IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Date of Submission Sunday, August 27, 2023.

Digital Object Identifier

Exploring Credit Card Approval Prediction with K-Nearest Neighbors (KNN) Classification

PRAJESH SHRESTHA[1], and UJJWAL PAUDEL[1]
[1]Department of Electronics and Computer Engineering, Thapathali Campus, Institute of Engineering, Tribhuvan University, Kathmandu, Nepal (e-mail: prajesh.762417@thc.tu.edu.np, ujjwal.762417@thc.tu.edu.np)

ABSTRACT K-Nearest Neighbors (KNN) is a machine learning algorithm that classifies data points by examining their proximity to neighboring data points.The study explores the fundamental principles of KNN and employing them to perform an extensive classification analysis on the renowned Credit Card Approval dataset. Our analysis involves a thorough examination of the dataset and the process of selecting the most important features used in training the classification model. The experimental results demonstrate a significant accuracy rate of 78% achieved through the KNN classification approach. The results of this study suggest that the KNN algorithm can be used to predict whether an applicant will be approved for a credit card.

INDEX TERMS Credit Card Approval, Classification, K-Nearest Neighbours (KNN)

I. INTRODUCTION

K-Nearest Neighbors (KNN) is a machine learning technique used for classification tasks. It operates by considering the neighboring data points to classify new instances. This approach makes KNN suitable for pattern recognition and classification tasks. It's particularly effective when the relationship between data points is complex or not easily defined by a simple equation. [1]

The Credit Card Approval dataset is a valuable resource for studying the factors that influence credit card approval. The dataset includes a variety of applicant characteristics, such as personal, financial, and credit-related information. KNN operates by identifying similar applicants based on their attributes, allowing the algorithm to classify new applications by considering the characteristics of their nearest neighbors. This dataset serves as a valuable resource for assessing and predicting the approval status of credit card applications.

II. METHODOLOGY

A. THEORY

The K-Nearest Neighbors (KNN) algorithm is a non-parametric supervised learning algorithm. It is used for both classification and regression tasks. Its primary objective is to predict the class of a given instance by assessing the classes of its neighboring data points. This algorithm capitalizes on the intuitive idea that instances with similar attributes often belong to the same class.

KNN's ability to make predictions based on local patterns within the data makes it particularly useful for tasks involving complex relationships that might not be easily captured by a formal equation. [2] [3]

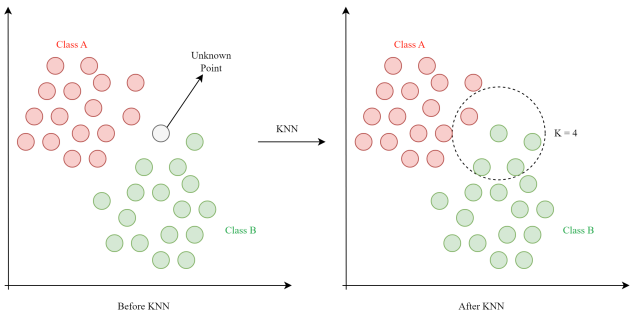

FIGURE 1. K-Nearest Neighbors Visualization

1) Choosing Number of Neighbors (k)

'K' is a hyperparameter that can be tuned to improve the accuracy of the model. A common rule of thumb for the starting value can be an odd number closer to the square root of the number of instance. A smaller 'k' value makes the classification sensitive to noise and outliers, potentially leading to overfitting. Conversely, a larger 'k' value smoothes out decision boundaries but might cause the algorithm to overlook local patterns. Selecting the optimal 'k' can be achieved through techniques like cross-validation or the elbow method, which involves assessing the algorithm's performance for different 'k' values and identifying the point of diminishing returns.

B. ALGORITHM

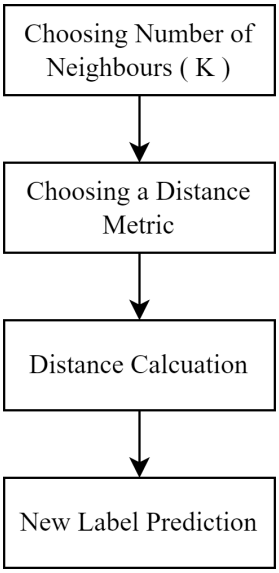

FIGURE 2. KNN Algorithm

1) Choosing a Distance Metric

The distance metric determines how the proximity or similarity between data points is measured. The most common distance metrics are Euclidean distance, Manhattan distance, and Minkowski distance.

2) Distance Calculation

In the "Distance Calculation" step of the K-Nearest Neighbors (KNN) algorithm, distances are computed between the new data point and all instances in the training dataset using the chosen distance metric. Subsequently, these distances are sorted in ascending order to identify the closest neighbors. The next phase involves selecting the K nearest neighbors from the sorted list.

3) New Label Prediction

The class label for the new data point is determined based on the classes of its K nearest neighbors. This involves two common strategies: majority voting and weighted voting.

Majority Voting:

In majority voting, the class label that appears most frequently among the K nearest neighbors is assigned to the new data point. This method assumes that the majority class among the neighbors is likely to be the correct label for the new instance.

$$PredictedClass = Mode(ClassLablesofKNearestNeighbors) \quad (1)$$

Weighted Voting:

In weighted voting, the class labels of the K nearest neighbors are taken into account along with their distances from the new data point. Closer neighbors have a stronger influence on the prediction. Each neighbor's vote is weighted by a factor inversely proportional to its distance from the new instance.

$$PredictedClass = \frac{WeightedSumofClassLabels}{SumofWeights} \quad (2)$$

Weighted voting gives more importance to closer neighbors and can improve accuracy when some neighbors are more relevant than others.

C. SYSTEM BLOCK DIAGRAM

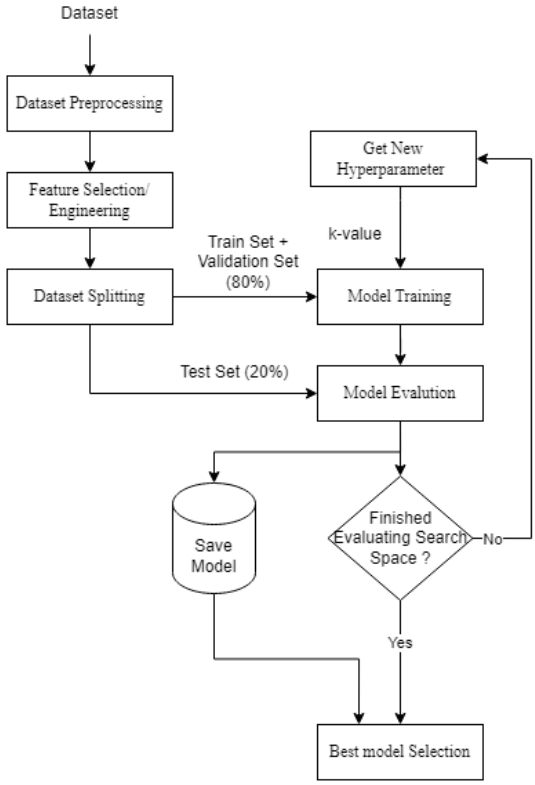

FIGURE 3. System Block Diagram

1) Data Preprocessing

Dataset Preprocessing involves cleaning, transforming, and organizing the raw data to make it suitable for analysis. It includes tasks like handling missing values, scaling features, and encoding categorical variables.

2) Feature Selection/Engineering

In this steps, task are performed to select relevant features or create new features that can enhance the model's performance. Correlation analysis used to identify relationships between features and the target variable and keep the relevant features.

3) Dataset Splitting

This block involves dividing the preprocessed data into training, validation, and testing subsets. The training set is used to train the model, the validation set helps tune hyperparameters, and the testing set evaluates the final model's performance.

4) Model Training

This block represents the training process of your machine learning model using the training dataset. It involves selecting an appropriate algorithm, feeding the data, and adjusting model parameters.

5) Model Evaluation

After training, the model's performance needs to be evaluated using the validation and testing datasets. Metrics like accuracy, precision, recall, F1-score, and others can be used to measure how well the model is performing.

6) Hyperparameter Tuning

This block includes optimizing the hyperparameters of the model to improve its performance. Techniques like grid search, random search, or Bayesian optimization can be used to find the best hyperparameter values.

D. INSTRUMENTATION TOOLS

1) Pandas and Numpy

- Pandas is employed for data processing, providing data structures like DataFrames that facilitate data manipulation and analysis.
- NumPy serves as the foundation for linear algebra operations, enabling efficient handling of arrays and matrices.

2) Matplotlib and Plotly

- Matplotlib and Plotly are visualization libraries utilized to create informative and visually appealing graphs, charts, and plots, aiding in data exploration and presentation.
- Matplotlib offers various plotting capabilities, while Plotly provides interactive and dynamic visualizations.

3) Seaborn

- Seaborn complements Matplotlib by providing a higher-level interface for statistical graphics, making it easier to create aesthetically pleasing and informative plots.

4) Scikit-learn

- *sklearn.metrics*: This submodule within scikit-learn provides various metrics for evaluating model performance, including accuracy, precision, recall, F1-score, ROC curves, and more.
- *sklearn.model_selection.train_test_split*: Used to split the dataset into training and testing subsets, a fundamental step in model evaluation.
- *sklearn.neighbors.KNeighborsClassifier*: Implements the K-Nearest Neighbors classifier, a core component of your classification workflow.
- *sklearn.preprocessing.LabelEncoder*: Employed for label encoding of categorical columns, transforming categorical data into numeric format suitable for machine learning algorithms.

III. RESULTS AND DISCUSSIONS

A. EXPLORATORY DATA ANALYSIS (EDA)

A credit score plays a pivotal role in assessing and approving credit card applications. Within this dataset, we are presented with two distinct sets of records. While the precise origins of this data collection remain undisclosed, it could either represent fictitious information or a sample from the real world. The overarching aim here is to conduct an insightful analysis to discern the specific features or categories that financial institutions typically record from individuals. These records are integral in their evaluation of whether a prospective client possesses a favorable history of credit card payments.

This dataset comprises two distinct sets of records. The first pertains to application data, encompassing information about individuals applying for credit cards. The second dataset focuses on credit status, providing a historical account of applicants' behavior regarding their repayment of credit loans. Evidently, these two datasets were integrated based on a shared client ID to create a unified dataset, from which valuable insights and patterns were extracted.

At the outset, a thorough examination of each record was conducted to scrutinize their attributes, row counts, and data types. Table 1 provides insights into the applicant data, revealing the presence of eight categorical data types, while the remaining data types are predominantly numerical. Conversely, Table 2 sheds light on the credit status data, comprising only three features. Notably, the categorical attribute 'STATUS' is identified as the target label, encompassing seven distinct notations, each signifying various actions taken by the clients.

TABLE 1. Description of Applicant Data

S.N.	Column	Non-Null Count	Dtype
1	ID	438,557	int64
2	CODE_GENDER	438,557	object
3	FLAG_OWN_CAR	438,557	object
4	FLAG_OWN_REALTY	438,557	object
5	CNT_CHILDREN	438,557	int64
6	AMT_INCOME_TOTAL	438,557	float64
7	NAME_INCOME_TYPE	438,557	object
8	NAME_EDUCATION_TYPE	438,557	object
9	NAME_FAMILY_STATUS	438,557	object
10	NAME_HOUSING_TYPE	438,557	object
11	DAYS_BIRTH	438,557	int64
12	DAYS_EMPLOYED	438,557	int64
13	FLAG_MOBIL	438,557	int64
14	FLAG_WORK_PHONE	438,557	int64
15	FLAG_PHONE	438,557	int64
16	FLAG_EMAIL	438,557	int64
17	OCCUPATION_TYPE	304,354	object
18	CNT_FAM_MEMBERS	438,557	float64

From Table 1, a noticeable observation emerges when examining the non-null count for the 'OCCUPATION_TYPE'

TABLE 2. Description of Credit Status Data

S.N.	Column	Non-Null Count	Dtype
1	ID	1,048,575	int64
2	MONTHS_BALANCE	1,048,575	int64
3	STATUS	1,048,575	object

feature; it appears to be significantly lower than that of other features. To substantiate this observation, a missing value plot, as shown in Fig. 4, was generated, confirming the presence of a substantial number of missing values within the 'OCCUPATION_TYPE' category. Further investigation revealed that the total count of 'NAN' values in this feature amounted to 134,203, which represents a considerable proportion of the overall dataset. To address this issue effectively, the decision was made to remove the 'OCCUPATION_TYPE' attribute from the dataset.

Each applicant is presumably uniquely identified by their ID number. Under this assumption, it was determined that there were only 47 instances of duplicate IDs within the applicant record data. These duplicates were subsequently removed from the dataset to ensure data integrity and consistency.

Both datasets contain some intriguing features. For instance, the 'DAYS_BIRTH' column represents the integer value of the number of days an individual has been alive, measured as a negative count from a specific reference date backward to their actual birthdate. Consequently, each entry in this column is represented as a negative number. To enhance interpretability, these negative values were transformed into the age of the individual.

For 'DAYS_EMPLOYED' column, which exhibits positive values when the individual is unemployed and negative values when the applicant has been employed for a certain number of days in the past. For simplification, positive values, representing unemployment, were replaced with zero, while negative values were retained to indicate the number of years the individual has been employed.

While the applicant records consist of a total of 18 features to define each data instance, it's important to note that not all of these features carry significant information for data mining and modeling purposes. For instance, the 'FLAG_MOBIL' column, which indicates whether an individual possesses a mobile phone or not, uniformly reveals that every client in the dataset indeed has a mobile phone. Consequently, this column was deemed redundant and was subsequently removed from the dataset.

Similarly, columns like 'FLAG_WORK_PHONE' and 'FLAG_PHONE,' which capture information regarding whether the clients provided their personal or work phone numbers, along with the 'FLAG_EMAIL' column, which signifies whether clients submitted their email addresses, were found to lack meaningful variation. Consequently, these columns were also eliminated from the dataset, streamlining it for more focused and efficient data analysis and modeling.

The issue of granting credit cards to applicants is inherently linked to outlier detection. Detecting instances of bad client behavior, particularly delayed payments on credit loans, is relatively rare within a large dataset. However, it's worth noting that this dataset contains numerical attributes that themselves exhibit outlier values. These outliers must be addressed and removed to facilitate accurate modeling and improve the effectiveness of subsequent analyses, particularly when applying techniques like K-Nearest Neighbors (KNN).

Figure 5 presents a series of box plots illustrating the numerical categories found within the applicant dataset, revealing several notable patterns. First, when examining the number of children, it becomes evident that the majority of applicants either have one or two children or none at all. This aligns with contemporary family sizes, where smaller households are prevalent. In terms of income, most credit card applicants appear to fall within the middle-class income range, with incomes showing relatively low variance. However, there exists a subset of applicants who exhibit significantly higher incomes. This could be attributed to individuals with a deep understanding of financial management, who comfortably handle regular expenses with credit cards while channeling their income into investments, given the minimal interest cost of credit card loans for them.

Regarding the age distribution of applicants, the majority cluster between 33 and 53 years old. This age range typically signifies individuals with stable employment and diverse income sources, providing them with the confidence to manage credit card payments effectively. The box plot for 'YEARS_EMPLOYED' reveals numerous outliers, notably individuals with employment durations exceeding 20 years. This may suggest government employees, who tend to have more secure and extended tenures compared to those in the private sector. Lastly, the 'CNT_FAM_MEMBERS' distribution highlights numerous outliers, indicating that certain applicants have significantly larger families compared to the majority. These insights shed light on the diverse characteristics of credit card applicants and their financial profiles, offering valuable information for subsequent analyses and decision-making.

To address the presence of outlier values within the dataset, all outliers, with the exception of those pertaining to 'AGE_YEARS,' were systematically removed. This removal process was executed by determining the lower and upper bounds for each numerical feature using their respective interquartile range (IQR).

Upon examining the Credit Status dataset, it becomes apparent that the target label, denoted as 'STATUS,' encompasses seven distinct labels. These labels encapsulate the applicant's historical behavior with respect to previous credit card loans. Given the multi-class nature of 'STATUS,' each class holding varying implications, a crucial preprocessing step involves mapping this multiclass problem into a sensible binary classification scheme. In this regard, we have defined 'Good Clients' as those who have paid in the same month ('C' class) or those who have not taken any loan ('X' class), based

on a moral evaluation of their credit behavior. Conversely, 'Bad Clients' are identified as individuals falling within the '2-5 STATUS' classes, indicating a delay in repayment. Additionally, clients in the '0' class, signifying a one-month delay, are also categorized as 'Good Clients,' whereas those in the '1' class, implying a delay of more than one month, are classified as 'Bad Clients.' This binary classification scheme offers a more interpretable and practical approach to modeling and analyzing credit client behavior, allowing for clearer insights into credit risk assessment.

Implementing the aforementioned procedure has led to the creation of the target label, as illustrated in the pie chart displayed in Figure 6. This visualization unmistakably highlights a substantial imbalance within the dataset, with approximately 99% of individuals categorized as 'Good Clients' and only 1% falling into the 'Bad Client' category.

In the context of predictive modeling, the primary objective is to effectively identify potential 'Bad Client' behavior among applicants. However, given the pronounced class imbalance prevalent in this dataset, it is paramount to employ specific techniques to rectify this imbalance, thereby rendering the dataset more amenable for machine learning algorithms and ensuring equitable and precise predictions.

To address this challenge, a strategic approach was implemented. Since each client record encompasses multiple instances of credit card-related activities, these activities were consolidated, and only the aggregate outcome, whether 'Good' or 'Bad,' was considered. This consolidation led to a revised dataset where the pie chart in Figure 7 illustrates a more balanced distribution, with approximately 88% classified as 'Good Clients' and 12% categorized as 'Bad Clients.' This rebalanced dataset better reflects the real-world scenario and enhances the predictive capabilities of the modeling process.

The two datasets were merged based on their shared client 'ID' to create a unified dataset. Given that the 'ID' feature does not hold any significance for the overall dataset, this column was subsequently removed. Furthermore, duplicate records were eliminated from the combined dataset. As a result, the final dataset comprised a total of 11,058 rows and included 12 features along with a class attribute. The distribution of the target variable from this unified dataset is depicted in the pie chart in Fig. 9.

Figure 10 illustrates a correlational heatmap depicting the relationships between all numerical features and the target attribute 'STATUS.' Upon examination, there is no feature that exhibits a high correlation with the target attribute.

Figure 11 presents a pie chart depicting the gender distribution among credit card applicants, while Figure 8 showcases the percentage of approved credit card applications by gender. Notably, females are the dominant group both in terms of credit card application submissions and approvals. This observation indirectly underscores the growing empowerment of women and their significant contribution to economic development, marking a contrast to societal norms of the past where they were predominantly confined to household roles.

The pie charts in Fig. 12 and Fig. 13 provide insights into the financial aspects and property ownership status of the applicants. These two features hold the potential to yield significant insights into the behavior of individuals when it comes to credit card loan repayments.

The bar plot in Fig. 14 provides a compelling insight into the number of children that applicants could potentially have. The data reveals an interesting pattern, with nearly 70% of applicants indicating that they do not have children. This observation can be interpreted as a reflection of the modern societal trend where many individuals or couples prioritize establishing financial stability before starting a family. Consequently, those who are married but have not yet achieved this level of financial stability may be more inclined to apply for credit cards, as they seek to fulfill their needs and manage their expenses through monthly payments.

Figure 15 illustrates the distribution of annual income, clearly indicating that the majority of applicants report an annual income ranging from 1 to 2 lakhs. The pie chart in Fig. 16 provides insights into the sources of income for the applicants. In general, a significant portion of applicants is likely to be employed in various sectors, enabling them to meet their monthly payment obligations. On the other hand, students comprise the smallest percentage, accounting for only 0.03% of the total applicants. This reflects the fact that most students do not engage in employment to repay credit loans.

Figure 17 presents a bar plot depicting the educational attainment of the clients. Upon examination, it becomes evident that the majority have completed secondary education, with higher education being the second most common level of achievement. In Fig. 18, the family status of the clients is illustrated. A significant portion of them are married, which indirectly underscores the notion that marriage often leads to increased expenses and greater financial responsibility to support the family. This dynamic can potentially drive individuals to seek credit card loans as a means to fulfill their desires and financial obligations. Fig. 19 displays the various types of housing arrangements preferred by the applicants, with a majority residing in either apartments or their own houses. Fig. 20, simply shows the age distribution of the clients.

Figure 21 reveals the distribution of the number of years that applicants have been employed. A prominent spike at the 0th value is noticeable, which corresponds to the unemployed category. This spike underscores that individuals without employment, designated as '0,' constitute a significant portion of the applicant pool. This observation implies that the unemployed are often in greater need of financial support, motivating them to apply for credit cards.

Finally, the scatter plot in Fig. 22 provides a compelling and insightful perspective on the dataset. It reveals that applicants with fewer years of employment or those who are unemployed, coupled with lower total income, are the ones whose credit card applications are predominantly rejected. This decision by the financial institution appears to be both

reasonable and understandable

B. FEATURE SELECTION

While irrelevant and unimportant features were already removed during the preprocessing steps, it was essential to compare both categorical and numerical attributes on a common basis to assess their correlation. To facilitate this, all categorical attributes were transformed into numerical attributes using label encoders. Subsequently, a correlation plot depicting the relationships between all features and the target label, 'STATUS,' is presented in the heatmap shown in Fig. 23.

Upon careful observation of the heatmap, it becomes evident that there is no discernible sign of significant correlation between the features and the target labels. In such a scenario, pursuing further feature selection or assigning higher weights to positively correlated features while applying Weighted KNN does not appear to be a suitable strategy.

C. K-NEAREST NEIGHBORS MODELING

The dataset was divided into a training set and a testing set in an 80:20 ratio. Subsequently, a basic KNN model was employed with a chosen k-value of 7. The results of this classification task are presented in Table 3. Notably, the accuracy achieved through this modeling effort peaked at 77%.

TABLE 3. Classification Report: KNN Model with K = 7

Class	Precision	Recall	F1-Score	Support
Good Client	78.85%	96.85%	86.93%	1744
Bad Client	21.43%	3.21%	5.58%	468
Accuracy			77.03%	2212
Macro Avg.	50.14%	50.03%	46.25%	2212
Weighted Avg.	66.70%	77.03%	69.72%	2212

The support counts for the 'Good Client' and 'Bad Client' labels underscore the dataset's imbalance, a factor that significantly impacts model performance. The corresponding confusion matrix for this modeling effort is displayed in Fig. 24.

Furthermore, it's worth noting that although weighted-KNN may not be the ideal choice for this dataset, an experiment was conducted wherein weighted-KNN was applied with random initialization and assigned high values to certain features. Surprisingly, this experiment yielded the same accuracy of 78.84%. However, the confusion matrix from this experiment, shown in Fig. 25, reveals an interesting outcome. The model predominantly predicts 'Good Clients,' with no predictions made for 'Bad Clients,' indicating that applying weighted-KNN to features that exhibit zero correlation with the target label does not yield meaningful results.

To determine the optimal number of nearest neighbors (k) for the KNN model, a grid search cross-validation was conducted, employing stratified k-folding with k values ranging from 2 to 20 neighbors. This rigorous process revealed that the highest accuracy was achieved when k was set to 10. In a 5-fold stratified cross-validation, the average accuracy reached 78%.

However, when the experiment was replicated using a basic KNN model with k equal to 10, the accuracy dropped significantly to just 63.88%, as detailed in the classification report presented in Table 4. This observation highlights the nuanced nature of model performance, where the selected hyperparameter, k, can impact the outcomes differently depending on the specific context and dataset.

TABLE 4. Classiication Report: GridSearchCV Applied KNN Model

Class	Precision	Recall	F1-Score	Support
Good Client	77.78%	75.86%	76.81%	1744
Bad Client	17.61%	19.23%	18.39%	468
Accuracy			63.88%	2212
Macro Avg.	47.70%	47.55%	47.60%	2212
Weighted Avg.	65.05%	63.88%	64.45%	2212

The confusion matrix for this specific experiment is depicted in Fig. 26. The disparity in performance between cross-validation and the simple experiment could likely be attributed to the quality of the data partitioning into training and validation sets. This explanation gains credence, particularly considering the highly imbalanced nature of the dataset.

Indeed, the dataset's significant class imbalance, with a limited number of 'Bad Client' instances compared to 'Good Client' instances, plays a pivotal role. As evident from the classification report, the support count for 'Bad Client' is a mere 468, while 'Good Client' instances amount to 1744. This imbalance can skew model learning and influence performance, especially in situations where random partitioning may lead to an insufficient representation of 'Bad Client' cases in the training and testing sets.

In an effort to address the class imbalance issue, Synthetic Minority Over-sampling Technique (SMOTE) was applied to create synthetic instances of the minority class ('Bad Client'). However, the outcomes from this strategy did not yield significant improvements in the KNN model's performance. The classification report for this experiment is detailed in Table 5.

TABLE 5. Classiication Report: KNN Model on Balanced Dataset using SMOTE

Class	Precision	Recall	F1-Score	Support
Good Client	60.49%	75.86%	67.31%	1744
Bad Client	38.99%	23.74%	29.51%	1133
Accuracy			55.34%	2877
Macro Avg.	49.74%	49.80%	48.41%	2877
Weighted Avg.	52.02%	55.34%	52.43%	2877

Several factors could contribute to the limited success of SMOTE in improving the KNN model's performance in this context. Firstly, while SMOTE is effective in generating synthetic instances to balance class distributions, its success can be influenced by the complexity and nature of the dataset. In

situations where the decision boundaries between classes are intricate and not easily discernible, as may be the case in this dataset, generating synthetic instances might not effectively capture the underlying patterns.

Secondly, the choice of k-value in the KNN algorithm becomes critical, especially after applying SMOTE. Selecting an appropriate k-value is inherently challenging, and an incorrect choice can lead to the model overfitting or underperforming. In this specific experiment, despite optimizing k via cross-validation, the model may still be sensitive to the intricate class distribution, leading to suboptimal results.

Additionally, SMOTE introduces synthetic data points that can potentially introduce noise into the dataset, further complicating the model's ability to distinguish between genuine and synthetic instances. This noise can impact the overall model performance.

In conclusion, the suitability of the dataset for KNN classification appears limited. Despite various attempts, it becomes evident that relying solely on the KNN model, or any method within its purview, may not yield promising results for this particular dataset.

IV. CONCLUSION

In conclusion, we explored the practical implementation of the K-Nearest Neighbors (KNN) algorithm for classification tasks, using the Credit Card Approval Dataset. Through a systematic approach, we undertook data analysis, feature enhancement, dataset balancing, model training, hyperparameter tuning, and rigorous evaluation. As an outcome, an accuracy rate of 78% achieved by the KNN model, thus demonstrating how KNN can be a valuable tool for making predictions in real-world situations.

REFERENCES

[1] J. H. Friedman, J. L. Bentley, and R. A. Finkel, "Some methods for classification and analysis of multivariate observations," in *Proceedings of the Fifth Berkeley Symposium on Mathematical Statistics and Probability*, 1967.

[2] D. Angluin and L. J. Valiant, "k-Nearest Neighbors in the 21st Century," in *Proceedings of the Twenty-Ninth Annual ACM Symposium on Theory of Computing*, 1997. DOI: 10.1145/258533.258660.

[3] T. Kanungo, D. M. Mount, N. S. Netanyahu, C. D. Piatko, R. Silverman, and A. Y. Wu, "An efficient k-means clustering algorithm: Analysis and implementation," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2002. DOI: 10.1109/34.677568.

PRAJESH S. is an enthusiastic undergraduate student pursuing Computer Engineering at Thapathali College, Institute of Engineering (IOE), Nepal. Curious about the vast world of Machine Learning and Artificial Intelligence, Prajesh enjoys immersing himself in learning and experimenting with diverse ML/AI models. He is fascinated by the potential applications of these technologies and aspires to utilize them for addressing real-world challenges. For any queries or potential collaborations, you can contact Prajesh at prajesh.762417@thc.tu.edu.np.

UJJWAL P. is a dedicated undergraduate student currently pursuing Computer Engineering at Thapathali College, Institute of Engineering (IOE), Nepal. With a keen interest in Machine Learning and Artificial Intelligence, he embraces the role of a passionate learner, constantly exploring and experimenting with various ML/AI models. Driven by a passion to make meaningful contributions to the AI field, he is committed to continuous growth and learning in this exciting and dynamic domain. For any inquiries or potential collaborations, you can reach out to Ujjwal at ujjwal.762416@thc.tu.edu.np.

APPENDIX A
PLOTS AND FIGURES

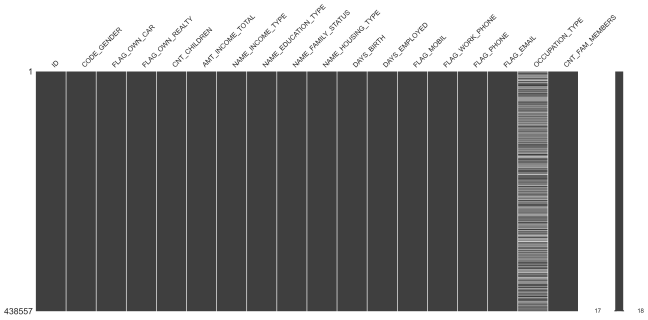

FIGURE 4. Missing Values Visualization Graph

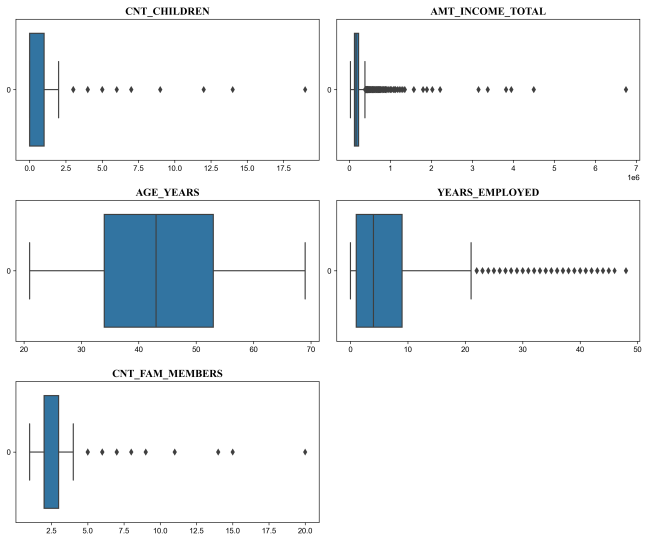

FIGURE 5. Box Plot: Outliers in each Numerical Attributes

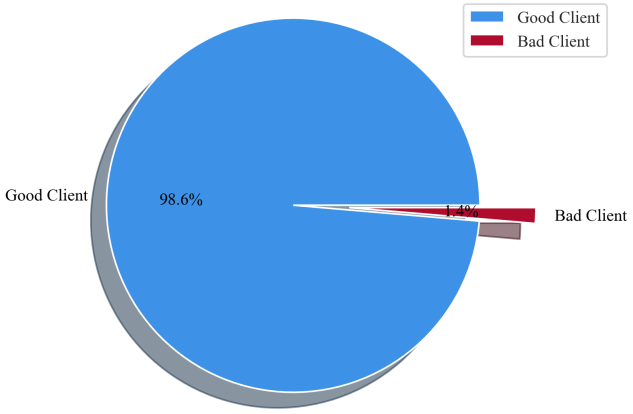

FIGURE 6. Pie Chart: Initial Target Distribution

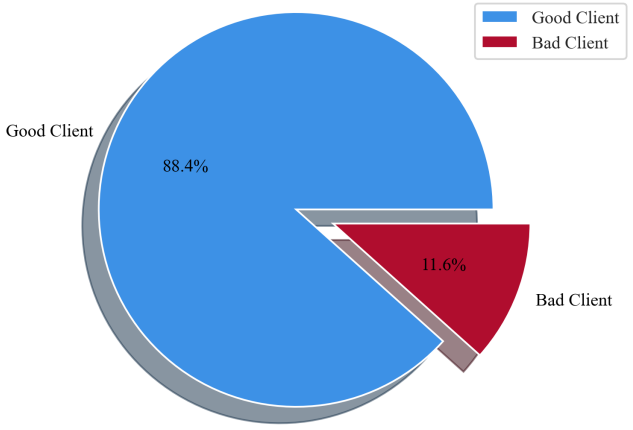

FIGURE 7. Pie Chart: Aggregated Target Distribution

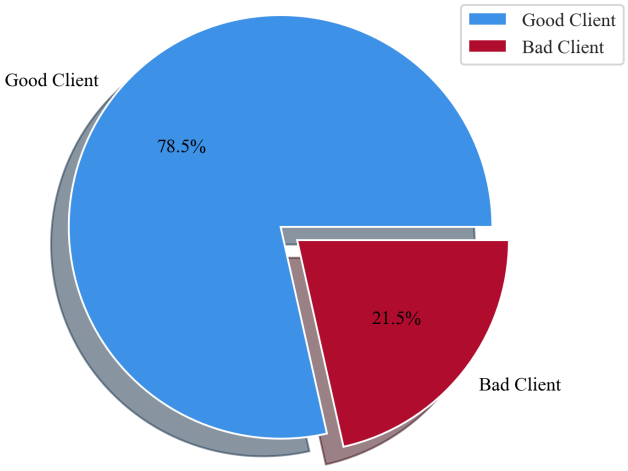

FIGURE 8. Pie Chart: Final Target Distribution

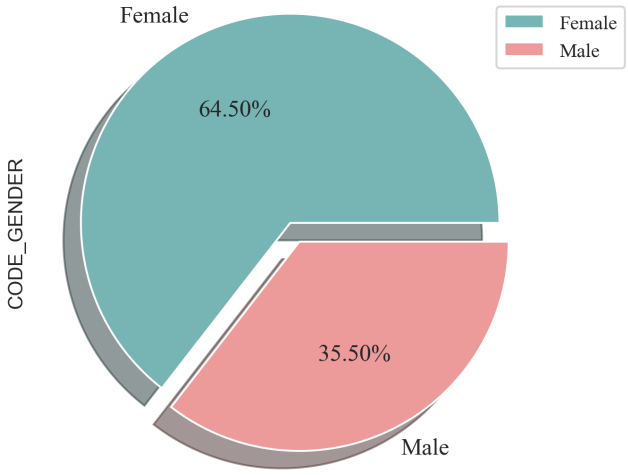

FIGURE 10. Pie Chart: Percentage of Application Submission based on Gender

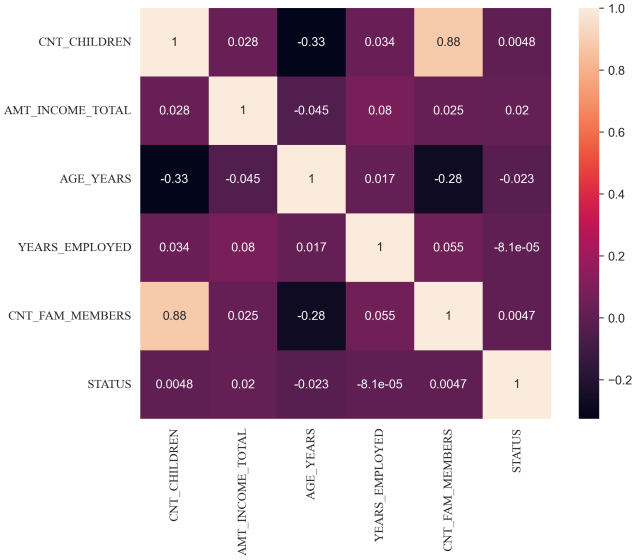

FIGURE 9. Correlational Heat Map for Numerical Attributes

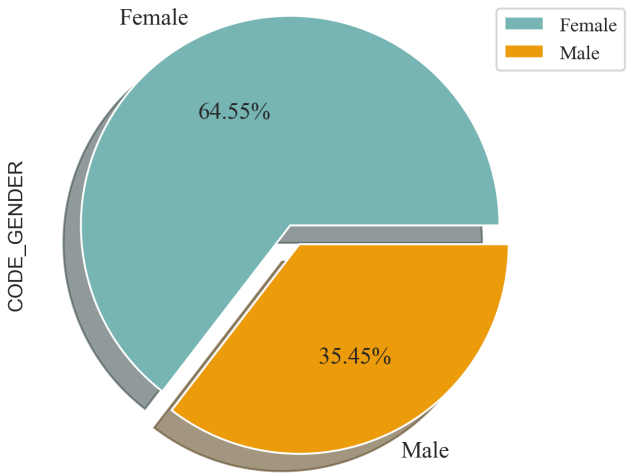

FIGURE 11. Pie Chart: Percentage of Application Approved based on Gender

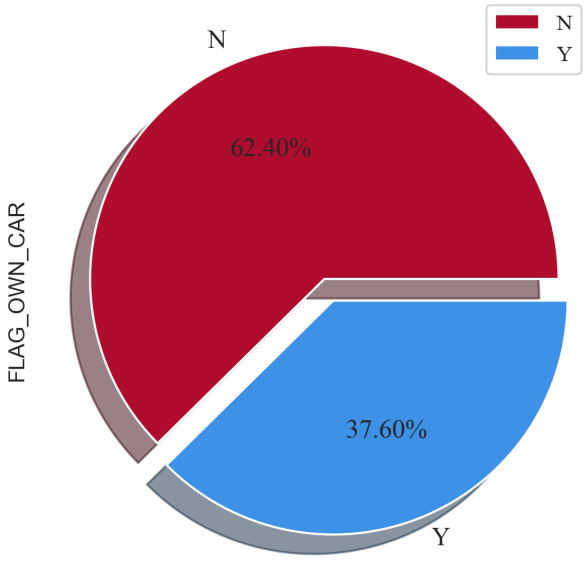

FIGURE 12. Pie Chart: Percentage of Applicants who owns Car

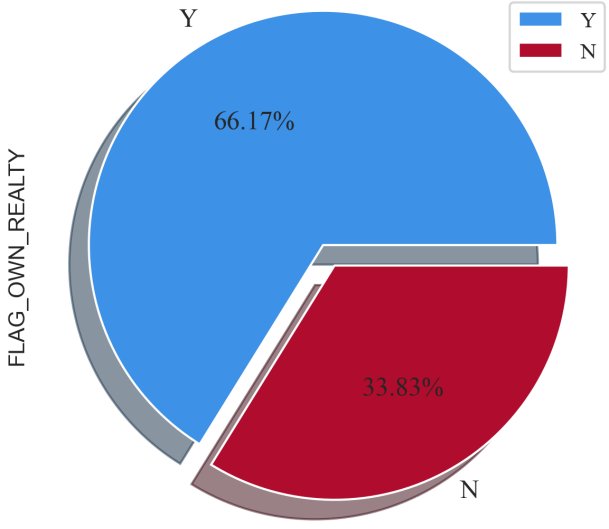

FIGURE 13. Pie Chart: Percentage of Applicants who has Real Estate Property

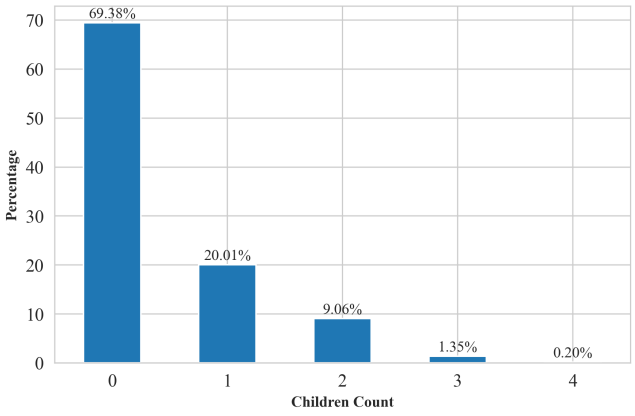

FIGURE 14. Children Counts Bar Plot

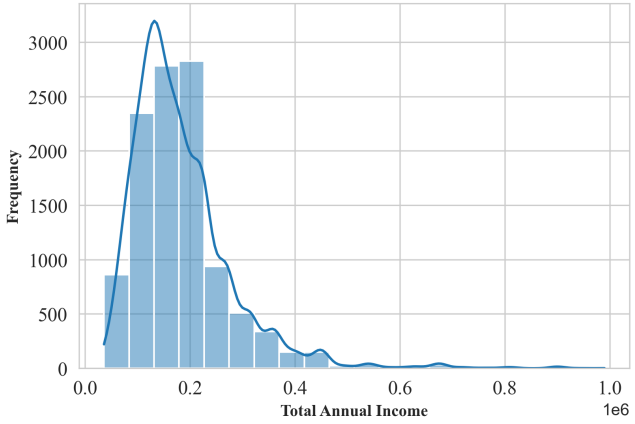

FIGURE 15. Histogram Plot: Total Annual Income

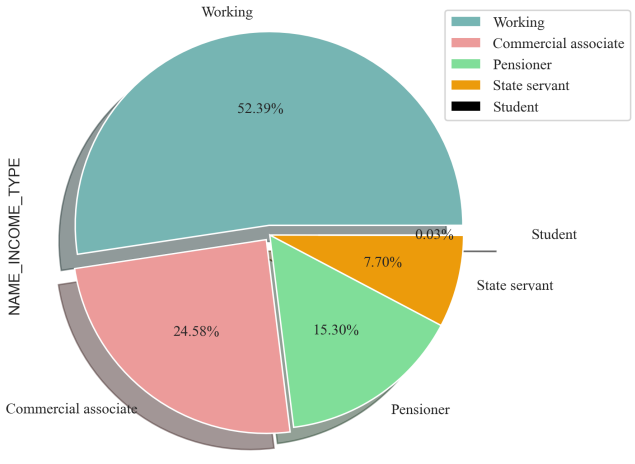

FIGURE 16. Pie Chart: Percentage of Applicants based on Income Types

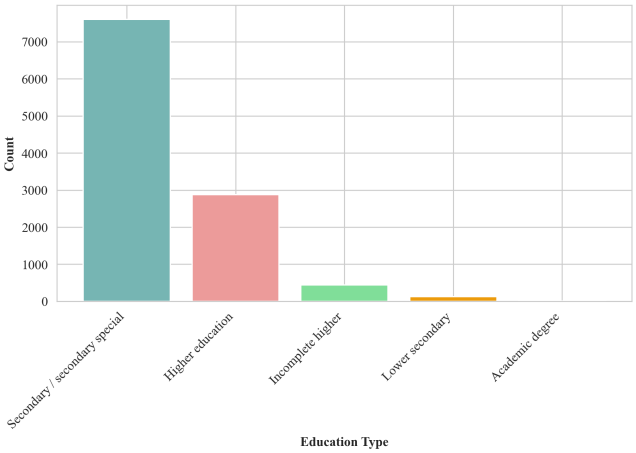

FIGURE 17. Pie Chart: Percentage of Applicants based on Education Acquired

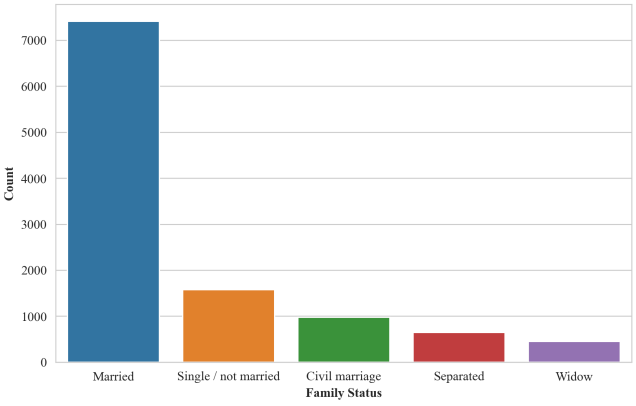

FIGURE 18. Pie Chart: Percentage of Applicants based on Family Status Types

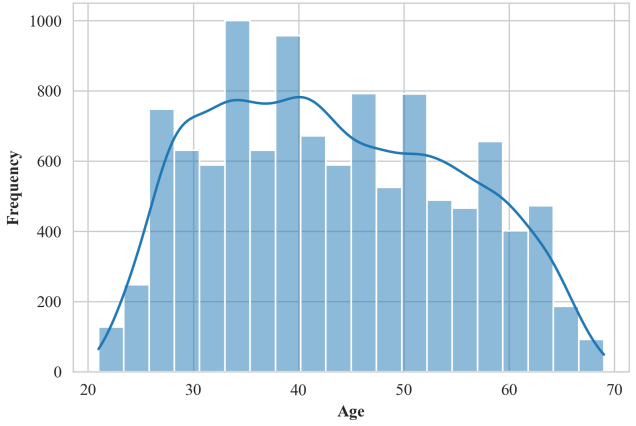

FIGURE 19. Histogram Plot: Applicant Age Distribution

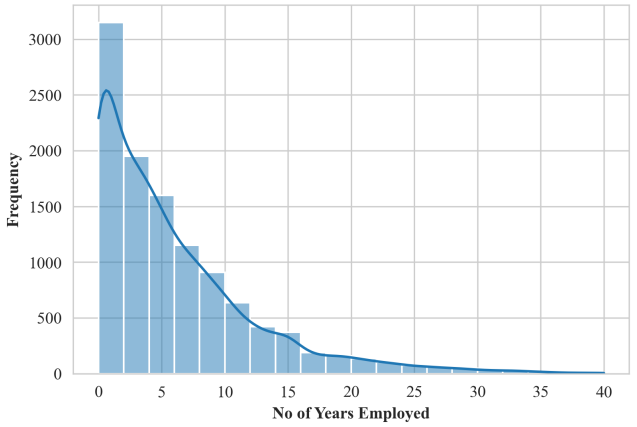

FIGURE 20. Histogram Plot: Applicants Years of Employment

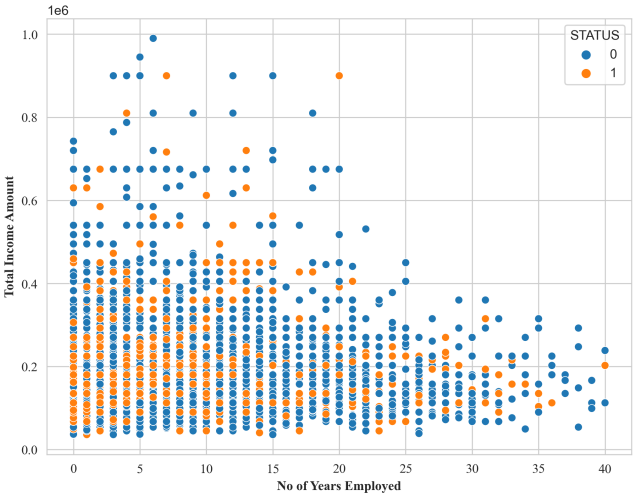

FIGURE 21. Scatter Plot: Years Employed vs Total Annual Income of Applicants

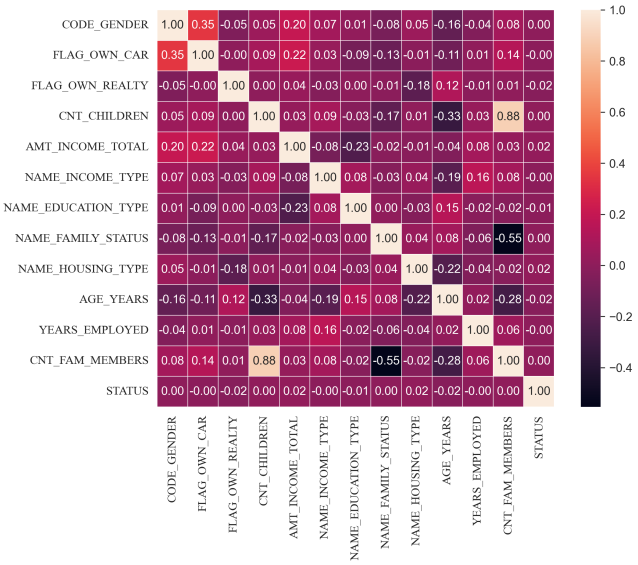

FIGURE 22. Correlation Heat Map between all Features and Target Label

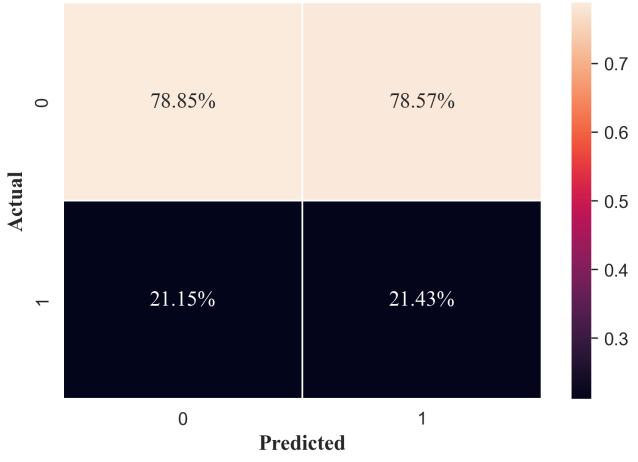

FIGURE 23. Confusion Matrix: Basic KNN Model with K = 7

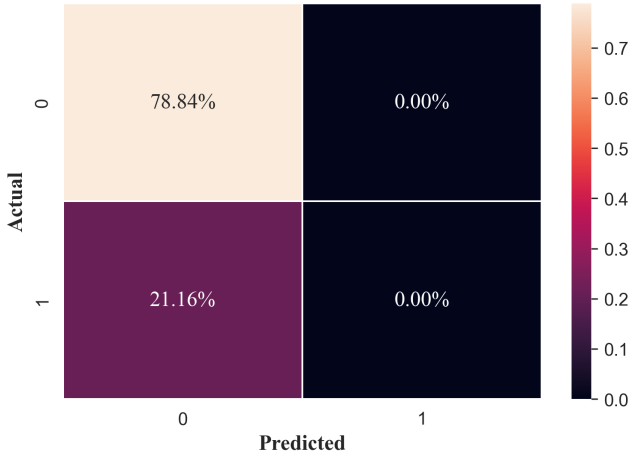

FIGURE 24. Confusion Matrix: Weighted KNN Model

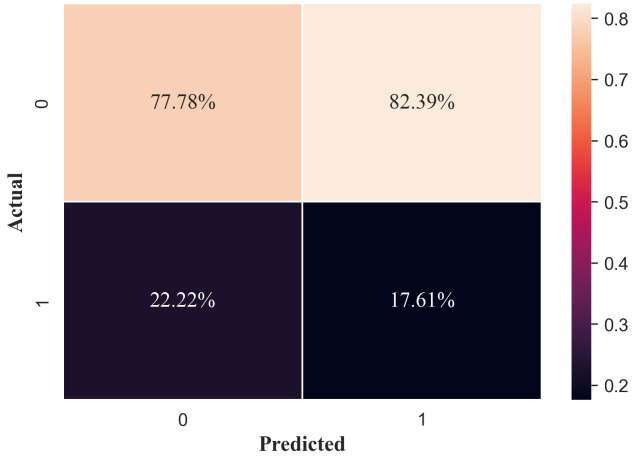

FIGURE 25. Confusion Matrix: KNN Model with K = 10

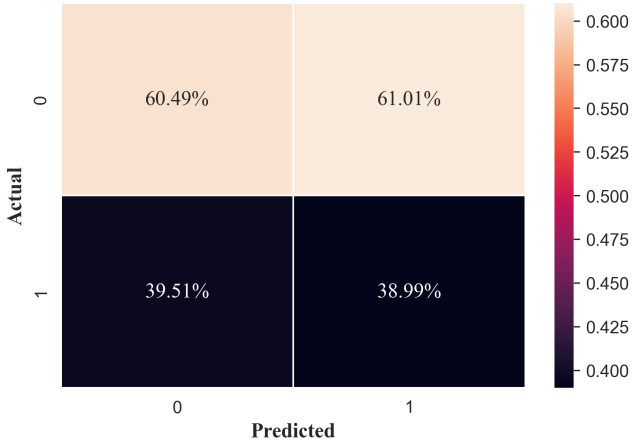

FIGURE 26. Confusion Matrix: KNN Model on Balanced Dataset using SMOTE

APPENDIX A
CODE

1) Imported Modules and Functions

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from pprint import pprint
from tqdm import tqdm
from matplotlib.font_manager import
    FontProperties

from sklearn.metrics import
    classification_report, accuracy_score
    , confusion_matrix
from sklearn.model_selection import
    train_test_split

import matplotlib.style as style
import matplotlib.pyplot as plt
from matplotlib import colors
from mpl_toolkits.mplot3d import Axes3D
from matplotlib.colors import
    ListedColormap
from IPython.display import Image
import plotly.express as px
import plotly.graph_objs as go
import plotly.figure_factory as ff
from plotly.subplots import
    make_subplots
import plotly.offline as pyo
from plotly import tools
import seaborn as sns

import missingno as msno #to visualize
    missing data

from imblearn.over_sampling import SMOTE
import itertools

from sklearn.model_selection import
    train_test_split
from sklearn.metrics import
    accuracy_score, confusion_matrix,
    precision_score,recall_score,
    roc_auc_score,f1_score,roc_curve
from sklearn.neighbors import
    KNeighborsClassifier

from sklearn.preprocessing import
    LabelEncoder #label encoding for
    categorical columns

import warnings
warnings.filterwarnings('ignore')
```

2) Dataset Exploration, Dataset Cleaning

```
pd.set_option('display.max_rows', 10)
pd.set_option('display.max_columns', 6)
app_df = pd.read_csv("./
    application_record.csv")
app_df.head(10)
credit_df = pd.read_csv("./credit_record
    .csv")
credit_df.head(8)
print("Application Record Shape: ",
    app_df.shape)
print("Credit Record Shape: ", credit_df
    .shape)
app_df.info()
credit_df.info()
app_df.describe()
app_df.isnull().sum()

plt_missing_1 = msno.matrix(app_df)
plt.show()

print("TOTAL NAN in 'OCCUPATION_TYPE': "
    , len(app_df[app_df['OCCUPATION_TYPE'
    ].isna()].OCCUPATION_TYPE))
# dropping occupation type which has
    many null values
app_df.drop('OCCUPATION_TYPE', axis=1,
    inplace=True)
# Checking duplicates in 'ID' column
len(app_df['ID']) - len(app_df['ID'].
    unique())
# Dropping duplicate entries from ID
    column
app_df = app_df.drop_duplicates('ID',
    keep='last')
# Checking Non-Numerical Columns
cat_columns = app_df.columns[(app_df.
    dtypes =='object').values].tolist()
print("Non-Numerical Columns: ")
cat_columns
# Checking Numerical Columns
print("Numerical Columns")
app_df.columns[(app_df.dtypes !='object'
    ).values].tolist()
# Checking unique values from
    Categorical Columns
for i in app_df.columns[(app_df.dtypes
    =='object').values].tolist():
    print(i,'\n')
    print(app_df[i].value_counts())
    print('
        ----------------------------------
        ')
app_df['CNT_CHILDREN'].value_counts()
```

```
print('Min DAYS_BIRTH :', app_df['DAYS_BIRTH'].min(),'\nMax DAYS_BIRTH :', app_df['DAYS_BIRTH'].max())

# Converting 'DAYS_BIRTH' values from Day to Years
app_df['DAYS_BIRTH'] = round(app_df['DAYS_BIRTH']/-365,0)
app_df.rename(columns={'DAYS_BIRTH':'AGE_YEARS'}, inplace=True)
app_df.head(1)

# Checking unique values greater than 0
app_df[app_df['DAYS_EMPLOYED']>0]['DAYS_EMPLOYED'].unique()

# As mentioned in document, if 'DAYS_EMPLOYED' is positive no, it means person currently unemployed, hence replacing it with 0
app_df['DAYS_EMPLOYED'].replace(365243, 0, inplace=True)
app_df['DAYS_EMPLOYED']

# Converting 'DAYS_EMPLOYED' values from Day to Years
app_df['DAYS_EMPLOYED'] = abs(round(app_df['DAYS_EMPLOYED']/-365,0))
app_df.rename(columns={'DAYS_EMPLOYED':'YEARS_EMPLOYED'}, inplace=True)

app_df['FLAG_MOBIL'].value_counts()

# As all the values in column are 1, hence dropping column
app_df.drop('FLAG_MOBIL', axis=1, inplace=True)
app_df['FLAG_WORK_PHONE'].value_counts()

# This column only contains 0 & 1 values for Mobile no submitted, hence dropping column
app_df.drop('FLAG_WORK_PHONE', axis=1, inplace=True)
app_df['FLAG_PHONE'].value_counts()

# This column only contains 0 & 1 values for Phone no submitted, hence dropping column
app_df.drop('FLAG_PHONE', axis=1, inplace=True)
app_df['FLAG_EMAIL'].value_counts()

# This column only contains 0 & 1 values for Email submitted, hence dropping column
app_df.drop('FLAG_EMAIL', axis=1, inplace=True)

app_df['CNT_FAM_MEMBERS'].value_counts()
```

3) Dataset Visualization

```
fig, axes = plt.subplots(3, 2, figsize=(12, 10))
font = FontProperties(family='Times New Roman', size = 10)
font_label = FontProperties(family='Times New Roman', size = 14, weight = "bold")
font_xticks = FontProperties(family='Times New Roman', size = 12)
font_yticks = FontProperties(family='Times New Roman', size = 12)
sns.set_style('whitegrid')

sns.boxplot(app_df['CNT_CHILDREN'], orient='h', ax=axes[0][0])
sns.boxplot(app_df['AMT_INCOME_TOTAL'], orient='h', ax=axes[0][1])
sns.boxplot(app_df['AGE_YEARS'], orient='h', ax=axes[1][0])
sns.boxplot(app_df['YEARS_EMPLOYED'], orient='h', ax=axes[1][1])
sns.boxplot(app_df['CNT_FAM_MEMBERS'], orient='h', ax=axes[2][0])

axes[0][0].set_title('CNT_CHILDREN', font = font_label)
axes[0][1].set_title('AMT_INCOME_TOTAL', font = font_label)
axes[1][0].set_title('AGE_YEARS', font = font_label)
axes[1][1].set_title('YEARS_EMPLOYED', font = font_label)
axes[2][0].set_title('CNT_FAM_MEMBERS', font = font_label)

axes[2][1].axis('off')
plt.tight_layout()
plt.show()

high_bound = app_df['CNT_CHILDREN'].quantile(0.999)
print('high_bound :', high_bound)
low_bound = app_df['CNT_CHILDREN'].quantile(0.001)
print('low_bound :', low_bound)
app_df = app_df[(app_df['CNT_CHILDREN'] >= low_bound) & (app_df['CNT_CHILDREN'] <= high_bound)]
```

```
high_bound = app_df['AMT_INCOME_TOTAL'].quantile(0.999)
print('high_bound :', high_bound)
low_bound = app_df['AMT_INCOME_TOTAL'].quantile(0.001)
print('low_bound :', low_bound)
app_df = app_df[(app_df['AMT_INCOME_TOTAL'] >= low_bound) & (app_df['AMT_INCOME_TOTAL'] <= high_bound)]
high_bound = app_df['YEARS_EMPLOYED'].quantile(0.999)
print('high_bound :', high_bound)
low_bound = app_df['YEARS_EMPLOYED'].quantile(0.001)
print('low_bound :', low_bound)
app_df = app_df[(app_df['YEARS_EMPLOYED']>=low_bound) & (app_df['YEARS_EMPLOYED']<=high_bound)]
high_bound = app_df['CNT_FAM_MEMBERS'].quantile(0.999)
print('high_bound :', high_bound)
low_bound = app_df['CNT_FAM_MEMBERS'].quantile(0.001)
print('low_bound :', low_bound)
app_df = app_df[(app_df['CNT_FAM_MEMBERS']>=low_bound) & (app_df['CNT_FAM_MEMBERS']<=high_bound)]

# categorizing 'STATUS' column to binary classification   0 : Good Client and 1 : bad client
credit_df['STATUS'].replace(['C', 'X'], 0, inplace=True)
credit_df['STATUS'].replace(['2','3','4','5'], 1, inplace=True)
credit_df['STATUS'] = credit_df['STATUS'].astype('int')

# Get the value counts as percentages
percentage_counts = credit_df['STATUS'].value_counts(normalize=True) * 100

# Create labels for the pie chart
labels = ['Good Client', 'Bad Client']

# Create a pie chart
plt.figure(figsize=(5, 5))
plt.pie(percentage_counts, explode = (0.1, 0.2), labels=labels, autopct='%1.1f%%', shadow=True, colors=[(0.24, 0.57, 0.90), (0.69, 0.05, 0.18)], textprops = {'family':'Times New Roman', 'size': 10, 'color':'black'})
plt.legend(loc = 'best', bbox_to_anchor = (1.15, 0.95), prop = font)
plt.show()

credit_df_trans = credit_df.groupby('ID').agg(max).reset_index()
credit_df_trans.drop('MONTHS_BALANCE', axis=1, inplace=True)
credit_df_trans.head()

credit_df_trans['STATUS'].value_counts(normalize=True)*100
# Get the value counts as percentages
percentage_counts = credit_df_trans['STATUS'].value_counts(normalize=True) * 100

# Create labels for the pie chart
labels = ['Good Client', 'Bad Client']

# Create a pie chart
plt.figure(figsize=(5, 5))
plt.pie(percentage_counts, explode = (0.1, 0.1), labels=labels, autopct='%1.1f%%', shadow=True, colors=[(0.24, 0.57, 0.90), (0.69, 0.05, 0.18)], textprops = {'family':'Times New Roman', 'size': 10, 'color':'black'})
plt.legend(loc = 'best', bbox_to_anchor = (1.15, 0.95), prop = font)
plt.show()

# merging the two datasets based on 'ID'
final_df = pd.merge(app_df, credit_df_trans, on='ID', how='inner')
final_df.head()

# dropping 'ID' column as it is having only unique values (not required for ML Model)
final_df.drop('ID', axis=1, inplace=True)

# checking if there are still duplicate rows in Final Dataframe
len(final_df) - len(final_df.drop_duplicates())

# Dropping duplicate records
final_df = final_df.drop_duplicates()
final_df.reset_index(drop=True ,inplace=True)

final_df.isnull().sum()
final_df['STATUS'].value_counts(normalize = True)*100
```

```
percentage_counts = final_df['STATUS'].value_counts(normalize = True)*100

labels = ['Good Client', 'Bad Client']

plt.figure(figsize=(5, 5))
plt.pie(percentage_counts, explode = (0.05, 0.05), labels=labels, autopct='%1.1f%%', shadow=True, colors =[(0.24, 0.57, 0.90), (0.69, 0.05, 0.18)], textprops = {'family':'Times New Roman', 'size': 10, 'color':'black'})
plt.legend(loc = 'best', bbox_to_anchor = (1.15, 0.95), prop = font)
plt.show()

plt.figure(figsize = (8, 6))
font = FontProperties(family='Times New Roman', size = 10)
font_label = FontProperties(family='Times New Roman', size = 14, weight = "bold")
font_xticks = FontProperties(family='Times New Roman', size = 10)
font_yticks = FontProperties(family='Times New Roman', size = 10)
sns.heatmap(final_df.corr(), annot=True, square = True)
plt.xticks(font = font_xticks)
plt.yticks(font = font_yticks)
plt.show()

# Create a 2x2 grid of subplots
fig, axes = plt.subplots(2, 2, figsize =(10, 6))

# Plot the first pie chart in the upper-left subplot
axes[0, 0].pie(final_df['CODE_GENDER'].value_counts(), labels=['Female', 'Male'], autopct='%1.2f%%')
axes[0, 0].set_title('% of Applications \nsubmitted based on Gender')

# Plot the second pie chart in the upper-right subplot
axes[0, 1].pie(final_df[final_df['STATUS'] == 0]['CODE_GENDER'].value_counts(), labels=['Female', 'Male'], autopct='%1.2f%%')
axes[0, 1].set_title('% of Applications \nApproved based on Gender')

# Plot the third pie chart in the lower-left subplot
axes[1, 0].pie(final_df['FLAG_OWN_CAR'].value_counts(), labels=['No', 'Yes'], autopct='%1.2f%%')
axes[1, 0].set_title('% of Applications \nsubmitted based on owning a Car')

# Plot the fourth pie chart in the lower-right subplot
axes[1, 1].pie(final_df['FLAG_OWN_REALTY'].value_counts(), labels=['Yes', 'No'], autopct='%1.2f%%')
axes[1, 1].set_title('% of Applications \nsubmitted based on owning Real Estate')
plt.subplots_adjust(wspace = 0.4, hspace = 10)

# Add borders between subplots
for ax in axes.flatten():
    ax.spines['top'].set_visible(True)
    ax.spines['right'].set_visible(True)

# Adjust layout
plt.tight_layout()

# Show the plot
plt.show()

# Plot the first pie chart (Gender)
g1 = final_df['CODE_GENDER'].value_counts().plot.pie(explode =[0.05, 0.05], labels=['Female', 'Male'], autopct='%1.2f%%', shadow= True, colors=["#76B5B3", "#EC9B9A"], textprops={'fontsize': 12, 'family':"Times New Roman"})
# g1.set_title("Percentage of Applications \nSubmitted based on Gender", font = font_label)
plt.legend(loc = 'best', prop = font, bbox_to_anchor = (1.15, 0.95))
plt.show()

# Plot the first pie chart (Gender)
g2 = final_df[final_df['STATUS'] == 0]['CODE_GENDER'].value_counts().plot.pie (explode=[0.05, 0.05], labels=['Female', 'Male'], autopct='%1.2f%%', shadow=True, colors=["#76B5B3", "#EC9B0B"], textprops={'fontsize': 12, 'family': 'Times New Roman'})
# g2.set_title("Percentage of Applications \nApproved based on Gender", font = font_label)
plt.legend(loc = 'best', prop = font, bbox_to_anchor = (1.15, 0.95))
```

```
plt.show()

# Plot the second pie chart (Car Ownership)
g3 = final_df['FLAG_OWN_CAR'].value_counts().plot.pie(explode=[0.05, 0.05], autopct='%1.2f%%', shadow=True, colors=[(0.69, 0.05, 0.18), (0.24, 0.57, 0.90)], textprops={'fontsize': 12, 'family': 'Times New Roman'})

# g3.set_title("Percentage of Applications \nSubmitted based on owning a Car", font = font_label)
plt.legend(loc = 'best', prop = font, bbox_to_anchor = (0.75, 0.85))
plt.show()

# Plot the third pie chart (Realty Ownership)
g4 = final_df['FLAG_OWN_REALTY'].value_counts().plot.pie(explode=[0.05, 0.05], autopct='%1.2f%%', shadow=True, colors=[(0.24, 0.57, 0.90), (0.69, 0.05, 0.18)], textprops={'fontsize': 12, 'family': 'Times New Roman'})
# g4.set_title("Percentage of Applications \nSubmitted based on owning Real Estate")
plt.legend(loc = 'best', prop = font, bbox_to_anchor = (0.99, 0.95))
plt.show()

# Create a 1x3 grid of subplots
fig, axes = plt.subplots(2, 2, figsize=(8, 8))


# Plot the first pie chart (Gender)
g1 = final_df['CODE_GENDER'].value_counts().plot.pie(explode=[0.1, 0.1], labels=['Female', 'Male'], autopct='%1.1f%%', shadow=True, colors=["#76B5B3", "#EC9B9A"], textprops={'fontsize': 10}, ax=axes[0][0])
g1.set_title("Percentage of Applications \nSubmitted based on Gender", font = font_label)


# Plot the first pie chart (Gender)
g2 = final_df[final_df['STATUS'] == 0]['CODE_GENDER'].value_counts().plot.pie(explode=[0.1, 0.1], labels=['Female', 'Male'], autopct='%1.1f%%', shadow=True, colors=["#76B5B3", "#EC9B0B"], textprops={'fontsize': 10}, ax=axes[0][1])
g2.set_title("Percentage of Applications \nApproved based on Gender", font = font_label)

# Plot the second pie chart (Car Ownership)
g3 = final_df['FLAG_OWN_CAR'].value_counts().plot.pie(explode=[0.1, 0.1], autopct='%1.1f%%', shadow=True, colors=["#80DE99", "#00CECB"], textprops={'fontsize': 10}, ax=axes[1][0])
g3.set_title("Percentage of Applications \nSubmitted based on owning a Car", font = font_label)

# Plot the third pie chart (Realty Ownership)
g4 = final_df['FLAG_OWN_REALTY'].value_counts().plot.pie(explode=[0.1, 0.1], autopct='%1.1f%%', shadow=True, colors=["#76B5B3", "#00CECB"], textprops={'fontsize': 10}, ax=axes[1][1])
g4.set_title("Percentage of Applications \nSubmitted based on owning Real Estate")

# Adjust layout
plt.tight_layout()

# Show the plot
plt.show()

# Plot the first pie chart (Gender)
g1 = final_df['CNT_CHILDREN'].value_counts().plot.pie(explode = [0.1, 0, 0, 0.1, 0.25], labels= final_df['CNT_CHILDREN'].value_counts().index, autopct='%1.1f%%', shadow= True, colors=["#76B5B3", "#EC9B9A", "#80DE99", "#EC9B0B", "#000000"], textprops={'fontsize': 10})
g1.set_xlabel("% of Applications submitted based on Children count")
# Adjust layout
plt.tight_layout()
plt.legend(loc = 'right', bbox_to_anchor = (1.05, 0.85))
# Show the plot
plt.show()
```

```

font = FontProperties(family='Times New Roman', size = 10)
font_label = FontProperties(family='Times New Roman', size = 10, weight = "bold")
font_xticks = FontProperties(family='Times New Roman', size = 12)
font_yticks = FontProperties(family='Times New Roman', size = 12)

# Calculate the percentage distribution of 'CNT_CHILDREN'
percentage_distribution = final_df['CNT_CHILDREN'].value_counts(normalize=True) * 100

# Create a bar plot
plt.figure(figsize=(6, 4))
percentage_distribution.plot(kind='bar')

# Add labels and percentages on top of the bars
for x, y in enumerate(percentage_distribution):
    plt.text(x, y + 1, f'{y:.2f}%', ha='center', fontsize = 10, fontfamily = 'Times New Roman')

# Set labels and title
plt.xlabel('Children Count', font = font_label)
plt.ylabel('Percentage', font = font_label)

# Show the plot
plt.xticks(rotation=0, font = font_xticks)
plt.yticks(font = font_yticks)
plt.tight_layout()
plt.savefig('./Figures/10_children_count_bar_plot.png', dpi = 300, bbox_inches = 'tight')
plt.show()

# This graph shows that, majority of applicatant's income lies between 1 to 3 lakh
font = FontProperties(family='Times New Roman', size = 10)
font_label = FontProperties(family='Times New Roman', size = 10, weight = "bold")
font_xticks = FontProperties(family='Times New Roman', size = 12)
font_yticks = FontProperties(family='Times New Roman', size = 12)
plt.figure(figsize=(6, 4))
sns.histplot(final_df['AMT_INCOME_TOTAL'], bins=20, kde=True)
plt.xlabel('Total Annual Income', font = font_label)
plt.ylabel('Frequency', font = font_label)
plt.xticks(font = font_xticks)
plt.yticks(font = font_yticks)
plt.savefig("./Figures/11_total_annual_income_dist.png", dpi = 300, bbox_inches = 'tight')
plt.show()

# Plot the first pie chart (Gender)
g1 = final_df['NAME_INCOME_TYPE'].value_counts().plot.pie(explode = (0.05, 0.03, 0, 0, 0.25), labels= final_df['NAME_INCOME_TYPE'].value_counts().index, autopct='%1.2f%%', shadow=True, colors=["#76B5B3", "#EC9B9A", "#80DE99", "#EC9B0B", "#000000"], textprops={'fontsize': 10, 'family': 'Times New Roman'})
# g1.set_xlabel("% of Applications submitted based on Income Type")
# Adjust layout
plt.tight_layout()
plt.legend(loc = 'right', bbox_to_anchor = (1.26, 0.85), prop = font)
plt.savefig('./Figures/12_income_types.png', dpi = 300, bbox_inches = 'tight')
plt.show()

font = FontProperties(family='Times New Roman', size = 10)
font_label = FontProperties(family='Times New Roman', size = 10, weight = "bold")
font_xticks = FontProperties(family='Times New Roman', size = 10)
font_yticks = FontProperties(family='Times New Roman', size = 10)

# Calculate the counts of each education type
education_counts = final_df['NAME_EDUCATION_TYPE'].value_counts()

# Create a bar graph
plt.figure(figsize=(7, 5))  # Adjust the figure size as needed
```

```
plt.bar(education_counts.index, education_counts.values, color=["#76B5B3", "#EC9B9A", "#80DE99", "#EC9B0B", "#000000"])

# Add labels and title
plt.xlabel("Education Type", font = font_label)
plt.ylabel("Count", font = font_label)

# Rotate x-axis labels if needed for better readability
plt.xticks(rotation = 45, ha='right', font = font_xticks)
plt.yticks(font = font_yticks)

# Show the plot
plt.tight_layout()
plt.savefig('./Figures/13_education_types.png', dpi = 300, bbox_inches = 'tight')
plt.show()

# Plot the first pie chart (Gender)
g1 = final_df['NAME_EDUCATION_TYPE'].value_counts().plot.pie(explode = (0.1, 0, 0.25, 0, 0.35), labels = final_df['NAME_EDUCATION_TYPE'].value_counts().index, autopct='%1.1f%%', shadow=True, colors=["#76B5B3", "#EC9B9A", "#80DE99", "#EC9B0B", "#000000"], textprops={'fontsize': 10, 'family': 'Times New Roman'})
# g1.set_xlabel("% of Applications submitted based on Education")
# Adjust layout
plt.tight_layout()
plt.legend(loc = 'right', bbox_to_anchor = (1.34, 0.85), prop = font)
# Show the plot
plt.show()

# This graph shows that, majority of applicatant's are married
# Sample data
family_status_counts = final_df['NAME_FAMILY_STATUS'].value_counts()

sns.set_style('whitegrid')

# Create a barplot using Seaborn
plt.figure(figsize=(8, 5))
sns.barplot(x=family_status_counts.index, y=family_status_counts.values)
plt.xticks(rotation = 0)  # Rotate x-axis labels for better visibility if needed
plt.xlabel('Family Status', font = font_label)
plt.ylabel('Count', font = font_label)
# plt.title('Applications submitted based on Family Status')
plt.xticks(font = font_xticks)
plt.yticks(font = font_yticks)
plt.savefig('./Figures/14_family_status.png', dpi = 300, bbox_inches = 'tight')
plt.show()

# This graph shows that, majority of applicatant's lives in House / Apartment
family_status_counts = final_df['NAME_HOUSING_TYPE'].value_counts()

# Create a barplot using Seaborn
plt.figure(figsize=(6, 4))
sns.barplot(x=family_status_counts.index, y=family_status_counts.values)
plt.xticks(rotation = 45)  # Rotate x-axis labels for better visibility if needed
plt.xlabel('Housing Types', font = font_label)
plt.ylabel('Count', font = font_label)
plt.xticks(font = font_xticks)
plt.yticks(font = font_yticks)
plt.savefig('./Figures/15_housing_types.png', dpi = 300, bbox_inches = 'tight')
plt.show()

plt.figure(figsize=(6, 4))
sns.histplot(final_df['AGE_YEARS'], bins=20, kde=True)
plt.xlabel('Age', font = font_label)
plt.ylabel("Frequency", font = font_label)
plt.xticks(font = font_xticks)
plt.yticks(font = font_yticks)
plt.savefig('./Figures/15_age_distribution.png', dpi = 300, bbox_inches = 'tight')
plt.show()

# This graph shows that, majority of applicatant's are Employed for 0 to 7 years
plt.figure(figsize=(6, 4))
sns.histplot(final_df['YEARS_EMPLOYED'], bins=20, kde=True)
```

```
plt.xlabel('No of Years Employed', font = font_label)
plt.ylabel("Frequency", font = font_label)
plt.xticks(font = font_xticks)
plt.yticks(font = font_yticks)
plt.savefig('./Figures/16_years_employed.png', dpi = 300, bbox_inches = 'tight')
plt.show()

# This graph shows that, majority of applications are rejected if Total income & years of Employment is less
plt.figure(figsize=(8, 6))
sns.scatterplot(data=final_df, x='YEARS_EMPLOYED', y='AMT_INCOME_TOTAL', hue='STATUS')
plt.xlabel('No of Years Employed', font = font_label)
plt.ylabel("Total Income Amount", font = font_label)
plt.xticks(font = font_xticks)
plt.yticks(font = font_yticks)
plt.savefig('./Figures/17_scatter_plot_employed_vs_income.png', dpi = 300, bbox_inches = 'tight')
plt.show()
```

4) Feature Selection

```
# Taking only the categorical Columns
cat_columns = final_df.columns[(final_df.dtypes =='object').values].tolist()
cat_columns

#Converting all Non-Numerical Columns to Numerical
from sklearn.preprocessing import LabelEncoder

for col in cat_columns:
    globals()['LE_{}'.format(col)] = LabelEncoder()
    final_df[col] = globals()['LE_{}'.format(col)].fit_transform(final_df[col])
final_df.head()

for col in cat_columns:
    print(col , "  : ", globals()['LE_{}'.format(col)].classes_)

correlation_matrix = final_df.corr()

plt.figure(figsize=(8, 6))
plt.xticks(font = font_xticks)
plt.yticks(font = font_yticks)
sns.heatmap(correlation_matrix, annot=True, fmt=".2f", linewidths = 0.01, square = True)
plt.savefig("./Figures/18_Correlation_all_heat_map.png", dpi = 300, bbox_inches = 'tight')
plt.show()
```

5) Data Modeling: KNN Models

```
features = final_df.drop(['STATUS'], axis=1)
label = final_df['STATUS']

from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test = train_test_split(features, label, test_size=0.2, random_state = 10)

# K Nearest Neighbor classification
from sklearn.neighbors import KNeighborsClassifier

knn_model = KNeighborsClassifier(n_neighbors = 7)

knn_model.fit(x_train, y_train)

print('KNN Model Accuracy : ', knn_model.score(x_test, y_test)*100, '%')

prediction = knn_model.predict(x_test)
print('\nConfusion matrix :')
print(confusion_matrix(y_test, prediction))

print('\nClassification report:')
print(classification_report(y_test, prediction))

font = FontProperties(family='Times New Roman', size = 12, weight = 'bold')
plt.figure(figsize=(6, 4))
sns.heatmap(confusion_matrix(y_test, prediction, normalize = 'pred'), fmt = '.2%', annot=True, linewidths=0.5, annot_kws={"fontsize": 12, 'family':'Times New Roman'})

# Add labels and title
plt.xlabel('Predicted', font = font)
plt.ylabel('Actual', font = font)
```

```
plt.savefig("./Figures/19_CM_simple_KNN.png", dpi = 300, bbox_inches = 'tight')
plt.show()

# scaling all features
from sklearn.preprocessing import MinMaxScaler
MMS = MinMaxScaler()
x_train_scaled = pd.DataFrame(MMS.fit_transform(x_train), columns=x_train.columns)
x_test_scaled = pd.DataFrame(MMS.transform(x_test), columns=x_test.columns)

# adding samples to minority class using SMOTE
from imblearn.over_sampling import SMOTE
oversample = SMOTE(sampling_strategy = 0.65, random_state = 69)

x_train_oversam, y_train_oversam = oversample.fit_resample(x_train, y_train)
x_test_oversam, y_test_oversam = oversample.fit_resample(x_test, y_test)

# after using SMOTE
y_train_oversam.value_counts(normalize = True) * 100

# Original majority and minority class
y_train.value_counts(normalize = True) * 100

# K Nearest Neighbor classification
knn_model = KNeighborsClassifier(n_neighbors = 10)
knn_model.fit(x_train_oversam, y_train_oversam)

print('KNN Model Accuracy : ', knn_model.score(x_test_oversam, y_test_oversam)*100, '%')

prediction = knn_model.predict(x_test_oversam)
print('\nConfusion matrix :')
print(confusion_matrix(y_test_oversam, prediction))

print('\nClassification report:')
print(classification_report(y_test_oversam, prediction))

font = FontProperties(family='Times New Roman', size = 12, weight = 'bold')
plt.figure(figsize=(6, 4))
sns.heatmap(confusion_matrix(y_test_oversam, prediction, normalize = 'pred'), fmt = '.2%', annot=True, linewidths=0.5, annot_kws={"fontsize": 12, 'family':'Times New Roman'})

# Add labels and title
plt.xlabel('Predicted', font = font)
plt.ylabel('Actual', font = font)
plt.savefig("./Figures/21_CM_SMOTE_balanced.png", dpi = 300, bbox_inches = 'tight')
plt.show()

import random
weights_list = []
for _ in range(12):
    weights_list.append(random.random())
len(weights_list)
print(weights_list)

clf_KNN = KNeighborsClassifier(metric_params = {'w': weights_list})
clf_KNN.fit(x_train, y_train)

print('KNN Model Accuracy : ', clf_KNN.score(x_test, y_test)*100, '%')

prediction = knn_model.predict(x_test)
print('\nConfusion matrix :')
print(confusion_matrix(y_test, prediction))

print('\nClassification report:')
print(classification_report(y_test, prediction))

font = FontProperties(family='Times New Roman', size = 12, weight = 'bold')
plt.figure(figsize=(6, 4))
sns.heatmap(confusion_matrix(y_test, prediction, normalize = 'pred'), fmt = '.2%', annot=True, linewidths=0.5, annot_kws={"fontsize": 12, 'family':'Times New Roman'})

# Add labels and title
plt.xlabel('Predicted', font = font)
plt.ylabel('Actual', font = font)
plt.savefig("./Figures/20_CM_weighted_KNN.png", dpi = 300, bbox_inches = 'tight')
```

```
plt.show()

features = ['CODE_GENDER', 'FLAG_OWN_CAR', 'FLAG_OWN_REALTY', 'CNT_CHILDREN',
        'AMT_INCOME_TOTAL', 'NAME_INCOME_TYPE', 'NAME_EDUCATION_TYPE',
        'NAME_FAMILY_STATUS', 'NAME_HOUSING_TYPE', 'AGE_YEARS',
        'YEARS_EMPLOYED', 'CNT_FAM_MEMBERS']
weighted = [0.2775000393397501, 0.34514382952446754, 0.03019638152758386,
    0.5569195160369883, 4, 0.9607734156146648, 0.08218285220614585,
    0.0731939869231566, 4, 0.09920635007728695, 0.2678344226623953,
    0.774548695366705]

clf_KNN = KNeighborsClassifier(metric_params = {'w': weighted})
clf_KNN.fit(x_train, y_train)

print('KNN Model Accuracy : ', clf_KNN.score(x_test, y_test)*100, '%')

prediction = knn_model.predict(x_test)
print('\nConfusion matrix :')
print(confusion_matrix(y_test, prediction))

print('\nClassification report:')
print(classification_report(y_test, prediction))

from sklearn.model_selection import KFold, StratifiedKFold, GridSearchCV

KNN = KNeighborsClassifier()

best_parameters = {'n_neighbors': np.arange(2, 12)}

# for information gain and entropy
model_entropy = GridSearchCV(estimator = KNN, param_grid = best_parameters,
    n_jobs = -1, verbose = 0, cv = StratifiedKFold(n_splits = 5, shuffle
    =True, random_state = 42))
model_entropy.fit(x_train, y_train)

print(model_entropy.best_params_)
print("Model accuracy score on Train set : ", model_entropy.best_score_)
y_pred = model_entropy.predict(x_test)
print(f"Accuracy after GridSearch & Stratified K-fold: ", accuracy_score(
    y_true = y_test, y_pred = y_pred))

clf_KNN = KNeighborsClassifier(n_neighbors = 10)
clf_KNN.fit(x_train, y_train)

print('KNN Model Accuracy : ', clf_KNN.score(x_test, y_test)*100, '%')

prediction = knn_model.predict(x_test)
print('\nConfusion matrix :')
print(confusion_matrix(y_test, prediction))

print('\nClassification report:')
print(classification_report(y_test, prediction))

font = FontProperties(family='Times New Roman', size = 12, weight = 'bold')
plt.figure(figsize=(6, 4))
sns.heatmap(confusion_matrix(y_test, prediction, normalize = 'pred'), fmt
    = '.2%', annot=True, linewidths=0.5, annot_kws={"fontsize": 12, 'family':'
    Times New Roman'})

# Add labels and title
plt.xlabel('Predicted', font = font)
plt.ylabel('Actual', font = font)
plt.savefig("./Figures/21_CM_best_KNN.png", dpi = 300, bbox_inches = 'tight')
plt.show()
```