TABLE OF CONTENTS

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

1.1 OVERVIEW

This project aims to predict credit card approvals using a dataset with various applicant features, including numerical (Age, Debt, Income) and categorical (Gender, Married, Ethnicity) attributes. The data undergoes preprocessing where missing values are handled, and features are standardized and encoded using StandardScaler and OneHotEncoder. A logistic regression model is built within a Pipeline to streamline preprocessing and model fitting. The dataset is split into training and testing sets to evaluate the model's performance. Additionally, a K-Nearest Neighbors (KNN) classifier is used for comparative analysis. A Tkinter-based graphical user interface (GUI) allows users to input applicant data and receive real-time credit card approval predictions, enhancing user interaction and demonstrating the model's practical application.The project also employs K-Means clustering to segment approved applicants based on income and credit score, providing insights for targeted marketing and risk assessment.

Overall, this project demonstrates the end-to-end process of data preprocessing, model training, evaluation, and user interaction, offering a comprehensive solution for predicting credit card approvals and analyzing applicant characteristics.

1.2 TECHNOLOGY

1.2.1. JUPYTER NOTEBOOK

The Jupyter notebook app is a server-client application that allows editing and running notebook documents via a web browser. The Jupyter Notebook App can be executed on a local desktop requiring no internet access (as described in this document) or can be installed on a remote server and accessed through the internet.

Figure 1.1 Jupyter

1.2.2. PANDAS LIBRARY

Pandas is an open-source Python Library providing high-performance data manipulation and analysis tool using its powerful data structures. The name Pandas isderived from the word Panel Data – an Econometrics from Multidimensional data.

Figure 1.2-Pandas

1.2.3. NUMPY LIBRARY

NumPy is becoming more popular and is being consumed in a variety of commercial systems. NumPy is one of the most powerful Python libraries because of its syntax, which is compact, powerful,and expressive together at the same time.

Figure 1.3 - Numpy

1.2.4. SKLEARN LIBRARAY

Sklearn or scikit-learn in Python is by far one of the most useful open-source libraries available that you can use for Machine Learning in Python. The scikit- learn library is an exhaustive collection of the most efficient tools for statistical modeling and Machine Learning. Some of these tools include regression, classification, dimensionality reduction, and clustering.

Figure 1.4 – Scikit learn

1.2.5.TKINTER LIBRARY

Tkinter is a Python binding to the Tk GUI toolkit. It is the standard Python interface to the Tk GUI toolkit, and is Python's de facto standard GUI. Python Tkinter is the fastest and easiest way to create GUI applications. Creating a GUI using Tkinter is an easy task.

Figure 1.5 – Tkinter

CHAPTER 2

CREDIT CARD APPROVAL

2.1 GENERAL PIPELINE

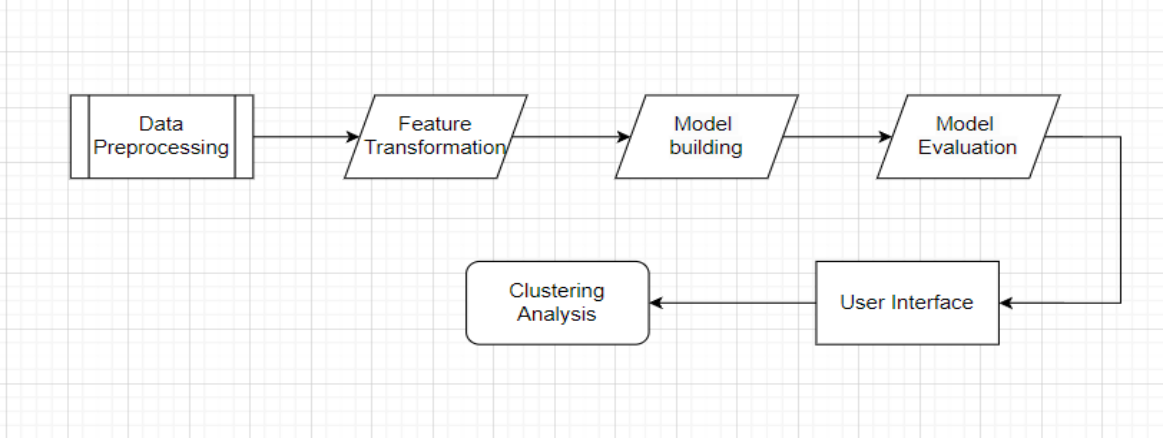

Figure 2.1 – Process Pipeline

In the Credit Card Approval Prediction Project, we preprocess the dataset by handling missing values and separating features into numerical and categorical types. We use StandardScaler for numerical features and OneHotEncoder for categorical features in a ColumnTransformer, combined in a Pipeline with a logistic regression model. The model is trained and evaluated by splitting the data into training and testing sets. A Tkinter GUI allows users to input data and receive credit card approval predictions. K-Means clustering analyzes approved applicants, offering insights for targeted marketing strategies. This approach ensures robust model performance and practical application through an intuitive interface.

2.1.1. DATA PREPROCESSING:

Data preprocessing, also known as data cleaning, is the process of identifying and correcting or removing corrupt, inaccurate, incomplete, or irrelevant data from a dataset. It's an important step in data mining, machine learning, and other data science tasks.

In our Credit Card Approval Prediction project, data preprocessing plays a crucial role in preparing the dataset for model training and evaluation. We start by loading the dataset from a CSV file, assuming it contains cleaned and formatted data. Next, we separate the dataset into features (X) and the target variable (y), where the target is the "Approved" column indicating credit card approval status. The

features are categorized into numerical and categorical types. Numerical features such as "Age," "Debt," and "Income" undergo standardization using StandardScaler to ensure all features have a similar scale, preventing any single feature from dominating the model training process due to its larger magnitude. Categorical features like "Gender," "Married," and "Industry" are encoded using OneHotEncoder to convert them into a numerical format suitable for machine learning algorithms. The ColumnTransformer combines these preprocessing steps into a unified pipeline, allowing seamless application to the entire dataset. This standardized and encoded dataset is then ready for training predictive models, ensuring that our machine learning algorithms can effectively learn from the data to predict credit card approval outcomes.

2.1.2. FEATURE TRANSFORMATION:

In our Credit Card Approval Prediction project, feature transformation plays a crucial role in preparing input data for machine learning models. We categorize our dataset into numerical and categorical features to apply appropriate transformations tailored to each type.

Numerical Feature Standardization

For numerical features such as "Age," "Debt," and "Income," we use the StandardScaler from scikit-learn. This technique standardizes numerical data by centering it around a mean of 0 and scaling it to have a standard deviation of 1. Standardization ensures that all numerical features contribute equally to the model training and avoids biases due to differing scales.

Categorical Feature Encoding

Categorical features like "Gender," "Married," and "Industry" are transformed using OneHotEncoder to convert them into a binary format suitable for machine learning algorithms. This encoding method creates new binary columns for each unique category, representing the presence or absence of a category in each data point.

ColumnTransformer Integration

The ColumnTransformer combines these feature transformations into a cohesive preprocessing pipeline that can be uniformly applied to both training and testing datasets. This standardized and transformed dataset ensures that our machine learning models can effectively learn patterns and make accurate predictions regarding credit card approval based on the provided features.

2.1.3. MODEL BUILDING:

In our Credit Card Approval Prediction project, we employ a logistic regression classifier to build our predictive model. Below are the key components and evaluation metrics used in our model building process:

Pipeline Construction

We construct a machine learning pipeline using scikit-learn's Pipeline module. This pipeline consists of a preprocessing step followed by a logistic regression classifier. The preprocessing step involves standardizing numerical features and encoding categorical features using a ColumnTransformer that combines StandardScaler and OneHotEncoder.

Training and Testing Split

We split our dataset into training and testing sets using a 80-20 ratio with train_test_split from scikit-learn. This ensures that our model is trained on a portion of the data and evaluated on unseen data to assess its generalization performance.

Model Training

The logistic regression classifier is trained on the training dataset using the fit method of our pipeline. This step involves learning the relationships between the features (after preprocessing) and the target variable ("Approved" or "Rejected").

Model Evaluation

After training, we evaluate the performance of our model using the following metrics:

Accuracy: The proportion of correct predictions (both true positives and true negatives) out of all predictions.

Precision: The ratio of true positive predictions to the total predicted positives, indicating the accuracy of positive predictions.

Recall (Sensitivity): The ratio of true positive predictions to the actual positives in the dataset, indicating the model's ability to identify positive cases.

F1 Score: The harmonic mean of precision and recall, providing a balanced measure of a model's accuracy.

These metrics give us insights into how well our logistic regression model performs in predicting credit

card approval based on the given applicant information. The model's accuracy and other performance metrics help us assess its effectiveness and guide potential improvements or adjustments to enhance prediction outcomes.

2.1.4. MODEL EVALUATION:

In our Credit Card Approval Prediction project, we evaluate our logistic regression model using key metrics: accuracy, precision, recall (sensitivity), and the F1 score. Accuracy measures overall correctness, precision assesses positive prediction quality, recall gauges positive instance capture, and the F1 score balances precision and recall. These metrics collectively reveal how well the model predicts credit card approval based on applicant data, guiding decision-making and highlighting areas for improvement.

2.1.5. CLUSTERING ANALYSIS:

In our project involving clustering analysis, we use K-means clustering to segment credit card applicants based on income and credit score. We assess the clustering using silhouette score to gauge the separation quality of clusters. This analysis helps identify distinct customer segments for targeted marketing strategies. The visualization of clusters and cluster centers aids in understanding customer behavior and guiding business decisions.

2.1.6. USER INTERFACE:

Our user interface for Credit Card approval prediction leverages Tkinter to create an interactive window where users can input applicant information such as gender, age, debt, marital status, and more. Upon clicking the "Predict" button, the system processes this data through a pre-trained logistic regression model to predict loan approval status, displaying the result directly.

2.2 APPLICATION OF CREDIT CARD APPROVAL SYSTEM

2.2.1. CHASE BANK

Chase Bank, part of JPMorgan Chase & Co., is one of the largest banks in the United States, offering a range of financial services including credit cards. Chase Bank employs credit card approval processes to assess applicants' creditworthiness and manage risk. Chase utilizes various criteria for credit card approval, such as credit score, income level, employment status, debt-to-income ratio, and payment history. They analyze these factors to determine an applicant's

likelihood of repayment and credit risk. Chase Bank's credit card approval process involves sophisticated algorithms and risk models to evaluate applications efficiently and accurately. By leveraging data analytics and machine learning, Chase aims to optimize its credit card approval workflows while maintaining responsible lending practices.

Figure 2.2 – Chase Bank

2.2.2. PAYPAL

PayPal is a leading digital payments platform that facilitates online money transfers and serves as an alternative to traditional paper methods like checks and money orders. As part of its services, PayPal offers credit products such as PayPal Credit, which allows users to make purchases and pay over time.PayPal employs credit card approval methodologies to assess applicants for credit products. They evaluate various factors like credit history, income, debt-to-income ratio, employment status, and payment behavior to determine creditworthiness and manage risk. The credit approval process at PayPal leverages sophisticated algorithms and data analytics to automate decision-making and optimize credit assessments. This helps PayPal efficiently extend credit while minimizing risk exposure. By integrating advanced technology into its credit approval workflows, PayPal aims to provide seamless and secure financial services to its customers, promoting financial inclusion and digital commerce.

Figure 2.3 - PayPal

2.2.3. AMERICAN EXPRESS

American Express (Amex) is a global financial services corporation known for its credit card services and travel-related offerings. Amex utilizes sophisticated credit card approval methodologies to assess applicants for its credit card products. This involves evaluating factors like credit history, income, debt levels, employment status, and payment behavior to determine creditworthiness and assign appropriate credit limits. The credit approval process at American Express leverages advanced analytics and machine learning algorithms for efficient and accurate assessments while managing risk effectively. Overall, Amex aims to deliver superior customer experiences and promote responsible credit use through data-driven insights and technological advancements.

Figure 2.4 – American Express

CHAPTER 3

PROPOSED SYSTEM

3.1 FLOW DIAGRAM

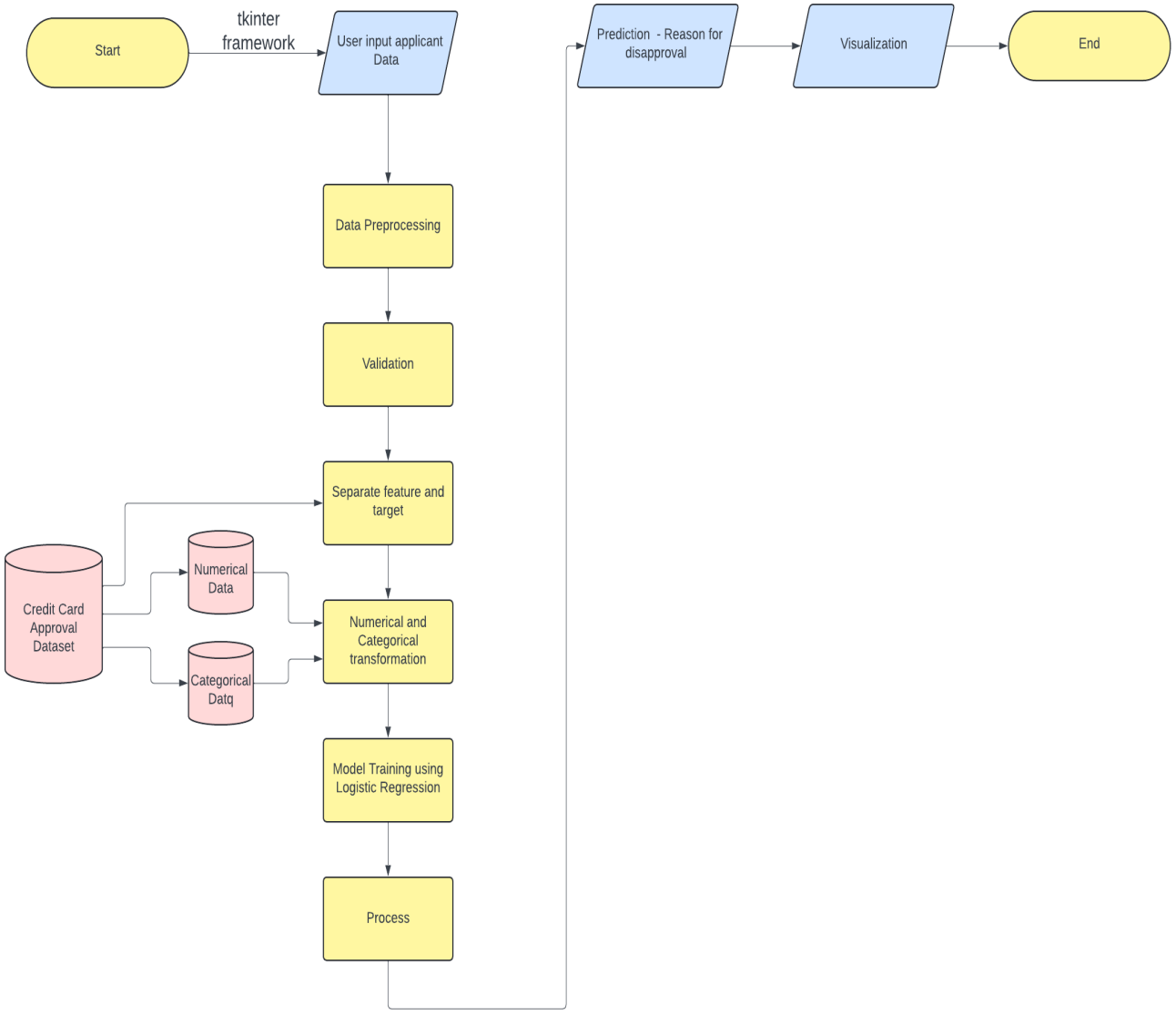

Figure 3.1 - Flow Chart

3.2 PROPOSED SOLUTION

3.2.1. IMPORTED PACKAGES:

- **Tkinter** - A standard library in Python for creating graphical user interfaces (GUIs), enabling the development of desktop applications with elements like buttons, labels, and input fields.
- **Numpy –** A core library for numerical computing in Python, offering support for arrays, mathematical functions, and linear algebra operations.
- **Pandas –** A powerful data manipulation and analysis library that provides data structures like DataFrames for handling and analyzing structured data efficiently.
- **matplotlib.pyplot** - A plotting library in Python that allows the creation of static, animated, and interactive visualizations such as line plots, scatter plots, and histograms.
- **Seaborn** - A statistical data visualization library based on Matplotlib that provides a high-level interface for drawing attractive and informative statistical graphics.
- **sklearn(scikit-learn)** - A popular open-source machine learning library in Python that provides simple and efficient tools for data mining, data analysis, and machine learning tasks, including classification, regression, clustering, and dimensionality reduction.

3.2.2. LIBRARIES USED:

- sklearn.model_selection.train_test_split - A function in Scikit-learn used to split datasets into random train and test subsets, facilitating model evaluation and validation.
- sklearn.pipeline.Pipeline - A utility in Scikit-learn that chains multiple steps (e.g., preprocessing and modeling) into a single workflow, ensuring that these steps are applied sequentially.
- sklearn.linear_model.LogisticRegression - A Scikit-learn class implementing logistic regression, a linear model used for binary classification problems.
- sklearn.cluster.KMeans - A Scikit-learn class that implements the K-means clustering algorithm, partitioning data into k distinct clusters based on feature similarity.

3.2.3. TECHNOLOGIES USED:

3.2.3.1. KMeans Clustering:

KMeans Clustering is an unsupervised machine learning algorithm that partitions a dataset into distinct clusters based on feature similarity. It works by iteratively assigning data points to the nearest cluster centroid and then updating the centroids to the mean of the assigned points. The process continues until the centroids stabilize. KMeans is widely used for pattern recognition and data segmentation, helping identify natural groupings within data. In this context, KMeans helps visualize and understand the distribution of applicants based on income and credit score, aiding in decision-making processes.

3.2.3.2. K-Nearest Neighbors (KNN):

K-Nearest Neighbors (KNN) is a simple, yet powerful supervised learning algorithm used for both classification and regression tasks. In classification, it assigns a data point to the class most common among its k nearest neighbors, where k is a user-defined constant. KNN is non-parametric and instance-based, meaning it makes decisions based on the entire dataset rather than a fixed model. It is particularly effective for small datasets and when the decision boundary is irregular. In this application, KNN helps predict credit card approval by comparing new applicants to similar past applicants.

3.2.3.3. Logistic Regression:

Logistic Regression is a statistical method used for binary classification problems. It models the probability that a given input belongs to a particular class by using a logistic function. The output is a probability value between 0 and 1, which can be thresholded to make a binary decision. Logistic Regression is widely appreciated for its simplicity, interpretability, and effectiveness in scenarios where the relationship between features and the outcome is approximately linear. In the context of credit card approval, Logistic Regression predicts whether an application will be approved based on various applicant features.

3.2.3.4. StandardScaler:

StandardScaler is a preprocessing tool from Scikit-Learn that standardizes features by removing the mean and scaling them to unit variance. This means each feature will have a mean of 0 and a standard deviation of 1, which is essential for ensuring that all features contribute equally to the model. Standardization is crucial when features have different scales, as it helps improve the convergence and performance of many machine learning algorithms. In this project, StandardScaler ensures that numerical features like income and credit score are standardized before being used in model training.

3.2.3.5. OneHotEncoder:

OneHotEncoder is a preprocessing technique from Scikit-Learn used to convert categorical variables into a format that can be provided to machine learning algorithms. It transforms each category value into a new binary column (0 or 1). This is essential for algorithms that cannot work with categorical data directly. By converting categorical features like gender and marital status into numerical format, OneHotEncoder allows these features to be included in the machine learning models. This ensures that the model can properly interpret and use categorical information to make accurate predictions.

3.2.3.6. Pipeline:

Pipeline is a utility in Scikit-Learn that allows for the sequential application of a series of transformations followed by a final estimator. By chaining steps together, it ensures that the entire process from data preprocessing to model training and prediction can be streamlined and made more robust. This is particularly useful for ensuring that all steps are applied consistently and correctly during cross-validation and when making predictions on new data. In this credit card approval application, the Pipeline combines data preprocessing steps (standardization and encoding) with the Logistic Regression model for a seamless workflow.

CHAPTER 4

PROJECT REQUIREMENT

4.1 HARWARE REQUIREMENTS:

- Operation System WINDOWS 7 AND ABOVE
- Any Processor Corei3 and above
- RAM of 512MB+
- Monitor of 14.1 or 15 -17 inch
- Keyboard and Mouse.

4.2 SOFTWARE REQUIREMENTS:

- Windows OS
- Python version 3.0.8 and above
- Jupyter Notebook (anocoda3) and above

4.3 DATASET LOADED:

We have used a dataset named “clean_dataset.csv” as “Credit_dataset” with 690 rows and 16 columns from Kaggle. This contains data about the users and if they are allowed to own a credit card or not.

	Gender	Age	Debt	Married	BankCustomer	Industry	Ethnicity	YearsEmpl
120	0	20.75	10.335	1	1	InformationTechnology	Black	
121	1	39.92	6.21	1	1	Materials	White	
122	1	25.67	12.5	1	1	InformationTechnology	White	
123	0	24.75	12.5	1	1	ConsumerStaples	White	
124	0	44.17	6.665	1	1	Materials	White	
125	0	23.5	9	1	1	Materials	White	
126	1	34.92	5	1	1	Utilities	Black	
127	1	47.67	2.5	1	1	CommunicationServices	Asian	
128	1	22.75	11	1	1	Materials	White	
129	1	34.42	4.25	1	1	ConsumerDiscretionary	Asian	
130	0	28.42	3.5	1	1	Industrials	White	
131	1	67.75	5.5	1	1	Education	Other	
132	1	20.42	1.835	1	1	Energy	White	
133	0	47.42	8	1	1	Education	Asian	
134	1	36.25	5	1	1	Energy	Asian	
135	1	32.67	5.5	1	1	Materials	Black	
136	1	48.58	6.5	1	1	Materials	Black	
137	1	39.92	0.54	0	0	ConsumerStaples	White	
138	1	33.58	2.75	1	1	CommunicationServices	White	
139	0	18.83	9.5	1	1	Industrials	White	

Figure 4.1 Credit_dataset

4.3.1. FEATURES IN DATASET:

- Gender: Categorical variable indicating the gender of the individual (1 for male, 0 for female).
- 2. Age: Numerical variable indicating the age of the individual.
- 3. Debt: Numerical variable indicating the amount of debt the individual has.
- 4. Married: Categorical variable indicating marital status (1 for married, 0 for not married).
- 5. BankCustomer: Categorical variable indicating whether the individual is a bank customer (1 for
- yes, 0 for no).
- 6. Industry: Categorical variable indicating the industry in which the individual is employed.
- 7. Ethnicity: Categorical variable indicating the ethnicity of the individual.
- 8. YearsEmployed: Numerical variable indicating the number of years the individual has been
- employed.
- 9. PriorDefault: Categorical variable indicating whether the individual has a history of prior default
- (1 for yes, 0 for no).
- 10. Employed: Categorical variable indicating current employment status (1 for employed, 0 for not
- employed).
- 11. CreditScore: Numerical variable indicating the individual's credit score.
- 12. DriversLicense: Categorical variable indicating whether the individual has a driver's license (1
- for yes, 0 for no).
- 13. Citizen: Categorical variable indicating citizenship status (e.g., "ByBirth" for citizens by birth,"ByOtherMeans" for naturalized citizens).
- 14. ZipCode: Numerical variable indicating the individual's residential zip code.
- 15. Income: Numerical variable indicating the individual's annual income.
- 16. Approved: Categorical variable indicating whether the individual's credit application was approved (1 for yes, 0 for no).

CHAPTER 5

IMPLEMENTATION

5.1. PROGRAM CODE

```
import tkinter as tk
from tkinter import ttk
from tkinter import messagebox
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler, OneHotEncoder
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.cluster import KMeans
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.metrics import accuracy_score

# Load the dataset (assuming the dataset is in CSV format)
data = pd.read_csv('clean_dataset.csv')

# Display basic statistics of the dataset
print(data.describe())

# Separate features (X) and target (y)
X = data.drop('Approved', axis=1)
y = data['Approved']

# Define categorical and numerical features
categorical_features = ['Gender', 'Married', 'BankCustomer', 'Industry',
'Ethnicity', 'PriorDefault', 'Employed', 'DriversLicense', 'Citizen']
numerical_features = ['Age', 'Debt', 'YearsEmployed', 'CreditScore', 'ZipCode',
'Income']

# Preprocessing pipeline for numerical features
numerical_transformer = StandardScaler()

# Preprocessing pipeline for categorical features
categorical_transformer = OneHotEncoder(drop='first')

# Combine preprocessing pipelines using ColumnTransformer
preprocessor = ColumnTransformer(
    transformers=[
        ('num', numerical_transformer, numerical_features),
        ('cat', categorical_transformer, categorical_features)
    ])
```

```
# Define the model pipeline
pipeline = Pipeline([
    ('preprocessor', preprocessor),
    ('classifier', LogisticRegression())
])

# Split data into train and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)

# Fit the model
pipeline.fit(X_train, y_train)

# Create the main application window
root = tk.Tk()
root.title("Loan Approval Predictor")

def predict_and_display_plots():
    try:
        # Get user input from GUI widgets
        new_applicant_data = {
            'Gender': int(gender_combobox.get()),
            'Age': int(age_entry.get()),
            'Debt': int(debt_entry.get()),
            'Married': int(married_combobox.get()),
            'BankCustomer': int(bank_customer_combobox.get()),
            'Industry': industry_entry.get(),
            'Ethnicity': ethnicity_entry.get(),
            'YearsEmployed': float(years_employed_entry.get()),
            'PriorDefault': int(prior_default_combobox.get()),
            'Employed': int(employed_combobox.get()),
            'CreditScore': int(credit_score_entry.get()),
            'DriversLicense': int(drivers_license_combobox.get()),
            'Citizen': citizen_entry.get(),
            'ZipCode': int(zip_code_entry.get()),
            'Income': int(income_entry.get())
        }

        # Reshape the input data into DataFrame with a single row
        new_applicant_df = pd.DataFrame([new_applicant_data])

        # Make a prediction using the trained model
        prediction = pipeline.predict(new_applicant_df)
        if prediction[0] == 1:
            prediction_result = "Approved"
            result_label.config(text="Prediction: Approved",
foreground="green")
            reason = "All thresholds met"
        else:
            prediction_result = "Rejected"
            result_label.config(text="Prediction: Rejected", foreground="red")
```

```
            # Get the reason for rejection
            reason = reason_for_approval_rejection(new_applicant_data)

        # Call display_plots with the new_applicant_data as an argument
        display_plots(new_applicant_data)
        plot_threshold_comparison(new_applicant_data)

        # Display the result and reason in a message box
        messagebox.showinfo("Prediction Result", f"Prediction:
{prediction_result}\nReason: {reason}")

    except ValueError:
        messagebox.showerror("Error", "Please enter valid numerical values.")

def adjust_thresholds_based_on_clusters(X_cluster, kmeans, new_applicant_data,
debt_threshold, credit_score_threshold, income_threshold):
    # Calculate centroids for each cluster
    centroids = kmeans.cluster_centers_

    # Predict cluster labels
    cluster_labels = kmeans.predict(X_cluster)

    # Find majority class within each cluster
    cluster_majority_class = []
    for label in np.unique(cluster_labels):
        cluster_indices = np.where(cluster_labels == label)[0]
        cluster_approved_count = np.sum(y_test.iloc[cluster_indices] == 1)
        cluster_not_approved_count = len(cluster_indices) -
cluster_approved_count
        majority_class = 1 if cluster_approved_count >
cluster_not_approved_count else 0
        cluster_majority_class.append(majority_class)
    print("Centroids:", centroids)
    # Calculate Euclidean distance between centroids and new applicant data
    distances = []
    for centroid in centroids:
        try:
            distance = np.linalg.norm(centroid - new_applicant_data.values)
            distances.append(distance)
        except TypeError as e:
            print("Error:", e)
            print("Centroid:", centroid)
            print("New Applicant Data:", new_applicant_data.values)
    # Adjust thresholds based on distances and majority class of clusters
    for idx, distance in enumerate(distances):
        if cluster_majority_class[idx] == 1:
            if distance < 0.1:  # Adjust this threshold based on the scale of
your data
                # Adjust thresholds for approved class
                # Calculate percentile-based thresholds
                debt_threshold = np.percentile(data['Debt'], 75)
                credit_score_threshold = np.percentile(data['CreditScore'], 25)
```

```
                income_threshold = np.percentile(data['Income'], 75)

        else:
            if distance < 0.1:  # Adjust this threshold based on the scale of
your data
                # Adjust thresholds for not approved class
                # Calculate percentile-based thresholds
                debt_threshold = np.percentile(data['Debt'], 75)
                credit_score_threshold = np.percentile(data['CreditScore'], 25)
                income_threshold = np.percentile(data['Income'], 75)
    return debt_threshold, credit_score_threshold, income_threshold


def display_plots(new_applicant_data):
    # KNN Plot
    plot_knn(new_applicant_data)

    # Concatenate numerical and one-hot encoded features for X_cluster
    global X_cluster, kmeans
    X_cluster = pd.concat([X_test[numerical_features],
pd.get_dummies(X_test[categorical_features], drop_first=True)], axis=1)

    # Call plot_kmeans_clusters with appropriate arguments
    plot_kmeans_clusters(X_cluster, new_applicant_data)
    plot_approved_not_approved_separately(new_applicant_data, X_test, y_test)

def plot_knn(new_applicant_data):
    # Preprocessing pipeline for categorical features
    categorical_transformer_knn = OneHotEncoder(drop='first')

    # Combine preprocessing pipelines using ColumnTransformer for KNN
    preprocessor_knn = ColumnTransformer(
        transformers=[
            ('cat', categorical_transformer_knn, categorical_features)
        ])

    # Define the model pipeline for KNN
    pipeline_knn = Pipeline([
        ('preprocessor', preprocessor_knn),
        ('classifier', KNeighborsClassifier())
    ])

    # Split data into train and test sets
    X_train_knn, X_test_knn, y_train_knn, y_test_knn = train_test_split(X, y,
test_size=0.2, random_state=42)

    # Fit the KNN model
    pipeline_knn.fit(X_train_knn, y_train_knn)

    # Make predictions using KNN
    y_pred_knn = pipeline_knn.predict(X_test_knn)
```

```
    # Evaluate KNN model
    accuracy_knn = accuracy_score(y_test_knn, y_pred_knn)
    print(f"KNN Accuracy: {accuracy_knn}")

    # Plot KNN results
    # Your plotting code here

def plot_kmeans_clusters(X_cluster, new_applicant_data):
    # KMeans Clustering
    global kmeans
    kmeans = KMeans(n_clusters=2, random_state=42, n_init=10)  # Explicitly set
n_init
    kmeans.fit(X_cluster)

    # Predict cluster labels
    cluster_labels = kmeans.predict(X_cluster)

    # Visualize clusters and classification
    plt.figure(figsize=(12, 6))

    # Plot clusters based on income and credit score
    plt.subplot(1, 2, 1)
    plt.scatter(X_cluster['Income'], X_cluster['CreditScore'],
c=cluster_labels, cmap='viridis', alpha=0.5)
    plt.scatter(new_applicant_data['Income'],
new_applicant_data['CreditScore'], marker='*', s=300, c='blue', label='New
Applicant')
    plt.xlabel('Income')
    plt.ylabel('Credit Score')
    plt.title('Clusters based on Income and Credit Score')
    plt.legend()

    # Plot classification based on KNN
    # Your plotting code here

    plt.tight_layout()
    plt.show()

def plot_approved_not_approved_separately(new_applicant_data, X_test, y_test):
    # Extract approved and not approved data from the test set
    approved_data = X_test[y_test == 1]
    not_approved_data = X_test[y_test == 0]

    # Plot Approved vs. Not Approved
    plt.figure(figsize=(12, 6))

    plt.subplot(1, 2, 1)
    plt.scatter(approved_data['Income'], approved_data['CreditScore'], c='g',
alpha=0.5, label='Approved')
    plt.scatter(not_approved_data['Income'], not_approved_data['CreditScore'],
c='r', alpha=0.5, label='Not Approved')
    plt.scatter(new_applicant_data['Income'],
```

```
new_applicant_data['CreditScore'], marker='*', s=300, c='blue', label='New
Applicant')
    plt.xlabel('Income')
    plt.ylabel('Credit Score')
    plt.title('Approved vs. Not Approved Applicants')
    plt.legend()

    plt.subplot(1, 2, 2)
    plt.scatter(approved_data['Debt'], approved_data['YearsEmployed'], c='g',
alpha=0.5, label='Approved')
    plt.scatter(not_approved_data['Debt'], not_approved_data['YearsEmployed'],
c='r', alpha=0.5, label='Not Approved')
    plt.scatter(new_applicant_data['Debt'],
new_applicant_data['YearsEmployed'], marker='*', s=300, c='blue', label='New
Applicant')
    plt.xlabel('Debt')
    plt.ylabel('Years Employed')
    plt.title('Approved vs. Not Approved Applicants')
    plt.legend()

    plt.tight_layout()
    plt.show()

def plot_threshold_comparison(new_applicant_data):
    # Define thresholds for comparison (these can be adjusted as needed)
    debt_threshold = 280  # Example threshold, replace with your logic
    credit_score_threshold = 5  # Example threshold, replace with your logic
    income_threshold = 1017  # Example threshold, replace with your logic

    # Adjust thresholds based on clusters
    debt_threshold, credit_score_threshold, income_threshold =
adjust_thresholds_based_on_clusters(
        X_cluster, kmeans, new_applicant_data, debt_threshold,
credit_score_threshold, income_threshold)

    # Plot comparison
    plt.figure(figsize=(10, 6))

    plt.bar(['Debt', 'Credit Score', 'Income'], [new_applicant_data['Debt'],
new_applicant_data['CreditScore'], new_applicant_data['Income']],
label='Applicant', alpha=0.6)
    plt.axhline(debt_threshold, color='r', linestyle='--', label='Debt
Threshold')
    plt.axhline(credit_score_threshold, color='g', linestyle='--',
label='Credit Score Threshold')
    plt.axhline(income_threshold, color='b', linestyle='--', label='Income
Threshold')

    plt.xlabel('Feature')
    plt.ylabel('Value')
    plt.title('Threshold Comparison')
    plt.legend()
```

```
    plt.show()

def reason_for_approval_rejection(new_applicant_data):
    # Define your thresholds for approval/rejection
    debt_threshold = 280  # Example threshold, replace with your logic
    credit_score_threshold = 5  # Example threshold, replace with your logic
    income_threshold = 1017  # Example threshold, replace with your logic

    # Adjust thresholds based on clusters
    debt_threshold, credit_score_threshold, income_threshold =
adjust_thresholds_based_on_clusters(
        X_cluster, kmeans, new_applicant_data, debt_threshold,
credit_score_threshold, income_threshold)

    reasons = []

    if new_applicant_data['Debt'] > debt_threshold:
        reasons.append('Debt exceeds threshold')
    if new_applicant_data['CreditScore'] < credit_score_threshold:
        reasons.append('Credit score below threshold')
    if new_applicant_data['Income'] < income_threshold:
        reasons.append('Income below threshold')

    if not reasons:
        reasons.append('All thresholds met')

    return ', '.join(reasons)

# Define the GUI layout and widgets
main_frame = ttk.Frame(root, padding="10")
main_frame.grid(row=0, column=0, sticky=(tk.W, tk.E, tk.N, tk.S))

# Create labels and input widgets for each feature
ttk.Label(main_frame, text="Gender (0=Female, 1=Male):").grid(row=0, column=0,
sticky=tk.W)
gender_combobox = ttk.Combobox(main_frame, values=[0, 1])
gender_combobox.grid(row=0, column=1)

ttk.Label(main_frame, text="Age:").grid(row=1, column=0, sticky=tk.W)
age_entry = ttk.Entry(main_frame)
age_entry.grid(row=1, column=1)

ttk.Label(main_frame, text="Debt:").grid(row=2, column=0, sticky=tk.W)
debt_entry = ttk.Entry(main_frame)
debt_entry.grid(row=2, column=1)

ttk.Label(main_frame, text="Married (0=No, 1=Yes):").grid(row=3, column=0,
sticky=tk.W)
married_combobox = ttk.Combobox(main_frame, values=[0, 1])
married_combobox.grid(row=3, column=1)

ttk.Label(main_frame, text="BankCustomer (0=No, 1=Yes):").grid(row=4, column=0,
```

```
sticky=tk.W)
bank_customer_combobox = ttk.Combobox(main_frame, values=[0, 1])
bank_customer_combobox.grid(row=4, column=1)

ttk.Label(main_frame, text="Industry:").grid(row=5, column=0, sticky=tk.W)
industry_entry = ttk.Entry(main_frame)
industry_entry.grid(row=5, column=1)

ttk.Label(main_frame, text="Ethnicity:").grid(row=6, column=0, sticky=tk.W)
ethnicity_entry = ttk.Entry(main_frame)
ethnicity_entry.grid(row=6, column=1)

ttk.Label(main_frame, text="Years Employed:").grid(row=7, column=0,
sticky=tk.W)
years_employed_entry = ttk.Entry(main_frame)
years_employed_entry.grid(row=7, column=1)

ttk.Label(main_frame, text="Prior Default (0=No, 1=Yes):").grid(row=8,
column=0, sticky=tk.W)
prior_default_combobox = ttk.Combobox(main_frame, values=[0, 1])
prior_default_combobox.grid(row=8, column=1)

ttk.Label(main_frame, text="Employed (0=No, 1=Yes):").grid(row=9, column=0,
sticky=tk.W)
employed_combobox = ttk.Combobox(main_frame, values=[0, 1])
employed_combobox.grid(row=9, column=1)

ttk.Label(main_frame, text="Credit Score:").grid(row=10, column=0, sticky=tk.W)
credit_score_entry = ttk.Entry(main_frame)
credit_score_entry.grid(row=10, column=1)

ttk.Label(main_frame, text="Driver's License (0=No, 1=Yes):").grid(row=11,
column=0, sticky=tk.W)
drivers_license_combobox = ttk.Combobox(main_frame, values=[0, 1])
drivers_license_combobox.grid(row=11, column=1)

ttk.Label(main_frame, text="Citizen:").grid(row=12, column=0, sticky=tk.W)
citizen_entry = ttk.Entry(main_frame)
citizen_entry.grid(row=12, column=1)

ttk.Label(main_frame, text="Zip Code:").grid(row=13, column=0, sticky=tk.W)
zip_code_entry = ttk.Entry(main_frame)
zip_code_entry.grid(row=13, column=1)

ttk.Label(main_frame, text="Income:").grid(row=14, column=0, sticky=tk.W)
income_entry = ttk.Entry(main_frame)
income_entry.grid(row=14, column=1)

# Button to make a prediction and display the result
predict_button = ttk.Button(main_frame, text="Predict",
command=predict_and_display_plots)
predict_button.grid(row=15, column=0, columnspan=2, pady=10)
```

```
# Label to display the result
result_label = ttk.Label(main_frame, text="Prediction: ", font=("Helvetica",
12))
result_label.grid(row=16, column=0, columnspan=2)

# Run the Tkinter event loop
root.mainloop()
```

5.2 OUTPUT

5.2.1. FOR APPROVAL

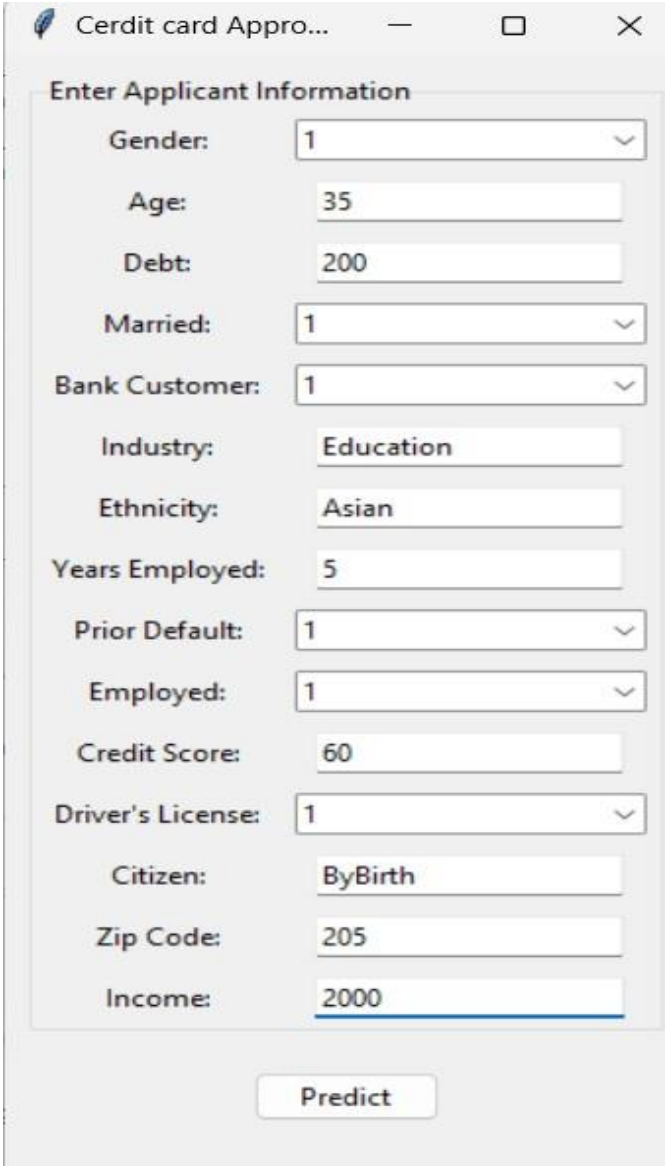

Figure 5.1-User Interface

```
KNN Accuracy: 0.8043478260869565
```

Figure 5.2 – KNN Accuracy

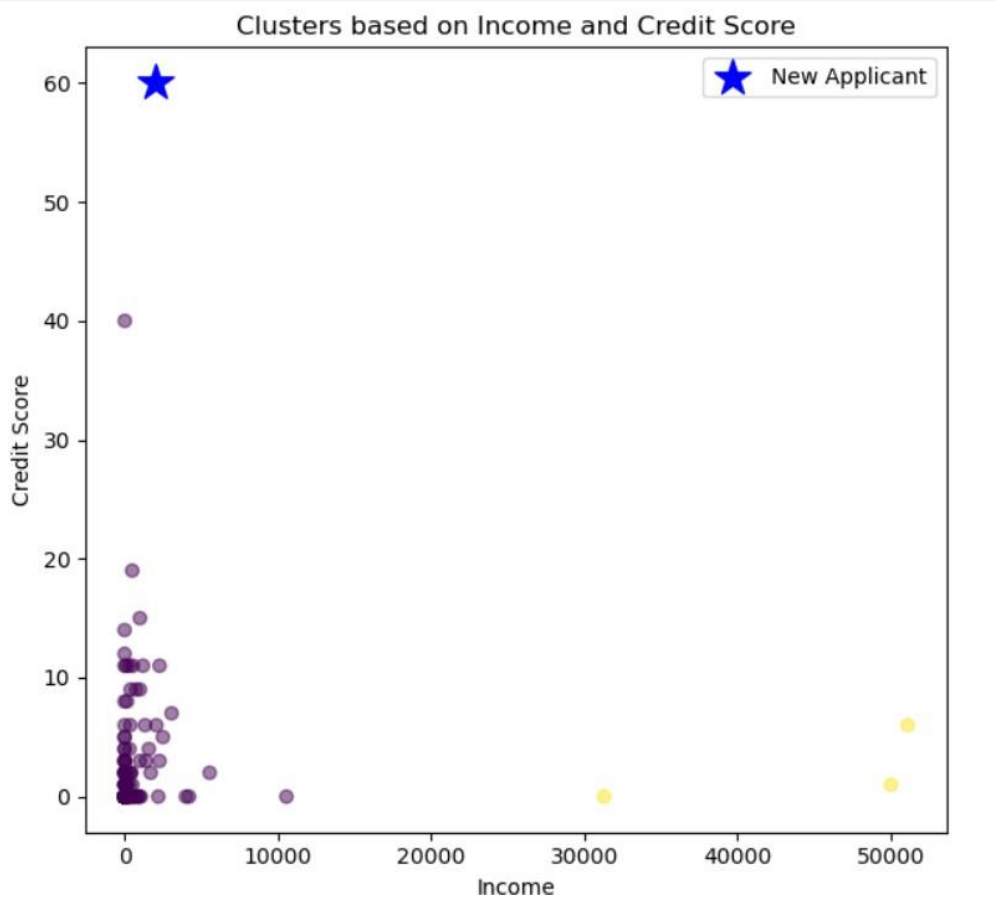

Figure 5.3 – Cluster based on Income and Credit Score

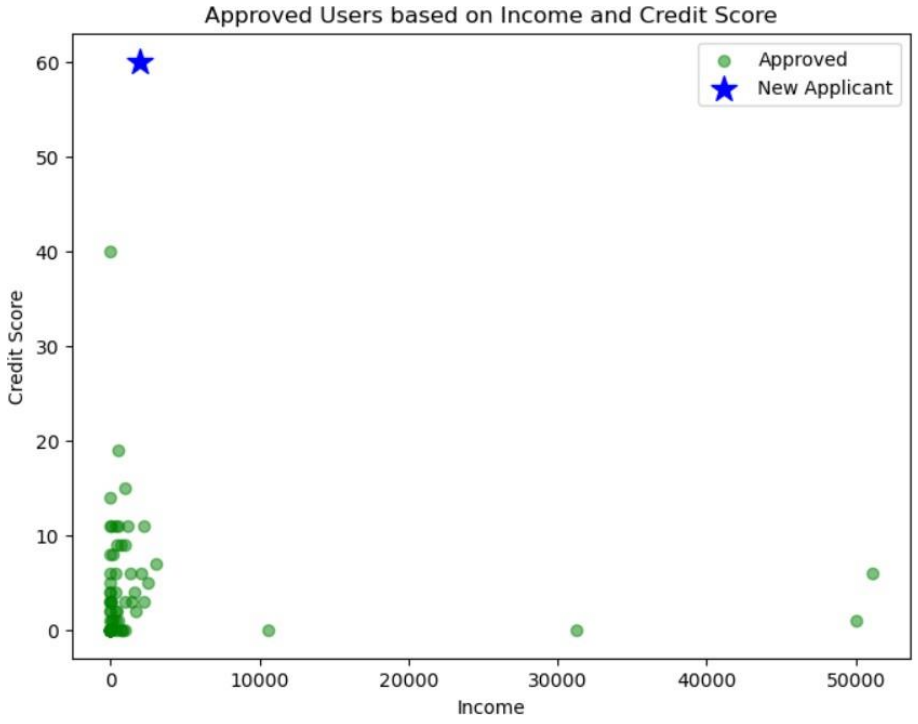

Figure 5.4 – Approved Users based on Income and Credit Score

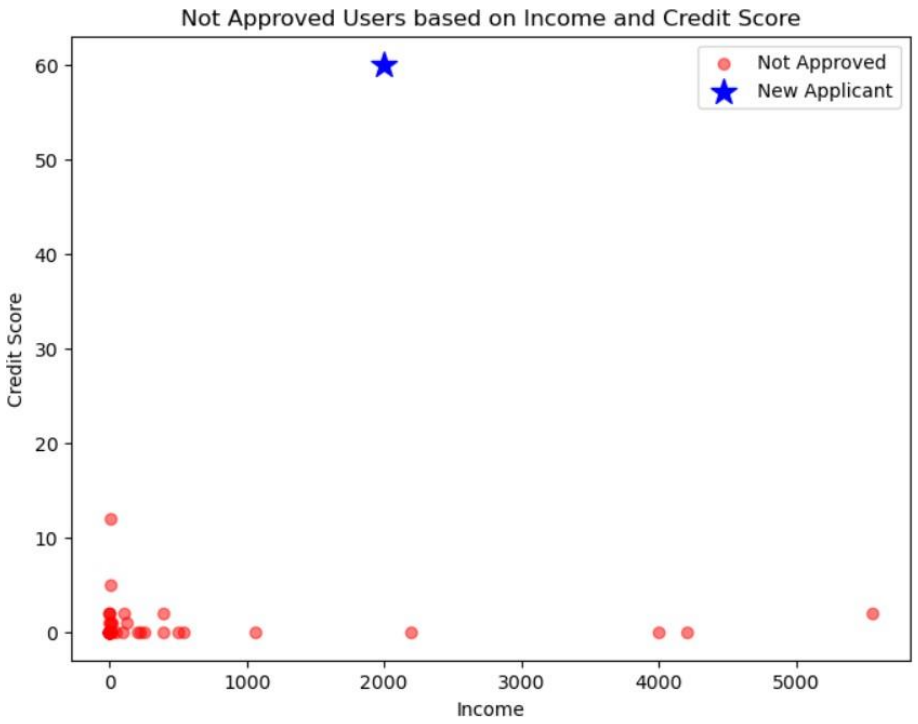

Figure 5.5– Not Approved Users based on Income and Credit Score

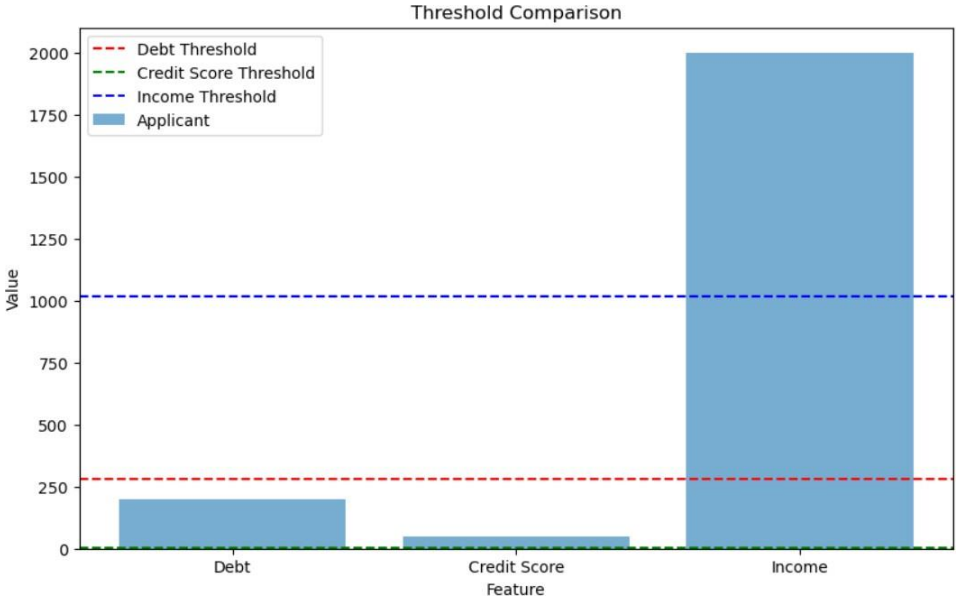

Figure 5.6 – Threshold vs Applicant Value

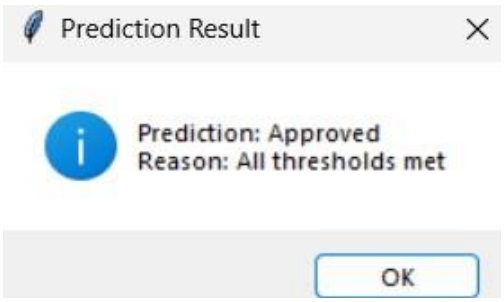

Figure 5.7 – Approval Status

5.2.2. FOR REJECTION

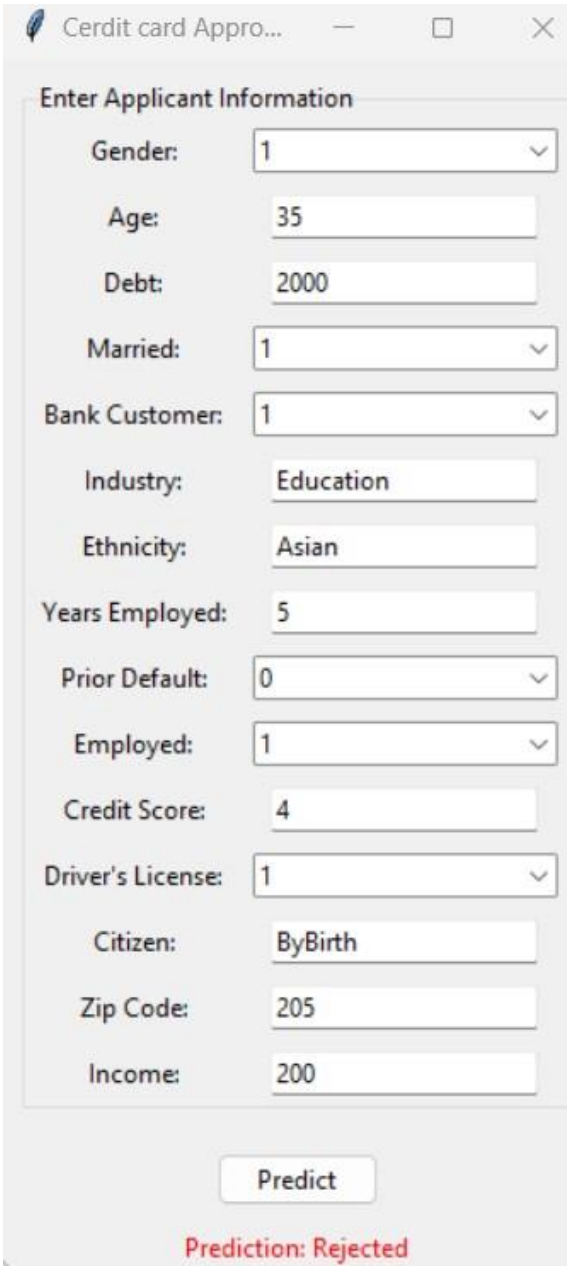

Figure 5.8 – User Interface

```
KNN Accuracy: 0.8188405797101449
```

Figure 5.9 – KNN Accuracy

1

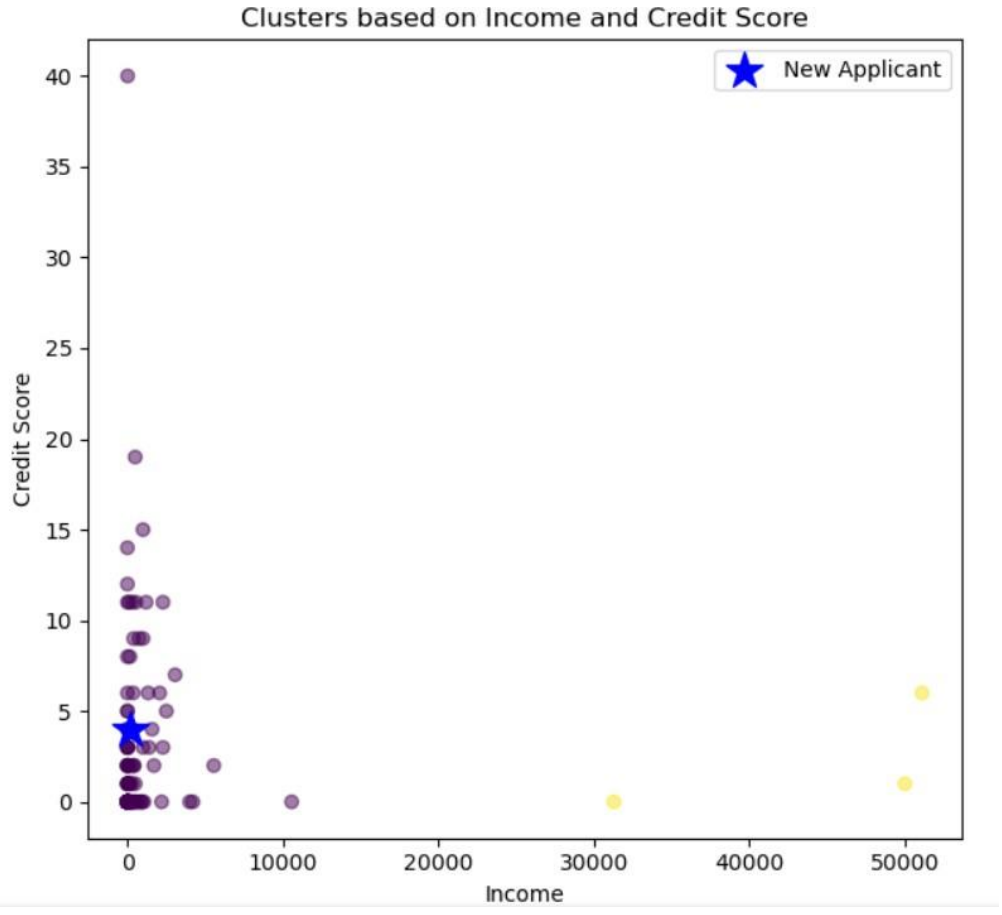

Figure5.10 – Cluster based on Income and Credit Score

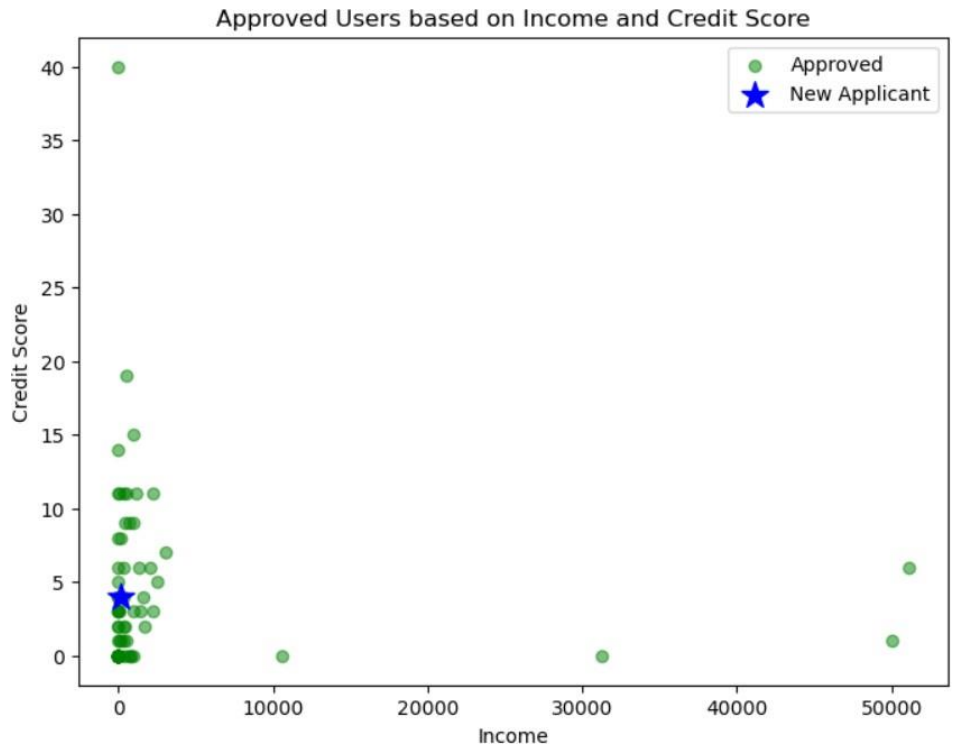

Figure 5.11 – Approved Users based on Income and Credit Score

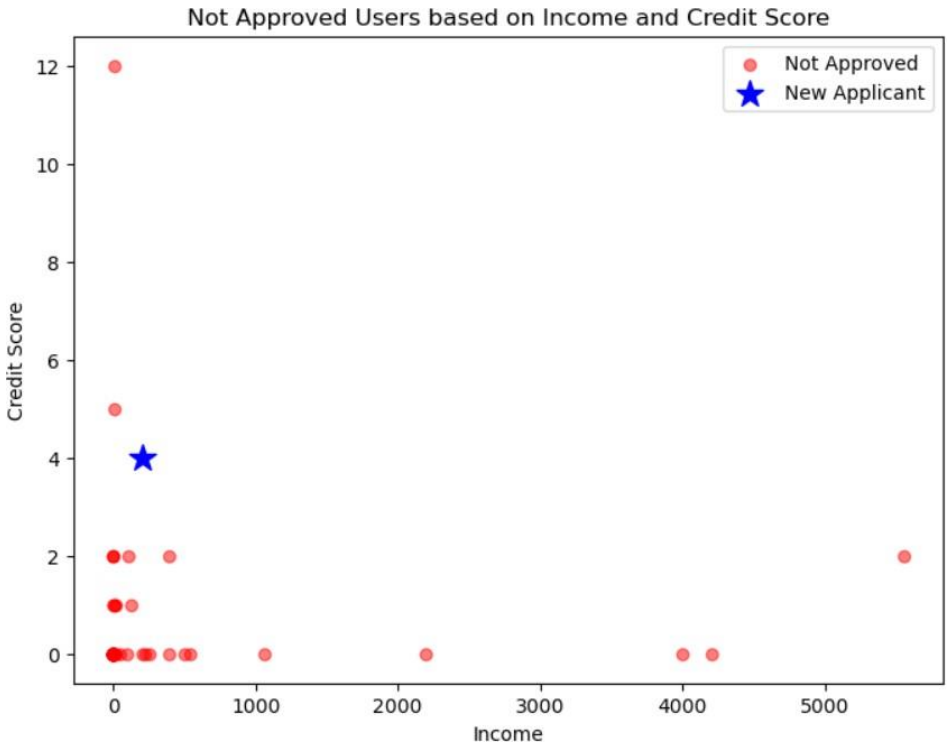

Figure 5.12– Not Approved Users based on Income and Credit Score

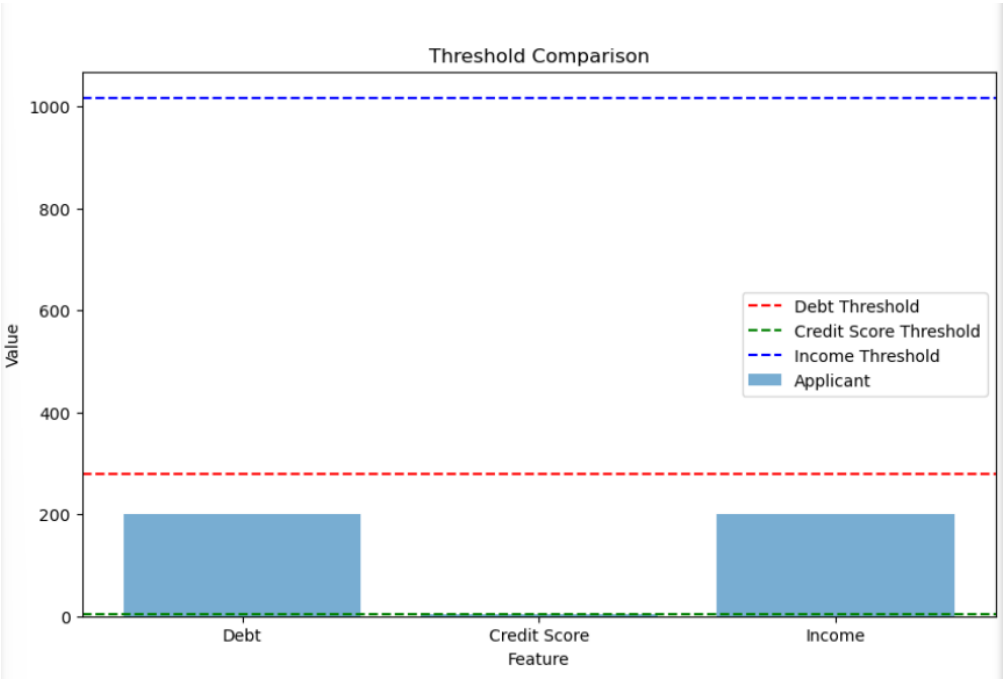

Figure 5.13 - Threshold vs Applicant Value

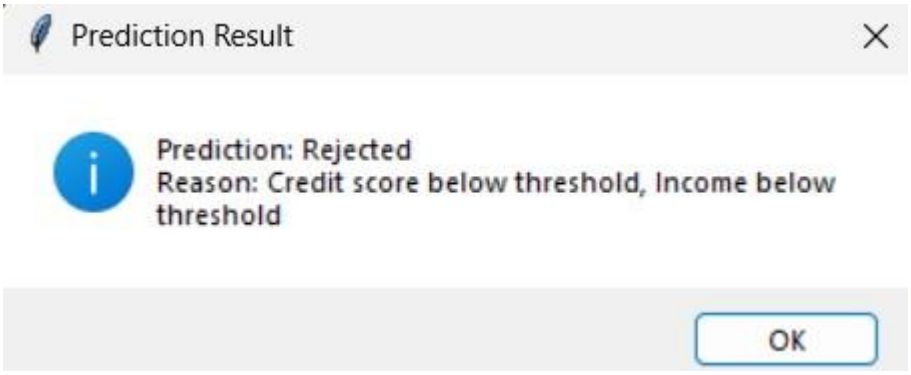

Figure 5.14 – Rejection given with reason

CHAPTER 6

CONCLUSION

The Credit Card Approval project exemplifies the effective integration of machine learning techniques with an intuitive graphical interface to streamline the credit card application process. Utilizing logistic regression and KNN classifiers, the application offers robust predictions on the approval status of credit card applications based on a diverse set of applicant features including age, income, credit score, and employment status.

Data preprocessing is meticulously handled through standardization for numerical features and one-hot encoding for categorical features, ensuring that the machine learning models are trained on well-prepared data. The system also incorporates visualizations such as K-means clustering and KNN classification plots, which provide users with a clear understanding of how their application compares to others in terms of key financial metrics. Additionally, threshold-based decision logic is employed to give straightforward reasons for approval or rejection, adding transparency to the process.

The user interface, built with Tkinter, allows for seamless data entry and real-time prediction generation. The inclusion of graphical plots further enriches the user experience by visually demonstrating the applicant's position relative to predefined approval criteria and historical applicant data. Overall, this project not only automates the decision-making process but also enhances user understanding and trust through clear, data-driven insights.

REFERENCES

1) https://www.kaggle.com/code/towhidultonmoy/project-predicting-credit-card-approvals
2) https://tanmeetsingh.github.io/Predicting-credit-card-approval/
3) https://nycdatascience.com/blog/student-works/data-analysis-on-credit-card-approval/
4) https://www.analyticsvidhya.com/blog/tag/credit-card-lead-prediction/
5) https://www.scribd.com/presentation/688592381/Credit-Card-Approval-Prediction
6) https://www.diva-portal.org/smash/get/diva2:1784289/FULLTEXT02
7) https://community.alteryx.com/t5/Alteryx-Machine-Learning-Discussions/Credit-Card-Approval-Prediction-Using-ML-on-Alteryx/td-p/1203679