DAB 303

Customer segmentation analysis

(In-Class Group Project - Final Report)

SUBMITTED TO: -

Samanta Rana

SUBMITTED BY: -

Manish Kataria (W0865937)

Sahil (W0861930)

Janvi (W0861408)

Parbhjot Kaur (W0859954)

Contents

Introduction

1. **What?**
   Customer segmentation in loan approvals categorizes applicants based on their likelihood of given information, using machine learning models.
2. **Why?**
   To identify high-risk and low-risk applicants, allowing lenders to make informed decisions, reduce financial losses, and offer personalized loan terms.
3. **Who?**
   Banks, financial institutions, credit unions, and fintech platforms use segmentation to assess loan applicants and manage risk effectively.
4. **When?**
   Segmentation is applied when a customer submits a loan application, helping classify them into different risk groups before approval.
5. **Where?**
   This system is implemented within banking institutions, lending platforms, and financial technology services that process loan applications.

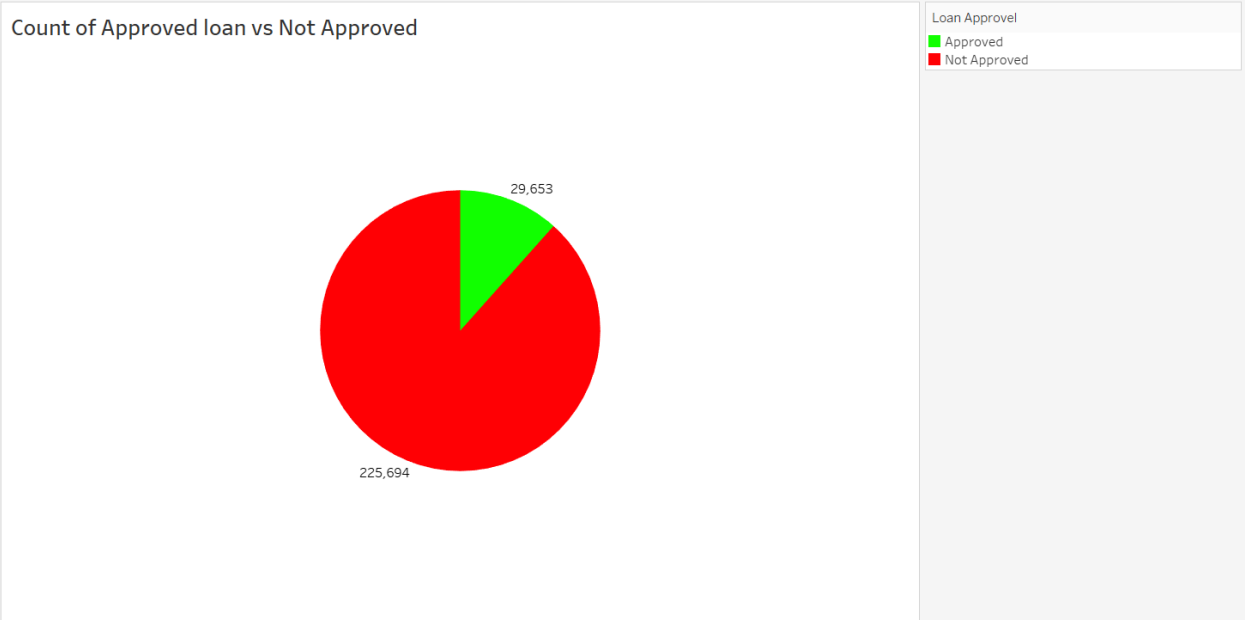
Aim

To preform customer segmentation analysis by develop an AI-powered Loan Approval System that automates and optimizes the decision-making process, ensuring accurate, fair, and efficient loan approvals for customers.

Exploratory Data Analysis (EDA)

- **Dataset:** Loan Default Dataset [1]
- **Key Features:**
  - In this dataset we have total **255347 entities** in **18 variables**
  - **Demographic Data:** Age, Education, Marital Status, Employment Type
  - **Financial Data:** Income, Credit Score, DTI Ratio, Has Mortgage
  - **Loan Details:** Loan Amount, Interest Rate, Loan Term, Loan Purpose
  - **Outcome Variable:** Default (0/1) (Not approved/approved).

- **Data illustration with help of Visualizations**



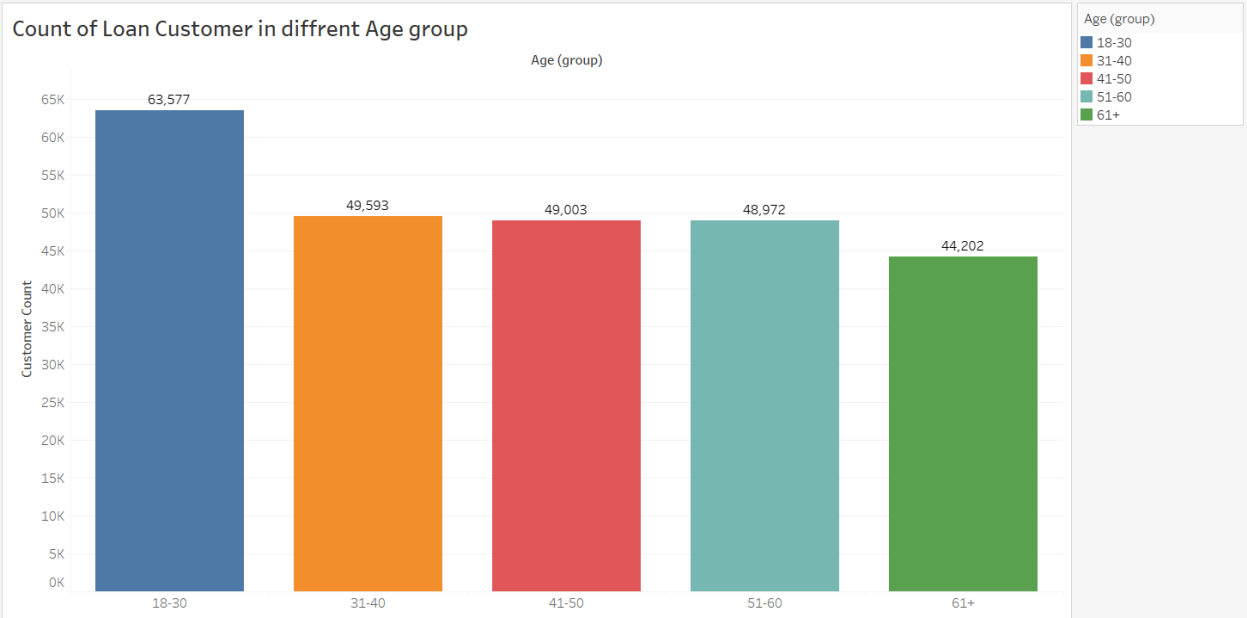
Here, we have a pie chart that shows the number of approved and not approved loan applications.

Key Elements:

- **Pie Chart:** This chart helps us visualize the proportions of total loan applications.
- **Two Slices:**
  - **Approved Loans (Green):** This slice represents the loans that got approved.
  - **Not Approved Loans (Red):** This slice shows the loans that were not approved.
- **Numbers:** Each slice has a number indicating the count of applications:
  - **29,653** loans were approved.
  - **225,694** loans were not approved.
- **Legend:** On the right, we see a legend that explains the colors:
  - **Green** stands for "Approved" loans.
  - **Red** represents "Not Approved" loans.
- **Title:** The title, *"Count of Approved Loans vs Not Approved"*, clearly tells us what the chart is about.

Interpretation:

Now, we can see that the red slice (Not Approved) is much larger than the green slice (Approved). This means most loan applications did not get approved. Moreover, this visualization helps us quickly understand the approval rate in their loan process, i.e., a small percentage of applications succeed while most are rejected.



Here, we have a bar chart that shows the number of loan customers in different age groups.

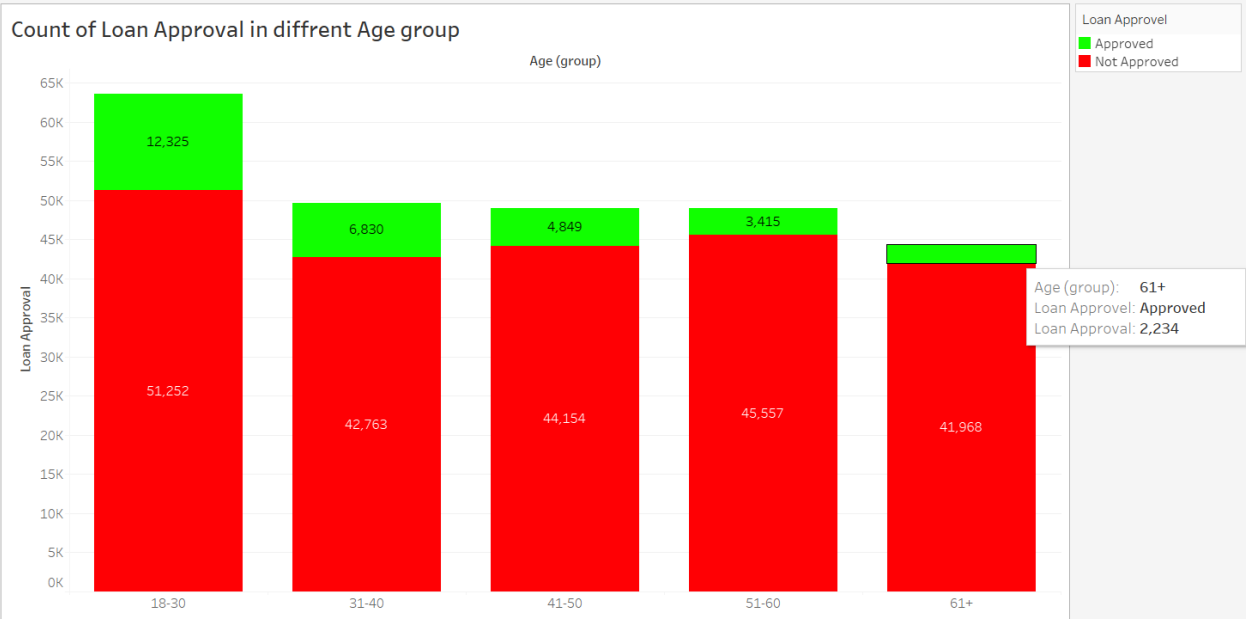
Key Elements:

- **Bar Chart:** The chart displays bars where each bar's height represents the number of loan customers in a particular age group.
- **Age Groups:** The x-axis lists the age groups:
  - 18-30
  - 31-40
  - 41-50
  - 51-60
  - 61+
- **Customer Count:** The y-axis shows the number of customers in each age group.

- **Bars:** Each age group has a colored bar, with the height showing the number of customers.
- **Numbers:** The exact customer count for each age group is listed above each bar.
- **Title:** The title, *"Count of Loan Customers in Different Age Groups,"* tells us what the chart is about.
- **Legend:** A legend on the right side explains the color coding for each age group.

Interpretation:

Now, we can see that the 18-30 age group has the highest number of loan customers (63,577). After this group, the number of loan customers decreases with each age group, with the 61+ group having the fewest customers (44,202).



ere, we have a stacked bar chart that shows the count of loan approvals and non-approvals across different age groups. Let's break it down:

Key Elements:

- **Stacked Bars:** Each bar represents an age group (18-30, 31-40, 41-50, 51-60, 61+). The bars are stacked, with the green part showing approved loans and the red part showing not approved loans.
- **Loan Approval Count:** The numbers inside each bar show how many loans were approved or not approved in each age group.

- **Age Groups (X-Axis):** The horizontal axis (x-axis) lists the different age groups.
- **Loan Approval Count (Y-Axis):** The vertical axis (y-axis) shows the count of loan approvals and non-approvals.
- **Title:** The title of the chart is *"Count of Loan Approval in Different Age Groups,"* clearly stating what the chart is about.
- **Legend:** A legend on the right explains that green represents "Approved" loans and red represents "Not Approved" loans.

Interpretation:

Now, we can easily see how loan approvals and rejections vary across different age groups. The chart gives a clear visual comparison of approval rates by age.

Here, we calculate the loan approval rate for each age group using the formula:

Approval Rate = (Number of Approved Loans / Total Number of Loans) * 100

1. 18-30 Age Group:

- **Approved Loans:** 12,325
- **Not Approved Loans:** 51,252
- **Total Loans:** 12,325 + 51,252 = 63,577
- **Approval Rate:** (12,325 / 63,577) * 100 = **19.4%**

2. 31-40 Age Group:

- **Approved Loans:** 6,830
- **Not Approved Loans:** 42,763
- **Total Loans:** 6,830 + 42,763 = 49,593
- **Approval Rate:** (6,830 / 49,593) * 100 = **13.8%**

3. 41-50 Age Group:

- **Approved Loans:** 4,849
- **Not Approved Loans:** 44,154
- **Total Loans:** 4,849 + 44,154 = 49,003
- **Approval Rate:** (4,849 / 49,003) * 100 = **9.9%**

4. 51-60 Age Group:

- **Approved Loans:** 3,415
- **Not Approved Loans:** 45,557
- **Total Loans:** 3,415 + 45,557 = 48,972
- **Approval Rate:** (3,415 / 48,972) * 100 = **7.0%**

5. 61+ Age Group:

- **Approved Loans:** 2,234
- **Not Approved Loans:** 41,968
- **Total Loans:** 2,234 + 41,968 = 44,202
- **Approval Rate:** (2,234 / 44,202) * 100 = **5.1%**

Summary of Approval Rates:

- **18-30:** 19.4%
- **31-40:** 13.8%
- **41-50:** 9.9%
- **51-60:** 7.0%
- **61+:** 5.1%

These calculations show that the approval rate decreases as the age group increases.

Count of Customers in diffrent Class

Has Dependents	Has Mortgage	Loan Approvel	Employment Type: Full-time	Part-time	Self-employed	Unemployed	Grand Total
Yes	Yes	Approved	1,256	1,653	1,524	1,822	6,255
		Not Approved	14,570	14,451	14,473	14,128	57,622
	No	Approved	1,469	1,855	1,802	2,035	7,161
		Not Approved	14,423	14,110	14,156	14,015	56,704
	Total		31,718	32,069	31,955	32,000	127,742
No	Yes	Approved	1,555	1,931	1,880	2,271	7,637
		Not Approved	14,342	14,051	14,115	13,655	56,163
	No	Approved	1,744	2,238	2,096	2,522	8,600
		Not Approved	14,297	13,872	13,660	13,376	55,205
	Total		31,938	32,092	31,751	31,824	127,605
Grand Total			63,656	64,161	63,706	63,824	255,347

Loan Approval
1,256 14,570

Here, we have a heat map table that shows the number of loan customers based on different factors.

Key Elements:

- **Heat Map Table:** The main element is a table where the cells are colored according to the numbers they represent.
- **Categories (Rows):** The rows are divided into different categories:
  - **Has Dependents:** Yes or No
  - **Has Mortgage:** Yes or No
  - **Loan Approval:** Approved or Not Approved
  - **Total:** The sum of Approved and Not Approved for each combination.
- **Categories (Columns):** The columns represent different employment types:
  - Full-time
  - Part-time
  - Self-employed
  - Unemployed
  - **Grand Total:** The sum of all employment types for each row category.
- **Numerical Values:** Each cell shows the count of loan customers for a specific combination of categories.
- **Color Coding:** The cells are colored using a gradient scale:
  - **Green:** Represents the lowest values.
  - **Red:** Represents the highest values.
  - **Gradient:** The colors gradually change from green to red, indicating how large or small the values are.
- **Legend:** A legend in the top right corner explains the value range and the corresponding colors.
- **Interpretation:**
  Now, the heat map makes it easy to spot patterns and trends in the data. Here are some key takeaways:

- **Dominance of "Not Approved":** The "Not Approved" rows generally have higher customer counts (shown in red), meaning there are many more loan rejections than approvals.
- **Large Overall Customer Counts:** The "Grand Total" rows (both for total customers and the overall grand total) show high numbers (in red), indicating a large dataset overall.
- **Employment Type Impact:** Even though "Not Approved" dominates, there are some differences in customer counts for each employment type. For example, the "Unemployed" category tends to have slightly fewer customers.
- **Dependency and Mortgage Influence:** The effect of having dependents or a mortgage on loan approval is not very strong but can be seen when closely looking at the numbers. The heat map gives a quick summary of the overall data distribution.

Objective

- To analyze key factors influencing loan approval.
- To build a machine learning model that predicts loan approval.
- To improve loan approval accuracy and minimize risks.
- To ensure fairness and transparency in decision-making.

Target Audience

- **Banks and Financial Institutions**: To improve loan decision-making.
- **Customers**: To receive quick and fair loan approvals.
- **Fintech Companies**: To integrate AI-powered decision systems.

Structure of the Model

1. **Data Preprocessing**
    - Handling missing values, encoding categorical data, and feature scaling.

2. **Feature Selection**
    - Selecting important variables that influence loan approval.
3. **Machine Learning Models**
    - **Logistic Regression** (Baseline model)
    - **Random Forest Classifier** (High accuracy & interpretability)
    - **Gradient Boosting** (Handles complex relationships)
4. **Evaluation Metrics**
    - Accuracy, Precision, AUC-ROC Curve
5. **Deployment**
    - A web-based loan approval calculator for user interaction.

Model and Calculator

```
import numpy as np
import pandas as pd
import joblib
from sklearn.preprocessing import StandardScaler, OneHotEncoder
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassifier
from sklearn.impute import SimpleImputer
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score

# Load dataset
df = pd.read_csv('Loan_default.csv')

# Impute missing numerical values with the median
num_imputer = SimpleImputer(strategy="median")
df[['Age', 'Income', 'LoanAmount', 'CreditScore', 'MonthsEmployed', 'NumCreditLines',
    'InterestRate', 'LoanTerm', 'DTIRatio']] = num_imputer.fit_transform(df[['Age', 'Income', 'LoanAmount', 'CreditScore',
    'MonthsEmployed', 'NumCreditLines', 'InterestRate', 'LoanTerm', 'DTIRatio']])

# Fill missing categorical values with "Unknown"
df[['Education', 'EmploymentType', 'MaritalStatus', 'LoanPurpose']] = df[['Education', 'EmploymentType',
                                                                          'MaritalStatus', 'LoanPurpose']].fillna("Unknown")

# Convert binary categorical features
df['HasMortgage'] = df['HasMortgage'].map({'Yes': 1, 'No': 0})
df['HasDependents'] = df['HasDependents'].map({'Yes': 1, 'No': 0})
df['HasCoSigner'] = df['HasCoSigner'].map({'Yes': 1, 'No': 0})

# Feature Engineering: Add Loan-to-Income Ratio
df["Loan_to_Income_Ratio"] = df["LoanAmount"] / df["Income"]
df["Loan_to_Income_Ratio"] = df["Loan_to_Income_Ratio"].fillna(df["Loan_to_Income_Ratio"].median())  # Fix inplace warning

# Define features and target variable
X = df[['Age', 'Income', 'LoanAmount', 'CreditScore', 'MonthsEmployed', 'NumCreditLines', 'InterestRate', 'LoanTerm', 'DTIRatio',
        'Education', 'EmploymentType', 'MaritalStatus', 'HasMortgage', 'HasDependents', 'LoanPurpose', 'HasCoSigner', 'Loan_to_Income_Ratio']]
y = df['Default']

# Encode categorical variables
column_transformer = ColumnTransformer([
    ('encoder', OneHotEncoder(handle_unknown='ignore'), ['Education', 'EmploymentType', 'MaritalStatus', 'LoanPurpose'])
], remainder='passthrough')
```

```
# Split dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Standardize numerical features
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(column_transformer.fit_transform(X_train))
X_test_scaled = scaler.transform(column_transformer.transform(X_test))

# Check for NaNs in transformed data
if np.isnan(X_train_scaled).sum() > 0 or np.isnan(X_test_scaled).sum() > 0:
    print("Warning: NaN values still exist in processed data!")
    # Drop rows with NaNs before model training
    X_train_scaled = pd.DataFrame(X_train_scaled).dropna().to_numpy()
    y_train = y_train.loc[X_train.index]  # Align y_train

# Train Models
logreg = LogisticRegression(max_iter=1000)
rf = RandomForestClassifier(n_estimators=200, max_depth=10, random_state=42)
gb = GradientBoostingClassifier(n_estimators=200, learning_rate=0.05, max_depth=5, random_state=42)

logreg.fit(X_train_scaled, y_train)
rf.fit(X_train_scaled, y_train)
gb.fit(X_train_scaled, y_train)

# Save models
joblib.dump(logreg, "logistic_regression.pkl")
joblib.dump(rf, "random_forest.pkl")
joblib.dump(gb, "gradient_boosting.pkl")
joblib.dump(scaler, "scaler.pkl")
joblib.dump(column_transformer, "column_transformer.pkl")

# Evaluate Models
y_pred_logreg = logreg.predict(X_test_scaled)
y_pred_rf = rf.predict(X_test_scaled)
y_pred_gb = gb.predict(X_test_scaled)

print(f"Logistic Regression Accuracy: {accuracy_score(y_test, y_pred_logreg):.4f}")
print(f"Random Forest Accuracy: {accuracy_score(y_test, y_pred_rf):.4f}")
print(f"Gradient Boosting Accuracy: {accuracy_score(y_test, y_pred_gb):.4f}")
```

```
Logistic Regression Accuracy: 0.8880
Random Forest Accuracy: 0.8865
Gradient Boosting Accuracy: 0.8876
```

- **Importing Libraries:**
  We start by importing the necessary libraries. These include **Numpy** and **Pandas** for handling and manipulating data. We also import tools from **Scikit-learn** for machine learning, like models and preprocessing functions. Additionally, we use **Joblib** to save and load models, and **SimpleImputer** to handle missing data. We also import **StandardScaler** and **OneHotEncoder** to preprocess our data.

- **Load Dataset:**
  We now load the dataset, **Loan_default.csv**, into a variable called df. This dataset contains information about loan applicants and whether they defaulted on their loans.

- **Handling Missing Data:**
  Here, we deal with missing values:

  Numerical Data: We use SimpleImputer to replace missing values in columns like Age, Income, and Loan Amount with their **median** values.

  Categorical Data: For missing categorical values, such as Education and EmploymentType, we fill them with the label **"Unknown"**.

- **Convert Binary Categorical Features:**
  Next, we convert binary **Yes/No** answers into **1/0**. For example, the columns **HasMortgage**, **HasDependents**, and **HasCoSigner** are changed so that **Yes** becomes **1** and **No** becomes **0**.

- **Feature Engineering:**
  We create a new feature called **Loan-to-Income Ratio** by dividing the **LoanAmount** by the **Income**. If there are any missing values in this new feature, we replace them with the **median** value of the ratio.

- **Feature and Target Variable Definition:**
  Now, we define:

  Features (X): These are all the columns except **Default**.

  Target (y): This is the **Default** column, which shows if a loan was defaulted or not.

- **Encoding Categorical Features:**
  We use **OneHotEncoder** to convert categorical features like Education and EmploymentType into a format that can be used by machine learning models. The ColumnTransformer helps us apply this encoding only to the relevant columns.

- **Data Split:**
  We then split our dataset into two parts:

  Training Data: 80% of the data is used to train the models.

  Testing Data: 20% of the data is held back to test the models.

- **Standardization:**
  We apply **StandardScaler** to our data to scale the numerical features, making sure they have a mean of 0 and a standard deviation of 1. This helps our models perform better. The scaling is done separately for the training and testing data.

- **Check for NaN Values After Transformation:**
  After scaling, we check if any NaN values remain in the transformed data. If we find any, we print a warning and remove those rows to avoid issues with model training.

- **Model Training:**
  We train three different models:

  Logistic Regression

  Random Forest

  Gradient Boosting

  Each model is trained using the scaled training data.

- **Save Trained Models:**
  After training the models, we save them (along with the preprocessing steps like scaling and encoding) using **Joblib**. This allows us to use them later without retraining.

- **Model Evaluation:**
  Finally, we use the trained models to make predictions on the testing data. We evaluate their performance by checking their accuracy, which tells us how well the models predicted loan defaults.

Model Performance Interpretation:

- **Logistic Regression Accuracy: 0.8880**
  - **Interpretation**: Here, the Logistic Regression model accurately predicted loan defaults 88.80% of the time. This means that, out of all the test cases, it correctly determined whether a customer would default on the loan in 88.8% of cases.
  - **Context**: Logistic Regression is a simpler, linear model used for classification tasks. It works well when the relationship between the features and the target variable is linear, which is why it performs well here.

- **Random Forest Accuracy: 0.8865**
    - **Interpretation**: The Random Forest model made correct predictions 88.65% of the time. This means 88.65% of the model's predictions were accurate.
    - **Context**: Random Forest is an ensemble model that combines multiple decision trees. It performs well with complex, non-linear data and is more robust than a single decision tree, especially when the data has diverse patterns like in this case.
- **Gradient Boosting Accuracy: 0.8876**
    - **Interpretation**: The Gradient Boosting model predicted loan defaults correctly 88.76% of the time. This means it was accurate in 88.76% of the test cases.
    - **Context**: Gradient Boosting is also an ensemble model, but it builds trees one after another, with each tree aiming to correct the errors of the previous one. It tends to do better when the relationships between features and the target are complex.
- **Overall Observations:**
    - Now, all three models (Logistic Regression, Random Forest, and Gradient Boosting) show very similar accuracy, ranging from **88.65% to 88.80%**.
    - This means the models are all performing well, and the small differences in accuracy suggest that they are almost equally effective.
    - Moreover, the slight differences in performance may be due to how the models handle data or the specific settings (hyperparameters) used during training.

```
from sklearn.metrics import roc_curve, auc
import matplotlib.pyplot as plt

# Calculate ROC curve and AUC for each model
fpr_logreg, tpr_logreg, _ = roc_curve(predictions_df['Actual'], predictions_df['Logistic Regression'])
fpr_rf, tpr_rf, _ = roc_curve(predictions_df['Actual'], predictions_df['Random Forest'])
fpr_gb, tpr_gb, _ = roc_curve(predictions_df['Actual'], predictions_df['Gradient Boosting'])

roc_auc_logreg = auc(fpr_logreg, tpr_logreg)
roc_auc_rf = auc(fpr_rf, tpr_rf)
roc_auc_gb = auc(fpr_gb, tpr_gb)

# Plot ROC Curve for all models
plt.figure(figsize=(10, 6))
plt.plot(fpr_logreg, tpr_logreg, color='blue', lw=2, label=f'Logistic Regression (AUC = {roc_auc_logreg:.2f})')
plt.plot(fpr_rf, tpr_rf, color='green', lw=2, label=f'Random Forest (AUC = {roc_auc_rf:.2f})')
plt.plot(fpr_gb, tpr_gb, color='purple', lw=2, label=f'Gradient Boosting (AUC = {roc_auc_gb:.2f})')

# Plot the diagonal line for random chance (No skill)
plt.plot([0, 1], [0, 1], color='gray', linestyle='--')

# Set plot labels and title
plt.title('Receiver Operating Characteristic (ROC) Curve')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.legend(loc='lower right')

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

ROC Curve and AUC Calculation and Plotting

Here, we calculate and plot the **Receiver Operating Characteristic (ROC) curve** for three models: **Logistic Regression**, **Random Forest**, and **Gradient Boosting**. We also compute the **Area Under the Curve (AUC)** for each model, which shows how well the model distinguishes between loan defaults and non-defaults.

1. Calculate the ROC Curve

- We calculate the **ROC curve** using the roc_curve() function. This function needs two inputs:
    - **Actual values**: The true labels (predictions_df['Actual']).
    - **Predicted probabilities**: The probabilities predicted by each model.

The fpr (False Positive Rate) and tpr (True Positive Rate) are computed for each model, which we use to plot the ROC curve.

2. Calculate the AUC (Area Under the Curve)

- We calculate the **AUC** using the auc() function. The AUC shows how well the model separates the two classes (default vs. non-default).
    - An **AUC of 0.5** means the model has no ability to distinguish between the classes (like random guessing).
    - An **AUC of 1** means the model can perfectly distinguish between the classes.

3. Plot the ROC Curve

- We plot the **ROC curve** for each model in different colors:
    - **Blue** for Logistic Regression.
    - **Green** for Random Forest.
    - **Purple** for Gradient Boosting.
- We also display the **AUC** for each model in the legend.

4. Plot the Diagonal Line for Random Chance

- The **gray dashed line** represents the ROC curve for a random classifier (i.e., a model that guesses randomly). The AUC of this line is 0.5, which is the baseline for random guessing. We want our models' ROC curves to be above this line.

5. Set Labels and Title

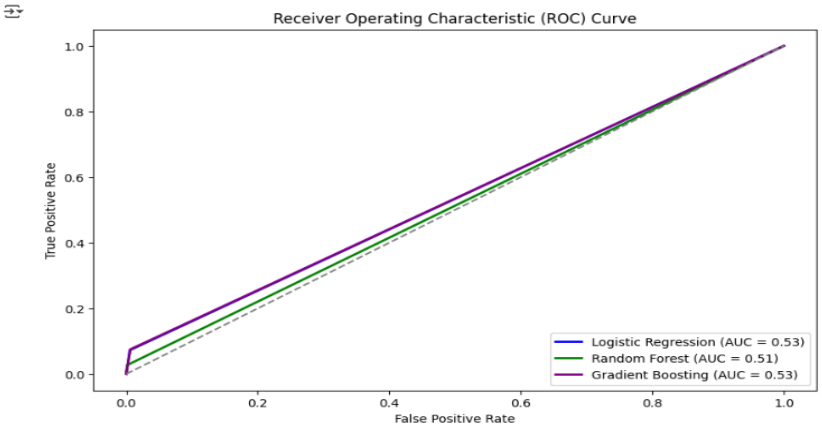
- We set the **title** of the plot as "Receiver Operating Characteristic (ROC) Curve."
- The **x-axis** represents the **False Positive Rate (FPR)**, which shows how many actual negatives are incorrectly predicted as positives.
- The **y-axis** represents the **True Positive Rate (TPR)**, which shows how many actual positives are correctly predicted.
- The **legend** is placed in the lower-right corner to label each model with its AUC.

6. Show the Plot

- This line displays the plot on the screen.

What Does the ROC Curve and AUC Mean?

- **ROC Curve**: This curve helps us assess how well the model performs by comparing the True Positive Rate (sensitivity) with the False Positive Rate (1 - specificity). The closer the curve is to the top-left corner, the better the model is at distinguishing between the two classes (default/no-default).
- **AUC (Area Under the Curve)**: A higher AUC indicates better model performance:
    - **AUC near 1**: Excellent model performance, meaning it can distinguish between positive and negative cases very well.
    - **AUC of 0.5**: The model performs like random guessing, with no predictive power.
    - **AUC near 0**: The model is worse than random guessing, incorrectly classifying the instances.



Explanation of the Graph:

- **Title**: "Receiver Operating Characteristic (ROC) Curve" – This tells us that we're looking at a graph that shows how well the classification models are performing.
- **Axes**:
    - **Y-axis (True Positive Rate - TPR)**: This measures how many of the actual positive cases the model correctly identifies. A higher TPR means the model is better at finding positives.
    - **X-axis (False Positive Rate - FPR)**: This measures how many actual negative cases the model incorrectly labels as positive. A lower FPR means fewer mistakes in identifying negatives.
- **The Diagonal Line (Dashed Gray)**: This represents random guessing. If a model's curve is above this line, it means it's doing better than guessing randomly.
- **Three Lines**:
    - **Blue**: Logistic Regression
    - **Green**: Random Forest
    - **Purple**: Gradient Boosting
    - These lines show how well each model performs at different thresholds for classification.
- **AUC (Area Under the Curve)**: Each model has an AUC value. AUC tells us how well the model distinguishes between positive and negative cases. An AUC of 1.0 is perfect, and 0.5 is like random guessing.

Interpretation:

- **Models are Learning**: All three models (Logistic Regression, Random Forest, and Gradient Boosting) have AUC values near 1, which is a good sign. It shows that these models are learning patterns from the data and doing better than random guessing.
- **Comparable Performance**: The AUC values of all three models are close to each other (around 0.51-0.53). This means they are performing similarly on this dataset, so we have a few models that are all doing equally well.
- **Visual Comparison**: The ROC curve makes it easy to compare how the models are performing. We can quickly see that Gradient Boosting and Logistic Regression are

performing just slightly better than Random Forest. This visual comparison helps us decide which model to choose.

```python
import numpy as np
import pandas as pd
import joblib
from sklearn.preprocessing import StandardScaler, OneHotEncoder
from sklearn.compose import ColumnTransformer

# Load trained models and preprocessing objects
logreg = joblib.load("logistic_regression.pkl")
rf = joblib.load("random_forest.pkl")
gb = joblib.load("gradient_boosting.pkl")
scaler = joblib.load("scaler.pkl")
column_transformer = joblib.load("column_transformer.pkl")

# Function to validate user inputs
def get_valid_input(prompt, dtype, valid_range=None, options=None):
    while True:
        try:
            value = input(prompt).strip()

            # Convert input to the required type
            if dtype == int:
                value = int(value)
            elif dtype == float:
                value = float(value)

            # Validate range for numerical inputs
            if valid_range and not (valid_range[0] <= value <= valid_range[1]):
                print(f"❌ Error: Value must be between {valid_range[0]} and {valid_range[1]}. Try again.")
                continue

            # Validate categorical options
            if options and value not in options:
                print(f"❌ Error: Please enter one of the following options: {', '.join(options)}. Try again.")
                continue

            return value

        except ValueError:
            print("❌ Error: Invalid format. Please enter a correct value.")
```

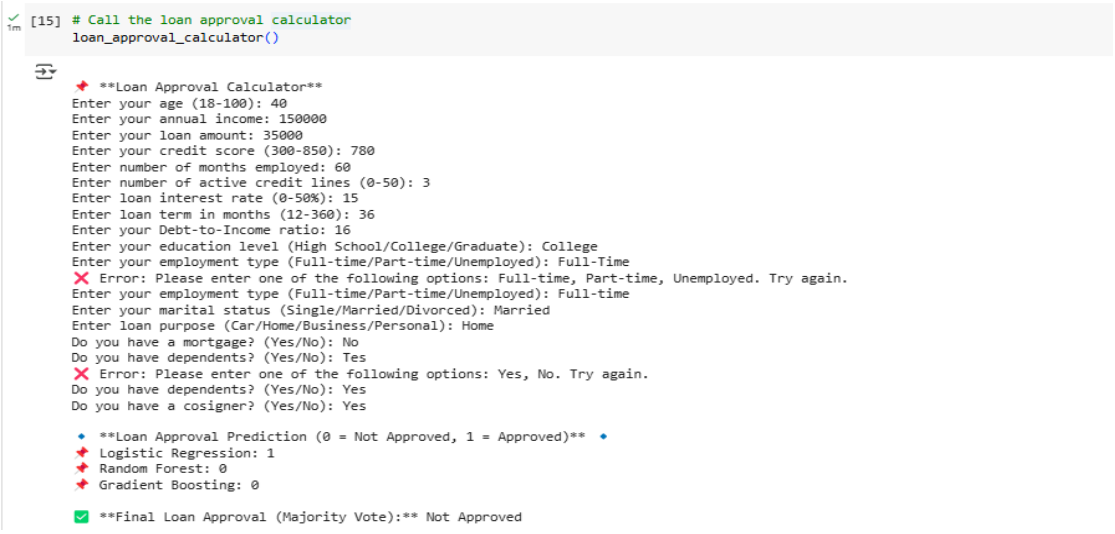
```python
def loan_approval_calculator():
    print("\n📌 **Loan Approval Calculator**")

    # Collect user input with validation
    age = get_valid_input("Enter your age (18-100): ", int, (18, 100))
    income = get_valid_input("Enter your annual income: ", float)
    loan_amount = get_valid_input("Enter your loan amount: ", float)
    credit_score = get_valid_input("Enter your credit score (300-850): ", int, (300, 850))
    months_employed = get_valid_input("Enter number of months employed: ", int, (0, 600))
    num_credit_lines = get_valid_input("Enter number of active credit lines (0-50): ", int, (0, 50))
    interest_rate = get_valid_input("Enter loan interest rate (0-50%): ", float, (0, 50))
    loan_term = get_valid_input("Enter loan term in months (12-360): ", int, (12, 360))
    dti_ratio = get_valid_input("Enter your Debt-to-Income ratio: ", float)

    # Validate categorical options
    education = get_valid_input("Enter your education level (High School/College/Graduate): ", str, options=["High School", "College", "Graduate"])
    employment_type = get_valid_input("Enter your employment type (Full-time/Part-time/Unemployed): ", str, options=["Full-time", "Part-time", "Unemployed"])
    marital_status = get_valid_input("Enter your marital status (Single/Married/Divorced): ", str, options=["Single", "Married", "Divorced"])
    loan_purpose = get_valid_input("Enter loan purpose (Car/Home/Business/Personal): ", str, options=["Car", "Home", "Business", "Personal"])

    # Validate Yes/No questions
    has_mortgage = get_valid_input("Do you have a mortgage? (Yes/No): ", str, options=["Yes", "No"])
    has_dependents = get_valid_input("Do you have dependents? (Yes/No): ", str, options=["Yes", "No"])
    has_cosigner = get_valid_input("Do you have a cosigner? (Yes/No): ", str, options=["Yes", "No"])

    # Convert Yes/No answers to binary values
    has_mortgage = 1 if has_mortgage == "Yes" else 0
    has_dependents = 1 if has_dependents == "Yes" else 0
    has_cosigner = 1 if has_cosigner == "Yes" else 0

    # Feature Engineering: Loan-to-Income Ratio
    loan_to_income_ratio = loan_amount / income if income > 0 else 0
```

```python
# Create DataFrame for input and include Loan_to_Income_Ratio
user_data = pd.DataFrame({
    'Age': [age],
    'Income': [income],
    'LoanAmount': [loan_amount],
    'CreditScore': [credit_score],
    'MonthsEmployed': [months_employed],
    'NumCreditLines': [num_credit_lines],
    'InterestRate': [interest_rate],
    'LoanTerm': [loan_term],
    'DTIRatio': [dti_ratio],
    'Education': [education],
    'EmploymentType': [employment_type],
    'MaritalStatus': [marital_status],
    'HasMortgage': [has_mortgage],
    'HasDependents': [has_dependents],
    'LoanPurpose': [loan_purpose],
    'HasCoSigner': [has_cosigner],
    'Loan_to_Income_Ratio': [loan_to_income_ratio]  # Add new feature
})

# Transform categorical features and scale numeric features
try:
    user_data_transformed = column_transformer.transform(user_data)
    user_data_scaled = scaler.transform(user_data_transformed)
except Exception as e:
    print(f"\n❌ Error in data transformation: {e}")
    return

# Make predictions using all models
logreg_prediction = logreg.predict(user_data_scaled)
rf_prediction = rf.predict(user_data_scaled)
gb_prediction = gb.predict(user_data_scaled)

# Display predictions for all models
print("\n🔹 **Loan Approval Prediction (0 = Not Approved, 1 = Approved)** 🔹")
print(f"📌 Logistic Regression: {logreg_prediction[0]}")
print(f"📌 Random Forest: {rf_prediction[0]}")
print(f"📌 Gradient Boosting: {gb_prediction[0]}")

# **Majority Vote Decision**: Loan is approved if at least 2 models predict approval
final_decision = (logreg_prediction[0] + rf_prediction[0] + gb_prediction[0]) >= 2
print(f"\n✅ **Final Loan Approval (Majority Vote):** {'Approved' if final_decision else 'Not Approved'}")
```

This **Loan Approval Calculator** uses three machine learning models (Logistic Regression, Random Forest, and Gradient Boosting) to predict if a loan application will be approved based on user input. Here's how we break it down:

- **Loading the Models and Preprocessing Tools**
  We first load the three trained models and two preprocessing tools (for scaling and transforming the input data). These tools were saved earlier after training.
- **Validating User Input**
  We have a function to make sure the user enters the correct information. For example, the user's input is checked to ensure it's a number, within a specific range, or matches a list of valid options. If the input is wrong, it will keep asking until we get valid data.
- **Collecting User Input**
  We ask the user to enter various details like **age**, **income**, **loan amount**, and **credit score**. These inputs are then validated using the validation function mentioned earlier.

- **Feature Engineering**
  Next, we calculate a new feature called the **Loan-to-Income Ratio** by dividing the loan amount by the user's income. This helps understand the proportion of income the user is requesting as a loan.
- **Preparing the Data**
  We create a table (DataFrame) with the user's information, including the calculated Loan-to-Income Ratio. This table is used for making predictions.
- **Transforming the Data**
  We then transform the data. Categorical data (like job type or marital status) is converted into numbers, and numerical data is scaled. This step makes sure the data is in the right format for the models, just like how the models were trained.
- **Making Predictions**
  Once the data is ready, we pass it through all three models to make predictions. Each model will either approve (1) or reject (0) the loan based on the input.
- **Showing Predictions**
  We display the predictions from each model so the user can see how each one voted. This gives the user an idea of the approval status from all three models.
- **Majority Vote Decision**
  The final decision is made based on a **majority vote**. If at least two out of the three models predict approval, the loan is approved. This helps make the decision more reliable by considering multiple models.
- **Final Loan Approval Output**
  We then display the final loan approval status to the user, based on the majority vote. If two or more models say yes, the loan is approved; otherwise, it is not.



Customer Testimonial

1. Amit K., UK – Experience with AI Lending at Kabbage by American Express

Testimonial: "Traditional banks rejected my loan application, but an AI-powered lender considered my income patterns and granted me a loan at reasonable interest. AI makes lending more inclusive!"

Source: Amit K.'s experience with Kabbage, an AI-powered small business loan platform, is referenced in multiple discussions around AI lending's inclusivity. Kabbage uses AI algorithms to assess borrower risk based on data like cash flow and income patterns, allowing them to approve loans faster and more fairly than traditional methods.**[2]**

2. JPMorgan Chase – AI-Driven Lending Decisions

Testimonial: "AI-driven credit scoring has allowed us to make fair and unbiased lending decisions, helping more people access credit. The automation has reduced approval time from weeks to minutes."

Source: JPMorgan Chase's Chief Risk Officer spoke about the bank's use of AI in improving credit scoring models, making lending decisions faster and fairer. AI helps reduce biases in traditional credit scoring, enabling more people to get loans.***[3]***

Benefits of the System

Faster Loan Processing – Reduces approval time from days to minutes.

Improved Accuracy – Data-driven decisions reduce manual errors.

Fair and Transparent – Eliminates human bias in loan approvals.

Risk Management – Identifies high-risk applicants, minimizing defaults.
Scalability – Easily integrates with banking systems.

Conclusion:

1. Segmentation Based on Features:

- We analyze loan applicants based on features like **Age**, **Income**, **Credit Score**, **Loan Amount**, and **Employment Type**.
- We can also group applicants into segments based on common characteristics, such as:
    - **Age Group Segmentation**: We already check how many applicants belong to different age groups. This could be extended by analyzing how approval rates change across age groups (e.g., younger applicants may have lower approval rates than older ones).
    - **Income Segmentation**: We could divide applicants into income brackets and see how income affects approval rates.
    - **Risk Segmentation**: Based on the predictions from our loan default models, we can categorize applicants into **low-risk** and **high-risk** groups.

2. Segmented Loan Approval Rates:

- We've calculated **approval rates** for different age groups. This is a type of segmentation based on demographics.
- We could extend this to other factors like education, marital status, or employment type to understand how these features impact loan approvals.

3. Predictive Segmentation:

- By using machine learning models to predict loan defaults, we're essentially dividing applicants into high-risk and low-risk categories.
- We can then segment applicants based on the likelihood of loan default and adjust our approval process accordingly.

Reference

[1] Loan Default Prediction Dataset

[2] https://www.americanexpress.com/en-us/business/blueprint/

[3] https://www.jpmorganchase.com/content/dam/jpmc/jpmorgan-chase-and-co/investor-relations/documents/annualreport-2023.pdf