University of Hertfordshire UH

School of Physics, Engineering and Computer Science

MSc Data Science Project

7PAM2002-0206-2023

Department of Physics, Astronomy and Mathematics

Data Science FINAL PROJECT REPORT

Project Title:

Predictive Analysis of Credit Card Approval by Machine Learning Algorithms

Student Name and SRN:

Bubly Babu

22031115

Supervisor: Dr Stephen Kane

Date Submitted: 01 – 05 - 2024

Word Count: 9501

DECLARATION STATEMENT

This report is submitted in partial fulfilment of the requirement for the degree of Master of Science in Data Science at the University of Hertfordshire.

I have read the guidance to students on academic integrity, misconduct and plagiarism information at **Assessment Offences and Academic Misconduct** and understand the University process of dealing with suspected cases of academic misconduct and the possible penalties, which could include failing the project module or course.

I certify that the work submitted is my own and that any material derived or quoted from published or unpublished work of other persons has been duly acknowledged. (Ref. UPR AS/C/6.1, section 7 and UPR AS/C/5, section 3.6). I have not used chatGPT, or any other generative AI tool, to write the report or code (other than where I have declared or referenced it).

I did not use human participants or undertake a survey in my MSc Project.

I hereby give permission for the report to be made available on module websites provided the source is acknowledged.

Student Name printed: BUBLY BABU

Student Name signature: 

Student SRN number: 22031115

UNIVERSITY OF HERTFORDSHIRE

SCHOOL OF PHYSICS, ENGINEERING AND COMPUTER SCIENCE

ACKNOWLEDGEMENT

My deepest gratitude is extended to God for giving me the courage and direction I needed to finish this undertaking.

My deepest appreciation goes to my supervisor **Dr Stephen Kane**, whose invaluable support, patience, and insights have steered me through the research and writing process. His expertise and encouragement have made a significant impact on my work.

Additionally, I would like to express my gratitude to the University of Hertfordshire's personnel and facilities for their support of my research. Their support and assets were essential in conquering many obstacles.

My family has given me support and faith in my abilities, for which I am very thankful. This trip would not have been possible without their unwavering understanding and support. I want to thank my friends one last time for supporting me during this project. I enjoyed the procedure immensely because of their company, guidance, and occasional interruptions.

To everyone who played a part in the success of this project, thank you for your invaluable contribution.

ABSTRACT

To predict credit card approval outcomes, this paper investigates the use of machine learning models. The project's main objective was to identify the most accurate model for this purpose. The dataset contains data such as credit history, income, and employment status. It is derived from the Credit Approval dataset in the University of California, Irvine (UCI) Machine Learning Repository. The study compared Logistic Regression, Decision Tree, and Random Forest models, assessing them with metrics such as accuracy, precision, recall, F1-score, and Receiver Operating Characteristic - Area Under the Curve (ROC AUC) to determine which model performed best.

Random Forest was the best-performing model, achieving an accuracy of 0.855 and demonstrating robust results across various metrics. While Logistic Regression exhibited stable performance, Decision Tree showed signs of overfitting. The report concludes that Random Forest is suitable for real-world applications due to its versatility and reliability. There is potential for future improvements, aimed at refining these models and incorporating new technologies to boost accuracy and improve customer interaction.

CONTENTS

1. INTRODUCTION

The internet has significantly boosted the usage of credit cards, making them a go-to payment option for many (Kibria and Sevkli, 2021). This trend is reinforced by the growing number of shops that accept credit cards and the increasing number of people carrying them in their wallets (Chakravorti and To, 2007). With this surge in popularity, banks are receiving an overwhelming number of credit card applications. However, not all applications are approved; many are turned down for reasons such as existing debt being too high, income levels not meeting the required threshold, or the applicant having too many credit checks. The traditional method of manually reviewing each application proves to be not only slow and prone to mistakes but also costly in terms of time and resources (Ursekar, 2020).

The process of determining whether to grant a credit card involves a detailed examination of the applicant's personal and financial situation. Factors such as the applicant's age, whether they have steady employment, their income, and their financial history are crucial in making this decision. Getting this right is critical; errors can lead to missing out on potential good customers or accepting those who pose a financial risk. As the use of credit cards continues to climb, with over a billion active in the United States alone and billions more globally, it's clear the demand is high. This situation underscores the need for banks to adopt more efficient, technology-driven approaches to process these applications. By doing so, banks can ensure they make informed, timely decisions, keeping pace with the growing demand for credit cards while minimizing the risk of error (Markova, 2021).

Adopting machine learning (ML) techniques for credit card approvals signifies a significant leap towards automation, enhancing both the speed and fairness of financial decisions. By utilizing ML, banks can swiftly sift through extensive data, spotting intricate behaviours that indicate the likelihood of a borrower defaulting. This allows for sharper, more impartial credit assessments, overcoming the drawbacks of older methods that could be slow and sometimes biased. The dynamic nature of ML models, which evolve as they consume more data, ensures they remain accurate and relevant amidst financial shifts. Furthermore, leveraging ML for credit evaluations dramatically accelerates the decision-making process, improving client experiences by delivering quicker credit application outcomes. This advancement not only optimises banks' operational efficiencies, enabling the handling of more applications but also paves the way for a future where credit availability more accurately reflects an individual's financial conduct, thereby effectively mitigating risks.

In the fast-paced world of financial services, the accuracy and speed of processing credit card applications are of utmost importance. This research endeavours to unravel a crucial query: Out of the various ML models available, which one is superior in accurately predicting the outcomes of credit card applications, namely, their approval or rejection by banking institutions? This question spotlights the pressing necessity for banks to evolve from traditional, often outdated credit scoring methods towards leveraging the advanced predictive analytics capabilities of ML. Such a transition is not merely about adopting new technology but signifies a strategic move towards enhancing the precision and efficiency of credit assessments. This shift is pivotal as it underscores our investigation's core objective: to dissect and demonstrate how ML technologies can transform the landscape of credit card approval processes, making them more streamlined, equitable, and in tune with the digital age's demands.

This project positions itself at the forefront of leveraging advanced data science to meet the modern demands of the banking sector, specifically in optimizing the credit card approval

process. This research aims to open the door for more efficient, fair, and accurate credit approval procedures by thoroughly analysing the performance of various machine learning algorithms. The anticipated findings from this study are expected not only to boost the efficiency of financial institutions but also to offer consumers faster and fairer access to credit facilities. As this exploration progresses, it holds the promise of uncovering valuable insights that could redefine the use of ML in enhancing financial services.

1.1 **Aim**

The project aims to determine the most effective ML model for accurately predicting credit card application outcomes—approval or denial by banks. Through a detailed comparison of three ML algorithms, this study intends to find the model that best enhances decision accuracy and processing efficiency for financial institutions.

1.2 **Objective**

This project is dedicated to leveraging cutting-edge data science methods to improve the credit card approval process. Its main goal is to examine and compare the predictive accuracy of three ML algorithms to see which best predicts the outcome of credit card applications. Through this analysis, the project seeks to identify the optimal algorithm for enhancing the decision-making processes in financial institutions, ensuring credit card approvals are more consistent and efficient. The ultimate objective is to refine how financial institutions handle credit approvals, making the process faster and more accurate for the benefit of both the banks and their customers.

2. BACKGROUND

In this part, a review of past research and expert perspectives on employing ML algorithms for forecasting credit card approvals is outlined. It encompasses an overview of past methodologies, results, and prospective avenues for advancing predictive analysis in credit card approval processes.

To determine the three models—Logistic Regression (LR), Linear Support Vector Classification (Linear SVC), and Naive Bayes Classifier (NBC)—effectiveness in predicting the results of credit card approval, an analysis was conducted as part of the investigation by Zhao (2022). According to the results, Linear SVC performed better than the others, producing an excellent balanced accuracy of 89.09% and an accuracy rate of 88.48%. Nevertheless, the study emphasises the complex process of selecting a model, stressing the impact of variables on predictive performance, including solver selection and dataset features. As a result, it promotes additional research into relevant topics including reject inference algorithms and selection bias, as well as ways to improve computational efficiency. Credit approval prediction models are expected to benefit from these efforts to become more accurate and useful in real-world scenarios.
A major issue facing the banking industry is examined in the study by Therese et al. (2022) on the use of supervised learning algorithms to forecast credit card approvals. The study aims to improve credit card approval decision accuracy by utilising K-Nearest Neighbours (KNN), Decision Tree (DT), NBC, and LR algorithms. With an astounding accuracy rate of 98.13%, the DT algorithm stands out as the most successful. Furthermore, the authors create a website with Flask Python libraries to streamline the credit card application procedure and reduce the likelihood of credit card approval risks for qualified applicants. The increasing occurrence of credit card usage in India highlights the importance of these predictive models in enabling smooth financial transactions. The efficiency of credit card approval is improved by this suggested method, which also demonstrates how ML algorithms can be used to solve important problems facing the banking sector.

In order to optimise decision-making frameworks within the banking industry, the study by Duan (2020) focuses on the practical application and evaluation of supervised data mining algorithms for credit card approval processes. In the context of credit scoring and fraud detection, the research comprises a thorough analysis of several models, including LR, DT, KNN, and Neural Networks. The study aims to replace manual review procedures with automated ones through the use of these algorithms, which will improve operational efficiency and lower labour costs. The main conclusions show that the Neural Network is the best-performing model, with an accuracy rate of 76.7%. The LR and DT (information-based) models come in second and third, respectively, with accuracy rates of 75.73%. On the other hand, KNN performs comparatively worse, with an accuracy rate of 67.48%. Additionally, the study makes use of methods and visualisation tools that are specific to each model in order to clarify the importance of various factors in credit approval decisions. Gradient Boosting Machine (GBM) is one tool that helps credit card applicants with their self-evaluation by helping to identify variable influences. Furthermore, the utilisation of LR facilitates the explanation of the likelihood-based correlation between variables and approval probabilities, providing a significant understanding of crucial elements like debt circumstances and credit score levels. This reduces the need for manual review processes and increases the overall effectiveness of credit card approval protocols by enabling financial institutions and applicants to make well-informed decisions.

Recognising the increasing reliance on automated processes to streamline credit assessment in the face of rising demand, a thorough investigation was undertaken (Bansal and Punjabi, 2021) into the efficacy of various supervised ML classifiers in forecasting credit card approvals. Their study aims to reduce risks associated with manual review protocols and speed up decision-making by closely examining variables like credit history and income. The authors show how their approach can improve model performance by using a variety of ML models, such as Random Forest (RF) and LR, along with extensive data preprocessing. The use of hyperparameter tuning techniques, which improves model robustness and helps determine the best classifier for predicting credit card approvals, is especially noteworthy. This analysis offers insightful information for future research directions and useful applications in credit assessment systems. The results show that the RF classifier—which achieved an impressive F1-Score of 0.868 and an Area Under the Curve (AUC) value of 0.865—is the most promising model for predicting credit card approval. Subsequent investigations could delve into the application of TensorFlow to enhance model precision and expand the range of supervised and unsupervised learning models to determine the best classifier for credit evaluation.

Sayjadah et al. (2018) conducted a study in which they used ML techniques to predict credit card default. This research was conducted in response to the growing challenge of rising credit card default rates. Their thorough analysis compares the performance of RF, Rpart DT, and LR algorithms in predicting credit default; RF is the most successful model, showing the best area under the curve metrics and accuracy. According to the research, big data has a profoundly transformative effect on banking operations. Financial institutions must use data analytics to navigate complex markets and understand customer behaviour. Additionally, it highlights how crucial predictive analytics is to proactive risk management and customised customer interaction. The study concludes by recommending the use of ML techniques in banking operations to improve operational efficiency and optimise decision-making processes.

Agarwal et al. (2020) explore the significance of sophisticated classification techniques in credit card default prediction. To determine the best methods for identifying possible defaulters, they carefully examine a variety of algorithms, including NBC, DT, KNN, and LR. The performance of the original dataset and one that underwent principal component analysis (PCA) processing are compared in the study. Evaluation of various metrics is done, and the results show that LR is the best model because of its higher accuracy in both datasets (Accuracy, Precision, F1-Score, Recall, and Receiver Operating Characteristic Curve (ROC)). These results highlight the importance of strong classification techniques in credit risk assessment, which is critical for banks. The relationship between PCA and classification algorithms for improved predictive modelling in credit card default prediction could be further investigated in future studies.

The literature review presents a comprehensive analysis of ML applications in predicting credit card approvals, highlighting the effectiveness of various algorithms from studies by Zhao (2022) to Agarwal et al. (2020). Key takeaways include the critical importance of model selection, feature engineering, and data preprocessing in enhancing predictive accuracy. Innovations such as model deployment via Flask and the use of GBM for analyzing variable impact demonstrate the field's progress towards practical financial applications. This body of research emphasizes the shift towards automating credit assessment processes, aiming for efficiency, fairness, and transparency. The insights gained pave the way for future research, suggesting further exploration of sophisticated models and advanced preprocessing techniques to refine credit approval predictions, reflecting ML's integral role in evolving the financial industry's decision-making frameworks.

3. DATASET

3.1 Data Collection

The dataset employed for this project, known as the "Credit Approval" dataset, originates from the UCIML Repository. This repository serves as a valuable resource within the ML community, offering a collection of databases, domain theories, and data generators specifically designed for empirical analysis of ML algorithms. The dataset has been anonymised to ensure the confidentiality and privacy of all attributes and values. Initially, the dataset comprises 690 instances and encompasses 16 variables labelled A1 through A16. These variables capture various characteristics associated with credit card applications. The first 15 variables provide insights into different aspects of the applications, such as income, employment status, and debt levels. The 16th variable serves as a crucial indicator of the application outcome, denoted by either a positive symbol "+" for accepted applications or a negative symbol "-" for rejected ones. It's interesting to note that the multivariate dataset reflects the diverse nature of the credit approval process by combining both numerical and categorical variables. The first look at the dataset is given below (Figure 1):

	A1	A2	A3	A4	A5	A6	A7	A8	A9	A10	A11	A12	A13	A14	A15	A16
0	b	30.83	0.000	u	g	w	v	1.250	t	t	1	f	g	202.0	0	+
1	a	58.67	4.460	u	g	q	h	3.040	t	t	6	f	g	43.0	560	+
2	a	24.50	0.500	u	g	q	h	1.500	t	f	0	f	g	280.0	824	+
3	b	27.83	1.540	u	g	w	v	3.750	t	t	5	t	g	100.0	3	+
4	b	20.17	5.625	u	g	w	v	1.710	t	f	0	f	s	120.0	0	+
5	b	32.08	4.000	u	g	m	v	2.500	t	f	0	t	g	360.0	0	+
6	b	33.17	1.040	u	g	r	h	6.500	t	f	0	t	g	164.0	31285	+
7	a	22.92	11.585	u	g	cc	v	0.040	t	f	0	f	g	80.0	1349	+
8	b	54.42	0.500	y	p	k	h	3.960	t	f	0	f	g	180.0	314	+
9	b	42.50	4.915	y	p	w	v	3.165	t	f	0	t	g	52.0	1442	+

Figure 1 : Credit Approval Dataset

The dataset was obtained directly from its primary source, where it underwent anonymization to ensure compliance with data privacy regulations and to uphold the confidentiality of sensitive information. In order to facilitate a deeper understanding and ease of interpretation, I decided to rename the variables with more meaningful descriptors. This renaming process was guided by a thorough review of a comprehensive blog (rstudio-pubs-static.s3.amazonaws.com, n.d.) that provided extensive insights into the characteristics of each feature. By adopting these new variable names (such as A1 representing Gender, A2 representing Age and so on), I aimed to enhance clarity and coherence in the analysis. Below, I provide a detailed mapping of the

original variable names to their newly assigned descriptors, which will serve as a valuable reference throughout our exploration of the dataset:

Original Variable Name	New Variable Name	Type	Description
A1	Gender	Categorical	The applicant's gender, including whether they identify as male or female.
A2	Age	Continuous	The candidate's age.
A3	Debt	Continuous	How much debt the applicant has.
A4	MaritalStatus	Categorical	The applicant's marital status, including whether or not they are married.
A5	BankCustomerStatus	Categorical	The bank where the applicant keeps an account.
A6	EducationLevel	Categorical	The education level of the applicant.
A7	Ethnicity	Categorical	The applicant's ethnicity.
A8	YearsEmployed	Continuous	The length of time the applicant has worked.
A9	PriorDefaultHistory	Categorical	Has the applicant previously failed to meet obligations on a credit account.
A10	EmploymentStatus	Categorical	The employment status of the applicant, whether they are employed or not.
A11	CreditScore	Continuous	The credit score of the applicant.
A12	DriversLicenseStatus	Categorical	The status of the applicants driver’s license, whether they posses a license or not.
A13	Citizen	Categorical	Citizenship of the applicant.
A14	ZipCode	Continuous	In which zipcode does the applicant reside.
A15	Income	Continuous	The annual income of the applicant.
A16	ApprovalStatus	Categorical	the applicant's approval status, indicating whether or not the application was accepted.

Upon reviewing the dataset summary, it becomes evident that there are missing data points, underscoring the necessity for thorough treatment during the analysis phase. This entails employing appropriate strategies for handling missing values to prevent bias or inaccuracies in the results.

3.2 Justification for Dataset Selection

The decision to utilize the "Credit Approval" dataset from the UCI ML Repository stemmed from its alignment with the project's objectives. This dataset presents a diverse array of attributes pertinent to credit card applications, facilitating an in-depth examination of trends and patterns within the credit approval domain. Encompassing demographic details, financial indicators, and credit background information, the dataset furnishes a comprehensive array of variables for analysis.
Furthermore, the dataset's anonymization adheres to data privacy regulations, ensuring the confidentiality of sensitive information while enabling robust analysis. This anonymization addresses ethical concerns related to personal data handling, highlighting the paramount importance of ethical considerations in research pursuits. The dataset's origin from a respected repository like the UCI ML Repository adds to its credibility and reliability. This repository is highly regarded in the ML community for providing top-notch datasets that have been extensively employed for empirical analysis and the development of algorithms.

In summary, the selection of the "Credit Approval" dataset was based on its alignment with the research question, the depth of its attributes, its compliance with ethical standards, and its reputation as a reliable data source. Its applicability to investigating credit approval processes resonates with the research goals and enables valuable insights into the determinants of credit decisions.

3.3 Exploratory Data Analysis (EDA)

Before embarking on model development, I conducted exploratory data analysis (EDA) to delve into the structure and features of the dataset. Through EDA, I aimed to uncover underlying patterns, anomalies, and relationships within the dataset. This process laid the foundation for informed decision-making in subsequent modelling stages. Moreover, EDA enabled me to identify potential preprocessing tasks, such as data cleaning, transformation, and feature engineering, essential for dataset preparation. Overall, EDA played a pivotal role as a preliminary step in comprehending the dataset's nuances and guiding subsequent analytical endeavours.

The distribution of credit application outcomes within the dataset is displayed in Figure 2, indicating a slight preference for denials over approvals, with 383 (or 55.5%) of the total 690 applications being rejected, against 307 (or 44.5%) that were approved. This distribution, tipping slightly towards rejections, mirrors the rigorous and discerning nature of credit evaluation in practice, capturing the breadth of applicant circumstances present within the data. The relatively even distribution is pivotal, fostering an environment where ML models can be crafted and assessed without an inherent lean towards either verdict. It provides a vital representation of actual credit approval conditions, characterized by complexity and diversity. The balance evident in this dataset is a key asset, crucial for ensuring that predictive models remain impartial and are equipped with the capacity to reflect the subtleties of real-world credit assessments.

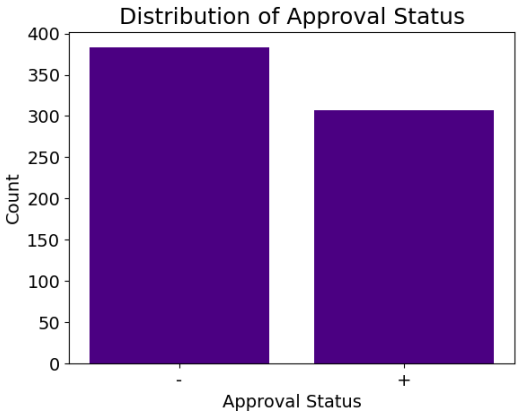

Figure 2 : Distribution of Approval Status

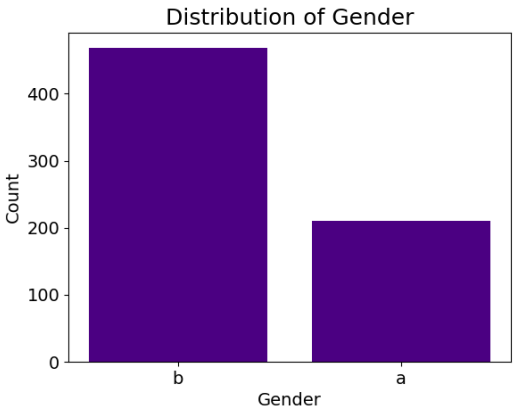

Figure 3 : Distribution of Gender

Figure 3 showcases the gender distribution among credit card applicants, displaying a disproportionate number of applications between two categories, 'b' and 'a'. Category 'b' accounts for a substantial majority, with 480 applicants or roughly 69.57% of the total, while category 'a' is represented by 210 applicants, constituting about 30.43%. This disparity may reflect broader societal trends such as differential access to financial resources, varying levels of financial literacy, or distinct attitudes towards credit and risk, suggesting that gender could play a nuanced role in the economics of credit applications. For credit issuers, these figures underscore the importance of nuanced, gender-inclusive financial products and equitable assessment strategies to ensure that access to credit is unbiased and reflective of a diverse population. Such demographic insights are crucial for the development of a fair credit system that recognizes and adapts to the unique financial realities presented by different gender groups, driving the financial sector towards more personalized and inclusive services.

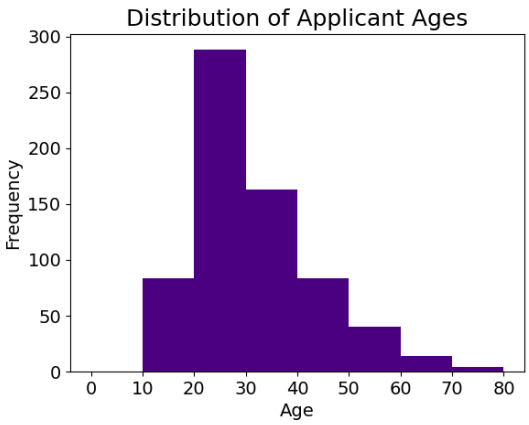

Figure 4 : Distribution of Age

The histogram in Figure 4 presents a compelling visualization of the age distribution among credit card applicants, with a significant skew toward the younger adult cohort. A substantial 41.3% of applicants are aged between 20 and 29, indicating a period in life where financial

support in the form of credit is perhaps most sought after, coinciding with many life-defining moments such as graduation, first-time employment, or the onset of independent living. The decrease in applicant numbers with advancing age is gradual yet pronounced, with those in their thirties representing 25.36%, and a further drop is seen in the subsequent decades. Notably, the 50-59 age group constitutes only 5.8% of applicants, while the 70-79 group registers barely 0.58%, underscoring a trend where credit applications diminish as individuals approach and enter retirement, a time when incomes typically become fixed and the impetus for new credit diminishes. This distribution provides valuable insight, suggesting that financial behavior and needs evolve distinctly with age. Credit companies can glean from this data that while younger individuals may be prime targets for credit outreach due to their higher propensity to seek credit, the needs and motivations of older customers, who may prioritize debt reduction or have different financial security, differ significantly. This understanding is crucial for designing age-appropriate credit products and services that cater effectively to the full spectrum of financial life stages.

Figure 5 presents a bar chart categorizing bank customers who have applied for a credit card, with a visually striking majority of 525 applicants, or 76.09% of the total, classified under 'g'. This group likely includes the bank's primary customers, possibly reflecting a more stable financial history or loyalty to the bank, which could influence their likelihood of seeking and obtaining credit. The 'p' category, accounting for 163 applicants or 23.62%, suggests a less predominant but still notable portion of the bank's clientele, potentially comprising newer members or those with a different type of banking service. Intriguingly, the 'gg' category is the smallest, with only 2 applicants, or 0.29%, indicating it is an exceptional or infrequently encountered customer classification within the bank's business model. This distribution is telling; it not only quantifies the demand for credit cards across different customer categories but also possibly correlates with the broader financial engagement and behavior patterns of the bank's clientele. These insights could guide financial institutions in developing strategic approaches to customer relations and product offerings, ensuring that their services are tailored to meet the predominant demands of the 'g' category while also catering to the specific needs of the less represented 'p' and 'gg' categories.

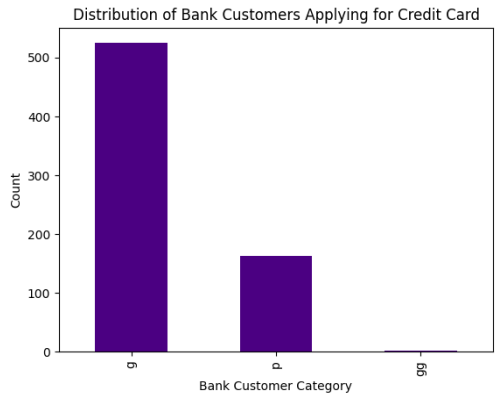

Figure 5 : Distribution of Bank Customer

Figure 6 displays the employment history lengths of individuals applying for credit cards, with the data overwhelmingly skewed towards those in the earlier stages of their working life. A vast 87.10% have been employed for 0-5 years, suggesting that a significant number of applicants are likely to be young professionals or those who have recently entered the workforce. Only a small fraction, 9.13%, have a work history of 5-10 years. Those with 10-15 years and 15-20 years of employment experience are even less common, at 2.75% and 0.87% respectively, while applicants with 20-25 years of employment are conspicuously absent from the dataset. This trend indicates that credit card services are mostly sought by those at the beginning of their employment journeys, perhaps at a life stage where establishing credit is a stepping stone to achieving financial milestones, such as purchasing a home or investing in education. The absence of longer employment histories could reflect a decrease in the need for new credit lines as individuals progress in their careers and potentially become more financially stable. This pattern is essential for credit institutions to consider, as it may influence not only

the risk profiles of potential customers but also the types of credit products and services that are in demand.

The histogram in Figure 7 indicates that the vast majority of credit card applicants fall within the lower income category, with an income of 10,000 or less, accounting for 98.26% of the total applicants in the dataset. This substantial proportion suggests that credit cards are primarily sought by those earning less, potentially to manage essential expenses or to maintain financial flexibility. The stark decrease in application numbers beyond this income range points to a lesser need or a different approach to credit among higher earners. It appears that credit cards are a more critical financial tool for those with limited income, possibly to cover day-to-day costs or unexpected expenses. For those with higher incomes, credit cards may be less of

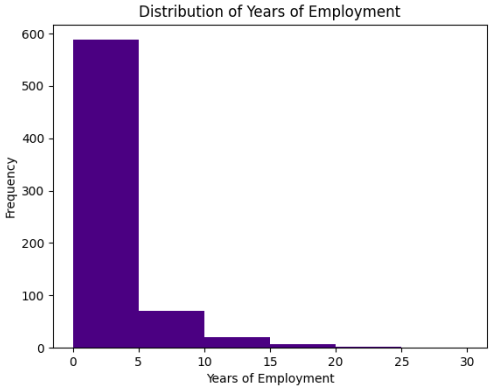

Figure 6 : Distribution of Years Employed

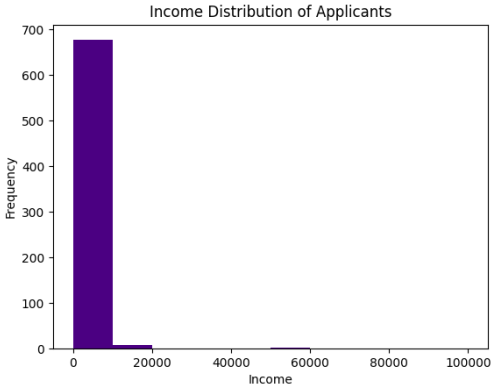

Figure 7 : Distribution of Income

a necessity or used differently, perhaps for benefits like rewards programs rather than for essential spending. These observations are crucial for credit issuers, highlighting the importance of providing financial options that support lower-income groups, while ensuring that the credit offered aligns with their ability to pay, to avoid leading them into debt.

During the exploratory data analysis, univariate assessments provided a wealth of knowledge on various aspects of our dataset, using histograms and bar charts to delve into attributes such as approval rates, gender distribution, age distribution, types of bank customers, employment longevity, and income ranges of credit card applicants. These graphical explorations highlighted distinct trends and characteristics inherent to the applicant pool, pinpointing specific traits that could influence the process of granting credit. Following this initial analysis, a bivariate exploration was undertaken to investigate how these individual features might correlate with the likelihood of an applicant receiving credit, offering further insight into the determinants of credit issuance.

The bar chart in Figure 8, upon closer inspection, does not substantiate a marked correlation between gender and credit card approval. Despite the apparent difference in approval rates—with about 69% of category 'a' and roughly 36% of category 'b' applicants receiving approvals—the data does not conclusively point to gender as a determining factor in the allocation of credit cards. The denial rates, sitting at 56% for category 'b' and 44% for category 'a', further imply that gender does not play a pivotal role in credit decisions. This suggests that credit issuers consider a variety of other elements, beyond gender, when evaluating applications. Such a finding supports the stance that lending practices are neutral to gender, focusing instead on a comprehensive assessment of each applicant's financial standing and history. It reinforces the complexity of credit evaluation and the necessity for potential

borrowers to understand that a multitude of factors, rather than gender alone, influence approval outcomes.

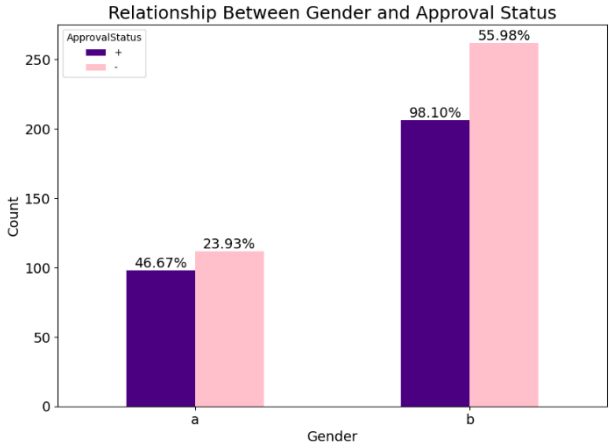

Figure 8 : Gender vs Approval Status

The bar chart in Figure 9 examining the link between education levels and credit approval outcomes presents a nuanced picture, where the educational background does not emerge as a decisive factor influencing the likelihood of obtaining credit. Variances are evident across educational groups, with certain levels like 'aa' experiencing a higher rate of denials, while others such as 'c' display a more even distribution between approvals and rejections. The lack of a clear, consistent pattern across the spectrum of educational qualifications suggests that while education is a component in the decision-making matrix, it does not singularly sway the outcome of credit applications. This inferred absence of a strong educational bias indicates a lending process that likely incorporates a more comprehensive evaluation of applicant profiles, taking into account a broader array of financial indicators and personal circumstances. The implication for credit institutions is the reinforcement of a lending paradigm that transcends educational achievement, ensuring a more equitable assessment of potential borrowers. For individuals, this means that educational attainment, albeit important, is not the sole determinant of creditworthiness, reinforcing the multifaceted nature of financial credibility assessments.

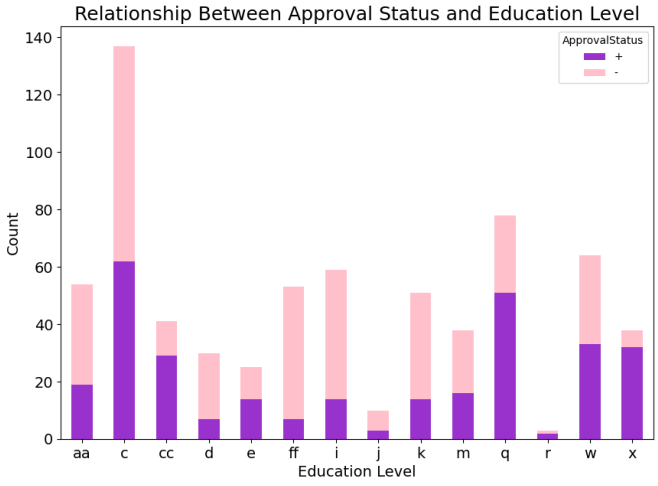

Figure 9 : Education Level vs Approval Status

The bar chart in Figure 10 vividly depicts the impact of prior defaults on the likelihood of credit approval, with the pink bars indicating a high approval count of 284 for applicants without a history of default and a comparatively low denial count of 77. In stark contrast, the purple bars illustrate that applicants with a prior default have faced a substantial number of denials, totaling 306, while only a small fraction, 23 in total, have been approved. This significant contrast not only confirms the strong correlation between a clean financial history and credit approval rates but also highlights the challenges faced by those with past defaults in securing new credit. The data underscores the enduring influence of past financial conduct on future credit opportunities, suggesting that lenders place considerable emphasis on an individual's financial track record as a reliable indicator of their future financial reliability. For potential borrowers, this chart is

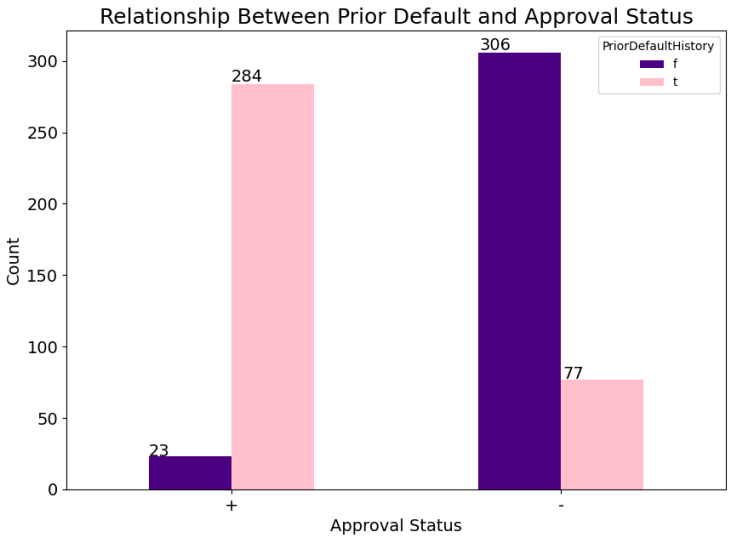

Figure 10 : Prior Default vs Approval Status

a potent reminder of the importance of avoiding financial delinquencies, whereas for financial institutions, it underscores the merit of incorporating detailed credit histories into their risk assessment models to predict creditworthiness effectively.

Credit score emerges as a pivotal factor in the approval of credit card applications, as demonstrated by the provided bar chart in Figure 11 which clearly illustrates a significant

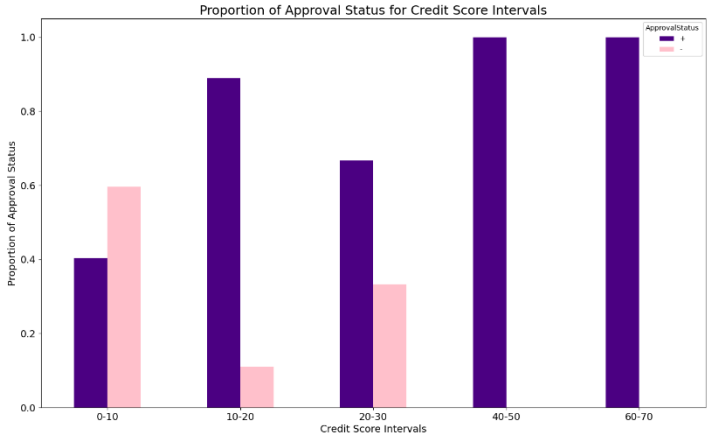

Figure 11 : Credit Score vs Approval Status

relationship between credit score intervals and approval rates. The visualization reveals a distinct trend wherein applicants with higher credit scores experience a substantially increased probability of approval, while those with lower scores face a corresponding decrease in approval likelihood. This trend aligns with standard lending protocols where credit score is utilized as a key indicator of financial health and risk. Notably, the plot exposes an intriguing nuance in the 10-20 credit score interval where approval rates are higher than those in the adjacent lower interval, suggesting the existence of a threshold effect within credit scoring that warrants further investigation. The lack of data for certain intervals, such as between 30-40 and beyond 70, may pose a limitation to the comprehensiveness of this analysis, indicating areas for future detailed exploration. Overall, these findings underscore the credit score as a crucial determinant in the credit approval process, reinforcing its role as a barometer of creditworthiness and spotlighting the importance for individuals to maintain robust credit histories for better financial opportunities.

The pie charts in Figure 12 serve as a visual exploration into the influence of driver's license ownership on credit card approval rates, revealing a minor differential in outcomes that suggests the presence of a driver's license is not a significant determinant in the approval process. Specifically, the approval rate for individuals without a driver's license stands at 43%, only slightly lower than the 46% approval rate for license holders, pointing to the fact that the credit card issuers likely do not weigh the possession of a driver's license heavily in their decision-making calculus. Instead, these subtle disparities in approval percentages direct our attention toward other, more impactful financial factors that are traditionally associated with fiscal responsibility. These factors could include an individual's track record with debt repayment, the stability and adequacy of their income concerning existing debts, and the overall credit history, which collectively paint a more telling picture of a person's financial health. In practical terms, for those seeking credit card approvals, prioritizing the maintenance of a strong credit history, minimizing debt loads, and ensuring consistent income flow are likely to bear more fruit in their quest for credit approval than the mere possession of a driver's license. This observation aligns with broader financial industry practices where financial entities emphasize concrete economic indicators over personal identification attributes when evaluating creditworthiness.

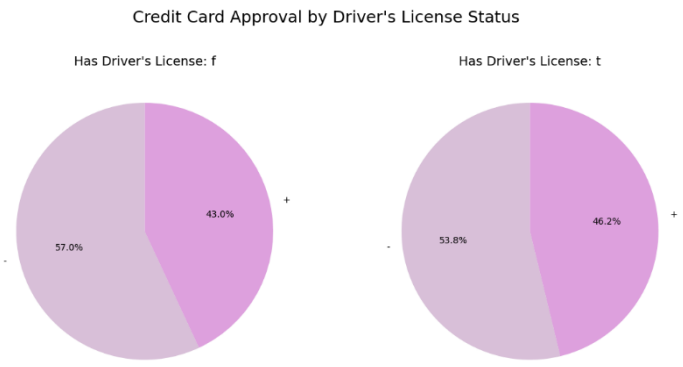

Figure 12 : Driver's License vs Approval Status

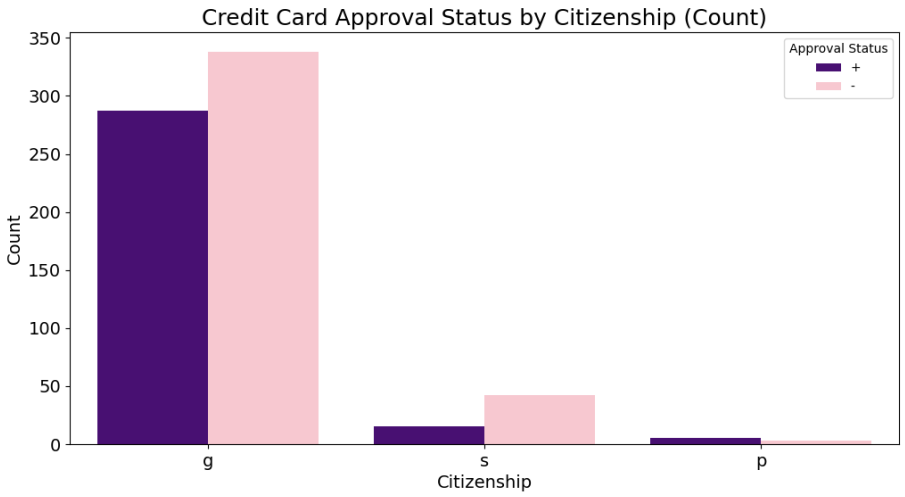

Figure 13 : Citizenship vs Approval Status

The provided bar chart (Figure 13) delineates the count of approved and denied credit card applications across citizenship classifications 'g', 's', and 'p'. The 'g' category predominates in terms of application frequency, with a substantial lead in both acceptances and rejections, implying that this group is the primary segment engaging with credit card services. However, this dominance does not necessarily correlate with a higher likelihood of approval within the group. The 's' and 'p' categories, while smaller in number, show a similar ratio of approved to denied applications, which suggests that citizenship, on its own, may not be a critical determinant in the credit card approval process. The data suggests only a marginal difference in approval percentages among the categories, pointing to a potentially minimal role of citizenship in influencing approval outcomes. Yet, this is a preliminary observation that doesn't account for other possibly influential factors or the actual proportion of approvals within each group. For a conclusive analysis, a deeper dive incorporating more nuanced statistical methods would be essential. This surface-level view indicates an absence of a stark connection between citizenship status and the rate of credit card approvals.

The pair plot in Figure 14 provides a detailed visualization of the distribution of individual numerical variables, such as 'Age', 'Debt', 'YearsEmployed', and 'Income', all showing a trend toward lower values with a small number of high-value outliers, particularly noticeable in the 'Income' data. The relationship between different variables displayed in the scatter plots reveals dense clustering around lower values, particularly for 'Debt', while 'CreditScore' shows a broader spread across its range. The absence of distinct linear patterns in these plots indicates that the connections between the variables might be complex or nonlinear, not easily discernible through visual inspection alone. This visualization emphasizes the necessity for advanced statistical analysis to validate any inferred relationships and the consideration of intricate model features that can better encapsulate these dynamics for the purpose of predicting credit card approvals. The observations from this pair plot hint at the utility of sophisticated feature engineering and the potential need for complex models or transformation of features to uncover and leverage the subtle interactions present in the dataset for predictive modeling.

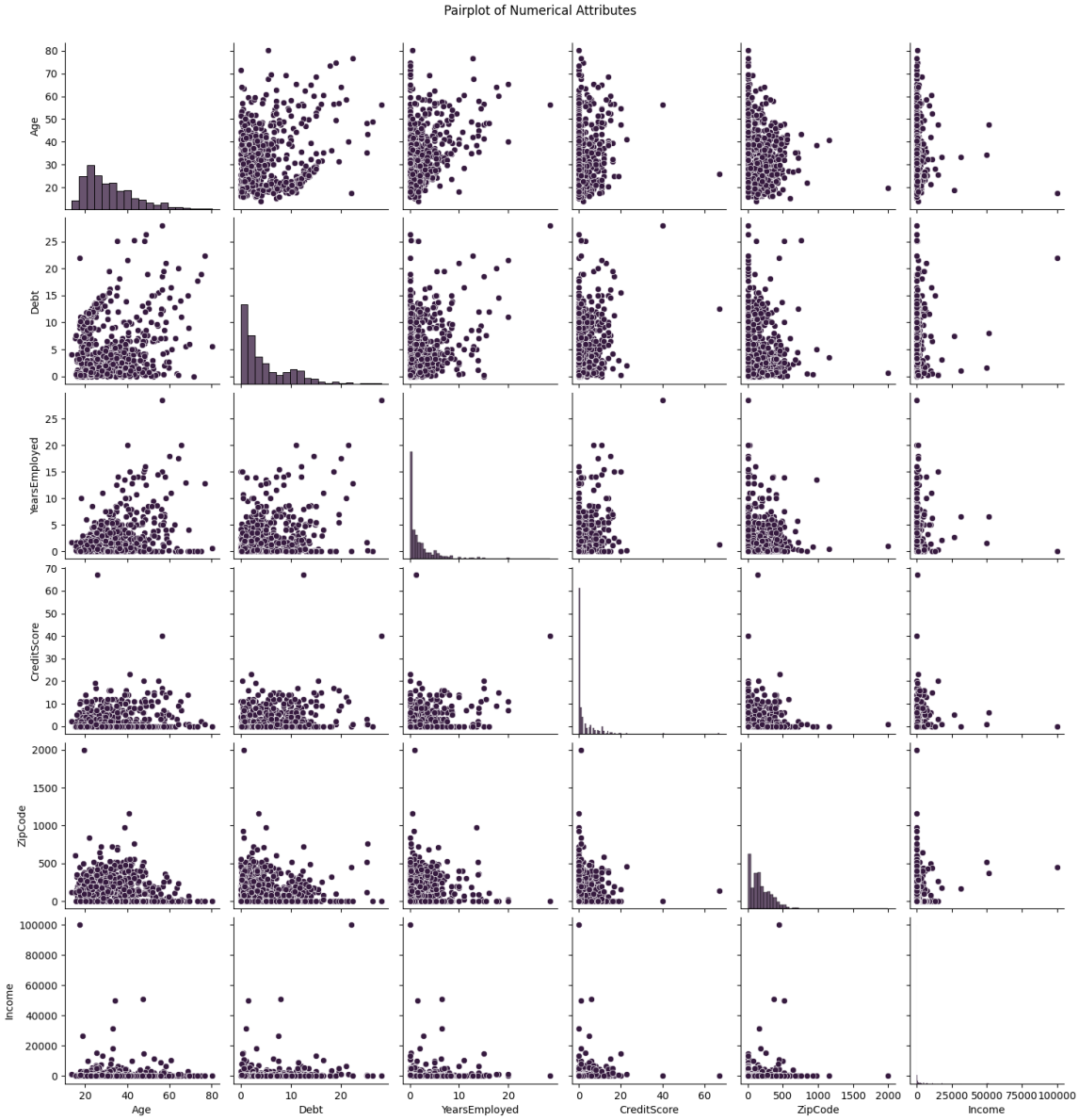

Figure 14 : Correlation Plot

The exploratory data analysis has underscored the significance of specific variables and has also shed light on the intricate interdependencies among them, suggesting that sophisticated modeling approaches may be necessary to unravel these complexities. The insights derived from this analysis provide a solid foundation for the next steps of data preprocessing and feature engineering, which are instrumental in developing potent predictive models. The knowledge acquired through EDA will steer the choice of models and assist in deciphering the outcomes of the models in relation to predicting credit card approvals.

3.4 Ethical Considerations

Ethical guidelines have been carefully integrated into this research from the outset, emphasizing the importance of privacy and the confidentiality of the data involved. Efforts to protect sensitive information have been thorough, with a commitment to handling data in a manner that respects ethical norms around its use. By placing ethical considerations at the

forefront of the research process, this study not only ensures the protection of individual privacy but also upholds the principles of transparency and responsibility in data science. This commitment to ethical practices throughout the research helps to ensure that the results are trustworthy and ethically responsible, fostering confidence in the integrity of the research findings.

4. METHODOLOGY

4.1 Data Preprocessing

4.1.1 Missing Values

The initial task was to assess the dataset for missing values. An examination revealed that several columns contained missing data, with some having up to 13 missing values. To gain a clearer understanding of the extent and distribution of these gaps, I created a bar plot (Figure 15) , which visually depicted the number of missing values across each column. This visualization made it easier to determine which features required imputation.

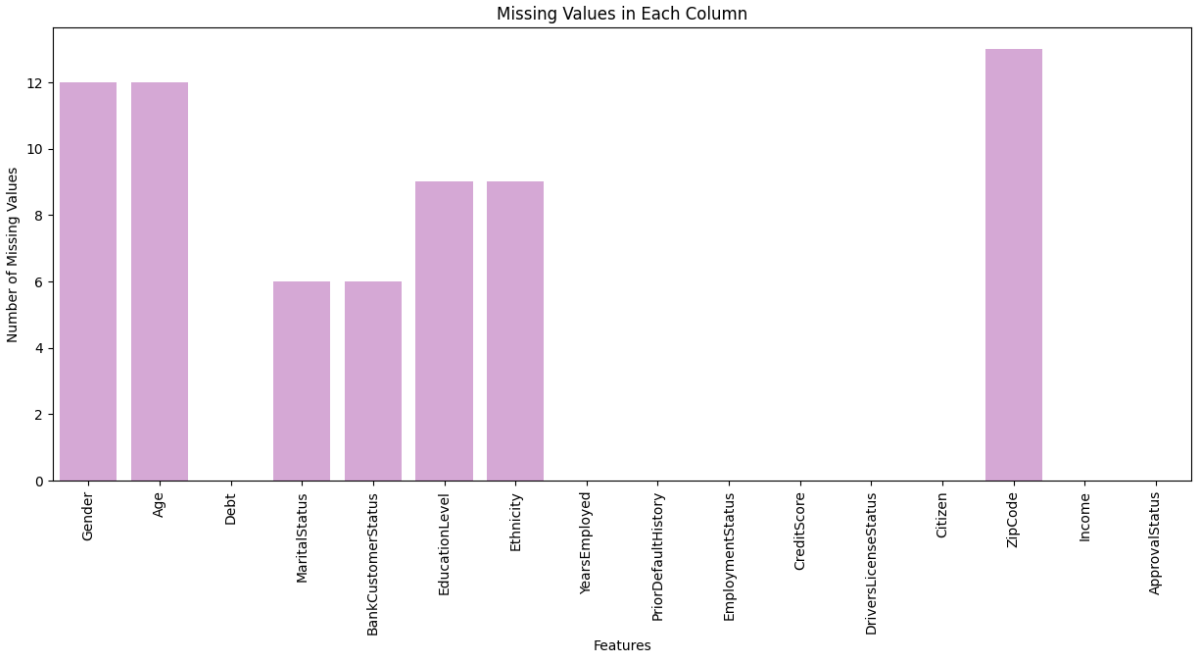

Figure 15 : Missing Values Plot

Once the presence of missing values was confirmed, I used different imputation techniques based on the data type to address these gaps:

For Numerical Columns : For columns containing numerical data, such as Age and ZipCode, I substituted the mean of each corresponding column for any missing values. This method ensured that the imputed values were consistent with the central tendency of the data, thereby maintaining the overall distribution and minimizing distortions.

For Categorical Columns : For categorical features like Gender and MaritalStatus, I used the mode—the most frequently occurring value—to fill in the missing data. This approach preserved the inherent characteristics of these features without introducing bias from non-representative values.

After completing the imputation process, a final check showed that all missing values had been appropriately handled, resulting in a complete dataset ready for further processing. This step was pivotal to ensure the robustness of the subsequent analysis and to prevent errors or biases that could affect the performance of ML models.

4.1.2 Label Encoding

After addressing missing values, the next stage in data preprocessing involved converting categorical data into a format compatible with ML models, which generally require numerical input. This was achieved through the use of label encoding, a method that gives each category in a categorical feature a distinct numerical value.

I initialized '**LabelEncoder'** from the '**sklearn.preprocessing'** library, a tool designed to convert categorical data into numerical labels. This step sets the stage for the transformation process. I used '**LabelEncoder**' to transform each categorical column in the dataset into its corresponding numeric representation. This involved iterating through all columns with 'object' data types and replacing each unique category with a distinct integer. By converting qualitative information into quantitative values, I ensured compatibility with ML models.

After applying label encoding, the dataset contained only numerical values, making it suitable for ML algorithms. The initial few rows of the transformed dataset demonstrated the successful application of this technique, showing categorical data replaced by corresponding integers as shown in Figure 16.

	Gender	Age	Debt	MaritalStatus	BankCustomerStatus	EducationLevel	Ethnicity	YearsEmployed	PriorDefaultHistory	EmploymentStatus	CreditScore	DriversLicenseStatus	Citizen	ZipCode	Income	ApprovalStatus
0	1	30.83	0.000	1	0	12	7	1.25	1	1	1	0	0	202.0	0	0
1	0	58.67	4.460	1	0	10	3	3.04	1	1	6	0	0	43.0	560	0
2	0	24.50	0.500	1	0	10	3	1.50	1	0	0	0	0	280.0	824	0
3	1	27.83	1.540	1	0	12	7	3.75	1	1	5	1	0	100.0	3	0
4	1	20.17	5.625	1	0	12	7	1.71	1	0	0	0	2	120.0	0	0

Figure 16 : Dataset after label encoding

Label encoding simplified the dataset, facilitating smooth integration with various ML models. This transformation provided a solid foundation for further steps in model development and evaluation, ensuring the data was in the correct format for analysis. It also reduced the likelihood of errors during model training, contributing to the robustness and accuracy of the project.

4.1.3 Handling Outliers

To maintain data integrity and avoid such distortions, I tackled the issue of outliers in the numerical features of the dataset. The initial step was identifying which data points were outliers. This was accomplished using the interquartile range (IQR), a common statistical method for determining the range of normal values. To manage the outliers, I implemented a capping strategy. This involved adjusting extreme values to fall within a predefined range based on the IQR. The goal was to limit the influence of outliers while preserving the essential characteristics of the dataset.

Before treating the outliers, the dataset contained 303 extreme values, particularly in numerical features like "Income" and "ZipCode." These outliers could significantly impact the accuracy of analyses and the performance of predictive models. After capping the outliers, the dataset became more stable, with all data points falling within acceptable bounds. This approach reduced the potential bias caused by extreme values, resulting in a more reliable dataset for analysis and model training.

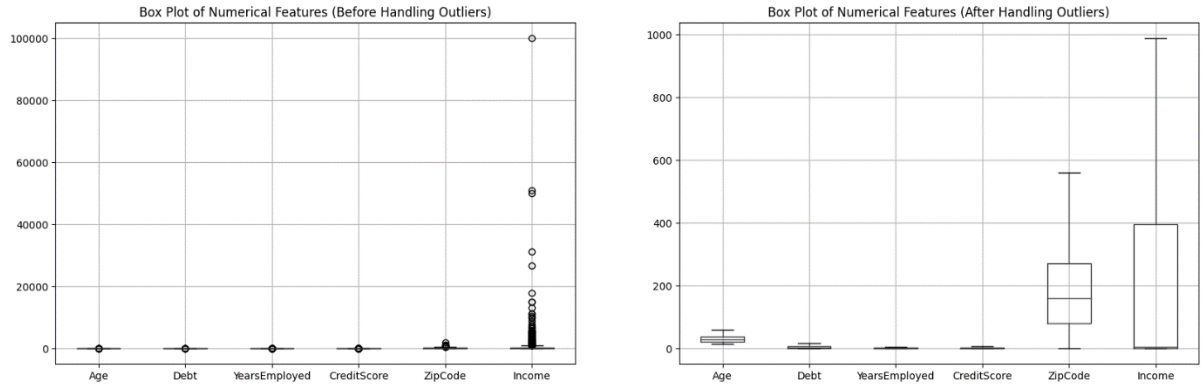

Figure 17 : Outlier Handling

To demonstrate the impact of outlier handling, I created box plots (Figure 17) that show the distribution of numerical features before and after addressing outliers. In the initial box plot, outliers are evident, with some values extending far beyond the typical range. This indicates the risk of distortion if these extreme values are not managed. In the subsequent box plot, which reflects the data after outlier capping, the distribution appears more controlled. The whiskers, representing the range of normal values, suggest that outliers were successfully managed, bringing them within acceptable boundaries.

By handling outliers in this manner, I ensured that the dataset was more balanced and suitable to accurate analysis and model development. This approach helps to minimize distortions and provides a more reliable basis for building ML models to predict credit card approval outcomes.

4.1.4 Data Splitting for Model Training and Testing

The next step involved dividing the dataset (training and testing sets) after it had been prepared by managing outliers and categorical data. This division is essential for training ML models and evaluating their performance. By separating the data, you can train models on one subset and validate them on another, ensuring a balanced approach to model development and preventing overfitting.

To begin, I defined the features and the target variable. The features comprised all columns except "ApprovalStatus," which served as the target variable, indicating whether a credit card application was approved or denied. I then used the '**train_test_split**' function from scikit-learn to create the train and test sets. The training set contained 70% of the data, providing ample information for the models to learn from, while the testing set comprised the remaining 30%, allowing for model validation. This 70/30 split is common in ML because it offers a good balance between training the model and testing its generalization capabilities.

To ensure consistency and reproducibility, I set a fixed random seed (42) for the data split. This ensures that the same random seed yields identical results across multiple runs, making the process more reliable. The resulting sets had the following shapes: 483 samples with 15 features each made up the training set (X_train), and 207 samples with the same 15 features made up the testing set (X_test). There were 483 samples in the target variable (y_train) of the training set, and 207 samples in the target variable (y_test) of the testing set.

This split provides a balanced approach to training and testing, allowing for robust model evaluation. I can make sure the model doesn't just remember the training data but can also

generalise to new, unseen data by training it on a larger dataset and validating it on a smaller, distinct set.

4.1.5 Scaling Data

To guarantee that numerical features are on a consistent scale, scaling is a crucial step in the preprocessing of data. By doing this, the ML model is kept from being overpowered by features with higher values. To normalise the data, I used MinMaxScaler, which converts numerical features to a standard range, usually between 0 and 1.

To ascertain the maximum and minimum values for each feature, the scaler was fitted to the training set. Used the scaler to fit the training and testing sets of data in order to align all numerical features within the same range. In order to maintain the relative relationships between data points, MinMaxScaler sets each feature to have a minimum value of 0 and a maximum value of 1. After scaling, I converted the transformed data back into DataFrames for easier handling and further analysis.

The outcome was a dataset with all numerical features scaled between 0 and 1, providing a consistent and balanced foundation for ML models. This uniform scaling helps improve the accuracy of the models and ensures they generalize better when tested on new data.

4.2 Model Selection and Training

4.2.1 Logistic Regression

The LR model was chosen to predict credit card approval outcomes. I used the scaled training data to train the model first, and then I used it to predict the results for the testing data. By contrasting the predictions with the testing set's actual outcomes, the accuracy of the model was assessed.

A confusion matrix was used to visualize how accurately the model predicted credit card approvals. It showed where the model correctly identified outcomes and where it made errors. To enhance the model, I used GridSearchCV to perform hyperparameter tuning, optimizing the regularization parameter (C), the penalty term, and the solver. This fine-tuning led to a more accurate model. For further refinement, I focused on features with the highest correlation to "ApprovalStatus" and trained the LR model with these selected features. The revised model achieved improved performance on both the training and testing sets.

I determined the F1-score, precision, recall, and the Receiver Operating Characteristic - Area Under the Curve (ROC AUC) score in addition to other metrics to assess the model's efficacy. These measures gave an extensive picture of the model's accuracy and dependability in predicting credit card approvals.

With these approaches, the LR model proved effective in predicting credit card approval outcomes, demonstrating a solid balance between accuracy and generalization. This thorough process of model selection, training, and tuning ensured the creation of a reliable predictive model.

4.2.2 Decision Tree

To predict credit card approval outcomes, next I trained a DT model, known for its simplicity and adaptability. Fitting the model on the scaled training data and testing the model's

predictions on the scaled test data were the first steps in the training process. By contrasting predictions with the test set's actual results, the initial accuracy was determined.

To optimize the model's performance, I used GridSearchCV to find the best hyperparameters, including tree depth and minimum samples for splits and leaf nodes. This fine-tuning improved the model's accuracy and robustness. To evaluate the DT's accuracy, I used a confusion matrix, which visualized where the model correctly predicted credit card approvals and where it made errors. By examining the feature importance, I identified the most significant features and retrained the model using these key attributes, resulting in enhanced performance.

To get a more complete picture of the efficacy of the DT, I also computed precision, recall, F1-score, and ROC AUC score in addition to accuracy. Additionally, I used k-fold cross-validation to make sure the model remained reliable for various data splits.

Overall, the improved DT model showed increased precision and dependability in predicting the results of credit card approval, demonstrating the importance of feature selection and hyperparameter adjustment for this kind of ML task.

4.2.3 Random Forest

I decided to use the RF model, a robust ensemble approach that mixes several decision trees to increase precision, to forecast credit card approval outcomes. Training the model on the scaled training set was the first step. I used it to make predictions on the scaled test data after it had finished training, and I assessed its accuracy by contrasting the results with the actual test set outcomes.

Critical hyperparameters, such as the number of trees, the maximum depth, and the minimum number of samples for internal splits and leaf nodes, were adjusted using GridSearchCV to optimise the model. This grid search helped identify the optimal settings for better performance. RF provides a measure of feature importance, allowing me to identify the most impactful features for credit card approval predictions. After pinpointing these key features, I retrained the model, which led to improved accuracy and reduced the risk of overfitting. Where the model predicted correctly and incorrectly was visually represented by the confusion matrix.

I computed metrics such as precision, recall, F1-score, and ROC AUC score to assess the model's efficacy and provide a thorough picture of its accuracy. I also used cross-validation to verify the robustness of the model and make sure it could function consistently across different data subsets.

After tuning and refining, the final RF model demonstrated improved accuracy and reliability in predicting credit card approval outcomes. This process underscores the effectiveness of ensemble learning in building reliable predictive models.

5. RESULTS

The study's findings evaluate the precision and dependability of three machine learning models—LR, DT, and RF—in terms of forecasting the results of credit card approval. Metrics like accuracy, precision, recall, F1-score, and ROC AUC score are used to evaluate each model's efficacy and provide a thorough understanding of its performance.

5.1 Confusion Matrices

A visual summary of each model's accurate and inaccurate predictions is given by confusion matrices. They highlight areas where models perform well and those where they misclassify results by classifying predictions into true positives (TP), true negatives (TN), false positives (FP), and false negatives (FN). This section discusses the confusion matrices for LR, DT, and RF, revealing their accuracy and error distribution.

For the LR model, the confusion matrix (Figure 18) shows 81 TNs and 95 TPs, indicating generally high accuracy. However, the model also has 18 FPs and 13 FNs, reflecting some prediction errors. The matrix's grid-based structure, with colour-coded cells, makes it easy to identify where the model performed accurately and where it did not.

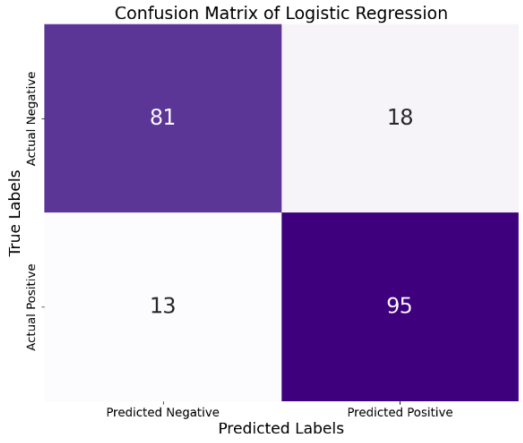

Figure 18 : Confusion Matrix of Logistic Regression

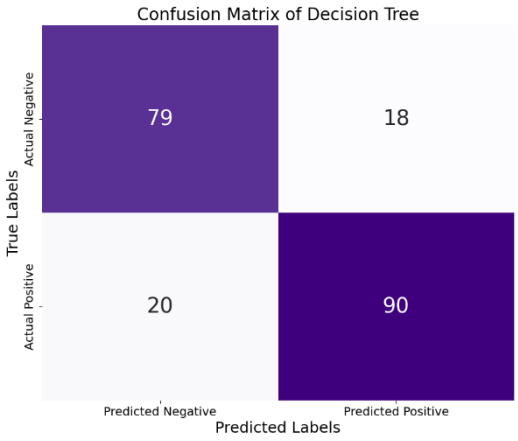

Figure 19 : Confusion Matrix of Decision Tree

The confusion matrix (Figure 19) for the DT model reveals a slightly higher error rate compared to LR. It has 79 TNs and 90 TPs, suggesting a decent balance, but also 18 FPs and 20 FNs, indicating more misclassifications. The visualization is similar to the previous model, providing a straightforward way to assess prediction accuracy.

The confusion matrix (Figure 20) for the RF model demonstrates relatively high accuracy, with 82 TNs and 95 TPs. This model has fewer FPs (15) and the same number of FNs (15) as the DT model, indicating it might be more accurate. The visual format allows for a quick assessment of model performance, emphasizing strengths and potential areas for improvement.

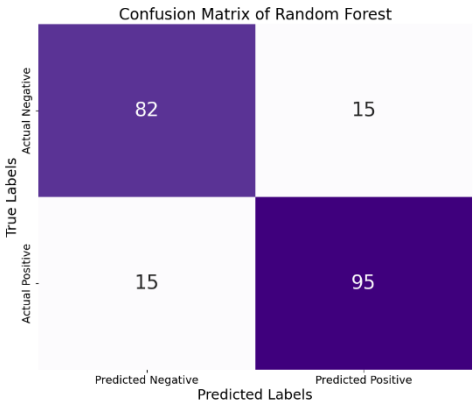

Figure 20 : Confusion Matrix of Random Forest

By showing the models' predictive accuracy and giving a visual depiction of their capacity to discern between favourable and unfavourable outcomes in credit card approval cases, these confusion matrices provide an efficient means of comparing the models.

5.2 Evaluation Metrics

To understand the performance of the ML models, several evaluation metrics were calculated: accuracy, precision, recall, F1-score, and ROC AUC score. Below are the definitions of each metric and the equations used to compute them:

- **Accuracy**: The proportion of accurately predicted events to all predicted events. It is calculated as:

$$Accuracy = \frac{TP + TN}{Total\ Samples}$$

- **Precision**: The proportion of accurately predicted positives to all true positives. The equation is:

$$Precision = \frac{TP}{TP + FP}$$

- **Recall**: The proportion of real to true positives. It is given by:

$$Recall = \frac{TP}{TP + FN}$$

- **F1-Score**: The harmonic mean of precision and recall, calculated as:

$$F1 - Score = 2 * \frac{Precision * Recall}{Precision + Recall}$$

- **ROC AUC Score**: The area under the curve of the ROC curve shows how well the model can distinguish between various classes..

Logistic Regression :

- **Training Data**: The LR model had an accuracy of 86.34%. Its precision was 89.43%, and recall was 86.18%, leading to an F1-score of 87.78%. The ROC AUC score was 86.36%.

- **Testing Data**: For the testing set, the model's accuracy was 85.02%, precision was 84.07%, and recall was 87.96%. The resulting F1-score was 85.97%, and the ROC AUC score was 84.89%.

Decision Tree :

- **Training Data**: The DT model had high accuracy at 92.13%, precision at 95.02%, and recall at 90.84%. The F1-score was 92.88%, indicating a balance between precision and recall. The ROC AUC score was 92.33%.
- **Testing Data**: The accuracy on the test set was 81.64%. Precision was 83.33%, the recall was 81.82%, and the F1-score was 82.57%. The ROC AUC score was 81.63%.

Random Forest :

- **Training Data**: The RF model had an accuracy of 97.10%, precision of 96.75%, and recall of 98.17%, resulting in an F1-score of 97.45%. The ROC AUC score was 99.72%.
- **Testing Data**: The accuracy on the test set was 85.51%. Precision was 86.36%, the recall was 86.36%, and the F1-score was 86.36%. The ROC AUC score was 90.75%.

These results demonstrate that while the DT performed well on the training set, the RF showed better generalization to the test set. LR exhibited consistent performance across training and testing datasets.

5.3 Performance Comparison

The graph in Figure 21 explores the key performance metrics of LR, DT, and RF. This

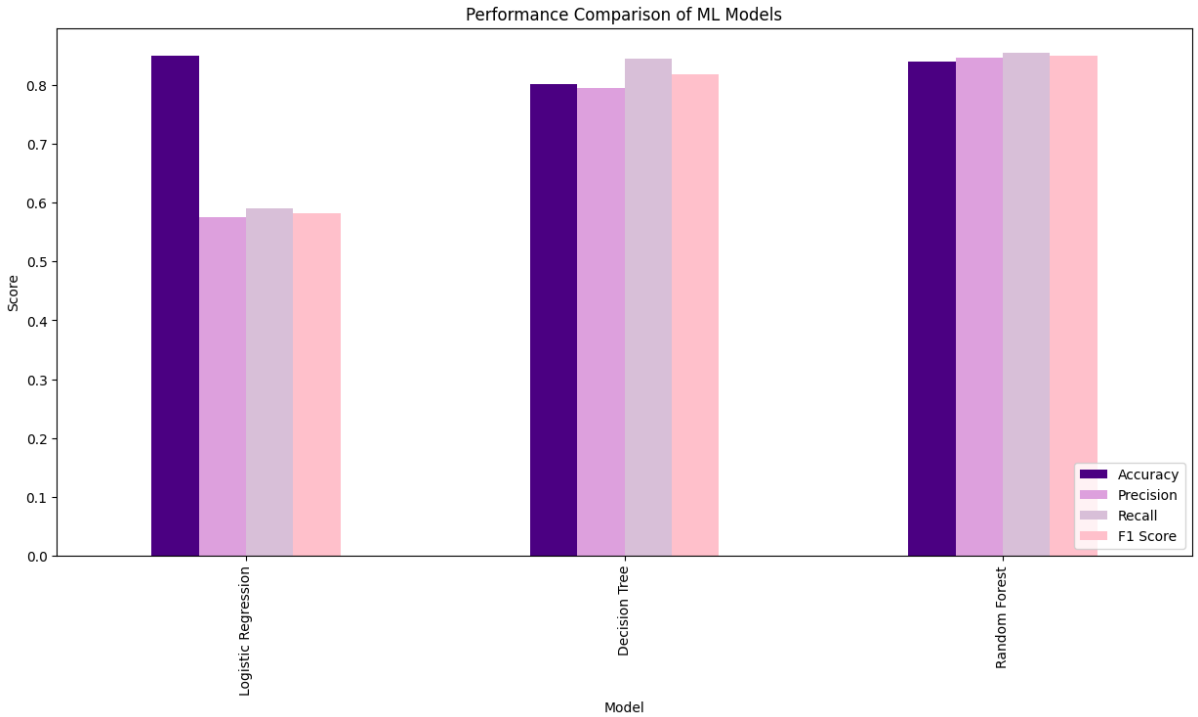

Figure 21: Performance Comparison Graph

comparison is visualized through a bar graph, where accuracy, precision, recall, and F1-score are plotted for each model, providing a clear view of their relative effectiveness. RF shows the

strongest performance across most metrics, highlighting its robustness and versatility in predicting credit card approvals. While LR offers reliability and simplicity, its metrics are slightly lower compared to RF. DT, despite its accuracy, seems prone to overfitting, indicating the need for further adjustments. This comparison suggests that while RF is the most suitable for this task, other models have their own merits, making them valuable for different contexts.

6. ANALYSIS AND DISCUSSION

This section delves into the analysis and discussion of the results derived from three ML models: LR, DT, and RF. The evaluation focused on their accuracy and overall performance, and a performance comparison graph.

6.1 Significance of Results

The results from this study showed different levels of accuracy and performance across the three ML models: LR, DT, and RF. With scores of 0.971 on the training set and 0.855 on the testing set, RF demonstrated the highest accuracy and a strong ability to generalise across different datasets. Its high recall (0.982) and F1-score (0.975) on the training set demonstrated its efficiency in recognizing positive cases, while its precision of 0.864 on the testing set highlighted its reliability in real-world applications.

LR had consistent accuracy, with 0.863 on the training set and 0.850 on the testing set, indicating a steady performance with a balanced approach to precision and recall. DT, on the other hand, displayed indications of overfitting, exhibiting a high training set accuracy of 0.921 but a low testing set accuracy of 0.816. This suggests that DT excels in identifying specific patterns in training data but has a harder time generalizing to unseen data, as reflected by its drop in F1-score from 0.929 to 0.826.

6.2 Best – Performing Model

A robust performance indicates that the RF model is suitable for diverse and complex datasets; its ensemble approach, which combines multiple decision trees, reduces overfitting and improves the model's ability to generalise to new data. The RF model was the best performer, exhibiting high accuracy and consistent evaluation metrics across training and testing datasets. While LR showed reliable accuracy, it did not perform as well as RF. Even so, it's still a valid option because of its simplicity and lower risk of overfitting. The DT model, though it achieved reasonable accuracy, showed signs of overfitting during the training phase. This means that in order to improve its resilience and generalisation to new data, it would benefit from more fine-tuning and cleaning.

6.3 Comparison with Literature

The findings from my project corroborate prior research demonstrating RF's success in predicting credit card approvals. Studies by Bansal and Punjabi (2021) and Sayjadah et al. (2018) had identified RF as a high-performing model regarding the accuracy and reliability, which is consistent with my project's results. Achieving an accuracy rate of 0.855 and a ROC AUC score of 0.907, RF proved to be a dependable choice for credit card approval prediction.

LR and DT in my project also yielded results in line with prior studies. Duan (2020) and Agarwal et al. (2020) highlighted the steady performance of LR, a characteristic reflected in my project. Therese et al. (2022) noted DT's success in credit card approval tasks, echoing my findings. However, DT’s tendency to overfit, as discussed in the literature, suggests that careful tuning is needed to avoid this issue. The significance of model selection, parameter tuning, and feature engineering in enhancing ML models for credit card approval prediction is highlighted by these findings.

6.4 Limitations of Results

One significant limitation is the overfitting observed in the DT model, indicating that it might struggle to extend to fresh or unforeseen data. Additionally, the range and variety of the dataset could influence the outcomes, creating a risk of biased predictions. It's important to keep these limitations in mind when using these models for real-world applications.

6.5 Alignment with Project Goals

Finding the best model for predicting credit card approval outcomes was the main goal of this project. The results demonstrate that both RF and LR meet this aim, offering high accuracy and consistent results. Although DT showed promise, its performance indicates that additional adjustments are necessary for it to effectively meet the project's goals.

7. CONCLUSION

This section serves to summarize the key outcomes of this project, highlight its practical uses, and suggest future steps for continued development. In this study, three ML models—LR, DT, and RF—were assessed for their effectiveness in predicting credit card approval outcomes. Among these models, RF stood out as the top performer, demonstrating consistent accuracy during both training and testing. LR also delivered reliable results, while DT showed signs of overfitting, indicating a need for additional refinement. These findings suggest that RF is well-suited for real-world applications due to its robust performance and adaptability to various datasets. LR, with its straightforward design, could be useful in contexts that require clear interpretations and minimal risk of overfitting.

7.1 Future Works

To advance the project, future enhancements could include leveraging TensorFlow to boost accuracy and explore more complex neural network structures. TensorFlow's advanced capabilities could improve model flexibility and scalability, supporting better visualization and analysis of results for transparent credit card approval predictions. Additionally, developing a website to help customers assess their credit card approval prospects and provide personalized feedback could offer significant value. This tool could identify reasons for rejection, such as low credit scores or insufficient income, and suggest ways to improve creditworthiness, potentially reducing manual review processes.

Integrating advanced ML technologies like TensorFlow with a customer-oriented website has the potential to revolutionize credit card approval systems. The improved accuracy and customized feedback would lead to more reliable predictions, offering a superior user experience. This strategy could set a new benchmark in the industry, illustrating the advantages of combining cutting-edge technology with customer-focused solutions.

8. REFERENCES

Agarwal, A., Rana, A., Gupta, K. and Verma, N. (2020). A Comparative Study and enhancement of classification techniques using Principal Component Analysis for credit card dataset. *2020 International Conference on Intelligent Engineering and Management (ICIEM)*. doi:https://doi.org/10.1109/iciem48762.2020.9160230.

Bansal, S. and Punjabi, T. (2021). *Comparison of Different Supervised Machine Learning Classifiers to Predict Credit Card Approvals*. *International Research Journal of Engineering and Technology*.

Chakravorti, S. and To, T. (2007). A theory of credit cards. *International Journal of Industrial Organization*, 25(3), pp.583–595. doi:https://doi.org/10.1016/j.ijindorg.2006.06.005.

Duan, L. (2020). Performance Evaluation and Practical Use of Supervised Data Mining Algorithms for Credit Card Approval. *2020 International Conference on Computing and Data Science (CDS)*. doi:https://doi.org/10.1109/cds49703.2020.00057.

Kibria, Md.G. and Sevkli, M. (2021). Application of Deep Learning for Credit Card Approval: A Comparison with Two Machine Learning Techniques. *International Journal of Machine Learning and Computing*, 11(4), pp.286–290. doi:https://doi.org/10.18178/ijmlc.2021.11.4.1049.

Markova, M. (2021). Credit card approval model: An application of deep neural networks. *SEVENTH INTERNATIONAL CONFERENCE ON NEW TRENDS IN THE APPLICATIONS OF DIFFERENTIAL EQUATIONS IN SCIENCES (NTADES 2020)*. doi:https://doi.org/10.1063/5.0040744.

Quinlan,J. R.. Credit Approval. UCI Machine Learning Repository. https://doi.org/10.24432/C5FS30.

rstudio-pubs-static.s3.amazonaws.com. (n.d.). *Analysis of Credit Approval Data*. [online] Available at: https://rstudio-pubs-static.s3.amazonaws.com/73039_9946de135c0a49daa7a0a9eda4a67a72.html [Accessed 29 Feb. 2024].

Sayjadah, Y., Hashem, I.A.T., Alotaibi, F. and Kasmiran, K.A. (2018). Credit Card Default Prediction using Machine Learning Techniques. *2018 Fourth International Conference on Advances in Computing, Communication & Automation (ICACCA)*. doi:https://doi.org/10.1109/icaccaf.2018.8776802.

Therese, M.J., Devi, A., Gurulakshmi, R., Sandhya, R. and Dharanyadevi, P. (2022). Credit Card Assent Using Supervised Learning. *2022 International Conference on Smart Technologies and Systems for Next Generation Computing (ICSTSN)*. doi:https://doi.org/10.1109/icstsn53084.2022.9761307.

Ursekar, K. (2020). *Predicting Credit Card Approvals*. [online] Medium. Available at: https://medium.com/@kaustubh.ursekar/predicting-credit-card-approvals-c5c30f405f92 [Accessed 6 Apr. 2024].

Zhao, Y. (2022). Credit Card Approval Predictions Using Logistic Regression, Linear SVM and Naïve Bayes Classifier. *2022 International Conference on Machine Learning and Knowledge Engineering (MLKE)*. doi:https://doi.org/10.1109/mlke55170.2022.00047.

9. APPENDICES

9.1 Extra Plot

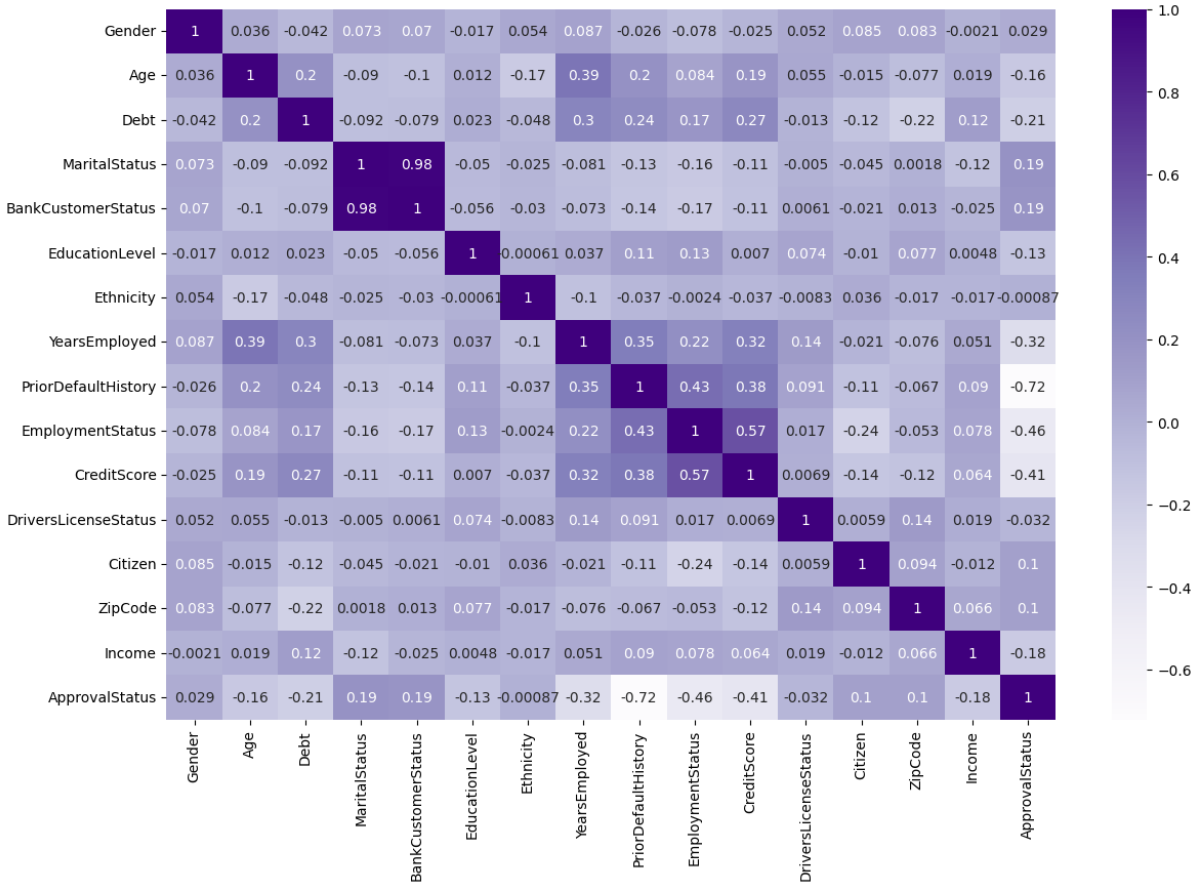

Figure 22: Correlation Matrix

9.2 Code

```
pip install ucimlrepo

from ucimlrepo import fetch_ucirepo

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

from sklearn.preprocessing import LabelEncoder
from sklearn.preprocessing import MinMaxScaler
from sklearn.model_selection import train_test_split

from sklearn.linear_model import LogisticRegression
```

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

from sklearn.metrics import accuracy_score, confusion_matrix, precision_score, recall_score,
f1_score, roc_auc_score
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import cross_val_score
from sklearn.feature_selection import SelectFromModel

# fetch dataset
credit_approval = fetch_ucirepo(id=27)

# data (as pandas dataframes)
X = credit_approval.data.features
y = credit_approval.data.targets

# metadata
print(credit_approval.metadata)

# variable information
print(credit_approval.variables)

#Displaying the feature and target variable values
print(X)
print(y)

# Reverse the order of columns in DataFrame X
X_reversed = X.iloc[:, ::-1]
print(X_reversed)

# Concatenate reversed X (features) and y (target variable) into a single DataFrame
data = pd.concat([X_reversed, y], axis=1)

#Shape of the dataframe
data.shape

#Printing the head of the dataframe
data.head(10)
```

The dataset, obtained from the UCI machine learning repository with strict data confidentiality measures, has been anonymized to protect sensitive information. However, referencing the information available in the provided blog link, we can identify the specific variable names associated with the dataset. These variables include Gender, Age, Debt, Marital Status, Banking Customer Status, Education Level, Ethnicity, Years of Employment, Prior Default History, Employment Status, Credit Score, Driver's License Status, Citizenship, Zip Code, Income, and Approval Status.

```
#Changing the varible names into meaningful values
```

```
data_cols = ['Gender', 'Age', 'Debt', 'MaritalStatus', 'BankCustomerStatus', 'EducationLevel', 'Ethnicity', 'YearsEmployed', 'PriorDefaultHistory', 'EmploymentStatus', 'CreditScore', 'DriversLicenseStatus','Citizen', 'ZipCode', 'Income', 'ApprovalStatus']
data.columns = data_cols

#Displaying the data after changing variable names
#Head
data.head(10)

#Tail
data.tail(10)

data.info()

data.describe()
```

Exploratory Data Analysis

Univariate Analysis

```
# Count the occurrences of each category in the target column
approval_status_counts = data['ApprovalStatus'].value_counts()

# Calculate the percentage of each category
total_samples = len(data)
approval_status_percentages = approval_status_counts / total_samples * 100

# Define colors for the bars
colors = '#4b0082'

# Plot the bar chart
plt.bar(approval_status_counts.index, approval_status_counts, color = colors)

# Add labels and title
plt.xlabel('Approval Status', size = 14)
plt.ylabel('Count', size = 14)
plt.title('Distribution of Approval Status', size = 18)
plt.xticks(size = 14)
plt.yticks(size = 14)
# Saving the plot image
plt.savefig('Approval Status.png')
# Show the plot
plt.show()


# Print the count and percentage of each category
for status, count, percentage in zip(approval_status_counts.index, approval_status_counts, approval_status_percentages):
    print(f"{status}: Count = {count}, Percentage = {percentage:.2f}%")
```

```
# Count the occurrences of each category in the 'Gender' column
gender_counts = data['Gender'].value_counts()

# Calculate the percentage of each category
gender_percentages = gender_counts / total_samples * 100

# Define colors for the bars
colors = '#4b0082' # Purple shade

# Plot the bar chart with specified colors
plt.bar(gender_counts.index, gender_counts, color=colors)

# Add labels and title
plt.xlabel('Gender', size = 14)
plt.ylabel('Count', size = 14)
plt.title('Distribution of Gender', size = 18)
plt.xticks(size = 14)
plt.yticks(size = 14)
# Saving the plot image
plt.savefig('Gender.png')
# Show the plot
plt.show()

# Print the count and percentage of each category
for gender, count, percentage in zip(gender_counts.index, gender_counts, gender_percentages):
    print(f"{gender}: Count = {count}, Percentage = {percentage:.2f}%")

# Define age categories
age_bins = np.arange(0, 81, 10)
age_labels = [f"{i}-{i+9}" for i in range(0, 80, 10)]  # Adjusted to have one fewer label

# Plot histogram for Age
plt.hist(data['Age'], bins=age_bins, color='#4b0082')

# Add labels and title
plt.xlabel('Age', size = 14)
plt.ylabel('Frequency', size = 14)
plt.title('Distribution of Applicant Ages', size = 18)
plt.xticks(size = 14)
plt.yticks(size = 14)
# Saving the plot image
plt.savefig('Age.png')
# Show the plot
plt.show()

# Calculate the count and percentage of each age category
age_categories = pd.cut(data['Age'], bins=age_bins, labels=age_labels, include_lowest=True)
age_counts = age_categories.value_counts()
age_percentages = age_counts / total_samples * 100
```

```
# Print the count and percentage of each age category
for age_category, count, percentage in zip(age_counts.index, age_counts, age_percentages):
    print(f"Age Category {age_category}: Count = {count}, Percentage = {percentage:.2f}%")

# Count the occurrences of each category in the BankCustomer column
bank_customer_counts = data['BankCustomerStatus'].value_counts()

# Calculate the percentage of each category
total_samples = len(data)
bank_customer_percentages = bank_customer_counts / total_samples * 100

# Plot the bar chart with purple color
bank_customer_counts.plot(kind='bar', color='#4b0082')

# Add labels and title
plt.xlabel('Bank Customer Category', size = 14)
plt.ylabel('Count', size = 14)
plt.title('Distribution of Bank Customers Applying for Credit Card', size = 18)
plt.xticks(size = 14)
plt.yticks(size = 14)
# Saving the plot image
plt.savefig('Bank Customer.png')
# Show the plot
plt.show()

# Print the count and percentage of each category
for category, count, percentage in zip(bank_customer_counts.index, bank_customer_counts,
bank_customer_percentages):
    print(f"{category}: Count = {count}, Percentage = {percentage:.2f}%")

# Define the bins for years of employment
employment_bins = [0, 5, 10, 15, 20, 25, 30]
employment_labels = ['0-5 yrs', '5-10 yrs', '10-15 yrs', '15-20 yrs', '20-25 yrs', '25-30 yrs']

# Plot the histogram
plt.hist(data['YearsEmployed'], bins=employment_bins, color='#4b0082')

# Add labels and title
plt.xlabel('Years of Employment', size = 14)
plt.ylabel('Frequency', size = 14)
plt.title('Distribution of Years of Employment', size = 18)
plt.xticks(size = 14)
plt.yticks(size = 14)
# Saving the plot image
plt.savefig('Years Employed.png')
# Show the plot
plt.show()

# Calculate the count and percentage of each category
```

```
employment_categories = pd.cut(data['YearsEmployed'], bins=employment_bins, labels=employment_labels, include_lowest=True)
employment_counts = employment_categories.value_counts()
employment_percentages = employment_counts / total_samples * 100

# Print the count and percentage of each category
for category, count, percentage in zip(employment_counts.index, employment_counts, employment_percentages):
    print(f"{category}: Count = {count}, Percentage = {percentage:.2f}%")

# Plotting histogram for income distribution
plt.hist(data['Income'], color='#4b0082', bins=10)

# Adding labels and title
plt.xlabel('Income', size = 14)
plt.ylabel('Frequency', size = 14)
plt.title('Income Distribution of Applicants', size = 18)
plt.xticks(size = 14)
plt.yticks(size = 14)
# Saving the plot image
plt.savefig('Income.png')
# Show the plot
plt.show()

# Define income categories
income_bins = [0, 10000, 20000, 30000, 40000, 50000, 60000, 70000, 80000, 90000, 100000]
income_labels = ['0-10000', '10001-20000', '20001-30000', '30001-40000', '40001-50000', '50001-60000', '60001-70000', '70001-80000', '80001-90000', '90001-100000']

# Count the occurrences of each income category
income_categories = pd.cut(data['Income'], bins=income_bins, labels=income_labels, include_lowest=True)
income_counts = income_categories.value_counts()
income_percentages = income_counts / total_samples * 100

# Print the count and percentage of each income category
for category, count, percentage in zip(income_counts.index, income_counts, income_percentages):
    print(f"{category}: Count = {count}, Percentage = {percentage:.2f}%")
```

Bivariate Analysis

```
# Create a crosstab for Gender vs Approval Status
gender_approval_ct = pd.crosstab(data['Gender'], data['ApprovalStatus'])

# Define new colors for the bars
dark_purple_color = '#4B0082'  # Dark purple shade
pastel_pink_color = '#FFC0CB'  # Pastel pink shade

# Plot the crosstab with colors and set figure size
```

```
ax = gender_approval_ct.plot(kind='bar',
                             color=[dark_purple_color, pastel_pink_color],
                             figsize=(10, 7))

# Add labels and title
ax.set_xlabel('Gender', size = 14)
ax.set_ylabel('Count', size = 14)
ax.set_title('Relationship Between Gender and Approval Status', size =18)

# Rotate the x-axis labels to show them horizontally
plt.xticks(size = 14, rotation=0)
plt.yticks(size = 14)

# Calculate the percentages and annotate the bars
for i, bar in enumerate(ax.patches):
    # Get the total samples for the gender category
    gender = 'a' if i < len(gender_approval_ct) else 'b'
    total_for_gender = gender_approval_ct.loc[gender].sum()

    # Calculate the percentage
    percentage = (bar.get_height() / total_for_gender) * 100

    # Annotate the bar with the percentage
    ax.annotate(f'{percentage:.2f}%', (bar.get_x() + bar.get_width() / 2, bar.get_height()),
ha='center', va='bottom', size = 14)

# Saving the plot image
plt.savefig('Gender vs Approval Status.png')
# Show the plot
plt.show()

colors = ['#9932CC', '#FFC0CB']  # Dark purple and pink

# Create a crosstabulation between Approval Status and Education Level
approval_education_ct = pd.crosstab(data['ApprovalStatus'], data['EducationLevel'])

# Plotting the crosstabulation with Approval Status on the x-axis and custom colors
fig, ax = plt.subplots(figsize=(10, 7))
approval_education_ct.T.plot(kind='bar', stacked=True, color=colors, ax=ax)

# Customize the plot with labels, title, and rotated x-axis labels for clarity
ax.set_xlabel('Education Level', size = 14)
ax.set_ylabel('Count', size = 14)
ax.set_title('Relationship Between Approval Status and Education Level', size = 18)
plt.xticks(size = 14, rotation=0)
plt.yticks(size = 14)
# Saving the plot image
plt.savefig('Education Level vs Approval Status.png')
# Display the plot
plt.show()
```

```
colors = ['#4B0082', '#FFC0CB']  # Dark purple and pastel pink

# Recreate the crosstab
prior_default_approval_ct = pd.crosstab(data['ApprovalStatus'], data['PriorDefaultHistory'])

# Plotting the bar chart using the new colors
ax = prior_default_approval_ct.plot(kind='bar',
                                    color=colors,
                                    figsize=(10, 7))

# Set the labels and title
ax.set_xlabel('Approval Status', size = 14)
ax.set_ylabel('Count', size = 14)
ax.set_title('Relationship Between Prior Default and Approval Status', size = 18)

# Rotate the x-axis labels to show them horizontally
plt.xticks(size = 14, rotation=0)
plt.yticks(size = 14)

# Annotate the bars with the count
for p in ax.patches:
    ax.annotate(str(p.get_height()), (p.get_x() * 1.005, p.get_height() * 1.005), size = 14)

# Saving the plot image
plt.savefig('Prior Default  vs Approval Status.png')
# Show the plot
plt.show()

# Creating intervals for the CreditScore using pd.cut
bins = range(0, max(data['CreditScore']) + 10, 10)  # Adjust the range and bin width as
necessary
labels = [f'{i}-{i+10}' for i in bins[:-1]]
data['CreditScoreBin'] = pd.cut(data['CreditScore'], bins=bins, labels=labels, right=False)

# Creating a crosstab with the binned CreditScore
CreditScore = pd.crosstab(data['CreditScoreBin'], data['ApprovalStatus'])

# Normalize the cross-tabulation by row
CreditScore_normalized = CreditScore.div(CreditScore.sum(1).astype(float), axis=0)

# Plot with specified colors in shades of purple and pink
colors = ['#4B0082', '#FFC0CB']  # A gradient of purple and pink colors

# Plotting
CreditScore_normalized.plot(kind='bar', stacked=False, color=colors, figsize=(17, 10))

# Rotating x-axis labels for better readability
plt.xticks(size = 14, rotation=0)
plt.yticks(size = 14)
```

```
# Place the legend inside the plot area at the upper left, but inside the plot
plt.legend(title='ApprovalStatus', loc='upper right')

# Setting the labels and title
plt.xlabel('Credit Score Intervals', size = 14)
plt.ylabel('Proportion of Approval Status', size =14)
plt.title('Proportion of Approval Status for Credit Score Intervals', size = 18)
# Saving the plot image
plt.savefig('Credit Score  vs Approval Status.png')
# Show the plot
plt.show()

data.drop('CreditScoreBin', axis=1, inplace=True)
data.head()

colors = ['#D8BFD8', '#DDA0DD']

# Create separate data for each category of 'DriversLicense'
license_groups = data.groupby('DriversLicenseStatus')

# Creating a figure and axes for the pie charts
fig, axs = plt.subplots(1, 2, figsize=(14, 7))

for ax, (license_status, group) in zip(axs, license_groups):
    status_counts = group['ApprovalStatus'].value_counts()
    ax.pie(status_counts, labels=status_counts.index, autopct='%1.1f%%', startangle=90,
colors=colors)
    ax.set_title(f'Has Driver\'s License: {license_status}', size = 14)

plt.suptitle('Credit Card Approval by Driver\'s License Status', size = 18)
# Saving the plot image
plt.savefig('Driver\'s License  vs Approval Status.png')
plt.show()

# Create a countplot for Credit Card Approval Status by Citizenship
plt.figure(figsize=(12, 6))
sns.countplot(x='Citizen', hue='ApprovalStatus', data=data, palette={'+': '#4B0082', '-': 'pink'})

plt.title('Credit Card Approval Status by Citizenship (Count)', size = 18)
plt.xlabel('Citizenship', size = 14)
plt.ylabel('Count', size = 14)
plt.legend(title='Approval Status')
plt.xticks(size = 14, rotation=0)
plt.yticks(size = 14)
# Saving the plot image
plt.savefig('Approval Status vs Citizenship.png')
plt.show()

# Set the color palette
sns.set_palette('rocket')
```

```
# Selecting only numerical columns
numeric_data = data.select_dtypes(include='number')

# Creating pairplot
sns.pairplot(data=numeric_data)
plt.suptitle('Pairplot of Numerical Attributes', y=1.02, size = 25)
plt.xticks(size = 14)
plt.yticks(size = 14)
# Saving the plot image
plt.savefig('Pairplot.png')
plt.show()
```

Data Preprocessing

```
# Check for missing values in the entire DataFrame
missing_values = data.isnull().sum()

# Plotting the missing values
plt.figure(figsize=(15, 6))
sns.barplot(x=missing_values.index, y=missing_values.values, color = '#DDA0DD')
plt.xticks(size = 14, rotation=90)
plt.yticks(size = 14)
plt.xlabel('Features', size = 14)
plt.ylabel('Number of Missing Values', size = 14)
plt.title('Missing Values in Each Column', size = 25)
plt.savefig('Missing Values.png')
plt.show()

print(missing_values)

# For numerical columns, fill missing values with the mean
for col in data.select_dtypes(include=['float64', 'int64']).columns:
    data[col] = data[col].fillna(data[col].mean())

# For categorical columns, fill missing values with the mode (the most frequently occurring
value)
for col in data.select_dtypes(include=['object']).columns:
    data[col] = data[col].fillna(data[col].mode()[0])

# Check if there are any missing values left
print(data.isnull().sum())

# Initialize LabelEncoder
lbl_en = LabelEncoder()

for col in data.columns:
    if data[col].dtypes=='object':
        data[col]=lbl_en.fit_transform(data[col])
```

```
#Displaying the head
data.head()

# Define a function to handle outliers by capping them to lower and upper bounds
def handle_outliers(series):
    # Calculate Q1 (25th percentile) and Q3 (75th percentile)
    Q1 = series.quantile(0.25)
    Q3 = series.quantile(0.75)

    # Calculate IQR (Interquartile Range)
    IQR = Q3 - Q1

    # Find lower and upper bounds for outliers
    lower_bound = Q1 - 1.5 * IQR
    upper_bound = Q3 + 1.5 * IQR

    # Cap outliers to lower and upper bounds
    series = series.clip(lower=lower_bound, upper=upper_bound)

    return series

# Select numerical columns for outlier detection and handling
numerical_columns = ['Age', 'Debt', 'YearsEmployed', 'CreditScore', 'ZipCode', 'Income']

# Count outliers before handling
outliers_before = (data[numerical_columns] < data[numerical_columns].quantile(0.25) - 1.5 *
(data[numerical_columns].quantile(0.75) - data[numerical_columns].quantile(0.25))) |
(data[numerical_columns] > data[numerical_columns].quantile(0.75) + 1.5 *
(data[numerical_columns].quantile(0.75) - data[numerical_columns].quantile(0.25)))
num_outliers_before = outliers_before.sum().sum()

# Handle outliers by capping them to lower and upper bounds
data_outliers_handled = data.copy()  # Create a copy of the original DataFrame
for column in numerical_columns:
    data_outliers_handled[column] = handle_outliers(data[column])

# Count outliers after handling
outliers_after = (data_outliers_handled[numerical_columns] <
data_outliers_handled[numerical_columns].quantile(0.25) - 1.5 *
(data_outliers_handled[numerical_columns].quantile(0.75) -
data_outliers_handled[numerical_columns].quantile(0.25))) |
(data_outliers_handled[numerical_columns] >
data_outliers_handled[numerical_columns].quantile(0.75) + 1.5 *
(data_outliers_handled[numerical_columns].quantile(0.75) -
data_outliers_handled[numerical_columns].quantile(0.25)))
num_outliers_after = outliers_after.sum().sum()

# Print the number of outliers before and after handling
print("Number of outliers before handling:", num_outliers_before)
print("Number of outliers after handling:", num_outliers_after)
```

```
# Visualize outliers using box plots
plt.figure(figsize=(20, 6))
plt.subplot(1, 2, 1)
data[numerical_columns].boxplot()
plt.title('Box Plot of Numerical Features (Before Handling Outliers)', size =16)
plt.subplot(1, 2, 2)
data_outliers_handled[numerical_columns].boxplot()
plt.title('Box Plot of Numerical Features (After Handling Outliers)', size = 16)
plt.savefig('Outliers.png')
plt.show()

# Split the data into features (X) and target variable (y)
X = data_outliers_handled.drop(columns=['ApprovalStatus'])  # Features
y = data_outliers_handled['ApprovalStatus']  # Target variable

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

# Print the shapes of the training and testing sets
print("X_train shape:", X_train.shape)
print("X_test shape:", X_test.shape)
print("y_train shape:", y_train.shape)
print("y_test shape:", y_test.shape)

# Initialize the MinMaxScaler
scaler = MinMaxScaler()

# Fit the scaler on the training data and transform both training and testing data
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Convert the scaled arrays back to DataFrames
X_train_scaled_df = pd.DataFrame(X_train_scaled, columns=X_train.columns)
X_test_scaled_df = pd.DataFrame(X_test_scaled, columns=X_test.columns)

# Print the scaled training and testing data
print("Scaled Training Data:")
print(X_train_scaled_df.head())
print("\nScaled Testing Data:")
print(X_test_scaled_df.head())
```

Logistic Regression

```
# Initialize the logistic regression model
logistic_reg_model = LogisticRegression()

# Train the model on the training data
logistic_reg_model.fit(X_train_scaled, y_train)
```

```
# Make predictions on the testing data
y_pred_lr = logistic_reg_model.predict(X_test_scaled)

# Evaluate the model's performance
accuracy_lr = accuracy_score(y_test, y_pred_lr)
print("Accuracy:", accuracy_lr)

# Calculate confusion matrix
conf_matrix_lr = confusion_matrix(y_test, y_pred_lr)

# Printing the confusion matrix
print(conf_matrix_lr)

# Define the hyperparameters grid
param_grid_lr = {
    'C': [0.001, 0.01, 0.1, 1, 10, 100],  # Regularization parameter
    'penalty': ['l1', 'l2'],              # Penalty term
    'solver': ['liblinear']               # Solver for optimization problem
}

# Initialize logistic regression model
logistic_reg_model = LogisticRegression()

# Initialize GridSearchCV
grid_search_lr = GridSearchCV(estimator=logistic_reg_model, param_grid=param_grid_lr,
cv=5, scoring='accuracy')

# Perform Grid Search Cross Validation
grid_search_lr.fit(X_train_scaled, y_train)

# Best parameters found during grid search
best_params_lr = grid_search_lr.best_params_
print("Best Parameters:", best_params_lr)

# Use the best estimator found during grid search
best_model_lr = grid_search_lr.best_estimator_

# Make predictions on the testing data using the best model
y_pred_lr = best_model_lr.predict(X_test_scaled)

# Evaluate the model's performance
accuracy_lr  = accuracy_score(y_test, y_pred_lr)
print("Accuracy:", accuracy_lr)

#get correlations of each features in dataset
corrmat_lr = data.corr()
top_corr_features_lr = corrmat_lr.index
plt.figure(figsize=(14,9))
#plot heat map
g=sns.heatmap(data[top_corr_features_lr].corr(),annot=True,cmap="Purples")
```

```
#absolute value of correlcation with ApprovalStatus
approvalStatus_corr_lr = np.abs(corrmat_lr['ApprovalStatus'])
#selecting columns with correlaction of 0.19 and above
mask = approvalStatus_corr_lr >= 0.17
approvalStatus_corr_lr = approvalStatus_corr_lr[mask]
approvalStatus_corr_lr.drop(index=['BankCustomerStatus','MaritalStatus','ApprovalStatus'],
inplace=True)
approvalStatus_corr_lr

# Selecting the features based on correlation
selected_features_lr = approvalStatus_corr_lr.index

# Extracting features and target variable
X = data[selected_features_lr]
y = data['ApprovalStatus']

# Splitting the data into training and testing sets
X_train_lr, X_test_lr, y_train_lr, y_test_lr = train_test_split(X, y, test_size=0.3,
random_state=40)

# Initialize the logistic regression model with a higher max_iter value
logistic_reg_model = LogisticRegression(max_iter=1000)

# Train the model on the training data
logistic_reg_model.fit(X_train_lr, y_train_lr)

# Make predictions on the testing data
y_pred_lr = logistic_reg_model.predict(X_test_lr)

# Evaluate the model's performance
accuracy_lr = accuracy_score(y_test_lr, y_pred_lr)
print("Accuracy:", accuracy_lr)

# Calculate confusion matrix for test set
conf_matrix_test_lr = confusion_matrix(y_test_lr, y_pred_lr)

# Plot confusion matrix for test set
plt.figure(figsize=(10, 8))  # Adjust figure size
sns.heatmap(conf_matrix_test_lr, annot=True, fmt="d", cmap="Purples", cbar=False,
        xticklabels=['Predicted Negative', 'Predicted Positive'],  # Adjusted for clarity
        yticklabels=['Actual Negative', 'Actual Positive'],  # Adjusted for clarity
        annot_kws={"size": 25})  # Increase size of the values inside the plot
plt.title('Confusion Matrix of Logistic Regression', size=20)
plt.xlabel('Predicted Labels', size=18)
plt.ylabel('True Labels', size=18)
plt.xticks(size=14)
plt.yticks(size=14)
plt.savefig('Conf_mat.png')
plt.show()
```

```
print(conf_matrix_test_lr)

# Evaluate model performance on training set
train_accuracy_lr = logistic_reg_model.score(X_train_lr, y_train_lr)
print("Training Accuracy:", train_accuracy_lr)

# Evaluate model performance on testing set
test_accuracy_lr = logistic_reg_model.score(X_test_lr, y_test_lr)
print("Testing Accuracy:", test_accuracy_lr)

# Calculate precision, recall, and F1-score for training data
train_precision_lr = precision_score(y_train_lr, logistic_reg_model.predict(X_train_lr))
train_recall_lr = recall_score(y_train_lr, logistic_reg_model.predict(X_train_lr))
train_f1_score_lr = f1_score(y_train_lr, logistic_reg_model.predict(X_train_lr))
train_roc_auc_lr = roc_auc_score(y_train_lr, logistic_reg_model.predict(X_train_lr))

# Calculate precision, recall, and F1-score for testing data
test_precision_lr = precision_score(y_test_lr, y_pred_lr)
test_recall_lr = recall_score(y_test_lr, y_pred_lr)
test_f1_score_lr = f1_score(y_test_lr, y_pred_lr)
test_roc_auc_lr = roc_auc_score(y_test_lr, y_pred_lr)

# Print the evaluation metrics
print("Evaluation Metrics:")
print("Training Data:")
print("  Accuracy:", train_accuracy_lr)
print("  Precision:", train_precision_lr)
print("  Recall:", train_recall_lr)
print("  F1-score:", train_f1_score_lr)
print("ROC AUC Score:", train_roc_auc_lr)

print("\nTesting Data:")
print("  Accuracy:", test_accuracy_lr)
print("  Precision:", test_precision_lr)
print("  Recall:", test_recall_lr)
print("  F1-score:", test_f1_score_lr)
print("ROC AUC Score:", test_roc_auc_lr)
```

Decision Tree

```
# Initialize the decision tree model
decision_tree_model = DecisionTreeClassifier()

# Train the model on the training data
decision_tree_model.fit(X_train_scaled, y_train)

# Make predictions on the testing data
y_pred_dt = decision_tree_model.predict(X_test_scaled)

# Evaluate the model's performance
```

```
accuracy_dt = accuracy_score(y_test, y_pred_dt)
print("Accuracy:", accuracy_dt)

#Calculate the confusion matrix
conf_matrix_dt = confusion_matrix(y_test, y_pred_dt)
#Printing the confusion matrix
print(conf_matrix_dt)

# Define the hyperparameters grid
param_grid_dt = {
   'max_depth': [3, 5, 7, 10],              # Maximum depth of the tree
   'min_samples_split': [2, 5, 10],          # Minimum number of samples required to split an
internal node
   'min_samples_leaf': [1, 2, 4]             # Minimum number of samples required to be at a leaf
node
}

# Initialize the decision tree classifier
dt_classifier = DecisionTreeClassifier(random_state=42)

# Initialize GridSearchCV with the classifier and hyperparameters grid
grid_search_dt = GridSearchCV(estimator=dt_classifier, param_grid=param_grid_dt, cv=5,
scoring='accuracy')

# Perform Grid Search Cross Validation
grid_search_dt.fit(X_train_scaled, y_train)

# Best parameters found during grid search
best_params_dt = grid_search_dt.best_params_
print("Best Parameters:", best_params_dt)

# Use the best estimator found during grid search
best_model_dt = grid_search_dt.best_estimator_

# Make predictions on the testing data using the best model
y_pred_dt = best_model_dt.predict(X_test_scaled)

# Evaluate the model's performance
accuracy_dt = accuracy_score(y_test, y_pred_dt)
print("Accuracy:", accuracy_dt)

# Initialize a decision tree classifier
decision_tree = DecisionTreeClassifier(max_depth=5, min_samples_leaf=1,
min_samples_split=2, random_state=42)

# Train the decision tree classifier
decision_tree.fit(X_train_scaled, y_train)

# Get feature importances
feature_importances = decision_tree.feature_importances_
```

```
# Sort feature importances in descending order
indices = feature_importances.argsort()[::-1]

# Select the top k features (e.g., top 5)
k = min(5, len(X.columns))  # Ensure k is not greater than the number of columns in X
selected_features_indices = indices[:k]

# Print selected feature indices
print("Selected Feature Indices:", selected_features_indices)

# Get names of features based on their importance scores
selected_features = data.columns[selected_features_indices]

# Print selected features
print("Selected Features:", selected_features)

# Select only the top k features from the training and testing data
X_train_selected_dt = X_train.iloc[:, selected_features_indices]
X_test_selected_dt = X_test.iloc[:, selected_features_indices]

# Initialize a decision tree classifier
decision_tree = DecisionTreeClassifier(max_depth=5, min_samples_leaf=1,
min_samples_split=2, random_state=42)

# Train the decision tree classifier on the selected features
decision_tree.fit(X_train_selected_dt, y_train)

# Make predictions on the testing data using the model trained on selected features
y_pred_selected_dt = decision_tree.predict(X_test_selected_dt)

# Evaluate the model's performance
accuracy_selected_dt = accuracy_score(y_test, y_pred_selected_dt)
print("Accuracy with Selected Features:", accuracy_selected_dt)

# Calculate confusion matrix for test set with selected features
conf_matrix_selected_dt = confusion_matrix(y_test, y_pred_selected_dt)

# Plot confusion matrix for test set with selected features
plt.figure(figsize=(10, 8))  # Adjust figure size
sns.heatmap(conf_matrix_selected_dt, annot=True, fmt="d", cmap="Purples", cbar=False,
        xticklabels=['Predicted Negative', 'Predicted Positive'],  # Adjusted for clarity
        yticklabels=['Actual Negative', 'Actual Positive'],  # Adjusted for clarity
        annot_kws={"size": 25})  # Increase size of the values inside the plot
plt.title('Confusion Matrix of Decision Tree', size=20)
plt.xlabel('Predicted Labels', size=18)
plt.ylabel('True Labels', size=18)
plt.xticks(size=14)
plt.yticks(size=14)
plt.savefig('Conf_mat_DT.png')
```

```
plt.show()
print(conf_matrix_selected_dt)

# Make predictions on the training data
y_train_pred_dt = decision_tree.predict(X_train_selected_dt)

# Calculate the accuracy on the training data
train_accuracy_dt = accuracy_score(y_train, y_train_pred_dt)
print("Training Accuracy:", train_accuracy_dt)

# Calculate the accuracy on the testing data
test_accuracy_dt = accuracy_score(y_test, y_pred_selected_dt)
print("Testing Accuracy:", test_accuracy_dt)

# Initialize the decision tree classifier
decision_tree = DecisionTreeClassifier(max_depth=3, min_samples_leaf=1,
min_samples_split=2, random_state=42)

# Perform k-fold cross-validation
num_folds = 5  # Number of folds
cv_scores = cross_val_score(decision_tree, X_train, y_train, cv=num_folds)

# Print the cross-validation scores
print("Cross-Validation Scores:", cv_scores)

# Calculate and print the mean and standard deviation of the cross-validation scores
mean_cv_score = cv_scores.mean()
std_cv_score = cv_scores.std()
print("Mean Cross-Validation Score:", mean_cv_score)
print("Standard Deviation of Cross-Validation Scores:", std_cv_score)

# Calculate evaluation metrics for training set
train_accuracy_dt = accuracy_score(y_train,  y_train_pred_dt)
train_precision_dt = precision_score(y_train,  y_train_pred_dt)
train_recall_dt = recall_score(y_train,  y_train_pred_dt)
train_f1_score_dt = f1_score(y_train,  y_train_pred_dt)
train_roc_auc_dt = roc_auc_score(y_train,  y_train_pred_dt)

# Calculate evaluation metrics for test set
test_accuracy_dt = accuracy_score(y_test, y_pred_selected_dt)
test_precision_dt = precision_score(y_test, y_pred_selected_dt)
test_recall_dt = recall_score(y_test, y_pred_selected_dt)
test_f1_score_dt = f1_score(y_test, y_pred_selected_dt)
test_roc_auc_dt = roc_auc_score(y_test,  y_pred_selected_dt)

# Print evaluation metrics
print("Evaluation Metrics for Training Set:")
print("Accuracy:", train_accuracy_dt)
print("Precision:", train_precision_dt)
print("Recall:", train_recall_dt)
```

```
print("F1-score:", train_f1_score_dt)
print("ROC AUC Score:", train_roc_auc_dt)
print("\n")

print("Evaluation Metrics for Test Set:")
print("Accuracy:", test_accuracy_dt)
print("Precision:", test_precision_dt)
print("Recall:", test_recall_dt)
print("F1-score:", test_f1_score_dt)
print("ROC AUC Score:", test_roc_auc_dt)
```

Random Forest

```
# Initialize the Random Forest classifier
random_forest_model = RandomForestClassifier(random_state=42)

# Train the model on the training data
random_forest_model.fit(X_train_scaled, y_train)

# Make predictions on the testing data
y_pred_rf = random_forest_model.predict(X_test_scaled)

# Evaluate the model's performance
accuracy_rf = accuracy_score(y_test, y_pred_rf)
print("Accuracy:", accuracy_rf)

# Define the hyperparameters grid
param_grid_rf = {
   'n_estimators': [100, 200, 300],       # Number of trees in the forest
   'max_depth': [None, 10, 20, 30],       # Maximum depth of the trees
   'min_samples_split': [2, 5, 10],       # Minimum number of samples required to split an internal node
   'min_samples_leaf': [1, 2, 4]          # Minimum number of samples required to be at a leaf node
}

# Initialize the Random Forest classifier
random_forest_model = RandomForestClassifier(random_state=42)

# Initialize GridSearchCV with the classifier and hyperparameters grid
grid_search_rf = GridSearchCV(estimator=random_forest_model, param_grid=param_grid_rf, cv=5, scoring='accuracy')

# Perform Grid Search Cross Validation
grid_search_rf.fit(X_train_scaled, y_train)

# Best parameters found during grid search
best_params_rf = grid_search_rf.best_params_
print("Best Parameters:", best_params_rf)
```

```
# Use the best estimator found during grid search
best_model_rf = grid_search_rf.best_estimator_

# Make predictions on the testing data using the best model
y_pred_rf = best_model_rf.predict(X_test_scaled)

# Evaluate the model's performance
accuracy_rf = accuracy_score(y_test, y_pred_rf)
print("Accuracy:", accuracy_rf)

# Initialize Random Forest classifier
random_forest_model = RandomForestClassifier(max_depth=None, min_samples_leaf=2,
min_samples_split=2, n_estimators=200)

# Train the model on the training data
random_forest_model.fit(X_train_scaled, y_train)

# Extract feature importances
feature_importances = random_forest_model.feature_importances_

# Create a selector object that will use the random forest classifier to identify
# features whose importance is greater than or equal to a specified threshold
feature_selector = SelectFromModel(random_forest_model, threshold='median')

# Fit the selector to the data
feature_selector.fit(X_train_scaled, y_train)

# Transform the data to include only the selected features
X_train_selected = feature_selector.transform(X_train_scaled)
X_test_selected = feature_selector.transform(X_test_scaled)

# Get the selected feature indices
selected_feature_indices = feature_selector.get_support(indices=True)

# Check if the selected feature indices are within bounds
selected_feature_indices = [i for i in selected_feature_indices if i < len(X.columns)]

# Get the names of the selected features
selected_features = X.columns[selected_feature_indices]

# Print the selected features
print("Selected Features:", selected_features)

# Initialize the RandomForestClassifier with the best hyperparameters
random_forest_model = RandomForestClassifier(max_depth=None, min_samples_leaf=2,
min_samples_split=2, n_estimators=200)

# Train the model on the training data with selected features
random_forest_model.fit(X_train_selected, y_train)
```

```
# Make predictions on the testing data with selected features
y_pred_rf_selected = random_forest_model.predict(X_test_selected)

# Evaluate the model's performance
accuracy_rf_selected = accuracy_score(y_test, y_pred_rf_selected)
print("Accuracy with Selected Features:", accuracy_rf_selected)

# Calculate confusion matrix for test set with selected features
conf_matrix_rf_selected = confusion_matrix(y_test, y_pred_rf_selected)

# Plot confusion matrix for test set with selected features
plt.figure(figsize=(10, 8)) # Adjust figure size
sns.heatmap(conf_matrix_rf_selected, annot=True, fmt="d", cmap="Purples", cbar=False,
        xticklabels=['Predicted Negative', 'Predicted Positive'],  # Adjusted for clarity
        yticklabels=['Actual Negative', 'Actual Positive'],  # Adjusted for clarity
        annot_kws={"size": 25})  # Increase size of the values inside the plot
plt.title("Confusion Matrix of Random Forest", size=20)
plt.xlabel('Predicted Labels', size=18)
plt.ylabel('True Labels', size=18)
plt.xticks(size=14)
plt.yticks(size=14)
plt.savefig('Conf_mat_RF.png')
plt.show()
print(conf_matrix_rf_selected)

# Train the model with the selected features from the training set
best_model_rf.fit(X_train_selected, y_train)

# predictions and evaluations can proceed with the correctly matched feature sets
y_train_pred = best_model_rf.predict(X_train_selected)

print("Training Set Evaluation:")
print("Accuracy:", accuracy_score(y_train, y_train_pred))
print("Precision:", precision_score(y_train, y_train_pred))
print("Recall:", recall_score(y_train, y_train_pred))
print("F1 Score:", f1_score(y_train, y_train_pred))
print("ROC AUC Score:", roc_auc_score(y_train,
best_model_rf.predict_proba(X_train_selected)[:, 1]))

# Evaluate the model on the testing set
print("\nTesting Set Evaluation:")
print("Accuracy with Selected Features:", accuracy_rf_selected)
print("Precision:", precision_score(y_test, y_pred_rf_selected))
print("Recall:", recall_score(y_test, y_pred_rf_selected))
print("F1 Score:", f1_score(y_test, y_pred_rf_selected))
print("ROC AUC Score:", roc_auc_score(y_test,
best_model_rf.predict_proba(X_test_selected)[:, 1]))

# Perform cross-validation
```

```
cv_scores = cross_val_score(best_model_rf, X_train_selected, y_train, cv=5, scoring='accuracy')

# Calculate the mean and standard deviation of the cross-validation scores
cv_mean = cv_scores.mean()
cv_std = cv_scores.std()

print(f"CV Accuracy Mean: {cv_mean:.4f}")
print(f"CV Accuracy Standard Deviation: {cv_std:.4f}")

# Already computed accuracy for Logistic Regression, Decision Tree, and Random Forest
accuracies = [accuracy_lr, accuracy_dt, accuracy_rf_selected]
precisions = [precision_score(y_test, y_pred_lr), precision_score(y_test, y_pred_dt), precision_score(y_test, y_pred_rf_selected)]
recalls = [recall_score(y_test, y_pred_lr), recall_score(y_test, y_pred_dt), recall_score(y_test, y_pred_rf_selected)]
f1_scores = [f1_score(y_test, y_pred_lr), f1_score(y_test, y_pred_dt), f1_score(y_test, y_pred_rf_selected)]

model_names = ['Logistic Regression', 'Decision Tree', 'Random Forest']

# Create a DataFrame to display the metrics
metrics_df = pd.DataFrame({
    'Model': model_names,
    'Accuracy': accuracies,
    'Precision': precisions,
    'Recall': recalls,
    'F1 Score': f1_scores
})

metrics_df.set_index('Model', inplace=True)
print(metrics_df)
colors = ['#4B0082', '#DDA0DD', '#D8BFD8' , '#FFC0CB']

# Plotting the metrics for comparison
metrics_df.plot(kind='bar', figsize=(15, 7), color = colors)
plt.title('Performance Comparison of ML Models', size = 25)
plt.xlabel('Models', size =14)
plt.ylabel('Score', size =14)
plt.xticks(size =14, rotation=90)
plt.yticks (size =14)
plt.legend(loc='lower right')
plt.savefig('Performance Comparison.png')
plt.show()
```