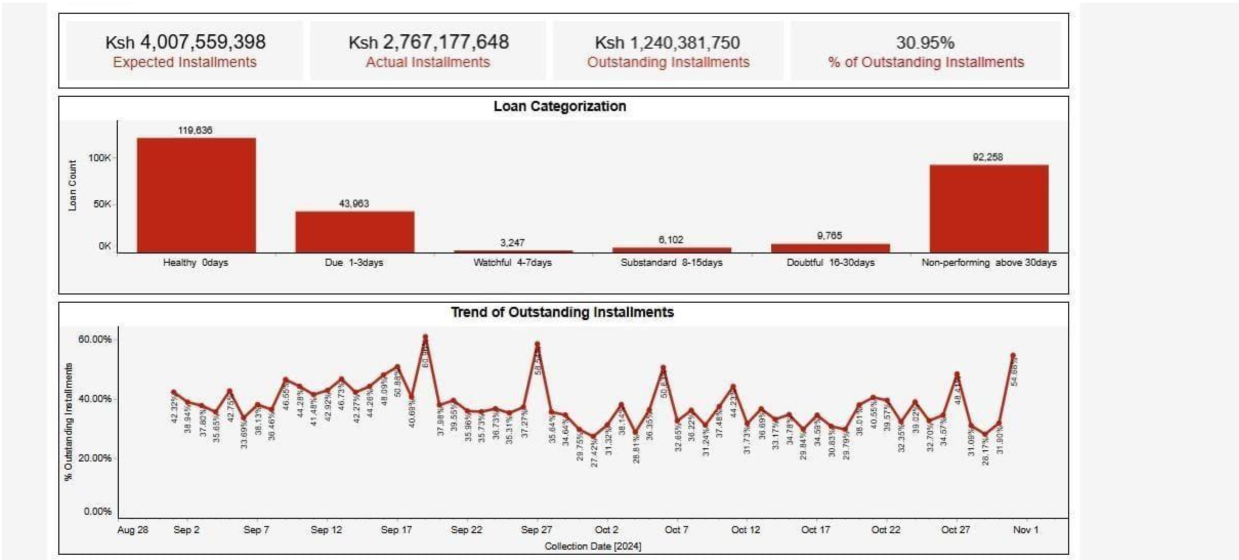


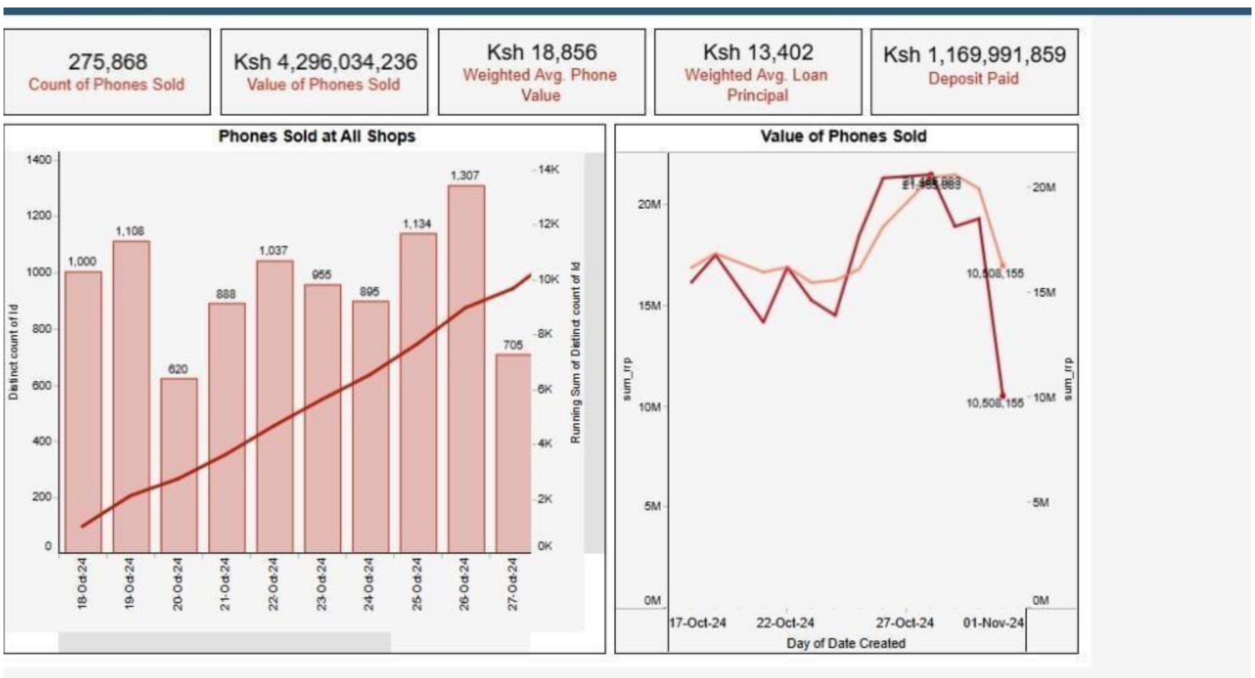
Customer Conversion Funnel

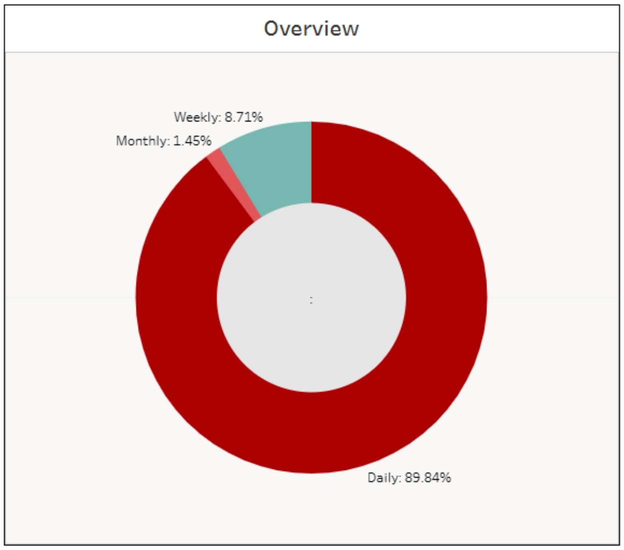


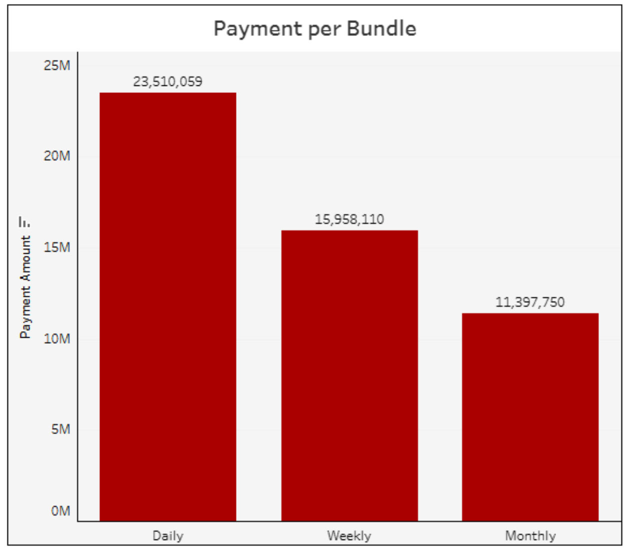

13.34% ▼ 1.9%
Over Loanbook (PayGo)
36.47%
Over OP (PayGo)
Ksh415,851,336
Non-Performing OP (PayGo)
65,450 ▼29%
NPL Customers (PayGo)
Overall Trend of NPL
%NPL
Month of collection_month
Month on Month Trend
%NPL
Month
25.04%
Mar-2024
23.21%
Apr-2024
17.97%
May-2024
15.75%
Jun-2024
13.28%
Jul-2024
10.84%
Aug-2024
5.82%
Sep-2024
0.02%
Oct-2024


2,312,934
Total Registered
684,705
Total Visited Shop
55.90%
Qualification Rate
Global Conversion Rate
16.13%
27200.00%
0.16%
Shop Conversion Rate
37.66%
79%
Scored & Verified
Total Qualified
Total Verified
Total Verified
Total Qualified
Day of Loan Date


Ksh 4,007,559,398
Expected Installments
Ksh 2,767,177,648
Actual Installments
Ksh 1,240,381,750
Outstanding Installments
30.95%
% of Outstanding Installments
Loan Categorization
Loan Count
119,636
43,963
3,247
6,102
9,765
92,258
Healthy 0days
Due 1-3days
Watchful 4-7days
Substandard 8-15days
Doubtful 16-30days
Non-performing above 30days
Trend of Outstanding Installments
% Outstanding Installments
Collection Date [2024]


275,868
Count of Phones Sold
Ksh 4,296,034,236
Value of Phones Sold
Ksh 18,856
Weighted Avg. Phone Value
Ksh 13,402
Weighted Avg. Loan Principal
Ksh 1,169,991,859
Deposit Paid
Phones Sold at All Shops
Distinct count of Id
Running Sum of Distinct count of Id
1,000
1,108
620
888
1,037
955
895
1,134
1,307
705
18-Oct-24
19-Oct-24
20-Oct-24
21-Oct-24
22-Oct-24
23-Oct-24
24-Oct-24
25-Oct-24
26-Oct-24
27-Oct-24
Value of Phones Sold
sum_rrp
10,508,155
Day of Date Created





Sample Modelling Code to Create Credit Score model:

```
# -*- coding: utf-8 -*-
"""Credit_score (1).ipynb

Automatically generated by Colab.

Original file is located at

https://colab.research.google.com/drive/1k440S2SuDd1mq_JKRy5V1SilBKq9L4B
"""

# from google.colab import files

# # Prompt to upload a file
# uploaded = files.upload()

# Commented out IPython magic to ensure Python compatibility.
# Load in our libraries import
pandas as pd import numpy as np
import re import sklearn import
seaborn as sns import
matplotlib.pyplot as plt
# %matplotlib inline

import warnings
warnings.filterwarnings('ignore')

from collections import Counter
```

```
from sklearn.ensemble import RandomForestClassifier, VotingClassifier
from sklearn.neighbors import KNeighborsClassifier from sklearn.svm
import SVC
from sklearn.model_selection import GridSearchCV, cross_val_score,
StratifiedKFold, learning_curve
from sklearn.feature_selection import SelectFromModel, SelectKBest
from sklearn.model_selection import StratifiedKFold from
sklearn.model_selection import GridSearchCV from sklearn.ensemble
import GradientBoostingClassifier from datetime import datetime

sns.set(style='dark', context='notebook', palette='deep')
pd.options.display.max_columns = 100

import pandas as pd

# Step 1: Read the Excel file into a DataFrame
training_csv = 'Copy_of_Days_in_Default.xlsx' df
= pd.read_excel(training_csv)
#Step 2: Read the Csv file into a Dataframe
training_csv2 = 'Onfon.csv' df1 =
pd.read_csv(training_csv2)

# df

# Convert the 'Msisdn' column to strings df['Msisdn']
= df['Msisdn'].astype(str)

# Replace '254' with '0' and store it as a string df['Msisdn']
= df['Msisdn'].str.replace('^254', '0')
 df.shape
df.info()

# Convert the 'Msisdn' column in df to a string data type df['Msisdn']
= df['Msisdn'].astype(str)
 df.info()
df1['MSISDN'] = df1['MSISDN'].astype(str).apply(lambda x: x.zfill(10))



# merged_df

import pandas as pd

# Check for common values
common_values = set(df['Msisdn']).intersection(df1['MSISDN'])
 if len(common_values) ==
0:
    print("No common values found in 'Msisdn' and 'MSISDN' columns.")
else:
```

```
    # Merge df and df1 on the 'Msisdn' and 'MSISDN' columns, respectively
merged_df    =    df.merge(df1,    left_on='Msisdn',    right_on='MSISDN',
how='inner')

    # Display the merged DataFrame
merged_df.head()

# # Print the unique values in each column
# for column in merged_df.columns:
#     unique_values = merged_df[column].unique()
#     unique_count = len(unique_values)

#     print(f"Column: {column}")
#     print(f"Number of Unique Values: {unique_count}")
#     print(f"Unique Values:")
#     print(unique_values)
#     print("\n")

# merged_df

merged_df.rename(columns={'Days In Default': 'days_defaulted'},
inplace=True)
# merged_df

# Calculate the minimum value
min_value = merged_df['days_defaulted'].min()

# Calculate the maximum value
max_value = merged_df['days_defaulted'].max()

# Calculate the median
median_value = merged_df['days_defaulted'].median()

# Calculate the mode (returns a Series, so we take the first element)
mode_value = merged_df['days_defaulted'].mode().iloc[0]

# Calculate the mean
mean_value = merged_df['days_defaulted'].mean()

# Display the results
print(f"Minimum Value: {min_value}")
print(f"Maximum Value: {max_value}")
print(f"Median Value: {median_value}")
print(f"Mode Value: {mode_value}") print(f"Mean
Value: {mean_value}")

# Define the ranges
# Define the ranges with intervals of 3
ranges = [(-float('inf'), 2), (3, 5), (6, 8), (9, 11), (12, 14), (15, 17),
(18, 20), (21, 23), (24, 26), (27, 29), (30, float('inf'))]
```

```
# Use pd.cut() to categorize 'days_defaulted' into the specified ranges
merged_df['range'] = pd.cut(merged_df['days_defaulted'], bins=[start-1 for
start, _ in ranges] + [float('inf')], labels=[f'{start}-{end}' for start,
end in ranges])

# Calculate the count of 'Msisdn' for each range
range_counts = merged_df['range'].value_counts().sort_index()

# Display the count of 'Msisdn' for each range print(range_counts)

# Define the ranges with intervals of 3
ranges = [(-float('inf'), 2), (3, 5), (6, 8), (9, 11), (12, 14), (15, 17),
(18, 20), (21, 23), (24, 26), (27, 29), (30, float('inf'))]
# Use pd.cut() to categorize 'days_defaulted' into the specified ranges
merged_df['range'] = pd.cut(merged_df['days_defaulted'], bins=[start-1 for
start, _ in ranges] + [float('inf')], labels=[f'{start}-{end}' for start,
end in ranges])

# Calculate the count of 'Msisdn' for each range
range_counts = merged_df['range'].value_counts().sort_index()

# Plot the histogram plt.figure(figsize=(10,
6))
plt.bar(range_counts.index, range_counts.values, width=0.8,
align='center', alpha=0.7) plt.xlabel('Ranges')
plt.ylabel('Frequency')
plt.title('Histogram of Ranges vs. Frequency')
plt.xticks(rotation=45, ha='right')
plt.tight_layout() plt.show()

6756/9797*100

# Create a new column based on the condition
merged_df['Target'] = np.where(merged_df['days_defaulted'] <= 15, 0, 1)
# merged_df

# Assuming 'Msisdn' is the column you want to set as the index in the
DataFrame 'merged_df'
merged_df.set_index('Msisdn', inplace=True)

# merged_df

merged_df.shape

merged_df.info()

merged_df.isnull().sum()

# Convert the 'DateOfBirth' column to datetime
merged_df['DateOfBirth'] = pd.to_datetime(merged_df['DateOfBirth'])
```

```
# Calculate age by subtracting the 'DateOfBirth' from the current date
current_date = datetime.now() merged_df['Age'] = (current_date -
merged_df['DateOfBirth']).astype('<m8[Y]')

# merged_df

"""Let me drop some columns
Msisdn- Not rellevant because it is a unique identifier
days_defaulted- Used to create the  dependent target varriable, so that is
why I see no significance
MSISDN-unique identifier of a customer too
DateOfBirth-used it to generate age column so it is no longer relevant
DATA_EXTRACTION_DATE-Just the date data was collected ,not significant
TOTAL_RECIEPTS_VALUE-This is totally blank so has no value range-this
isjust a range column I created to check range in the days defaulted
to choose the threshhold
"""  merged_df = merged_df.drop([ "days_defaulted", "MSISDN",
"DateOfBirth", "DATA_EXTRACTION_DATE", "TOTAL_RECIEPTS_VALUE","range"],
axis=1)




merged_df.info()

merged_df.isnull().sum()


"""**LET ME HANDLE MISSING VALUES IN age column NOW**

there are 3 missing values in age column,let me check which percentage of
the data is it
"""

3/9759*100

"""WE can see it is a verry smaller portion of the data , therefore I will
drop them"""

merged_df.dropna(subset=['Age'], inplace=True)

merged_df.isnull().sum()

"""I will check how to handle missing values in Gender column later in the
code after tackling some bivariate statistics"""



"""## BIVARIATE & MULTIVARIATE ANALYSIS"""
```

```
## Let me change Age to a Categorical Varriable now




"""Now since we only have categorical varriables , we will not base the
predictive ability of variables on correlation ,I will explore other means
to to gauge the predictive abilities of your columns. Correlation is
generally not the best measure for categorical-categorical relationships
because it's designed for continuous variables and doesn't provide a
complete picture of the associations in categorical data

## Let's Explore Chi-Square Test

## Let us check Relation betweeen columns and the Target
"""  from scipy.stats import
chi2_contingency

# Create a list of columns (excluding the 'Target' column) for which you
want to calculate p-values
columns_to_test = ['Gender', 'PAYMENT_TYPE', 'RECHARGE_MODE',
'SUBSCRIPTION_PERIOD', 'TRANSACTION_PERIOD', 'COUNT_OF_BLOCKS',
'TRANSACTION_RECIEVERS', 'LIMIT', 'TOPUP_CONSUMPTION',
'TOPUP_CONSUMPTION_6', 'ACTIVE_STATUS', 'TRANSACTIONS_RECIEVED',
'ACCOUNT_DEBIT', 'ACCOUNT_CREDIT', 'PAYMENT_TRXN',
'NUMBER_OF_PAYMENT_TRXN', 'VALUE_SENT', 'HIGHEST_VALUE', 'VALUE_RECIEVED',
'TRXN_COUNT', 'PAYMENT_VALUE_2', 'PAYMENT_COUNT_2', 'BANK_TRANS',
'OTHER_RECEIPTS', 'TOTAL_BANK_TRANS', 'TOTAL_PAYMENTS',
'TOTAL_MONEY_IN_3', 'TOTAL_MONEY_OUT_3', 'CURR_POINTS', 'RDM_POINTS_6',
'LOAN_COUNT', 'LOAN_BLACKLIST', 'COUNT_OF_OVERDUE_LOANS',
'LOAN_BLACKLIST_DAYS', 'CURR_DEVICE_MAKE', 'CURR_DEVICE_RRP']

# Iterate over the columns and calculate p-values for
column in columns_to_test:
    contingency_table = pd.crosstab(merged_df[column],
merged_df['Target'])
    chi2, p, _, _ = chi2_contingency(contingency_table)     print(f"Chi-
squared Statistic for {column}: {chi2}")     print(f"P-Value for
{column}: {p}")

    # Interpret the results
if p < 0.05:
        print(f"There is a statistically significant association between
{column} and Target.")
else:
        print(f"There is no statistically significant association between
{column} and Target.")
print("\n")

"""## MULTIVARIATE ANALYSIS
```

```
## will calculate p-values for associations between all pairs of columns
and store them in a DataFrame. You can ## we then inspect the p-values to
identify columns that seem to be related based on statistical
significance.
"""

# from itertools import combinations
# from scipy.stats import chi2_contingency

# # Create a list of independent variable columns (exclude 'Target' if
necessary)
# independent_columns = ['Gender', 'PAYMENT_TYPE', 'RECHARGE_MODE',
'SUBSCRIPTION_PERIOD', 'TRANSACTION_PERIOD', 'COUNT_OF_BLOCKS',
'TRANSACTION_RECIEVERS', 'LIMIT', 'TOPUP_CONSUMPTION',
'TOPUP_CONSUMPTION_6', 'ACTIVE_STATUS', 'TRANSACTIONS_RECIEVED',
'ACCOUNT_DEBIT', 'ACCOUNT_CREDIT', 'PAYMENT_TRXN',
'NUMBER_OF_PAYMENT_TRXN', 'VALUE_SENT', 'HIGHEST_VALUE', 'VALUE_RECIEVED',
'TRXN_COUNT', 'PAYMENT_VALUE_2', 'PAYMENT_COUNT_2', 'BANK_TRANS',
'OTHER_RECEIPTS', 'TOTAL_BANK_TRANS', 'TOTAL_PAYMENTS',
'TOTAL_MONEY_IN_3', 'TOTAL_MONEY_OUT_3', 'CURR_POINTS', 'RDM_POINTS_6',
'LOAN_COUNT', 'LOAN_BLACKLIST', 'COUNT_OF_OVERDUE_LOANS',
'LOAN_BLACKLIST_DAYS', 'CURR_DEVICE_MAKE', 'CURR_DEVICE_RRP']

# # Create a dictionary to store chi-square results
# chi_square_results = {}

# # Iterate over all combinations of independent variables
# for combo in combinations(independent_columns, 2):
#     var1, var2 = combo

#     contingency_table = pd.crosstab(merged_df[var1], merged_df[var2])
#     chi2, p, _, _ = chi2_contingency(contingency_table)
#     chi_square_results[f"{var1} vs. {var2}"] = {
#         "Chi-squared Statistic": chi2,
#         "P-Value": p
#     }

#     print(f"Chi-squared Statistic for {var1} vs. {var2}: {chi2}")
#     print(f"P-Value for {var1} vs. {var2}: {p}")

#     # Interpret the results
#     if p < 0.05:
#         print(f"There is a statistically significant association between
{var1} and {var2}.")
#     else:
#         print(f"There is no statistically significant association
between {var1} and {var2}.")
#     print("\n")
```

```
# # You can access the results in chi_square_results dictionary for
further analysis

"""## We can now explore CramÃ©r's V

CramÃ©r's V: CramÃ©r's V is a measure of association between two
categorical variables. It ranges from 0 to 1, with higher values
indicating a stronger association. It's an extension of the chi-square
test statistic.
"""
from scipy.stats import chi2_contingency

# Create a list of columns (excluding the 'Target' column) for which you
want to calculate CramÃ©r's V
columns_to_test = ['Gender', 'PAYMENT_TYPE', 'RECHARGE_MODE',
'SUBSCRIPTION_PERIOD', 'TRANSACTION_PERIOD', 'COUNT_OF_BLOCKS',
'TRANSACTION_RECIEVERS', 'LIMIT', 'TOPUP_CONSUMPTION',
'TOPUP_CONSUMPTION_6', 'ACTIVE_STATUS', 'TRANSACTIONS_RECIEVED',
'ACCOUNT_DEBIT', 'ACCOUNT_CREDIT', 'PAYMENT_TRXN',
'NUMBER_OF_PAYMENT_TRXN', 'VALUE_SENT', 'HIGHEST_VALUE', 'VALUE_RECIEVED',
'TRXN_COUNT', 'PAYMENT_VALUE_2', 'PAYMENT_COUNT_2', 'BANK_TRANS',
'OTHER_RECEIPTS', 'TOTAL_BANK_TRANS', 'TOTAL_PAYMENTS',
'TOTAL_MONEY_IN_3', 'TOTAL_MONEY_OUT_3', 'CURR_POINTS', 'RDM_POINTS_6',
'LOAN_COUNT', 'LOAN_BLACKLIST', 'COUNT_OF_OVERDUE_LOANS',
'LOAN_BLACKLIST_DAYS', 'CURR_DEVICE_MAKE', 'CURR_DEVICE_RRP']

# Function to calculate CramÃ©r's V def
cramers_v(confusion_matrix):
    chi2 = chi2_contingency(confusion_matrix)[0]
n = confusion_matrix.sum().sum()     phi2 = chi2
/ n
    r, k = confusion_matrix.shape
    phi2corr = max(0, phi2 - ((k - 1) * (r - 1)) / (n - 1))
rcorr = r - ((r - 1)**2) / (n - 1)     kcorr = k - ((k -
1)**2) / (n - 1)
    return np.sqrt(phi2corr / min((kcorr - 1), (rcorr - 1)))

# Iterate over the columns and calculate CramÃ©r's V for
column in columns_to_test:
    confusion_matrix = pd.crosstab(merged_df[column], merged_df['Target'])
cramers_v_value = cramers_v(confusion_matrix.values)     print(f"CramÃ©r's
V for {column}: {cramers_v_value}")

    # Interpret the results
if cramers_v_value >= 0.1:
        print(f"There is a moderate to strong association between {column}
and Target.")     else:         print(f"There is no or a weak association
between {column} and Target.")     print("\n")

"""**Let me explore this on Crammers-V**"""
```

```
from itertools import combinations
import numpy as np import pandas
as pd
from scipy.stats import chi2_contingency

# Create a list of independent variable columns (exclude 'Target' if
necessary)
independent_columns = ['Gender', 'PAYMENT_TYPE', 'RECHARGE_MODE',
'SUBSCRIPTION_PERIOD', 'TRANSACTION_PERIOD', 'COUNT_OF_BLOCKS',
'TRANSACTION_RECIEVERS', 'LIMIT', 'TOPUP_CONSUMPTION',
'TOPUP_CONSUMPTION_6', 'ACTIVE_STATUS', 'TRANSACTIONS_RECIEVED',
'ACCOUNT_DEBIT', 'ACCOUNT_CREDIT', 'PAYMENT_TRXN',
'NUMBER_OF_PAYMENT_TRXN', 'VALUE_SENT', 'HIGHEST_VALUE', 'VALUE_RECIEVED',
'TRXN_COUNT', 'PAYMENT_VALUE_2', 'PAYMENT_COUNT_2', 'BANK_TRANS',
'OTHER_RECEIPTS', 'TOTAL_BANK_TRANS', 'TOTAL_PAYMENTS',
'TOTAL_MONEY_IN_3', 'TOTAL_MONEY_OUT_3', 'CURR_POINTS', 'RDM_POINTS_6',
'LOAN_COUNT', 'LOAN_BLACKLIST', 'COUNT_OF_OVERDUE_LOANS',
'LOAN_BLACKLIST_DAYS', 'CURR_DEVICE_MAKE', 'CURR_DEVICE_RRP']

# Create a dictionary to store Cramer's V results cramers_v_results
= {}

# Iterate over all combinations of independent variables for
combo in combinations(independent_columns, 2):
    var1, var2 = combo

    contingency_table = pd.crosstab(merged_df[var1], merged_df[var2])
chi2, _, _, _ = chi2_contingency(contingency_table)

    n = contingency_table.sum().sum()
phi2 = chi2 / n
    r, k = contingency_table.shape
    phi2_corr = max(0, phi2 - ((k-1)*(r-1)) / (n-1))
r_corr = r - ((r-1)**2) / (n-1)     k_corr = k -
((k-1)**2) / (n-1)
    cramers_v = np.sqrt(phi2_corr / (min((k_corr-1), (r_corr-1))))

    cramers_v_results[f"{var1} vs. {var2}"] = cramers_v

    print(f"Cramer's V for {var1} vs. {var2}: {cramers_v}")

    # Interpret the results
if cramers_v > 0.1:
        print(f"There is a moderate association between {var1} and
{var2}.")     else:         print(f"There is no substantial
association between {var1} and
{var2}.")
print("\n")

# You can access the results in cramers_v_results dictionary for further
analysis
```

```
"""**I will drop the COUNT_OF_OVERDUE_LOANS, and retain LOAN_BLACKLIST,
because LOAN_BLACKLIST has lower P value and higher Crammers V in relation
to the Target**

**I will drop the SUBSCRIPTION_PERIOD, and retain LOAN_BLACKLIST, because
LOAN_BLACKLIST has lower P value and higher Crammers V in relation to the
Target**

## BIVARIATE ANALYSIS

## LET ME EXPLORE GENDER
"""




# Explore Gender vs Target
g = sns.catplot(x="Gender", y="Target", data=merged_df, kind="bar",
height=6, palette="muted")
g.despine(left=True)
g.set_ylabels("Target probability")

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

# # Replace missing values (NaN) in 'Gender' column with 'Missing' for
visualization
# merged_df['Gender'].fillna('Missing', inplace=True)

# # Create a bar plot to show the distribution of 'Gender' by 'Target'
values
# plt.figure(figsize=(12, 6))
# sns.countplot(data=merged_df, x='Gender', hue='Target', palette={0:
'blue', 1: 'green', 'Missing': 'red'})
# plt.title('Distribution of Gender by Target')
# plt.xlabel('Gender')
# plt.ylabel('Count')

# plt.tight_layout()
# plt.show()

"""**To check Percentage**"""

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

# # Replace missing values (NaN) in 'Gender' column with 'Missing' for
visualization
# merged_df['Gender'].fillna('Missing', inplace=True)
```

```
# # Calculate the percentage of 'Gender' by 'Target' values #
percentage_df = (merged_df.groupby(['Gender', 'Target']).size() /
len(merged_df)).reset_index(name='Percentage')

# # Create a bar plot to show the distribution of 'Gender' by 'Target'
values
# plt.figure(figsize=(12, 6))
# sns.barplot(data=percentage_df, x='Gender', y='Percentage',
hue='Target', palette={0: 'blue', 1: 'green', 'Missing': 'red'})
# plt.title('Distribution of Gender by Target')
# plt.xlabel('Gender')
# plt.ylabel('Percentage of Count')

# # Print the percentages for each class and each 'Target' value
# for index, row in percentage_df.iterrows():
#     gender = row['Gender']
#     target = row['Target']
#     percentage = row['Percentage']
#     print(f"Gender: {gender}, Target: {target}, Percentage:
{percentage:.2%}")

# plt.tight_layout()
# plt.show()

"""**WE can see from the above that both Default and non default is
dominated by Male**"""

# import pandas as pd
# import numpy as np

# # Convert the 'Gender' column to string data type before imputation
# merged_df['Gender'] = merged_df['Gender'].astype(str)


# # Calculate the overall proportions of 'M' and 'F' in the existing data
# gender_proportions = merged_df['Gender'].value_counts(normalize=True)

# # Function to impute missing values based on overall proportions
# def impute_gender(row):
#     if row['Gender'] == 'Gender_Missing':
#         # Randomly impute "M" or "F" based on the overall proportions #
imputed_gender = np.random.choice(gender_proportions.index,
p=gender_proportions.values)
#         return imputed_gender
#     else:
#         return row['Gender']

# # Apply the function to impute missing values
# merged_df['Gender'] = merged_df.apply(impute_gender, axis=1)

# # Check if there are still missing values
```

```
# missing_values_count = (merged_df['Gender'] == 'Gender_Missing').sum()
# print(f"Missing values in 'Gender' column after imputation:
{missing_values_count}")

# import pandas as pd
# import numpy as np

# # Calculate proportions of 'M' and 'F' based on 'Target' values
# proportions_by_target = merged_df.groupby(['Target',
'Gender']).size().unstack(fill_value=0)
# proportions_by_target =
proportions_by_target.div(proportions_by_target.sum(axis=1), axis=0)

# # Function to impute missing values based on 'Target' and proportions
# def impute_gender(row):
#     if pd.isna(row['Gender']):
#         target_value = row['Target']
#         imputed_gender =
np.random.choice(proportions_by_target.loc[target_value].index,
p=proportions_by_target.loc[target_value].values) #
return imputed_gender
#     else:
#         return row['Gender']

# # Apply the function to impute missing values
# merged_df['Gender'] = merged_df.apply(impute_gender, axis=1)

# # Check if there are still missing values
# missing_values_count = merged_df['Gender'].isna().sum() #
print(f"Missing values in 'Gender' column after imputation:
{missing_values_count}")
 merged_df.dropna(subset=['Gender'],
inplace=True)



# Get the count of missing values (NaN) for each column missing_values
= merged_df.isnull().sum()

# Print the unique values and missing value counts for each column for
column in merged_df.columns:
    unique_values = merged_df[column].unique()
unique_count = len(unique_values)     missing_count =
missing_values[column]     print(f"Column '{column}':")
    print(f"Unique Values ({unique_count} unique values):")
print(unique_values)
    print(f"Missing Values Count: {missing_count}")
print("\n")

# import pandas as pd
# import scipy.stats as stats
```

```
# # Define a list of columns to test
# columns_to_test = merged_df.columns.difference(['Age'])  # Exclude 'Age'
# # Create an empty dictionary to store the results
# anova_results = {}

# # Perform ANOVA for each column
# for column in columns_to_test:
#     unique_values = merged_df[column].unique()
#     if len(unique_values) > 1:  # Check if there are multiple groups in
the column
#         try:
#             # Perform ANOVA #
f_statistic, p_value =
stats.f_oneway(*[merged_df['Age'][merged_df[column] == group_value] for
group_value in unique_values])

#             # Store the results
#             anova_results[column] = {'F-statistic': f_statistic,
'pvalue': p_value}
#         except KeyError:
#             print(f"Column '{column}' not found in the dataset.")

# # Function to interpret the p-value
# def interpret_p_value(p_value):
#     if p_value < 0.05:
#         return "Strong Relationship"
#     elif p_value < 0.1:
#         return "Moderate Relationship"
#     else:
#         return "Weak Relationship"

# # Print the results
# for column, results in anova_results.items():
#     p_value = results['p-value']
#     relationship_strength = interpret_p_value(p_value)
#     print(f"{column}: {relationship_strength} (p-value: {p_value:.4f})")



# import pandas as pd
# import scipy.stats as stats

# # Define a list of columns to test
# columns_to_test = merged_df.columns.difference(['Age'])  # Exclude 'Age'
# # Create an empty dictionary to store the results
# kw_results = {}

# # Perform Kruskal-Wallis test for each column
# for column in columns_to_test:
#     unique_values = merged_df[column].unique()
```

```
#     if len(unique_values) > 1:  # Check if there are multiple groups in
the column #         try:
#             # Perform Kruskal-Wallis test
#             groups = [merged_df['Age'][merged_df[column] == group_value]
for group_value in unique_values]
#             k_statistic, p_value = stats.kruskal(*groups)

#             # Store the results
#             kw_results[column] = {'Kruskal-Statistic': k_statistic,
'pvalue': p_value}
#         except KeyError:
#             print(f"Column '{column}' not found in the dataset.")

# # Function to interpret the p-value
# def interpret_p_value(p_value):
#     if p_value < 0.05:
#         return "Strong Relationship"
#     elif p_value < 0.1:
#         return "Moderate Relationship"
#     else:
#         return "Weak Relationship"

# # Print the results
# for column, results in kw_results.items():
#     p_value = results['p-value']
#     relationship_strength = interpret_p_value(p_value)
#     print(f"{column}: {relationship_strength} (p-value: {p_value:.4f})")



# import pandas as pd
# import scipy.stats as stats

# # Define a list of columns to test
# columns_to_test = merged_df.columns.difference(['Age'])  # Exclude 'Age'
# # Create an empty dictionary to store the results
# pbs_results = {}

# # Perform Point-Biserial Correlation for each column
# for column in columns_to_test:
#     try:
#         # Ensure 'Age' is treated as a continuous variable
#         age = merged_df['Age'].astype(float)
#         # Convert the categorical variable to
a binary variable
#         binary_variable = (merged_df[column] ==
merged_df[column].mode()[0]).astype(int)

#         # Calculate Point-Biserial Correlation
#         pbs_corr, p_value = stats.pointbiserialr(binary_variable, age)
#         # Store the results
```

```
#         pbs_results[column] = {'Point-Biserial Correlation': pbs_corr,
'p-value': p_value} #
except KeyError:
#         print(f"Column '{column}' not found in the dataset.")
# # Function to interpret the p-value and correlation strength
# def interpret_p_value(p_value):
#     if p_value < 0.05:
#         return "Strong Relationship"
#     elif p_value < 0.1:
#         return "Moderate Relationship"
#     else:
#         return "Weak Relationship"

# # Print the results
# for column, results in pbs_results.items():
#     p_value = results['p-value']
#     correlation_strength = interpret_p_value(p_value)
#     pbs_corr = results['Point-Biserial Correlation']
#     print(f"{column}: {correlation_strength}, Point-Biserial
Correlation: {pbs_corr:.4f}, p-value: {p_value:.4f}")



"""## Let me explore Age"""

# # Explore Age vs Survived
# g = sns.FacetGrid(merged_df, col='Target')
# g = g.map(sns.distplot, "Age")

# # Filter the DataFrame for Target = 0 and create a distplot
# plt.figure(figsize=(12, 6))
# plt.subplot(1, 2, 1)
# sns.histplot(merged_df[merged_df['Target'] == 0]['Age'], color='blue',
kde=True)
# plt.title('Distribution of Age for Target = 0')

# # Filter the DataFrame for Target = 1 and create a distplot
# plt.subplot(1, 2, 2)
# sns.histplot(merged_df[merged_df['Target'] == 1]['Age'], color='green',
kde=True)
# plt.title('Distribution of Age for Target = 1')

# plt.tight_layout()
# plt.show()

"""**Since I am able to see that most people lie between 30 to 50 range in
years**

**From the above distant plots we are able to notice a notable pattern for
the positive class which we are much interested in ,From the above plots
we can seee and maybe passively conclude that people aged between 0 to 40
```

range are more likely to default in which is the vice vasa state considering persons aged between 40 to 100 years range , So in a nutshell , Age and default rate have inverse relationship

Explore PAYMENT_TYPE
"""

```
# Filter the DataFrame for Target = 0 and create a distplot
plt.figure(figsize=(12, 6)) plt.subplot(1, 2, 1)
sns.histplot(merged_df[merged_df['Target'] == 0]['PAYMENT_TYPE'],
color='blue', kde=True)
plt.title('Distribution of PAYMENT_TYPE for Target = 0')

# Filter the DataFrame for Target = 1 and create a distplot plt.subplot(1,
2, 2)
sns.histplot(merged_df[merged_df['Target'] == 1]['PAYMENT_TYPE'],
color='green', kde=True)
plt.title('Distribution of Gender for Target = 1')

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

"""**I can conclude that payment method is highly skewed to one side having a longer tail and therefore I will drop it because it can cause bias in my final result**

Explore COUNT_OF_BLOCKS
"""

```
# Filter the DataFrame for Target = 0 and create a distplot
plt.figure(figsize=(12, 6)) plt.subplot(1, 2, 1)
sns.histplot(merged_df[merged_df['Target'] == 0]['COUNT_OF_BLOCKS'],
color='blue', kde=True)
plt.title('Distribution of COUNT_OF_BLOCKS for Target = 0')

# Filter the DataFrame for Target = 1 and create a distplot plt.subplot(1,
2, 2)
sns.histplot(merged_df[merged_df['Target'] == 1]['COUNT_OF_BLOCKS'],
color='green', kde=True)
plt.title('Distribution of COUNT_OF_BLOCKS for Target = 1')

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

"""**I can conclude that COUNT_OF_BLOCKS is highly skewed to one side having a longer tail and therefore I will drop it because it can cause bias in my final result**

Explore TRANSACTION_RECIEVERS
"""

```
# Filter the DataFrame for Target = 0 and create a distplot
plt.figure(figsize=(12, 6)) plt.subplot(1, 2, 1)
```

```
sns.histplot(merged_df[merged_df['Target'] == 0]['TRANSACTION_RECIEVERS'],
color='blue', kde=True)
plt.title('Distribution of TRANSACTION_RECIEVERS for Target = 0')

# Filter the DataFrame for Target = 1 and create a distplot plt.subplot(1,
2, 2)
sns.histplot(merged_df[merged_df['Target'] == 1]['TRANSACTION_RECIEVERS'],
color='green', kde=True)
plt.title('Distribution of TRANSACTION_RECIEVERS for Target = 1')

plt.tight_layout() plt.show()

"""## Explore LIMIT"""

# Filter the DataFrame for Target = 0 and create a distplot
plt.figure(figsize=(12, 6)) plt.subplot(1, 2, 1)
sns.histplot(merged_df[merged_df['Target'] == 0]['LIMIT'], color='blue',
kde=True)
plt.title('Distribution of LIMIT for Target = 0')

# Filter the DataFrame for Target = 1 and create a distplot plt.subplot(1,
2, 2)
sns.histplot(merged_df[merged_df['Target'] == 1]['LIMIT'], color='green',
kde=True)
plt.title('Distribution of LIMIT for Target = 1')

plt.tight_layout() plt.show()

"""## Explore PAYMENT_TRXN"""

# Filter the DataFrame for Target = 0 and create a distplot
plt.figure(figsize=(12, 6)) plt.subplot(1, 2, 1)
sns.histplot(merged_df[merged_df['PAYMENT_TRXN'] == 0]['PAYMENT_TRXN'],
color='blue', kde=True)
plt.title('Distribution of PAYMENT_TRXN for Target = 0')

# Filter the DataFrame for Target = 1 and create a distplot plt.subplot(1,
2, 2)
sns.histplot(merged_df[merged_df['Target'] == 1]['PAYMENT_TRXN'],
color='green', kde=True)
plt.title('Distribution of PAYMENT_TRXN for Target = 1')
 plt.tight_layout()
plt.show()

"""## Explore PAYMENT_COUNT_2"""

# Filter the DataFrame for Target = 0 and create a distplot
plt.figure(figsize=(12, 6)) plt.subplot(1, 2, 1)
sns.histplot(merged_df[merged_df['PAYMENT_COUNT_2'] ==
0]['PAYMENT_COUNT_2'], color='blue', kde=True)
plt.title('Distribution of PAYMENT_COUNT_2 for Target = 0')
```

```
# Filter the DataFrame for Target = 1 and create a distplot plt.subplot(1, 2, 2)
sns.histplot(merged_df[merged_df['Target'] == 1]['PAYMENT_COUNT_2'], color='green', kde=True)
plt.title('Distribution of PAYMENT_COUNT_2 for Target = 1')
 plt.tight_layout()
plt.show()

"""## Explore LOAN_COUNT"""

# Filter the DataFrame for Target = 0 and create a distplot
plt.figure(figsize=(12, 6)) plt.subplot(1, 2, 1)
sns.histplot(merged_df[merged_df['LOAN_COUNT'] == 0]['LOAN_COUNT'], color='blue', kde=True)
plt.title('Distribution of LOAN_COUNT for Target = 0')

# Filter the DataFrame for Target = 1 and create a distplot plt.subplot(1, 2, 2)
sns.histplot(merged_df[merged_df['Target'] == 1]['LOAN_COUNT'], color='green', kde=True)
plt.title('Distribution of LOAN_COUNT for Target = 1')
 plt.tight_layout()
plt.show()

## Explore PAYMENT_VALUE_2

# Filter the DataFrame for Target = 0 and create a distplot
plt.figure(figsize=(12, 6)) plt.subplot(1, 2, 1)
sns.histplot(merged_df[merged_df['PAYMENT_VALUE_2'] == 0]['PAYMENT_VALUE_2'], color='blue', kde=True)
plt.title('Distribution of PAYMENT_VALUE_2 for Target = 0')

# Filter the DataFrame for Target = 1 and create a distplot plt.subplot(1, 2, 2)
sns.histplot(merged_df[merged_df['Target'] == 1]['PAYMENT_VALUE_2'], color='green', kde=True)
plt.title('Distribution of PAYMENT_VALUE_2 for Target = 1')

plt.tight_layout() plt.show()

"""We can see that te above columns have resembling charts when plotted in relation to the target and from the chi square tests too have same characteristics and non signifficant so I will drop them

**Let me drop some more columns beacause they are too lesser  signifficant to my Target**
"""

# dropping the columns
```

```
merged_df.drop(['PAYMENT_TYPE','COUNT_OF_BLOCKS','TRANSACTION_RECIEVERS','
LIMIT','PAYMENT_TRXN','PAYMENT_COUNT_2','LOAN_COUNT','PAYMENT_VALUE_2','CO
UNT_OF_OVERDUE_LOANS','RECHARGE_MODE'], axis=1, inplace=True)

merged_df.info()

"""Now we are set to go because our data has no missing values anymore"""
# # Define a list of columns to be converted to string
# columns_to_convert = [col for col in merged_df.columns if col not in [
'Age','Gender', 'Target']]

# # Change the data type of selected columns to string
# merged_df[columns_to_convert] =
merged_df[columns_to_convert].astype(str)

merged_df.info()

"""## Detecting outliers"""

import matplotlib.pyplot as plt

# Assuming 'Age' is the column you want to visualize age_data
= merged_df['Age']

# Create a histogram plot to show the distribution of 'Age'
plt.figure(figsize=(12, 6))
plt.hist(age_data, bins=20, color='blue', alpha=0.7, edgecolor='black',
density=True) plt.xlabel('Age') plt.ylabel('Density')
plt.title('Distribution of Age')

plt.tight_layout() plt.show()



# import matplotlib.pyplot as plt
# # Assuming 'Age' is the column you want to visualize
# age_data = merged_df['Age']

# # Create a scatterplot to visualize the distribution of 'Age'
# plt.figure(figsize=(12, 6))
# plt.scatter(age_data, age_data.index, alpha=0.5, color='blue')
# plt.xlabel('Age')
# plt.ylabel('Density')
# plt.title('Scatterplot of Age vs. Density')

# plt.tight_layout()
# plt.show()

"""from the above we can see that we have some customers who are above the
age of 80"""
```

```
import pandas as pd

# Define a function to detect outliers using the IQR method def
detect_outliers(df):
    outliers = []     for col in df.columns:
if pd.api.types.is_numeric_dtype(df[col]):
            Q1 = df[col].quantile(0.25)
            Q3 = df[col].quantile(0.90)
IQR = Q3 - Q1
            lower_bound = Q1 - 1.5 * IQR
upper_bound = Q3 + 1.5 * IQR
            column_outliers = df[(df[col] < lower_bound) | (df[col] >
upper_bound)]
            outliers.append((col, len(column_outliers)))
return outliers

# Detect outliers in the entire DataFrame merged_df outliers
= detect_outliers(merged_df)

# Print the columns with the count of outliers for
col, count in outliers:
    print(f"Column '{col}' has {count} outliers.")

# You can choose a threshold for the count of outliers to identify columns
with significant outliers
# For example, if you want to consider columns with more than 10 outliers
as problematic:
problematic_columns = [col for col, count in outliers if count > 10]
print("Problematic columns with significant outliers:",
problematic_columns)

"""**I set  my  threshhold for the outlier detection to be 0.25 to 0.90
because from my analysis and scatterplot, some people aged more than 70
years are buying phones and making payments. Depending on the context of
your analysis, these data points may not be outliers but rather legitimate
observations. Therefore, it's essential to consider**"""
import pandas as pd

# Assuming merged_df is your DataFrame
# Define a function to remove outliers using the IQR method
def remove_outliers(df, column_name):     Q1 =
df[column_name].quantile(0.25)
    Q3 = df[column_name].quantile(0.90)
IQR = Q3 - Q1
    lower_bound = Q1 - 1.5 * IQR
upper_bound = Q3 + 1.5 * IQR
    df_no_outliers = df[(df[column_name] >= lower_bound) &
(df[column_name] <= upper_bound)]
return df_no_outliers
```

```
# Specify the column you want to check for outliers (e.g., 'Age')
column_to_check = 'Age'

# Remove outliers from the DataFrame
merged_df_no_outliers = remove_outliers(merged_df, column_to_check)

# merged_df_no_outliers now contains the data with outliers removed

"""**therefore, I will drop the one outlier because it comprises a lesser
portion of my dataset**"""

merged_df=merged_df_no_outliers

# merged_df.Age = pd.qcut(merged_df.Age.values, 20).codes
 merged_df

"""SINCE WE HAVE CONFIRMED OUR DATASET HAS NO MISSING VALUES AND OUTLIERS
WE CAN NOW EXPLORE OUR TARGET VARRIABLE

## TARGET DISTRIBUTION
"""


ax = sns.countplot(x = merged_df.Target ,palette="Set3")
sns.set(font_scale=1.5) ax.set_ylim(top = 10000)
ax.set_xlabel('Default') ax.set_ylabel('Frequency') fig
= plt.gcf() fig.set_size_inches(10,6)
ax.set_ylim(top=10000) plt.show()

"""## Let's Explore some of our Variables

## Let's Explore Age Now
"""
# Explore Age vs Target
g = sns.FacetGrid(merged_df, col='Target') g
= g.map(sns.distplot, "Age")



"""**From the above distant plots we are able to notice a notable pattern
for the positive class which we are much interested in ,From the above
plots we can seee and maybe passively conclude that people aged between 0
to 50 range  are likely to default in  which is the vice vasa state
considering persons aged between 50 to 100 years range , So in a nutshell
, Age and default rate have inverse relationship"""



"""we can see also an inverse relationshop between the count of overdue
loans and the target varriable
```

```
## Let's Explore NUMBER_OF_PAYMENT_TRXN
"""

# Explore COUNT_OF_OVERDUE_LOANS vs Target with wider spacing g =
sns.FacetGrid(merged_df, col='Target', col_wrap=2)  # Set col_wrap to
control the number of columns
g = g.map(sns.distplot, "NUMBER_OF_PAYMENT_TRXN")

# Show the plot plt.show()



"""## Explore ACCOUNT_DEBIT"""

# Explore ACCOUNT_DEBIT vs Target with wider spacing
g = sns.FacetGrid(merged_df, col='Target', col_wrap=2)  # Set col_wrap to
control the number of columns
g = g.map(sns.distplot, "ACCOUNT_DEBIT")

# Show the plot plt.show()



"""## Explore ACCOUNT_CREDIT"""

# Explore ACCOUNT_CREDIT vs Target with wider spacing
g = sns.FacetGrid(merged_df, col='Target', col_wrap=2)  # Set col_wrap to
control the number of columns
g = g.map(sns.distplot, "ACCOUNT_CREDIT")

# Show the plot plt.show()

"""## Explore SUBSCRIPTION_PERIOD"""

# Explore SUBSCRIPTION_PERIOD vs Target with wider spacing
g = sns.FacetGrid(merged_df, col='Target', col_wrap=2)  # Set col_wrap to
control the number of columns
g = g.map(sns.distplot, "SUBSCRIPTION_PERIOD")

# Show the plot plt.show()

## Explore TOTAL_BANK_TRANS

# Explore TOTAL_BANK_TRANS vs Target with wider spacing
g = sns.FacetGrid(merged_df, col='Target', col_wrap=2)  # Set col_wrap to
control the number of columns
g = g.map(sns.distplot, "TOTAL_BANK_TRANS")

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

```
# dropping the columns
merged_df.drop(['SUBSCRIPTION_PERIOD'], axis=1, inplace=True)

merged_df.info()

# categorical_columns = [
#      'TRANSACTION_PERIOD',
#     'TOPUP_CONSUMPTION', 'TOPUP_CONSUMPTION_6', 'ACTIVE_STATUS',
#     'TRANSACTIONS_RECIEVED', 'ACCOUNT_DEBIT', 'ACCOUNT_CREDIT',
#     'NUMBER_OF_PAYMENT_TRXN', 'VALUE_SENT', 'HIGHEST_VALUE',
#     'VALUE_RECIEVED', 'TRXN_COUNT', 'BANK_TRANS', 'OTHER_RECEIPTS',
#     'TOTAL_BANK_TRANS', 'TOTAL_PAYMENTS', 'TOTAL_MONEY_IN_3',
#     'TOTAL_MONEY_OUT_3', 'CURR_POINTS', 'RDM_POINTS_6',
#     'LOAN_BLACKLIST_DAYS', 'CURR_DEVICE_MAKE', 'CURR_DEVICE_RRP'
# ]

# # Convert the specified columns to the 'category' data type
# merged_df[categorical_columns] =
merged_df[categorical_columns].astype('category')
 merged_df.info()

"""## Let us do final null check before splitting our data into train and
test"""
merged_df.isnull().sum()

# Get the count of missing values (NaN) for each column missing_values
= merged_df.isnull().sum()

# Print the unique values and missing value counts for each column for
column in merged_df.columns:
    unique_values = merged_df[column].unique()
    unique_count = len(unique_values)
missing_count = missing_values[column]
print(f"Column '{column}':")
    print(f"Unique Values ({unique_count} unique values):")
print(unique_values)
    print(f"Missing Values Count: {missing_count}")
print("\n")

# List the column names in your dataset print(merged_df.columns)



"""## Building our credit scoring model

## Let us split our data into 3, Train, Test and Validation sets
"""
from sklearn.model_selection import train_test_split
```

```
# Splitting the data into train (70%) and temporary (30%) train_df,
temp_df = train_test_split(merged_df, test_size=0.3,
random_state=42)

# Further splitting the temporary data into validation (15%) and test
(15%)
validation_df, test_df = train_test_split(temp_df, test_size=0.5,
random_state=42)

# Printing the shapes of the resulting datasets print(f"Train
data shape: {train_df.shape}") print(f"Validation data shape:
{validation_df.shape}") print(f"Test data shape:
{test_df.shape}")
 train_df

# List the column names in your dataset print(train_df.columns)

# List the column names in your dataset print(validation_df.columns)

# List the column names in your dataset print(test_df.columns)

"""## Let me do one hot ENCODING FOR THE CATEGORICAL COLUMNS"""

# import pandas as pd

# # Assuming merged_df is your DataFrame
# # Create an empty DataFrame to store the one-hot encoded columns
# merged_df_encoded = pd.DataFrame()

# # Loop through each column in your DataFrame and one-hot encode it # for
column in merged_df.columns:
#     if merged_df[column].dtype == 'object':
#         one_hot = pd.get_dummies(merged_df[column], prefix=column) #
merged_df_encoded = pd.concat([merged_df_encoded, one_hot], axis=1)
#     else:
#         merged_df_encoded = pd.concat([merged_df_encoded,
merged_df[column]], axis=1)

# # Now, merged_df_encoded contains all columns one-hot encoded

columns_to_encode = ['Gender', 'LOAN_BLACKLIST']

# Assuming you have 'train_df', 'validation_df', and 'test_df' train_df
= pd.get_dummies(train_df, columns=columns_to_encode,
prefix=columns_to_encode, drop_first=True)
validation_df = pd.get_dummies(validation_df, columns=columns_to_encode,
prefix=columns_to_encode, drop_first=True)
test_df = pd.get_dummies(test_df, columns=columns_to_encode,
prefix=columns_to_encode, drop_first=True)

train_df.info()
```

```
 train_df
validation_df.info()

test_df.info()

# Handle missing values in the 'Gender' column using proportions as
previously shown # ...

# Check for missing values
missing_values_train = train_df.isnull().sum().sum()
missing_values_validation = validation_df.isnull().sum().sum()
missing_values_test = test_df.isnull().sum().sum() print(f"Missing
values in train: {missing_values_train}") print(f"Missing values in
validation: {missing_values_validation}") print(f"Missing values in
test: {missing_values_test}")

# Split features and target for training set
X_train = train_df.drop(columns=['Target']) y_train
= train_df['Target']

# Split features and target for validation set X_validation
= validation_df.drop(columns=['Target']) y_validation =
validation_df['Target']

from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, precision_score, recall_score,
f1_score

# Assuming you have one-hot encoded 'train_df', 'validation_df', and
'test_df'

# Split the data into features (X) and the target (y)
X_train, y_train = train_df.drop(columns=['Target']), train_df['Target']
X_validation, y_validation = validation_df.drop(columns=['Target']),
validation_df['Target']
X_test, y_test = test_df.drop(columns=['Target']), test_df['Target']

# Initialize and train the Random Forest model random_forest
= RandomForestClassifier(random_state=42)
random_forest.fit(X_train, y_train)

# Make predictions on the validation set
y_validation_pred = random_forest.predict(X_validation)

# Calculate accuracy for the validation set
validation_accuracy = accuracy_score(y_validation, y_validation_pred)
print("Validation Accuracy:", validation_accuracy)

# Calculate precision, recall, and F1 score for the validation set
validation_precision = precision_score(y_validation, y_validation_pred)
validation_recall = recall_score(y_validation, y_validation_pred)
```

```
validation_f1 = f1_score(y_validation, y_validation_pred)
print("Validation Precision:", validation_precision) print("Validation
Recall:", validation_recall) print("Validation F1 Score:",
validation_f1)

# Make predictions on the test set y_test_pred
= random_forest.predict(X_test)

# Calculate accuracy for the test set test_accuracy
= accuracy_score(y_test, y_test_pred) print("Test
Accuracy:", test_accuracy)

# Calculate precision, recall, and F1 score for the test set
test_precision = precision_score(y_test, y_test_pred)
test_recall = recall_score(y_test, y_test_pred) test_f1 =
f1_score(y_test, y_test_pred) print("Test Precision:",
test_precision) print("Test Recall:", test_recall)
print("Test F1 Score:", test_f1)


from sklearn.metrics import confusion_matrix

# Make predictions on the validation set
validation_predictions = random_forest.predict(X_validation)

# Compute the confusion matrix
confusion = confusion_matrix(y_validation, validation_predictions)

# Print the confusion matrix
print("Confusion Matrix:") print(confusion)

# import numpy as np
# import matplotlib.pyplot as plt
# from sklearn.model_selection import learning_curve

# # Define a function to plot the learning curve
# def plot_learning_curve(estimator, title, X, y, ylim=None, cv=None,
#                         n_jobs=None, train_sizes=np.linspace(.1, 1.0,
5)):
#     plt.figure() #
plt.title(title) #     if
ylim is not None:
#         plt.ylim(*ylim)
#     plt.xlabel("Training examples")
#     plt.ylabel("Score")
#     train_sizes, train_scores, test_scores = learning_curve(
#         estimator, X, y, cv=cv, n_jobs=n_jobs, train_sizes=train_sizes)
#     train_scores_mean = np.mean(train_scores, axis=1)
#     train_scores_std = np.std(train_scores, axis=1)
#     test_scores_mean = np.mean(test_scores, axis=1)
#     test_scores_std = np.std(test_scores, axis=1)
```

```
#     plt.grid()

#     plt.fill_between(train_sizes, train_scores_mean - train_scores_std,
#                      train_scores_mean + train_scores_std, alpha=0.1, #
color="r")
#     plt.fill_between(train_sizes, test_scores_mean - test_scores_std, #
test_scores_mean + test_scores_std, alpha=0.1, color="g")
#     plt.plot(train_sizes, train_scores_mean, 'o-', color="r",
#              label="Training score")
#     plt.plot(train_sizes, test_scores_mean, 'o-', color="g",
#              label="Cross-validation score")

#     plt.legend(loc="best")
#     return plt

# # You can use this function to plot the learning curve for your model
# plot_learning_curve(random_forest, "Learning Curve", X_train, y_train,
cv=5) # plt.show()

import numpy as np import
matplotlib.pyplot as plt
from sklearn.model_selection import learning_curve

# Create a function to plot the learning curve def
plot_learning_curve(estimator, X, y, title, cv=None,
train_sizes=np.linspace(0.1, 1.0, 5)):
    plt.figure()
plt.title(title)
plt.xlabel("Training examples")
plt.ylabel("Score")
     train_sizes, train_scores, test_scores =
learning_curve(        estimator, X, y, cv=cv,
train_sizes=train_sizes, scoring='accuracy'
    )      train_scores_mean = np.mean(train_scores,
axis=1)     train_scores_std = np.std(train_scores,
axis=1)     test_scores_mean = np.mean(test_scores,
axis=1)     test_scores_std = np.std(test_scores,
axis=1)

    plt.grid()
     plt.fill_between(train_sizes, train_scores_mean - train_scores_std,
train_scores_mean        +        train_scores_std,        alpha=0.1,
color="r")
    plt.fill_between(train_sizes, test_scores_mean - test_scores_std,
test_scores_mean + test_scores_std, alpha=0.1, color="g")
    plt.plot(train_sizes, train_scores_mean, 'o-', color="r",
label="Training score")
    plt.plot(train_sizes, test_scores_mean, 'o-', color="g",
label="Cross-validation score")
```

```
    plt.legend(loc="best")
return plt

# Assuming you have defined and trained the 'random_forest' model as shown
in your previous code

# Combine the training and validation sets for the learning curve
X_combined = pd.concat([X_train, X_validation]) y_combined =
pd.concat([y_train, y_validation])

# Plot the learning curve
plot_learning_curve(random_forest, X_combined, y_combined, "Learning Curve
(Random Forest)") plt.show()



"""## HyperParameter Tunning"""

from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, precision_score, recall_score,
f1_score

# Assuming you have one-hot encoded 'train_df', 'validation_df', and
'test_df'

# Split the data into features (X) and the target (y)
X_train, y_train = train_df.drop(columns=['Target']), train_df['Target']
X_validation, y_validation = validation_df.drop(columns=['Target']),
validation_df['Target']
X_test, y_test = test_df.drop(columns=['Target']), test_df['Target']

# Initialize the Random Forest model
random_forest = RandomForestClassifier(random_state=42)

# Define a range of hyperparameters to search param_grid
= {
    'n_estimators': [100, 200, 300],
'max_depth': [None, 10, 20],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4]
}

# Create a Grid Search model
grid_search = GridSearchCV(estimator=random_forest, param_grid=param_grid,
cv=3, scoring='accuracy')

# Fit the model to find the best hyperparameters grid_search.fit(X_train,
y_train)
```

```
# Get the best estimator and hyperparameters
best_estimator = grid_search.best_estimator_ best_params
= grid_search.best_params_
 print("Best Estimator:",
best_estimator) print("Best
Parameters:", best_params)

# Make predictions on the validation set with the best model
y_validation_pred = best_estimator.predict(X_validation)

# Calculate accuracy for the validation set
validation_accuracy = accuracy_score(y_validation, y_validation_pred)
print("Validation Accuracy:", validation_accuracy)

# Calculate precision, recall, and F1 score for the validation set
validation_precision = precision_score(y_validation, y_validation_pred)
validation_recall = recall_score(y_validation, y_validation_pred)
validation_f1 = f1_score(y_validation, y_validation_pred)
print("Validation Precision:", validation_precision) print("Validation
Recall:", validation_recall) print("Validation F1 Score:",
validation_f1)

# Make predictions on the test set with the best model y_test_pred
= best_estimator.predict(X_test)

# Calculate accuracy for the test set
test_accuracy = accuracy_score(y_test, y_test_pred) print("Test
Accuracy:", test_accuracy)

# Calculate precision, recall, and F1 score for the test set
test_precision = precision_score(y_test, y_test_pred) test_recall
= recall_score(y_test, y_test_pred)
test_f1 = f1_score(y_test, y_test_pred)
print("Test Precision:", test_precision)
print("Test Recall:", test_recall) print("Test
F1 Score:", test_f1)

from sklearn.model_selection import RandomizedSearchCV

# Initialize the Random Forest model
random_forest = RandomForestClassifier(random_state=42)

# Define a range of hyperparameters to search param_distributions
= {
    'n_estimators': [100, 200, 300],
    'max_depth': [None, 10, 20],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4]
}
```

```
# Create a Randomized Search model random_search
= RandomizedSearchCV(
estimator=random_forest,
    param_distributions=param_distributions,
    n_iter=10,  # Adjust the number of iterations as needed
cv=3,     scoring='accuracy',     random_state=42
)

# Fit the model to find the best hyperparameters
random_search.fit(X_train, y_train)

# Get the best estimator and hyperparameters
best_estimator = random_search.best_estimator_ best_params
= random_search.best_params_
 print("Best Estimator:",
best_estimator) print("Best
Parameters:", best_params)

# Make predictions on the validation set with the best model
y_validation_pred = best_estimator.predict(X_validation)

# Calculate accuracy for the validation set
validation_accuracy = accuracy_score(y_validation, y_validation_pred)
print("Validation Accuracy:", validation_accuracy)

# Calculate precision, recall, and F1 score for the validation set
validation_precision = precision_score(y_validation, y_validation_pred)
validation_recall = recall_score(y_validation, y_validation_pred)
validation_f1 = f1_score(y_validation, y_validation_pred)
print("Validation Precision:", validation_precision) print("Validation
Recall:", validation_recall) print("Validation F1 Score:",
validation_f1)

# Make predictions on the test set with the best model
y_test_pred = best_estimator.predict(X_test)

# Calculate accuracy for the test set test_accuracy
= accuracy_score(y_test, y_test_pred) print("Test
Accuracy:", test_accuracy)

# Calculate precision, recall, and F1 score for the test set
test_precision = precision_score(y_test, y_test_pred)
test_recall = recall_score(y_test, y_test_pred) test_f1 =
f1_score(y_test, y_test_pred) print("Test Precision:",
test_precision) print("Test Recall:", test_recall)
print("Test F1 Score:", test_f1)



# Make predictions on the validation set with the best model
y_validation_pred = best_estimator.predict(X_validation)
```

```
# Calculate accuracy for the validation set
validation_accuracy = accuracy_score(y_validation, y_validation_pred)
print("Validation Accuracy:", validation_accuracy)

# Calculate precision, recall, and F1 score for the validation set
validation_precision = precision_score(y_validation, y_validation_pred)
validation_recall = recall_score(y_validation, y_validation_pred)
validation_f1 = f1_score(y_validation, y_validation_pred)
print("Validation Precision:", validation_precision) print("Validation
Recall:", validation_recall) print("Validation F1 Score:",
validation_f1)

# Make predictions on the test set with the best model y_test_pred
= best_estimator.predict(X_test)

# Calculate accuracy for the test set test_accuracy
= accuracy_score(y_test, y_test_pred) print("Test
Accuracy:", test_accuracy)

# Calculate precision, recall, and F1 score for the test set
test_precision = precision_score(y_test, y_test_pred)
test_recall = recall_score(y_test, y_test_pred) test_f1 =
f1_score(y_test, y_test_pred) print("Test Precision:",
test_precision) print("Test Recall:", test_recall)
print("Test F1 Score:", test_f1)


from sklearn.metrics import confusion_matrix

# Calculate the confusion matrix for the validation set
validation_confusion_matrix = confusion_matrix(y_validation,
y_validation_pred)
print("Validation Confusion Matrix:")
print(validation_confusion_matrix)

# Calculate the confusion matrix for the test set
test_confusion_matrix = confusion_matrix(y_test, y_test_pred)
print("Test Confusion Matrix:") print(test_confusion_matrix)

# import joblib

# # Assuming you have the best hyperparameters from your tuning process
# best_hyperparameters = {
#     'max_depth': 20,
#     'min_samples_split': 5,
#     'n_estimators': 300,
#     'random_state': 42
# }

# # Create and train the tuned model using the best hyperparameters
```

```
# tuned_model = RandomForestClassifier(**best_hyperparameters)
# tuned_model.fit(X_train, y_train)

# # Save the tuned model to a file
# joblib.dump(tuned_model, 'tuned_random_forest.pkl')

import pandas as pd
from sklearn.ensemble import RandomForestClassifier import
joblib

# Assuming you have the best hyperparameters from your tuning process
best_hyperparameters = {     'max_depth': 20,
    'min_samples_split': 5,
    'n_estimators': 300,
    'random_state': 42
}
# Train the model using the original X_train DataFrame (with 'msisdn' as
index)
tuned_model = RandomForestClassifier(**best_hyperparameters)
tuned_model.fit(X_train, y_train)

# Save the tuned model to a file
joblib.dump(tuned_model, 'tuned_random_forest.pkl')

# Reset the index of X_train to convert 'msisdn' from index to a regular
column
X_train_reset_index = X_train.reset_index()

# Assuming 'msisdn' is a column after resetting the index
Msisdn_train = X_train_reset_index['Msisdn']

# Reset the index of X_test and X_validation to access the 'Msisdn' column
Msisdn_test = X_test.reset_index()['Msisdn']
Msisdn_validation = X_validation.reset_index()['Msisdn']

# Concatenate 'Msisdn' values from X_train, X_test, and X_validation
Msisdn_all = pd.concat([Msisdn_train, Msisdn_test, Msisdn_validation],
ignore_index=True)

# Save the combined 'Msisdn' values to a CSV file
Msisdn_all.to_csv('Msisdn.csv', index=False)




import numpy as np import
matplotlib.pyplot as plt
from sklearn.model_selection import learning_curve

# Define a function to plot the learning curve def
plot_learning_curve(estimator, title, X, y, cv, train_sizes):
```

```
    plt.figure()
plt.title(title)
    plt.xlabel("Training examples")
plt.ylabel("Score")
    train_sizes, train_scores, test_scores = learning_curve(estimator, X,
y, cv=cv, train_sizes=train_sizes)
    train_scores_mean = np.mean(train_scores, axis=1)
train_scores_std = np.std(train_scores, axis=1)
test_scores_mean = np.mean(test_scores, axis=1)     test_scores_std
= np.std(test_scores, axis=1)
     plt.fill_between(train_sizes, train_scores_mean - train_scores_std,
train_scores_mean         +         train_scores_std,         alpha=0.1,
color="r")
    plt.fill_between(train_sizes, test_scores_mean - test_scores_std,
test_scores_mean + test_scores_std, alpha=0.1, color="g")
    plt.plot(train_sizes, train_scores_mean, 'o-', color="r",
label="Training score")
    plt.plot(train_sizes, test_scores_mean, 'o-', color="g",
label="Cross-validation score")

    plt.legend(loc="best")
return plt

# Define your hyperparameters
best_hyperparameters = {
'max_depth': 20,
    'min_samples_split': 5,
    'n_estimators': 300,
    'random_state': 42
}

# Create the model with the best hyperparameters
tuned_model = RandomForestClassifier(**best_hyperparameters)
# Specify your desired training sizes for the learning curve
train_sizes = np.linspace(0.1, 1.0, 10)

# Plot the learning curve
plot_learning_curve(tuned_model, "Learning Curve", X_train, y_train, cv=5,
train_sizes=train_sizes)

plt.show()



# import joblib

# # Assuming you have already trained your Random Forest model
(random_forest)
# # Save your trained model to a file
# joblib.dump(random_forest, 'random_forest.pkl')
```

```
# # To load the model back later:
# # loaded_model = joblib.load('random_forest_model.pkl')

# # Now you can use the loaded_model for predictions

# import joblib
# from sklearn.ensemble import RandomForestClassifier

# # Assuming you have already trained your Random Forest model
(random_forest)
# # Save your trained model with the .sav extension
# joblib.dump(random_forest, 'random_forest.sav')

# import xgboost as xgb
# from sklearn.metrics import accuracy_score, precision_score,
recall_score, f1_score

# # Assuming you have one-hot encoded 'train_df', 'validation_df', and
'test_df'

# # Split the data into features (X) and the target (y)
# X_train, y_train = train_df.drop(columns=['Target']), train_df['Target']
# X_validation, y_validation = validation_df.drop(columns=['Target']),
validation_df['Target']
# X_test, y_test = test_df.drop(columns=['Target']), test_df['Target']

# # Initialize and train the XGBoost model
# xgb_model = xgb.XGBClassifier(random_state=42)
# xgb_model.fit(X_train, y_train)

# # Make predictions on the validation set
# y_validation_pred = xgb_model.predict(X_validation)

# # Calculate accuracy for the validation set
# validation_accuracy = accuracy_score(y_validation, y_validation_pred)
# print("Validation Accuracy:", validation_accuracy)

# # Calculate precision, recall, and F1 score for the validation set
# validation_precision = precision_score(y_validation, y_validation_pred)
# validation_recall = recall_score(y_validation, y_validation_pred)
# validation_f1 = f1_score(y_validation, y_validation_pred)
# print("Validation Precision:", validation_precision)
# print("Validation Recall:", validation_recall)
# print("Validation F1 Score:", validation_f1)

# # Make predictions on the test set
# y_test_pred = xgb_model.predict(X_test)

# # Calculate accuracy for the test set
# test_accuracy = accuracy_score(y_test, y_test_pred)
# print("Test Accuracy:", test_accuracy)
```

```
# # Calculate precision, recall, and F1 score for the test set
# test_precision = precision_score(y_test, y_test_pred)
# test_recall = recall_score(y_test, y_test_pred)
# test_f1 = f1_score(y_test, y_test_pred)
# print("Test Precision:", test_precision)
# print("Test Recall:", test_recall)
# print("Test F1 Score:", test_f1)



# from sklearn.metrics import confusion_matrix

# # Make predictions on the validation set
# validation_predictions = xgb_model.predict(X_validation)

# # Compute the confusion matrix
# confusion = confusion_matrix(y_validation, validation_predictions)

# # Print the confusion matrix
# print("Confusion Matrix:")
# print(confusion)


from sklearn.naive_bayes import MultinomialNB
from sklearn.metrics import accuracy_score, precision_score, recall_score,
f1_score

# Assuming you have one-hot encoded 'train_df', 'validation_df', and
'test_df'

# Split the data into features (X) and the target (y)
X_train, y_train = train_df.drop(columns=['Target']), train_df['Target']
X_validation, y_validation = validation_df.drop(columns=['Target']),
validation_df['Target']
X_test, y_test = test_df.drop(columns=['Target']), test_df['Target']

# Initialize and train the Multinomial Naive Bayes model
nb_model = MultinomialNB() nb_model.fit(X_train,
y_train)

# Make predictions on the validation set
y_validation_pred = nb_model.predict(X_validation)

# Calculate accuracy for the validation set
validation_accuracy = accuracy_score(y_validation, y_validation_pred)
print("Validation Accuracy:", validation_accuracy)

# Calculate precision, recall, and F1 score for the validation set
validation_precision = precision_score(y_validation, y_validation_pred)
validation_recall = recall_score(y_validation, y_validation_pred)
```

```
validation_f1 = f1_score(y_validation, y_validation_pred)
print("Validation Precision:", validation_precision) print("Validation
Recall:", validation_recall) print("Validation F1 Score:",
validation_f1)

# Make predictions on the test set y_test_pred
= nb_model.predict(X_test)

# Calculate accuracy for the test set
test_accuracy = accuracy_score(y_test, y_test_pred) print("Test
Accuracy:", test_accuracy)

# Calculate precision, recall, and F1 score for the test set
test_precision = precision_score(y_test, y_test_pred)
test_recall = recall_score(y_test, y_test_pred) test_f1 =
f1_score(y_test, y_test_pred) print("Test Precision:",
test_precision) print("Test Recall:", test_recall)
print("Test F1 Score:", test_f1)


from sklearn.metrics import confusion_matrix

# Make predictions on the validation set
validation_predictions = nb_model.predict(X_validation)

# Compute the confusion matrix
confusion = confusion_matrix(y_validation, validation_predictions)

# Print the confusion matrix
print("Confusion Matrix:") print(confusion)


from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score, precision_score, recall_score,
f1_score

# Assuming you have one-hot encoded 'train_df', 'validation_df', and
'test_df'

# Split the data into features (X) and the target (y)
X_train, y_train = train_df.drop(columns=['Target']), train_df['Target']
X_validation, y_validation = validation_df.drop(columns=['Target']),
validation_df['Target']
X_test, y_test = test_df.drop(columns=['Target']), test_df['Target']

# Initialize and train the Decision Tree model decision_tree
= DecisionTreeClassifier(random_state=42)
decision_tree.fit(X_train, y_train)

# Make predictions on the validation set
y_validation_pred = decision_tree.predict(X_validation)
```

```
# Calculate accuracy for the validation set
validation_accuracy = accuracy_score(y_validation, y_validation_pred)
print("Validation Accuracy:", validation_accuracy)

# Calculate precision, recall, and F1 score for the validation set
validation_precision = precision_score(y_validation, y_validation_pred)
validation_recall = recall_score(y_validation, y_validation_pred)
validation_f1 = f1_score(y_validation, y_validation_pred)
print("Validation Precision:", validation_precision) print("Validation
Recall:", validation_recall) print("Validation F1 Score:",
validation_f1)

# Make predictions on the test set y_test_pred
= decision_tree.predict(X_test)

# Calculate accuracy for the test set test_accuracy
= accuracy_score(y_test, y_test_pred) print("Test
Accuracy:", test_accuracy)

# Calculate precision, recall, and F1 score for the test set
test_precision = precision_score(y_test, y_test_pred)
test_recall = recall_score(y_test, y_test_pred) test_f1 =
f1_score(y_test, y_test_pred) print("Test Precision:",
test_precision) print("Test Recall:", test_recall)
print("Test F1 Score:", test_f1)


from sklearn.metrics import confusion_matrix

# Make predictions on the validation set
validation_predictions = decision_tree.predict(X_validation)

# Compute the confusion matrix
confusion = confusion_matrix(y_validation, validation_predictions)

# Print the confusion matrix
print("Confusion Matrix:") print(confusion)


from sklearn.ensemble import GradientBoostingClassifier
from sklearn.metrics import accuracy_score, precision_score, recall_score,
f1_score

# Assuming you have one-hot encoded 'train_df', 'validation_df', and
'test_df'

# Split the data into features (X) and the target (y)
X_train, y_train = train_df.drop(columns=['Target']), train_df['Target']
X_validation, y_validation = validation_df.drop(columns=['Target']),
validation_df['Target']
```

```
X_test, y_test = test_df.drop(columns=['Target']), test_df['Target']

# Initialize the Gradient Boosting model
gradient_boosting = GradientBoostingClassifier(n_estimators=100,
random_state=42)
gradient_boosting.fit(X_train, y_train)

# Make predictions on the validation set
y_validation_pred = gradient_boosting.predict(X_validation)

# Calculate accuracy for the validation set
validation_accuracy = accuracy_score(y_validation, y_validation_pred)
print("Validation Accuracy:", validation_accuracy)

# Calculate precision, recall, and F1 score for the validation set
validation_precision = precision_score(y_validation, y_validation_pred)
validation_recall = recall_score(y_validation, y_validation_pred)
validation_f1 = f1_score(y_validation, y_validation_pred)
print("Validation Precision:", validation_precision) print("Validation
Recall:", validation_recall) print("Validation F1 Score:",
validation_f1)

# Make predictions on the test set
y_test_pred = gradient_boosting.predict(X_test)

# Calculate accuracy for the test set
test_accuracy = accuracy_score(y_test, y_test_pred) print("Test
Accuracy:", test_accuracy)

# Calculate precision, recall, and F1 score for the test set
test_precision = precision_score(y_test, y_test_pred)
test_recall = recall_score(y_test, y_test_pred) test_f1 =
f1_score(y_test, y_test_pred) print("Test Precision:",
test_precision) print("Test Recall:", test_recall)
print("Test F1 Score:", test_f1)


from sklearn.metrics import confusion_matrix

# Make predictions on the validation set
validation_predictions = gradient_boosting.predict(X_validation)

# Compute the confusion matrix
confusion = confusion_matrix(y_validation, validation_predictions)

# Print the confusion matrix
print("Confusion Matrix:") print(confusion)

  from sklearn.ensemble import
AdaBoostClassifier from sklearn.tree import
DecisionTreeClassifier
```

```
from sklearn.metrics import accuracy_score, precision_score, recall_score,
f1_score

# Assuming you have one-hot encoded 'train_df', 'validation_df', and
'test_df'

# Split the data into features (X) and the target (y)
X_train, y_train = train_df.drop(columns=['Target']), train_df['Target']
X_validation, y_validation = validation_df.drop(columns=['Target']),
validation_df['Target']
X_test, y_test = test_df.drop(columns=['Target']), test_df['Target']

# Initialize the base Decision Tree model
base_model = DecisionTreeClassifier(random_state=42)

# Initialize and train the AdaBoost model using the base model adaboost
= AdaBoostClassifier(base_model, n_estimators=100, random_state=42)
adaboost.fit(X_train, y_train)

# Make predictions on the validation set
y_validation_pred = adaboost.predict(X_validation)

# Calculate accuracy for the validation set
validation_accuracy = accuracy_score(y_validation, y_validation_pred)
print("Validation Accuracy:", validation_accuracy)

# Calculate precision, recall, and F1 score for the validation set
validation_precision = precision_score(y_validation, y_validation_pred)
validation_recall = recall_score(y_validation, y_validation_pred)
validation_f1 = f1_score(y_validation, y_validation_pred)
print("Validation Precision:", validation_precision) print("Validation
Recall:", validation_recall) print("Validation F1 Score:",
validation_f1)

# Make predictions on the test set y_test_pred
= adaboost.predict(X_test) # Calculate
accuracy for the test set test_accuracy =
accuracy_score(y_test, y_test_pred)
print("Test Accuracy:", test_accuracy)

# Calculate precision, recall, and F1 score for the test set
test_precision = precision_score(y_test, y_test_pred)
test_recall = recall_score(y_test, y_test_pred) test_f1 =
f1_score(y_test, y_test_pred) print("Test Precision:",
test_precision) print("Test Recall:", test_recall)
print("Test F1 Score:", test_f1)


from sklearn.metrics import confusion_matrix

# Make predictions on the validation set
```

```
validation_predictions = adaboost.predict(X_validation)

# Compute the confusion matrix
confusion = confusion_matrix(y_validation, validation_predictions)

# Print the confusion matrix
print("Confusion Matrix:") print(confusion)

  from sklearn.model_selection import cross_val_score,
KFold from sklearn.ensemble import RandomForestClassifier

# Define the number of folds and cross-validation splitter num_folds
= 5
kfold = KFold(n_splits=num_folds, shuffle=True, random_state=42)

# Define your feature data (X) and target data (y) - Ensure train_df is
defined and populated
X = train_df.drop(columns=['Target']) y
= train_df['Target']

# Select your model
model = RandomForestClassifier(random_state=42)

# Perform cross-validation
scores = cross_val_score(model, X, y, cv=kfold, scoring='accuracy')

# Analyze cross-validation results
mean_accuracy = scores.mean() std_accuracy
= scores.std()
 print("Mean Accuracy:", mean_accuracy)
print("Standard Deviation:", std_accuracy)
"""## let  us check Variable Importance"""

clf = RandomForestClassifier(n_estimators=50, max_features='sqrt') clf
= clf.fit(X_train, y_train)

features = pd.DataFrame() features['feature'] =
X_train.columns features['importance'] =
clf.feature_importances_
features.sort_values(by=['importance'], ascending=True, inplace=True)
features.set_index('feature', inplace=True)

import matplotlib.pyplot as plt

# Assuming you have already created the 'features' DataFrame and want to
create a horizontal bar plot
# Adjust the 'fontsize' parameter as needed ax =
features.plot(kind='barh', figsize=(10, 50))
ax.set_ylabel("Feature", fontsize=30)  # Adjust the fontsize as needed
plt.show()
```

```
import pandas as pd

# Assuming you have binary predictions for train, validation, and test
datasets
# You can create a DataFrame with all data rows and their respective
predicted values

# Example binary predictions (replace with your actual binary predictions)
train_predictions = loaded_model.predict(train_df)
validation_predictions = loaded_model.predict(validation_df)
test_predictions = loaded_model.predict(test_df)

# Example probabilities (replace with your actual probabilities)
train_probabilities = loaded_model.predict_proba(train_df)[:, 1]
validation_probabilities = loaded_model.predict_proba(validation_df)[:, 1]
test_probabilities = loaded_model.predict_proba(test_df)[:, 1]

# Create DataFrames for train, validation, and test datasets
train_df = pd.DataFrame({'Index': range(len(train_predictions)),
'Predicted_Target': train_predictions, 'Probability_of_Defaults':
train_probabilities})
validation_df = pd.DataFrame({'Index': range(len(train_predictions),
len(train_predictions) + len(validation_predictions)), 'Predicted_Target':
validation_predictions, 'Probability_of_Defaults':
validation_probabilities})
test_df = pd.DataFrame({'Index': range(len(train_predictions) +
len(validation_predictions), len(train_predictions) +
len(validation_predictions) + len(test_predictions)), 'Predicted_Target':
test_predictions, 'Probability_of_Defaults': test_probabilities})

# Combine them into one DataFrame
all_predictions_df = pd.concat([train_df, validation_df, test_df],
ignore_index=True)

# Now, you have a DataFrame 'all_predictions_df' with all data rows, their
respective binary predictions, and probability of defaulting.
# It includes the 'Index', 'Predicted_Target', and
'Probability_of_Defaults' columns.

# Define your score calculation logic here
def calculate_score(probability):
    # For example, you can convert the probability to a score between 0
and 100
    # Adjust this logic based on your specific requirements
    return int(probability * 100)
```

```
# Apply the score calculation to each row
all_predictions_df['Score'] =
all_predictions_df['Probability_of_Defaults'].apply(calculate_score)
```

CUSTOMER SUPPORT CHATBOT

This is a sample code I used in computing a customized customer support chatbot to help me Customer care team in handling customer inquiries therefore improving operational efficiency

```
!pip install fuzzywuzzy
!pip install transformers

import transformers
from datetime import datetime
import pandas as pd
from fuzzywuzzy import process  # For fuzzy matching

# Initialize the text-generation pipeline with multiple models
def init_chatbot():
    print("Initializing chatbot...")

    # DialogGPT model for conversation
    dialog_model_name = "microsoft/DialoGPT-medium"
    dialog_tokenizer =
transformers.AutoTokenizer.from_pretrained(dialog_model_name,
use_fast=True)
    dialog_model =
transformers.AutoModelForCausalLM.from_pretrained(dialog_model_name)
    dialog_chatbot = transformers.pipeline(
        "text-generation",
        model=dialog_model,
        tokenizer=dialog_tokenizer,
        pad_token_id=dialog_tokenizer.eos_token_id
    )

    # Additional GPT-2 model for handling different types of responses
    additional_model_name = "gpt2"
    additional_tokenizer =
transformers.AutoTokenizer.from_pretrained(additional_model_name,
use_fast=True)
```

```
    additional_model =
transformers.AutoModelForCausalLM.from_pretrained(additional_model_name)
    additional_chatbot = transformers.pipeline(
        "text-generation",
        model=additional_model,
        tokenizer=additional_tokenizer,
        pad_token_id=additional_tokenizer.eos_token_id
    )

    print("Chatbot is ready to talk!")
    return dialog_chatbot, additional_chatbot

# Handle greetings based on time of day
def handle_greetings():
    current_hour = datetime.now().hour
    if 5 <= current_hour < 12:
        return "Good morning!"
    elif 12 <= current_hour < 17:
        return "Good afternoon!"
    elif 17 <= current_hour < 21:
        return "Good evening!"
    else:
        return "Hello there!"

# Existing data extracted from the document
data = {
    'Question': [
        'Who is Onfon Mobile?',
        'How do I get a phone with Onfon Mobile?',
        'Where are your devices located?',
        'What do I need when getting an Onfon device?',
        'Who qualifies for an Onfon Mobile device?',
        'What are the Payment Plans available on the Onfon Mobile?',
        'Can I change my payment plan?',
        'How do I pay for my phone loan?',
        'What are the requirements for making loan payments?',
        'What happens if I do not make payments on time.',
        'Is my phone locked immediately or is there a grace period to
pay?',
        'Can I get more than one phone on loan?',
        'Can I make payments for another number?',
        'Is there a till number for making payments?',
        'Can I purchase Airtime with Onfon Mobile?',
        'What should I do if I lose my phone and haven’t finished paying
for the loan?',
```

```
        'Does my phone have a warranty?',
        'Can my phone be repaired after it gets spoilt?',
        'How can I contact Onfon Mobile?',
        'Are there other services under Onfon Mobile?',
        'What phones do you sell?'
    ],
    'Answer': [
        'Onfon Mobile is a device financing company that helps to issue
smartphones to customers on loan services.',
        'To get a smartphone via Onfon Mobile, register with us via *797#.
After dialing *797#, go through the registration process, and you shall
receive a message telling you whether you qualify for a smartphone(s) or
not. Customer registers for the service. Onfon Mobile scores the customer,
and if they qualify, they are shown a list of devices they qualify for. If
you receive a message that you qualify for a phone, you can dial *797#
again to see devices you qualify for. Make your loan payment. Purchase
airtime.',
        'You can access the Onfon Mobile service(s) from Safaricom shops
across the country as well as some Safaricom Dealership shops within the
country.',
        'You will need your national ID for verification within the shop
where you shall be going to get your preferred device.',
        'Everyone who meets the credit scoring requirements can access an
Onfon Mobile phone.',
        'For a device bought under Onfon Mobile, there are payments that
are to be made depending on the payment plan that the customer wants. We
currently have the 12-month loan, and customers can change the plan after
they are enrolled in the 12-month payment plan.',
        'Yes, you can change your payment plan on our Onfon Mobile app.',
        'To make payments for your phone loan, you may use our USSD by
dialing *797# and choosing option 2. Alternatively, a customer may use the
Onfon Mobile Application to make their payment. Within the application,
choose the pay option, and the M-Pesa SDK shall pop up, and the customer
can make their payment. You may also pay using M-Pesa by choosing the Lipa
na Mpesa option, selecting the Paybill option, entering Business Number
622645, and entering the borrower\'s ID/Phone Number.',
        'The customer should ensure that they make their payments as per
their payment plans, failure to do this will lead to their phones being
locked.',
        'Your device will be locked if you do not pay your installments on
time.',
        'You do have a grace period of about 10-15 minutes before your
phone is locked.',
        'No, you can only be issued one phone to pay for on loan. Once you
finish your payment for that phone, you can get access to another phone.',
```

```
        'Yes, you can make payments for another number.',
        'We have a business paybill number. Simply use the Paybill option
on M-Pesa and key in the business number 622645, then use the customer ID
number or phone number as the account number, then enter the amount to be
paid, finish off with your Mpesa password, and you will have made your
payment.',
        'Yes, you can. To purchase airtime, simply dial *797# and choose
the buy airtime option on the USSD.',
        'If you lose your phone, report the case to the police
immediately. Acquire an OB number and a police abstract. Then share the
scanned documents with customercare@onfonmobile.com.',
        'Yes, the phone has a warranty of 1 year from the manufacturer.
Terms and conditions apply.',
        'Yes, for phone damages under warranty, simply visit the nearest
service center and get the assistance you need with repair.',
        'For further inquiries, reach out to Onfon Mobile via our email
customercare@onfonmobile.com or call our customer care line 0709491700 or
find us on all our social media pages as Onfon Mobile.',
        'Yes, we have a referral code program in place that allows our
customers to earn money by referencing other customers to us. You earn a
commission for the customer who registers and purchases from us.',
        'Our phone options cut across all Android devices.'
    ]
}

# Convert existing data to DataFrame
existing_df = pd.DataFrame(data)

# Fuzzy matching function to match queries with existing questions
def fuzzy_match_query(query, df, threshold=80):
    result = process.extractOne(query, df['Question'])
    print(f"Fuzzy match result: {result}")  # Debugging print statement
    if result:
        best_match, score, _ = result  # Unpacking three values
        if score >= threshold:
            matched_answer = df[df['Question'] ==
best_match]['Answer'].values[0]
            return matched_answer
    return None

# Generate a response using the chatbot models
def generate_response(chatbots, query):
    dialog_chatbot, additional_chatbot = chatbots

    # Try DialogGPT first
```

```
    response_dialog = dialog_chatbot(query, max_length=50,
num_return_sequences=1)
    response_dialog_text = response_dialog[0]['generated_text']

    # If DialogGPT response is too generic, try the additional model
    if "sorry" in response_dialog_text.lower() or
len(response_dialog_text) < 20:
        response_additional = additional_chatbot(query, max_length=50,
num_return_sequences=1)
        response_additional_text =
response_additional[0]['generated_text']
        return response_additional_text

    return response_dialog_text

# Combine everything into the chat function
def chat(chatbots, query):
    greeting = handle_greetings()

    # Check if the query matches any existing question using fuzzy
matching
    matched_answer = fuzzy_match_query(query, existing_df)
    if matched_answer:
        return f"{greeting} {matched_answer}"

    # Generate a response using the chatbot models
    response = generate_response(chatbots, query)
    return f"{greeting} {response}"

# Initialize the chatbot
chatbots = init_chatbot()

# Example usage
query = "How do I get a phone with Onfon Mobile?"
response = chat(chatbots, query)
print(response)

# Initialize the chatbot
chatbots = init_chatbot()

# Interactive loop for asking questions
while True:
    query = input("You: ")
    if query.lower() in ["exit", "quit"]:
        print("Chatbot: Goodbye!")
```

```
        break
    response = chat(chatbots, query)
    print(f"Chatbot: {response}")
```

Conclusion

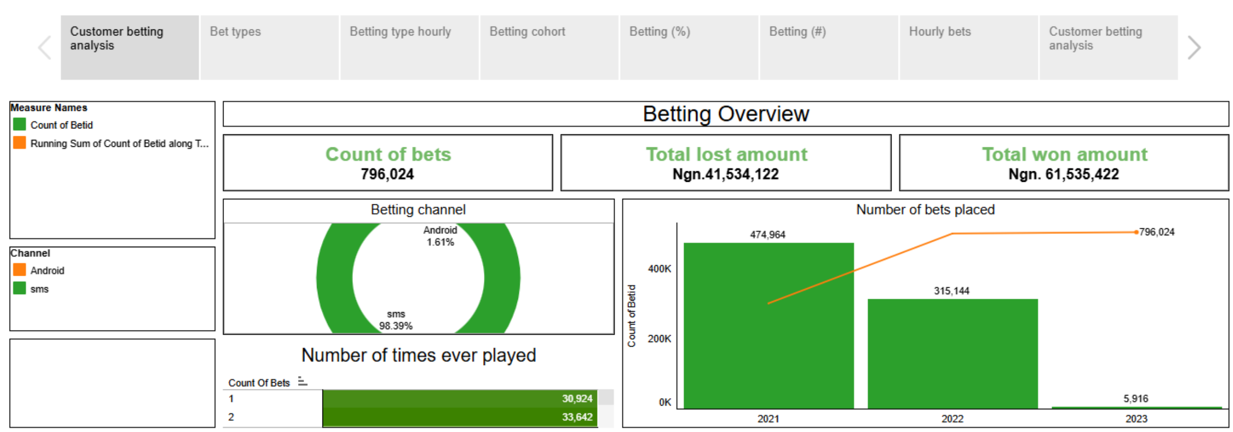
By integrating data analytics, machine learning, and AI, we achieved significant improvements for the device financing companyboosting credit scoring, customer satisfaction, and data-driven decision-making. This project positioned them for sustainable growth and a stronger market presence across Africa.

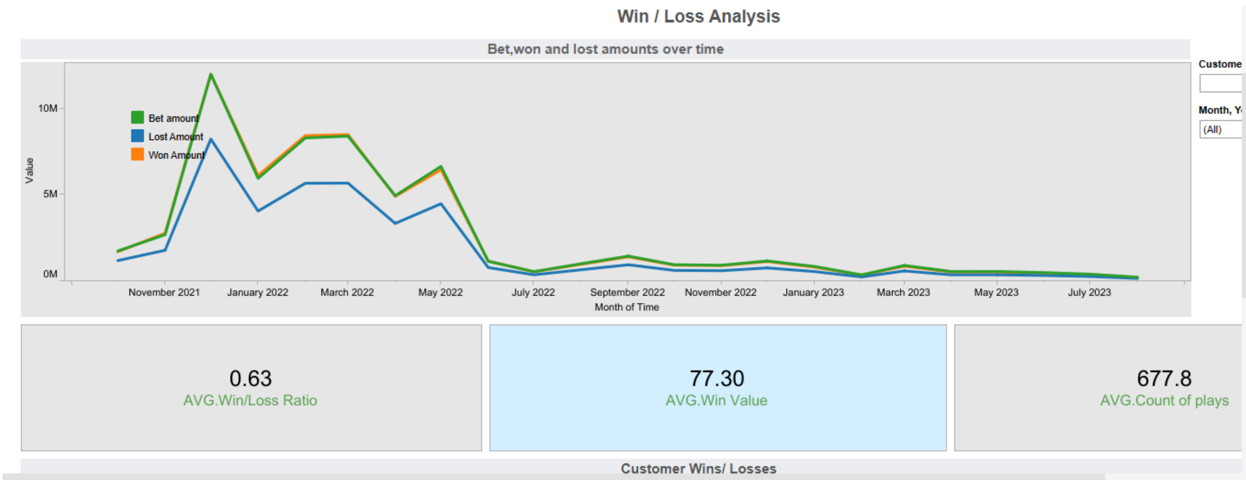
PROJECT 2

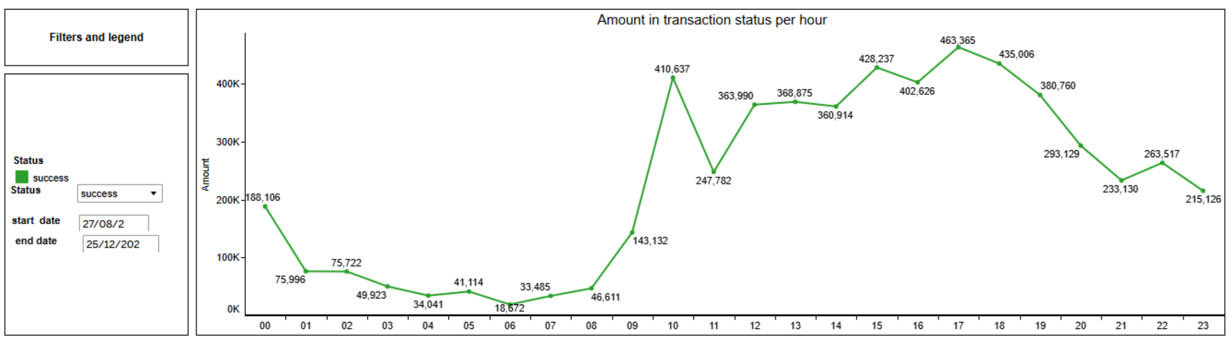
INDUSTRY: Betting and Cassino

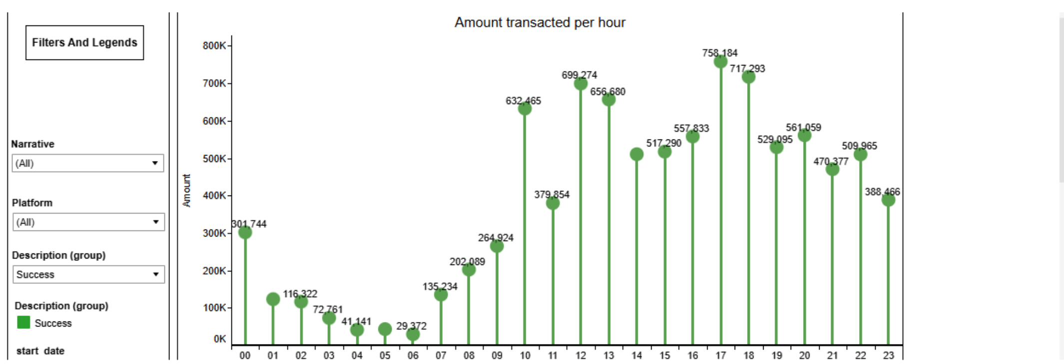
In my previous role , I got the opportunity to work for a betting company , where I computed deposits analyses, customer churn analyses, winning rates, betting analyses, cashflow analyses , win /loss analyses, house cashflow, gross gaming revenues, house cashflows ,transactions ,customer conversion rates , Cashflows forecasting machine learning models and onboarding and churn rates.

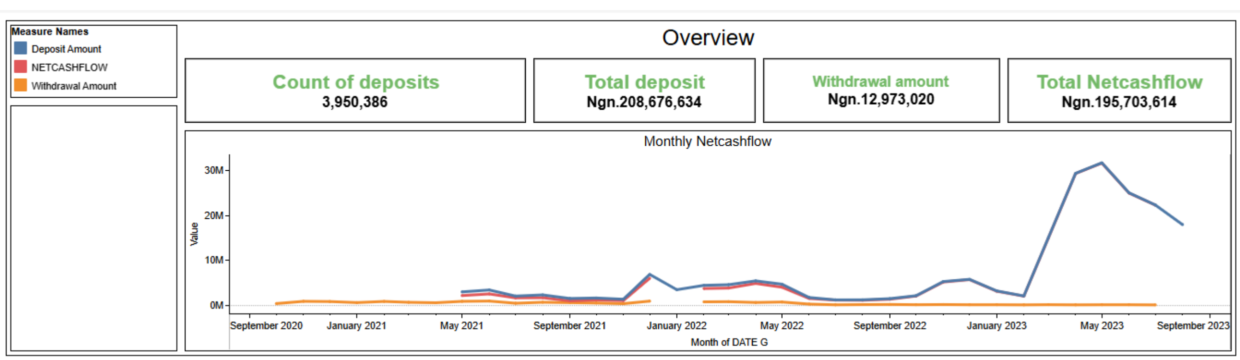
Betting



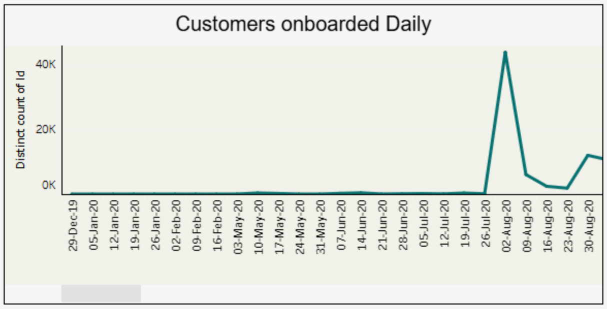

Win / Loss Analysis
Bet,won and lost amounts over time
Bet amount
Lost Amount
Won Amount
Value
10M
5M
0M
November 2021
January 2022
March 2022
May 2022
July 2022
September 2022
November 2022
January 2023
March 2023
May 2023
July 2023
Month of Time
Custome
Month, Y
(All)
0.63
AVG.Win/Loss Ratio
77.30
AVG.Win Value
677.8
AVG.Count of plays
Customer Wins/ Losses

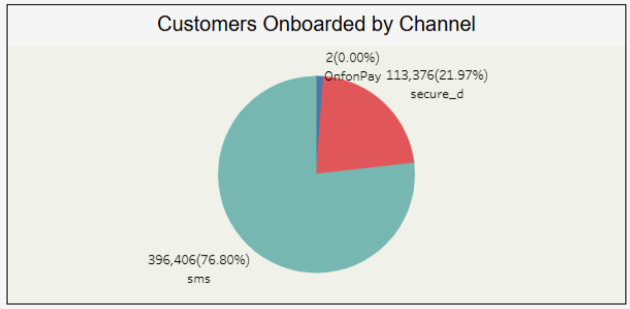

Filters and legend
Status
success
Status
success
start date
27/08/2
end date
25/12/202
Amount in transaction status per hour
Amount
400K
300K
200K
100K
0K
188,106
75,996
75,722
49,923
34,041
41,114
18,672
33,485
46,611
143,132
410,637
247,782
363,990
368,875
360,914
428,237
402,626
463,365
435,006
380,760
293,129
233,130
263,517
215,126
00 01 02 03 04 05 06 07 08 09 10 11 12 13 14 15 16 17 18 19 20 21 22 23


Filters And Legends
Narrative
(All)
Platform
(All)
Description (group)
Success
Description (group)
Success
start date
Amount transacted per hour
Amount
800K
700K
600K
500K
400K
300K
200K
100K
0K
301,744
116,322
72,761
41,141
29,372
135,234
202,089
264,924
632,465
379,854
699,274
656,680
517,290
557,833
758,184
717,293
529,095
561,059
470,377
509,965
388,466
00 01 02 03 04 05 06 07 08 09 10 11 12 13 14 15 16 17 18 19 20 21 22 23


Measure Names
Deposit Amount
NETCASHFLOW
Withdrawal Amount
Overview
Count of deposits
3,950,386
Total deposit
Ngn.208,676,634
Withdrawal amount
Ngn.12,973,020
Total Netcashflow
Ngn.195,703,614
Monthly Netcashflow
Value
30M
20M
10M
0M
September 2020
January 2021
May 2021
September 2021
January 2022
May 2022
September 2022
January 2023
May 2023
September 2023
Month of DATE G

516,142	174,304	33.92%	₦ 139,212
Count of Customers	Converted Customers	Conversion rate	Refferal Income





FILTERS

Start Date 25/06/2019

End Date 23/08/2023

Registrati... (All)

Measure Names

Converted Customers

Joining Year

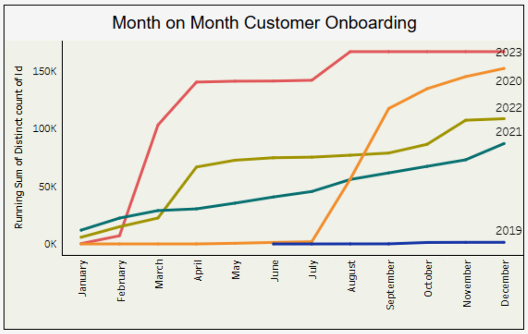
2019

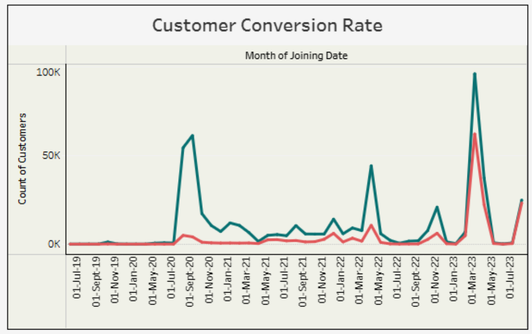
2020

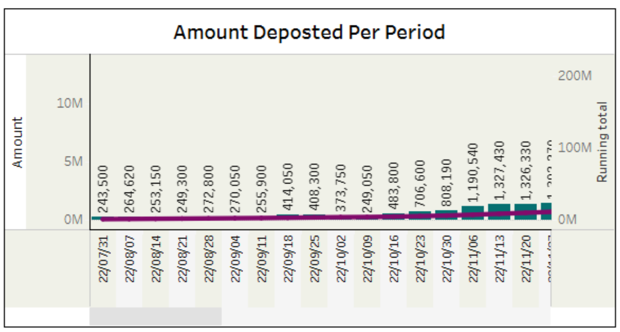
2021

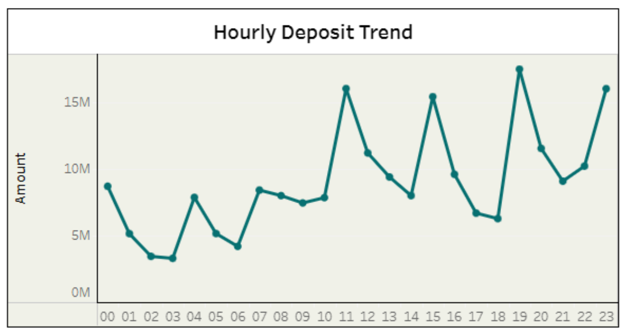
2022

2023



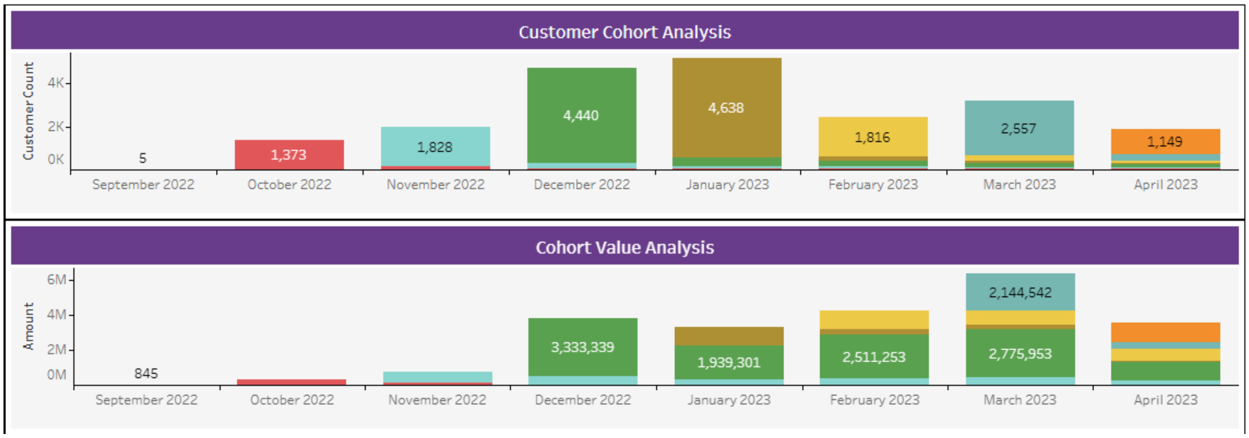






Customer Retention Analysis

Customer D..	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
28/09/2022	100.00%	50.00%					50.00%	50.00%	50.00%	50.00%	50.00%						
29/09/2022	100.00%	33.33%	66.67%	66.67%	66.67%	100.00%	66.67%	100.00%	100.00%	66.67%	66.67%	33.33%	33.33%	66.67%	33.33%	33.33%	33.33
03/10/2022	100.00%																
04/10/2022	100.00%		2.38%	14.29%	2.38%											2.38%	
05/10/2022	100.00%	7.84%	1.96%	1.96%				1.96%				1.96%					
06/10/2022	100.00%	3.39%	3.39%	1.69%						1.69%	1.69%	1.69%	1.69%		3.39%		
07/10/2022	100.00%	2.12%	2.65%	0.53%	1.59%				0.53%	1.06%	1.06%			2.12%	4.23%	3.70%	3.17
08/10/2022	100.00%	6.29%	1.71%	1.71%	1.14%	0.57%	1.14%	1.14%	1.71%	1.14%	1.14%	0.57%	0.57%	1.14%	4.00%	1.71%	0.57
09/10/2022	100.00%	5.05%	4.55%	3.03%	3.03%	2.02%	2.53%	2.53%	0.51%	1.01%	1.52%	1.01%	1.01%	0.51%	2.02%	1.01%	2.02
10/10/2022	100.00%	2.38%	2.38%	3.57%	1.19%	2.38%	2.38%	1.19%	1.19%		2.38%	2.38%	2.38%	1.19%	1.19%	2.38%	4.76
11/10/2022	100.00%	6.67%	6.67%	1.90%	4.76%	1.90%	0.95%	2.86%	2.86%	1.90%	2.86%	2.86%	1.90%		1.90%	2.86%	1.90
12/10/2022	100.00%	8.57%	2.86%	5.71%	11.43%			2.86%		2.86%	2.86%		2.86%	2.86%	5.71%		
13/10/2022	100.00%	3.23%	3.23%	9.68%	3.23%	3.23%	3.23%	3.23%	6.45%	3.23%					6.45%	3.23%	
14/10/2022	100.00%	5.56%	11.11%	5.56%			5.56%	5.56%	5.56%	11.11%	5.56%	5.56%	5.56%			16.67%	



In addition to the above, I created a machine learning model to predict customer acquisition and retention rates from difrerent business decisions like marketing campaigns, grace periods, betting bonusses and more.

All these efforts helped the business owners to make proper financial decisions to achieve higher profits by maintaining high positive net cashflows and also improve customer retention by 80 percent, and increased customer acquisition among others.

PROJECT 3

INDUSTRY:
BUSINESS INTELLIGENCE ANALYST

At Infotrace Analytics, I worked as a Business Intelligence Analyst on a key project for Digital Sacco, providing data-driven insights that enhanced decision-making, improved member services, and optimized financial performance. Using Power BI, I conducted the following analyses:

Loan Portfolio Analysis: Assessed loan performance, highlighting default trends and potential risks, which helped Digital Sacco strengthen its risk management and maintain a healthy loan portfolio.

Customer Segmentation: Grouped members based on demographics and financial behavior, enabling targeted marketing efforts and tailored loan products, improving customer engagement and satisfaction.

Churn Analysis: Analyzed member attrition patterns to inform retention strategies, reducing customer churn and fostering long-term relationships.

Cashflow Analysis: Evaluated cash inflows and outflows, providing insights that improved liquidity management and ensured that sufficient funds were available for loan disbursements.

Customer Conversion Rates from Campaigns: Tracked and measured the effectiveness of marketing campaigns, allowing the Sacco to enhance campaign strategies, boost conversion rates, and improve overall ROI.

RFM (Recency, Frequency, Monetary) Analysis: Helped Digital Sacco better understand member behavior and identify high-value members, enabling more effective engagement and retention strategies.

Collections & Non-Performing Loan (NPL) Analysis: Monitored collection patterns and NPL trends, identifying causes and providing actionable recommendations to improve collections and reduce losses.

These analyses helped Digital Sacco improve operational efficiency, manage risk, and better serve its members, contributing to the organization's mission of promoting financial independence through affordable credit.