

A Study on whether Automated Machine Learning or Generative AI can replace the Human Data Scientist

By

Ryan Kho Yuen Thian



**FACULTY OF COMPUTING AND
INFORMATION TECHNOLOGY**

**TUNKU ABDUL RAHMAN UNIVERSITY OF
MANAGEMENT AND TECHNOLOGY
KUALA LUMPUR**

**ACADEMIC YEAR
2024/25**

A Study on whether Automated Machine Learning or Generative AI can replace the Human Data Scientist

By

Ryan Kho Yuen Thian

Supervisor: Ms Fatin Izzati Binti Ramli

A project report submitted to the
Faculty of Computing and Information Technology
in partial fulfillment of the requirement for the
Bachelor of Computer Science (Honours)

Department of Mathematical and Data Science
Faculty of Computing and Information Technology
Tunku Abdul Rahman University of Management and Technology
Kuala Lumpur

Copyright by Tunku Abdul Rahman University of Management and Technology.

All rights reserved. No part of this project documentation may be reproduced, stored in retrieval system, or transmitted in any form or by any means without prior permission of Tunku Abdul Rahman University of Management and Technology.

Declaration

The project submitted herewith is a result of my own efforts in totality and in every aspect of the project works. All information that has been obtained from other sources had been fully acknowledged. I understand that any plagiarism, cheating or collusion or any sorts constitutes a breach of TAR University rules and regulations and would be subjected to disciplinary actions.

RyanK

Ryan Kho Yuen Thian

Bachelor of Computer Science (Honours) in Data Science

ID: 22WMR04097

Abstract

The rapid advancements in Artificial Intelligence (AI) and Automated Machine Learning (AutoML) have sparked a growing debate on whether these technologies can effectively replace human Data Scientists in the process of developing and implementing machine learning models. With automated tools becoming more sophisticated, the question of whether human expertise is still necessary is increasingly relevant. This study aims to explore the potential of Generative AI (represented by ChatGPT Data Analyst) and AutoML (represented by H2O AutoML) in executing machine learning tasks that are traditionally handled by human Data Scientists. The focus is on assessing whether these technologies can perform these tasks at a comparable level in terms of accuracy, efficiency, and adaptability. To evaluate these approaches, we use a credit risk assessment case study, with the German Credit dataset serving as the benchmark. Each of the three approaches—human Data Scientist, H2O AutoML, and ChatGPT Data Analyst—follows the CRISP-DM (Cross Industry Standard Process for Data Mining) methodology to build, evaluate, and deploy predictive models. By adhering to established Machine Learning Best Practices, this case study ensures a fair comparison across all methods. The strengths and weaknesses of each approach are critically analyzed, focusing on key factors such as capability, efficiency, reliability and flexibility. The human Data Scientist brings a deep understanding of domain-specific knowledge and customization, while AutoML and Generative AI aim to streamline the process, potentially reducing human involvement. The results of this study provide insights into whether Generative AI and AutoML can replace or complement human Data Scientists. This study also explores the possibility that a hybrid approach, combining human expertise with Generative AI and AutoML technologies, may offer the most promising solution.

Keywords: AutoML, Generative AI, Data Scientist, H2O, ChatGPT Data Analyst, Machine Learning Best Practices, Methodology, Strengths, Weaknesses

Acknowledgement

I would like to express my deepest gratitude to my supervisor, Ms Fatin Izzati Binti Ramli, for her invaluable guidance, unwavering support, and insightful feedback throughout the duration of this project. Her expertise and encouragement have been instrumental in shaping the direction of my research and helping me navigate through challenges.

Furthermore, I extend my appreciation to Tunku Abdul Rahman University of Management and Technology for providing access to exclusive research papers through its subscription services. This access has been invaluable in conducting thorough literature reviews and enhancing the depth of my research.

Table of Contents

Declaration.....	ii
Abstract.....	iv
Acknowledgement	v
Table of Contents	v
Figures and/or Tables	viii
1 Introduction	2
1.1 The Data Scientist Profession	2
1.2 AutoML	2
1.3 Generative AI.....	2
1.4 Problem Statement.....	3
1.5 Proposed Solution.....	3
1.6 Scope of Study.....	4
1.7 Chapter Summary and Evaluation	5
1.7.1 Chapter 1 Summary and Evaluation:	5
1.7.2 Chapter 2 Summary and Evaluation:	5
1.7.3 Chapter 3 Summary and Evaluation:	5
1.7.4 Chapter 4 Summary and Evaluation:	6
1.7.5 Chapter 5 Summary and Evaluation:	6
1.7.6 Chapter 6 Summary and Evaluation:	6
2 Literature Review.....	9
2.1 Background of ChatGPT Data Analyst (CDA)	9
2.1.1 What is ChatGPT Data Analyst (CDA)?	9
2.1.2 ChatGPT Data Analyst (CDA) Features.....	9
2.1.3 ChatGPT Data Analyst (CDA) Limitations	10
2.2 Works Related to the Use of ChatGPT Data Analyst (CDA).....	12
2.2.1 Building a Simple Model to forecast the Beginning of Ionospheric Storms from a Ionosonde dataset	12
2.2.2 LLMS Enhance Automated Machine Learning in Clinical Research.....	12
2.3 Background of AutoML.....	13
2.3.1 What is AutoML?	13
2.3.2 A Survey of Open Source AutoML Frameworks	15
2.4 Works Related to the Use of AutoML	17
2.4.1 Automated ML for Predictive Quality in Production	17
2.4.2 Man versus Machine: AutoML and Human Experts' Role in Phishing Detection	17
2.5 Evaluation	18
3 Methodology and Requirements Analysis.....	21
3.1 Selection of AutoML Framework.....	21
3.2 Case Study Approach.....	23
3.3 Machine Learning Algorithms.....	24
3.4 CRISP-DM Methodology	25
3.5 Machine Learning Best Practices	26
3.6 Development Tools and Frameworks	33
3.6.1 AutoML Approach (H2O AutoML)	33
3.6.2 Generative AI Approach (ChatGPT Data Analyst)	33
3.6.3 Human Data Scientist Approach.....	34
4 System Design	36
4.1 Algorithm Design for Credit Risk Assessment.....	36

4.2	Machine Learning Best Practices from Design Perspective	39
4.3	Machine Learning Algorithms Proposed by Human Data Scientist	57
4.3.1	Logistic Regression.....	57
4.3.2	Decision Tree.....	58
4.3.3	Random Forest.....	60
4.3.4	Gradient Boosting Machine.....	62
4.3.5	Neural Networks.....	64
5	Implementation and Testing	68
5.1	AutoML Approach.....	68
5.1.1	Machine Learning Model Implementation	68
5.1.2	User Interface & API Integration Implementation.....	113
5.1.3	System Integration Testing.....	117
5.2	Generative AI Approach.....	123
5.2.1	Machine Learning Model Implementation	123
5.2.2	User Interface & API Integration Implementation.....	139
5.2.3	System Integration Testing.....	142
5.3	Human Data Scientist Approach.....	146
5.3.1	Machine Learning Model Implementation	146
5.3.2	User Interface & API Integration Implementation.....	177
5.3.3	System Integration Testing.....	179
6	System Deployment.....	187
6.1	Overall View of Deployment.....	187
6.1.1	Resources/Components Needed.....	187
6.1.2	How the Components Work Together	187
6.2	Reasons for Local Deployment.....	188
6.3	How Communication between Streamlit and FastAPI is Achieved	188
6.3.1	Initialising a FastAPI Application Instance (FastAPI).....	188
6.3.2	Requesting for Prediction (Streamlit).....	189
6.3.3	Handling Prediction Request (FastAPI).....	189
6.3.4	Requesting for Explanation Plot (Streamlit)	190
6.3.5	Returning Explanation Plot (FastAPI)	190
6.4	Deployment Procedures.....	191
6.5	Challenges Faced during Deployment.....	191
7	Discussions and Conclusion.....	194
7.1	Summary	194
7.2	Achievements.....	194
7.3	Contributions	201
7.4	Limitations and Future Improvements.....	203
7.5	Issues and Solutions.....	206
References	209	
Appendices	218	

Figures and/or Tables

Figure 1.1 Summary of the Methods to be Compared	4
Figure 3.1. H2O AI high-level architecture. From <i>Practical Automated Machine Learning Using H2O.ai</i> (p. 122), by Salil Ajgaonkar, 2022, Livery Place, Birmingham: Packt. Copyright [2022] by Packt Publishing.....	22
Figure 3.2. Different clients communicating with the same H2O server in the Development Environment. From <i>Practical Automated Machine Learning Using H2O.ai</i> (p. 124), by Salil Ajgaonkar, 2022, Livery Place, Birmingham: Packt. Copyright [2022] by Packt Publishing.....	23
Figure 3.3. CRISP-DM. From <i>Understanding CRISP-DM and its importance in Data Science projects</i> by Z. Luna, 2021, Medium. https://medium.com/analytics-vidhya/understanding-crisp-dm-and-its-importance-in-data-science-projects-91c8742c9f9b . Copyright 2021 by Zipporah Luna.....	25
Figure 3.4. Flow Interface. From <i>Getting started with H2O using Flow</i> by P. Pandey, 2019, H2O.ai. https://h2o.ai/blog/2019/getting-started-with-h2o-using-flow/ . Copyright 2024 by H2O.ai	33
Figure 3.5. Error Checking and Outlier Detection. From <i>ChatGPT Advanced Data Analysis: Guide & Use Cases</i> by Akkio, 2024, Akkio. https://www.akkio.com/post/chatgpt-advanced-data-analysis . Copyright 2024 by Akkio	34
Figure 4.1. Proposed Algorithm Design for Credit Risk Assessment Script	38
Figure 4.2. Sample User Interface Design	56
Figure 4.3. The Process of Entering Input and Returning the Prediction. Adapted from <i>How to build complete end-to-end ML model, Backend RestAPI using FastAPI and front-end UI using Streamlit</i> by B. Gorad, 2022, Medium. https://medium.com/@goradbj/how-to-build-complete-end-to-end-ml-model-backend-restapi-using-fastapi-and-front-end-ui-using-22f64bf04476 . Copyright 2022 by Balwant Gorad	56
Figure 4.4. Key Assumptions for Implementing Logistic Regression. Adapted from <i>What is Logistic Regression? Equation, Assumptions, Types and Best Practices</i> by V. Kanade, 2022, Spiceworks. https://www.spiceworks.com/tech/artificial-intelligence/articles/what-is-logistic-regression/ . Copyright 2024 by Spiceworks Inc.....	57
Figure 4.5. An example of a Decision Tree. From <i>What is a Decision Tree?</i> by Master's in Data Science, 2024, Master's in Data Science	59
Figure 4.6. Random Forest. From <i>Random Forests</i> by R. Yehoshua, 2023, Medium	61
Figure 4.7. Gradient Boosting. From <i>Gradient Boosting in Machine Learning</i> by A. K. Singh, 2021, Medium	62
Figure 4.8. Neural Networks Architecture. From <i>Artificial Neural Networks and its Applications</i> by H. , 2023, GeeksforGeeks. https://www.geeksforgeeks.org/artificial-neural-networks-and-its-applications/ . Copyright 2024 by Harikiran78.....	64
Figure 6.1. Deployment	187
Figure 6.2. For Initialising FastAPI	188
Figure 6.3. Requesting Prediction	189
Figure 6.4. How Prediction is made.....	189
Figure 6.5. Requesting SHAP Plot.....	190
Figure 6.6. Returning SHAP Plot.....	190

Chapter 1

Introduction

1 Introduction

This chapter introduces the key topics in this study, namely the Data Scientist profession, Automated Machine Learning (AutoML) and Generative AI. After that, the Problem Statement, Proposed Solution and the study's Scope are presented.

1.1 The Data Scientist Profession

In 2012, Harvard Business Review authors, Thomas H. Davenport and D.J. Patil, introduced the idea that working as a Data Scientist was deemed the 'sexiest job of the 21st century' (Davenport & Patil, 2022). However, the validity of this claim in the present and future remains a subject of inquiry. It is known that Data Scientists play a crucial role in determining the pertinent questions for their teams and devising effective strategies to answer these questions through data analysis. On a daily basis, a data scientist may engage in activities, such as identifying trends and patterns within datasets to unveil valuable insights, utilising data analysis tools such as Python, R, Julia or SAS and developing algorithms and data models for predicting results (Coursera Staff, 2023). Yet, uncertainties loom over the Data Science profession due to technological advancements. AutoML (Automated Machine Learning) and Generative AI are two recent technologies that pose potential threats to the data scientist's job.

1.2 AutoML

AutoML (Automated Machine Learning), aiming to automate the whole Machine Learning (ML) workflow, offers techniques and procedures to democratise access to machine learning for individuals without expertise in the field, to enhance machine learning efficiency and to expedite research advancements (AutoML, n.d.). Even though these automation tools were initially crafted to alleviate the mundane aspects of developing machine learning models or to fill the gap for the absence of expertise, these tools have evolved to such an extent that they can now surpass the capabilities of human experts in certain situations. Examples of AutoML frameworks are H2O.ai AutoML, AutoKeras, TPOT, AutoGluon and Auto-sklearn (DataScientest, 2024).

1.3 Generative AI

Generative AI, on the other hand, refers to algorithms capable of generating novel content, spanning code, text, images, audio, videos and simulations (McKinsey & Company, 2023). A familiar example of Generative AI is ChatGPT, where GPT stands for Generative Pre-trained Transformer. In addition to its ability to compose poems, songs and invent stories, it can also summarise information and generate program code in response to prompts. In July 2023, OpenAI announced a ChatGPT plug-in called "Code Interpreter" which has now been renamed "ChatGPT

Data Analyst” (we shall use the acronym “CDA” to refer to it). This is a new subscription-based feature, available to ChatGPT Plus and Enterprise accounts, which enables users to upload their datasets/files directly into the ChatGPT interface, allowing it to analyse trends and patterns in data as well as generating data visualisations (Selvaraj, 2023a). A data scientist can also use ChatGPT Data Analyst (CDA) to assist in data pre-processing, feature engineering, model selection and evaluation, as well as performance optimisation.

1.4 Problem Statement

Given the powerful features of Generative AI and AutoML, which are capable of generating code and automating various machine learning tasks, there has been speculation about the potential for these technologies to replace data scientists. Some media articles on this issue include: “Can AutoML replace Data Scientists?” (Pascual, 2021), “Will ChatGPT take Data Science Jobs?” (Selvaraj, 2023b), and “Will ChatGPT Put Data Analysts Out Of Work?” (Marr, 2023). Even if the idea of Generative AI and AutoML completely replacing the data scientist seems premature or over the top now, these technologies still have the potential to transform the data scientist’s or data analyst’s job in a major way. If it eventually turns out that Generative AI/AutoML merely plays the role of a capable assistant to the data scientist, it is still important to know how to take advantage of these new technologies in order to boost one’s productivity and keep ahead of the competition. Regardless of the differing perspectives individuals may have regarding these technologies, it is crucial to note that the capabilities of these technologies are still evolving, emphasising the importance of staying updated through ongoing studies.

1.5 Proposed Solution

Because of the genuine concerns that Data Scientists and Data Analysts have regarding these technologies, this study aims to analyse the strengths and weaknesses of both the Manual (human) and Automated approaches (Generative AI, AutoML). I shall do this by assessing the 3 approaches against established machine learning best practices, which coincidentally will be incorporated within the various sub-processes of the Data Science process. The Data Science process used in the study will be mostly based on CRISP-DM. All this will be described in detail in the Methodology section. The case study chosen is Credit Risk Assessment (aka Prediction of Loan Default) which will involve Responsible AI issues (bias, interpretability/explainability). One Generative AI tool (ChatGPT Data Analyst) and one AutoML tool will be selected. Figure 1.1 below is a summary of the approaches/methods that will be compared for this study.

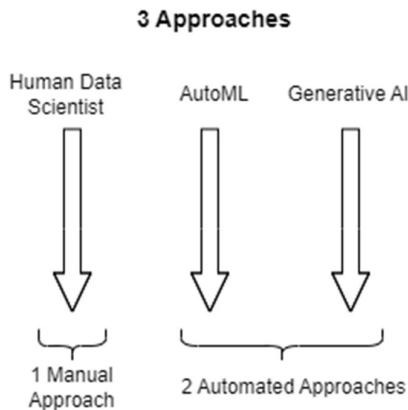


Figure 1.1 Summary of the Methods to be Compared

Research Objectives

- To explore the 3 approaches (human Data Scientist, AutoML, Generative AI) of machine learning model development, evaluation and deployment.
- To compare and evaluate the 3 approaches using a realistic case study (Credit Risk Assessment) and implementing established Machine Learning Best Practices.
- To come to a conclusion as to whether or not Generative AI or AutoML can replace human Data Scientists.

Research Question

R1. How do the 3 approaches (human Data Scientist, AutoML, Generative AI) compare in terms of the execution of established Machine Learning Best Practices?

At the end of the study, I will be able to answer the question: “Will AutoML or Generative AI replace the human Data Scientist?”. If the answer is “No”, which areas and approximately how much of the human Data Scientist’s work will be taken over by AutoML/Generative AI? If the human Data Scientist still has a role to play, will the way forward be a hybrid approach where the human Data Scientist collaborates with AutoML/Generative AI to solve Machine Learning problems?

1.6 Scope of Study

In this comparison study, I selected only one tool, namely ChatGPT Data Analyst, to represent Generative AI. I selected ChatGPT Data Analyst because of the widespread use of the ChatGPT base product and also because competing tools like Google Bard currently do not have the same kind of file upload processing and analysis capabilities that ChatGPT Data Analyst has.

One AutoML tool will be chosen from a pool of Open Source AutoML frameworks/tools. Selection will be based on criteria such as extent of automation features, comprehensiveness of algorithms support, performance, explainability/interpretability, and so forth.

I am using Credit Risk Assessment (prediction of loan default) as our case study. This is a Supervised Classification machine learning problem so the comparison study will be limited to this context. Having said that, supervised classification ML problems constitute a large portion of the overall ML problems.

Given more time, the study can be more comprehensive by covering Unsupervised Learning, Regression, Reinforcement Learning, as well as exploring larger data sets.

1.7 Chapter Summary and Evaluation

1.7.1 Chapter 1 Summary and Evaluation:

The Data Science profession, at one time revered, is now under threat from AutoML and Generative AI. AutoML and Generative AI are defined. The problem statement of this study is whether AutoML or Generative AI can take over the Data Scientist's job. The proposed solution to this question is to assess how well the 3 approaches (human Data Scientist, AutoML, Generative AI) implement Machine Learning Best Practices. This can be achieved by mostly following a data science methodology like CRISP-DM to implement a realistic machine learning case study. The scope of this study and its limitations are then defined.

1.7.2 Chapter 2 Summary and Evaluation:

Chapter 2 begins with more in-depth information on ChatGPT Data Analyst (CDA), including its benefits as well as its limitations. After that, two works on the use of CDA are discussed. The first is related to storm prediction, whereas the second is related to clinical research. For AutoML, a list of the stages/tasks in the machine learning life cycle where typically AutoML tools can automate is discussed. Following that, two works on the use of AutoML are discussed. The first work is related to prediction of quality in industrial processes whereas the second work is related to phishing detection.

1.7.3 Chapter 3 Summary and Evaluation:

Chapter 3 commences with the selection of an AutoML Framework, specifically H2O AutoML, designated for this study. Following this, it delves into the case study chosen to assess three distinct approaches: Human Data Scientist, Generative AI and AutoML. This case study involves a Credit Risk Assessment utilising the renowned German Credit Risk Dataset. Subsequently, it outlines potential Machine Learning Algorithms suitable for this project, including Logistic Regression, Decision Tree and Random Forest. Furthermore, the chapter indicates the adoption

of a CRISP-DM methodology, albeit not adhering strictly to its specific processes. Finally, it concludes with a compilation of Machine Learning Best Practices along with why they are important.

1.7.4 Chapter 4 Summary and Evaluation:

Chapter 4 starts by laying out an algorithm designed specifically for assessing credit risk, presented in a flowchart format. Following that, the chapter dives into a detailed explanation of each of the best practices in machine learning, tailored specifically for the credit risk assessment case study. For each best practice, possible solutions are presented in a platform-independent way on how to perform the best practice. Occasional mention is made on how H2O AutoML implement a solution but without going into system specifics. The proposed user interface design is introduced, offering users a way to input data and see the predicted outcomes ('Likely to Default' or 'Unlikely to Default'), along with explanations for these predictions. Additionally, the design of the deployment setup is also presented, with an API acting as an intermediary between the user and the backend Model. Wrapping up, the chapter provides a detailed rundown of the five machine learning algorithms selected for use within the Human Data Scientist approach: Logistic Regression, Decision Tree, Random Forest, Gradient Boosting Machine and Neural Networks. Note that the specific machine learning algorithms for the other two approaches (Generative AI and AutoML) are intentionally left unspecified, allowing them the flexibility to choose the most suitable models for the credit risk assessment case study.

1.7.5 Chapter 5 Summary and Evaluation:

The selected case study (Credit Risk Assessment a.k.a Loan Default Prediction) was implemented using 3 different approaches namely, H2O AutoML, ChatGPT Data Analyst (CDA), and Human Data Scientist. To ensure that the case study implementations are of substance and good quality, all 3 approaches were required to implement a list of Machine Learning Best Practices using a process closely resembling the CRISP-DM methodology. For each approach, the Machine Learning Model implementation is covered first followed by the User Interface & API Integration implementation and System Integration Testing. All 3 approaches were able to implement the case study and ML best practices successfully but there were some differences in some situations.

1.7.6 Chapter 6 Summary and Evaluation:

Chapter 6 discusses the deployment of the best models from three approaches: Human Data Scientist, AutoML, and Generative AI. The deployment is done locally using Streamlit as the client and FastAPI as the server. Each approach typically requires the following components: files for Streamlit, FastAPI, the ML model and the Scaler object. The deployment involves running both Streamlit and FastAPI in separate windows, allowing users to input data, receive predictions, and view explanations via a web interface. Local deployment is chosen for ease of

testing and demonstration, avoiding the complexities of network configurations. The communication between Streamlit and FastAPI is established through HTTP requests, with FastAPI handling predictions and explanations. Challenges faced during deployment include dependency management, ensuring communication between services, debugging and deploying a KerasClassifier model for the Human Data Scientist approach, which required unconventional methods due to limited documentation. Lastly, the chapter also details the step-by-step deployment procedure and how the components interact to provide predictions and explanations to the user.

Chapter 2

Literature Review

2 Literature Review

There have been studies done to evaluate AutoML, as well as ChatGPT Data Analyst (CDA), to compare their performances against the human data scientist. Many of these studies were done to see how well AutoML or CDA perform in a particular domain (e.g. clinical trials, storm prediction, industrial process control, phishing detection). Although this current study uses a credit risk assessment case study, the focus is not so much on how well AutoML or CDA perform in this domain. This study takes a more generic (less domain-centric) approach, which is to evaluate how well AutoML or CDA perform in regard to established machine learning best practices.

2.1 Background of ChatGPT Data Analyst (CDA)

2.1.1 What is ChatGPT Data Analyst (CDA)?

Formerly known as Advanced Data Analysis (and before that, Code Interpreter), CDA is a plug-in functionality integrated into GPT-4 within ChatGPT, enabling users to directly upload files for data analysis (MIT, 2024). Exclusively accessible to paid accounts (ChatGPT Plus and Enterprise), this capability empowers users to perform data analysis tasks directly in ChatGPT using only natural language prompts. Behind the scenes, CDA will execute the necessary program code to perform the data analysis tasks. This tool is particularly beneficial for users seeking to delve into data exploration, code creation and problem-solving with the support of AI tools (MIT, 2024). CDA is compatible with various file formats, encompassing text and image files, complete documents like PDFs, code or other data files (MIT, 2024). While CDA operates through Python, it relies on the ChatGPT model underneath, which possesses the capability to comprehend and interpret various programming languages. Consequently, it can adeptly translate code across different programming languages or comprehend files written in languages other than Python (MIT, 2024).

2.1.2 ChatGPT Data Analyst (CDA) Features

In the realm of **Data Analysis and Visualization**, utilising CDA enables you to upload a dataset and direct it to identify errors or outliers. Subsequently, you can instruct CDA to cleanse the data or execute various transformations on it. You can pose general inquiries about the data, such as identifying the top-performing salesperson for the month, and CDA will generate and execute the necessary Python code accordingly. Additionally, you can request CDA to generate data visualizations, and following their creation, you can instruct CDA to make adjustments, such as altering colors, fonts, or annotating images differently.

According to a journal article discussing the potentials and limitations of employing the ChatGPT Data Analyst (CDA) plugin for hydrological analyses (Irvine et al., 2023), researchers noted a

significant **time-saving** advantage in utilising CDA due to its ability to swiftly establish initial scripts, thereby expediting the data analysis process. Although user intervention is typically necessary for script generation, ChatGPT serves as a helpful preliminary step (Irvine et al., 2023). Users are not required to write any code, which proves advantageous for individuals with minimal or no scripting expertise, enabling them to conduct data analyses solely through natural language prompts.

In the broader domain of **Code Development and Testing**, you have the capability to direct CDA to assess your code by uploading the corresponding file (e.g., .py, .js, .html, .css), whereby CDA will identify and highlight any errors detected. Alternatively, you can sketch an image, upload a photo of it and task CDA with generating the HTML, CSS and JS code. Furthermore, CDA excels in solving mathematical problems effortlessly. You can upload screenshots containing equations or formulas, and CDA will furnish accurate solutions (Paterson, 2024).

Regarding **File Conversion**, CDA exhibits the ability to convert files from one format to another. For instance, you could upload an .mp3 audio file for transcription or provide a dataset and request CDA to generate a spreadsheet from it or transform an article into a PowerPoint presentation. Additionally, you can enlist CDA to modify the contents of files, such as rearranging or sorting data. Furthermore, CDA can be utilized for image, sound and video editing (Paterson, 2024).

One of the major grievances regarding ChatGPT was its **inability to access the internet** for up-to-date information. However, this issue has been addressed in the ChatGPT Plus account. With the addition of the "Browse with Bing" feature, ChatGPT can now retrieve current information from the web whenever it deems recent context necessary (OpenAI, 2024b).

2.1.3 ChatGPT Data Analyst (CDA) Limitations

One of the limitations of CDA is that it **only supports Python**. Moreover, it is **not possible to install new external packages** into its environment. Hence, the tasks it can solve are limited by the libraries it supports. WVU researchers (West Virginia University, 2023) conducting research on bioinformatics complained of lack of support for specialised software specific to bioinformatics in CDA. CDA cannot connect to databases which means the necessary data must first be retrieved/downloaded and then uploaded to CDA as CSV or Excel files.

Another limitation of the CDA tool pertains to the file **upload size limit**, which can diminish its applicability in many situations involving large datasets such as the hydrological scenarios reported by (Irvine et al., 2023). According to OpenAI, the file size limit is 512MB but some users reported that in reality it is only as high as 170MB (Sha, 2023).

Even upon successfully loading a sizable file, CDA lacks the necessary CPU power to effectively process it. The **absence of parallel processing support** for extensive datasets renders CDA incapable of handling high-performance computing tasks. Additionally, each execution cell is subject to a time limit, automatically halting execution if the code surpasses this threshold. Consequently, complex operations on large files are unfeasible. Attempting computationally intensive tasks or processing large jobs may lead to timeouts or failures (Yosifova, 2024).

As many have found out, ChatGPT/CDA doesn't consistently deliver desired outcomes and may occasionally generate errors. When the data is organised efficiently and the task is clear-cut, CDA tends to make fewer errors, though they can still arise. The most effective approach is to request CDA to display results after each step, enabling you to identify and clarify any errors promptly. Through this process, the AI can self-correct and rectify them. It can become challenging to troubleshoot code without the comprehensive error messages typically offered by an Integrated Development Environment (IDE) (Yosifova, 2024).

Another limitation lies in **state management**. If the conversation extends too long or encounters interruptions, the interpreter may lose session state, necessitating the user to reload data or redefine variables or functions. The underlying runtime environment can be volatile at times, with previously defined variables potentially disappearing and previously generated code being forgotten, necessitating reworking of prior tasks. At times, CDA can appear disoriented and fails to complete its tasks. In essence, for complex tasks, CDA may require multiple nudges/prompts to guide it to accomplish a task accurately. In environments where rapid results are frequently required, this behaviour could pose a significant drawback. There are numerous complaints about this behaviour of ChatGPT in the OpenAI Developer Forum (CodingTheSmartWay, 2023).

A significant concern when uploading data to CDA is the lack of assurance regarding **privacy and security**. Data analysis should be forbidden if there's no guarantee that company or client data will remain safe from breaches and cyberattacks. Consequently, numerous large corporations have opted to prohibit employees from using ChatGPT/CDA in order to safeguard their data. Entrusting financial information and customer data to CDA can pose a risky proposition (Piltch, 2023).

Even after OpenAI upgraded ChatGPT 3.5 to 4.0, **hallucinations** still persist. Hallucination is a term used to describe unexpected or nonsensical outputs generated by artificial intelligence systems, particularly those based on generative models like GPT (Generative Pre-trained Transformer) models. They can occur due to various reasons, including lack of context in the

input, lack of understanding of complex concepts, overfitting and noise in the data (DigitalOcean, 2024; BIAŁOZOR, 2023).

2.2 Works Related to the Use of ChatGPT Data Analyst (CDA)

2.2.1 Building a Simple Model to forecast the Beginning of Ionospheric Storms from a Ionosonde dataset

Gillett (2023) evaluated the ChatGPT Data Analyst (CDA) tool, highlighting its proficiency in data loading, interpretation, analysis, and recommendation presentation. However, CDA showed memory limitations, especially in recalling previously generated code, often requiring rewriting. Despite this, users could retrieve code segments from chat history. Gillett prompted CDA through various data analysis tasks, including data visualization, predictive modeling, and performance evaluation. CDA encountered challenges in rendering interactive plots and generating models with satisfactory performance metrics, prompting human intervention to rectify errors and improve results. While CDA demonstrated data analysis capabilities, it functioned more as a collaborative aid than a standalone solution, highlighting the continued need for human expertise in data science tasks.

Analysis: CDA cannot completely replace the human data scientist, who is still needed to perform the fine-tuning and to guide CDA in the performance of tasks.

2.2.2 LLMS Enhance Automated Machine Learning in Clinical Research

Arasteh et al. (2023) emphasised the transformative role of Large Language Models (LLMs) in integrating data science with clinical research, showcasing their ability to simplify complex machine learning (ML) methods and expand their applicability in medicine. Their analysis of four extensive clinical trials revealed LLMs' potential to enhance, rather than replace, specialised training and resources, thereby democratising access to advanced data processing in healthcare.

Positive highlights of the study include the consistent performance of models selected by ChatGPT Data Analyst (CDA), matching or surpassing manually crafted ML techniques across diverse datasets. CDA demonstrated the capacity to independently select, train and optimize effective ML models, implementing techniques for model robustness and generalizability, such as regularization and model selection based on validation.

However, the study also identified limitations, including uncertainty about CDA's performance on less curated datasets requiring sophisticated statistical approaches. Discrepancies in performance metrics between CDA and original studies, despite using the same model, raised questions about data preprocessing, dataset splitting and model configurations. Additionally, the

study highlighted the influence of the quality of prompts on the model's performance, suggesting potential variations based on different prompts.

Analysis: CDA is highly capable in performing ML tasks independently, even exceeding the performance of human data scientists. However, the authors did qualify this assessment by stating that the results may possibly be different if the dataset was less curated and if the quality of prompt engineering is not as good.

2.3 Background of AutoML

2.3.1 What is AutoML?

AutoML involves automating different aspects of developing machine learning models, making machine learning more accessible for individuals and organisations lacking expertise in machine learning and data science (Amanpriyanshusms, 2024). It encompasses techniques and tools that automate the selection and fine-tuning of machine learning models. The objective of AutoML is to simplify the ML development process, enabling those with limited data science knowledge to construct and implement high-performing machine learning models (Amanpriyanshusms, 2024). Hence, the adoption of AutoML streamlines the development process, enabling the generation of the required code for initiating machine learning model development with just a few lines of code (IBM, n.d.). Some AutoML frameworks also exhibit significant potential in delivering solutions in regulated industries by offering results that are both explainable and reproducible (IBM, n.d.).

AutoML systems have the capability to automate various stages in the construction of a machine learning model, as listed below (OpenAI, 2024a). Note that not all AutoML frameworks support all the ML automations listed below. For example, few open source AutoML frameworks support the automation of Deployment and Model Serving and also Monitoring and Maintenance. These features are only supported by cloud providers AutoML frameworks and commercial AutoML frameworks like DataRobot and H2O.ai Driverless AI. Moreover, some AutoML frameworks focus only on Neural Networks and Deep Learning where as others focus mainly on the traditional ML algorithms. Below are some of the automations that AutoML frameworks can perform (Salehin et al., 2024; J., 2023; OpenAI, 2024a):

- **Data Preprocessing**

When it comes to handling missing values, AutoML frameworks can automatically impute missing values or apply suitable strategies for handling them. During data cleaning, these frameworks identify and handle outliers, noise, or inconsistencies in the dataset.

- **Feature engineering**

Feature engineering is another aspect where AutoML shines, as it automatically generates new features or transforms existing ones to improve model performance.

- **Model Selection and Configuration**

AutoML frameworks take care of algorithm selection by automatically choosing appropriate machine learning algorithms based on the dataset characteristics and the problem type (e.g., classification, regression).

- **Hyperparameter Tuning**

Hyperparameter optimization is another crucial aspect that AutoML handles, automatically tuning model hyperparameters to achieve optimal performance.

- **Model Architecture Search**

Some AutoML frameworks can explore and select optimal neural network architectures for deep learning tasks through model architecture search.

- **Model Training and Cross-Validation**

In terms of model training, AutoML frameworks automate the process by training multiple models with different configurations without manual intervention. They also handle cross-validation, performing this technique to estimate model performance and generalize well to unseen data.

- **Ensemble Methods**

AutoML frameworks can automatically create model ensembles to combine predictions from multiple models and improve overall performance.

- **Model Evaluation**

AutoML frameworks automatically calculate various performance metrics to assess model quality (e.g., accuracy, precision, recall, F1-score) during performance evaluation.

- **Model Interpretation**

AutoML frameworks can also provide insights into model decisions, feature importance, and potential biases through model interpretation.

- **Deployment and Model Serving**

Deployment of trained models into production environments can be automated by AutoML frameworks, streamlining the process. Some AutoML frameworks also provide APIs or endpoints for making predictions with deployed models, ensuring accessibility.

- **Scalability**

Scalability ensures that deployed models can handle high volumes of requests efficiently. Quite a few AutoML frameworks provide support for parallel processing, enabling multiple

tasks to be executed simultaneously. For example, during hyperparameter optimization or model training, parallel processing allows different configurations or subsets of data to be processed concurrently, reducing overall computation time. Many AutoML frameworks leverage distributed computing to handle large datasets and complex computations. They can distribute tasks across multiple nodes or machines, allowing for parallel processing and faster execution of machine learning tasks.

- **Monitoring and Maintenance**

Some AutoML frameworks can continuously monitor model performance and detect drift or degradation in model quality over time through model monitoring. Additionally, they automate the process of retraining models with new data to keep them up-to-date and maintain their accuracy.

- **Documentation and Reporting**

AutoML frameworks generate reports or documentation summarizing the entire machine learning process, including data preprocessing steps, model selection, and evaluation results, thus automating documentation. They also automatically create visualizations to help understand the data, model performance, and other relevant aspects, aiding in visualization.

2.3.2 A Survey of Open Source AutoML Frameworks

Below we present the major open source AutoML tools that are currently being used with their pros and cons.

- **Auto-Sklearn**

Auto-Sklearn, an early AutoML framework developed as an extension of Scikit-Learn in Python, gained popularity for its user-friendly nature and integration with Scikit-Learn. It offers ease of use, time-saving through quick model iteration, and leverages ensemble methods and meta-learning for high-performing models. Scalability, internal cross-validation, and prevention of overfitting are additional advantages. However, it lacks customization for advanced users, is resource-intensive, and produces black-box models, limiting interpretability. It also doesn't cover every algorithm or technique, such as deep learning and time series (Bahmani, 2023).

- **AutoKeras**

AutoKeras, developed by researchers at Texas A&M University, is a deep learning-focused AutoML tool based on the Keras library and TensorFlow framework. It offers a user-friendly interface, strong support for deep learning tasks, and uses Neural Architecture Search (NAS) techniques to find optimal neural network architectures. Users can define custom search spaces for flexibility. However, it only supports deep learning tasks and produces black-box models, which can be resource-intensive and lack interpretability. Additionally, while it

offers some customization, it may not satisfy advanced users seeking more control over workflows (Jin et al., 2024).

- **H2O AutoML**

H2O AutoML, a part of the H2O-3 platform developed by H2O.ai, is a robust automated machine learning (AutoML) tool known for its user-friendly interface and broad support for various ML tasks. It leverages state-of-the-art algorithms, ensemble methods, and meta-learning for optimized model building. H2O AutoML efficiently scales to large datasets and distributed environments, offering tools for model interpretation and explainability. However, it has limited customization, can be computationally intensive, and may pose challenges in understanding complex ensemble models. Users also need to learn the H2O ecosystem for effective utilization (H2O, 2024b).

- **PyCaret**

PyCaret offers a low-code interface, enabling users to achieve complex machine learning tasks with minimal coding and manual configuration. It supports a wide range of tasks such as classification, regression, and NLP, with scalability to large datasets. PyCaret facilitates model deployment and provides interpretability tools. However, it lacks customization for advanced users and is resource-intensive. Additionally, it focuses on traditional ML algorithms and does not directly support neural networks and time series (Ali, 2023).

- **TPOT**

TPOT, the Tree-based Pipeline Optimization Tool, automates ML pipeline optimization using genetic programming. It offers a simple interface suitable for users of varying ML expertise and supports parallel processing for faster exploration. Leveraging genetic programming and ensemble methods, TPOT builds highly optimized models, providing insights into the generated pipelines for improved model interpretation. However, it can be resource-intensive, especially for complex tasks, and lacks customization compared to manual workflows. Additionally, TPOT primarily focuses on traditional ML algorithms, which may limit its effectiveness for tasks requiring deep learning architectures (Shu, 2019).

- **AutoGluon**

AutoGluon, an open-source AutoML toolkit from the MXNet team at AWS, provides a user-friendly interface for automating ML pipelines. It supports various tasks like classification, regression, object detection, and NLP, with flexibility in defining custom search spaces. Leveraging state-of-the-art algorithms and ensemble methods, AutoGluon builds optimized models. However, it's computationally intensive, requiring significant resources, and despite offering interpretability tools, understanding complex ensemble predictions can be

challenging. Additionally, while it's flexible, advanced users may find its customization options limited, and beginners might face a learning curve (Yin, 2022).

- **Ludwig**

Ludwig, an open-source AutoML library by Uber, enables building and training deep learning models without code. It supports various ML tasks including classification, regression, and sequence modeling, also offering interpretation tools. Ludwig automates feature engineering and integrates deep learning with traditional ML algorithms. It's user-friendly with YAML configuration files, but lacks customization compared to manual workflows. Users must understand YAML syntax. While leveraging TensorFlow, Ludwig's performance may vary and training can be resource-intensive. Users still require basic deep learning knowledge (Uber, 2019).

2.4 Works Related to the Use of AutoML

2.4.1 Automated ML for Predictive Quality in Production

The study conducted by Krauß et al. (2020) explores the potential of AutoML in industrial settings through a case study focused on optimizing the efficiency of a complex process chain comprising six sequential processes. Despite the non-optimized Random Forest model outperforming the AutoML-generated model in terms of F1-Score, using AutoML required significantly less expertise and effort compared to traditional manual approaches. While concerns about overfitting with Auto-sklearn have been raised, the study's learning curve showed no signs of overfitting, suggesting its suitability for prolonged use. Auto-sklearn emerges as a viable option for modeling tasks where minimal machine learning knowledge exists and improvements beyond baseline models are sought. Moreover, it can serve as a useful starting point for data scientists, facilitating manual implementation and subsequent refinement of solutions. The authors propose that augmenting AutoML systems with additional data preparation procedures and domain-specific expertise could lead to further improvements. Auto-sklearn does not provide direct support for model deployment.

Analysis: AutoML is useful for establishing baseline models but beyond that, human domain expertise is required.

2.4.2 Man versus Machine: AutoML and Human Experts' Role in Phishing Detection

Purwanto et al. (2021) discusses the development of machine learning models using both AutoML frameworks and traditional manual methods within the context of phishing detection

systems. The emergence of AutoML frameworks offers potential solutions, raising questions about their efficacy compared to manually trained models and their ability to assist non-experts. Experiments were performed to examine the efficacy of six prominent AutoML frameworks against manually constructed machine learning models across ten distinct phishing datasets. The findings suggest that AutoML-based approaches surpass manually engineered models in intricate classification scenarios, particularly in datasets characterized by less discriminative features, overlapping classes, or significant non-linearities. Despite AutoML's ability to produce competitive models, challenges remain in handling unlabeled data and incremental updates, emphasizing the continued need for human expertise in the detection pipeline.

Analysis: Models generated by AutoML are superior to handcrafted ones in complex classification tasks. The authors qualify this assessment by stating that the study was mainly focused on supervised classification.

2.5 Evaluation

From the above literature reviews on the use of ChatGPT Data Analyst (CDA) for data science tasks, the majority of the feedback is that this generative AI technology is still not yet fully mature. On the usage side, ChatGPT/CDA still suffer from quite a number of glitches such as providing answers which are not quite right, inconsistent behaviour, instability, hallucinations, and limited software libraries and programming languages support. However, when things go smoothly, CDA can produce quite remarkable results as reported in one of the literature reviews above. Therefore, given time for OpenAI to resolve these glitches, the use of CDA as a data analysis tool is very promising.

On the infrastructure side, ChatGPT in general is not scalable as it does not support parallel and distributed processing. Upgrading from ChatGPT Plus to the ChatGPT Enterprise version is supposed to increase processing speed by 2 times but still this is not enough for processing large datasets. However, regarding data privacy and security, OpenAI claimed that the ChatGPT Enterprise version is Service Organisation Control Type 2 (SOC 2) compliant (OpenAI, 2023). SOC 2 is a cybersecurity framework, developed by the American Institute of Certified Public Accountants (AICPA), that ensures third-party service providers store and process the client's data securely. SOC 2 certification showcases an organization's commitment to safeguarding data and maintaining system integrity, reducing the risk of security breaches. However, SOC 2 compliance is not a blanket assurance of security (Sangfor, 2023).

From the above literature reviews on the use of AutoML for data science tasks, the feedback is that due to automation provided by AutoML, a ML project can get up to speed very fast. However, it should be noted that default processing (assumptions made by automation) may not always be the ideal solution in which case input from a data scientist may be necessary. In other words, if the ML task is

straightforward and the common variety which does not require a data scientist to step in, then AutoML would be a big advantage not just because it obviates the need for technical expertise but also provides substantial time savings. The literature review also revealed that, in certain scenarios, AutoML-generated models approach can outperform human-crafted models. This is especially so in areas like model training and selection as well as hyperparameter tuning where AutoML can efficiently search through a wide range of models and hyperparameters to find the best-performing ones. This process can be time-consuming and subjective for human data scientists, who may not have the expertise or resources to explore every possible combination thoroughly. Having said that, it should be noted that, unlike ChatGPT/CDA, AutoML is built for a limited and special purpose which is to automate the ML process. Therefore, AutoML would not be able to assist in providing advice on areas like the business domain, which ChatGPT/CDA and the human data scientist can.

Chapter 3

Methodology and Requirements Analysis

3 Methodology and Requirements Analysis

The goal of this study is to answer the Research Question:

How do the 3 approaches (human Data Scientist, AutoML, Generative AI) compare in terms of the execution of established Machine Learning Best Practices?

Before moving into the Machine Learning Best Practices, I will first perform a survey of the major open source AutoML frameworks to select the AutoML which I think is the best one to use in this study.

3.1 Selection of AutoML Framework

For this study, I will select only 1 open source AutoML. From the list of AutoML frameworks in the previous chapter in section 2.3.2, I can see that a couple of them are mainly focused on neural networks and deep learning, namely AutoKeras and Ludwig. For this comparison study, I am not specifically concerned with deep learning problems only. I want to consider and explore all the major ML algorithms including deep learning. As AutoKeras and Ludwig do not have extensive support for the traditional ML algorithms, I shall omit them from selection.

Since my case study (Credit Risk Assessment) involves regulatory compliance, it is important for our AutoML to have built-in features for explainability and interpretability. Auto-Sklearn does not provide built-in features specifically dedicated to explainability and interpretability so I shall omit it from selection.

A very important consideration, in the selection of AutoML for this study, is the comprehensiveness of algorithms support. PyCaret and TPOT have no and weak support for neural networks deep learning respectively. Therefore, I shall omit them from selection.

I am left with AutoGluon and H2O AutoML. Both have comprehensive support for a variety of machine learning algorithms. Both are designed to scale to large datasets and distributed computing environments. Performance may vary depending on the dataset, the task and the specific configurations used. According to a study by Ferreira et al. (2021), AutoGluon performed well for multi-class classification in general ML but H2O AutoML performed well for the binary classification and regression in deep learning. AutoGluon also did well for regression in deep learning. Hence it is not an easy decision to choose either AutoML. In the end, the deciding factor was the availability and quality of the documentation for the two AutoML tools. There are at least 3 published books on H2O AutoML but none exclusively on AutoGluon. Online resources for H2O AutoML also seem to be more extensive. Hence, the AutoML that I will use in this comparison study is **H2O AutoML**

The high-level architecture of H2O AutoML, which is shown in Figure 3.1, consists of 2 main layers which are the Client layer and JVM (Java Virtual Machine) layer.

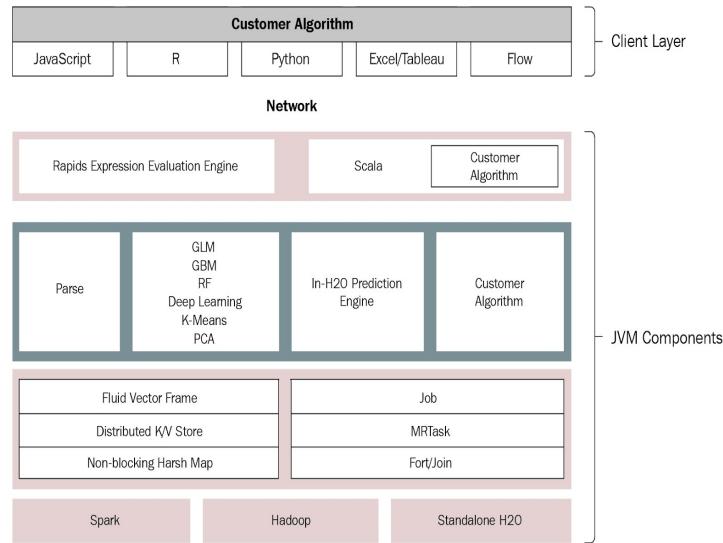


Figure 3.1. H2O AI high-level architecture. From *Practical Automated Machine Learning Using H2O.ai* (p. 122), by Salil Ajgaonkar, 2022, Livery Place, Birmingham: Packt. Copyright [2022] by Packt Publishing.

One of the clients is H2O Flow, which is the web user interface that non-technical users can use to configure the whole ML lifecycle. The different clients communicate with a single H2O Server as shown in Figure 3.2 below.

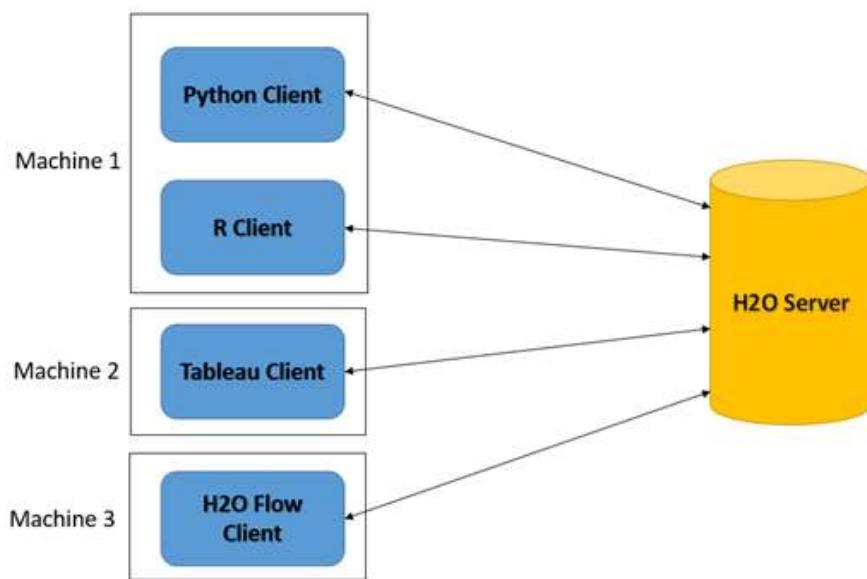


Figure 3.2. Different clients communicating with the same H2O server in the Development Environment.
From *Practical Automated Machine Learning Using H2O.ai* (p. 124), by Salil Ajgaonkar, 2022, Livery Place, Birmingham: Packt. Copyright [2022] by Packt Publishing.

The H2O Server is made up of several JVM components. Note that H2O AutoML is capable of supporting cluster computing as the JVM layer lies on top of the Hadoop and Spark cluster processing engines.

3.2 Case Study Approach

In order to evaluate the 3 methods (human Data Scientist, AutoML, Generative AI) on their execution of ML best practices, I will need to develop and implement a concrete ML model. The case study we have chosen to implement is Credit Risk Assessment, in other words loan default prediction. This is Supervised Classification machine learning.

There are several reasons for choosing Credit Risk Assessment as our case study. Firstly, it is a well-defined problem. This clarity in problem formulation allows for a straightforward evaluation of ML models based on metrics such as accuracy, precision, recall, and F1-score. Secondly, credit risk assessment is a critical task in the financial industry with significant real-world implications. Thirdly, credit risk assessment datasets often exhibit class imbalance, with a small proportion of defaults compared to non-defaults. This provides an opportunity to evaluate ML best practices for handling class imbalance. Fourthly, Credit risk assessment raises ethical considerations related to fairness, transparency and bias mitigation. Evaluating ML best practices in this context allows for assessing techniques for ensuring fairness in lending decisions, avoiding

discriminatory outcomes, and promoting ethical AI. Fifthly, credit risk assessment models require continuous monitoring and adaptation to changing economic conditions, market dynamics and borrower behaviour. Evaluating ML best practices in this context involves assessing techniques for model monitoring, updating and retraining to maintain model performance over time. Finally, high-quality datasets for credit risk assessment, such as the German Credit Risk dataset, are readily available for research and experimentation. These datasets contain a diverse range of features related to borrowers' demographics, financial history, and credit behaviour, allowing for comprehensive model development and evaluation.

The dataset chosen for this study is the German Credit Risk dataset which has been widely used, in academic research and industry applications, as a benchmark for evaluating the performance of different machine learning algorithms and techniques in credit risk assessment. This realistic dataset contains information about credit applicants and their credit risk profiles. It includes features such as demographic information (age, gender, marital status, etc.), financial information (income, existing liabilities, savings, etc.), credit history (past credit performance, number of existing credit lines, duration of credit, etc.), employment information (employment status, job stability, etc.) and other features, such as purpose of the loan, property ownership, etc. An important reason this particular dataset was chosen (instead of for e.g. Lenders Club dataset) is its size (one thousand records). Our chosen AutoML, i.e. H2O.ai AutoML, can be computationally intensive, especially for large datasets.

3.3 Machine Learning Algorithms

The Credit Risk Assessment Model's choice of ML algorithms depends on the problem's nature, which in this case is a supervised binary classification task. *Logistic Regression* is a classic algorithm which is specifically designed for binary classification tasks, where the target variable has two possible outcomes. It provides understandable results and can handle non-linear relationships through feature engineering. Using regularisation techniques, it can handle multicollinearity and overfitting. It is computationally efficient, relatively fast to train and can handle datasets with many features. *Decision trees* and their ensemble methods, like *Random Forests*, are effective for modelling complex relationships and providing transparent, interpretable models. Random forests, in particular, are known for their high accuracy, robustness, and feature importance estimation. Another ensemble method, *Gradient Boosting Machine (GBM)*, iteratively improves model performance by sequentially fitting weak learners to minimise a predefined loss function. GBM is scalable, interpretable and provides high predictive performance. In this study, we will not be using the *XGBoost* algorithm because H2O.ai AutoML requires GPU to run XGBoost. Finally, Multi-Layer Perceptron (MLP) and Deep Learning *Neural Networks* are capable of capturing complex non-linear relationships but

require a larger amount of data and computational resources. The downside is that they are often considered black-box models, lacking interpretability. In conclusion, the nature of the dataset, including the complexity of relationships and available resources, will determine the choice of algorithms.

3.4 CRISP-DM Methodology

Before I get started on the list of machine learning best practices, I first need to place these ML best practices within the appropriate phases of the Machine Learning Life Cycle. For this study, I have chosen the CRISP-DM process model (Figure 3.3) to represent the Machine Learning Life Cycle but I will not strictly adhere to it 100% (I will delve into the details in the System Design section). CRISP-DM was chosen because it is a widely accepted and robust framework for guiding machine learning and data science projects. It provides a structured approach to ensure that ML projects are well planned, executed efficiently, and deliver successful outcomes aligned with business objectives. Note however that, in this study, there will be a variation. In the approach adopted, Data Understanding and Data Preparation will intermix with each other, not strictly following each other. Maybe both areas should be combined and called “Data Pre-Processing”.

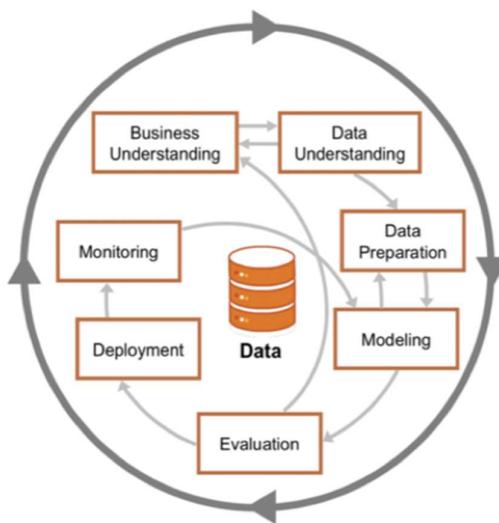


Figure 3.3. CRISP-DM. From *Understanding CRISP-DM and its importance in Data Science projects* by Z. Luna, 2021, Medium. <https://medium.com/analytics-vidhya/understanding-crisp-dm-and-its-importance-in-data-science-projects-91c8742c9f9b>. Copyright 2021 by Zipporah Luna.

3.5 Machine Learning Best Practices

In this section, I will list the Machine Learning Best Practices that will be implemented by each of the 3 approaches (human Data Scientist, AutoML, Generative AI). By implementing established Machine Learning Best Practices, this case study ensures a fair comparison across all approaches. For each ML best practice, I will explain why it is important in Machine Learning. First, let's define what a *best practice* is. A best practice refers to a method or technique that is widely recognised as the most effective and efficient way to achieve a particular outcome within a specific domain. Best practices are typically established through experience, research, and consensus within a particular industry, field or discipline.

Machine learning best practices refer to guidelines, methodologies, and approaches that are recognised as effective for developing, deploying, and maintaining machine learning models in a reliable, efficient, and ethical manner. These best practices encompass various stages of the machine learning lifecycle, including data preprocessing, model development, evaluation, deployment and monitoring.

For each Machine Learning Best Practice used for evaluation, an explanation will be given why this best practice is important in Machine Learning to justify its use in this comparison study. Each approach (human Data Scientist, AutoML, Generative AI) will be evaluated based on the quality of the solution and also how efficiently it implemented the best practice. The pros and cons of the 3 approaches in implementing the best practice will also be discussed.

1. Identify the Business Problem and Define Clear Objectives

Understanding the intricacies of business operations is essential within the realm of Data Science as it serves to connect technical solutions with practical business challenges. This connection ensures that Machine Learning (ML) projects yield tangible benefits for organizations by aligning them with strategic business objectives, thereby fostering outcomes that are both impactful and successful (AWS, 2024).

2. Preliminary Data Understanding

The focus here, at this stage, is more on the general characteristics of the dataset such as dataset size, number of features, data types and summary statistics. It is also important to know and understand the meaning of all the features in the dataset. The purpose is to get an initial feel for the data to be modelled.

3. Identify Data Quality Issues

Identifying data quality issues, such as missing values, outliers, duplicates, inconsistencies, and errors, is essential for maintaining data integrity, ensuring model performance, enhancing interpretability, mitigating bias, supporting decision-making, improving resource efficiency and ensuring compliance and governance (Gudivada et al., 2017).

4. Remove Duplicate Data

Removing duplicates from datasets is important to maintain data integrity and accuracy. This process helps prevent biases and inaccuracies, enabling more accurate and efficient ML models to be built. By ensuring the quality of the data, these models can generalise well to new data and support reliable decision-making processes (Gudivada et al., 2017).

5. Fix Data Inconsistencies

Maintaining consistency in field or feature values within machine learning holds significant importance for multiple reasons. Reliable and uniform values contribute to robust input data, enhancing model performance, whereas inconsistencies may introduce disturbances, impeding pattern recognition. Consistent values play a pivotal role in preserving dataset integrity, mitigating the risk of erroneous deductions or biased forecasts. They facilitate the interpretation of model predictions, enabling the attribution of decisions to distinct data attributes. Models trained on consistent data often exhibit greater resilience and improved generalization capabilities. Additionally, consistent values streamline effective feature engineering by unveiling meaningful correlations and assisting in the identification of pertinent features (Lin, 2023).

6. Handle Imbalanced Data

The occurrence of imbalanced data arises when the classes within the target variable lack equal representation, presenting hurdles in classification endeavors. Conventional accuracy metrics may provide misleading insights in such scenarios, prompting the adoption of measures such as precision, recall, F1-score, or AUC-ROC for comprehensive assessment. Many machine learning algorithms exhibit bias toward the majority class, resulting in subpar performance for the minority class. Imbalanced data can skew decision boundaries in favor of the majority class, impeding accurate classification of minority instances. Mitigating class imbalance aids in establishing more equitable decision boundaries and reduces errors associated with the minority class, thereby enhancing model generalization capabilities to unseen data (Kotsiantis et al., 2005).

7. Partition the Dataset

Partitioning a dataset into training, validation, and test sets is indispensable for accurate model assessment, mitigating overfitting, fine-tuning hyperparameters, model selection, preventing data leakage, and evaluating generalisation performance. This practice is fundamental in machine learning, guaranteeing the dependability and resilience of your models (Nassar, 2023).

8. Perform Exploratory Data Analysis (EDA) on Train Set

Exploratory Data Analysis (EDA) encompasses the examination and visualisation of datasets to comprehend their architecture, patterns, interrelationships, and possible anomalies before constructing ML models. It facilitates the acquisition of insights into dataset properties, including feature distribution, summary metrics, the presence of missing values, and outliers, thus enabling informed decisions within the ML pipeline. EDA frequently reveals connections and correlations among features, informing the process of feature engineering for more insightful models. Moreover, EDA influences the selection of ML algorithms and hyperparameters and directs the formulation of model validation strategies. Additionally, it promotes communication and collaboration among stakeholders by effectively conveying significant findings and insights through visual representations and summaries (Sahoo et al., 2019).

9. Handle Missing Values

Addressing missing values is a critical step in data preprocessing for various reasons. The presence of missing values can undermine the integrity of the dataset and result in biased or inaccurate model outcomes. Additionally, many machine learning algorithms are unable to process missing values and may generate errors during analysis. Moreover, missing values can distort statistical properties such as the mean and correlation, thereby skewing data assumptions. Ignoring them by discarding data rows leads to reduced sample size and model reliability, especially if non-random. By addressing missing values, unbiased analysis can be achieved, enhancing the model's resilience to variations in the data. Transparent handling of missing values also improves model interpretability, facilitating a clear explanation of the decisions made (Emmanuel et al., 2021).

10. Handle Outliers

Addressing outliers is essential for the development of precise, dependable, and comprehensible machine learning models. Outliers have the potential to distort statistical properties, resulting in inaccurate forecasts or biased estimations. Certain algorithms are sensitive to outliers, which can hinder the model's ability to generalise to new data. Proper management of outliers enhances the robustness of the model and increases prediction accuracy. Moreover, outliers may violate the assumptions of statistical techniques, obscuring relationships between variables and complicating model interpretation. Removing outliers unveils underlying data patterns, thereby improving interpretability. In certain scenarios, outliers may require transformation to better adhere to the assumptions of statistical techniques or models to be employed later in the analysis. Additionally, outliers may indicate data collection errors or rare events, which can impact the quality of the dataset. Addressing

outliers ensures the integrity of the dataset, mitigates model instability and overfitting, and enhances generalisation performance (Thanmai et al., 2023).

11. Perform Feature Engineering

Feature engineering is essential in machine learning for several reasons. It improves model performance by selecting or creating relevant features that help identify meaningful patterns in the data. This process also enhances model interpretability by focusing on understandable and problem-relevant features, aiding stakeholder trust. Engineers can transform nonlinear data relationships into linear ones, facilitating effective pattern capture by linear models. Incorporating domain knowledge into feature creation captures important data characteristics, making models more robust to variations and noise and enhancing generalisation performance (MSOE, 2024; Btd, 2024).

12. Encode Categorical Features into Numerical values

Transforming categorical values into numerical representations is fundamental in machine learning for various reasons. Numerous algorithms mandate numerical input, thereby broadening the scope of applicable algorithms. Numerical representations facilitate mathematical operations, thereby streamlining computation and pattern recognition. Employing numerical treatment for categorical variables empowers algorithms to capture nuanced relationships, potentially enhancing model performance compared to discrete treatment (Udilač, 2023).

13. Perform Feature Scaling

Feature scaling is crucial for most ML algorithms, excluding tree-based models and frequency-based Naive Bayes. It normalises the range of independent variables, ensuring each feature contributes equally to distance and gradient calculations. Algorithms like gradient descent converge faster with scaled features, as it facilitates larger steps toward the minimum cost function, expediting convergence. Without scaling, features with larger scales may dominate, hindering the learning process. Scaling ensures equal contribution from all features. Additionally, failure to scale features can cause numerical instability or overflow issues in computations, particularly for sensitive algorithms (Shivanipickl, 2023).

14. Perform Feature Selection

Feature selection is of paramount importance in machine learning for multiple reasons. It enhances model effectiveness by prioritising the most informative attributes, thereby facilitating pattern recognition. The exclusion of irrelevant or redundant features mitigates overfitting, a scenario in which the model assimilates noise instead of generalisable patterns. Moreover, feature selection diminishes model complexity, leading to expedited training and inference times, particularly crucial for applications with limited resources or real-time

requirements. Furthermore, it promotes model interpretability by streamlining the feature set, thereby facilitating comprehension and communication of results to stakeholders. Additionally, concentrating on pertinent features fosters model generalisation, resulting in superior performance in real-world scenarios (Theng & Bhoyar, 2023).

15. Choose the Right ML Algorithm

The performance of a machine learning system is greatly influenced by the selection of model architecture and algorithm. Various models exhibit distinct strengths and weaknesses, and opting for the most suitable model for a specific task can result in superior predictive accuracy, generalisation, and overall performance. Model selection facilitates the choice of models that offer both accuracy and interpretability. In certain domains such as banking, regulatory compliance may necessitate the avoidance of black-box models that lack interpretability (Pushkar, 2023).

16. Select Model Evaluation Metrics

Various evaluation metrics may emphasise distinct facets of model performance, including accuracy, precision, recall or computational efficiency. The selection of metrics hinges on the trade-offs between these aspects and the particular needs of your application. Evaluation metrics should be robust to changes in the data distribution. By selecting appropriate metrics, you can accurately assess model performance and make informed decisions (Ghosh, 2024).

17. Perform Cross Validation

Cross-validation offers a more dependable assessment of a model's performance compared to a single train-test split. It mitigates the risk of overfitting by training the model on various data subsets. Through averaging performance across multiple folds, cross-validation furnishes a more resilient estimate of the model's performance. Furthermore, cross-validation frequently aids in pinpointing optimal hyperparameters that optimise performance by evaluating the model across diverse data subsets. It enables the maximisation of available data efficiency by utilising all data points for both training and validation, which proves especially advantageous in scenarios with constrained datasets (Pargent et al., 2023).

18. Perform Regularization

Regularization is a technique used in machine learning to prevent overfitting and improve the generalisation performance of models (Kotsilieris et al., 2022). It introduces a penalty term to the model's loss function, discouraging overly complex models with large coefficients. In L1 (Lasso) regularization, the penalty term can shrink some coefficients to exactly zero, effectively performing feature selection. In L2 (ridge) regularization, the penalty term discourages large coefficients, effectively shrinking them towards zero, spreading the impact of the features more evenly across the model. The choice between L1

and L2 regularization along with the value of the regularization parameter, is typically determined through cross-validation, where different parameter values are evaluated on validation data to select the optimal one. The Early Stopping technique can also be used for models like neural networks to halt training once the model's performance begins declining on the validation set, preventing overfitting. Another regularization technique, which is commonly applied in deep learning, is Dropout, which randomly removes neurons during training, helping the model learn robust features and become less reliant on specific weights (YoavAvneon, 2023).

19. Hyperparameter Optimization

Hyperparameter optimisation helps to improve the performance of models by finding the optimal set of hyperparameters. Hyperparameters control the behaviour and complexity of machine learning models. Tuning these hyperparameters can help prevent overfitting by finding the right balance between bias and variance in the model. A well-optimised model is more likely to perform consistently in real-world scenarios. Different datasets and problems may require different hyperparameter settings. Hyperparameter optimisation allows models to adapt to the specific characteristics of the data (Pargent et al., 2023).

20. Use Pipelines and/or Column Transformers for Preprocessing

Pipelines and Column Transformers are workflows that automate and organise the steps involved in data preprocessing, feature engineering, model training, evaluation and deployment. They promote reproducibility, modularity, scalability, automation, and experimentation, ultimately leading to more efficient and reliable model development workflows (Chouliaras et al., 2023).

21. Track the Model Development Process (MLOps)

Monitoring and logging model evaluation metrics provide insights into the performance of the model during training i.e. whether it is improving or stagnating. It also allows you to compare different experiments, hyperparameter settings, or model architectures to identify the most effective approaches. This practice can recreate and reproduce the exact conditions under which a model was trained, ensuring reproducibility of experiments (Chouliaras et al., 2023).

22. Perform Ensemble Learning

Ensemble learning should be considered when dealing with complex problems and when aiming for a high predictive accuracy. Ensemble methods combine multiple individual models and often produce more accurate predictions compared to individual models. They are less sensitive to noise and variations in the training data and therefore less likely to overfit the data. In short, ensembles can reduce both bias and variance, leading to better

generalisation performance. However, ensembles' higher complexity means they are also computationally-intensive (Naderalvojoud & Hernandez-Boussard, 2024).

23. Model Interpretability and Explainability

Interpretability and explainability offer insights into the decision-making process of a model, thereby cultivating trust in its predictions. In regulated sectors like healthcare or finance, comprehending the rationale behind a model's predictions is imperative for accountability and regulatory adherence. Grasping the most influential features in the model's decision-making and the underlying prediction mechanisms facilitates collaboration between machine learning professionals and domain experts. Domain experts contribute valuable insights into the significance of specific features, enhancing the efficacy of the models. Moreover, interpretable models simplify communication with stakeholders unfamiliar with ML concepts. They empower practitioners to detect and rectify biases within the data or model, thus promoting ethical decision-making (Chouliaras et al., 2023).

24. Ensure Scalability

Scalability is important because ML algorithms often require training on large datasets. Scalability ensures that algorithms can efficiently process and analyse large volumes of data. Many ML models, such as deep learning models, can be highly complex and have millions of parameters. Scalability allows these models to be trained efficiently on large datasets and complex architectures, enabling them to capture intricate patterns and make accurate predictions. Scalable ML algorithms and frameworks enable faster training times by leveraging parallel and distributed computing techniques. This accelerates the model development process, allowing data scientists to iterate more quickly and experiment with different algorithms and hyperparameters (Chouliaras et al., 2023).

25. Model Deployment

Following ML best practices when deploying a ML model is essential for achieving optimal performance, reliability, scalability, security, cost efficiency, compliance, user experience, and continuous improvement (Heymann et al., 2022).

26. Production Model Monitoring

Production model monitoring is important for ensuring the performance, reliability, security, compliance, and business impact of deployed ML models. Monitoring model performance allows organisations to detect signs of degradation, such as declining accuracy or increasing prediction errors, and trigger model retraining or recalibration (Chouliaras et al., 2023).

27. Document the Model

Documentation is important in machine learning to ensure reproducibility, knowledge transfer, maintainability, transparency, collaboration and regulatory compliance. Model documentation should be part of a data science team's best practices (Chouliaras et al., 2023).

3.6 Development Tools and Frameworks

3.6.1 AutoML Approach (H2O AutoML)

H2O AutoML is accessible and usable across various programming languages, tools, and software environments. This includes clients like Python, R, Excel, Tableau and H2O Flow. For this project, I will utilise the **H2O AutoML Python API** and write out every single **Python code** in a **Jupyter Notebook** to interact with the H2O server. This approach provides more flexibility than **H2O Flow** (refer to Figure 3.4)—a web-based interface that runs in the web browser and only provides limited functionality.

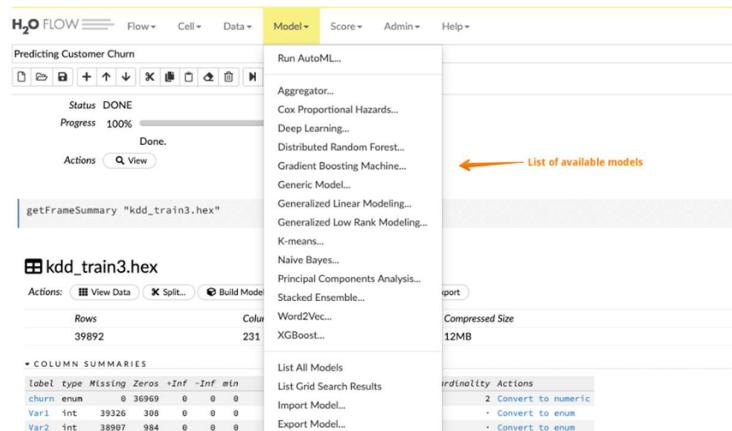


Figure 3.4. Flow Interface. From *Getting started with H2O using Flow* by P. Pandey, 2019, H2O.ai.

<https://h2o.ai/blog/2019/getting-started-with-h2o-using-flow/>. Copyright 2024 by H2O.ai.

3.6.2 Generative AI Approach (ChatGPT Data Analyst)

To interact with ChatGPT Data Analyst, I'll need to upload the German Credit Risk Dataset into **ChatGPT's user interface** and send message prompts to it. Refer to Figure 3.5 as an example of how this interaction works.

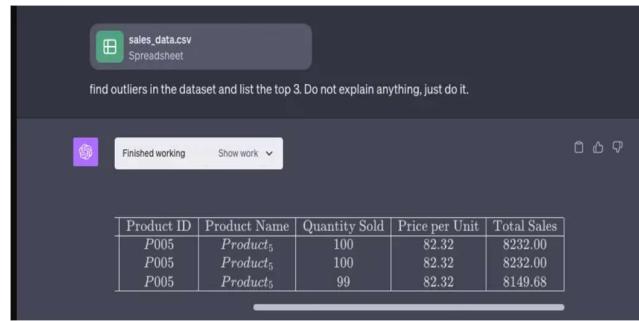


Figure 3.5. Error Checking and Outlier Detection. From *ChatGPT Advanced Data Analysis: Guide & Use Cases* by Akkio, 2024, Akkio. <https://www.akkio.com/post/chatgpt-advanced-data-analysis>. Copyright 2024 by Akkio.

3.6.3 Human Data Scientist Approach

For the human data scientist approach, I'll use Jupyter Notebook and Spyder. Jupyter Notebook offers an interactive platform for coding and visualization, while Spyder is a Python-centric IDE with tools for scientific computing and data analysis. As for the machine learning frameworks, I will be using **Scikit-learn** and **Tensorflow-Keras**. Scikit-learn is Python's user-friendly ML library with a wide array of tools for data analysis and model building. TensorFlow-Keras merges TensorFlow's power with Keras' simplicity, enabling seamless creation and training of deep learning models.

Chapter 4

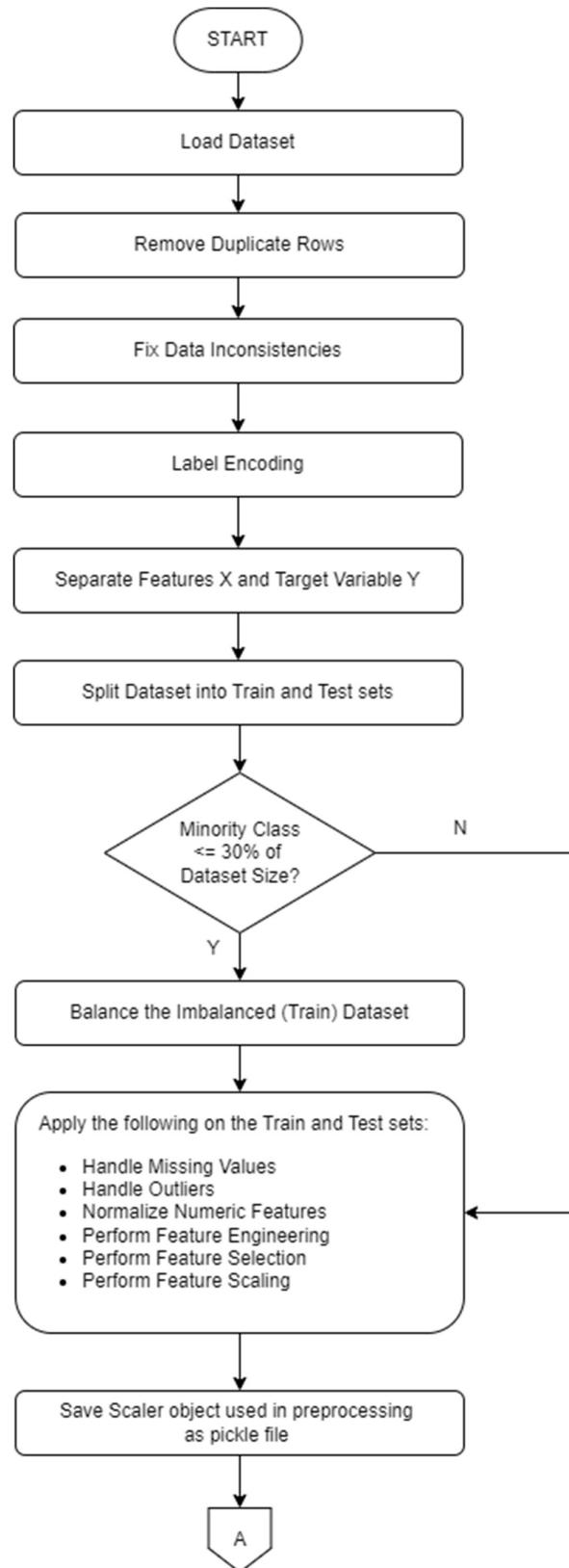
System Design

4 System Design

In this chapter, I will present the algorithm design of the Credit Risk Assessment model. After that, the machine learning best practices introduced in Chapter 3 will be revisited to discuss them from the design perspective. Following that, background information on the ML algorithms used for modelling will be given.

4.1 Algorithm Design for Credit Risk Assessment

The algorithm design for this project can be found in Figure 4.1. Note that this algorithm design is meant for the eventual final version of the script that will be used to create the Credit Risk Assessment model. This is not the ML lifecycle workflow which a data scientist goes through. This is the desired outcome after all the experiments and explorations have been done. As such, the algorithm does not include components like EDA which is an exploratory process. This algorithm design for the script can be considered as the data scientist's design. With this design, if the data scientist wants to add another classifier, it is possible that all that's needed is adding the definition of the new classifier and its hyperparameters to the existing list of classifiers. Note that the order of the various processing steps in this algorithm is significant. This is explained in detail in each individual component. Since I am not yet in implementation stage, how H2O.ai AutoML and ChatGPT Data Analyst will develop the model is still unknown. For example, the classifiers and/or hyperparameters that they will select may be quite different from what a human data scientist uses.



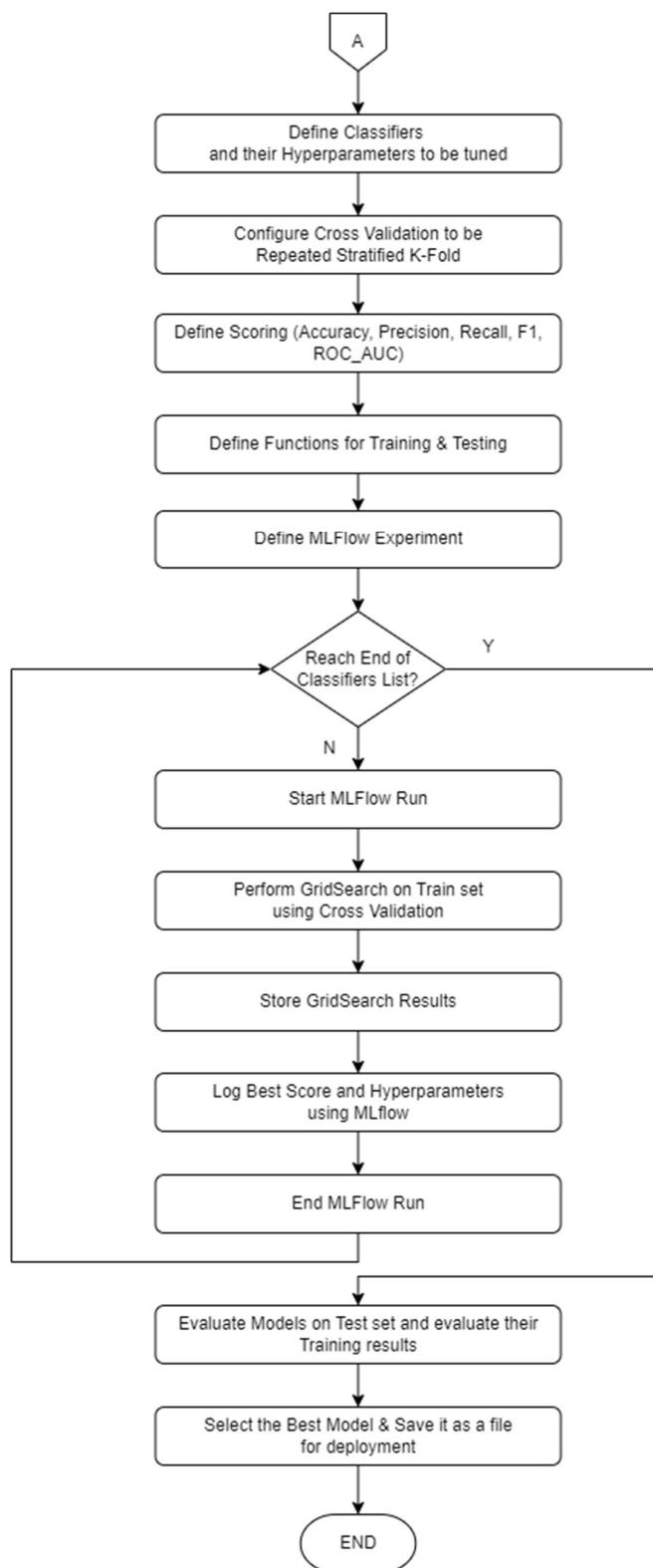


Figure 4.1. Proposed Algorithm Design for Credit Risk Assessment Script.

Notes:

- The 30% or less minority class proportion criteria is not a hard and fast rule. There will be more discussion on imbalanced dataset later under the “Balance the Imbalanced Dataset” best practice.
- From Figure 4.1, you can see that the same preprocessing is applied to both the Train and Test sets.
- The algorithm incorporates MLOps by using MLflow to log experiments. An example of an experiment is the running of a classifier (e.g. logistic regression) on the dataset where metric scores and hyperparameters are logged for later reference.
- The best model from all the runs will be saved in a file for deployment. In addition, in the deployment environment, when new unseen data is input to the model, this data must also undergo preprocessing. This means that the Scaler object which was used on the Train set must also be saved so that it can be used to scale data entered by the user in the deployment environment.

4.2 Machine Learning Best Practices from Design Perspective

In Chapter 3, the Machine Learning Best Practices were discussed to explain why they are important in ML. In this chapter, since this is a chapter on design, for each ML best practice, solutions will be offered but will stop short of implementation details. In some situations, it may not be possible to offer a solution. This is because the solution(s) will depend on data analysis results. And data analysis results will not be available till implementation stage. This is particularly the case with ChatGPT Data Analyst because at this stage, I haven't issued any prompts to ChatGPT Data Analyst yet and therefore it is unknown what solutions it will recommend. For ChatGPT Data Analyst, I have to wait till the Implementation stage to see what it can offer.

1. Understand the Business Problem and Define Clear Objectives

a) Background

Credit risk assessment is a critical function in the financial industry, particularly for lenders such as banks, credit unions, and online lenders. When lending money, these institutions face the inherent risk that borrowers may fail to repay their loans, leading to financial losses and potential disruptions to their operations (The Investopedia Team, 2024). Therefore, accurately assessing the creditworthiness of borrowers is essential for mitigating this risk and ensuring the financial health of the lender.

Loan default prediction, a key aspect of credit risk assessment, involves predicting the probability that a borrower will default on their loan based on various factors. These factors may include demographic information (e.g., age, income), credit history (e.g., credit score, previous defaults), employment status, loan characteristics (e.g., loan amount, term), and macroeconomic indicators (e.g., unemployment rate, inflation). By analyzing these factors, lenders can identify high-risk borrowers and adjust their lending decisions accordingly.

The stakeholders, apart from the data scientist, include business leaders, risk managers, lending professionals, and regulatory experts. The data scientist will need to understand their perspectives, concerns, and priorities related to credit risk management and loan default prediction. Not only does the data scientist need to understand the lending criteria and the organisation's risk appetite, he/she also needs to know the relevant laws, regulations, and guidelines that impact credit risk assessment and compliance requirements. These include data privacy concerns, model interpretability issues, and data biases. The data scientist will also have to determine the availability, accessibility, and quality of data sources. The data scientist will need to collaborate with the Data Engineering team who will be responsible for acquiring the data which may come from multiple sources including the organisation's internal transaction processing systems and external systems such as credit bureaus. [Note: For this study, since it is not possible to access data from a financial institution's systems, we shall be using a ready-made dataset which is the German Credit dataset. As such, data collection and data engineering will not be in the scope of this study.] In order to establish clear success criteria, the data scientist must know the performance metrics relevant to the organisation's loan portfolio management. These metrics may include accuracy, precision, recall, ROC-AUC score, profitability metrics, and alignment with regulatory compliance requirements.

Below is an example of a Problem Statement which includes the project objectives as well as the success criteria:

“Increasing loan default rates pose a significant risk to our financial institution's profitability and stability. To mitigate this risk and ensure responsible lending practices, there is a critical need to develop a robust credit risk assessment modelling framework. This framework should leverage advanced analytics and machine learning techniques to accurately predict the probability of default for individual loan applicants based on their credit profile, financial history, and other relevant factors. By enhancing our ability to identify high-risk borrowers early in the lending process, we aim to optimize loan approval decisions, minimize losses from defaults, and maintain a healthy loan portfolio performance. The ultimate goal is to strike a balance between risk management and business growth by making informed lending

decisions based on comprehensive credit risk assessment methodologies. To evaluate the effectiveness of the credit risk assessment model, the following success criteria will be used:

- **ROC-AUC Score:** Achieve a minimum ROC-AUC score of 0.75, indicating strong discrimination between default and non-default cases.
- **Accuracy:** Attain an accuracy of at least 80% on the test dataset, ensuring reliable predictions of loan defaults.
- **Recall:** Maintain a recall above 70% to minimize false negatives to capture a significant portion of default cases.
- **Profitability Metrics:** Improve profitability metrics such as net interest margin (NIM) and return on assets (ROA) by at least 5% compared to baseline performance, demonstrating the model's positive impact on the financial institution's bottom line."

b) Evaluation

Although ChatGPT, with its gigantic knowledge of many domains, can be valuable in this phase, it needs to be guided or steered to produce the relevant information. ChatGPT does not have contextual knowledge about the organisation and its problems unless it is told. Even after it has been given the official factual information, it is still at a disadvantage compared to the human data scientist. An insider, who is well-acquainted with how things work in the organisation such as the culture, inter-departmental politics, attitudes or predispositions of the management and employees in the various departments, will be at an advantage as this type of unofficial information can tell him/her whether or not the management and various stakeholders are receptive to a particular solution. For example, knowing that the management is very conservative and risk-adverse, the data scientist will have to dig out a lot of hard facts such as what competitors have been doing to improve their credit assessment performance in order to convince the management to endorse the project. So, the data scientist basically have to "sell" the idea of the model prediction framework to the management. To be successful in this task, besides technical competence, intuition and also salesmanship skills are also required. The data scientist's reputation and previous achievements also play a major part in how much trust the management has in him/her. Although ChatGPT may have tremendous amounts of knowledge, note that this knowledge is limited to publicly available information, which if not free, can be bought. On the other hand, the data scientist, through his/her networking efforts, may have access to privileged information that is not generally available such as what competitors are doing. This definitely gives the data scientist a leg up over ChatGPT.

In conclusion, in this critical phase of the ML cycle, the human factor plays a vital role in initiating and driving the project forward. An experienced and capable Data Scientist, in collaboration with other domain experts, can have a big impact in the success of this phase of the project. ChatGPT can only play a supporting role, mainly using its vast knowledge to generate factual information on the behest of the data scientist. AutoML frameworks, both open source and commercial, do not profess to offer solutions in the business understanding phase of the machine learning cycle.

2. Preliminary Data Understanding

As mentioned in the Methodology chapter, this study will be using the German Credit dataset (Hofmann, 1994). The German Credit dataset has been used widely in research as a benchmark for evaluating the performance of different machine learning algorithms and techniques in credit risk assessment. It was originally introduced by Professor Hofmann from the University of Hamburg in the late 1980s. There is actually more than one version of this dataset. The dataset used in this study was downloaded from the UCI Machine Learning Repository which made available 2 versions. The original dataset, in the format provided by Prof. Hofmann, contains categorical/symbolic attributes. This version will be used in this study. The other version contains purely numeric data (the original categorical data converted to numbers). Another version of the German Credit dataset can be found on Kaggle. This version, used for competitions, has been changed substantially to make the data readable. In the original dataset, the categorical values are in codes. Another difference is that the number of features/attributes has been cut down from 20 to 10 only.

This realistic dataset contains information about credit applicants and their credit risk profiles. It includes features such as demographic information (age, gender, marital status, etc.), financial information (income, existing liabilities, savings, etc.), credit history (past credit performance, number of existing credit lines, duration of credit, etc.), employment information (employment status, job stability, etc.) and other features such as purpose of the loan, property ownership, etc. The German Credit dataset has 1000 rows. It has 20 features/attributes plus a target variable (last column, Credit Risk). The codes and associated interpretation for the features and target variable are as follows:

1. Checking Account Status (Categorical)

- A11: ... < 0 DM
- A12: 0 <= ... < 200 DM

- A13: ... ≥ 200 DM / salary assignments for at least 1 year
 - A14: no checking account
2. Duration in Months (Integer)
- Duration of the credit in months
3. Credit History (Categorical)
- A30: no credits taken/all credits paid back duly
 - A31: all credits at this bank paid back duly
 - A32: existing credits paid back duly till now
 - A33: delay in paying off in the past
 - A34: critical account/other credits existing (not at this bank)
4. Purpose (Categorical)
- A40: car (new)
 - A41: car (used)
 - A42: furniture/equipment
 - A43: radio/television
 - A44: domestic appliances
 - A45: repairs
 - A46: education
 - A47: (vacation - does not exist in the dataset)
 - A48: retraining
 - A49: business
 - A410: others
5. Credit Amount (Integer)
- Amount of the credit in DM
6. Savings Account/Bonds (Categorical)
- A61: ... < 100 DM
 - A62: $100 \leq \dots < 500$ DM
 - A63: $500 \leq \dots < 1000$ DM
 - A64: ... ≥ 1000 DM
 - A65: unknown/no savings account
7. Employment Duration (Categorical)
- A71: unemployed
 - A72: ... < 1 year
 - A73: $1 \leq \dots < 4$ years
 - A74: $4 \leq \dots < 7$ years
 - A75: ... ≥ 7 years
8. Instalment Rate Percentage (Integer)
- Instalment rate in percentage of disposable income
9. Personal Status and Sex (Categorical)
- A91: male : divorced/separated
 - A92: female : divorced/separated/married
 - A93: male : single
 - A94: male : married/widowed
 - A95: female : single
10. Other Debtors/Guarantors (Categorical)
- A101: none
 - A102: co-applicant
 - A103: guarantor
-

11. Present Residence Duration (Integer)
 - Present residence since
12. Property (Categorical)
 - A121: real estate
 - A122: if not A121 : building society savings agreement/life insurance
 - A123: if not A121/A122 : car or other
 - A124: unknown / no property
13. Age (Integer)
 - Age in years
14. Other Installment Plans (Categorical)
 - A141: bank
 - A142: stores
 - A143: none
15. Housing (Categorical)
 - A151: rent
 - A152: own
 - A153: for free
16. Number of Existing Credits at this Bank (Integer)
 - Number of existing credits at this bank
17. Job Type (Categorical)
 - A171: unemployed/unskilled - non-resident
 - A172: unskilled - resident
 - A173: skilled employee/official
 - A174: management/self-employed/highly qualified employee/officer
18. Number of people being liable to provide maintenance for (Integer)
 - Number of Dependents
19. Telephone (Categorical)
 - A191: none
 - A192: yes
20. Foreign Worker (Categorical)
 - A201: yes
 - A202: no
21. Credit Risk - Target Variable (Categorical)
 - 1: Good credit
 - 2: Bad credit

The intention, at this point, is acquire a general understanding of the dataset such as: (1) Size of dataset, (2) Number of attributes/features, (3) Data types of features, (4) Summary statistics of each feature, and (5) the Unique values of categorical features, along with their counts and percentages. Regarding 5, attention should be paid, in particular, to the target variable, Credit Risk, as I want to know if the target variable is imbalanced, and if so, how much.

A data scientist would typically obtain the information mentioned above using Python commands after loading the dataset. Similarly, with H2O AutoML, this information can be obtained using either its Python API or through the H2O Web Flow GUI. With ChatGPT

Data Analyst (CDA), natural language prompts will be used to generate the code to obtain the necessary information. The commands and output for this step will be shown and evaluated in Chapter 4 (Implementation).

3. Identify Data Quality issues

There are several methods for identifying outliers such as: Statistical methods (e.g. BoxPlots, Z-Score, IQR), Distance-Based Methods (e.g. KNN), Density-Based Methods (e.g. DBSCAN, Isolation Forest), and ML-based approaches (e.g. One-Class SVM, Autoencoder) (Ski, 2024). To detect data inconsistencies related to categorical features, one could display the unique values of each categorical feature along with their frequency counts. Very high or low frequency counts may indicate data inconsistencies. Visualizations are also effective in identifying data inconsistencies. One could use domain knowledge e.g. eyeballing for the different values that may mean the same thing e.g. “USA”, “US”, “U.S.A”, etc. Identifying missing values and duplicate rows is more a matter of implementation-dependent syntax so I don’t go any further than this in the design stage.

4. Remove Duplicate Data

Duplicate rows are identified and removed before the dataset is split. This is so that after the dataset is split, the proportions of the train and test sets are still retained. If duplicates are deleted after splitting the dataset, the proportions of the split may not be preserved especially if there are many duplicate rows. The method of removing duplicates is more a matter of implementation-dependent syntax so I don’t go any further than this in the design stage.

5. Fix Data Inconsistencies

Fixing inconsistent values in categorical features will vary depending on the situation so I will not go any further than this in the design stage.

6. Split the Dataset

From the summary statistics obtained in “Preliminary Data Understanding”, we can see that the German Credit dataset is slightly imbalanced with the target variable, “Credit Risk”, having a Majority:Minority class percentage proportions of 70:30. In other words, 30% of the loan recipients are “Bad Credit” risk. When we split the dataset into Train and Test sets, we want to ensure that this 70-30 split is preserved in both sets of data otherwise they would not be representative of the original dataset. Frameworks like Scikit-Learn are able to “stratify” the data in such a way when performing the split. But this is implementation so I will not go further than this here.

7. Balance the Imbalanced Dataset

As mentioned above, the German Credit dataset is slightly imbalanced with a 70:30 ratio in the number of Majority and Minority classes respectively. In the algorithm design diagram, the 30% threshold is arbitrary. What the designer is saying is that it is still worth exploring making the dataset balanced even though the imbalance is not great. It would actually be wise to use both approaches, i.e. with and without balancing, to see which approach produces the better performing model. Before getting into the different ways of correcting data imbalance, it should be noted that data balancing should be done only on the Train set. It is important to evaluate the model's performance on the original (unbalanced) test set. This is because if you balance the entire dataset, you won't have an unbiased test set to evaluate model performance accurately (Badr, 2019). Balancing the entire dataset can potentially introduce data leakage. Data leakage occurs when information from the test set is inadvertently used to train the model, leading to overly optimistic performance estimates (scikit-learn, 2024). Therefore, it's generally recommended to balance the training set only.

Back to how to balance the data. The most common method is using Resampling techniques, which includes Random oversampling/undersampling, SMOTE (Synthetic Minority Over-sampling Technique), and ADASYN (Adaptive Synthetic Sampling). There are also Algorithmic Techniques which uses Class Weights or Cost-sensitive Learning. Since the German Credit dataset has only 1000 rows, it would not be a good idea to perform undersampling since this would result in fewer rows in the dataset and hence loss of information. Instead, the minority class should be upsampled using either SMOTE or ADASYN. One potential drawback of SMOTE is that it can generate noisy synthetic samples. ADASYN is an extension of SMOTE that aims to address the drawback of generating noisy synthetic samples in regions of feature space where the density of minority class samples is already high. It generates more synthetic samples in regions where the density of minority class samples is lower and fewer synthetic samples in regions where the density is higher. ADASYN tends to focus on generating synthetic samples near the decision boundary, which can help improve the generalization performance of classifiers (Brownlee, 2021).

H2O.ai AutoML has a class balancing feature which if set, will result in the minority class being oversampled as well as the majority class being undersampled. The size of the resulting new dataset can be set (H2O, 2024a). We will leave it at that as this is getting into implementation.

8. Handle Missing Values

For numerical features, imputing with the mean or median can be a simple and effective approach, especially if the distribution of the feature is approximately normal or only slightly skewed. The advantages of using mean for imputation is that the overall mean of the feature distribution is preserved. It is also computationally efficient and straightforward to implement. However, mean imputation is sensitive to outliers in the data, especially if the feature distribution is skewed or contains extreme values (Alam et al., 2023). Outliers can disproportionately influence the mean and result in biased imputations. As a result, it may not accurately represent the central tendency of the data. Median imputation, on the other hand, is robust to outliers in the data because it is less affected by extreme values compared to the mean (Alam et al., 2023). It preserves the median of the feature distribution, which can provide a more representative measure of central tendency, especially if the feature distribution is skewed or contains outliers. Median imputation may not accurately represent the true mean of the feature distribution, especially if the distribution is approximately normal or symmetric. It may underestimate variability in the data compared to mean imputation, as it tends to result in imputed values closer to the center of the distribution. For categorical features, imputing with the mode (most frequent value) can be appropriate (Alam et al., 2023), especially if the feature has a small number of categories. A more drastic approach would be to delete the rows containing missing values which is not recommended as this will result in loss of information.

9. Handle Outliers

Before taking any actions on outliers, one should understand the cause and nature of outliers before deciding on the handling strategy. This requires domain knowledge and exploratory data analysis. Outliers can arise due to measurement errors, data entry mistakes, or genuine variability in the dataset. There are several strategies for handling outliers. If it is determined that the outliers are noise or errors, a simple approach would be to delete the rows containing the outliers (Sharma, 2021). However, this can result in loss of information. Another strategy is Winsorization, which is a technique that involves setting a threshold and replacing extreme values beyond the threshold with the threshold value (Singh, 2023). In this method, outliers are retained but their influence is limited. Transformations are another way of reducing the effect of outliers by adjusting the scale of data points. For features with positively skewed distributions, applying transformations e.g. logarithmic, square root and Box-Cox transformations can help reduce the influence of outliers by stabilising the variance of the data and make the distribution more symmetric (S, 2020). If the outliers are due to errors, imputing the outliers with median or mean values can be effective. One could also consider

using model algorithms which are less sensitive to outliers e.g. random forests and algorithms which use median-based loss functions.

10. Exploratory Data Analysis (EDA)

Note that prior to this, at “Preliminary Data Understanding” and “Identify Data Quality Issues”, some basic data summary statistics and data analysis have been done. It is recommended that EDA should be performed after duplicates have been removed and data inconsistencies have been fixed as they can influence the results of EDA. In addition, EDA should be performed on the Train set only which means prior to EDA, the dataset must be split. The reason for performing EDA on the Train set is that, in the real world, the model is trained on historical data (Train set) and then deployed to make predictions on unseen data (Test set). Moreover, using information from the Test set can inadvertently influence the modelling process, in other words data leakage. If EDA were performed on the combined Train and Test sets, any insights gained from analysing the Test set could bias the modelling process and compromise the validity of model evaluation metrics. Some of the tasks to be performed during EDA include: feature distributions, correlations analysis between numeric features, analysing features’ correlations to target variable, missing values analysis, outlier detection and exploring opportunities for feature engineering. In my opinion, EDA should not be performed at a fixed point in the ML lifecycle. It should be run more than once. For example, EDA is done before missing values and outliers are handled. After missing values and outliers are handled, EDA (maybe not everything) should be run again to see how features’ distributions are affected. In fact, after categorical features have been converted to numeric, correlation analysis and feature-target variable relationship analysis should be done again.

11. Perform Feature Engineering

One of the ways feature engineering can help to improve the performance of machine learning models is binning or discretization. This involves grouping numeric features into predefined intervals. This can help capture non-linear relationships and reduce the impact of outliers (ChenDataBytes, 2024). In the German Credit dataset, this technique can be applied to the numeric features. For example, *Duration of Credit (months)* can be categorised as: “Short Term”, “Medium Term” and “Long Term”. *Credit Amount* can be categorised as: “Small”, “Medium” and “Large”. Age can be categorised as: “20s and lower”, “30s”, “40s”, “50s”, “Seniors”. For categorical features, the existing categories can be merged, resulting in fewer categories. Further ideas on how to create new features or change existing ones can emerge after exploratory data analysis (EDA) has been done. For example the *Purpose*

feature, if the frequency count for the category “Retraining” is too low, it can be combined with the “Other” feature.

12. Encode Categorical Features into Numerical Values

This step is mandatory for those model algorithms which can only accept numeric inputs and should be done before feature selection. For categorical features which have a natural order e.g. for *Employment Duration* (“Unemployed”, “< 1 year”, “< 4 years”, “< 7 years”, “> 7 years”), Ordinal Encoding can be used to encode the categories to integers (0, 1, 2, 3, 4). For categorical features with nominal values i.e. without meaningful order, Label Encoding or One-Hot Encoding can be used. Label Encoding is similar to Ordinal Encoding but unlike the latter, it doesn’t assign significance to the order. If a feature has many categories (high dimensionality), One-Hot Encoding can result in too many features. Although the *Purpose* feature has quite a number of categories, it may still be desirable to use One-Hot Encoding if the data scientist, based on his/her domain knowledge, thinks the target variable will exhibit significant variation across different purposes.

13. Perform Feature Scaling on Numeric Features

Standardization scales features to have a mean of 0 and a standard deviation of 1 whereas Normalization scales features to a specified range, typically [0, 1] (Drpa, 2023). Whether to use Standardization (Z-Score normalization) or Normalization (Min-Max scaling) will depend on the characteristics of the dataset and the ML algorithms (classifiers) that will be used to develop the model. Standardization is suitable for ML algorithms that assume normal distributions of features e.g. logistic regression. Normalization may not be appropriate if the dataset contains outliers or if preserving the shape of the distribution is important. The decision to use Standardization or Normalization will be made at the implementation stage after the characteristics of the dataset have been analysed.

14. Perform Feature Selection

Feature selection is a vital step in building predictive models as it helps to improve model performance, reduce overfitting, and improve model interpretability. One feature selection technique is Correlation Analysis (which may have been done during EDA). By calculating the correlation between each numeric feature and the target variable, features with high absolute correlation values (positive or negative) are more likely to be predictive whereas those with low correlation values can be removed. Statistical tests like ANOVA or chi-square tests can also determine if a feature is a statistically significant predictor of the target variable. This technique evaluates each feature independently of others and selects the most informative ones based on a predefined significance level (Kaushik, 2024).

Decision trees and tree-based ensemble methods such as Random Forest and Gradient Boosting have inherent capability to perform feature selection as they can provide feature importances as part of their training process (Ajaymehta, 2023).

15. Track the Model Development Process (MLOps)

MLFlow will be used to track experiments performed during the model development process. In the model development workflow, an experiment will be defined for the run of each different classifier (ML algorithm). The name of the classifier (model), the best score for that classifier and the hyperparameters used will be saved to the MLFlow system. Later, the information logged for these experiments can be retrieved for comparison (Medhi, 2023). Some MLFlow code will need to be inserted into the model development code.

H2O.ai AutoML has a Web GUI called Flow which allows you to interactively explore and visualize the results of model training runs. You can view detailed information about each model, including metrics such as accuracy, AUC, and RMSE, as well as hyperparameters and other relevant statistics.

16. Select Model Evaluation Metrics

In the context of loan default, *false negatives* (predicted as “No Default” but actually does) have more severe financial consequences than false positives (predicted as “Default” but actually not). Therefore, it is more important to minimise False Negatives (FN). As such, evaluation metrics like *Recall* and *F1* score are more relevant. The Recall formula: $\text{Recall} = \text{TP} / (\text{TP} + \text{FN})$. Recall measures the proportion of actual positive instances (in this case, defaulted loans) that are correctly identified by the model. A higher recall indicates that the model is better at avoiding false negatives (missed defaults) and correctly identifying defaulted loans. The F1 score balances *precision* i.e. the ability to avoid false positives (false alarms) and recall i.e. the ability to capture all actual defaults (true positives (actual defaults)). The formula for Precision: $\text{Precision} = \text{TP} / (\text{TP} + \text{FP})$. The formula for F1 score: $\text{F1 score} = 2 \times (\text{Precision} \times \text{Recall}) / (\text{Precision} + \text{Recall})$. An alternate formula for F1 score is: $\text{F1 score} = (2 \times \text{TP}) / (2 \times \text{TP} + \text{FP} + \text{FN})$. It is particularly useful when there is an uneven class distribution or when false positives and false negatives have different costs. A higher F1 score indicates better overall performance, considering both precision and recall. Another metric that will be used is the ROC-AUC Curve which is a commonly used metric for evaluating the performance of binary classification models, including those used in loan default prediction tasks. ROC-AUC can be a useful metric for imbalanced classes because it evaluates the model's ability to distinguish between the positive and negative classes across different threshold values, making it robust to class imbalance. ROC-AUC measures the model's ability to correctly rank instances, with higher scores indicating better discrimination

between positive and negative instances. This is relevant in scenarios where prioritizing true positives (correctly predicting defaults) over false positives (incorrectly predicting defaults) is important.

17. Perform Cross Validation

Cross-Validation is commonly used when there is only a moderate amount of data, as is the case with the German Credit dataset. K-Fold cross-validation is the most common with a few variations. K-Fold Cross-Validation divides the dataset into k equal-sized folds. The model is trained k times, each time using $k-1$ folds for training and the remaining fold for validation. The final performance metric is the average of the performance metrics across all k folds. One variation of K-Fold is Stratified K-Fold Cross-Validation. Stratified K-Fold Cross-Validation ensures that each fold contains approximately the same proportion of instances from each class as the original dataset. This is particularly important when dealing with imbalanced datasets (Frenzel, 2023). Another variation is Repeated K-Fold Cross-Validation. Repeated K-Fold Cross-Validation repeats the K-Fold Cross-Validation process multiple times with different random splits of the data. Repeated K-Fold Cross-Validation is useful for obtaining more reliable performance estimates, especially when the dataset is small (Brownlee, 2020). Yet another variation is Repeated Stratified K-Fold Cross-Validation. This is the version that we will use. It combines both stratification and repetition (and randomisation) (scikit-learn, 2024). I will not consider Leave-One-Out Cross-Validation as that is computationally expensive.

18. Choose the Right ML Algorithm

For the Credit Risk Assessment model, the classifiers that will be used to develop the model are: (1) Logistic Regression (2) Decision Trees (3) Random Forests (4) Gradient Boosted Machine (GBM) and (5) Neural Networks. Logistic Regression was chosen because it is a binary classifier (and the Credit Risk Assessment model is binary classification problem) although it is a linear model. Decision Trees was chosen because it forms the basis for Random Forests and also it is quite different from Logistic Regression as it is non-linear. Random Forests is a generally good performer and is an ensemble model. Gradient Boosted Machine is also an ensemble but it works in a different way from Random Forests as GBM is a gradient boosting ensemble whereas Random Forests is a bagging ensemble. Although Neural Networks are considered black boxes and hence may not be favoured by financial institutions due to its lack of explainability and interpretability, the reason Neural Networks is included in this study is because it is very different from the other algorithms mentioned above. The SHAP framework will be used to interpret it. I will start with the simpler Scikit-

Learn Multi-Layer Perceptron (MLP) and then the TensorFlow KerasClassifier to see if there is any improvement.

As for H2O.ai AutoML, the user can use the Web GUI, Flow, to specify that AutoML is to run all the eligible algorithms or he/she can be selective by choosing only certain algorithms. I will get into the details in the implementation stage. In this study, I will not be using the XGBoost algorithm because it requires a GPU for it to run in H2O.ai AutoML.

19. Perform Regularization

For *Logistic Regression*, either L1 (Lasso) or L2 (Ridge) regularisation is applied. L1/L2 is a penalty term which is added to the Loss function to prevent overfitting and improve the generalisation performance of the model. L1 penalizes large coefficients and drives some coefficients to exactly zero, effectively performing feature selection. L2 also penalizes large coefficients and encourages smaller coefficients but without making them zero, effectively reducing their impact (Favilla, 2023). If you think that only a few features influence loan default target variable, use L1. On the other hand, if you think more than a few features influence loan default target variable but to varying degrees, use L2.

For *Decision Trees*, there are several techniques that can be used to prevent overfitting including Tree Pruning, setting Minimum Samples per Leaf, setting Minimum Samples per Split, limiting the Maximum Depth of the tree and limiting the Maximum Number of Features (Pramod, 2023).

The Bootstrap hyperparameter of *Random Forests* if set to True (default) uses bootstrap sampling to create multiple subsets of the training data for training each tree. By introducing randomness into this process, it helps to reduce overfitting (Lyu, 2021). There are several additional regularization techniques that be applied: Number of Trees (n_estimators), Maximum Depth of Trees, Minimum Samples per Leaf, Minimum Samples per Split, Maximum Number of Features (Sachinsoni, 2023).

For *Gradient Boosted Machine*, regularization techniques that can be applied include: Shrinkage (learning rate), Maximum Depth of Trees, Minimum Samples per Leaf, Minimum Samples per Split, Maximum Number of Features, and Early Stopping. Early stopping is a technique used to stop the training process when the model's performance on a validation set no longer improves (Tuychiev, 2023).

For *Neural Networks*, L1 or L2 regularization is applied to penalize large weights in the network. Another regularization technique is Dropout which randomly drops a fraction of the neurons along with their connections from the neural network during training (Sagar, 2019). Batch Normalization is used to normalize the activations of each layer in the neural network by adjusting and scaling the activations based on the mean and variance of each mini-batch (Ioffe & Szegedy, 2015). Early stopping is a technique used to stop the training

process when the model's performance on a validation set no longer improves (Sagar, 2019).

20. Perform Hyperparameter Tuning

For Logistic Regression, the important hyperparameters to be tuned are: Solver, Penalty, and Regularization Strength. In “Perform Regularization”, we have discussed what Penalty and Regularization Strength are used for. This leaves Solver, which is the algorithm to be used in the optimisation problem. The possible values are: “newton-cg”, “lbfgs” (default), “liblinear”, “sag”, “saga”. We will stick with the default “lbfgs” because it is memory-efficient and performs relatively well (Adam, 2023). The other algorithms are more suitable for large datasets.

For Decision Trees, as I mentioned in “Perform Regularization”, hyperparameters such as Maximum Depth of the tree, Maximum Number of Features, Minimum Samples per Leaf, Minimum Samples per Split, and Tree Pruning are used to generalise the model and avoid overfitting. The hyperparameter *Criterion* is the splitting criteria which is either “Gini” or “Entropy” (default). I stick with the default because Gini Impurity tends to favour the majority class when the dataset is imbalanced (Python Programmer, 2023). For the German credit dataset, which is not large and has a moderate number of features, using the default value of "best" for the *Splitter* hyperparameter is typically recommended. This strategy ensures that the decision tree makes informed decisions based on the best available information at each node, leading to more interpretable and potentially better-performing models. Since the German Credit dataset is imbalanced, it is probably unnecessary to make adjustments to the *Class Weights* since the train set has been balanced. However, in case balancing does not help, it is worth looking into class weights. Since decision trees can be prone to overfitting, *Pruning* parameters like *min_impurity_decrease*, *min_impurity_split*, or *ccp_alpha* can help control tree growth and prevent overfitting.

Random Forest shares some of the same hyperparameters as Decision Trees so I won't go through them again. Most of them are used to regularise the model. The *Number of Trees* (*n_estimators*) in the random forest is also used to reduce overfitting. As mentioned in “Perform Regularization”, *Bootstrap Sampling* introduces randomness and helps reduce variance (Lyu, 2021).

For Gradient Boosted Machine (GBM), the *Learning Rate* hyperparameter controls the contribution of each tree to the final prediction. Lower values make the model more robust but require more trees. Typically it is set 0.01 to 0.1 (default). The *Number of Trees* hyperparameter can be set to 40-70. The *Tree Depth* hyperparameter can be set to a maximum of 3-5. The *Subsample Ratio* hyperparameter determines the fraction of samples to be used

for training each tree. Setting it to a value less than 1.0 (e.g. 0.5 to 0.8) can help prevent overfitting. The *Early Stopping* hyperparameter has already been covered under “Perform Regularization” (Tuychiev, 2023).

For Neural Networks, the hyperparameters to tune include: Number of Hidden Layers, Number of Neurons in each Layer, Activation Function, Learning Rate, Optimizer, Mini-Batch size, Number of Epochs, Dropout Rate, and L1/L2 Regularization. The greater the number of layers, the higher the complexity of the model, hence the higher the chance of overfitting. One or two layers is often enough to build a good model. Similarly, the more the number of neurons, the higher the complexity of the model, hence the higher the chance of overfitting (Bismi, 2023; Brownlee, 2022). The activation function is a non-linear transformation function. A few of the popular ones used in practice are Rectified Linear Activation Function (ReLU), Exponential Linear Unit (ELU), Sigmoid, Softmax and Tanh (Nair, 2023). The Learning Rate controls the magnitude of the step made by the optimizer when learning from the provided training data during the optimization process. A lower value increases the likelihood of the model discovering a more optimum solution, but it comes at the expense of longer computational time (Bismi, 2023). The Optimizer is the gradient-based optimization algorithm. Common ones include Adam and SGD. The mini-batch size determines the number of training samples processed by the neural network in each training step. Typically, higher values lead to better model performance. An epoch refers to one complete pass through the entire training dataset during the training phase. Having too many epochs may result in overfitting, whereas having too few epochs can lead to underfitting (Bismi, 2023). The Dropout layer is a special layer that acts as a regularizer by randomly deactivating neurons by setting their unit values to zero. Dropout Rate controls the number of units set to zero during each training iteration. By randomly disabling neurons, dropout injects noise and unpredictability into the neural network, making it more robust and reducing the risk of overfitting to the training data (Vivek4x59, 2024).

21. Perform Ensemble Learning

This is the area where H2O.ai AutoML shines. Apart from the bagging and gradient-boosting ensembles, H2O.ai AutoML can also create *Stacked* ensembles. Whereas bagging and gradient-boosting ensembles are composed of weak learners, stacking works by optimally combining a diverse set of strongly predictive models. H2O.ai AutoML can create an *All Models* stacked ensemble by combining the entire list of models investigated in an AutoML run. It can also create a *Best of Family* ensemble, which contains at most, six

component models, but is quite comparable to the All Models ensemble in performance (H2O, 2024b).

22. Model Interpretability and Explainability

For the hand-crafted Credit Risk Assessment model, the SHAP framework will be used to explain how loan default prediction is made. The SHAP values will be used to interpret the models and understand the impact of each feature on the predictions. H2O.ai AutoML have a built-in implementation of model explainability methods for explaining the predictions of an AutoML model. The explainability interface provides different explainability graphs that help users get detailed information about the models trained (H2O, 2024c).

23. Ensure Scalability

Since the German Credit dataset has only 1000 rows, it is not possible to illustrate how well our Credit Risk Assessment model will perform with a large dataset e.g. 100k rows. H2O.ai AutoML has a very scalable architecture as it can support parallel and distributed computing. H2O Sparkling Water is open source platform which leverages Apache Spark, allowing machine learning workflows to be scaled across distributed clusters of machines (H2O, 2024d). Unfortunately, due to the limitations of the hardware used in this study, it is not possible to illustrate this powerful H2O platform.

24. Model Deployment

After the different models have been evaluated and a best model has been identified, the best model will be deployed to the local server. A web GUI, developed with Streamlit, will be used to accept input of loan information. The proposed user interface design is shown in Figure 4.2. This information will be forwarded to an API developed using FastAPI. FastAPI is a contemporary web framework designed for swiftly constructing APIs using Python, leveraging standard Python type hints to ensure efficient performance (FastAPI, n.d.). For the hand-crafted solution, the model along with the Scaler object, which was used in preprocessing, will be deployed as Pickle files. For the H2O AutoML implementation, the Model will deployed as a MOJO model. The API will preprocess the input loan information, encoding and scaling the data, and pass it on to the model for prediction. The predicted result will be returned to the Web GUI for display. The model serving diagram is something like the diagram shown in Figure 4.3.

Screen 1	Screen 2	Screen 3
Loan Default Predictor Please fill in all the details below: 1. Checking Account Status <input type="button" value="Select an option v"/> 2. Duration (in months) <input type="button" value="Enter value"/> 3. Credit History <input type="button" value="Select an option v"/> 4. Purpose <input type="button" value="Select an option v"/> 5. Credit Amount <input type="button" value="Enter value"/> 6. Savings Account/Bonds <input type="button" value="Select an option v"/> 7. Employment Duration <input type="button" value="Select an option v"/> 8. Instalment Rate % <input type="button" value="Enter value"/> 9. Personal Status & Sex <input type="button" value="Select an option v"/> 10. Other Debtors/Guarantors <input type="button" value="Select an option v"/>	11. Present Residence Duration <input type="button" value="Enter value"/> 12. Property <input type="button" value="Select an option v"/> 13. Age <input type="button" value="Enter value"/> 14. Other Installment Plans <input type="button" value="Select an option v"/> 15. Housing <input type="button" value="Select an option v"/> 16. No. of Existing Credits at this Bank <input type="button" value="Enter value"/> 17. Job Type <input type="button" value="Select an option v"/> 18. No. of People being liable to provide maintenance for <input type="button" value="Enter value"/> 19. Telephone <input type="button" value="Select an option v"/> 20. Foreign Worker <input type="button" value="Select an option v"/> <input type="button" value="PREDICT"/>	Outcome <input type="button" value="Likely to DEFAULT"/> Explanation XXXXXXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXXXXXX

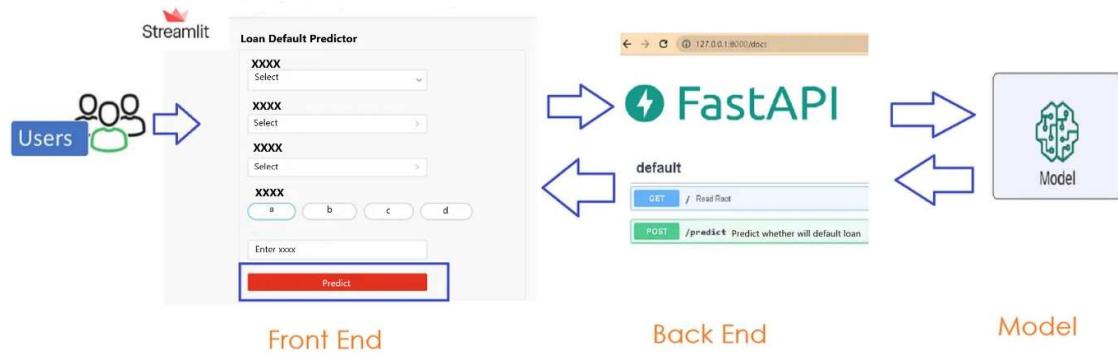
Figure 4.2. Sample User Interface Design.

Figure 4.3. The Process of Entering Input and Returning the Prediction. Adapted from *How to build complete end-to-end ML model, Backend RestAPI using FastAPI and front-end UI using Streamlit* by B. Gorad, 2022, Medium. <https://medium.com/@goradbj/how-to-build-complete-end-to-end-ml-model-backend-restapi-using-fastapi-and-front-end-ui-using-22f64bf04476>. Copyright 2022 by Balwant Gorad.

25. Production Model Monitoring

For a mission-critical application like Loan Default Prediction model, it is vital to monitor after it has been deployed to production. However, I will not be implementing this because it takes a substantial effort to build the infrastructure for the monitoring system from scratch. The H2O ecosystem has a component called H2O MLOps but it is commercial and

proprietary so we don't have a production monitoring system to compare with as well (H2O, 2022).

4.3 Machine Learning Algorithms Proposed by Human Data Scientist

As mentioned earlier, I will allow the AutoML and Generative AI approaches to decide on the Machine Learning models they intend to use. However, for the Human Data Scientist approach, I will be using 5 specific Machine Learning Algorithms, which are Logistic Regression, Decision Tree, Random Forest, Gradient Boosting Machine (GBM) and Neural Networks. It is possible that I may add additional ones, for example Stacked Ensemble models.

4.3.1 Logistic Regression

Logistic regression is a fundamental machine learning algorithm used for binary classification tasks, where the goal is to predict the probability that an instance belongs to one of two classes. It is a classification algorithm rather than a regression algorithm (Acharya, 2023). The algorithm models the relationship between a set of input features and the probability of a binary outcome using the logistic function, also known as the sigmoid function (refer to Figure 4.4). The sigmoid function is an S-shaped curve that maps any real-valued input to a value between 0 and 1, which can be interpreted as a probability (Acharya, 2023).

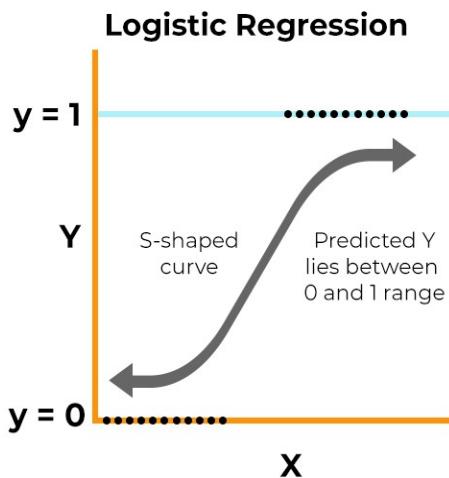


Figure 4.4. Key Assumptions for Implementing Logistic Regression. Adapted from *What is Logistic Regression? Equation, Assumptions, Types and Best Practices* by V. Kanade, 2022, Spiceworks. <https://www.spiceworks.com/tech/artificial-intelligence/articles/what-is-logistic-regression/>. Copyright 2024 by Spiceworks Inc.

The logistic regression model assumes a linear relationship between the input features and the log-odds of the probability of the positive class (Statistics Solutions, 2024). Mathematically, the logistic regression model can be represented as:

$$P(y = 1|x) = \frac{1}{1+e^{-(w^T x+b)}}$$

Where:

- $P(y = 1|x)$ represents the probability that the output y is equal to 1 given the input features x .
- w is the weight vector (coefficients) associated with each input feature.
- x is the input feature vector.
- b is the bias term.
- e is the base of the natural logarithm.

The model parameters (weights and bias) are learned from the training data using optimization algorithms such as gradient descent, which minimises a loss function that measures the difference between the predicted probabilities and the actual class labels. The most common loss function used in logistic regression is the binary cross-entropy loss (Kashaboina, 2023).

Once the model is trained, it can be used to make predictions by computing the probability that an instance belongs to the positive class using the learned parameters and the input features. The predicted probability can then be converted into a binary prediction by applying a threshold (usually 0.5), where instances with a predicted probability above the threshold are classified as positive, and those below the threshold are classified as negative.

Logistic regression offers several advantages, including simplicity, interpretability and efficiency. It can handle both numerical and categorical input features and is less prone to overfitting compared to more complex models. Additionally, it provides interpretable coefficients that indicate the strength and direction of the relationship between each input feature and the outcome (Amiyaranjanrout, 2023).

In summary, logistic regression is a powerful and widely-used algorithm for binary classification tasks, leveraging the logistic function to model the probability of the positive class based on input features. Its simplicity, interpretability and effectiveness make it a popular choice for a wide range of applications in various fields, including healthcare, finance, marketing, and more.

4.3.2 Decision Tree

The decision tree algorithm is a versatile and intuitive machine learning technique used for both classification and regression tasks. It creates a tree-like structure where each internal node represents a decision based on a feature, each branch represents the outcome of the decision and each leaf node represents the final decision or prediction (Thevapalan & Le, 2023). An example of a decision tree is shown in Figure 4.5.

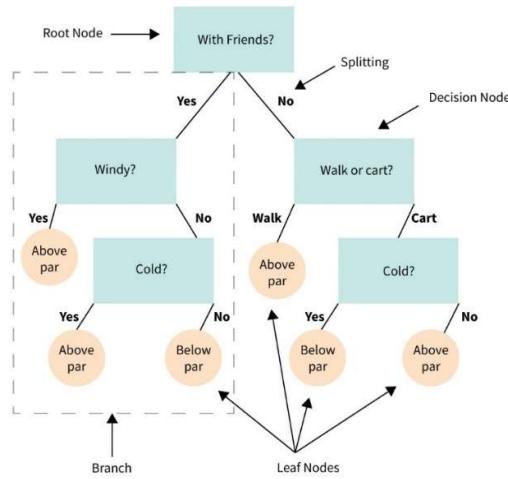


Figure 4.5. An example of a Decision Tree. From *What is a Decision Tree?* by Master's in Data Science, 2024, Master's in Data Science.

<https://www.mastersindatascience.org/learning/machine-learning-algorithms/decision-tree/>. Copyright 2024 by Master's In Data Science.

The decision tree algorithm operates by recursively partitioning the feature space into subsets that are as pure as possible with respect to the target variable. The goal is to create simple yet effective rules for predicting the target variable based on the input features (Thevapalan & Le, 2023).

The process of building a decision tree typically involves the following steps (Ivy, 2023):

1. Feature Selection:

The algorithm selects the best feature to split the data at each node based on a criterion like Gini impurity or information gain. The feature that maximises the purity of the resulting subsets or information gain is chosen as the splitting criterion.

2. Splitting:

Once the best feature is selected, the data is split into subsets based on different values of that feature. For categorical features, each possible value defines a separate branch in the tree. For numerical features, the data is split into two subsets based on a threshold value.

3. Recursive Partitioning:

The splitting process is repeated recursively for each subset created at the current node. The algorithm continues to partition the data until a stopping criterion is met, such as reaching a maximum tree depth, achieving a minimum number of samples per leaf or when further splitting does not improve the model's performance.

4. Stopping Criteria:

To prevent overfitting, the decision tree algorithm uses stopping criteria to determine when to stop growing the tree. These criteria include maximum tree depth, minimum number of samples per leaf node and minimum impurity decrease for a split.

5. Leaf Node Prediction:

Once the tree has been grown, each leaf node represents a final decision or prediction for the target variable. For classification tasks, the majority class in the leaf node is assigned as the predicted class. For regression tasks, the mean or median value of the target variable in the leaf node is assigned as the predicted value.

The algorithm offers several advantages, including simplicity, interpretability and the ability to handle both numerical and categorical data. Decision trees are also robust to outliers and can capture non-linear relationships between features and the target variable (Attard, 2024).

However, it is prone to overfitting, especially when the tree is allowed to grow too deep or when the dataset is noisy (Attard, 2024). Techniques like pruning, limiting tree depth and using ensemble methods like random forests can help mitigate overfitting and improve the generalization performance of decision trees (Thulo, 2023).

In summary, the decision tree algorithm is a powerful and widely-used machine learning technique that creates interpretable models by recursively partitioning the feature space based on simple decision rules. Its simplicity and flexibility make it a popular choice for a wide range of classification and regression tasks in various domains, including finance, healthcare and marketing.

4.3.3 Random Forest

Random forest is a powerful ensemble learning technique used for both classification and regression tasks. It builds multiple decision trees during training and outputs the mode (classification) or mean (regression) prediction of the individual trees as the final prediction (Ashtari , 2024). The random forest algorithm introduces randomness in the tree-building process to improve the performance and robustness of individual decision trees. Figure 4.6 illustrates a random forest.

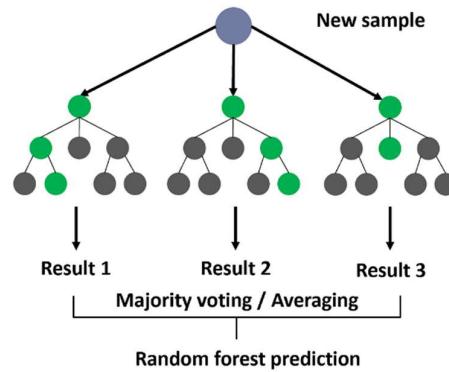


Figure 4.6. Random Forest. From *Random Forests* by R. Yehoshua, 2023, Medium.
<https://medium.com/@roiyeoh/random-forests-98892261dc49>. Copyright 2024 by Dr. Roi Yehoshua.

The random forest algorithm operates as follows (Lyu, 2021):

1. Bootstrapped Sampling:

Random forest starts by randomly sampling the training data with replacement to create multiple bootstrap samples. Each bootstrap sample is used to train a separate decision tree in the ensemble.

2. Random Feature Selection:

During the construction of each decision tree, a random subset of features is selected as candidates for splitting at each node. This random feature selection helps introduce diversity among the trees in the ensemble and reduces the correlation between them.

3. Tree Construction:

For each decision tree in the ensemble, the tree-building process follows the same recursive partitioning approach as traditional decision trees. At each node, the algorithm selects the best feature among the random subset of features and splits the data based on a criterion such as Gini impurity or information gain.

4. Ensemble Aggregation:

Once all the decision trees are built, the random forest aggregates their predictions to make the final prediction. For classification tasks, the mode (most frequent class) of the individual tree predictions is taken as the final prediction. For regression tasks, the mean or median of the individual tree predictions is calculated.

The random forest algorithm offers several advantages over individual decision trees, including improved accuracy, robustness to overfitting, and resistance to noise in the data. By averaging the predictions of multiple trees trained on different subsets of the data, random forest reduces the variance of the model and provides more stable and reliable predictions (Fleiss, 2023).

Additionally, random forest provides built-in feature importance scores that indicate the relative importance of each feature in predicting the target variable. This information can be useful for feature selection and interpretation of the model.

Despite its advantages, random forest has some limitations, including increased computational complexity and potential difficulties in interpreting the individual trees in the ensemble (Fleiss, 2023).

In summary, the random forest algorithm is a powerful and widely used ensemble learning technique that combines the strengths of multiple decision trees to create robust and accurate predictive models. Its ability to handle both classification and regression tasks, along with its flexibility and interpretability, make it a popular choice for a wide range of machine learning applications in various domains.

4.3.4 Gradient Boosting Machine

Gradient Boosting Machine (GBM) is a powerful machine learning technique used for both regression and classification tasks. It builds an ensemble of weak learners (typically decision trees) sequentially, where each new learner corrects the errors made by the previous ones. GBM combines the predictions of these weak learners to create a strong predictive model (Saini, 2024).

Figure 4.7 illustrates how Gradient Boosting works.

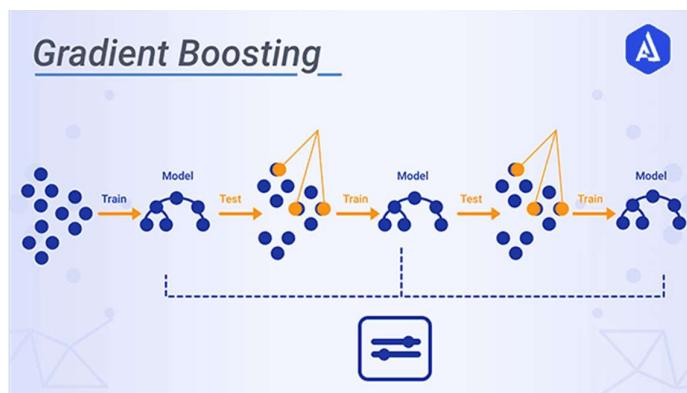


Figure 4.7. Gradient Boosting. From *Gradient Boosting in Machine Learning* by A. K. Singh, 2021, Medium.

<https://anupkrsingh.medium.com/gradient-boosting-in-machine-learning-8446c8d31bb1>.

Copyright 2021 by Anup Kumar Singh.

The gradient boosting machine algorithm operates as follows:

1. **Initialization** (Kılıç, 2023; Kharkar, 2023):

GBM starts by initializing the ensemble with a simple model, often the mean (for regression) or the majority class (for classification) of the target variable. This initial model serves as the starting point for building the ensemble.

2. **Sequential Model Building** (Kılıç, 2023):
GBM builds the ensemble sequentially by adding new weak learners (e.g., decision trees) to the ensemble one at a time. Each new learner is trained to minimize the errors (residuals) made by the existing ensemble on the training data.
3. **Gradient Descent Optimization** (Kharkar, 2023):
To train each new weak learner, GBM uses gradient descent optimization to minimize a loss function that measures the difference between the actual target values and the predictions of the current ensemble. The gradient of the loss function with respect to the ensemble's predictions provides the direction in which to update the ensemble to reduce the errors.
4. **Weak Learner Training** (JavaTpoint, 2021):
During each iteration, GBM trains a new weak learner (e.g., decision tree) on the residuals (errors) of the current ensemble. The new learner focuses on capturing the patterns in the data that were not captured by the existing ensemble.
5. **Learning Rate** (Kharkar, 2023):
GBM introduces a learning rate hyperparameter, which controls the contribution of each new weak learner to the ensemble. A smaller learning rate makes the ensemble more robust to overfitting but requires more iterations to converge, while a larger learning rate accelerates the learning process but increases the risk of overfitting.
6. **Ensemble Aggregation** (Kharkar, 2023):
Once all the weak learners are trained, GBM aggregates their predictions by summing them together. For regression tasks, the final prediction is the sum of the predictions of all weak learners. For classification tasks, the final prediction is obtained by applying a sigmoid or softmax function to the sum of the predictions to obtain probabilities for each class.

By iteratively adding new weak learners to the ensemble and optimizing their predictions to minimize the errors of the current ensemble, GBM gradually improves the model's predictive performance. The ensemble's ability to learn complex relationships in the data and its robustness to noise make GBM a powerful and widely-used machine learning algorithm in various domains (FasterCapital, n.d.).

However, GBM is prone to overfitting if not properly regularized, and it can be computationally expensive and sensitive to hyperparameters. Regularization techniques like adjusting the learning rate, restricting tree depth restrictions and applying early stopping are commonly used to mitigate these challenges and improve the generalization performance of GBM (Kharkar, 2023).

In summary, the gradient boosting machine algorithm is an ensemble learning technique that builds an ensemble of weak learners sequentially to create a strong predictive model. Its ability to learn complex relationships in the data and its robustness to noise make it a popular choice for a wide range of machine learning tasks in various domains.

4.3.5 Neural Networks

A neural network (Figure 4.8) is a powerful machine learning algorithm inspired by the structure and function of the human brain. It consists of interconnected layers of artificial neurons, known as nodes or units, organised into input, hidden, and output layers. Each neuron receives input from the previous layer, processes it using an activation function and passes the output to the next layer (Awan, 2023).

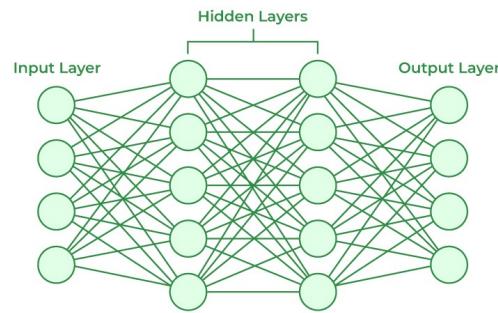


Figure 4.8. Neural Networks Architecture. From *Artificial Neural Networks and its Applications* by H. , 2023, GeeksforGeeks. <https://www.geeksforgeeks.org/artificial-neural-networks-and-its-applications/>. Copyright 2024 by Harikiran78.

The operation of a neural network can be summarized into the following steps (Sarita, 2023; Awan, 2023):

1. Input Layer:

This receives the raw input data, which could be features from a dataset or pixels from an image. Each neuron in the input layer represents a feature or pixel value.

2. Hidden Layers:

There may be one or more hidden layers. Each hidden layer consists of multiple neurons that perform computations on the input data. These computations involve linear transformations (weighted sums of inputs) followed by nonlinear transformations (activation functions) to introduce nonlinearity and enable the network to learn complex patterns in the data.

3. Weights and Biases:

Each connection between neurons in adjacent layers is associated with a weight, which represents the strength of the connection. The weights are learnable parameters that are adjusted during the training process to minimize the error between the predicted outputs and the actual targets. Additionally, each neuron has a bias term, which allows the network to learn more complex patterns by shifting the activation function.

4. Activation Functions:

They introduce nonlinearity into the neural network, enabling it to learn complex relationships in the data. Common activation functions include sigmoid, tanh, ReLU (Rectified Linear Unit) and softmax. These functions transform the weighted sum of inputs into an output that is passed to the next layer.

5. Output Layer:

It produces the final predictions or outputs based on the processed information from the hidden layers. The number of neurons in the output layer depends on the task at hand—regression tasks typically have one output neuron, while classification tasks have multiple output neurons corresponding to each class label.

6. Forward Propagation:

During this phase, input data is passed through the network layer by layer, with each layer performing computations and passing the results to the next layer. The final output of the network is obtained from the output layer.

7. Backpropagation:

This is the process of training the neural network by updating the weights and biases to minimize the difference between the predicted outputs and the actual targets. It involves calculating the gradient of the loss function with respect to the network's parameters and adjusting the parameters using gradient descent optimization algorithms.

8. Training:

The neural network is trained using a labeled dataset, where the inputs are paired with their corresponding targets. During training, the network learns to map inputs to outputs by iteratively adjusting its parameters (weights and biases) to minimize the prediction error. Training typically involves multiple iterations (epochs) over the entire dataset.

9. Evaluation:

Once the neural network is trained, it can be evaluated on a separate validation or test dataset to assess its performance. Evaluation metrics such as accuracy, loss, precision, recall and F1-score are used to measure the model's performance on unseen data and identify areas for improvement.

In summary, a neural network is a versatile and powerful machine learning algorithm that learns complex patterns and relationships in data by processing information through interconnected layers of artificial neurons. Its ability to learn from data and make predictions across various domains, including image recognition, natural language processing and time series forecasting, makes it one of the most widely-used and impactful techniques in modern machine learning.

Chapter 5

Implementation and Testing

5 Implementation and Testing

This section covers the implementation and testing of each of the three proposed approaches. For each approach, the Machine Learning Model implementation is first covered followed by the User Interface & API Integration implementation and then System Integration Testing.

For each approach, the entire ML life cycle will be covered, with the exception of “Monitoring” (which will be a future enhancement). Note that “Business Understanding” has already been covered in Chapter 4 (Design). Only selected code segments and visualizations/results will be shown. For more technical details, the reader can refer to the source code files. As for the System Integration Testing part, it covers all the test cases that have been carried out for each approach.

5.1 AutoML Approach

H2O AutoML has a Web GUI called Flow which allows a ML model to be developed using GUI. Because the functionality of this GUI is quite limited and, more importantly, I want my work to be reproducible, I chose to use the programmatic approach which involves using the H2O Python API as well as normal Python code.

Since H2O AutoML does not provide much built-in support for EDA (Exploratory Data Analysis), the visualizations below were generated using the Python libraries matplotlib and seaborn. It should also be mentioned that, for imbalanced data treatment, I did not use H2O AutoML’s class balancing feature because it works by performing oversampling on the minority class and undersampling on the majority class. And this is fixed and cannot be changed. I do not want to undersample the majority class; I rather just upsample the minority class to match count in the majority class. To do this, I used an external library, imblearn.

5.1.1 Machine Learning Model Implementation

Step 1: Initializing an H2O Cluster

We must first initialize an H2O cluster in Python, which is necessary to set up the distributed computing environment that powers H2O’s data processing and machine learning tasks. This is shown below.

```
[2]: h2o.init()

Checking whether there is an H2O instance running at http://localhost:54321.... not found.
Attempting to start a local H2O server...
; Java HotSpot(TM) 64-Bit Server VM (build 17.0.6+9-LTS-190, mixed mode, sharing)
Starting server from C:\ProgramData\Anaconda3\envs\ForFYI\lib\site-packages\h2o\backend\bin\h2o.jar
Ice root: C:\Users\ryank\AppData\Local\Temp\tmpdr84ixzw
JVM stdout: C:\Users\ryank\AppData\Local\Temp\tmpdr84ixzw\h2o_ryank_started_from_python.out
JVM stderr: C:\Users\ryank\AppData\Local\Temp\tmpdr84ixzw\h2o_ryank_started_from_python.err
Server is running at http://127.0.0.1:54321
Connecting to H2O server at http://127.0.0.1:54321 ... successful.
Warning: Your H2O cluster version is (7 months and 8 days) old. There may be a newer version available.
Please download and install the latest version from: https://h2o-release.s3.amazonaws.com/h2o/latest_stable.html

H2O_cluster_uptime: 04 secs
H2O_cluster_timezone: Asia/Kuala_Lumpur
H2O_data_parsing_timezone: UTC
H2O_cluster_version: 3.44.0.3
H2O_cluster_version_age: 7 months and 8 days
H2O_cluster_name: H2O_from_python_ryank_ya8dof
H2O_cluster_total_nodes: 1
H2O_cluster_free_memory: 3.922 Gb
H2O_cluster_total_cores: 8
H2O_cluster_allowed_cores: 8
H2O_cluster_status: locked, healthy
H2O_connection_url: http://127.0.0.1:54321
H2O_connection_proxy: {"http": null, "https": null}
H2O_internal_security: False
Python_version: 3.8.18 final
```

Snippet 5.1: Initializing an H2O Cluster

Step 2: Loading the Dataset

Snippet 5.2 shows the dataset being loaded into an H2O Dataframe, rather than a Pandas Dataframe, and having its columns named since the dataset file does not come with the column names.

1.2 Load dataset

```
: raw_data_path = '../Dataset/german.data'
: data_frame = h2o.import_file(raw_data_path)
Parse progress: [██████████] (done) 100%
```

1.3 Add Column Names to the Dataframe

Currently, the dataframe has no column names. Here, I am adding them so that it is easier to identify each column

```
: column_names = ['checking_acct_status', 'duration_months', 'credit_history', 'purpose', 'credit_amt', 'savings_acct_bonds', 'employment_duration', 'inst_data_frame.columns = column_names
```

Snippet 5.2. Loading the Dataset

Step 3: An Overview of the Dataset

Snippet 5.3 shows 10 sample records from the dataset while Snippet 5.4 shows some statistics about the dataset, such as the minimum value, maximum value and number of missing values per column. There are no missing values in all the columns so no imputation is required.

data_frame.head()										
checking_acct_status	duration_months	credit_history	purpose	credit_amt	savings_acct_bonds	employment_duration	instalment_rate_%	personal_status_sex	other_del	
A11	6	A34	A43	1169	A65	A75	4	A93		
A12	48	A32	A43	5951	A61	A73	2	A92		
A14	12	A34	A46	2096	A61	A74	2	A93		
A11	42	A32	A42	7882	A61	A74	2	A93		
A11	24	A33	A40	4870	A61	A73	3	A93		
A14	36	A32	A46	9055	A65	A73	2	A93		
A14	24	A32	A42	2835	A63	A75	3	A93		
A12	36	A32	A41	6948	A61	A73	2	A93		
A14	12	A32	A43	3059	A64	A74	2	A91		
A12	30	A34	A40	5234	A61	A71	4	A94		

[10 rows x 21 columns]

Snippet 5.3. A few records of the dataset

data_frame.describe()										
type	enum	int	enum	enum	int	enum	enum	int	int	
min		4.0			250.0				1.0	
mean		20.903			3271.258				2.973	
max		72.0			18424.0				4.0	
sigma		12.059814452756373			2822.7368759604406				1.1187146743126786	
zeros		0			0				0	
missing	0	0	0	0	0	0	0	0	0	
0	A11	6.0	A34	A43	1169.0	A65	A75	4.0		
1	A12	48.0	A32	A43	5951.0	A61	A73	2.0		
2	A14	12.0	A34	A46	2096.0	A61	A74	2.0		
3	A11	42.0	A32	A42	7882.0	A61	A74	2.0		
4	A11	24.0	A33	A40	4870.0	A61	A73	3.0		
5	A14	36.0	A32	A46	9055.0	A65	A73	2.0		
6	A14	24.0	A32	A42	2835.0	A63	A75	3.0		
7	A12	36.0	A32	A41	6948.0	A61	A73	2.0		
8	A14	12.0	A32	A43	3059.0	A64	A74	2.0		
9	A12	30.0	A34	A40	5234.0	A61	A71	4.0		

[1000 rows x 21 columns]

Snippet 5.4. Some Statistics about the Dataset

Step 4: Check for Duplicates and Data Inconsistencies before Splitting

Duplicate rows were checked for and there were none (refer to Snippet 5.5).

```
print(f"Number of duplicate rows: {duplicates.shape[0]}")
```

Number of duplicate rows: 0

Snippet 5.5. No duplicate rows

With reference to the german.doc file, which was provided along with the dataset, the unique values for each feature were used to check for any data inconsistencies in 14 categorical columns. Snippet 5.6 shows that there were no data inconsistencies.

<pre>#The expected valid values for some of the columns c1 = ['A11', 'A12', 'A13', 'A14'] c3 = ['A30', 'A31', 'A32', 'A33', 'A34'] c4 = ['A40', 'A41', 'A42', 'A43', 'A44', 'A45', 'A46', 'A47', 'A48', 'A49', 'A410'] c6 = ['A61', 'A62', 'A63', 'A64', 'A65'] c7 = ['A71', 'A72', 'A73', 'A74', 'A75'] c9 = ['A91', 'A92', 'A93', 'A94', 'A95'] c10 = ['A101', 'A102', 'A103'] c12 = ['A121', 'A122', 'A123', 'A124'] c14 = ['A141', 'A142', 'A143'] c15 = ['A151', 'A152', 'A153'] c17 = ['A171', 'A172', 'A173', 'A174'] c19 = ['A191', 'A192'] c20 = ['A201', 'A202'] c21 = [1,2] df_dict = {'checking_acct_status': c1, \ 'credit_history': c3, \ 'purpose': c4, \ 'savings_acct_bonds': c6, \ 'employment_duration': c7, \ 'personal_status_sex': c9, \ 'other_debtors_guarantors': c10, \ 'property': c12, \ 'other_installment_plans': c14, \ 'housing': c15, \ 'job_type': c17, \ 'telephone': c19, \ 'foreign_worker': c20, \ 'credit_risk': c21} print("Length of dictionary: ", len(df_dict)) Length of dictionary: 14</pre>	<pre>valid_columns = [] # Iterate through each column in the DataFrame for column in pandas_df.columns: if column in df_dict: valid_values = df_dict[column] is_valid = True # Check each value in the column for idx, value in pandas_df[column].items(): if value not in valid_values: print(f"Invalid value in column '{column}' at index {idx}: {value}") is_valid = False pandas_df.at[idx, column] = None if is_valid: valid_columns.append(column) # Display the valid columns and their count print("\nValid columns:") for col in valid_columns: print(col) print(f"\nNumber of valid columns: {len(valid_columns)}") # Convert back to H2O frame if needed rectified_data_frame = H2OFrame(pandas_df)</pre> <p>Valid columns:</p> <p>checking_acct_status credit_history purpose savings_acct_bonds employment_duration personal_status_sex other_debtors_guarantors property other_installment_plans housing job_type telephone foreign_worker credit_risk</p> <p>Number of valid columns: 14</p>
---	---

Snippet 5.6. Check for Data Inconsistencies

Step 5: Label Encoding

Here, label encoding is done to convert the categorical columns into numerical ones so that they can be fitted by the machine learning models, which only take numerical data. Snippet 5.7 shows the mapping displayed for each categorical feature.

```

from sklearn.preprocessing import LabelEncoder
import pandas as pd
# To Convert the H2O Frame to Pandas DataFrame
le_pandas_df = rectified_data_frame.as_data_frame()

cols = ['checking_acct_status', 'credit_history', 'purpose', 'savings_acct_bonds', 'employment_duration', 'personal_status_sex', \
        'other_debtors_guarantors', 'property', 'other_installment_plans', 'housing', 'job_type', 'telephone', 'foreign_worker']

# Dictionary to store the label encoders and their mappings
label_encoders = {}
mappings = {}

# Apply label encoding to each column
for col in cols:
    le = LabelEncoder()
    mappings[col] = {}

    if col == 'purpose':
        # Add the new category 'A47' to the column temporarily
        temp_series = pd.Series(['A47'])
        temp_col = pd.concat([le_pandas_df[col].astype(str), temp_series], ignore_index=True)
        le.fit(temp_col)

        # Encode the column in the original DataFrame
        le_pandas_df[col] = le.transform(le_pandas_df[col].astype(str))
        # Remove the temporary 'A47' entry if it was added
        if 'A47' in le_pandas_df[col].values:
            le_pandas_df = le_pandas_df[le_pandas_df[col] != le.transform(['A47'])[0]]
    elif col == 'personal_status_sex':
        # Add the new category 'A95' to the column temporarily
        temp_series = pd.Series(['A95'])
        temp_col = pd.concat([le_pandas_df[col].astype(str), temp_series], ignore_index=True)
        le.fit(temp_col)

        # Encode the column in the original DataFrame
        le_pandas_df[col] = le.transform(le_pandas_df[col].astype(str))
        # Remove the temporary 'A95' entry if it was added
        if 'A95' in le_pandas_df[col].values:
            le_pandas_df = le_pandas_df[le_pandas_df[col] != le.transform(['A95'])[0]]
    else:
        le_pandas_df[col] = le.fit_transform(le_pandas_df[col])

    label_encoders[col] = le
    mappings[col] = dict(zip(le.classes_, le.transform(le.classes_)))

# Print the mappings for each column
for col, mapping in mappings.items():
    print(f"Mappings for {col}:")
    print(mapping)

```

Mappings for checking_acct_status:
 ('A11': 0, 'A12': 1, 'A13': 2, 'A14': 3)
 Mappings for credit_history:
 ('A30': 0, 'A31': 1, 'A32': 2, 'A33': 3, 'A34': 4)
 Mappings for purpose:
 ('A41': 0, 'A42': 1, 'A43': 2, 'A44': 3, 'A45': 4, 'A46': 5, 'A47': 6, 'A48': 7, 'A49': 8, 'A49': 10)
 Mappings for savings_acct_bonds:
 ('A61': 0, 'A62': 1, 'A63': 2, 'A64': 3, 'A65': 4)
 Mappings for employment_duration:
 ('A71': 0, 'A72': 1, 'A73': 2, 'A74': 3, 'A75': 4)
 Mappings for personal_status_sex:
 ('A91': 0, 'A92': 1, 'A93': 2, 'A94': 3, 'A95': 4)
 Mappings for other_debtors_guarantors:
 ('A101': 0, 'A102': 1, 'A103': 2)
 Mappings for property:
 ('A121': 0, 'A122': 1, 'A123': 2, 'A124': 3)
 Mappings for other_installment_plans:
 ('A141': 0, 'A142': 1, 'A143': 2)
 Mappings for housing:
 ('A151': 0, 'A152': 1, 'A153': 2)
 Mappings for job_type:
 ('A171': 0, 'A172': 1, 'A173': 2, 'A174': 3)
 Mappings for telephone:
 ('A191': 0, 'A192': 1)
 Mappings for foreign_worker:
 ('A201': 0, 'A202': 1)
 Parse progress: [██████████] (done) 100%

Snippet 5.7. Label Encoding of the categorical Features

Step 6: Adjusting the target of the dataset suitable for binary classification

The dataset has 700 occurrences of the class ‘1’ (70%) and 300 occurrences of the class ‘2’ (30%). However, since many binary classification algorithms assume that the target values are binary (0 and 1), I have to replace values 1 with 0 and values 2 with 1. Moreover, when doing binary classification with H2O AutoML, the target column has to be categorical. Snippet 5.8 shows the codes to perform these tasks.

```

# Replace values 1 with 0 and 2 with 1
rectified_data_frame['credit_risk'] = (rectified_data_frame['credit_risk'] == 1).ifelse(0, 1)

# Convert 'credit_risk' column to categorical
rectified_data_frame['credit_risk'] = rectified_data_frame['credit_risk'].asfactor()

```

Snippet 5.8. Adjusting the target values and target data type

Step 7: Stratified Train-Test Split

To perform a train-test split while ensuring that the credit_risk column in both the train and test sets has 70% of 1 and 30% of 2, I perform stratification, ensuring that the proportion of each class in the target variable is preserved in both the train and test datasets. The left diagram of Snippet 5.9 below shows the process of splitting the dataset with stratification while the right diagram shows the proportion of each class in each dataset, which is 70% and 30%.

```
# Stratification
stratifiedSplit = rectified_data_frame['credit_risk'].stratified_split(test_frac=0.2, seed=42)

# Train dataframe
train = rectified_data_frame[stratifiedSplit=="train"]

# Test dataframe
test = rectified_data_frame[stratifiedSplit=="test"]

# Print the shape of train and test sets
print(f"Training set shape: {train.shape}")
print(f"Testing set shape: {test.shape}")

Training set shape: (800, 21)
Testing set shape: (200, 21)

# Function to calculate and print unique values and proportions
def print_unique_values_and_proportions(df, column_name, dataset_name):
    counts = df[column_name].table()
    total_count = counts['Count'].sum()
    proportions = (counts['Count'] / total_count) * 100

    print(f"Unique values and proportions in {dataset_name} dataset:")
    for level, count, proportion in zip(counts['credit_risk'].as_data_frame().values.flatten(), counts['Count'].as_data_frame().values.flatten(), proportions):
        print(f"\tValue: {level}, Count: {count}, Proportion: {proportion:.2f}%")

# Print unique values and proportions for 'credit_risk' in each dataset
print_unique_values_and_proportions(rectified_data_frame, "credit_risk", "full")
print_unique_values_and_proportions(train, "credit_risk", "training")
print_unique_values_and_proportions(test, "credit_risk", "testing")

Unique values and proportions in full dataset:
Value: 0, Count: 700, Proportion: 70.00%
Value: 1, Count: 300, Proportion: 30.00%

Unique values and proportions in training dataset:
Value: 0, Count: 560, Proportion: 70.00%
Value: 1, Count: 240, Proportion: 30.00%

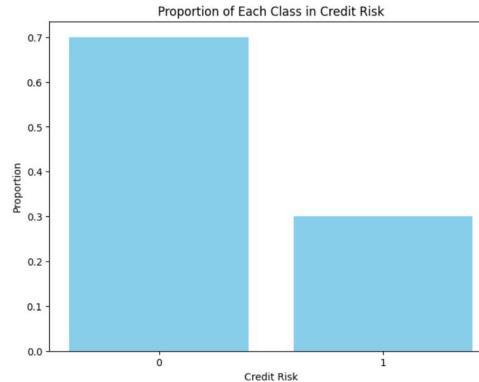
Unique values and proportions in testing dataset:
Value: 0, Count: 140, Proportion: 70.00%
Value: 1, Count: 60, Proportion: 30.00%
```

Snippet 5.9. Stratified Train-Test Split

Step 8: Some Exploratory Data Analyses before Class Balancing

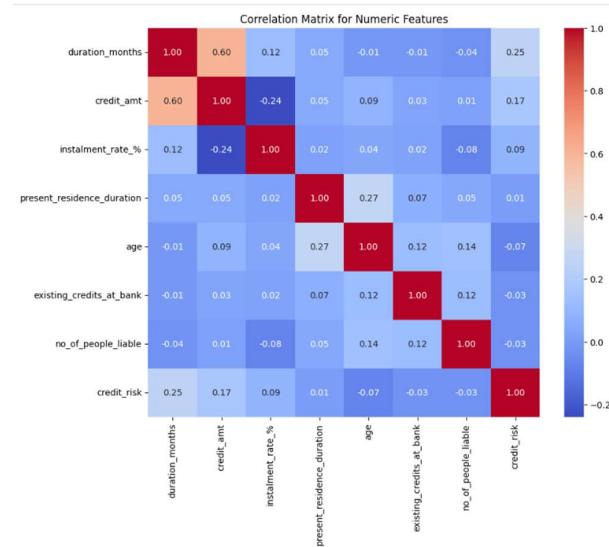
*Note that not all data visualizations are shown here. Please refer to the coding files for more of these.

Snippet 5.10 confirms that 70% and 30% of the training data belong to 0 (Good Credit) and 1 (Bad Credit) respectively.



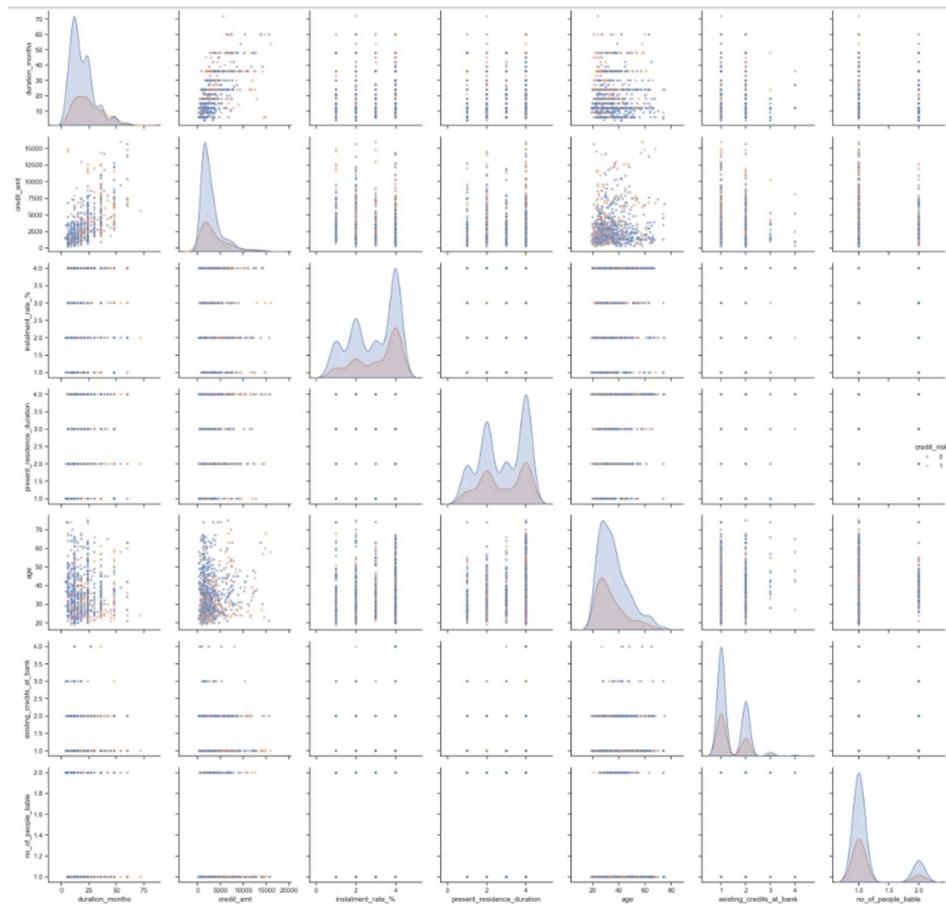
Snippet 5.10. Proportion of Classes in Training Dataset

Snippet 5.11 shows the correlation matrix for the numeric variables. Most of the variables have weak correlations (0.1 to 0.4 or -0.1 to -0.4) with one another while only a few variables have moderate (0.4 to 0.7 or -0.4 to -0.7) correlations with one another. Since there are no highly correlated variables here, there is none to be removed.



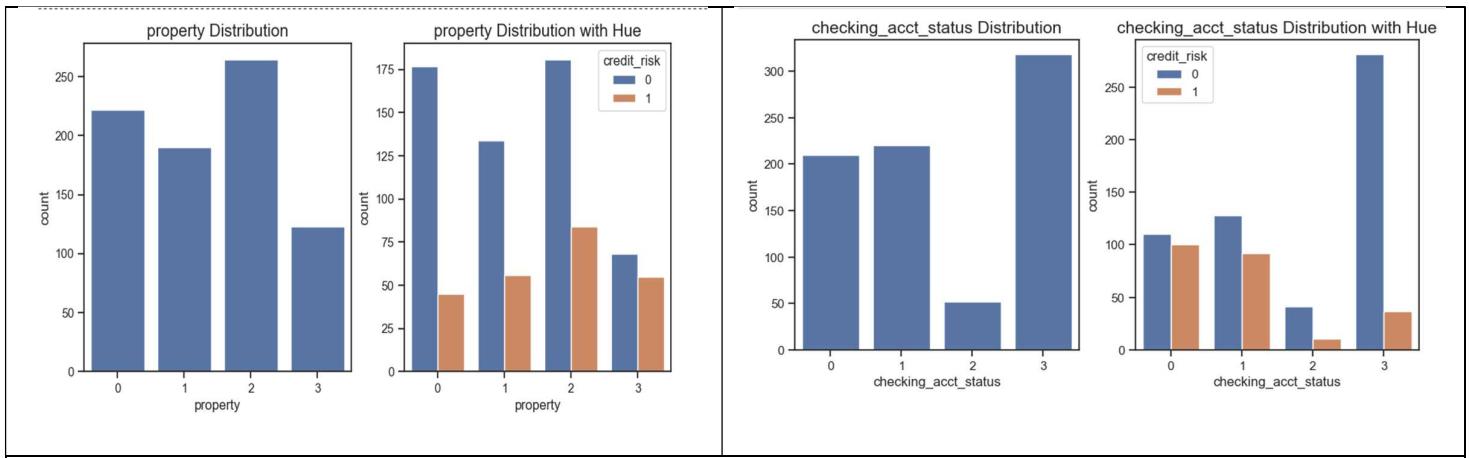
Snippet 5.11. Correlation Matrix for Numeric Variables

Snippet 5.12 shows a collection of pair plots for numeric variables with hue indicating the credit risk. The blue colour represents “Good Credit” while the orange colour represents “Bad Credit”. The hue helps us identify how the relationships between variables vary based on the target. The KDE plots indicate that the data distribution for 'duration_months' and 'present_residence_duration' are quite different between the 2 classes. The scatter plots show that the patterns and correlations of the data are approximately the same for both classes as their points are plotted at approximately the same locations.



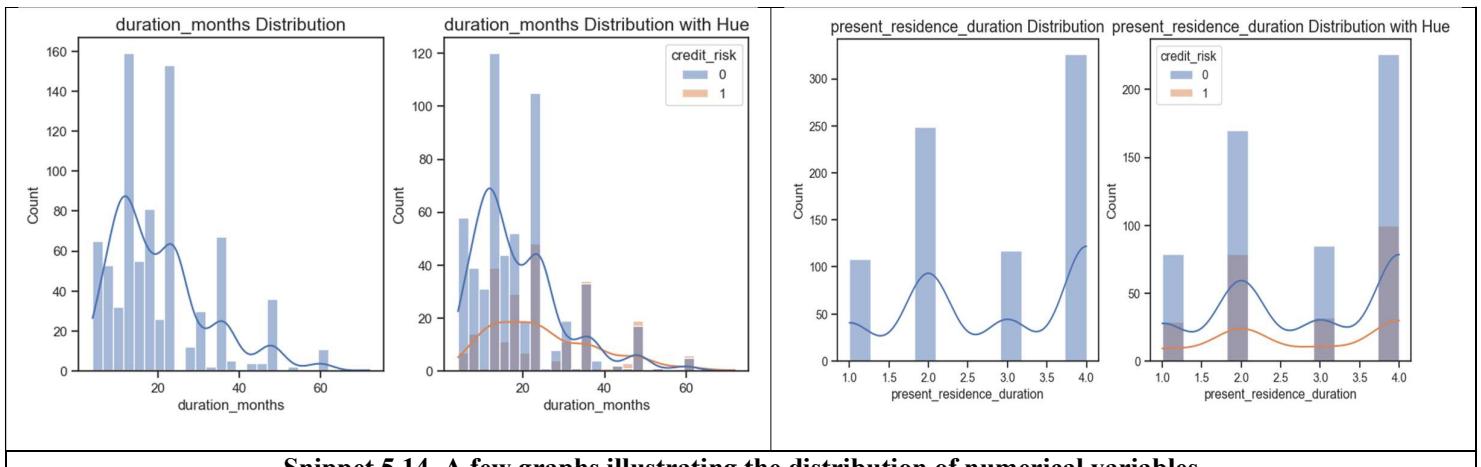
Snippet 5.12. Pair plots for Numeric Features with Hue

Graphs to illustrate the distribution of categorical variables were also plotted. Snippet 5.13 shows a few examples: graphs for ‘property’ and ‘checking_acct_status’. Referring to the coding file, with the target put into consideration, for some variables, the distribution of the categories is not the same, such as ‘checking_acct_status’, ‘purpose’ and ‘employment_duration’. For some variables, the distribution of the categories is the same, such as ‘credit_history’ and ‘other_installment_plans’. The chart of the ‘property’ column shows that each category has a comparable frequency. Hence, there are enough rows for each category in the data for the ML algorithm to learn effectively. The bars belonging to class 0 are generally taller than the bars belonging to class 1.



Snippet 5.13. A few graphs illustrating the distribution of categorical variables

Graphs to illustrate the distribution of numerical variables were also plotted. Snippet 5.14 shows a few examples: graphs for ‘duration_months’ and ‘present_residence_duration’. ReThe distribution of data for duration_months, credit_amt, age, existing_credits_at_bank and no_of_people_liable are right skewed. The distribution of data for instalment_rate_% and present_residence_duration are left skewed.



Snippet 5.14. A few graphs illustrating the distribution of numerical variables

Step 9: Balancing the Imbalanced Training Dataset

To balance the training dataset, I used the ADASYN technique, which generates synthetic samples for the minority class, converts the resampled training data into an H2O frame and calculates the proportion of the 'credit_risk' classes in the balanced dataset. Finally, it displays the proportions of each class in the balanced training set, which shows that the dataset is more evenly distributed. Note that only the training dataset is balanced to improve model learning, while the test dataset remains unbalanced to ensure an accurate evaluation of the model's real-world performance. Refer to Snippet 5.15.

```
from imblearn.over_sampling import ADASYN
ada = ADASYN(sampling_strategy='minority', random_state=42)

train_pd = train.as_data_frame()

X_train = train_pd.drop(columns=['credit_risk'])
y_train = train_pd['credit_risk']

X_train_resampled, y_train_resampled = ada.fit_resample(X_train, y_train)

train_balanced_pd = pd.DataFrame(X_train_resampled, columns=X_train.columns)
train_balanced_pd['credit_risk'] = y_train_resampled

train_balanced_h2o = h2o.H2OFrame(train_balanced_pd)

# Get the frequency table for 'credit_risk' in the balanced training set
credit_risk_counts_balanced = train_balanced_h2o["credit_risk"].table()

# Convert the H2OFrame to a pandas DataFrame
credit_risk_counts_balanced_df = credit_risk_counts_balanced.as_data_frame()

# Calculate the proportions in percentage
credit_risk_counts_balanced_df['Proportion'] = (credit_risk_counts_balanced_df['Count'] / credit_risk_counts_balanced_df['Count'].sum()) * 100

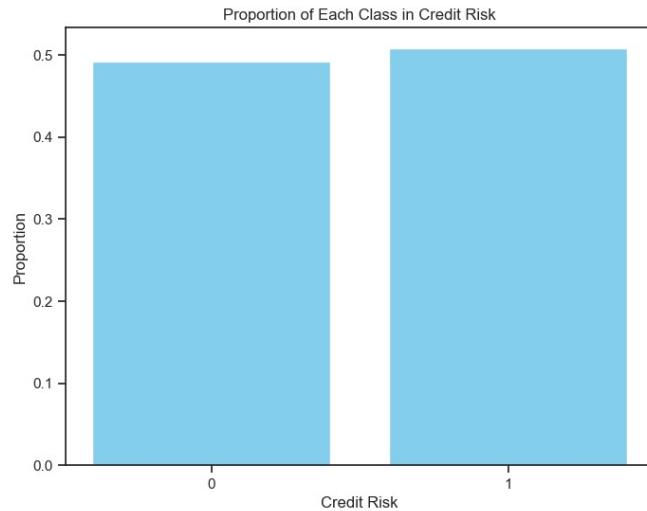
# Display the proportions
print("Proportions in the balanced training set:")
print(credit_risk_counts_balanced_df)

Parse progress: ||| (done) 100%
Proportions in the balanced training set:
  credit_risk  Count  Proportion
0            0    560   49.209139
1            1    578   50.790861
```

Snippet 5.15. Balancing the Imbalanced Training Dataset

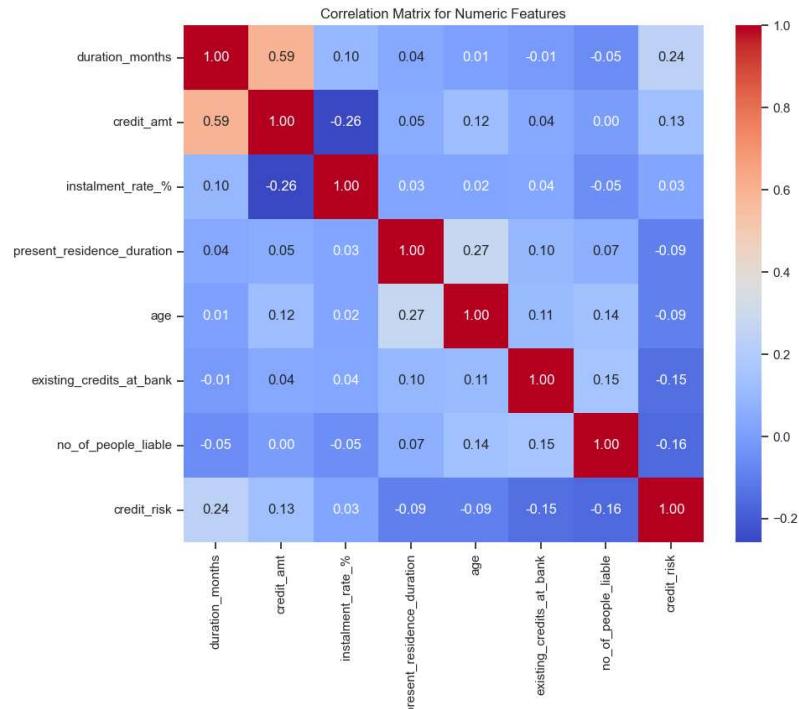
Step 10: EDA after Balancing

The proportions of the **target variable classes** after balancing is visually shown below. They are almost the same.



Snippet 5.16

If you compare the colours in the **Correlation Matrix** before and after balancing the training data, they are almost the same. If you look closer, there are minor changes.



Snippet 5.17

The following code snippet is another way to determine whether or not there are **variables with noticeable correlations** with each other:

```
necessary_cols = train_all_pd.columns
corr_unstacked = train_all_pd[necessary_cols].corr().unstack().abs()
ranked_corr = corr_unstacked.sort_values(kind="quicksort", ascending=False)
ranked_corr = ranked_corr[ranked_corr != 1]

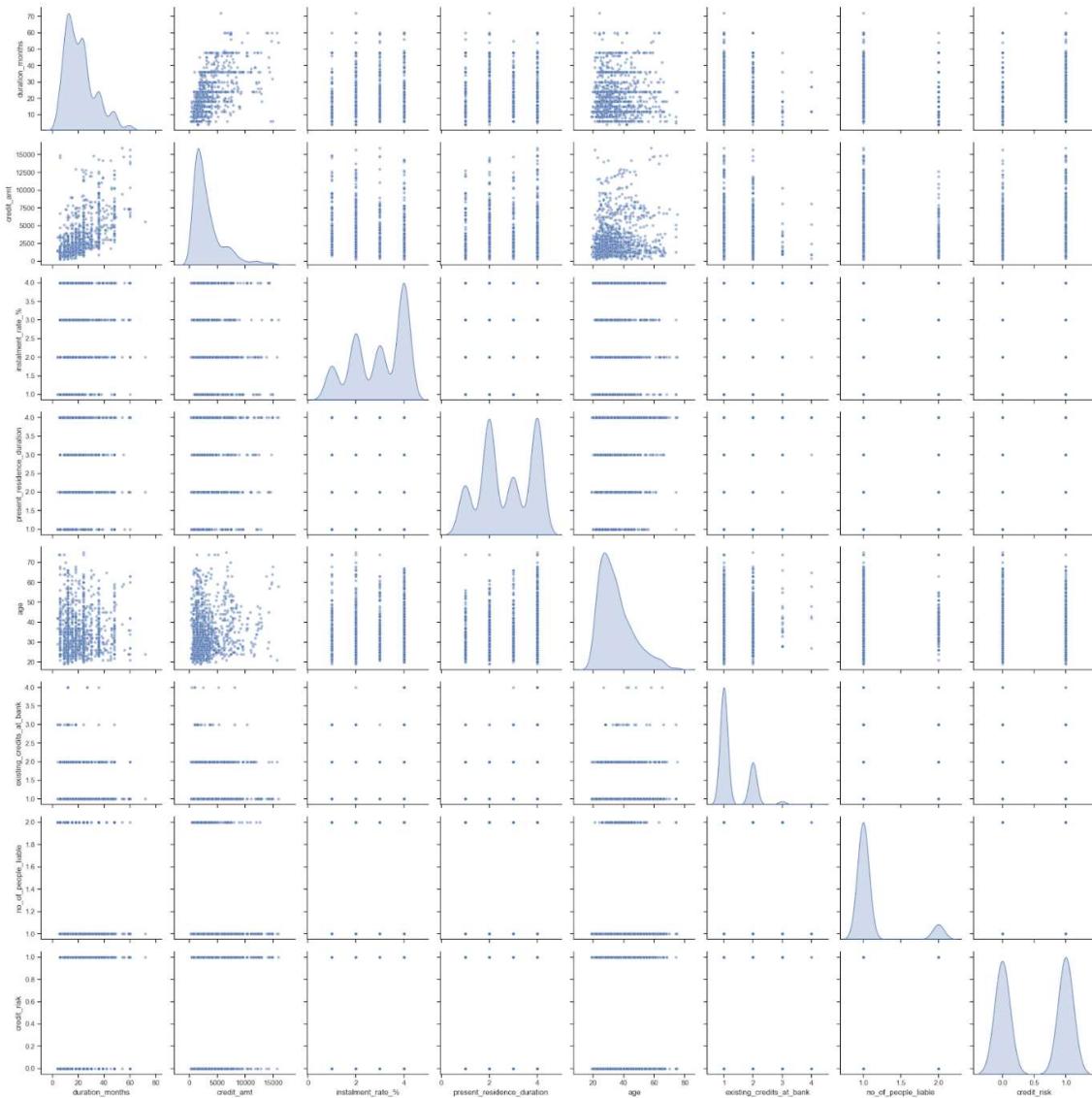
ranked_corr.head(10)
```

credit_amt	duration_months	0.594764
duration_months	credit_amt	0.594764
credit_history	existing_credits_at_bank	0.472106
existing_credits_at_bank	credit_history	0.472106
credit_risk	checking_acct_status	0.460655
checking_acct_status	credit_risk	0.460655
job_type	telephone	0.400633
telephone	job_type	0.400633
property	credit_amt	0.323873
credit_amt	property	0.323873

dtype: float64

Snippet 5.18

Note that the correlations are listed for both directions, which is why you see pairs of the same numbers. The highest correlation, beyond 0.5, is that between “credit_amt” and “duration_months”. Since they are still below the high correlation category (0.7 to 0.9x), neither of them will be dropped. Next, let’s see the **Pairplots for the Numeric features**:



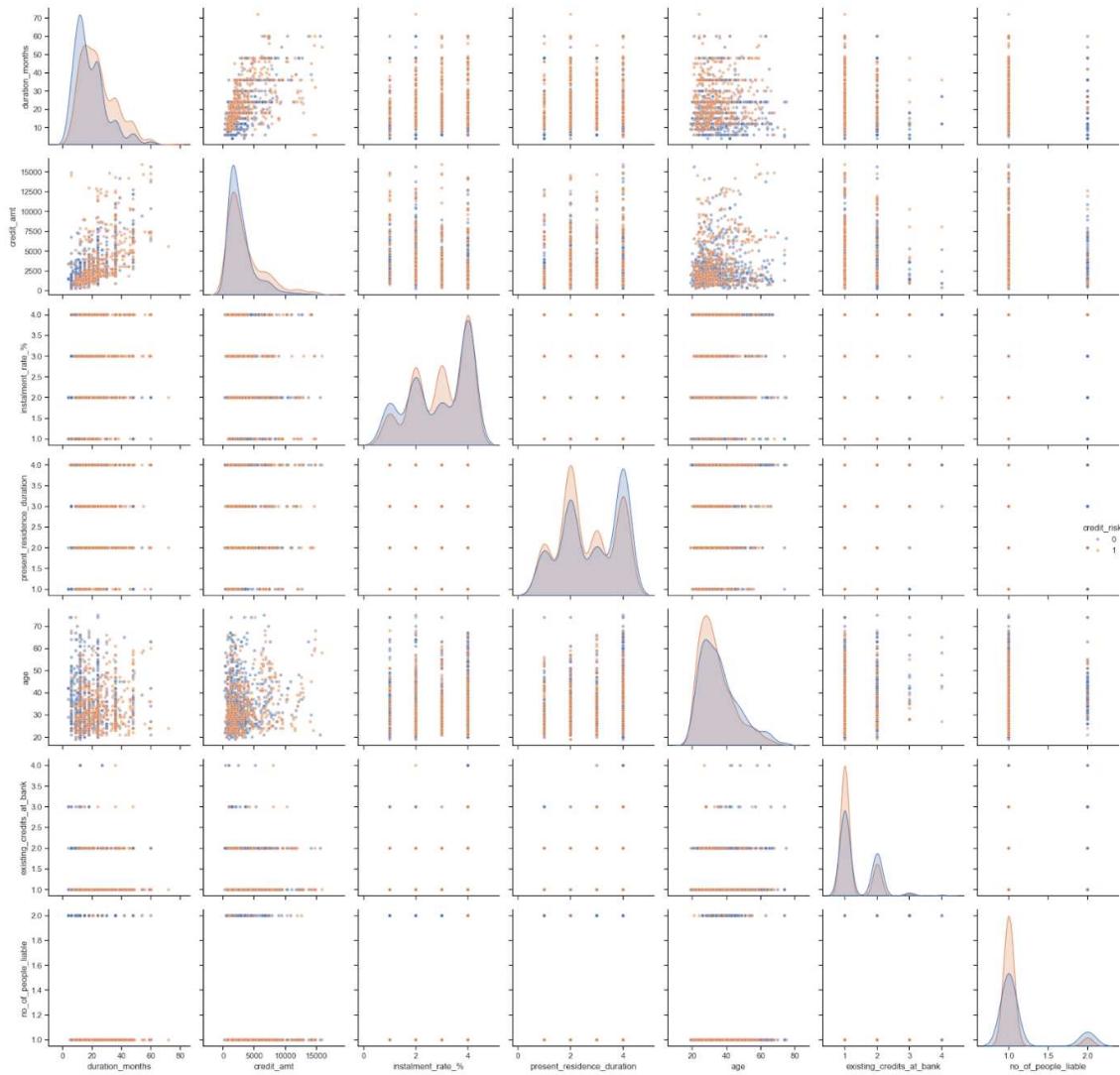
Snippet 5.19

The distribution of data for almost all the variables here is not normally distributed. They are either left-skewed or right-skewed. However, referring to the bottom right plot, the equal height of the cones at 0 and 1 indicates that the two classes (0 and 1) in the credit_risk variable are approximately equally represented in the training dataset after balancing.

Referring to the off-diagonal plots, most of the variables have very little correlation with each other, such as 'between present_residence_duration and no_of_people_liable' and 'between existing_credits_at_bank and present_residence_duration'.

Only a few variables have some correlation with each other, such as 'between duration_months and credit_amt'

The Pairplots for **Numeric features** (With Hue as credit_risk) are as follows:

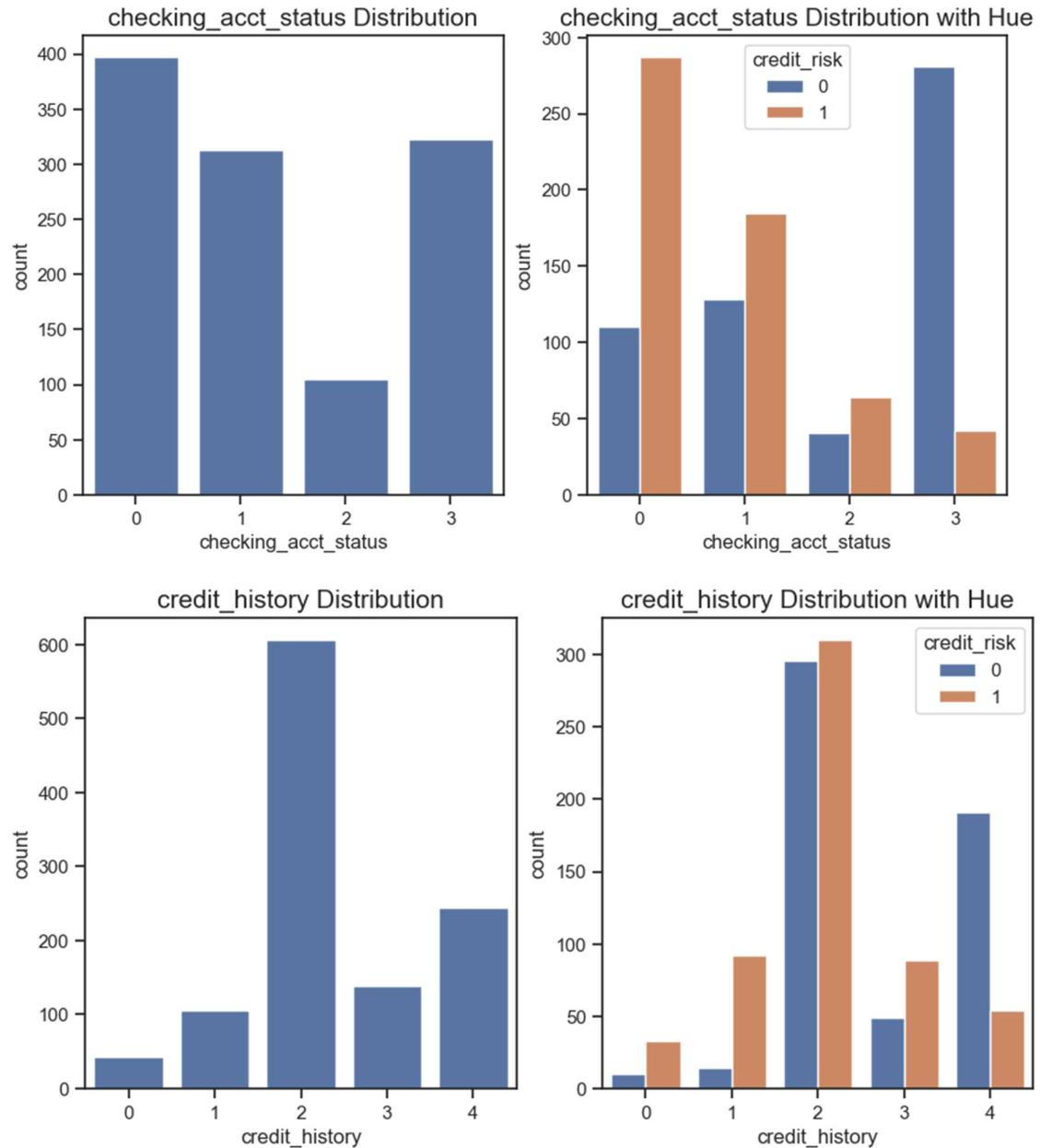


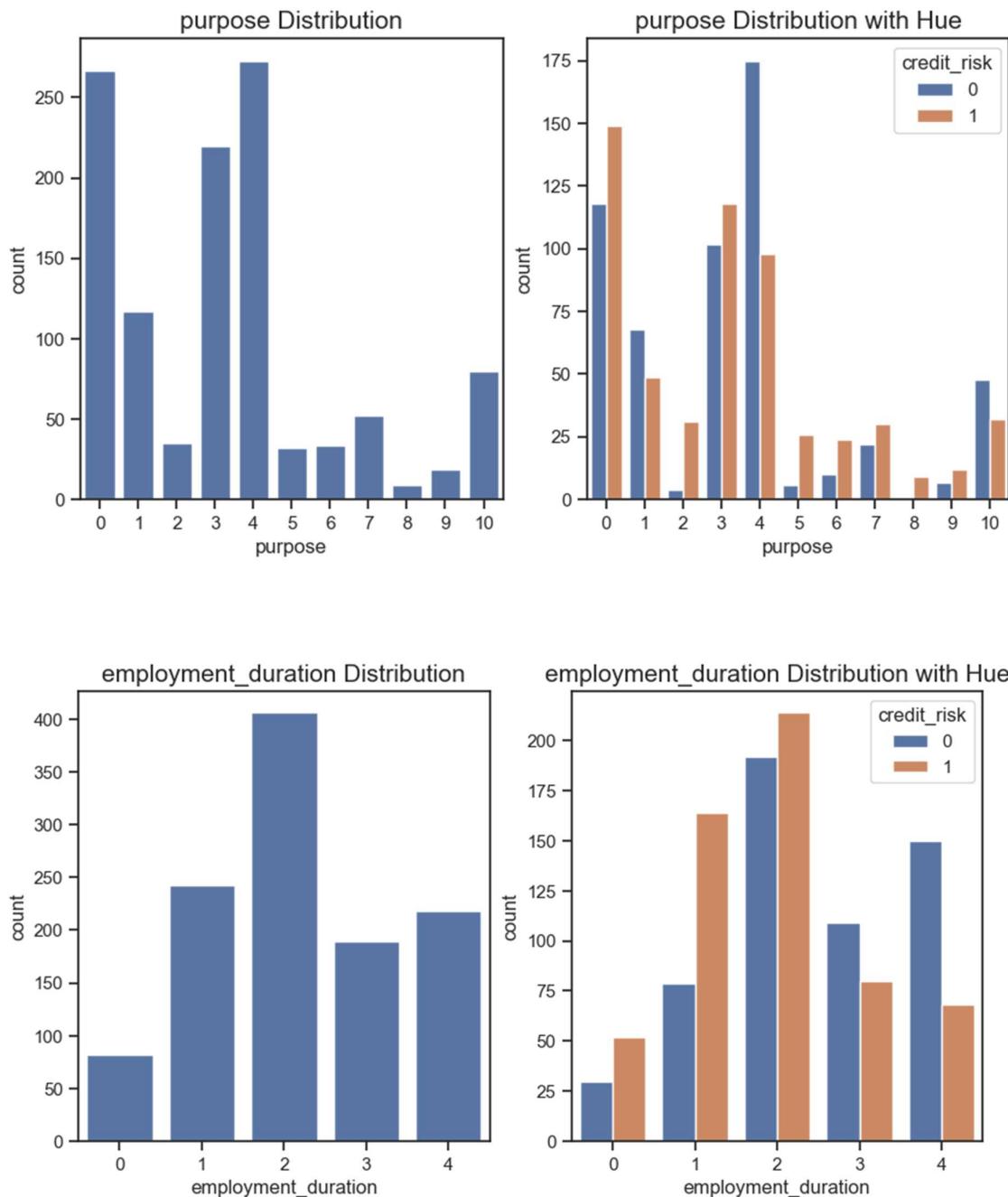
Snippet 5.20

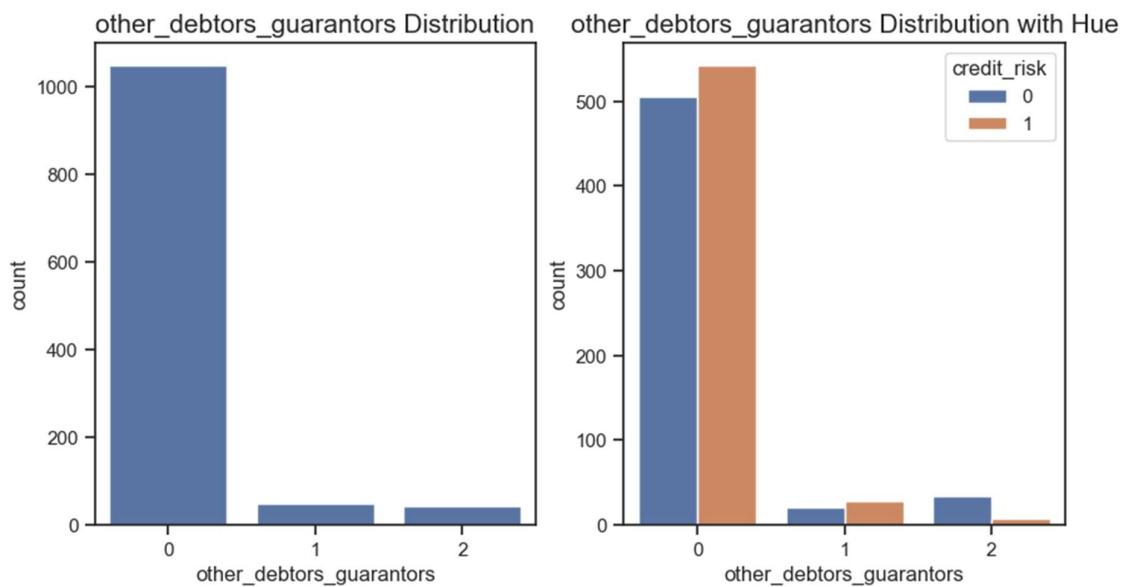
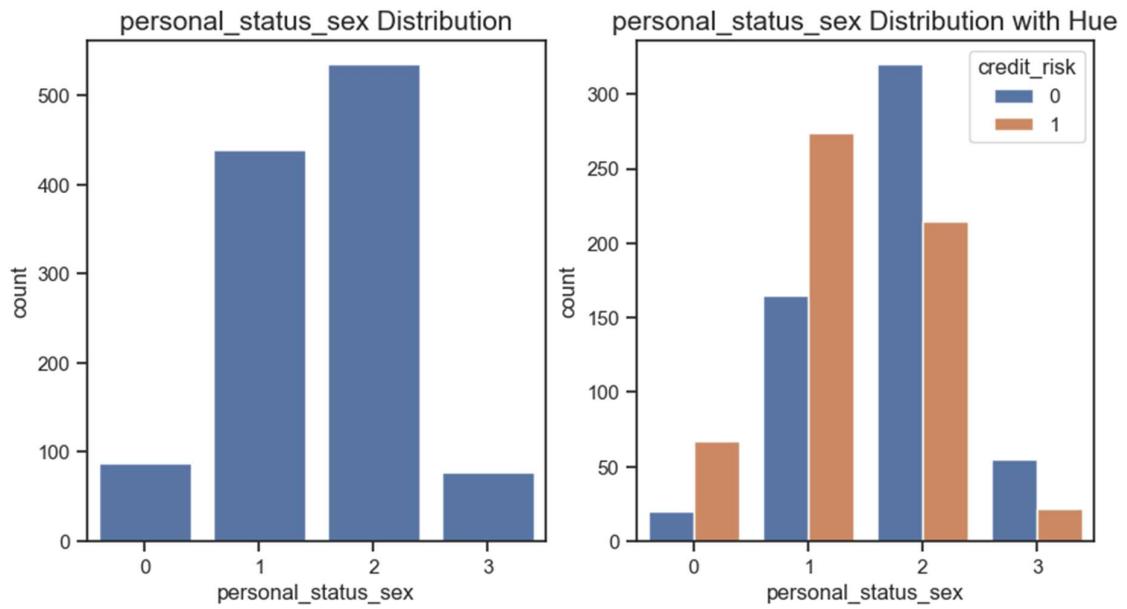
The density estimates belonging to class 1 have increased in size and height and most of their distribution has become quite similar to the one belonging to class 0.

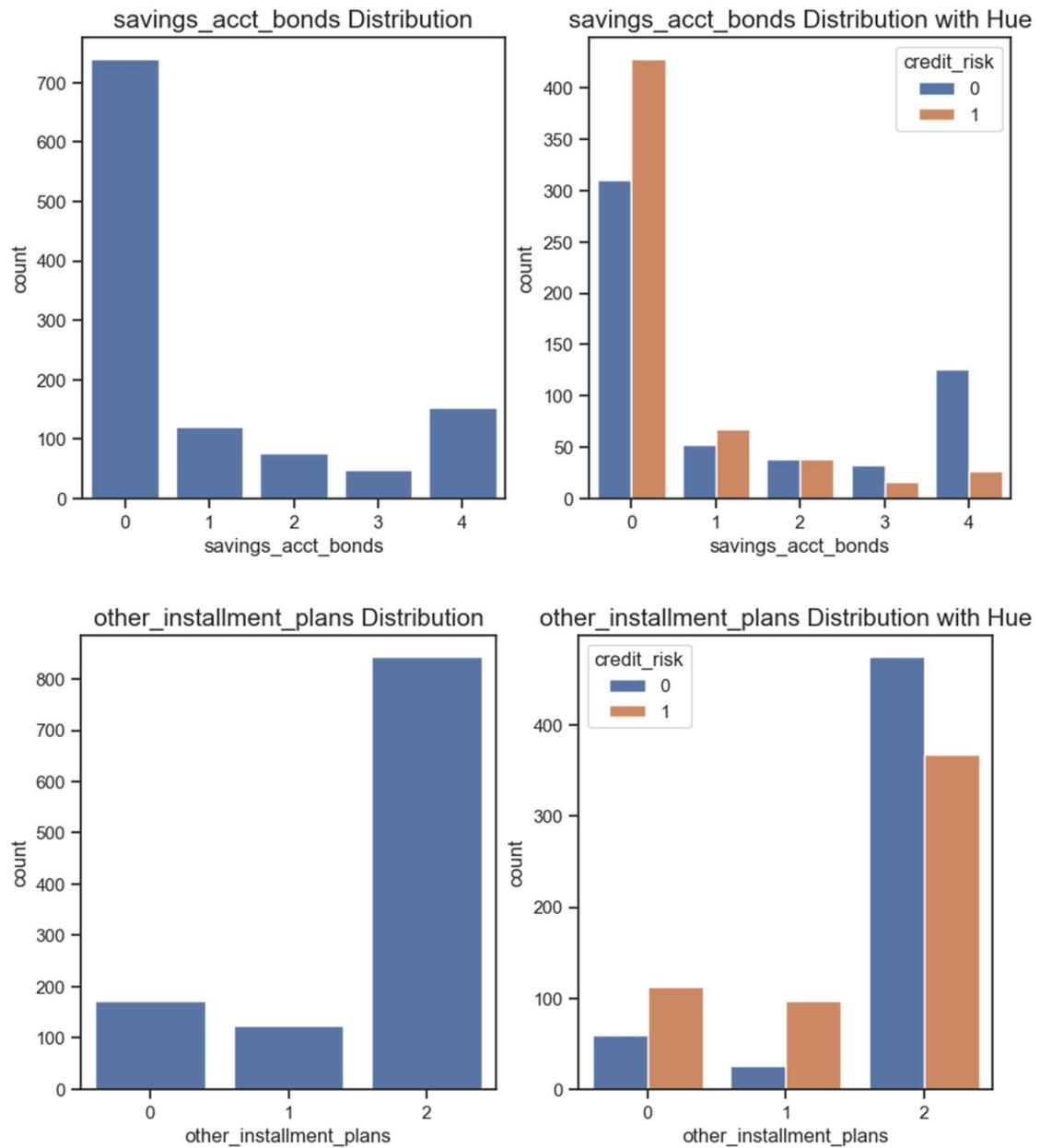
As for the scatter plots, they appear similar to the ones plotted before balancing in terms of pattern & correlation. However, it is obvious that the number of records belonging to class 1 has increased significantly because of the increased number of orange points.

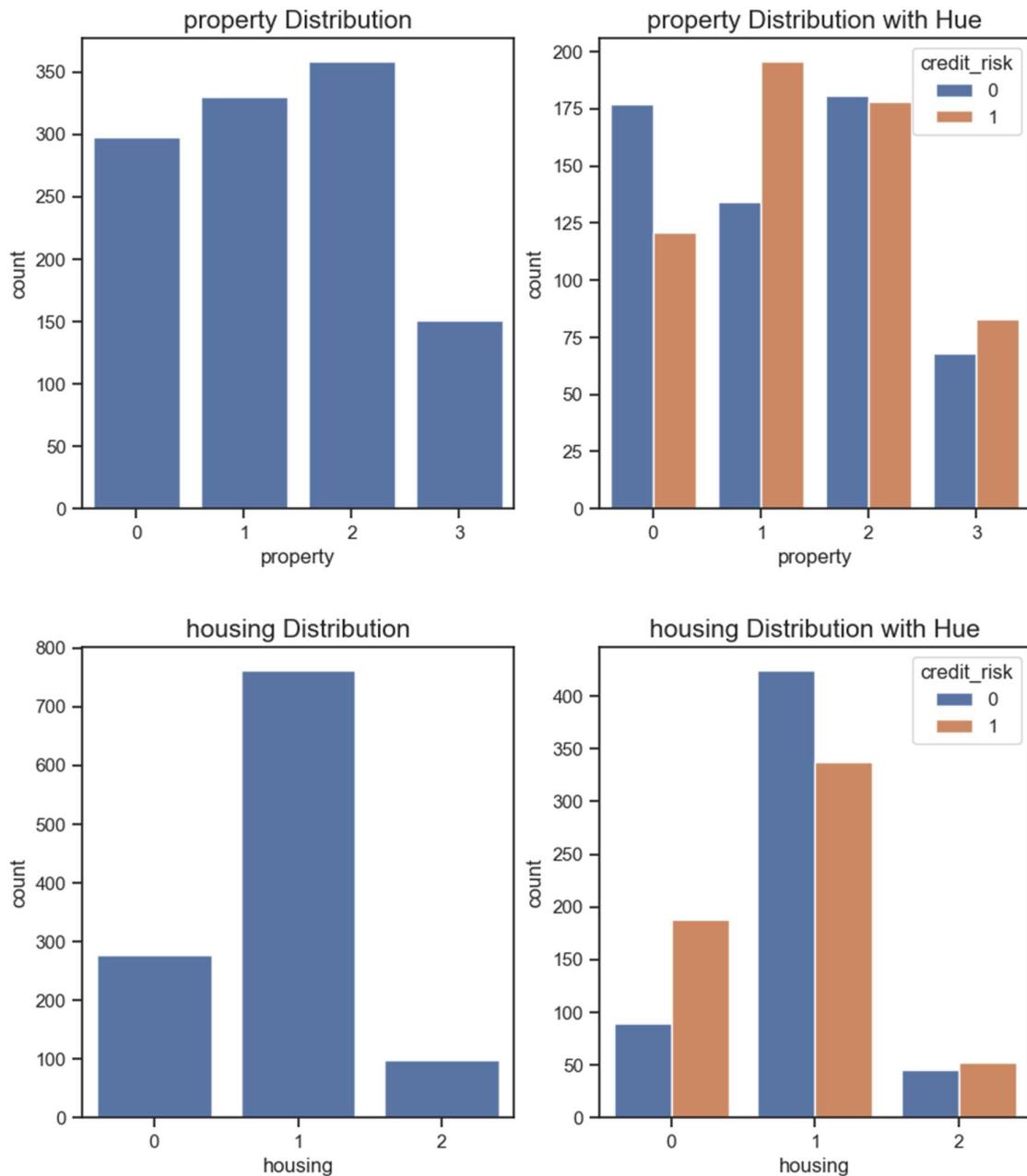
Next, we consider the distributions of **Categorical features with Hue**:

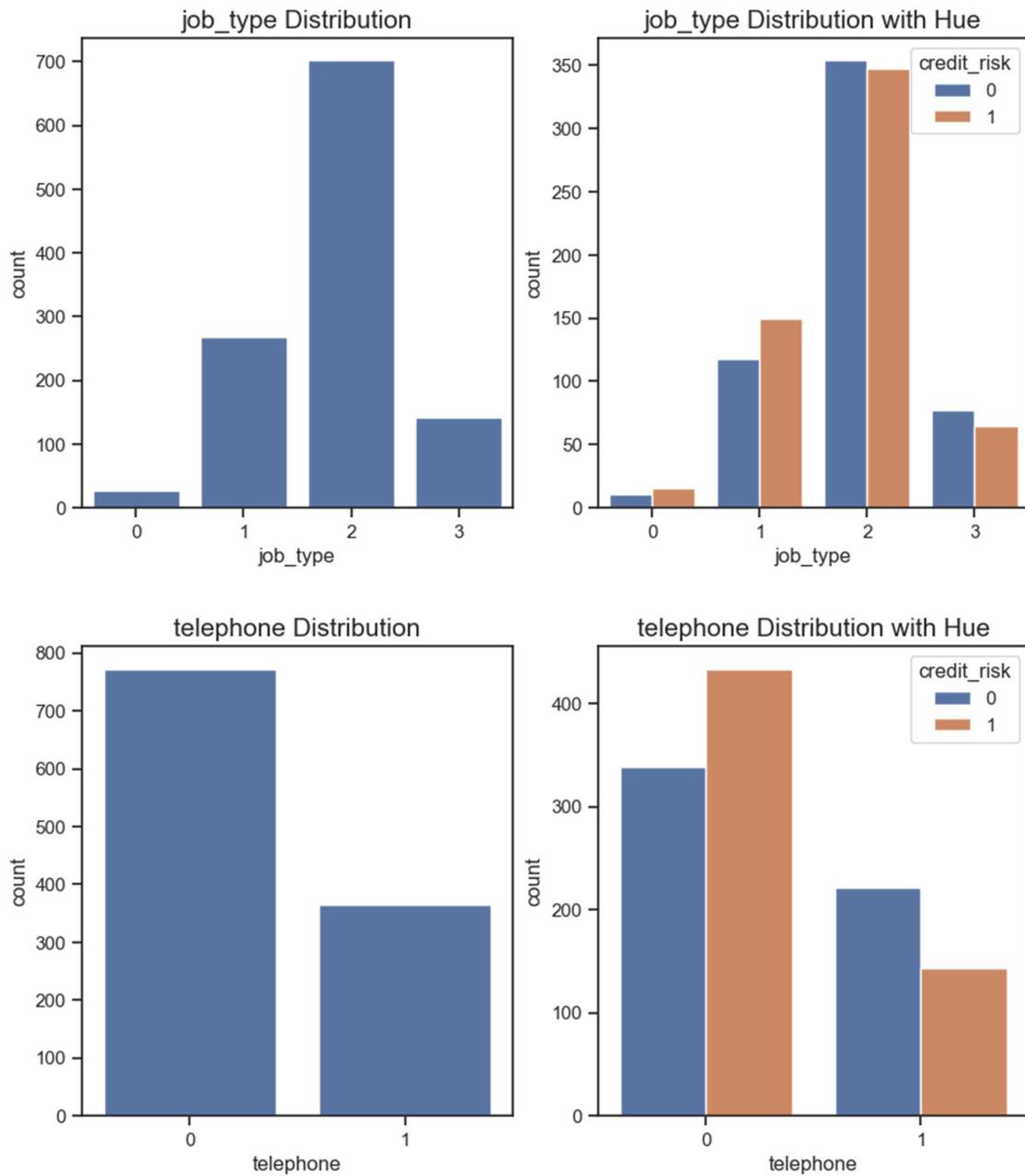


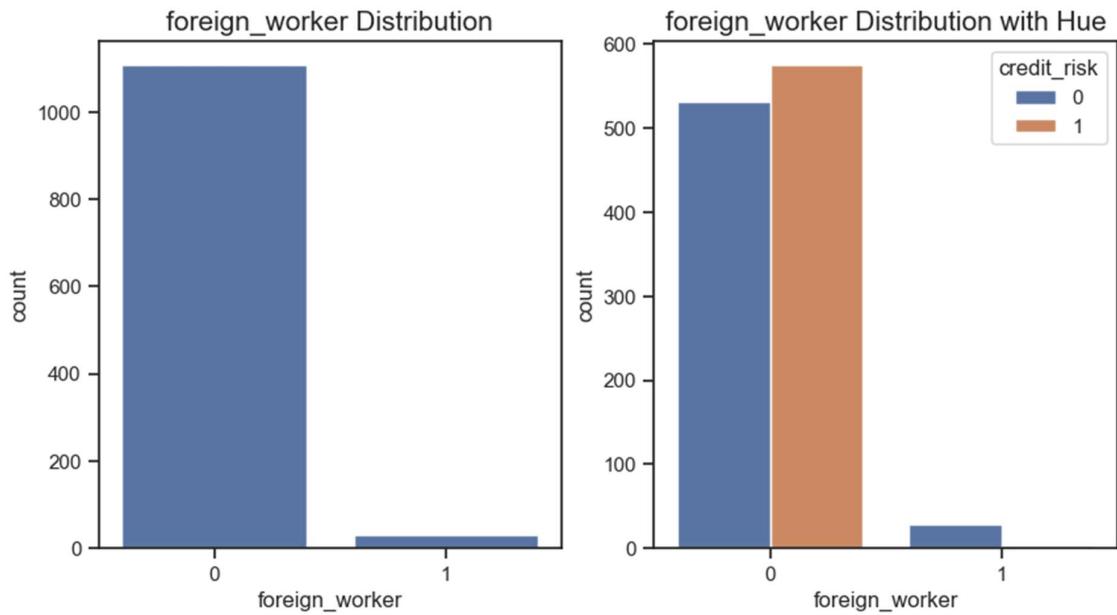










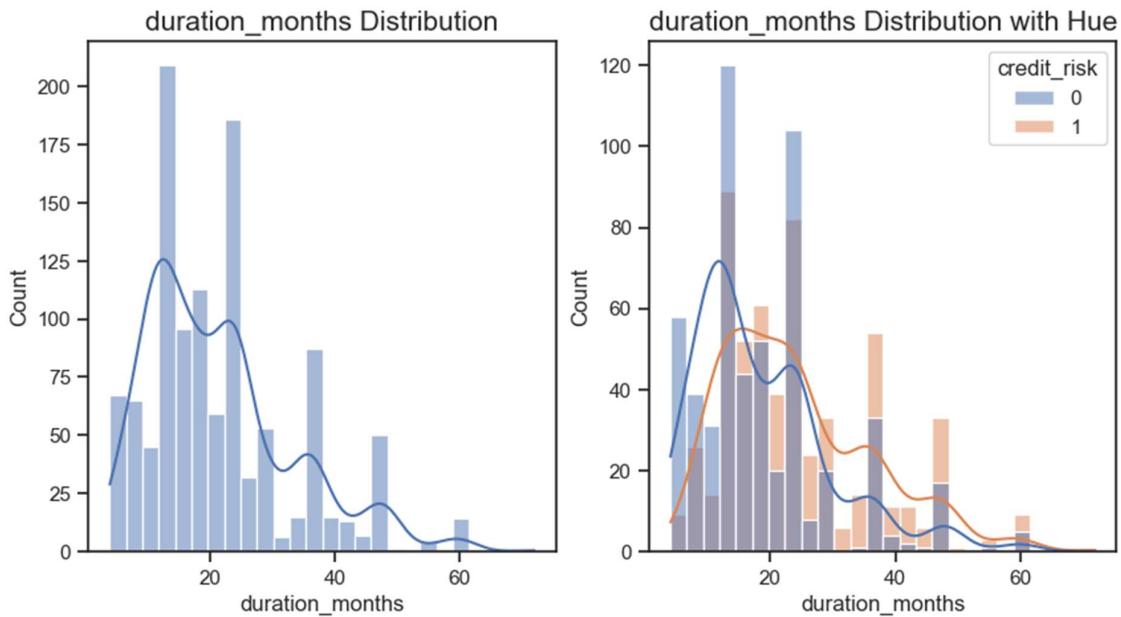


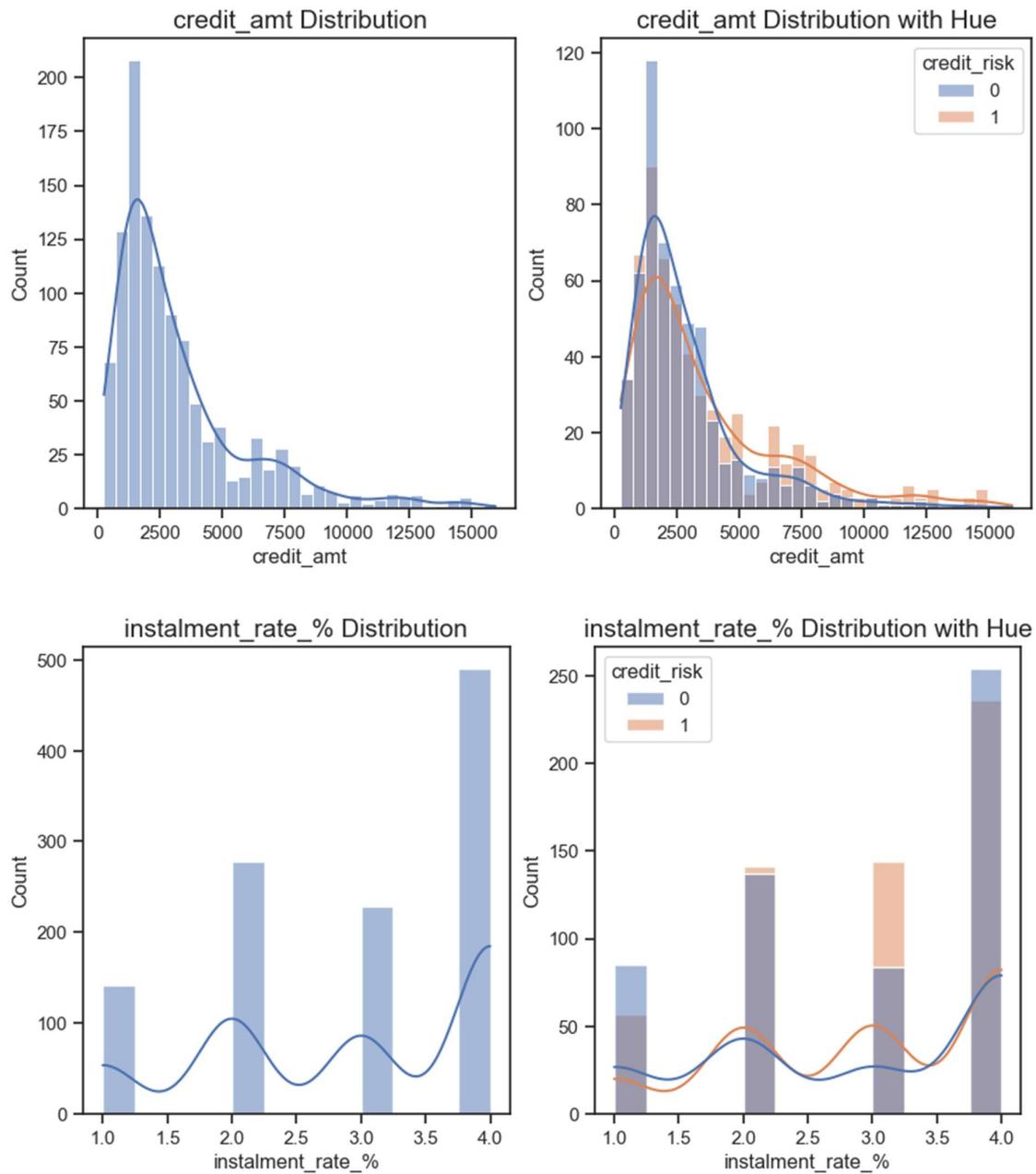
Snippets 5.21

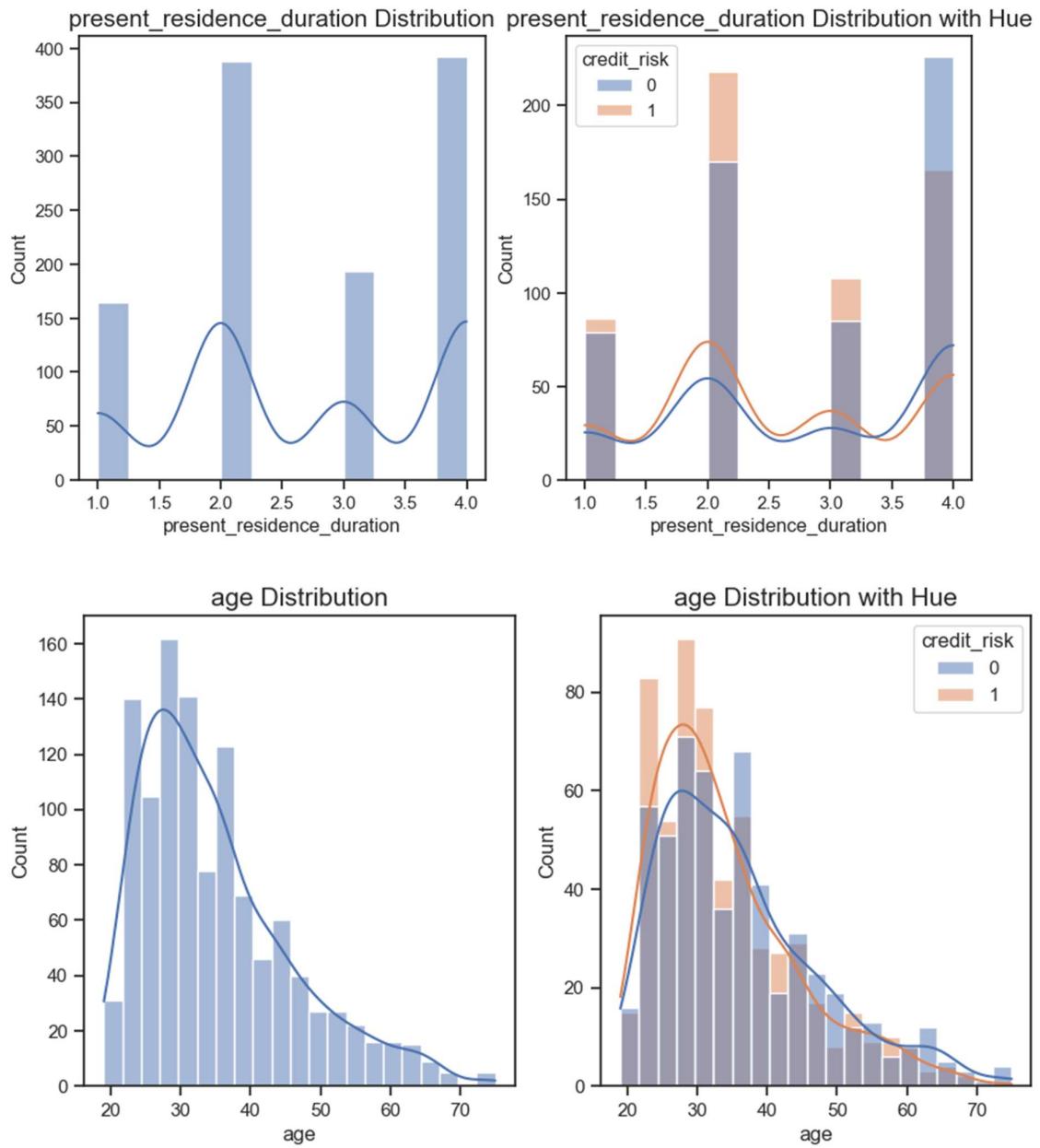
With the target put into consideration, for all variables, the distribution of the categories is not the same, such as 'checking_acct_status', 'purpose' and 'employment_duration'.

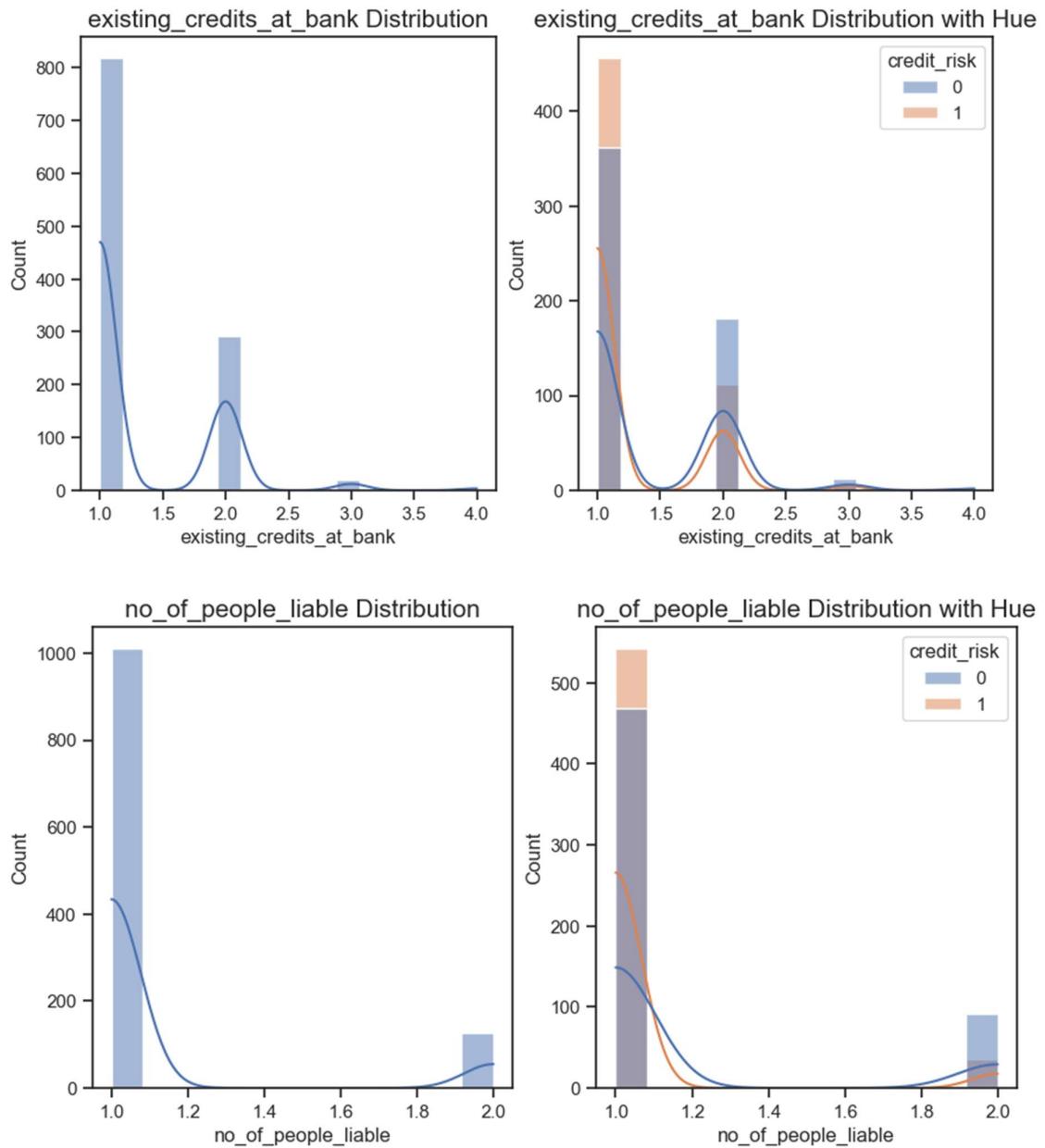
The chart of the 'property' column shows that each category has a comparable frequency. Hence, there are enough rows for each category in the data for the ML algorithm to learn effectively.

We now look at the **distribution of Numerical features with Hue**.





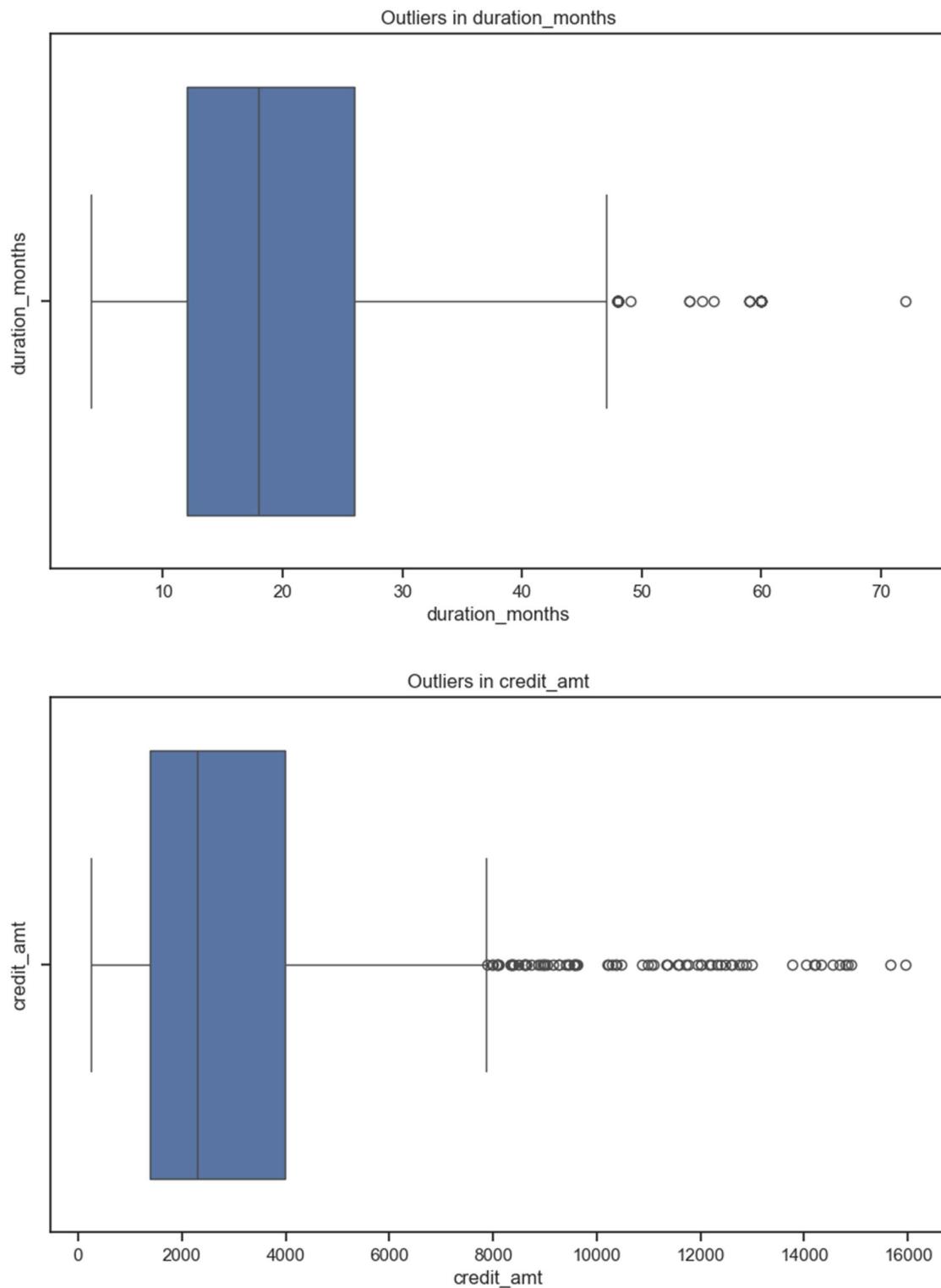


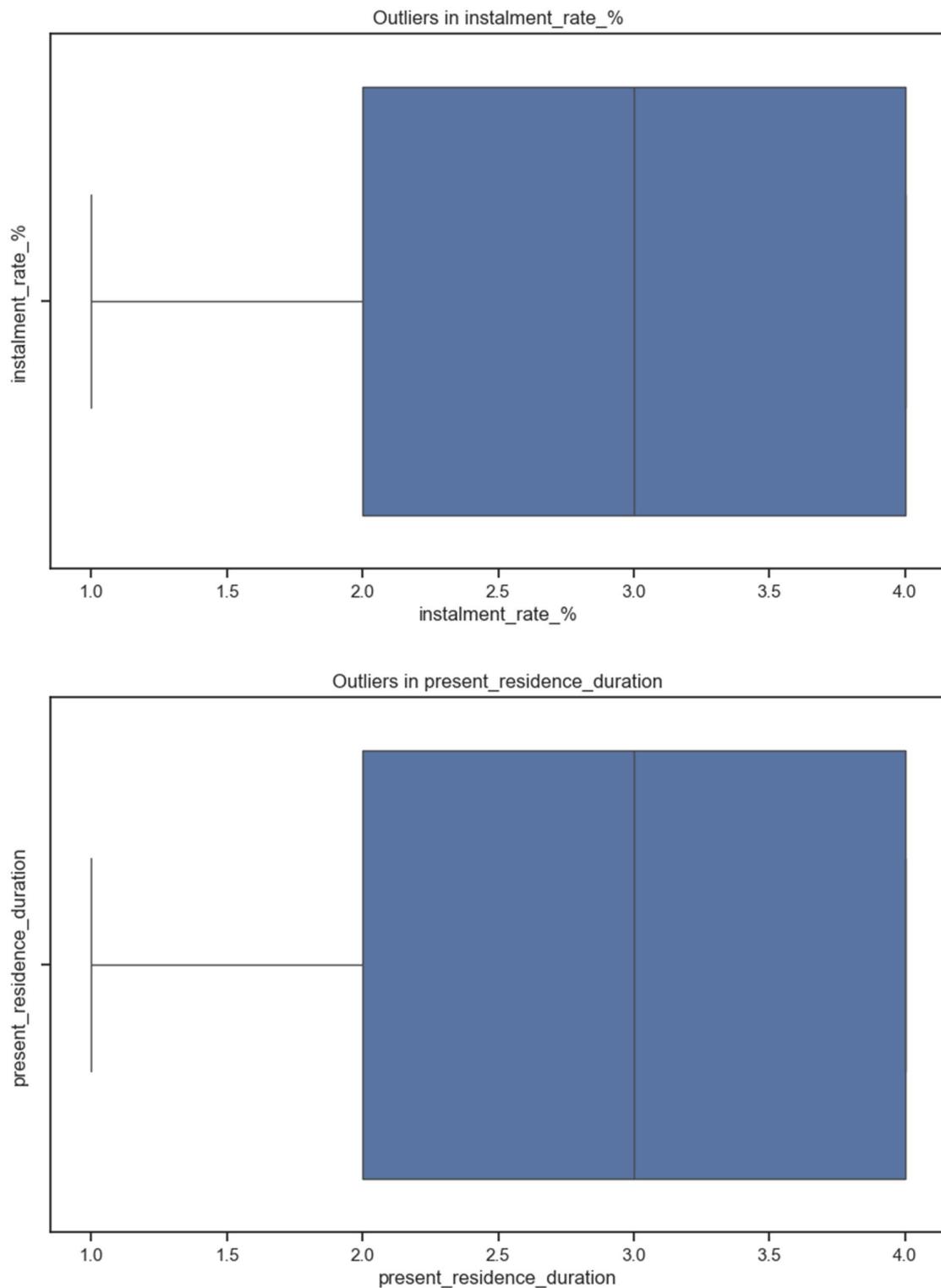


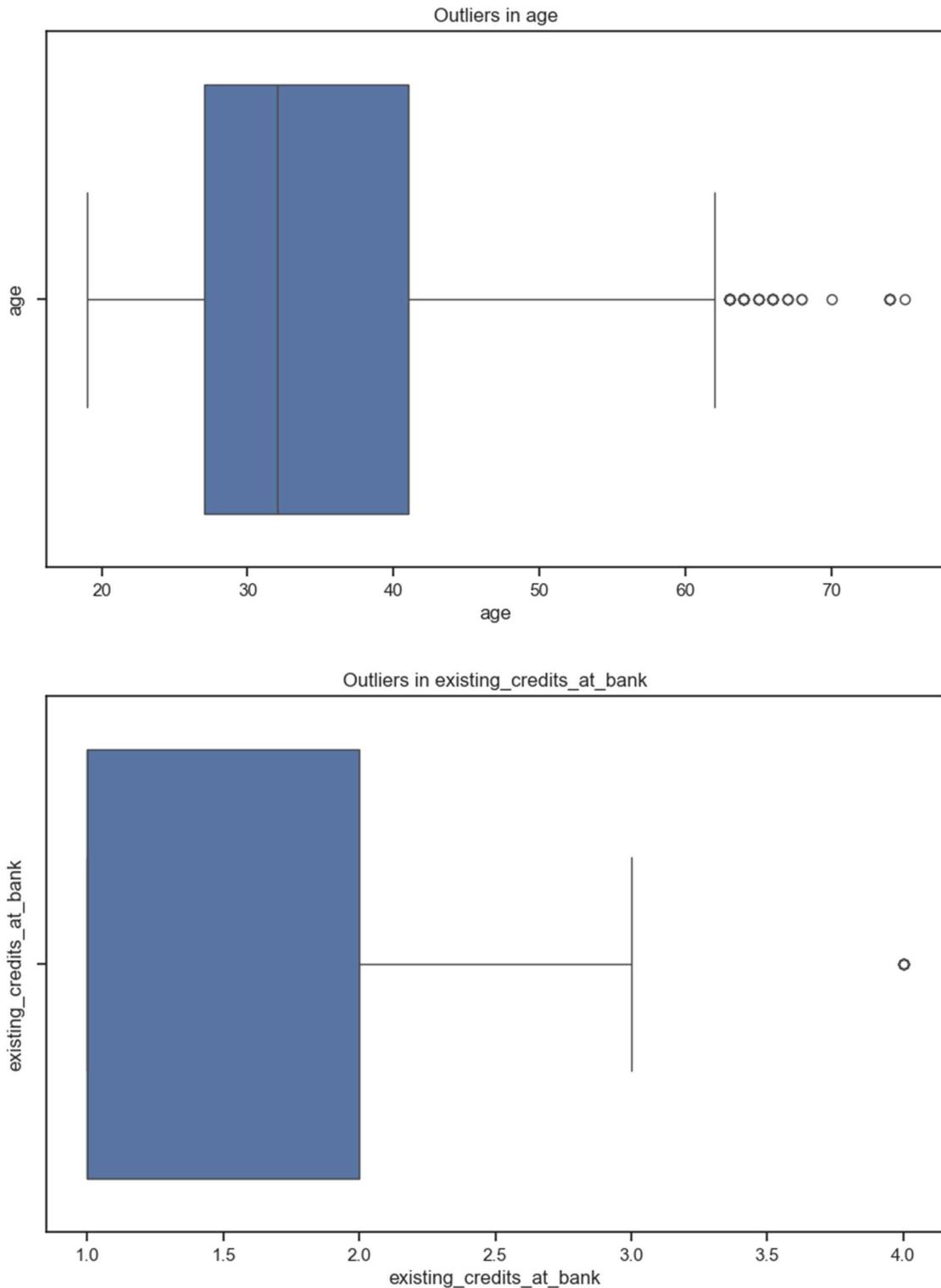
Snippets 5.22

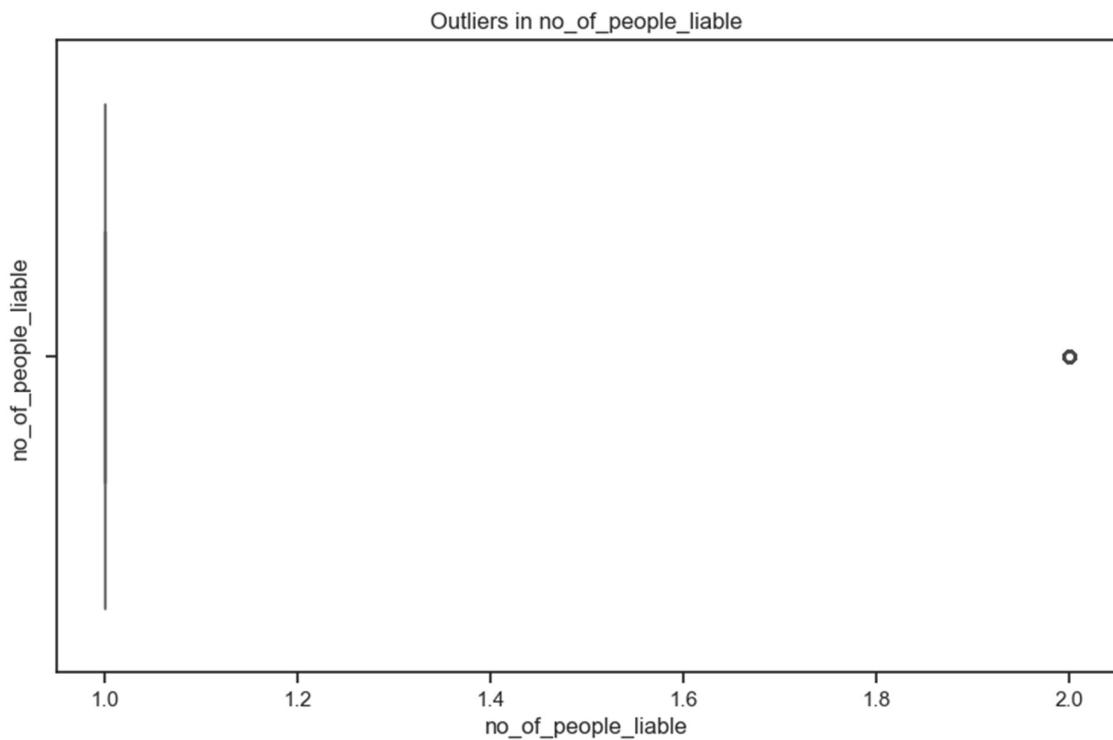
The distribution of data for `duration_months`, `credit_amt`, `age`, `existing_credits_at_bank` and `no_of_people_liable` are right skewed. The distribution of data for `instalment_rate_%` and `present_residence_duration` are left skewed.

We now look at possible **Outliers** by generating BoxPlots:









Snippets 5.23

From the boxplots, outliers are found in the columns: duration_months, credit_amt, age, existing_credits_at_bank, no_of_people_liable.

Although the boxplots indicate outliers in some columns, I believe the values are not wrong or out-of-scale. All of the numeric columns have skewed distributions and handling outliers in skewed datasets without careful consideration could result in the loss of important information or lead to incorrect conclusions about the underlying patterns in the data. Therefore, I am not treating any outliers.

Step 10: Perform Feature Engineering

H2O AutoML does not provide any special support for feature engineering so normal hand-written Python code is used to perform this task.

a) duration_months

- Short term (0-12 months): Generally, a duration of up to one year is considered short term. (Represented by 1)

- Medium term (13-36 months): Durations from over a year to three years are often considered medium term, capturing a mid-range period. (Represented by 2)
- Long term (37 months and above): Durations longer than three years typically fall into the long-term category, representing more extended periods. (Represented by 3)

```
# This pandas dataframe will be feature engineered
train_balanced_pd = train_balanced_h2o.as_data_frame()
# Categorize duration_months with string labels (Training Dataset)
train_balanced_pd['duration_category'] = pd.cut(train_balanced_pd['duration_months'],
                                                bins=[0, 12, 36, float('inf')],
                                                labels=['Short term', "Medium term", "Long term"])

# Map string labels to integers
duration_mapping = {"Short term": 1, "Medium term": 2, "Long term": 3}
train_balanced_pd['duration_category'] = train_balanced_pd['duration_category'].map(duration_mapping).astype(int)

[ ] # This pandas dataframe will be feature engineered
test_pd = test.as_data_frame()
# Categorize duration_months with string labels (Test Dataset)
test_pd['duration_category'] = pd.cut(test_pd['duration_months'],
                                       bins=[0, 12, 36, float('inf')],
                                       labels=['Short term', "Medium term", "Long term"])

# Map string labels to integers
duration_mapping = {"Short term": 1, "Medium term": 2, "Long term": 3}
test_pd['duration_category'] = test_pd['duration_category'].map(duration_mapping).astype(int)
```

Snippet 5.24

b) credit_amt

- Small (0-2000): This range is for relatively small loan amounts. (Represented by 1)
- Medium (2001-5000): Medium-sized loans often fall in this range. (Represented by 2)
- Large (5001 and above): Larger loan amounts exceed 5000. (Represented by 3)

```
# Categorize credit_amt with string labels (Training Dataset)
train_balanced_pd['credit_category'] = pd.cut(train_balanced_pd['credit_amt'],
                                              bins=[0, 2000, 5000, float('inf')],
                                              labels=['Small', "Medium", "Large"])

# Map string labels to integers for credit_category
credit_mapping = {"Small": 1, "Medium": 2, "Large": 3}
train_balanced_pd['credit_category'] = train_balanced_pd['credit_category'].map(credit_mapping).astype(int)

[ ] # Categorize credit_amt with string labels (Test Dataset)
test_pd['credit_category'] = pd.cut(test_pd['credit_amt'],
                                    bins=[0, 2000, 5000, float('inf')],
                                    labels=['Small', "Medium", "Large"])

# Map string labels to integers for credit_category
credit_mapping = {"Small": 1, "Medium": 2, "Large": 3}
test_pd['credit_category'] = test_pd['credit_category'].map(credit_mapping).astype(int)
```

Snippet 5.25

c) age

- 20s and lower (0-29): This category includes individuals in their twenties and younger. (Represented by 1)

- 30s (30-39): This category includes individuals in their thirties. (Represented by 2)
- 40s (40-49): This category includes individuals in their forties. (Represented by 3)
- 50s (50-59): This category includes individuals in their fifties. (Represented by 4)
- Seniors (60 and above): This category includes individuals aged 60 and above, typically considered senior citizens. (Represented by 5)

```
# Categorize age with string labels (Training Dataset)
train_balanced_pd['age_category'] = pd.cut(train_balanced_pd['age'],
                                             bins=[0, 29, 39, 49, 59, float('inf')], 
                                             labels=["20s and lower", "30s", "40s", "50s", "Seniors"])

# Map string labels to integers for age_category
age_mapping = {"20s and lower": 1, "30s": 2, "40s": 3, "50s": 4, "Seniors": 5}
train_balanced_pd['age_category'] = train_balanced_pd['age_category'].map(age_mapping).astype(int)

# Categorize age with string labels (Test Dataset)
test_pd['age_category'] = pd.cut(test_pd['age'],
                                 bins=[0, 29, 39, 49, 59, float('inf')], 
                                 labels=["20s and lower", "30s", "40s", "50s", "Seniors"])

# Map string labels to integers for age_category
age_mapping = {"20s and lower": 1, "30s": 2, "40s": 3, "50s": 4, "Seniors": 5}
test_pd['age_category'] = test_pd['age_category'].map(age_mapping).astype(int)
```

Snippet 5.26

The results from the above feature engineering are as shown below (note the new columns):

	duration_months	duration_category	credit_amt	credit_category	age	age_category
0	6	1	1169	1	67	5
1	48	3	5951	3	22	1
2	12	1	2096	2	49	3
3	42	3	7882	3	45	3
4	24	2	4870	2	53	4

	duration_months	duration_category	credit_amt	credit_category	age	age_category
0	36	2	9055	3	35	2
1	30	2	8072	3	25	1
2	24	2	3430	2	31	2
3	6	1	2647	2	44	3
4	10	1	2241	2	48	3

Snippet 5.27

Step 11: Perform Feature Selection

Several feature selection methods are used to obtain the final list.

a) Method 1: Correlation Analysis

```
import numpy as np
def correlation_analysis(df, threshold=0.8):
    corr_matrix = df.corr().abs()
    upper = corr_matrix.where(np.triu(np.ones(corr_matrix.shape), k=1).astype(bool))
    to_drop = [column for column in upper.columns if any(upper[column] > threshold)]
    return df.drop(to_drop, axis=1)
X_corr = correlation_analysis(X_train)
```

b) Method 2: ANOVA

```
anova_selector = SelectKBest(f_classif, k=10) # Select top 10 features
X_anova = anova_selector.fit_transform(X_train, y_train)
```

c) Method 3: Chi-Square

```
chi2_selector = SelectKBest(chi2, k=10) # Select top 10 features
X_chi2 = chi2_selector.fit_transform(X_train, y_train)
```

d) Method 4: Random Forest Feature Importance

```
target = 'credit_risk'
features = train_balanced_h2o_2.columns
features.remove(target)

rf_model = H2ORandomForestEstimator(ntrees=100, seed=42)
rf_model.train(x=features, y=target, training_frame=train_balanced_h2o_2)

# Get variable importance
variable_importance = rf_model.varimp(use_pandas=True)
```

e) Combining the Results

```
selected_features = {
    'correlation': X_corr.columns.tolist(),
    'anova': X_train.columns[anova_selector.get_support()].tolist(),
    'chi2': X_train.columns[chi2_selector.get_support()].tolist()
}

print("Selected features by method:")
for method, features in selected_features.items():
    print(f"\nMethod: {method}")
    print(f"Number of features selected: {len(features)}")
    print(f"Selected features: {features}")
print("\nMethod: Random Forest Variable Importance")
print("Number of features selected: ", 10)
print(variable_importance.head(10))

Selected features by method:

Method: correlation
Number of features selected: 20
Selected features: ['checking_acct_status', 'credit_history', 'purpose', 'savings_acct_bonds', 'employment_duration', 'instalment_rate_%', 'personal_status_sex', 'other_debtors_guarantors', 'age_category', 'duration_category', 'property', 'other_installment_plans', 'existing_credits']

Method: anova
Number of features selected: 10
Selected features: ['checking_acct_status', 'credit_history', 'savings_acct_bonds', 'employment_duration', 'personal_status_sex', 'other_installment_plans', 'existing_credits']

Method: chi2
Number of features selected: 10
Selected features: ['checking_acct_status', 'credit_history', 'savings_acct_bonds', 'employment_duration', 'personal_status_sex', 'other_debtors_guarantors', 'other_installment_plans', 'age_category', 'duration_category', 'property']

Method: Random Forest Variable Importance
Number of features selected: 10
      variable  relative_importance  scaled_importance  percentage
0  checking_acct_status       3748.653564        1.000000   0.199628
1     credit_history       1490.322998        0.397562   0.079365
2    savings_acct_bonds      1431.126709        0.381771   0.076212
3        purpose          1216.177612        0.324431   0.064765
4  duration_category       1081.482666        0.288499   0.057593
5  personal_status_sex      1077.821167        0.287522   0.057398
6  employment_duration      1051.159424        0.280410   0.055978
7    age_category          881.781616        0.235226   0.046958
8      property           870.117310        0.232115   0.046337
9 other_installment_plans      800.304565        0.213491   0.042619
```

f) Columns to be kept

- 'checking_acct_status' - Selected by all methods.
- 'credit_history' - Selected by all methods.
- 'purpose' - Selected by correlation & Feature Importance
- 'savings_acct_bonds' - Selected by all methods.
- 'employment_duration' - Selected by all methods.
- 'personal_status_sex' - Selected by all methods.
- 'other_debtors_guarantors' - Selected by Correlation & Chi-square
- 'other_installment_plans' - Selected by all methods.
- 'existing_credits_at_bank' - Selected by Correlation & Anova
- 'no_of_people_liable' - Selected by Correlation & Anova
- 'telephone' - Selected by Correlation, Anova & Chi-square
- 'foreign_worker' - Selected by Correlation & Chi-square
- 'duration_category' - Selected by Correlation, Anova & Chi-square
- 'age_category' - Selected by Correlation & Feature Importance

Step 12: Perform Feature Scaling

Standardization for numeric features offers a robust solution for feature scaling in credit risk analysis. By centering features around their mean and scaling them based on their standard deviation, Standardization ensures that outliers, which may exist in credit-related data, do not disproportionately influence the scaling process. This method preserves the distribution of original data, aiding in the interpretation of feature importance, and is effective across a variety of machine learning algorithms commonly employed in credit risk modeling, ensuring consistent and reliable performance.

While normalization rescales features to a predetermined range (typically [0, 1]), it's more sensitive to outliers compared to Standardization, potentially skewing the scaled features when dealing with variables prone to extreme values, such as credit amounts or durations. Additionally, normalization alters the distribution of features, which might affect the interpretation of their importance and relationships in credit risk analysis.

Therefore, Standardization is chosen. This is applied to both Training and Test sets:

```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
train_pd.iloc[:, :-1] = scaler.fit_transform(train_pd.iloc[:, :-1])
test_pd.iloc[:, :-1] = scaler.transform(test_pd.iloc[:, :-1])
```

Another important thing to do is to save the Scaler object which will be used in the deployment environment:

```
from pickle import dump
dump(scaler, open('standard_scaler.pkl', 'wb'))
```

Step 13: Perform Modelling with AutoML

a) Setting Up an AutoML experiment with H2O

```
# To initialize H2O AutoML with AUC as the sort metric
aml = H2OAutoML(max_models=15, seed=1, nfolds=5, sort_metric="AUC")
features = X_train_final.columns
label = 'credit_risk'
```

- max_models=15: Specifies the maximum number of models that AutoML will train and evaluate during its run. In this case, it will train up to 15 models.
- seed=1: Sets the random seed for reproducibility. It ensures that if the same seed and data are used, the results (such as model selection and evaluation metrics) will be the same each time.
- nfolds=5: Defines the number of folds for cross-validation during the model training process. Cross-validation helps assess the model's performance and generalization ability by splitting the data into multiple subsets (folds) for training and validation.
- sort_metric="AUC": Specifies the metric used to sort and select the best model. In this case, it sorts models based on the Area Under the ROC Curve (AUC), a common metric used to evaluate binary classification models. A higher AUC indicates better model performance in distinguishing between positive and negative classes.

b) Training of AutoML Model

aml.train(x=features, y=label, training_frame=train_final)

Model Details		Confusion Matrix (Act/Pred) for max f1 @ threshold = 0.44346873772134054							
		0	1	Error	Rate				
H2OStackedEnsembleEstimator	: Stacked Ensemble	495.0	65.0	0.1161	(85.0/580.0)				
Model Key:	StackedEnsemble_BestOfFamily_1_AutoML_1_20240728_200307	31.0	547.0	0.0538	(31.0/578.0)				
Total		526.0	612.0	0.0844	(98.0/1138.0)				
Model Summary for Stacked Ensemble:									
key	value	Maximum Metrics: Maximum metrics at their respective thresholds							
Stacking strategy	cross_validation	metric	threshold	value	idx				
Number of base models (used / total)	4/5	max f1	0.4434687	0.9163277	218.0				
# GBM base models (used / total)	1/1	max f2	0.3530528	0.9503714	240.0				
# DRF base models (used / total)	1/2	max f0point5	0.6720045	0.9337108	148.0				
# GLM base models (used / total)	1/1	max accuracy	0.4434687	0.9156415	218.0				
# DeepLearning base models (used / total)	1/1	max precision	0.9988734	1.0	0.0				
Metalearner algorithm	GLM	max recall	0.1646302	1.0	315.0				
Metalearner fold assignment scheme	Random	max specificity	0.9988734	1.0	0.0				
Metalearner nfolds	5	max absolute_mcc	0.4434687	0.8325723	218.0				
Metalearner_fold_column	None	max min_per_class_accuracy	0.5070573	0.9100346	199.0				
Custom metalearner hyperparameters	None	max mean_per_class_accuracy	0.4434687	0.9151477	218.0				
ModelMetricsBinomialGLM: stackedensemble									
** Reported on train data. **									
MSE:	0.06895509562419774	max tns	0.9988734	580.0	0.0				
RMSE:	0.2625930228018211	max fns	0.9988734	580.0	0.0				
LogLoss:	0.24341963104723094	max fps	0.0009834	580.0	399.0				
AUC:	0.9774221453287197	max tps	0.1646302	578.0	315.0				
AUCPR:	0.9786894469415999	max tnr	0.9988734	1.0	0.0				
Gini:	0.9548442906574395	max fnr	0.9988734	0.9775087	0.0				
Null degrees of freedom:	1137	max fpr	0.0009834	1.0	399.0				
Residual degrees of freedom:	1133	max tpr	0.1646302	1.0	315.0				
Null deviance:	1577.3182610640472								
Residual deviance:	554.0230802634976								
AIC:	564.0230802634976								
group	cumulative_data_fraction	lower_threshold	lift	cumulative_lift	response_rate	score	cumulative_response_rate	cumulative_score	captur
1	0.0105448	0.9984968	1.9688581	1.9688581	1.0	0.9990112		0.9990112	0.02
2	0.0202109	0.9946954	1.9688581	1.9688581	1.0	0.9974085		0.9982447	0.01
3	0.0307557	0.9773740	1.9688581	1.9688581	1.0	0.9824466		0.9928282	0.02
4	0.0413005	0.9708026	1.9688581	1.9688581	1.0	0.9733911		0.9878655	0.02
5	0.0500879	0.9662641	1.5750865	1.8997754	0.8	0.9687575		0.9649123	0.01
6	0.1001757	0.9447166	1.9343168	1.9170461	0.9824561	0.9543351		0.9736842	0.09
7	0.1502636	0.9138156	1.8652340	1.8997754	0.9473684	0.9299569		0.9649123	0.9562684
8	0.2003515	0.8856960	1.8306926	1.8825047	0.9298246	0.8991020		0.9561404	0.9419768
9	0.3005272	0.8043659	1.7097979	1.8249358	0.8684211	0.8462190		0.9269006	0.9100575
10	0.3998243	0.6877522	1.4461524	1.7308643	0.7345133	0.7565694		0.8791209	0.8719385
11	0.5	0.5292505	1.1916773	1.6228374	0.6052632	0.6085033		0.8242531	0.8191589
12	0.6001757	0.3534670	0.7599102	1.4788056	0.3859649	0.4345688		0.7510981	0.7549667
13	0.6994728	0.2096140	0.6098233	1.3554450	0.3097345	0.2774383		0.6884422	0.6871769
14	0.7996485	0.1121276	0.2936017	1.2224229	0.1491228	0.1609521		0.6208791	0.6212542
15	0.8998243	0.0541717	0.1554362	1.1036373	0.0789474	0.0824350		0.5605469	0.5612685
16	1.0	0.0003555	0.0690827		1.0	0.0350877	0.0254514	0.5079086	0.5075926

Snippets 5.28

Cross-Validation Metrics Summary:							
	mean	sd	cv_1_valid	cv_2_valid	cv_3_valid	cv_4_valid	cv
accuracy	0.8208239	0.0185707	0.8154507	0.8061674	0.8533333	0.8138528	0.8
auc	0.8935862	0.0147672	0.8890731	0.895125	0.9172725	0.8771470	0.8
err	0.1791761	0.0185707	0.1845494	0.1938326	0.1466667	0.1861472	0.1
err_count	40.8	4.494441	43.0	44.0	33.0	43.0	
f0point5	0.8132643	0.0381137	0.8051529	0.7703281	0.8733205	0.820029	0
f1	0.8306521	0.0134053	0.8230452	0.8307692	0.8465116	0.8401487	0.8
f2	0.8509023	0.0321017	0.8417509	0.9015025	0.8212996	0.8612805	0.8
lift_top_group	1.975298	0.1116761	1.9914529	2.0088496	1.9911505	1.7906977	2.0
logloss	0.4123658	0.0287646	0.4183112	0.4091567	0.3673416	0.4465987	0.4
max_per_class_error	0.2465072	0.0596304	0.2241379	0.3421053	0.1946903	0.2647059	0.2
--	--	--	--	--	--	--	--
mean_per_class_error	0.1804705	0.0196208	0.1847185	0.1931765	0.1464523	0.1943684	0.1
mse	0.1313975	0.0083517	0.1333277	0.1311204	0.1173822	0.1386643	0.1
null_deviance	315.8413	6.028866	323.05124	314.8293	311.957	320.7926	30
pr_auc	0.8972381	0.0164780	0.8901876	0.8858590	0.9253811	0.8981997	0.8
precision	0.803051	0.0569124	0.7936508	0.7346939	0.8921568	0.8071428	0.7
r2	0.4726456	0.0353542	0.4666793	0.4755083	0.5304621	0.4376602	0.4
recall	0.8662709	0.0562610	0.8547009	0.9557522	0.8053097	0.875969	0.8
residual_deviance	187.79787	15.047662	194.933	185.75711	165.30374	206.32863	18
rmse	0.3623366	0.0117092	0.3651407	0.3621055	0.3426108	0.3723766	0.3
specificity	0.772788	0.0889787	0.7758621	0.6578947	0.9017857	0.7352941	0.7

Snippet 5.29**c) Generation of Leaderboard of Models**

```
model_leaderboard = aml.leaderboard
model_leaderboard.head(rows=model_leaderboard.nrows)
```

model_id	auc	logloss	aucpr
StackedEnsemble_BestOfFamily_1_AutoML_1_20240728_200307	0.893375	0.412627	0.897242
StackedEnsemble_AllModels_1_AutoML_1_20240728_200307	0.89327	0.412792	0.897374
GBM_3_AutoML_1_20240728_200307	0.892483	0.415773	0.895395
GBM_grid_1_AutoML_1_20240728_200307_model_1	0.890277	0.420063	0.897345
GBM_5_AutoML_1_20240728_200307	0.890172	0.422298	0.896696
GBM_4_AutoML_1_20240728_200307	0.889658	0.420305	0.897309
GBM_2_AutoML_1_20240728_200307	0.888676	0.422926	0.888446
GBM_grid_1_AutoML_1_20240728_200307_model_2	0.885637	0.42915	0.885156
GBM_grid_1_AutoML_1_20240728_200307_model_3	0.884698	0.428425	0.884414
XRT_1_AutoML_1_20240728_200307	0.883047	0.435332	0.88752
DRF_1_AutoML_1_20240728_200307	0.882056	0.465358	0.884235
GLM_1_AutoML_1_20240728_200307	0.868844	0.455396	0.8604
GBM_1_AutoML_1_20240728_200307	0.865296	0.46097	0.852797
DeepLearning_1_AutoML_1_20240728_200307	0.865046	0.494023	0.855762
DeepLearning_grid_1_AutoML_1_20240728_200307_model_1	0.858899	0.537059	0.847994
DeepLearning_grid_2_AutoML_1_20240728_200307_model_1	0.847962	0.592429	0.836292
DeepLearning_grid_3_AutoML_1_20240728_200307_model_1	0.844734	0.601547	0.826159

[17 rows x 7 columns]

Snippet 5.30

Although I set the maximum number of models trained by AutoML to be 15, there are 17 here because H2O AutoML will train Stacked Ensemble Models by default. The models are sorted

based on the AUC metric, which is the default metric to evaluate binary classification models.
Select the best model:

```
best_single_model = h2o.get_model("GBM_3_AutoML_1_20240728_200307_model_1")
```

Although the Stacked Ensemble Models have better performance metrics, interpreting the overall behavior of stacked ensemble models can indeed be challenging due to their complexity as they combine multiple models. Since the GBM model has approximately the same performance as the them, it is selected as the best model.

d) Display Best Model Details

The following code displays the Model Type, Model ID, Hyperparameters, Performance Metrics, Training Summary, Variable Importance and Model Parameters based on the training and validation data:

```
print(best_single_model)
```

<pre>Model Details ===== H2OGradientBoostingEstimator : Gradient Boosting Machine Model Key: GBM_grid_1_AutoML_1_20240728_200307_model_1 Model Summary: number_of_trees number_of_internal_trees model_size_in_bytes min_depth max_depth mean_depth min_leaves max_leaves mean_leaves -- ----- -- ----- 57 57 27320 7 14 9.21053 15 44 33.3158 ModelMetricsBinomial: gbm ** Reported on train data. ** MSE: 0.09194964444202466 RMSE: 0.3032319977212574 LogLoss: 0.31012385520231217 Mean Per-Class Error: 0.12455820563519525 AUC: 0.95457859614434 AUCPR: 0.9578902667892383 Gini: 0.909151922886801 Confusion Matrix (Act/Pred) for max f1 @ threshold = 0.43924555089773043 0 1 Error Rate -- -- 0 467 93 0.1661 (93.0/560.0) 1 48 530 0.083 (48.0/578.0) Total 515 623 0.1239 (141.0/1138.0)</pre>

metric	threshold	value	idx
max f1	0.439246	0.882598	226
max f2	0.263808	0.923754	282
max f0point5	0.716536	0.900881	129
max accuracy	0.535711	0.880492	194
max precision	0.98678	1	0
max recall	0.083427	1	359
max specificity	0.98678	1	0
max absolute_mcc	0.535711	0.761528	194
max min_per_class_accuracy	0.517959	0.877163	200
max mean_per_class_accuracy	0.535711	0.88074	194
max tns	0.98678	560	0
max fns	0.98678	577	0
max fps	0.0106629	560	399
max tps	0.083427	578	359
max tnr	0.98678	1	0
max fnr	0.98678	0.99827	0
max fpr	0.0106629	1	399
max tpr	0.083427	1	359

```
ModelMetricsBinomial: gbm
** Reported on cross-validation data. **

MSE: 0.13481022202702428
RMSE: 0.3671651154821551
LogLoss: 0.4200626148346719
Mean Per-Class Error: 0.18775951557093426
AUC: 0.8902774345032131
AUCPR: 0.8973450271831928
Gini: 0.7805548690064261

Confusion Matrix (Act/Pred) for max f1 @ threshold = 0.47563923184382406
      0   1   Error Rate
----- -----
0   434 126 0.225 (126.0/560.0)
1   87 491 0.1505 (87.0/578.0)
Total 521 617 0.1872 (213.0/1138.0)

Maximum Metrics: Maximum metrics at their respective thresholds
metric      threshold      value      idx
----- -----
max f1      0.475639 0.821757 209
max f2      0.158194 0.881616 319
max f0point5 0.58859 0.825315 170
max accuracy 0.480088 0.81283 207
max precision 0.987937 1 0
max recall 0.0503202 1 377
max specificity 0.987937 1 0
max absolute_mcc 0.475639 0.626637 209
max min_per_class_accuracy 0.539309 0.807958 188
max mean_per_class_accuracy 0.480088 0.812296 207
max tns 0.987937 560 0
max fns 0.987937 577 0
max fps 0.00570327 560 399
max tps 0.0503202 578 377
max tnr 0.987937 1 0
max fnr 0.987937 0.99827 0
max fpr 0.00570327 1 399
max tpr 0.0503202 1 377
```

Cross-Validation Metrics Summary:							
	mean	sd	cv_1_valid	cv_2_valid	cv_3_valid	cv_4_valid	cv_5_valid
accuracy	0.826872	0.0312555	0.798246	0.828947	0.872807	0.797357	0.837004
auc	0.88969	0.0274208	0.856235	0.907482	0.922893	0.86836	0.893478
err	0.173128	0.0312555	0.201754	0.171053	0.127193	0.202643	0.162996
err_count	39.4	7.09225	46	39	29	46	37
f0point5	0.823615	0.0381444	0.787293	0.807927	0.879218	0.797774	0.845865
f1	0.833494	0.0301	0.832117	0.844622	0.872247	0.788991	0.829493
f2	0.845338	0.0462199	0.882353	0.884808	0.865385	0.780399	0.813743
lift_top_group	1.97173	0.0810209	1.83871	1.96552	1.98261	2.04505	2.02679
logloss	0.422215	0.0439962	0.472333	0.394078	0.365915	0.457822	0.420928
max_per_class_error	0.233173	0.0769502	0.346154	0.258929	0.13913	0.225225	0.196429
mcc	0.656681	0.061761	0.601785	0.666164	0.745911	0.594572	0.674973
mean_per_class_accuracy	0.824077	0.034267	0.786601	0.827432	0.872913	0.79687	0.836568
mean_per_class_error	0.175923	0.034267	0.213399	0.172568	0.127087	0.20313	0.163432
mse	0.135716	0.0158293	0.15183	0.126856	0.114183	0.149874	0.135839
pr_auc	0.898106	0.0215121	0.885424	0.915208	0.925888	0.874675	0.889336
precision	0.817999	0.0512907	0.76	0.785185	0.883929	0.803738	0.857143
r2	0.456077	0.0645676	0.387972	0.492418	0.543231	0.400213	0.456551
recall	0.854473	0.0646379	0.919355	0.913793	0.86087	0.774775	0.803571
rmse	0.367886	0.0216832	0.389653	0.356169	0.33791	0.387136	0.368563
specificity	0.793681	0.0962423	0.653846	0.741071	0.884956	0.818966	0.869565

Variable Importances:			
variable	relative_importance	scaled_importance	percentage
checking_acct_status	224.705	1	0.279211
purpose	87.9581	0.391438	0.109294
savings_acct_bonds	74.4719	0.331421	0.0925363
personal_status_sex	73.1409	0.325498	0.0908825
credit_history	66.0506	0.293944	0.0820723
duration_category	60.6836	0.270059	0.0754034
employment_duration	49.1425	0.218698	0.0610629
age_category	44.6155	0.198551	0.0554377
other_installment_plans	42.5322	0.18928	0.0528491
telephone	26.3486	0.117259	0.0327399
existing_credits_at_bank	20.2365	0.0900582	0.0251452
other_debtors_guarantors	17.8774	0.0795595	0.0222139
no_of_people_liable	10.0447	0.0447016	0.0124812
foreign_worker	6.97798	0.031054	0.00867061

Snippets 5.31

Step 14: Making Predictions using the Best Model

The following code generates the predictions for the model based on the testing dataset:

```
predictions = best_single_model.predict(test_final)
```

For binary models, labels are derived based on the maximum F1 threshold from the model. For classification tasks, the threshold for predictions is chosen as follows:

- If I only use training data, the threshold comes from the train data model
- If I train the model with training and validation data, the threshold comes from the validation data model.
- If I train the model with training data and set a parameter called nfolds, the threshold comes from the training data model.
- If I train the model with the training and validation data and also set a parameter called nfolds, the threshold from the validation data model is used

Since I trained the model with training data and set the nfolds parameter, the maximum F1 threshold is coming from the training data model, which is 0.433389. If the value under the p1 column is greater than the max F1 threshold value, then the predicted class would indeed be class 1. Conversely, if the value is less than or equal to the threshold, the predicted class would be 0.

The code below computes the GBM model's performance on the test dataset.

```
performance = best_single_model.model_performance(test_data=test_final)
print(performance)
```

For binary classification problems like this one, H2O uses the model and the given dataset to compute the threshold that will give the maximum F1 for the given dataset.

```
ModelMetricsBinomial: gbm
** Reported on test data. **

MSE: 0.19730799569335994
RMSE: 0.4441936466152571
LogLoss: 0.6012578731779863
Mean Per-Class Error: 0.2761904761904762
AUC: 0.7492857142857143
AUCPR: 0.47771329750656978
Gini: 0.49857142857142867

Confusion Matrix (Act/Pred) for max f1 @ threshold = 0.34611405987529975
      0   1   Error   Rate
-----+-----+-----+
0    93   47   0.3357  (47.0/140.0)
1    13   47   0.2167  (13.0/60.0)
Total 106   94   0.3     (60.0/200.0)

Maximum Metrics: Maximum metrics at their respective thresholds
metric      threshold      value      idx
-----+-----+-----+-----+
max f1        0.346114  0.61039   92
max f2        0.125701  0.74026   143
max f0point5  0.619745  0.568182  60
max accuracy   0.718417  0.745     45
max precision  0.776423  0.6       33
max recall    0.0226026  1         195
max specificity 0.969393  0.992857  0
max absolute_mcc 0.346114  0.41099   92
max min_per_class_accuracy 0.424126  0.7     78
max mean_per_class_accuracy 0.346114  0.72381   92
max tns       0.969393  139      0
max fns       0.969393  60       0
max fps       0.0171075  140      197
max tps       0.0226026  60       195
max tnr       0.969393  0.992857  0
max fnr       0.969393  1         0
max fpr       0.0171075  1         197
max tpr       0.0226026  1         195
```

Snippets 5.32

1. Accuracy: This shows the number of correct predictions made as a ratio of all predictions made. In H2O, two values will be displayed. The left value is the threshold that provides the best balance between precision and recall. The right value shows the accuracy of the model.
2. Logloss: It evaluates how close the model's predicted value is to the actual target value. The lower the logloss value, the better the model's performance.

3. AUC: This evaluate how well the model is able to distinguish between true positives and false positives. An AUC of 1 indicates a perfect classifier whereas an AUC of 0.5 indicates a poor classifier.
4. AUCPR: This evaluates how well the model is able to distinguish between precision recall pairs or points. Higher values indicating better model performance. It provides insights into the model's ability to correctly identify positive instances and the reliability of those positive predictions.
5. F1: It measures how well a binary classifier can classify positive cases (given a threshold value). The left value represents the threshold value at which the model's predicted probabilities are converted into binary class labels. The right value represents the F1 score corresponding to the given threshold

Step 15: Explain How the Model makes Predictions

This single line of code displays a collection of plots and tables that summarise the model's characteristics and performance:

```
best_single_model.explain(test_final)
```

1. Feature Importance Plot

They show the relative importance of each feature in making predictions. This helps identify which features the model relies on the most.

2. Partial Dependence Plots

They show the relationship between a feature and the predicted outcome, averaged over the dataset. They help in understanding how the model's predictions change with changes in individual features.

3. SHAP Summary Plot

It shows an overview of feature importance and their contributions to individual predictions. Each point represents a SHAP value for a feature and instance in the dataset, providing insights into feature interactions and non-linear effects.

4. Model Metrics Table

It displays various performance metrics, providing a quick summary of the model's overall performance.

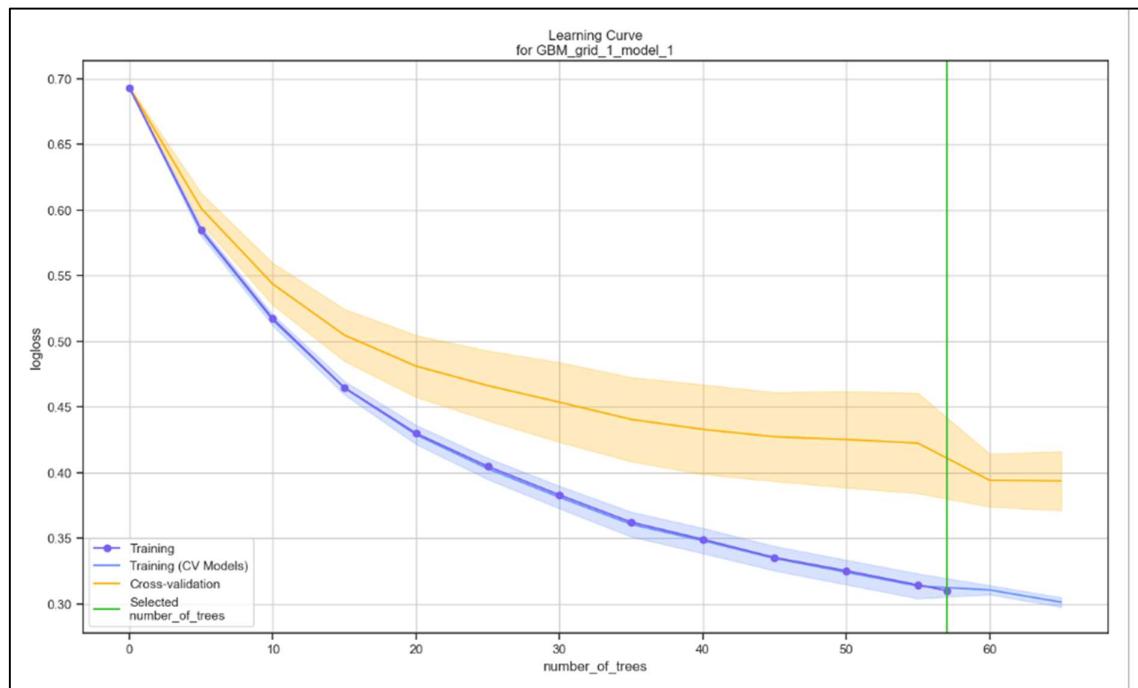
5. Residual Analysis

It shows the differences between actual and predicted values (residuals) to identify patterns that might suggest issues like heteroscedasticity or model bias.

6. Confusion Matrix

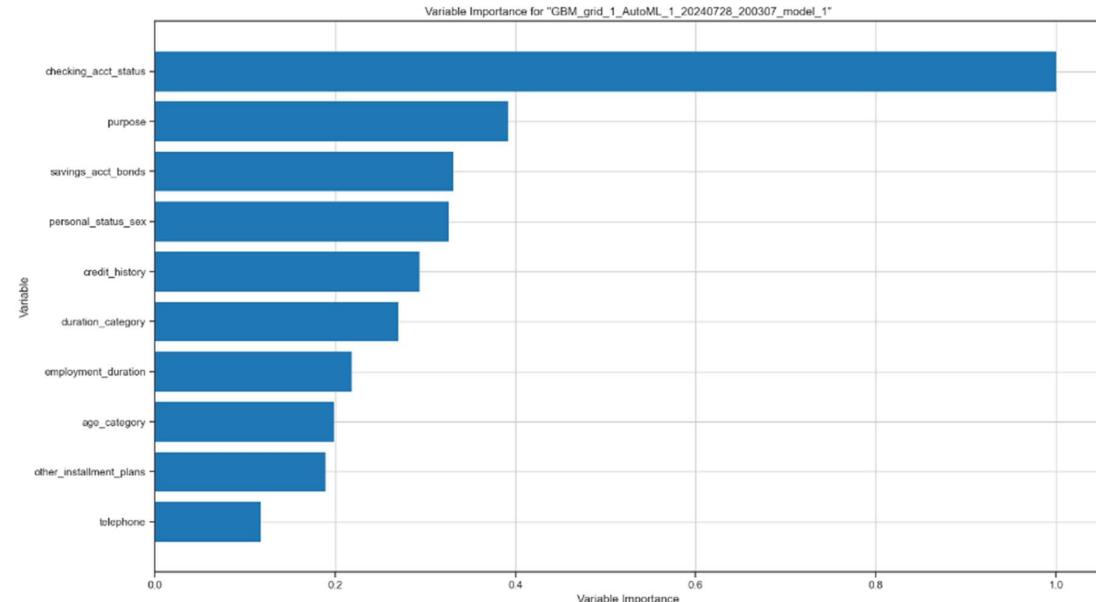
It shows the counts of true positives, true negatives, false positives, and false negatives, providing a detailed breakdown of model predictions.

Confusion Matrix				
Confusion matrix shows a predicted class vs an actual class.				
GBM_grid_1_AutoML_1_20240728_200307_model_1				
Confusion Matrix (Act/Pred) for max f1 @ threshold = 0.34611405987529975				
	0	1	Error	Rate
0	93.0	47.0	0.3357	(47.0/140.0)
1	13.0	47.0	0.2167	(13.0/60.0)
Total	106.0	94.0	0.3	(60.0/200.0)



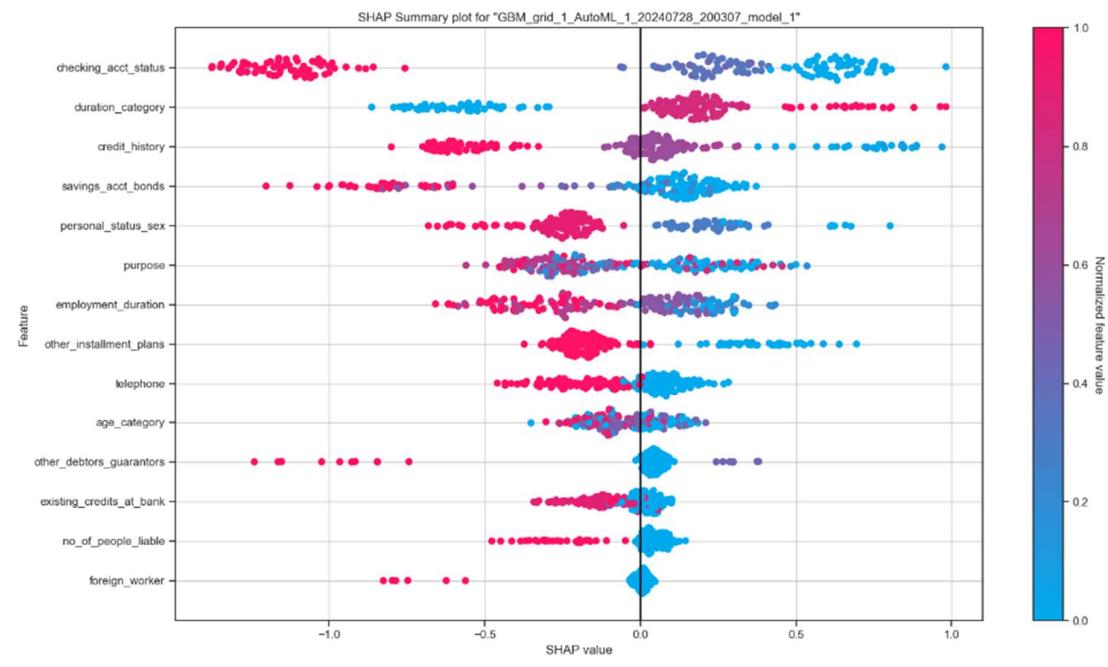
Variable Importance

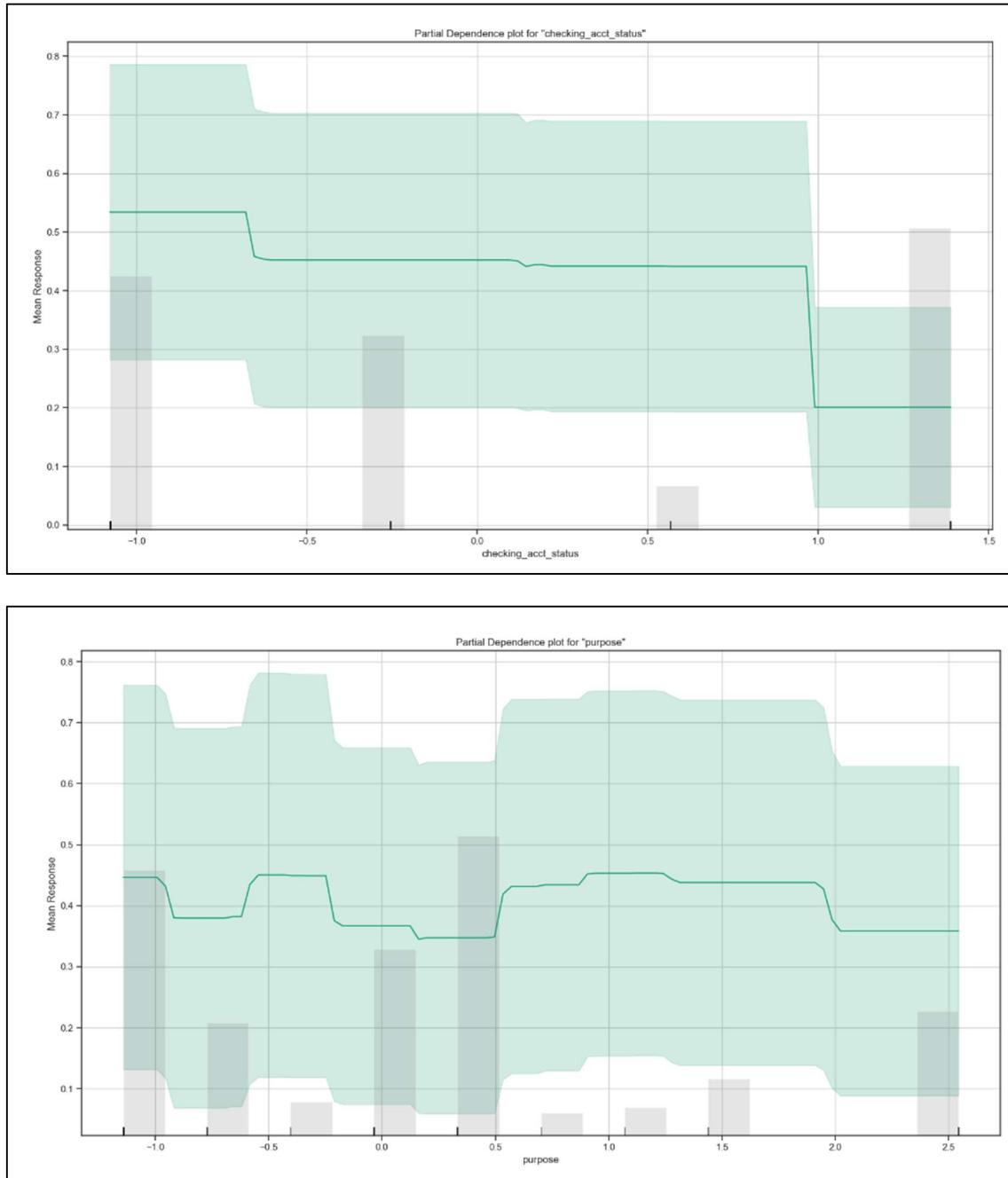
The variable importance plot shows the relative importance of the most important variables in the model.

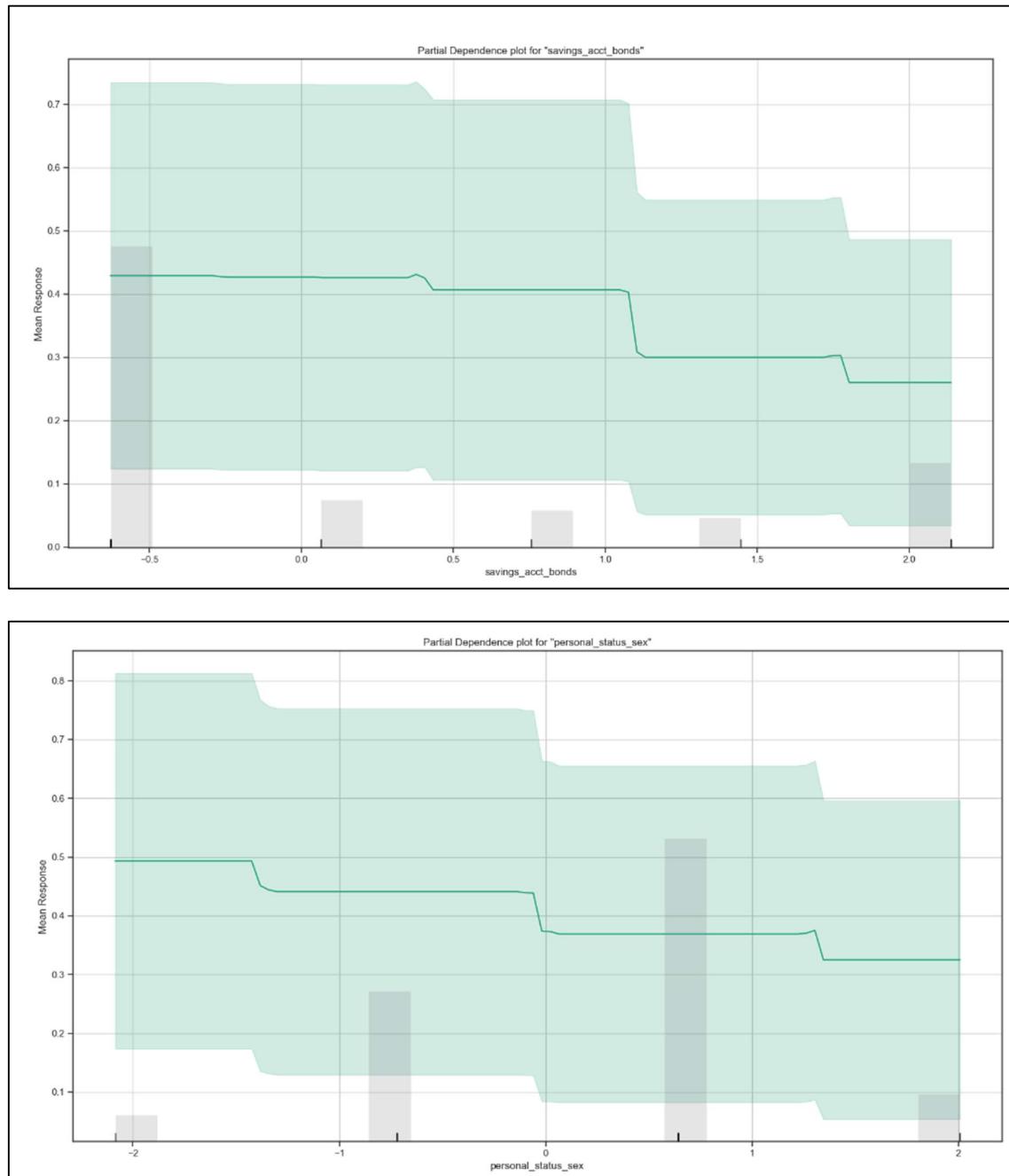


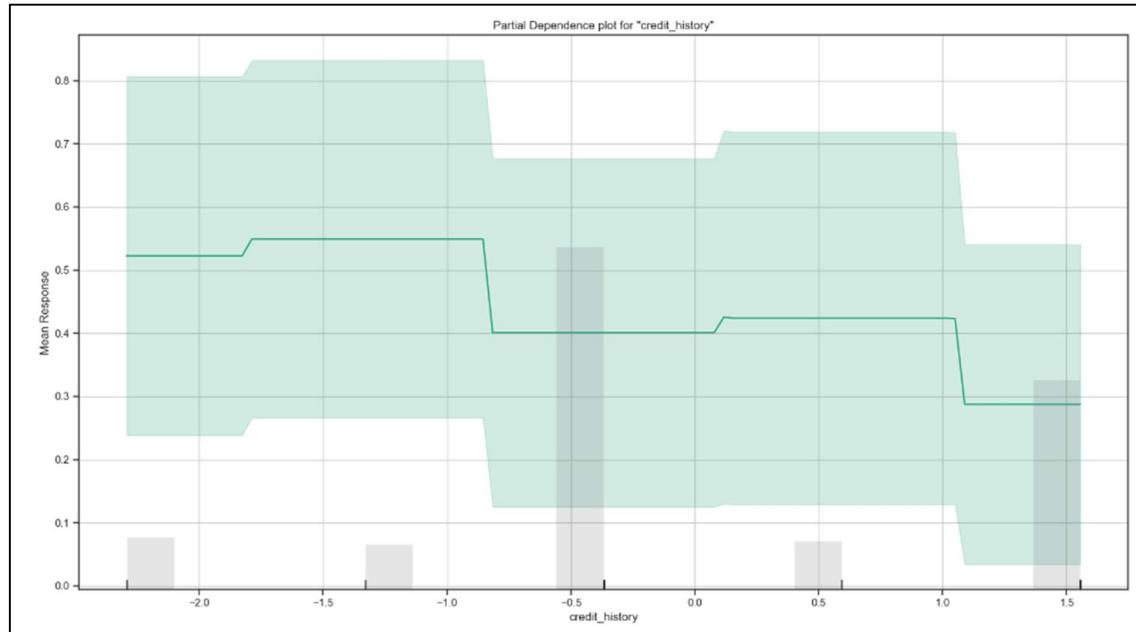
SHAP Summary

SHAP summary plot shows the contribution of the features for each instance (row of data). The sum of the feature contributions and the bias term is equal to the raw prediction of the model, i.e., prediction before applying inverse link function.









Snippets 5.33

Step 16: Prepare Model for Deployment

1. Download the Model as MOJO

```
file_path = "Best_AutoML_model3.zip"
best_single_model.download_mojo(file_path)
```

2. Import the Model

```
imported_model = h2o.import_mojo(file_path)
```

3. Test Prediction by MOJO

```
In [156]: # Ensure that the columns match those used for training
single_row = h2o.H2OFrame({'checking_acct_status': [-1.07781], 'credit_history': [1.55589], 'purpose': [0.333162], 'savings_a
# Make predictions using the MOJO model
single_prediction = imported_model.predict(single_row)

# Print the single prediction
print(single_prediction)
Parse progress: |██████████| (done) 100%
generic prediction progress: |
predict      p0      p1
 0  0.904454  0.0955457
[1 row x 3 columns]

In [157]: # Ensure that the columns match those used for training
single_row = h2o.H2OFrame({'checking_acct_status': [-0.255727], 'credit_history': [-0.368678], 'purpose': [0.333162], 'saving
# Make predictions using the MOJO model
single_prediction = imported_model.predict(single_row)

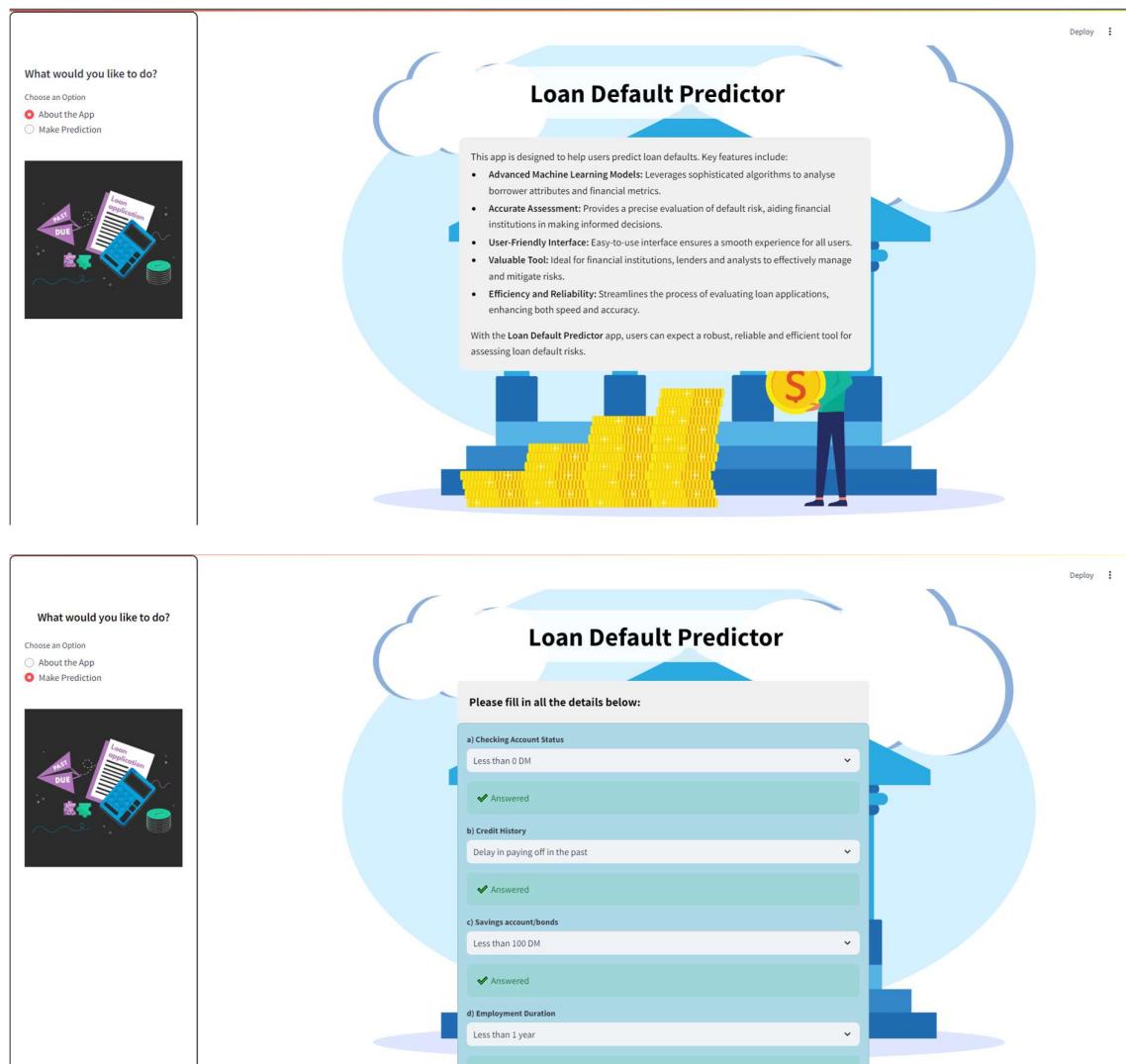
# Print the single prediction
print(single_prediction)
Parse progress: |██████████| (done) 100%
generic prediction progress: |
predict      p0      p1
 1  0.198746  0.801254
[1 row x 3 columns]
```

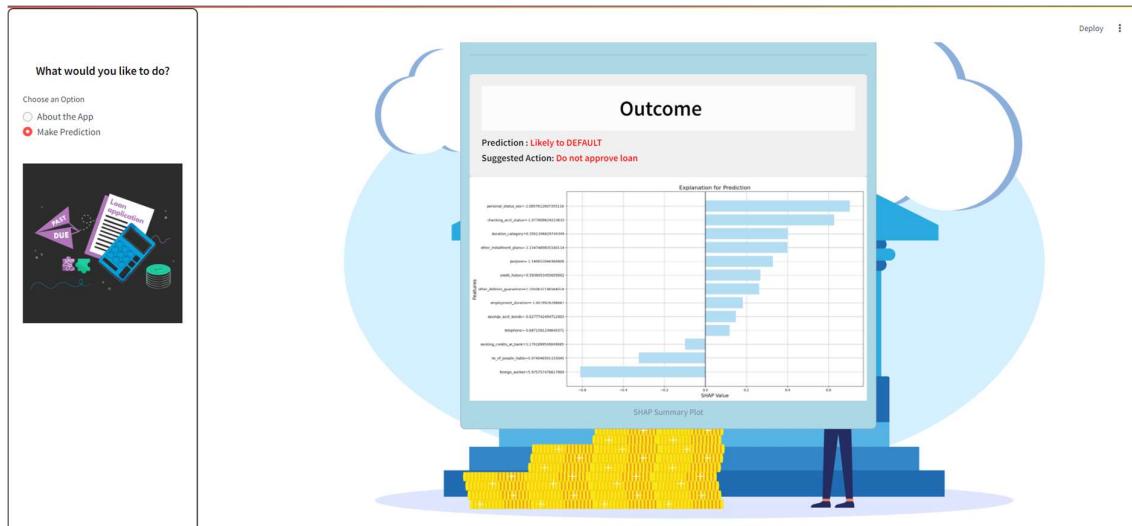
5.1.2 User Interface & API Integration Implementation

There are basically 2 main components: (1) the User Interface and (2) the API which interacts with the Model. This has been described in chapters 4 (Design) and 6 (System Deployment). Although Streamlit can directly load, train, and display the results of ML models within the UI, we want to decouple the frontend (UI) logic from the backend (data preprocessing, ML model execution) logic to achieve a more modular design. FastAPI was selected to host the backend logic and Streamlit only the frontend UI logic.

a) Actual User Interface

The screenshot below is the Streamlit user interface. After the user submits the form fields, it is passed to the API which does some data preprocessing and invokes the Model object. The prediction (Default or Not Default) is returned to the user along with the SHAP explanations.





Snippets 5.34

Below I describe the algorithms of both components. To view the code, see the source files “uiAutoML.py” and “fastapiAutoMLv2.py”.

b) FastAPI Backend Algorithm

1. Initialize FastAPI Application
2. Create a FastAPI app instance.
3. Initialize H2O
4. Start the H2O environment using `h2o.init()`.
5. Load the Loan Default Prediction Model (MOJO)
6. Import the MOJO model from a zip file (`Best_AutoML_model4.zip`).
7. Load the Standard Scaler
8. Load the StandardScaler object from a file (`standard_scaler.pkl`).
9. Define Input Schema
10. Define a `LoanApplication` class to represent the structure of the input data for loan applications, with attributes like `checking_acct_status`, `credit_history`, `purpose`, etc.
11. Define Label Mappings
12. Create a dictionary `label_mappings` that holds encoding mappings for categorical variables.
13. Define Helper Functions
14. Create a function `apply_label_encoding(column, mapping)` to apply label encoding based on the provided mapping.

15. Predict Endpoint (/predict)
16. Define a POST endpoint /predict that accepts a LoanApplication request.
17. Extract loan application data and convert it into a pandas DataFrame.
18. Create preprocessing steps to apply label encoding for each categorical column.
19. Apply these transformations using ColumnTransformer.
20. Preprocess the input DataFrame.
21. Scale the preprocessed data using the StandardScaler object.
22. Create an H2O frame with scaled values.
23. Pass the H2O frame to the MOJO model for prediction.
24. Extract the predicted results.
25. Return the predictions in the API response.
26. SHAP Explanation Endpoint (/show-shap)
27. Define a POST endpoint /show-shap that generates and returns a SHAP (SHapley Additive exPlanations) plot for the model's prediction.
28. Generate a SHAP explanation plot for the user's input data.
29. Save the plot to a buffer as a PNG image.
30. Return the image as a response to the API request.

c) Streamlit User Interface Algorithm

1. Import necessary libraries:
 - streamlit as st
 - requests for handling HTTP requests
 - Image from PIL for handling image data
 - BytesIO for image byte stream handling
2. Set the title of the application as "Loan Default Predictor".
3. Define the main function:
 - a. Set the background and design elements of the app using CSS.
 - b. Add a sidebar with options to navigate between "About the App" and "Make Prediction".

- c. Depending on user selection, display either introduction page or loan prediction page.
4. Define the `introduce()` function:
- a. Apply custom CSS for styling the introduction content.
 - b. Display information about the app, its features, and its benefits for users.
5. Define the `check_response()` function:
- a. Check if all user inputs (answers) are filled.
 - b. Return False if any answer is missing, otherwise return True.
6. Define the `update_response(index)` function:
- a. Check if the user has filled the required input for a specific question.
 - b. Display appropriate success or error messages depending on input validation.
7. Define the `get_prediction(data)` function:
- a. Send the user-provided data to an external prediction API.
 - b. Return the response and status code from the API.
8. Define the `get_shap()` function:
- a. Make a request to fetch and display the SHAP summary plot from the backend.
 - b. If the request is successful, display the image; otherwise, show an error message.
9. Define the `get_info_page()` function:
- a. Initialize session variables to store user inputs, predictions, and validation statuses.
 - b. Create a form with multiple questions (related to loan information and borrower details).
 - Questions use dropdowns and input fields for features such as account status, credit history, employment duration, etc.
 - Update responses for each question and validate them using `update_response()`.
 - c. Include a checkbox to confirm data entry.
 - d. Add a submit button to trigger the prediction process.
 - e. Upon form submission:
 - Validate that all questions are answered.
 - If valid, make a prediction request using `get_prediction()` and display the results.
 - Based on the prediction, show whether the loan is likely to default or not.

5.1.3 System Integration Testing

ID	Test Objective	Test Data	Precondition	Steps	Expected Result	Actual Result	Status
1	View the Application	None	The Frontend and Backend servers must be running.	<ol style="list-style-type: none"> 1. Open conda prompt & navigate to the directory containing streamlit script using cd 2. Enter the command “streamlit run <filename>.py” on conda prompt 3. Wait for a message to be displayed on conda prompt 4. View Streamlit app on the browser 5. Open another conda prompt & navigate to the directory containing fastapi script using cd 6. Enter the command “uvicorn <filename>:app – reload” 7. Once there is a message saying the server has started, you can make predictions on the application. 	<p>A message is displayed on the conda prompt stating that the Streamlit app can be viewed on the browser.</p> <p>The app can be viewed on the browser.</p> <p>A message is displayed on the conda prompt stating that the model has been built and that the FastAPI server has started.</p>	<p>A message is displayed on the conda prompt stating that the Streamlit app can be viewed on the browser.</p> <p>The app can be viewed on the browser.</p> <p>A message is displayed on the conda prompt stating that the model has been built and that the FastAPI server has started.</p>	Pass

2	To navigate to the “Make Prediction” page	None	Must be on the “About the App” page	1. Click on the “Make Prediction” radio button on the sidebar	The user is directed to the “Make Prediction” page	The user is directed to the “Make Prediction” page	Pass
3	To navigate to the “About the App” page	None	Must be on the “Make Prediction” page	1. Click on the “About the App” radio button	The user is directed to the “About the App” page	The user is directed to the “About the App” page	Pass
4	To check whether the form will be submitted if no value is entered but the check box is ticked	None	Must be on the “Make Prediction” page and the input fields are blank	1. Scroll to the bottom of the page and tick the check box 2. Click the ‘Predict’ button	The page displays an error message asking the user to answer all the input fields and an error message at each input field.	The page displays an error message asking the user to answer all the input fields and an error message at each input field.	Pass
5	To check whether the form will be submitted if all input fields are entered but the check box is not ticked	1. Checking Account Status = Less than 0 DM 2. Credit History = Delay in paying off in the past 3. Savings account/bonds = Less than 100 DM 4. Employment Duration = Unemployed 5. Personal Status and Sex = Male: single 6. Other Debtors/Guarantors = None 7. Other Instalment Plans = Bank 8. Number of People being Liable to Provide Maintenance for = 5.00 9. Has Telephone? = None 10. Duration = Long term	All input fields must be answered but the check box is not ticked. Must be on the “Make Prediction” page	1. Scroll to the bottom of the page and click the ‘Predict’ button	The page displays an error message asking the user to confirm the details before submitting the form.	The page displays an error message asking the user to confirm the details before submitting the form.	Pass

		11. Purpose = Car (new) 12. Age = 20s and lower 13. Is Foreign Worker? = Yes 14. Existing Credits at Bank = 5.00					
6	To predict that a defaulter will default	1. Checking Account Status = Less than 0 DM 2. Credit History = Delay in paying off in the past 3. Savings account/bonds = Less than 100 DM 4. Employment Duration = Unemployed 5. Personal Status and Sex = Male: single 6. Other Debtors/Guarantors = None 7. Other Instalment Plans = Bank 8. Number of People being Liable to Provide Maintenance for = 5.00 9. Has Telephone? = None 10. Duration = Long term 11. Purpose = Car (new) 12. Age = 20s and lower 13. Is Foreign Worker? = Yes 14. Existing Credits at Bank = 5.00	Must be on the “Make Prediction” page	1. Enter the test data into the input fields 2. Scroll to the bottom of the page and tick the check box. 3. Click the ‘Predict’ button	The page’s response is that the person is likely to default and the suggested action is not to approve the loan. A SHAP summary plot will be displayed after the prediction.	The page’s response is that the person is likely to default and the suggested action is not to approve the loan. A SHAP summary plot will be displayed after the prediction.	Pass

7	To predict that a defaulter will default	1. Checking Account Status = No checking account 2. Credit History = Delay in paying off in the past 3. Savings account/bonds = Less than 100 DM 4. Employment Duration = Less than 1 year 5. Personal Status and Sex = Female: divorced/separated/married 6. Other Debtors/Guarantors = None 7. Other Instalment Plans = Bank 8. Number of People being Liable to Provide Maintenance for = 1.00 9. Has Telephone? = Yes 10. Duration = Long term 11. Purpose = Car (new) 12. Age = 30s 13. Is Foreign Worker? = No 14. Existing Credits at Bank = 7.00	Must be on the “Make Prediction” page	1. Enter the test data into the input fields 2. Scroll to the bottom of the page and tick the check box. 3. Click the ‘Predict’ button	The page’s response is that the person is likely to default and the suggested action is not to approve the loan. A SHAP summary plot will be displayed after the prediction.	The page’s response is that the person is likely to default and the suggested action is not to approve the loan. A SHAP summary plot will be displayed after the prediction.	Pass
8	To predict that a non-defaulter will not default	1. Checking Account Status = Greater than or equal to 200 DM / salary assignments for at least 1 year 2. Credit History = No credits taken/all	Must be on the “Make Prediction” page	1. Enter the test data into the input fields 2. Scroll to the bottom of the page and tick the check box.	The page’s response is that the person is not likely to default and the suggested action is to approve the loan. A SHAP summary plot will be displayed after the prediction.	The page’s response is that the person is not likely to default and the suggested action is to approve the loan. A SHAP summary plot will be displayed after the prediction.	Pass

	<p>credits paid back duly</p> <p>3. Savings account/bonds = Greater than or equal to 1000 DM</p> <p>4. Employment Duration = Greater than or equal to 7 years</p> <p>5. Personal Status and Sex = Male: married/widowed</p> <p>6. Other Debtors/Guarantors = None</p> <p>7. Other Instalment Plans = Bank</p> <p>8. Number of People being Liable to Provide Maintenance for = 3.00</p> <p>9. Has Telephone? = Yes</p> <p>10. Duration = Short term</p> <p>11. Purpose = Business</p> <p>12. Age = 30s</p> <p>13. Is Foreign Worker? = No</p> <p>14. Existing Credits at Bank = 1.00</p>		3. Click the ‘Predict’ button	summary plot will be displayed after the prediction.	summary plot will be displayed after the prediction.		
9	To predict that a non-defaulter will not default	<p>1. Checking Account Status = Greater than or equal to 200 DM / salary assignments for at least 1 year</p> <p>2. Credit History = No credits taken/all credits paid back duly</p> <p>3. Savings account/bonds =</p>	Must be on the “Make Prediction” page	<p>1. Enter the test data into the input fields</p> <p>2. Scroll to the bottom of the page and tick the check box.</p> <p>3. Click the ‘Predict’ button</p>	The page’s response is that the person is not likely to default and the suggested action is to approve the loan. A SHAP summary plot will be displayed after the prediction.	The page’s response is that the person is not likely to default and the suggested action is to approve the loan. A SHAP summary plot will be displayed after the prediction.	Pass

	<p>Greater than or equal to 1000 DM</p> <p>4. Employment Duration = Greater than or equal to 7 years</p> <p>5. Personal Status and Sex = Female: divorced/separated/married</p> <p>6. Other Debtors/Guarantors = None</p> <p>7. Other Instalment Plans = None</p> <p>8. Number of People being Liable to Provide Maintenance for = 3.00</p> <p>9. Has Telephone? = Yes</p> <p>10. Duration = Short term</p> <p>11. Purpose = Furniture/equipment</p> <p>12. Age = 30s</p> <p>13. Is Foreign Worker? = No</p> <p>14. Existing Credits at Bank = 1.00</p>				
--	---	--	--	--	--

Table 5.1. Test Cases Conducted for AutoML Approach

5.2 Generative AI Approach

Since this is the first time I am using CDA to develop a ML model, I wanted to experiment with 2 different strategies. First, I started with “minimal guidance” where I do not provide too much instructions to CDA. The reason for this is to assess how powerful and independent CDA is. In other words, I wanted to see how comprehensive CDA’s generated code is and what its quality is like, without giving CDA much guidance. Later, if it turned out that the results are not so good, I will adopt a more guided approach by giving explicit instructions.

NOTE: In the sections that follow, the prompts that I issue to CDA will be in bold. Responses from CDA will be unbolted and italicized. Generated code will be in Courier New font and font size will be smaller.

5.2.1 Machine Learning Model Implementation

Let's start with the minimal guidance approach. This is the prompt I entered:

For the file I just uploaded, I want you to create a high-performing binary classification model (after evaluating multiple models) in Python for predicting loan default. After you have selected the best model, it should be deployable to a production environment (using Streamlit for frontend and FastApi as backend). I am not giving you too many instructions because I want you to use your own expertise in machine learning best practices and domain knowledge to accomplish the task. However, I require you to interpret the results and justify the decisions that you take in the construction and selection of the models

As you can see from my prompt message above, I only informed CDA what type of ML classifier is to be built and mentioned the context/domain which is Loan Default. However, I was quite specific in telling CDA I want this loan default predictor app to be implemented using the Python programming language and the frameworks Streamlit and FastAPI (which will be used in deployment). Other than that, I wanted CDA to use “its own brains” (so-to-speak) to plan, design, and generate the appropriate code. Note that I also said I wanted a “high-performing” model. This is what CDA said it planned to do:

CDA RESPONSE:

Step 1: Data Preprocessing

First, we need to load and preprocess the data. Based on the given data structure, it seems to follow a specific encoding format which likely needs to be decoded for better understanding and processing.

Step 2: Exploratory Data Analysis (EDA)

Next, we will perform EDA to understand the distribution of the data and to check for any missing values, outliers, or imbalanced classes.

Step 3: Model Selection and Training

We'll evaluate several binary classification models, such as Logistic Regression, Random Forest, Gradient Boosting, and XGBoost. We'll use cross-validation to find the best model and optimize hyperparameters.

Step 4: Model Evaluation

After training, we will evaluate the models using metrics such as accuracy, precision, recall, F1 score, and ROC-AUC. We will select the best-performing model based on these metrics.

Step 5: Model Deployment

We will prepare the selected model for deployment using Streamlit for the frontend and FastAPI for the backend.

Here is the step-by-step Python code for this process:

```
import pandas as pd

from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.preprocessing import LabelEncoder, StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassifier
from xgboost import XGBClassifier

from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score,
roc_auc_score
import joblib

# Load the dataset
data = pd.read_csv('/mnt/data/german.txt', sep=' ')

# Adding column names (based on the common understanding of the dataset)
columns = ['Status_checking', 'Duration', 'Credit_history', 'Purpose', 'Credit_amount',
'Savings_account', 'Present_employment', 'Installment_rate', 'Personal_status',
'Other_debtors', 'Residence_since', 'Property', 'Age', 'Other_installment_plans',
'Housing', 'Existing_credits', 'Job', 'Number_liable', 'Telephone', 'Foreign_worker',
'Target']

data.columns = columns

# Convert categorical variables to numeric using Label Encoding
for column in data.select_dtypes(include=['object']).columns:
    le = LabelEncoder()
    data[column] = le.fit_transform(data[column])

# Splitting data into features and target
X = data.drop('Target', axis=1)
y = data['Target']

# Standardizing the features
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Splitting data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0.2,
random_state=42)

# Defining the models to be evaluated
models = {
    'Logistic Regression': LogisticRegression(),
    'Random Forest': RandomForestClassifier(),
    'Gradient Boosting': GradientBoostingClassifier(),
    'XGBoost': XGBClassifier(use_label_encoder=False, eval_metric='logloss')
}
```

```

# Evaluating models
results = {}
for model_name, model in models.items():
    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)
    results[model_name] = {
        'Accuracy': accuracy_score(y_test, y_pred),
        'Precision': precision_score(y_test, y_pred),
        'Recall': recall_score(y_test, y_pred),
        'F1 Score': f1_score(y_test, y_pred),
        'ROC-AUC': roc_auc_score(y_test, y_pred)
    }

# Selecting the best model based on F1 Score
best_model_name = max(results, key=lambda x: results[x]['F1 Score'])
best_model = models[best_model_name]

# Saving the best model
joblib.dump(best_model, 'best_model.pkl')
joblib.dump(scaler, 'scaler.pkl')

# Displaying the results
results_df = pd.DataFrame(results).T
print(results_df)

```

[NOTE: The Streamlit and FastAPI portion will be discussed in the “User Interface & API Integration” section.]

However, after the code was generated, I noticed that the code for data preprocessing was quite sparse. The code generated include conversion of categorical features to numeric using Label Encoding, feature scaling for the entire dataset, and splitting the dataset into training and test sets. The code omitted important preprocessing steps such as checking for missing values, inconsistent values, outliers, and imbalanced classes. There was practically no EDA done (even though CDA explicitly said it would do this in Step 2). There were no considerations for feature engineering and feature selection. For the modelling stage, it used 4 algorithms but only used default hyperparameters. There is no hyperparameter-tuning at all despite CDA saying it would do that in Step 3. These omissions are very glaring i.e. CDA explicitly saying it would do certain tasks but actually it did not fulfil its promises. Despite these shortcomings, CDA did do something positive. Apparently, it recognised the dataset as the German Credit dataset. Perhaps from the file name (german.txt) and the fact that I told CDA that the ML problem is about loan default. Although the file uploaded, german.txt, did not contain any column headings, CDA managed to include the column headings of the german credit dataset when it created the data frame to hold the dataset. Below is my feedback to CDA:

Are you sure you have included all the machine learning best practices? I see very little code for preprocessing, no EDA, no variables selection, and no use of regularization to reduce overfitting. Did you take measures to avoid data leakage and check classes imbalance? Please be thorough.

CDA RESPONSE:

You are right. A thorough approach should include extensive data preprocessing, exploratory data analysis (EDA), feature selection, regularization, handling class imbalance, avoiding data leakage, and evaluating the model thoroughly. Here is a more detailed approach, including these aspects:

*Step 1: Data Preprocessing and EDA**Load the Data**Inspect the Data (check for missing values, data types, etc.)**Handle Missing Values**Encode Categorical Variables**Scale Numerical Variables**Check for Class Imbalance**Split the Data (train/test split)**Step 2: Feature Selection**Correlation Analysis**Feature Importance using models**Step 3: Model Selection and Training**Define Models**Hyperparameter Tuning using GridSearchCV or RandomizedSearchCV**Cross-Validation to avoid data leakage**Step 4: Model Evaluation**Evaluate Models using accuracy, precision, recall, F1 score, and ROC-AUC**Select the Best Model**Step 5: Deployment Preparation**Save the Model and Scaler**Step 6: Deployment**Setup FastAPI Backend**Setup Streamlit Frontend**Here is the comprehensive Python code:*

```
import pandas as pd
import numpy as np

from sklearn.model_selection import train_test_split, GridSearchCV, StratifiedKFold
from sklearn.preprocessing import LabelEncoder, StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassifier
from xgboost import XGBClassifier

from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score,
roc_auc_score, confusion_matrix
from imblearn.over_sampling import SMOTE
```

```

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

# Load the dataset
data = pd.read_csv('/mnt/data/german.txt', sep=' ')

# Adding column names
columns = ['Status_checking', 'Duration', 'Credit_history', 'Purpose', 'Credit_amount',
'Savings_account', 'Present_employment', 'Installment_rate', 'Personal_status',
'Other_debtors', 'Residence_since', 'Property', 'Age', 'Other_installment_plans',
'Housing', 'Existing_credits', 'Job', 'Number_liable', 'Telephone', 'Foreign_worker',
'Target']
data.columns = columns

# Exploratory Data Analysis (EDA)
print("First few rows of the dataset:")
print(data.head())

print("\nData types and missing values:")
print(data.info())

# Checking class imbalance
print("\nClass distribution:")
print(data['Target'].value_counts())

# Encode categorical variables
for column in data.select_dtypes(include=['object']).columns:
    le = LabelEncoder()
    data[column] = le.fit_transform(data[column])

# Splitting data into features and target
X = data.drop('Target', axis=1)
y = data['Target']

# Handling class imbalance using SMOTE
smote = SMOTE(random_state=42)
X_res, y_res = smote.fit_resample(X, y)

# Standardizing the features
scaler = StandardScaler()
X_res_scaled = scaler.fit_transform(X_res)

# Splitting data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X_res_scaled, y_res, test_size=0.2,
random_state=42, stratify=y_res)

# Define models and hyperparameters
models = {

```

```

    'Logistic Regression': LogisticRegression(),
    'Random Forest': RandomForestClassifier(),
    'Gradient Boosting': GradientBoostingClassifier(),
    'XGBoost': XGBClassifier(use_label_encoder=False, eval_metric='logloss')
}

params = {
    'Logistic Regression': {
        'C': [0.1, 1, 10, 100],
        'penalty': ['l1', 'l2'],
        'solver': ['liblinear']
    },
    'Random Forest': {
        'n_estimators': [100, 200, 300],
        'max_depth': [5, 10, 20],
        'min_samples_split': [2, 5, 10]
    },
    'Gradient Boosting': {
        'learning_rate': [0.01, 0.1, 0.2],
        'n_estimators': [100, 200, 300],
        'max_depth': [3, 5, 7]
    },
    'XGBoost': {
        'learning_rate': [0.01, 0.1, 0.2],
        'n_estimators': [100, 200, 300],
        'max_depth': [3, 5, 7]
    }
}

# Perform Grid Search with Cross-Validation
best_models = {}
for model_name in models.keys():
    grid = GridSearchCV(models[model_name], params[model_name], cv=StratifiedKFold(n_splits=5), scoring='f1', n_jobs=-1)
    grid.fit(X_train, y_train)
    best_models[model_name] = grid.best_estimator_

# Evaluating models
results = {}
for model_name, model in best_models.items():
    y_pred = model.predict(X_test)
    results[model_name] = {
        'Accuracy': accuracy_score(y_test, y_pred),
        'Precision': precision_score(y_test, y_pred),
        'Recall': recall_score(y_test, y_pred),
        'F1 Score': f1_score(y_test, y_pred),
        'ROC-AUC': roc_auc_score(y_test, y_pred)
    }

```

```

}

# Selecting the best model based on F1 Score
best_model_name = max(results, key=lambda x: results[x]['F1 Score'])
best_model = best_models[best_model_name]

# Save the best model and the scaler
joblib.dump(best_model, 'best_model.pkl')
joblib.dump(scaler, 'scaler.pkl')

# Displaying the results
results_df = pd.DataFrame(results).T
print("\nModel evaluation results:")
print(results_df)

# Plotting confusion matrix for the best model
y_pred_best = best_model.predict(X_test)
conf_matrix = confusion_matrix(y_test, y_pred_best)
plt.figure(figsize=(10, 7))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues')
plt.title(f'Confusion Matrix for {best_model_name}')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.show()

# Additional insights
print(f"\nBest model selected: {best_model_name}")

```

I examined the code and found out that CDA only did some very minimal data analysis such as printing first few rows of the dataset, the counts of missing values, and counts of the 2 target variable classes. This can hardly be called EDA! However, there is now generated code for handling target variable classes imbalance using SMOTE. But there is no code generated to perform feature selection (which CDA said it would do). No Correlation Analysis (which CDA said it would do also). There is now generated code for hyperparameter optimization using Grid Search with Cross Validation and StratifiedKFold. This is my feedback to CDA:

You didn't really perform any EDA, no features selection done, and I forgot to mention earlier that you did not attempt to perform any outliers analysis and management. You still failed to execute the generated code. I want you to run the code in your runtime environment. Please proceed and don't forget to do things again or perform tasks in a half-baked fashion.

CDA RESPONSE:

Let's go through the complete process step by step, ensuring to cover all aspects of EDA, feature selection, outlier analysis, preprocessing, model evaluation, and deployment preparation. I will execute the code in my environment as requested

... [I skip what CDA said it would do and just display the generated code as this has more substance]

```

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split, GridSearchCV, StratifiedKFold
from sklearn.preprocessing import LabelEncoder, StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassifier
from xgboost import XGBClassifier
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score,
roc_auc_score, confusion_matrix
from imblearn.over_sampling import SMOTE
import joblib
# Load the dataset
data = pd.read_csv('/mnt/data/german.txt', sep=' ', header=None)
# Adding column names (based on the common understanding of the dataset)
columns = ['Status_checking', 'Duration', 'Credit_history', 'Purpose', 'Credit_amount',
'Savings_account', 'Present_employment', 'Installment_rate', 'Personal_status',
'Other_debtors', 'Residence_since', 'Property', 'Age', 'Other_installment_plans',
'Housing', 'Existing_credits', 'Job', 'Number_liable', 'Telephone', 'Foreign_worker',
'Target']
data.columns = columns
# Exploratory Data Analysis (EDA)
data_info = data.info()
data_head = data.head()
# Checking class imbalance
class_distribution = data['Target'].value_counts()
# Visualizing distributions
fig, axs = plt.subplots(5, 5, figsize=(20, 15))
for i, column in enumerate(data.columns, 1):
    sns.histplot(data[column], ax=axs[(i-1)//5, (i-1)%5])
    axs[(i-1)//5, (i-1)%5].set_title(column)
plt.tight_layout()
plt.show()
# Visualizing correlations
plt.figure(figsize=(15, 10))
correlation_matrix = data.corr()
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm')
plt.title('Correlation Matrix')
plt.show()
# Handling outliers using IQR
for column in data.select_dtypes(include=['int64', 'float64']).columns:
    Q1 = data[column].quantile(0.25)
    Q3 = data[column].quantile(0.75)
    IQR = Q3 - Q1
    filter = (data[column] >= Q1 - 1.5 * IQR) & (data[column] <= Q3 + 1.5 * IQR)
    data = data.loc[filter]
# Encode categorical variables

```

```

for column in data.select_dtypes(include=['object']).columns:
    le = LabelEncoder()
    data[column] = le.fit_transform(data[column])

# Splitting data into features and target
X = data.drop('Target', axis=1)
y = data['Target']

# Handling class imbalance using SMOTE
smote = SMOTE(random_state=42)
X_res, y_res = smote.fit_resample(X, y)

# Standardizing the features
scaler = StandardScaler()
X_res_scaled = scaler.fit_transform(X_res)

# Splitting data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X_res_scaled, y_res, test_size=0.2,
random_state=42, stratify=y_res)

# Define models and hyperparameters
models = {
    'Logistic Regression': LogisticRegression(),
    'Random Forest': RandomForestClassifier(),
    'Gradient Boosting': GradientBoostingClassifier(),
    'XGBoost': XGBClassifier(use_label_encoder=False, eval_metric='logloss')
}

params = {
    'Logistic Regression': {
        'C': [0.1, 1, 10, 100],
        'penalty': ['l1', 'l2'],
        'solver': ['liblinear']
    },
    'Random Forest': {
        'n_estimators': [100, 200, 300],
        'max_depth': [5, 10, 20],
        'min_samples_split': [2, 5, 10]
    },
    'Gradient Boosting': {
        'learning_rate': [0.01, 0.1, 0.2],
        'n_estimators': [100, 200, 300],
        'max_depth': [3, 5, 7]
    },
    'XGBoost': {
        'learning_rate': [0.01, 0.1, 0.2],
        'n_estimators': [100, 200, 300],
        'max_depth': [3, 5, 7]
    }
}

# Perform Grid Search with Cross-Validation
best_models = {}

```

```

for model_name in models.keys():
    grid = GridSearchCV(models[model_name], params[model_name],
    cv=StratifiedKFold(n_splits=5), scoring='f1', n_jobs=-1)
    grid.fit(X_train, y_train)
    best_models[model_name] = grid.best_estimator_
# Evaluating models
results = {}
for model_name, model in best_models.items():
    y_pred = model.predict(X_test)
    results[model_name] = {
        'Accuracy': accuracy_score(y_test, y_pred),
        'Precision': precision_score(y_test, y_pred),
        'Recall': recall_score(y_test, y_pred),
        'F1 Score': f1_score(y_test, y_pred),
        'ROC-AUC': roc_auc_score(y_test, y_pred)
    }
# Selecting the best model based on F1 Score
best_model_name = max(results, key=lambda x: results[x]['F1 Score'])
best_model = best_models[best_model_name]
# Save the best model and the scaler
joblib.dump(best_model, 'best_model.pkl')
joblib.dump(scaler, 'scaler.pkl')
# Displaying the results
results_df = pd.DataFrame(results).T
# Plotting confusion matrix for the best model
y_pred_best = best_model.predict(X_test)
conf_matrix = confusion_matrix(y_test, y_pred_best)
plt.figure(figsize=(10, 7))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues')
plt.title(f'Confusion Matrix for {best_model_name}')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.show()
results_df, best_model_name

```

The new version of the script now contains code to generate visualizations for features' distributions and Correlation Matrix. However, when CDA attempts to run the entire script, there was an error message saying that the library "imblearn" cannot be found. After checking what packages are available in the CDA runtime, it was revealed that *imblearn* does not exist there. CDA tried to pip install *imblearn* into its runtime but failed. It is ironic that CDA generated the code to use *imblearn* but at the same time its runtime does not allow *imblearn* to be installed.

Another issue I noticed is that CDA is splitting the dataset very late i.e. after most data preprocessing has been done. This will cause problems. Firstly, this will cause Data Leakage when Standardization is performed during Feature Scaling. Data leakage occurs when information from outside the training dataset is used to create the model, which can lead to overly optimistic performance estimates and poor generalization to new, unseen data. If standardization is performed on the entire dataset before splitting, the mean and standard

deviation (used in the Standardization formula) are calculated using information from both the training and test sets. This means that the scaling applied to the training set is influenced by the data in the test set, which leaks information about the test set into the model. The test data is implicitly used to determine the scale of the features in the training set. This means the model has already been exposed to information from the test set, resulting in overly optimistic model performance estimates during testing. Splitting the dataset after classes imbalance treatment will result in the entire dataset to be balanced. This is incorrect because the Test set is not supposed to be balanced as it represents real-world data which may not be balanced. Below is my feedback to CDA on the timing of dataset splitting and the imblearn issue:

I noticed that you split the dataset at quite a late stage i.e. just before model training. Is there a risk of data leakage in doing that? Despite having told you twice about performing features selection, you still have not done so. Since you are having a problem with imblearn, I suggest that you use the sklearn resample package instead.

CDA RESPONSE:

You are correct; splitting the dataset late can lead to data leakage if preprocessing steps (like outlier removal and scaling) are applied to the entire dataset before the split. To avoid data leakage, it's important to split the data into training and testing sets first and then apply preprocessing only to the training set. Also, feature selection should be included as part of the preprocessing pipeline.

[The rest of the output is omitted. The new version of the script appears below.]

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split, GridSearchCV, StratifiedKFold
from sklearn.preprocessing import LabelEncoder, StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassifier
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score,
roc_auc_score, confusion_matrix
from sklearn.utils import resample
import joblib

# Load the dataset
data = pd.read_csv('german.txt', sep=' ', header=None)

# Adding column names (based on the common understanding of the dataset)
columns = ['Status_checking', 'Duration', 'Credit_history', 'Purpose', 'Credit_amount',
'Savings_account', 'Present_employment', 'Installment_rate', 'Personal_status',
'Other_debtors', 'Residence_since', 'Property', 'Age', 'Other_installment_plans',
'Housing', 'Existing_credits', 'Job', 'Number_liable', 'Telephone', 'Foreign_worker',
'Target']
data.columns = columns

# Check the data information
data.info()

# Display the target column
```

```

data['Target']

# Split the data into training and testing sets
X = data.drop('Target', axis=1)
y = data['Target']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42, stratify=y)

# Handle class imbalance using resampling
X_train['Target'] = y_train
X_majority = X_train[X_train.Target == 1]
X_minority = X_train[X_train.Target == 2]

X_minority_upsampled = resample(X_minority,
                                replace=True,           # sample with replacement
                                n_samples=len(X_majority),   # to match majority class
                                random_state=42) # reproducible results

X_train_upsampled = pd.concat([X_majority, X_minority_upsampled])

y_train = X_train_upsampled.Target
X_train = X_train_upsampled.drop('Target', axis=1)

# Encode categorical variables on the training set
for column in X_train.select_dtypes(include=['object']).columns:
    le = LabelEncoder()
    X_train[column] = le.fit_transform(X_train[column])

for column in X_test.select_dtypes(include=['object']).columns:
    le = LabelEncoder()
    X_test[column] = le.fit_transform(X_test[column])

# Handling outliers using IQR on the training set
def handle_outliers(df, target):
    indices_to_keep = np.full(df.shape[0], True)
    for column in df.select_dtypes(include=['int64', 'float64']).columns:
        Q1 = df[column].quantile(0.25)
        Q3 = df[column].quantile(0.75)
        IQR = Q3 - Q1
        filter = (df[column] >= Q1 - 1.5 * IQR) & (df[column] <= Q3 + 1.5 * IQR)
        indices_to_keep &= filter
    return df[indices_to_keep], target[indices_to_keep]

X_train, y_train = handle_outliers(X_train, y_train)

# Standardizing the features on the training set
scaler = StandardScaler()

```

```

X_train_scaled = pd.DataFrame(scaler.fit_transform(X_train), columns=X_train.columns)
X_test_scaled = pd.DataFrame(scaler.transform(X_test), columns=X_test.columns)

# Feature Selection using RandomForest importance
rf = RandomForestClassifier(random_state=42)
rf.fit(X_train_scaled, y_train)

feature_importances = pd.DataFrame(rf.feature_importances_, index=X_train.columns,
columns=['importance']).sort_values('importance', ascending=False)
selected_features = feature_importances.head(10).index.tolist()
print("\nSelected features from RandomForest:")
print(selected_features)

# Select only the important features
X_train_selected = X_train_scaled[selected_features]
X_test_selected = X_test_scaled[selected_features]

# Define models and hyperparameters
models = {
    'Logistic Regression': LogisticRegression(max_iter=1000),
    'Random Forest': RandomForestClassifier(random_state=42),
    'Gradient Boosting': GradientBoostingClassifier(random_state=42)
}

params = {
    'Logistic Regression': {
        'C': [0.1, 1, 10, 100],
        'penalty': ['l1', 'l2'],
        'solver': ['liblinear']
    },
    'Random Forest': {
        'n_estimators': [100, 200, 300],
        'max_depth': [5, 10, 20],
        'min_samples_split': [2, 5, 10]
    },
    'Gradient Boosting': {
        'learning_rate': [0.01, 0.1, 0.2],
        'n_estimators': [100, 200, 300],
        'max_depth': [3, 5, 7]
    }
}

# Perform Grid Search with Cross-Validation focusing on recall
best_models = {}
for model_name in models.keys():
    grid = GridSearchCV(models[model_name], params[model_name],
                        cv=StratifiedKFold(n_splits=5), scoring='recall', n_jobs=-1)
    grid.fit(X_train_selected, y_train)
    best_models[model_name] = grid.best_estimator_

```

```
best_models[model_name] = grid.best_estimator_

# Evaluating models
results = {}
for model_name, model in best_models.items():
    y_pred = model.predict(X_test_selected)
    results[model_name] = {
        'Accuracy': accuracy_score(y_test, y_pred),
        'Precision': precision_score(y_test, y_pred),
        'Recall': recall_score(y_test, y_pred),
        'F1 Score': f1_score(y_test, y_pred),
        'ROC-AUC': roc_auc_score(y_test, y_pred)
    }

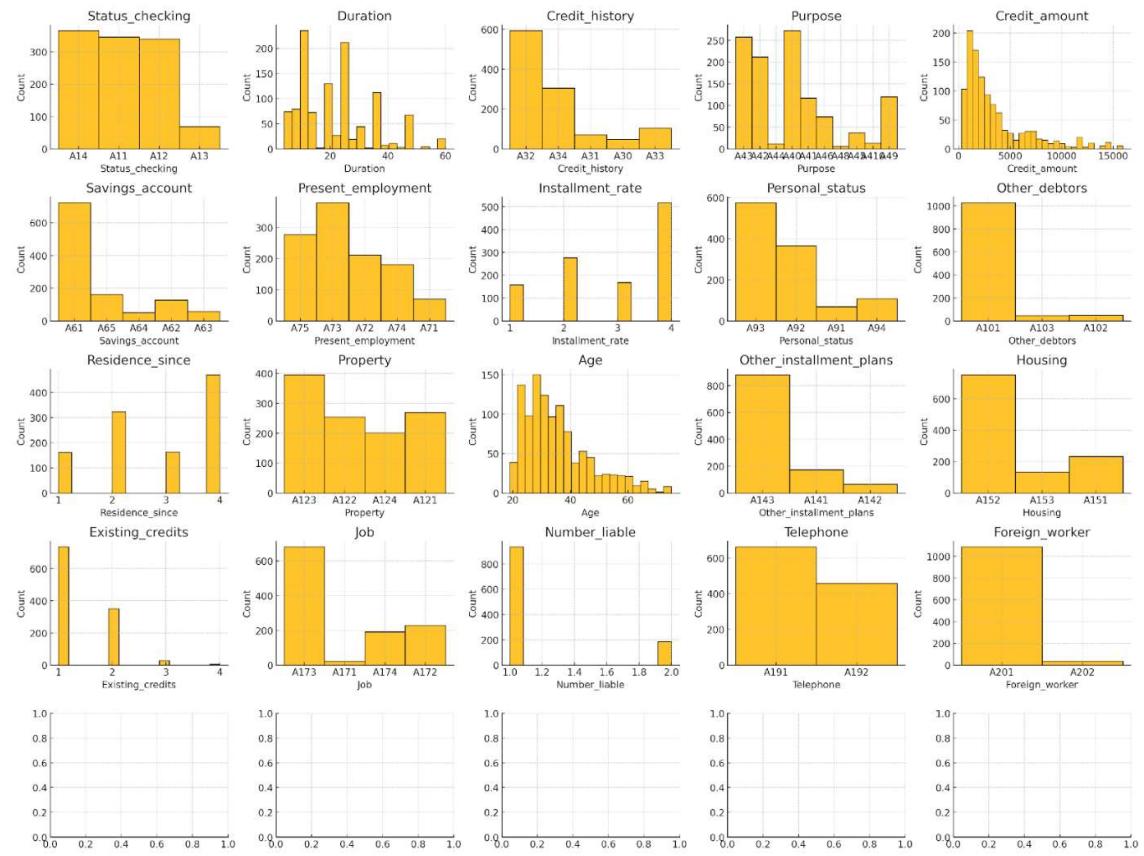
# Selecting the best model based on Recall
best_model_name = max(results, key=lambda x: results[x]['Recall'])
best_model = best_models[best_model_name]

# Save the best model and the scaler
joblib.dump(best_model, 'best_model.pkl')
joblib.dump(scaler, 'scaler.pkl')

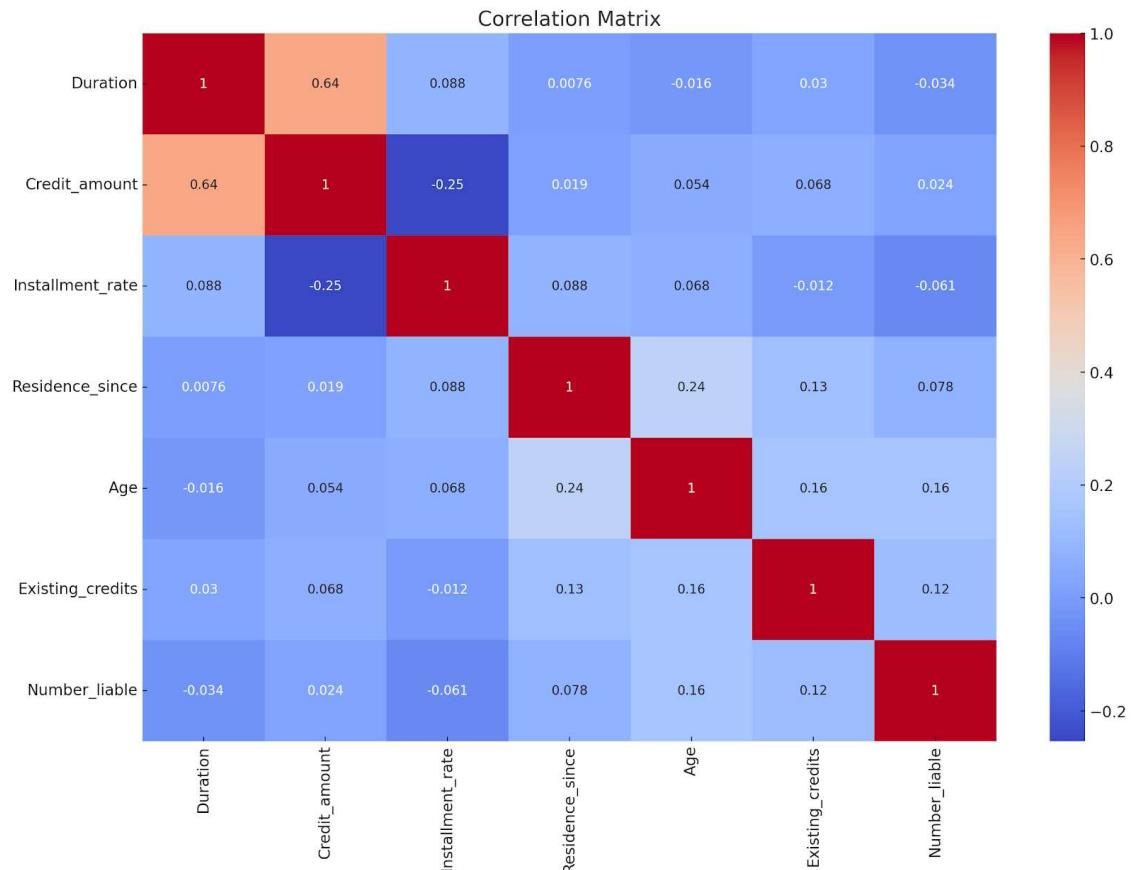
# Displaying the results
results_df = pd.DataFrame(results).T
print(results_df)

# Plotting confusion matrix for the best model
y_pred_best = best_model.predict(X_test_selected)
conf_matrix = confusion_matrix(y_test, y_pred_best)
plt.figure(figsize=(10, 7))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues')
plt.title(f'Confusion Matrix for {best_model_name}')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.show()
```

The visualisations for the Features' Distributions appear as follows:



Snippet 5.35



Snippet 5.36

The Correlation Matrix:

As you can see from the latest version of the script, the code for splitting the dataset is now repositioned to a much earlier point in the script. This solves the data leakage problem related to Feature Scaling and also avoiding the test set to be balanced during the imbalanced data handling process. In addition, the code for Feature Selection is finally in place.

When script execution reaches the Modelling logic, an unanticipated problem occurred. The CDA runtime failed, saying:

It seems there are persistent issues with running long computations like grid search in the current environment. To ensure you can proceed effectively, here's the complete code for you to run in your local environment

Despite numerous attempts to run on different days and at different times, the CDA runtime kept failing and never once successfully performed a grid search. Even when the hyperparameter values were cut down or random grid search was used, the CDA runtime still failed. The conclusion is that it is not possible to perform Hyperparameter Optimization on the CDA

runtime. Without hyperparameter optimization, it is not possible to find the best model and Hyperparameters. Fortunately, I had no problem running the entire script on my laptop.

The Best Model (based on the Recall score) is:

```
GradientBoostingClassifier
GradientBoostingClassifier(max_depth=7, n_estimators=300, random_state=42)
```

Snippet 5.37

The metrics for all 3 algorithms are:

	Accuracy	Precision	Recall	F1 Score	ROC-AUC
Logistic Regression	0.745	0.844961	0.778571	0.810409	0.722619
Random Forest	0.735	0.780645	0.864286	0.820339	0.648810
Gradient Boosting	0.735	0.777070	0.871429	0.821549	0.644048

Snippet 5.38

To prepare for deployment, 2 objects must be saved: (1) the Best Model object and (2) the Scaler object which was used to perform Standardization on the numeric features of the Training set. This same scaler object will be used to perform Standardization on the numeric features that will be keyed in via the UI input screen in the deployment environment. CDA has intelligently anticipated this task to save these objects. In the first code generation, it had generated the code to perform this task. Both objects will be saved as “.pkl” files. These 2 files will be referenced by the API that the end user will interact with.

5.2.2 User Interface & API Integration Implementation

I will not go through step-by-step how the initial version of the Streamlit and FastAPI scripts evolve to their final versions because that would make the documentation very long.

However, I will list out the prompts that were issued and then show the final versions of the 2 scripts.

Prompts:

1. Improve the user-friendliness of the Streamlit app that you generated
2. Provide the updated code for FastAPI, ensuring that the dataframe is limited to the selected features
3. Although your codes for Streamlit and FastAPI enable the prediction of whether the person will default on the loan, it does not provide any sort of explanation to explain the prediction made. How would you provide the explanation? Provide the code and output
4. Use an alternative for st.pyplot

Not all the code were obtained from CDA. For the SHAP logic, most of the code is hand-

coded because CDA could not provide the desired code.

a) Actual User Interface

User Actions
Fill in the loan application details on the main screen and click 'Submit' to get the prediction.

Loan Default Prediction

Loan Application Details

- Personal Information
- Loan Details
- Financial Information
- Credit History
- Property Information

Submit

User Actions
Fill in the loan application details on the main screen and click 'Submit' to get the prediction.

Loan Default Prediction

Loan Application Details

Personal information

- Status of existing checking account
 - A11: ... < 0 DM
- Age in years
 - 18
- Personal status and sex
 - A91: male, divorced/separated

Loan Details

- Credit amount
 - 0
- Duration in months
 - 6
 - 6
 - 72
- Purpose of the loan
 - A40: car (new)

User Actions
Fill in the loan application details on the main screen and click 'Submit' to get the prediction.

Loan Default Prediction

Loan Application Details

Credit history

- A30: no credits taken/all credits paid back duly

Property information

- Property
 - A121: real estate

Submit

Prediction: No Default

higher == lower

-0.078 -0.078 -0.078 -2.44

checking = -0.145 Credit_history = -0.341 Personal_status = -0.059 Present_employment = -1.941

Snippet 5.39**b) FastAPI Backend Algorithm**

- Create a FastAPI application instance (app).
- Load machine learning model from file (final_best_model.pkl).
- Load data scaler from file (final_scaler.pkl).
- Load training data (X_train) from CSV file (fin_gen_X_train_resampled.csv).
- List of selected features: ['Status_checking', 'Credit_amount', 'Age', 'Duration', 'Savings_account', 'Purpose', 'Present_employment', 'Credit_history', 'Property', 'Personal_status'].
- Define a data structure for input using Application class, with fields:
- Status_checking, Credit_amount, Age, Duration, Savings_account, Purpose, Present_employment, Credit_history, Property, Personal_status.
- Create an SHAP Plot endpoint /shapplot (GET request) that:
 - Creates SHAP explainer for the model using X_train and the selected features.
 - Calculates SHAP values for the selected input data (data_selected).
 - Saves SHAP force plot as an HTML file (gen_shap_force_plot.html).
 - Returns the saved SHAP plot as an HTML file.
- Create the Prediction endpoint /predict (POST request) that:
 - Receives input data (application details).
 - Converts the input data into a DataFrame.
 - Scales the data using the loaded scaler.
 - Selects the necessary features.
 - Uses the pre-trained model to predict the outcome (loan default prediction).
- Returns the prediction result as a response.

c) Streamlit User Interface Algorithm

- Use Streamlit to set up the interface.
- Define and apply custom styles for background, buttons, scrollbars, and layout.
- Set the title as "Loan Default Prediction".
- Add a sidebar for user actions
 - Sidebar title: "User Actions".
- Instructions: Fill in loan application details and click 'Submit'.
- Create mapping dictionaries for categorical variables:
- Status_checking, Credit_history, Purpose, Savings_account, Present_employment, Personal_status, Property.
- Create input form (st.form) to gather loan application details:
 - Personal Information: Status_checking, Age, Personal_status.
 - Loan Details: Credit_amount, Duration, Purpose.
 - Financial Information: Savings_account, Present_employment.
 - Credit History: Credit_history.
 - Property Information: Property.
- On form submission:
 - Convert categorical variables to numerical values using mapping dictionaries.
 - Send a POST request to the FastAPI prediction endpoint.

- Display the prediction result:
- If result is default, show an error message.
- If no default, show a success message.
- Send a GET request to the SHAP plot endpoint.
- Display the SHAP force plot in a scrollable container.

5.2.3 System Integration Testing

ID	Test Objective	Test Data	Precondition	Steps	Expected Result	Actual Result	Status
1	View the Application	None	The Frontend and Backend servers must be running.	<ol style="list-style-type: none"> 1. Open conda prompt & navigate to the directory containing streamlit script using cd 2. Enter the command “streamlit run <filename>” on conda prompt 3. Wait for a message to be displayed on conda prompt 4. View Streamlit app on the browser 5. Open another conda prompt & navigate to the directory containing fastapi script using cd 6. Enter the command “uvicorn <filename>:app – reload” 7. Once there is a message saying the server has started, you can make predictions on the application. 	<p>A message is displayed on the conda prompt stating that the Streamlit app can be viewed on the browser.</p> <p>The app can be viewed on the browser.</p> <p>A message is displayed on the conda prompt stating that the model has been built and that the FastAPI server has started.</p>	<p>A message is displayed on the conda prompt stating that the Streamlit app can be viewed on the browser.</p> <p>The app can be viewed on the browser.</p> <p>A message is displayed on the conda prompt stating that the model has been built and that the FastAPI server has started.</p>	Pass

2	To view the input fields to be entered	None	Must be on the application page	Open each of the 5 container expanders, such as “Personal details”	Displays the available input fields in each container expander	Displays the available input fields in each container expander	Pass
3	To predict using the default values set by the application	Use default values provided by the application	Must be on the application page	1. Click on the “Submit” button	Displays the prediction as well as a SHAP force plot	Displays the prediction as well as a SHAP force plot	Pass
4	To predict that a defaulter will default	1. Status of existing checking account = A11: ... < 0DM 2. Age in years = 18 3. Personal status and sex = A93: male, single 4. Credit Amount = 20000 5. Duration in months = 51 6. Purpose of the loan = A40: car (new) 7. Savings account/bonds = A61: ... < 100 DM 8. Present employment since = A71: unemployed 9. Credit history = A33: delay in paying off in the past 10. Property = A124: unknown / no property	Must be on the application page	1. Open each of the 5 container expanders, such as “Personal details” 2. Enter data into each input field using the test data 3. Click on the “Submit” button	Displays the prediction that the person will default. Also displays a SHAP force plot.	Displays the prediction that the person will default. Also displays a SHAP force plot.	Pass
5	To predict that a defaulter will default	1. Status of existing checking account = A14: no checking account 2. Age in years = 22	Must be on the application page	1. Open each of the 5 container expanders, such as “Personal details”	Displays the prediction that the person will default. Also displays a SHAP force plot.	Displays the prediction that the person will default. Also displays a SHAP force plot.	Pass

		<p>3. Personal status and sex = A92: female, divorced/separated/married</p> <p>4. Credit Amount = 30000</p> <p>5. Duration in months = 56</p> <p>6. Purpose of the loan = A40: car (new)</p> <p>7. Savings account/bonds = A61: ... < 100 DM</p> <p>8. Present employment since = A72: ... < 1 year</p> <p>9. Credit history = A33: delay in paying off in the past</p> <p>10. Property = A124: unknown / no property</p>		<p>2. Enter data into each input field using the test data</p> <p>3. Click on the “Submit” button</p>			
6	To predict that a non-defaulter will not default	<p>1. Status of existing checking account = A13: ... >= 200 DM / salary assignments for at least year</p> <p>2. Age in years = 45</p> <p>3. Personal status and sex = A94: male, married/widowed</p> <p>4. Credit Amount = 29999</p> <p>5. Duration in months = 39</p> <p>6. Purpose of the loan = A40: car (new)</p> <p>7. Savings account/bonds =</p>	Must be on the application page	<p>1. Open each of the 5 container expanders, such as “Personal details”</p> <p>2. Enter data into each input field using the test data</p> <p>3. Click on the “Submit” button</p>	<p>Displays the prediction that the person will not default. Also displays a SHAP force plot.</p>	<p>Displays the prediction that the person will not default. Also displays a SHAP force plot.</p>	Pass

		A64: ... >= 1000 DM 8. Present employment since = A75: ... >= 7 years 9. Credit history = A31: all credits at this bank paid back duly 10. Property = A121: real estate				
7	To predict that a non-defaulter will not default	1. Status of existing checking account = A13: ... >= 200 DM / salary assignments for at least year 2. Age in years = 60 3. Personal status and sex = A92: female, divorced/separated/married 4. Loan Details = 40000 5. Duration in months = 39 6. Purpose of the loan = A40: car (new) 7. Savings account/bonds = A64: ... >= 1000 DM 8. Present employment since = A75: ... >= 7 years 9. Credit history = A31: all credits at this bank paid back duly 10. Property = A121: real estate	Must be on the application page	1. Open each of the 5 container expanders, such as "Personal details" 2. Enter data into each input field using the test data 3. Click on the "Submit" button	Displays the prediction that the person will not default. Also displays a SHAP force plot.	Displays the prediction that the person will not default. Also displays a SHAP force plot. Pass

Table 5.2. Test Cases Conducted for Generative AI Approach

5.3 Human Data Scientist Approach

The human approach to machine learning involves a data scientist guiding the entire process of model development, from data collection and preprocessing to model selection and evaluation. This approach emphasizes the importance of domain expertise, intuition, and experience to make critical decisions throughout the workflow. Unlike AutoML or Generative AI-driven approaches, the human approach allows for nuanced understanding, creative problem-solving, and ethical considerations, ensuring that models align with real-world goals and values.

5.3.1 Machine Learning Model Implementation

The human approach to ML model development is dependent on the data scientist's knowledge and skills and so may vary from individual to individual. Two different types of neural networks, TensorFlow Keras Deep Learning Classifier and Multi-Layer Perceptron (MLP), will be used. In the AutoML approach, the best performing Deep Learning algorithm ranked number 14. I decided to give neural networks another go, hopefully the human-tuned algorithms will perform better. MLFlow will be integrated into the process and used for tracking experiments, thus bringing in some MLOps functionality. Based on the .ipynb file contents, these are the steps taken to develop the model: [Note that only non-routine code will be shown and minor steps may not be shown.]

Sequence of Steps

1. Load the dataset into a dataframe.
2. Check the contents of the dataset.
3. Add Column headings to dataframe.
4. Check the dimensions of the dataframe.
5. Display basic information about the dataset (e.g. column names, non-null count for each column, and data type)
6. Display basic statistics about each column (count, mean, std dev, min, max, quartiles).
7. Check for Missing Values (there were none in the dataset)
8. Check for Duplicate Rows (there were none in the dataset)
9. Check for Inconsistent Data Values

CODE:

```
#The expected valid values for some of the columns
c1 = ['A11', 'A12', 'A13', 'A14']
c3 = ['A30', 'A31', 'A32', 'A33', 'A34']
c4 = ['A40', 'A41', 'A42', 'A43', 'A44', 'A45', 'A46', 'A47', 'A48', 'A49', 'A410']
c6 = ['A61', 'A62', 'A63', 'A64', 'A65']
c7 = ['A71', 'A72', 'A73', 'A74', 'A75']
c9 = ['A91', 'A92', 'A93', 'A94', 'A95']
c10 = ['A101', 'A102', 'A103']
c12 = ['A121', 'A122', 'A123', 'A124']
c14 = ['A141', 'A142', 'A143']
c15 = ['A151', 'A152', 'A153']
c17 = ['A171', 'A172', 'A173', 'A174']
c19 = ['A191', 'A192']
c20 = ['A201', 'A202']
```

```

c21 = [1,2]

df_dict = {'checking_acct_status': c1, \
           'credit_history': c3, \
           'purpose': c4, \
           'savings_acct_bonds': c6, \
           'employment_duration': c7, \
           'personal_status_sex': c9, \
           'other_debtors_guarantors': c10, \
           'property': c12, \
           'other_installment_plans': c14, \
           'housing': c15, \
           'job_type': c17, \
           'telephone': c19, \
           'foreign_worker': c20, \
           'credit_risk': c21}
print("Length of dictionary: ", len(df_dict))

valid_columns = []

# Iterate through each column in the DataFrame
for column in df.columns:
    if column in df_dict:
        valid_values = df_dict[column]
        is_valid = True
        # Check each value in the column
        for idx, value in df[column].items():
            if value not in valid_values:
                print(f"Invalid value in column '{column}' at index {idx}: {value}")
                is_valid = False
        if is_valid:
            valid_columns.append(column)

# Display the valid columns and their count
print("\nValid columns:")
for col in valid_columns:
    print(col)
print(f"\nNumber of valid columns: {len(valid_columns)}")

```

OUTPUT

```

Valid columns:
checking_acct_status
credit_history
purpose
savings_acct_bonds
employment_duration
personal_status_sex
other_debtors_guarantors
property
other_installment_plans
housing
job_type
telephone
foreign_worker
credit_risk

```

Number of valid columns: 14

COMMENT:

None of the categorical/discrete columns contain any invalid values so there is no data inconsistency in those columns.

10. Display the Count of Unique Values for each column

11. Encode Categorical Columns to Numeric using Label Encoding

Here, I am converting the categorical columns into numerical ones so that they can be fitted by machine learning models, which only take numerical data. The code below performs label encoding and displays the mapping for each feature.

CODE:

```
from sklearn.preprocessing import LabelEncoder
cols = ['checking_acct_status', 'credit_history', 'purpose', 'savings_acct_bonds',
'employment_duration', 'personal_status_sex', \
         'other_debtors_guarantors', 'property', 'other_installment_plans',
'housing', 'job_type', 'telephone', 'foreign_worker']

# Dictionary to store the label encoders and their mappings
label_encoders = {}
mappings = {}

# Create a copy of the dataframe to hold the final encoded values
df_final = df.copy()

# Apply label encoding to each column
for col in cols:
    le = LabelEncoder()

    if col == 'purpose':
        # Add the new category 'A47' to the column temporarily
        temp_series = pd.Series(['A47'])
        temp_col = pd.concat([df_final[col].astype(str), temp_series], ignore_index=True)
        le.fit(temp_col)

        # Encode the column in the original dataframe
        df_final[col] = le.transform(df_final[col].astype(str))
        # Remove the temporary 'A47' entry if it was added
        if 'A47' in df_final[col].values:
            df_final = df_final[df_final[col] != le.transform(['A47'])[0]]
    elif col == 'personal_status_sex':
        # Add the new category 'A95' to the column temporarily
        temp_series = pd.Series(['A95'])
        temp_col = pd.concat([df_final[col].astype(str), temp_series], ignore_index=True)
        le.fit(temp_col)

        # Encode the column in the original dataframe
        df_final[col] = le.transform(df_final[col].astype(str))
        # Remove the temporary 'A95' entry if it was added
        if 'A95' in df_final[col].values:
            df_final = df_final[df_final[col] != le.transform(['A95'])[0]]
    else:
        df_final[col] = le.fit_transform(df_final[col])

    label_encoders[col] = le
    mappings[col] = dict(zip(le.classes_, le.transform(le.classes_)))

# Print the mappings for each column
for col, mapping in mappings.items():
    print(f"Mappings for {col}:")
    print(mapping)
```

OUTPUT:

```
Mappings for checking_acct_status:
{'A11': 0, 'A12': 1, 'A13': 2, 'A14': 3}
Mappings for credit_history:
{'A30': 0, 'A31': 1, 'A32': 2, 'A33': 3, 'A34': 4}
Mappings for purpose:
{'A40': 0, 'A41': 1, 'A410': 2, 'A42': 3, 'A43': 4, 'A44': 5, 'A45': 6, 'A46': 7, 'A47': 8, 'A48': 9, 'A49': 10}
```

```

Mappings for savings_acct_bonds:
{'A61': 0, 'A62': 1, 'A63': 2, 'A64': 3, 'A65': 4}
Mappings for employment_duration:
{'A71': 0, 'A72': 1, 'A73': 2, 'A74': 3, 'A75': 4}
Mappings for personal_status_sex:
{'A91': 0, 'A92': 1, 'A93': 2, 'A94': 3, 'A95': 4}
Mappings for other_debtors_guarantors:
{'A101': 0, 'A102': 1, 'A103': 2}
Mappings for property:
{'A121': 0, 'A122': 1, 'A123': 2, 'A124': 3}
Mappings for other_installment_plans:
{'A141': 0, 'A142': 1, 'A143': 2}
Mappings for housing:
{'A151': 0, 'A152': 1, 'A153': 2}
Mappings for job_type:
{'A171': 0, 'A172': 1, 'A173': 2, 'A174': 3}
Mappings for telephone:
{'A191': 0, 'A192': 1}
Mappings for foreign_worker:
{'A201': 0, 'A202': 1}

```

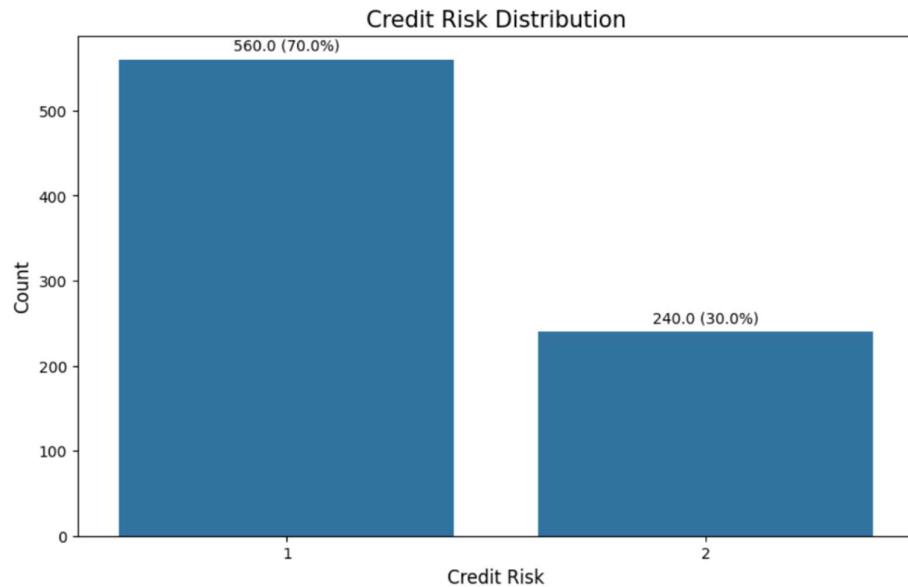
12. Split Dataset into Training and Test sets

I only show one line of code (although there is more than one) which is the one that performs the split:

```
# To perform the train-test split with stratification
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, stratify=y,
random_state=42)
```

Note the “stratify” parameter. We want a stratified split because the target variable classes are imbalanced.

13. Perform EDA (Exploratory Data Analysis) before Balancing



Snippet 5.40

I will just show the visualizations (but not all of them) without the code. Refer to the .ipynb file if you wish to see the code.

The bar chart above confirms that 70% and 30% of the training data belong to 0 (Good Credit) and 1 (Bad Credit) respectively.



Snippet 5.41

We can observe that most of the variables have weak correlations (0.1 to 0.4 or -0.1 to -0.4) with one another while only a few variables have moderate (0.4 to 0.7 or -0.4 to -0.7) correlations with one another. Since there are no highly correlated variables here, there is none to be removed.

Variables with Noticeable Correlations with Each Other:

```

train_df.columns

Index(['checking_acct_status', 'duration_months', 'credit_history', 'purpose',
       'credit_amt', 'savings_acct_bonds', 'employment_duration',
       'instalment_rate_%', 'personal_status_sex', 'other_debtors_guarantors',
       'present_residence_duration', 'property', 'age',
       'other_installment_plans', 'housing', 'existing_credits_at_bank',
       'job_type', 'no_of_people_liable', 'telephone', 'foreign_worker',
       'credit_risk'],
      dtype='object')

necessary_cols = train_df.columns
corr_unstacked = train_df[necessary_cols].corr().unstack().abs()
ranked_corr = corr_unstacked.sort_values(kind="quicksort", ascending=False)
ranked_corr = ranked_corr[ranked_corr != 1]

ranked_corr.head(10)

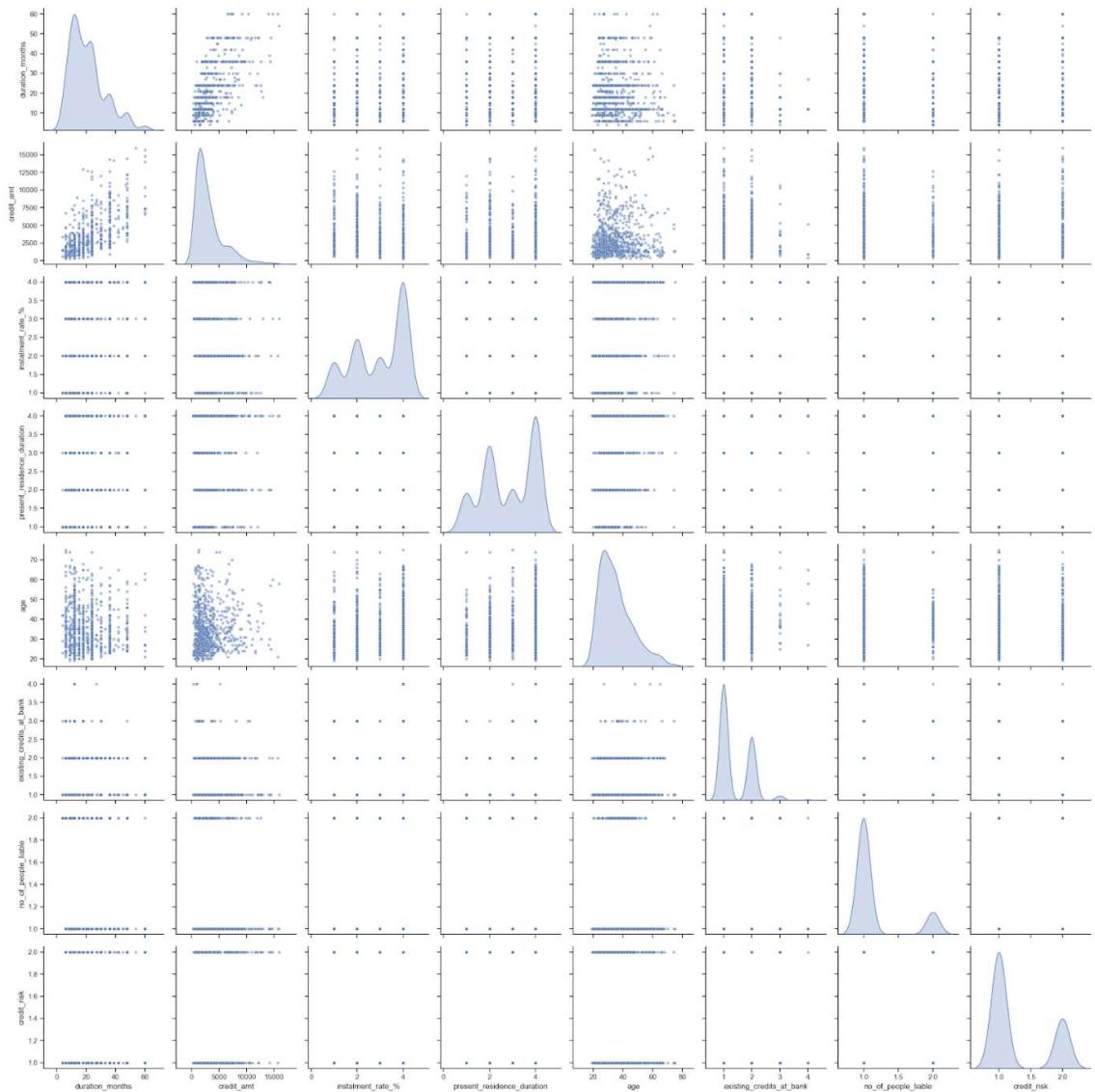
duration_months      credit_amt          0.637919
credit_amt           duration_months    0.637919
credit_history        existing_credits_at_bank 0.448377
existing_credits_at_bank credit_history   0.448377
telephone            job_type           0.382302
job_type              telephone          0.382302
housing               property           0.344972
property              housing            0.344972
credit_risk            checking_acct_status 0.336676
checking_acct_status  credit_risk         0.336676
dtype: float64

```

Snippet 5.42

Note that the pairs of same values are due to the same pair but in different directions.

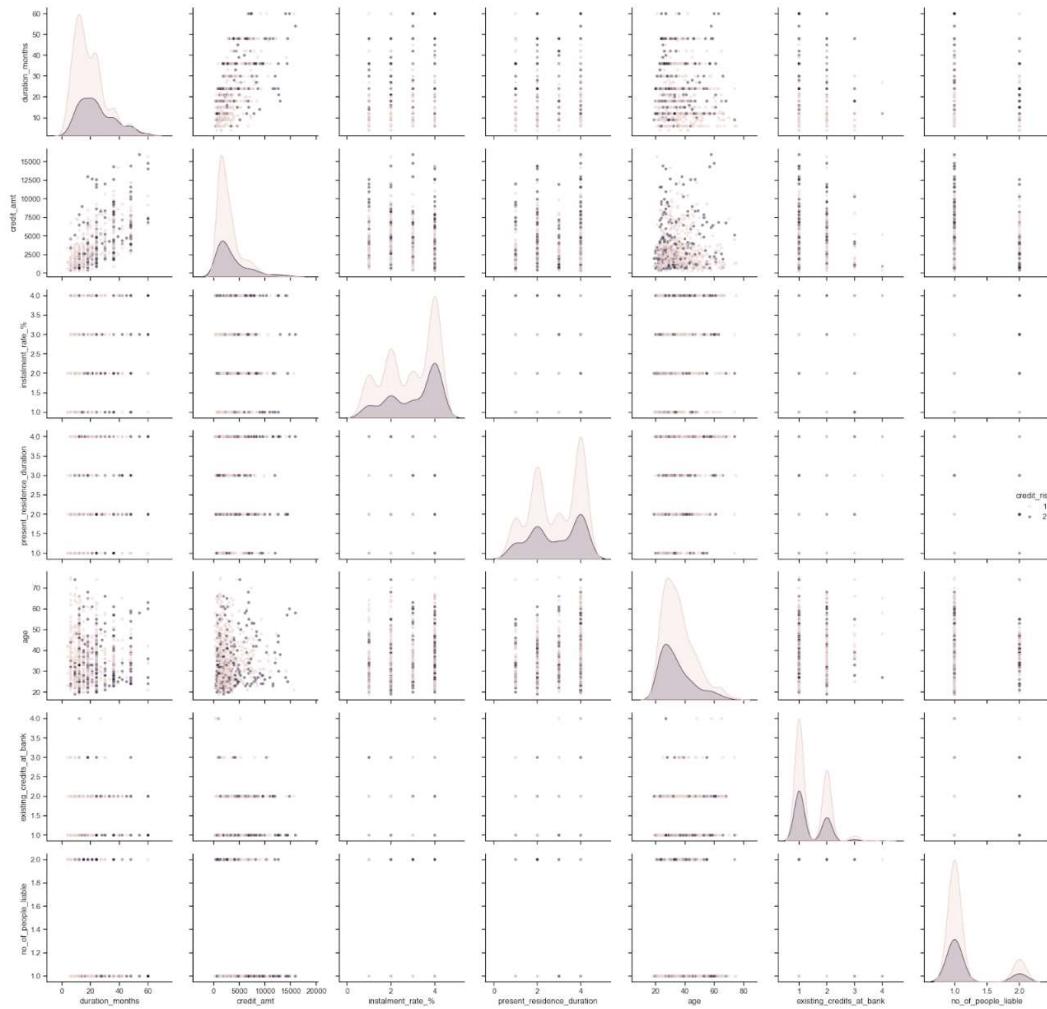
Pairplots for Numeric Variables (Without Hue as credit_risk):



Snippet 5.43

- The KDE plots on the diagonal show the distribution of each individual variable. The distribution of data for all the variables here is not normally distributed. They are either left-skewed or right-skewed.
- The off-diagonal plots show the relationship between two variables.
- Most of the variables have very little correlation with each other, such as 'between present_residence_duration and no_of_people_liable' and 'between existing_credits_at_bank and present_residence_duration'.
- Only a few variables have some correlation with each other, such as 'between duration_months and credit_amt'

Pairplots for Numeric Variables (With Hue as credit_risk):

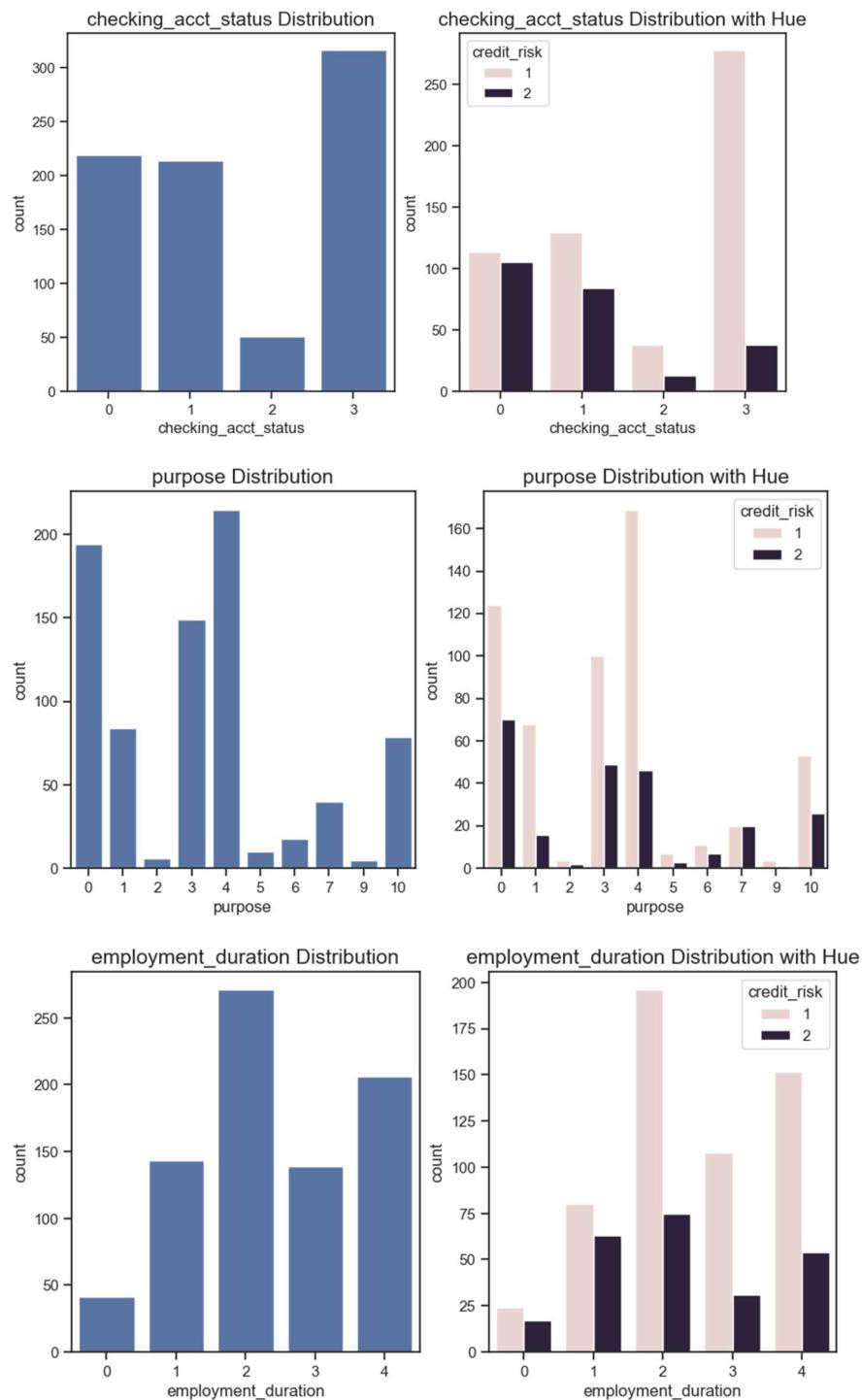


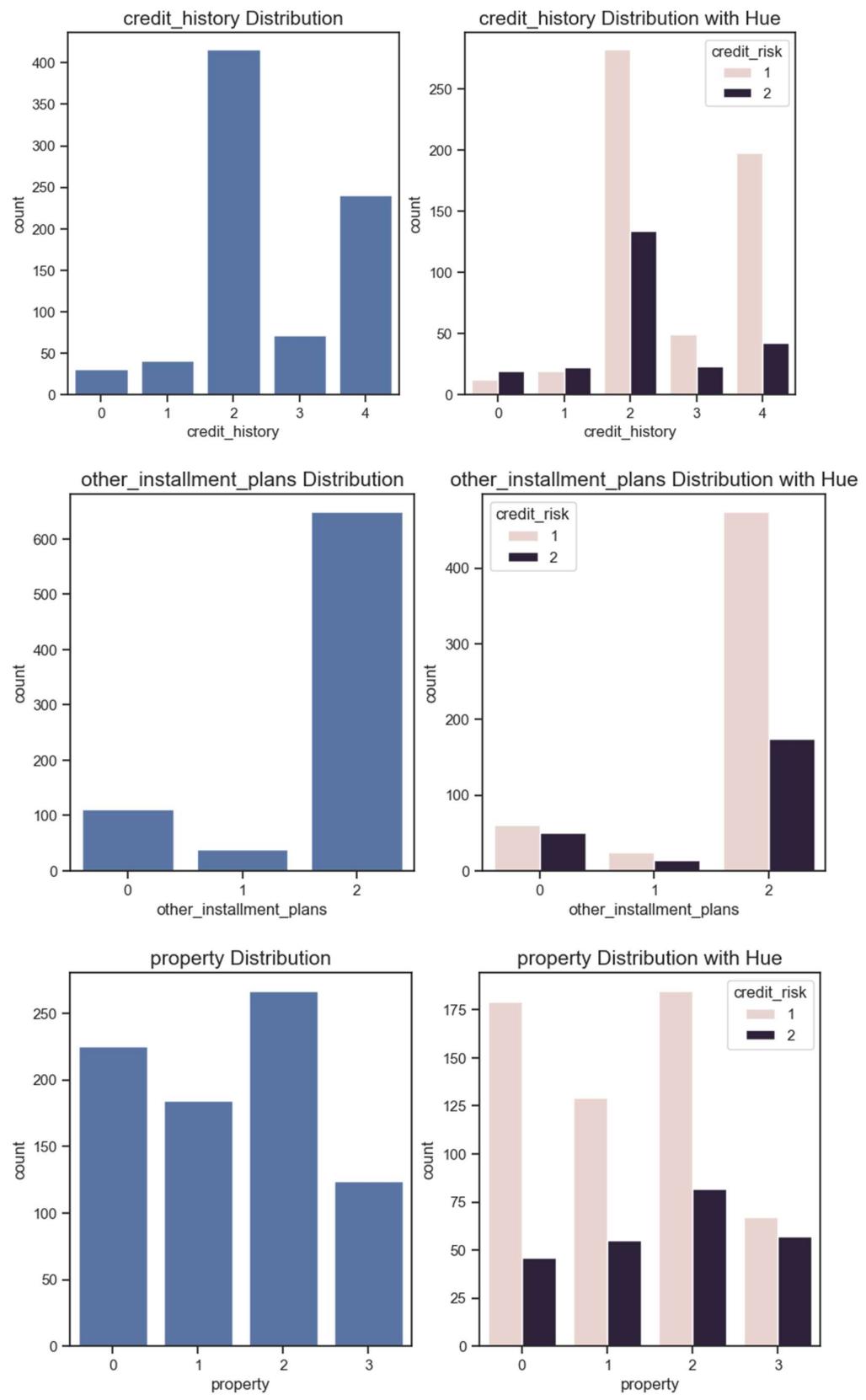
Snippet 5.44

- The hue helps us identify how the relationships between variables vary based on the target.
- Based on the KDE plots, the distribution of data for 'duration_months' and 'present_residence_duration' are quite different between the 2 classes.
- Based on the scatter plots, the patterns and correlations of the data are approximately the same for both classes as their points are plotted at approximately the same locations.

Distribution of Categorical Variables:

[Note that not all the distributions are shown.]

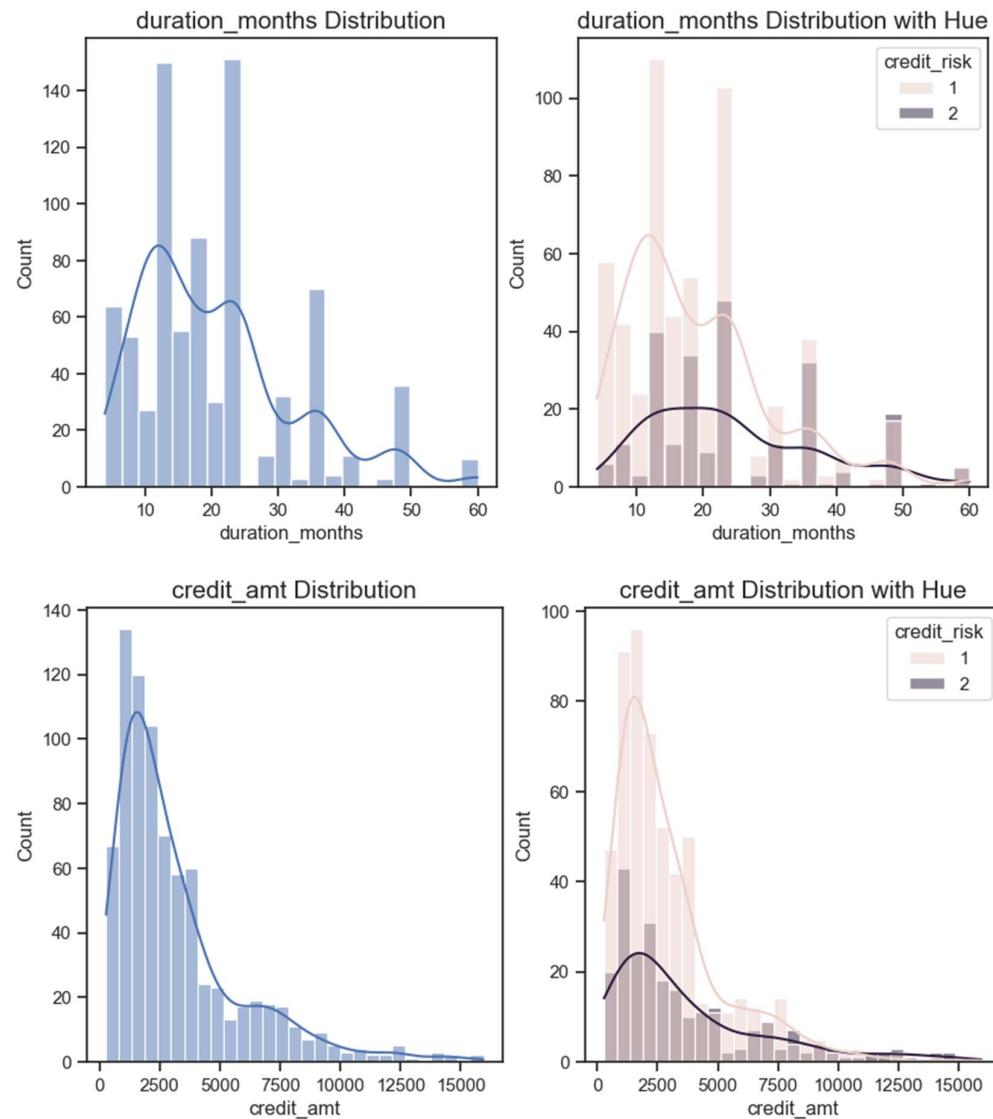


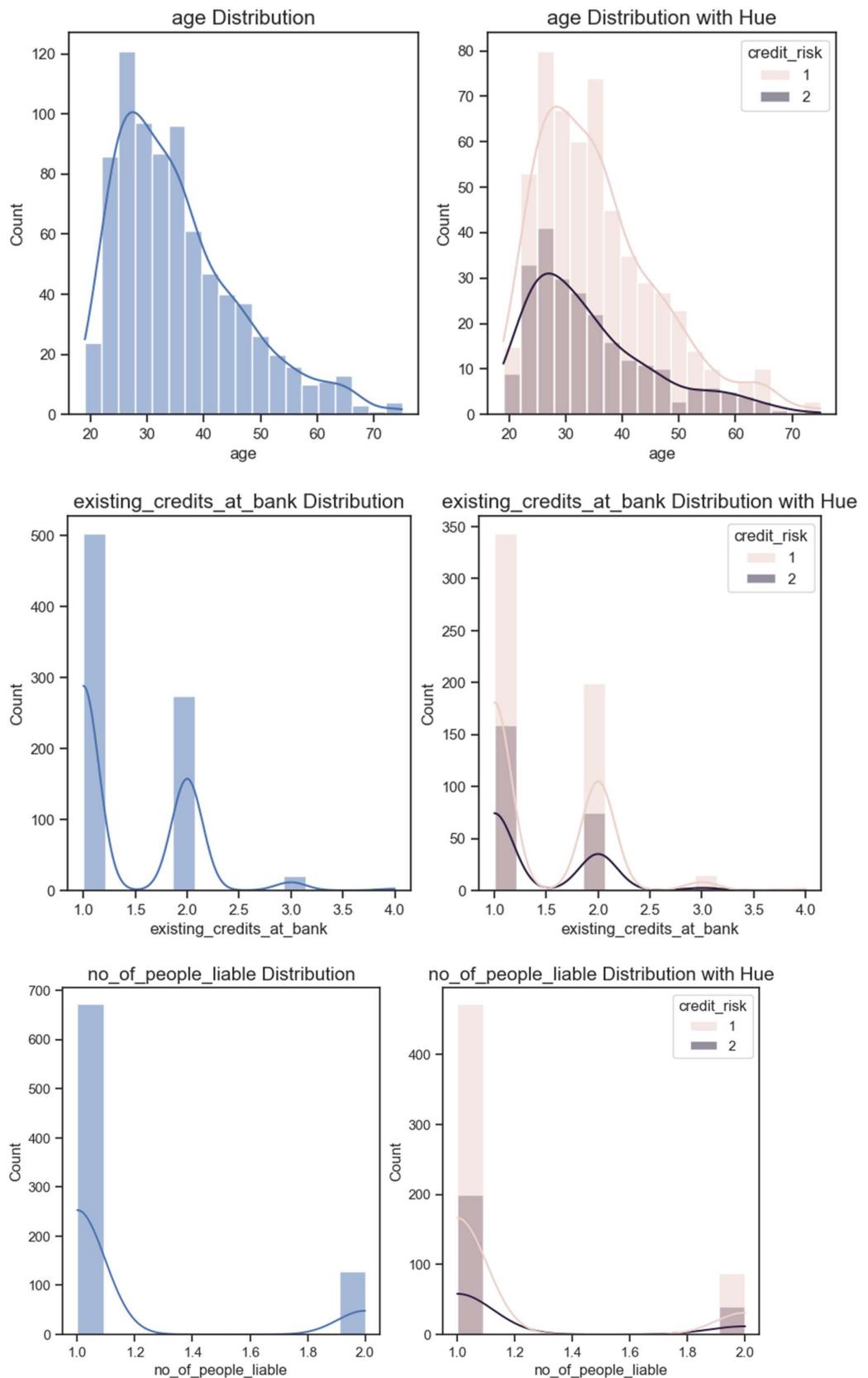


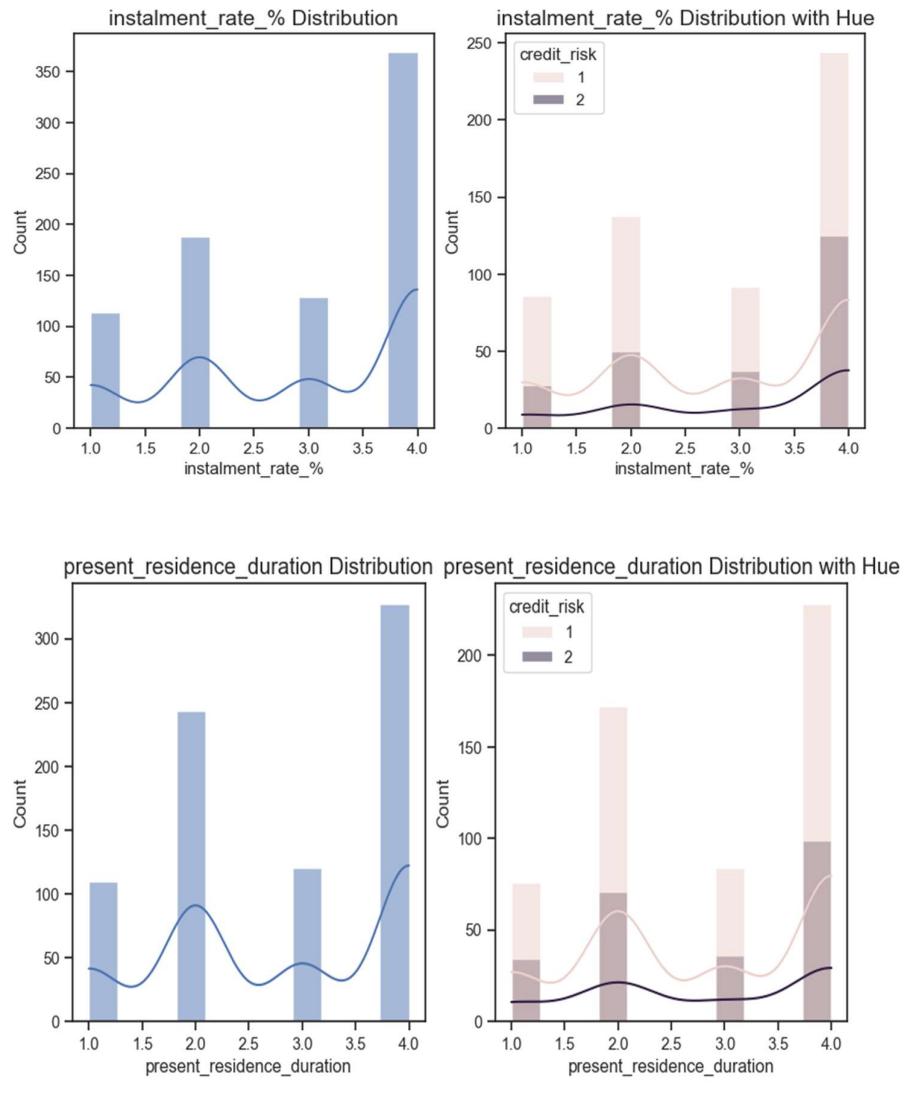
Snippets 5.45

With the target considered, for some variables, the distribution of the categories is not the same, such as 'checking_acct_status', 'purpose' and 'employment_duration'. For some variables, the distribution of the categories is the same, such as 'credit_history' and 'other_installment_plans'. The chart of the 'property' column shows that each category has a comparable frequency. Hence, there are enough rows for each category in the data for the ML algorithm to learn effectively. The bars belonging to class 1 are generally taller than the bars belonging to class 2.

Distributions of Numerical Variables:





**Snippets 5.46**

The distribution of data for duration_months, credit_amt, age, existing_credits_at_bank and no_of_people_liable are right skewed. The distribution of data for instalment_rate_% and present_residence_duration are left skewed. This will be confirmed after balancing the train dataset by calculating the skewness.

14. Balance the Imbalanced Dataset

I used the ADASYN oversampling technique from the *imblearn* library to balance the target variable (Credit_Risk) classes. Note that only the Training set should be balanced. The Test set should not be balanced because it is considered real-world data which should not be manipulated in this way.

CODE:

```

from imblearn.over_sampling import ADASYN
ada = ADASYN(sampling_strategy='minority', random_state=42)
X_train_resampled, y_train_resampled = ada.fit_resample(X_train, y_train)
new_train_df = pd.concat([X_train_resampled, y_train_resampled], axis=1)

print("\nProportion in Train Set after Upsampling:")
print(y_train_resampled.value_counts(normalize=True) * 100)
print(y_train_resampled.value_counts())

print("\nProportion in Train Set before Upsampling:")
print(y_train.value_counts(normalize=True) * 100)
print(y_train.value_counts())

```

OUTPUT:

Proportion in Train Set after Upsampling:

```

credit_risk
2 50.963222
1 49.036778
Name: proportion, dtype: float64

```

```

credit_risk
2 582
1 560
Name: count, dtype: int64

```

Proportion in Train Set before Upsampling:

```

credit_risk
1 70.0
2 30.0
Name: proportion, dtype: float64

```

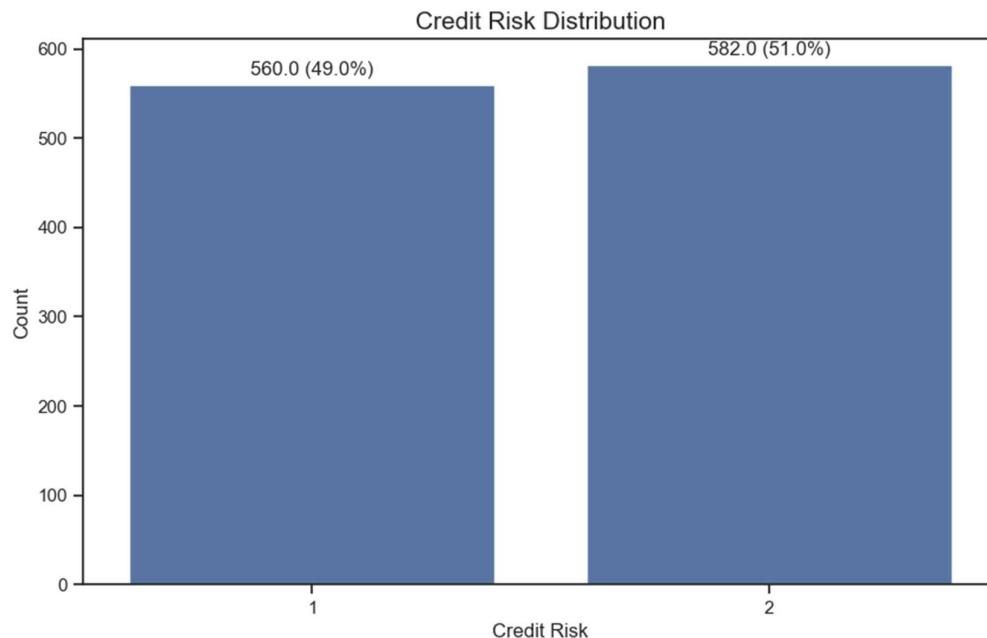
```

credit_risk
1 560
2 240
Name: count, dtype: int64

```

15. Perform EDA After Balancing

The target variable (Credit Risk) distribution after balancing now looks like this:



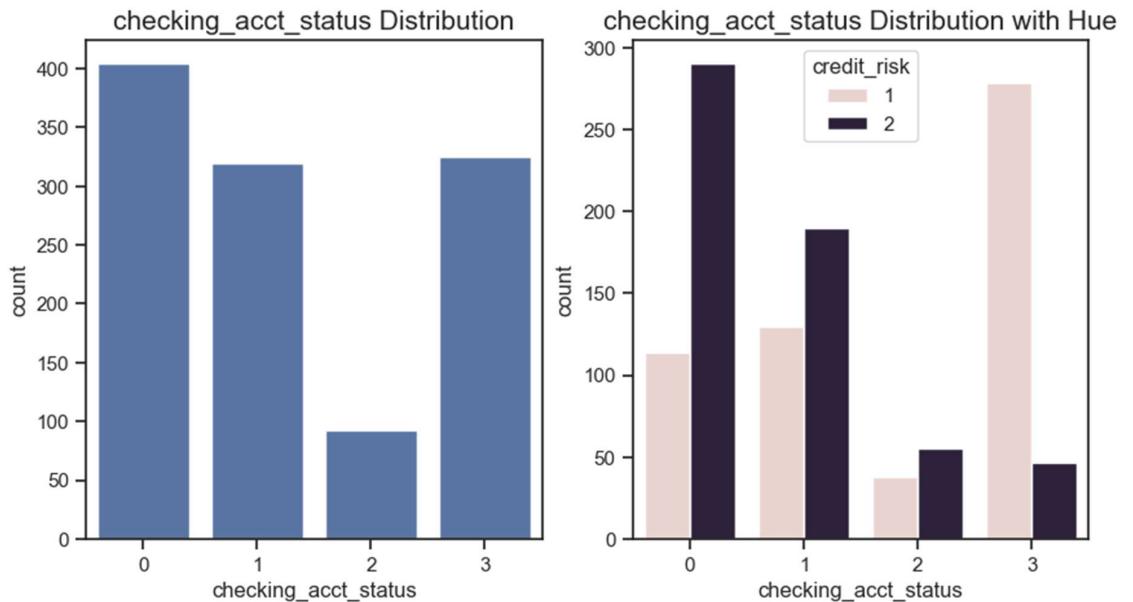
Snippets 5.47

Variables with Noticeable Correlations with Each Other:

duration_months	credit_amt	0.639176
credit_amt	duration_months	0.639176
credit_risk	checking_acct_status	0.450944
checking_acct_status	credit_risk	0.450944
existing_credits_at_bank	credit_history	0.445432
credit_history	existing_credits_at_bank	0.445432
telephone	job_type	0.413051
job_type	telephone	0.413051
credit_amt	telephone	0.345013
telephone	credit_amt	0.345013
dtype: float64		

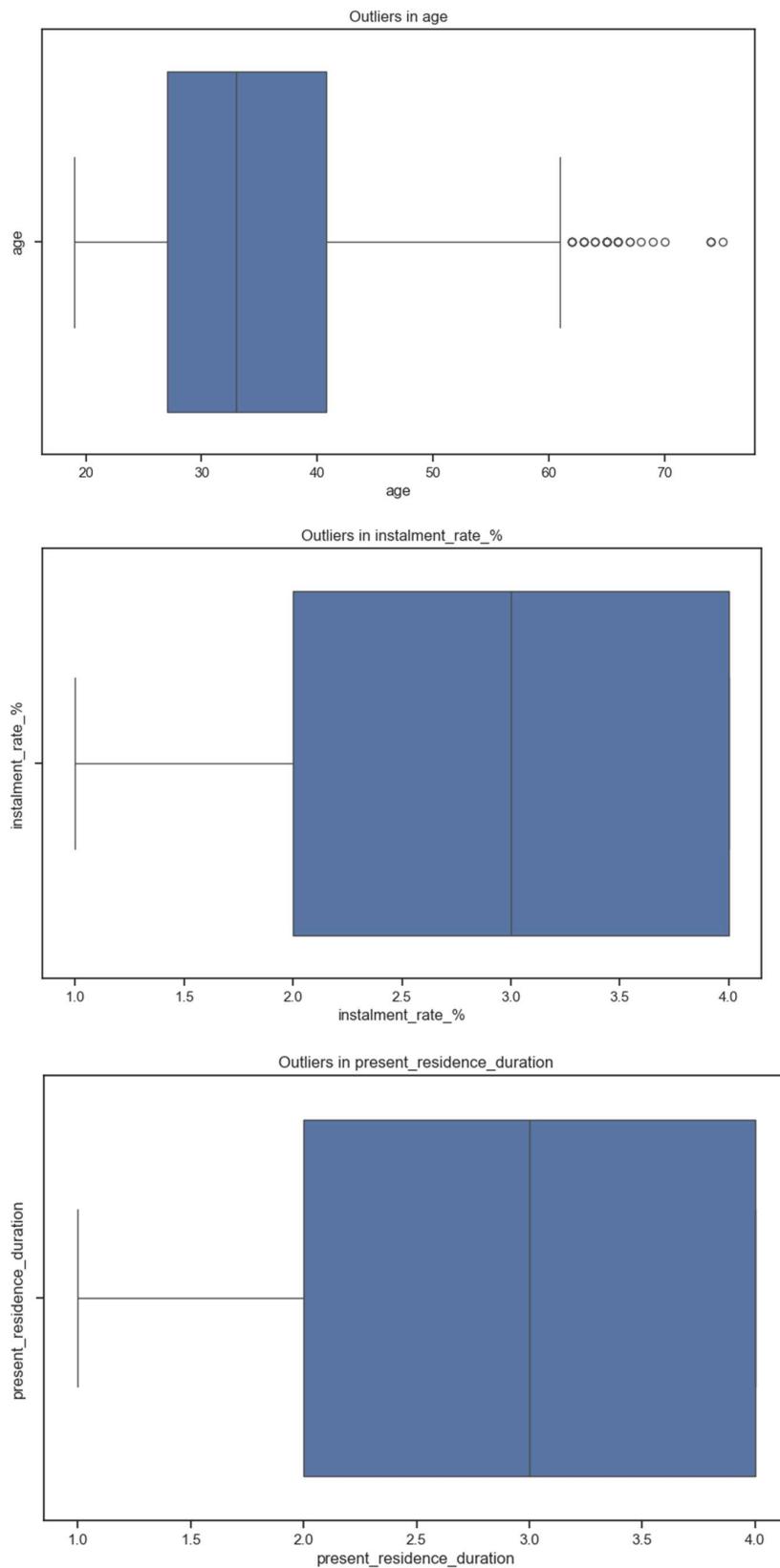
Snippets 5.48

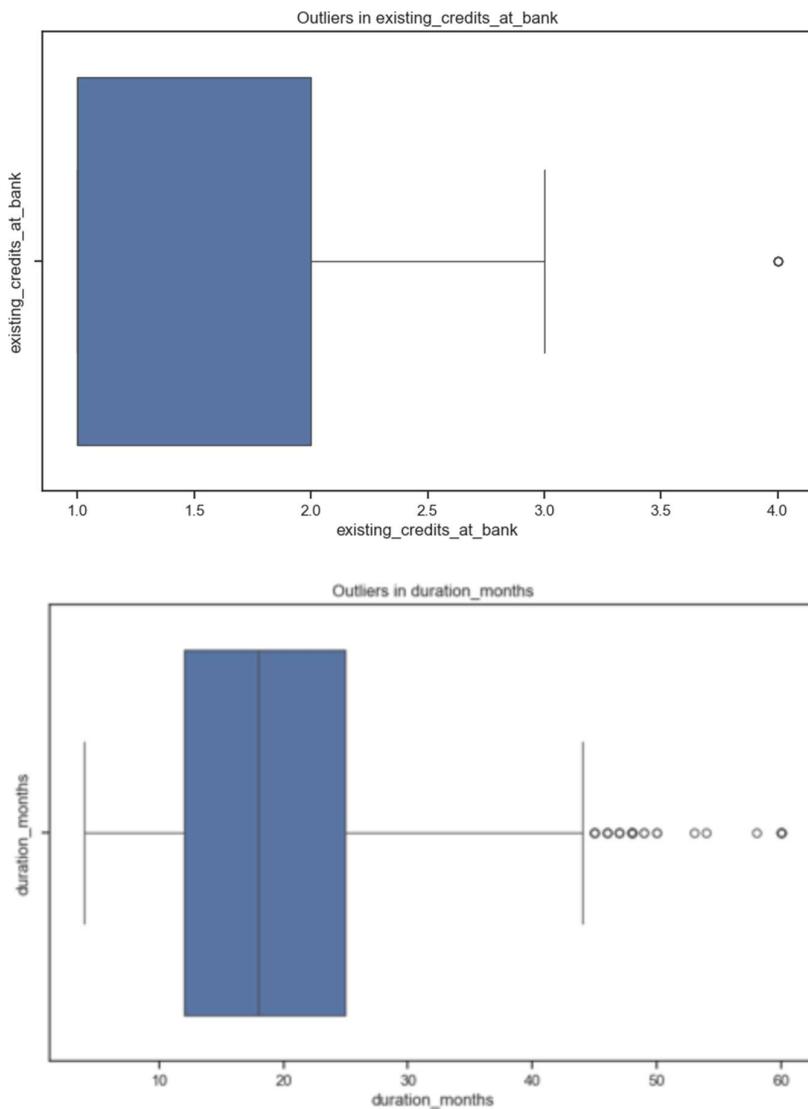
Before balancing, the correlation between checking_acct_status and credit_risk was 0.336676. After balancing, it increased to 0.450944 (as shown above). From the charts below, you can see that the distribution of checking_acct_status has changed as well (compared to before balancing). Later during feature selection, we shall see if checking_acct_status turns up also.

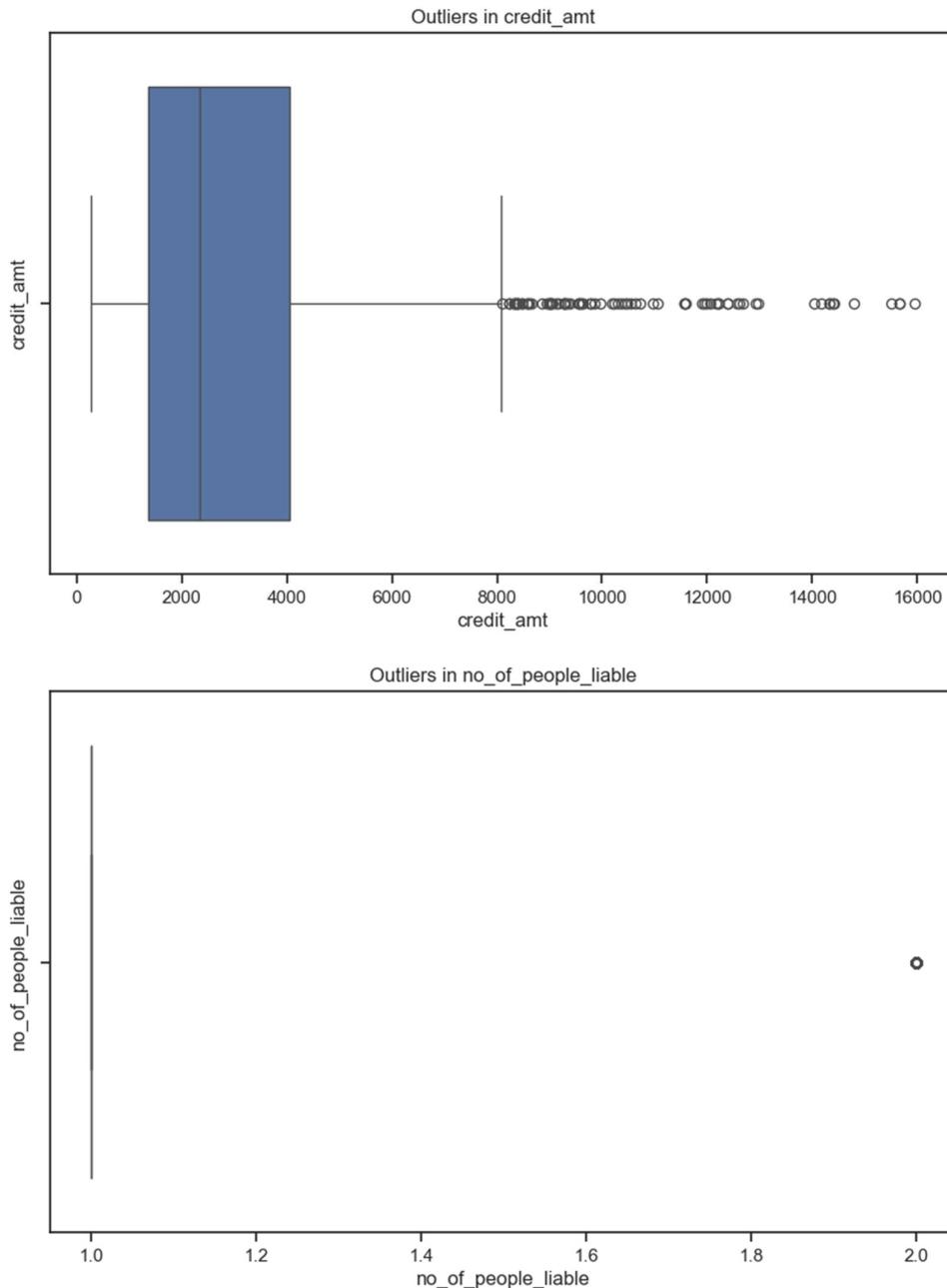


Snippets 5.49

The distributions of the rest of the features did not experience major changes so I am not displaying them here. Instead, we will visualise Outliers using Boxplots:







Snippets 5.50

From the boxplots, outliers are found in the columns:

`duration_months, credit_amt, age, existing_credits_at_bank, no_of_people_liable`

Although the boxplots indicate outliers in some columns, I believe the values are not wrong or out-of-scale. All of the numeric columns have skewed distributions and handling outliers in skewed datasets without careful consideration could result in the loss of important

information or lead to incorrect conclusions about the underlying patterns in the data. Therefore, I am not treating any outliers.

16. Perform Feature Engineering

The following features were chosen for feature engineering, specifically binning.

a) duration_months

- Short term (0-12 months): Generally, a duration of up to one year is considered short term. (Represented by 1)
- Medium term (13-36 months): Durations from over a year to three years are often considered medium term, capturing a mid-range period. (Represented by 2)
- Long term (37 months and above): Durations longer than three years typically fall into the long-term category, representing more extended periods. (Represented by 3)

CODE:

```
# Categorize duration_months with string labels (Training Dataset)
new_train_df['duration_category'] = pd.cut(new_train_df['duration_months'],
                                             bins=[0, 12, 36, float('inf')],
                                             labels=["Short term", "Medium term", "Long term"])

# Map string labels to integers
duration_mapping = {"Short term": 1, "Medium term": 2, "Long term": 3}
new_train_df['duration_category'] = new_train_df['duration_category'].map(duration_mapping).astype(int)

# Categorize duration_months with string labels (Test Dataset)
test_df['duration_category'] = pd.cut(test_df['duration_months'],
                                       bins=[0, 12, 36, float('inf')],
                                       labels=["Short term", "Medium term", "Long term"])

# Map string labels to integers
duration_mapping = {"Short term": 1, "Medium term": 2, "Long term": 3}
test_df['duration_category'] = test_df['duration_category'].map(duration_mapping).astype(int)
```

b) credit_amt

- Small (0-2000): This range is for relatively small loan amounts. (Represented by 1)
- Medium (2001-5000): Medium-sized loans often fall in this range. (Represented by 2)
- Large (5001 and above): Larger loan amounts exceed 5000. (Represented by 3)

```
# Categorize credit_amt with string labels (Training Dataset)
new_train_df['credit_category'] = pd.cut(new_train_df['credit_amt'],
                                         bins=[0, 2000, 5000, float('inf')],
                                         labels=["Small", "Medium", "Large"])

# Map string labels to integers for credit_category
credit_mapping = {"Small": 1, "Medium": 2, "Large": 3}
new_train_df['credit_category'] = new_train_df['credit_category'].map(credit_mapping).astype(int)

# Categorize credit_amt with string labels (Test Dataset)
test_df['credit_category'] = pd.cut(test_df['credit_amt'],
                                    bins=[0, 2000, 5000, float('inf')],
                                    labels=["Small", "Medium", "Large"])

# Map string labels to integers for credit_category
credit_mapping = {"Small": 1, "Medium": 2, "Large": 3}
```

```
test_df['credit_category'] = test_df['credit_category'].map(credit_mapping).astype(int)
```

c) age

- 20s and lower (0-29): This category includes individuals in their twenties and younger. (Represented by 1)
- 30s (30-39): This category includes individuals in their thirties. (Represented by 2)
- 40s (40-49): This category includes individuals in their forties. (Represented by 3)
- 50s (50-59): This category includes individuals in their fifties. (Represented by 4)

```
# Categorize age with string labels (Training Dataset)
new_train_df['age_category'] = pd.cut(new_train_df['age'],
                                       bins=[0, 29, 39, 49, 59, float('inf')],
                                       labels=["20s and lower", "30s", "40s", "50s", "Seniors"])

# Map string labels to integers for age_category
age_mapping = {"20s and lower": 1, "30s": 2, "40s": 3, "50s": 4, "Seniors": 5}
new_train_df['age_category'] = new_train_df['age_category'].map(age_mapping).astype(int)

# Categorize age with string labels (Test Dataset)
test_df['age_category'] = pd.cut(test_df['age'],
                                 bins=[0, 29, 39, 49, 59, float('inf')],
                                 labels=["20s and lower", "30s", "40s", "50s", "Seniors"])

# Map string labels to integers for age_category
age_mapping = {"20s and lower": 1, "30s": 2, "40s": 3, "50s": 4, "Seniors": 5}
test_df['age_category'] = test_df['age_category'].map(age_mapping).astype(int)
```

RESULTS

Note the 3 new category columns below.

	duration_months	duration_category	credit_amt	credit_category	age	age_category
0	36	2	8335		3	47
1	12	1	804		1	38
2	36	2	5371		3	28
3	36	2	3990		2	29
4	48	3	8487		3	24

Snippet 5.51

17. Perform Feature Selection

I chose 4 methods of feature selection and arrive at the final list of selected features by combining them.

a) Method 1: Correlation Analysis

```
def correlation_analysis(df, threshold=0.8):
    corr_matrix = df.corr().abs()
    upper = corr_matrix.where(np.triu(np.ones(corr_matrix.shape), k=1).astype(bool))
    to_drop = [column for column in upper.columns if any(upper[column] > threshold)]
    return df.drop(to_drop, axis=1)
X_corr = correlation_analysis(X_train_resampled)
```

b) Method 2: ANOVA

```
anova_selector = SelectKBest(f_classif, k=10) # Select top 10 features
X_anova = anova_selector.fit_transform(X_train_resampled, y_train_resampled)
```

c) Method 3: Chi-Square

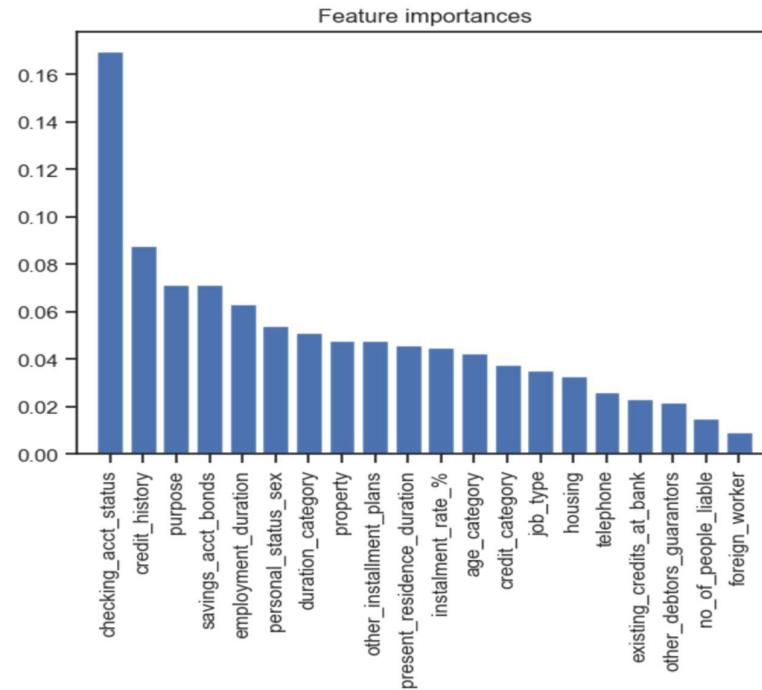
```
chi2_selector = SelectKBest(chi2, k=10) # Select top 10 features
X_chi2 = chi2_selector.fit_transform(X_train_resampled, y_train_resampled)
```

d) Method 4: Random Forest Feature Importance

```
forest = RandomForestClassifier(n_estimators=100, random_state=42)
forest.fit(X_train_resampled, y_train_resampled)
importances = forest.feature_importances_
indices = np.argsort(importances)[::-1]
print("Feature ranking:")
for f in range(X_train_resampled.shape[1]):
    print(f"\t{f + 1}. feature {X_train_resampled.columns[indices[f]]} ({importances[indices[f]]})")
```

Feature ranking:

1. feature checking_acct_status (0.16963987535167116)
2. feature credit_history (0.08801856359012689)
3. feature purpose (0.07147532683639074)
4. feature savings_acct_bonds (0.07128872641091681)
5. feature employment_duration (0.06346282640426831)
6. feature personal_status_sex (0.05418095034862139)
7. feature duration_category (0.05107143944772368)
8. feature property (0.04775598360394187)
9. feature other_installment_plans (0.047736818166979916)
10. feature present_residence_duration (0.04610214069041712)
11. feature instalment_rate_% (0.044957843530393074)
12. feature age_category (0.04258427794017122)
13. feature credit_category (0.03770237151657336)
14. feature job_type (0.03551243231573361)
15. feature housing (0.03296540508773741)
16. feature telephone (0.026084903941962716)
17. feature existing_credits_at_bank (0.02314500223184131)
18. feature other_debtors_guarantors (0.021957727103005933)
19. feature no_of_people_liable (0.0150947067839666)
20. feature foreign_worker (0.009262678697557083)



Snippet: 5.52

Combining the Results

```
# Combine results
selected_features = {
    'correlation': X_corr.columns.tolist(),
    'anova': X_train_resampled.columns[anova_selector.get_support()].tolist(),
    'chi2': X_train_resampled.columns[chi2_selector.get_support()].tolist(),
    'feature_importance': X_train_resampled.columns[indices[:10]].tolist() # Top 10 features from RandomForest
}

print("Selected features by method:")
for method, features in selected_features.items():
    print(f"\nMethod: {method}")
    print(f"Number of features selected: {len(features)}")
    print(f"Selected features: {features}")

print("\nFeature importance scores for top 10 features:")
for idx in indices[:10]:
    print(f"Feature: {X_train_resampled.columns[idx]}, Importance: {importances[idx]:.4f}")
```

OUTPUT:

```

Selected features by method:

Method: correlation
Number of features selected: 20
Selected features: ['checking_acct_status', 'credit_history', 'purpose', 'savings_acct_bonds', 'employment_duration', 'instalment_rate_%', 'personal_status_sex', 'other_debtors_guarantors'

Method: anova
Number of features selected: 10
Selected features: ['checking_acct_status', 'credit_history', 'savings_acct_bonds', 'employment_duration', 'personal_status_sex', 'other_installment_plans', 'housing', 'existing_credits_at']

Method: chi2
Number of features selected: 10
Selected features: ['checking_acct_status', 'credit_history', 'savings_acct_bonds', 'employment_duration', 'personal_status_sex', 'other_installment_plans', 'housing', 'telephone', 'foreign_worker']

Method: feature_importance
Number of features selected: 10
Selected features: ['checking_acct_status', 'credit_history', 'purpose', 'savings_acct_bonds', 'employment_duration', 'personal_status_sex', 'duration_category', 'property', 'other_installment_plans', 'present_residence_duration']

Feature importance scores for top 10 features:
Feature: checking_acct_status, Importance: 0.1696
Feature: credit_history, Importance: 0.0880
Feature: purpose, Importance: 0.0715
Feature: savings_acct_bonds, Importance: 0.0713
Feature: employment_duration, Importance: 0.0635
Feature: personal_status_sex, Importance: 0.0542
Feature: duration_category, Importance: 0.0511
Feature: property, Importance: 0.0478
Feature: other_installment_plans, Importance: 0.0477
Feature: present_residence_duration, Importance: 0.0461

```

Snippet 5.53

Columns to be kept

- 'checking_acct_status' - Selected by all methods.
- 'credit_history' - Selected by all methods.
- 'savings_acct_bonds' - Selected by all methods.
- 'employment_duration' - Selected by all methods.
- 'personal_status_sex' - Selected by all methods.
- 'other_installment_plans' - Selected by all methods.
- 'housing' - Selected by ANOVA, Chi-Square, and Correlation.
- 'telephone' - Selected by ANOVA, Chi-Square, and Correlation.
- 'duration_category' - Selected by all methods.
- 'purpose' - Selected by Correlation and Feature Importance.
- 'property' - Selected by Correlation and Feature Importance.
- 'present_residence_duration' - Selected by Correlation and Feature Importance.
- 'foreign_worker' - Selected by Chi-Square and Correlation.
- 'existing_credits_at_bank' - Selected by ANOVA and Correlation.

18. Perform Feature Scaling

Standardization for numeric features offers a robust solution for feature scaling in credit risk analysis. By centering features around their mean and scaling them based on their standard deviation, Standardization ensures that outliers, which may exist in credit-related data, do not disproportionately influence the scaling process. This method preserves the distribution of original data, aiding in the interpretation of feature importance, and is effective across a variety of machine learning algorithms commonly employed in credit risk modeling, ensuring consistent and reliable performance.

While normalization rescales features to a predetermined range (typically [0, 1]), it's more sensitive to outliers compared to Standardization, potentially skewing the scaled features when

dealing with variables prone to extreme values, such as credit amounts or durations. Additionally, normalization alters the distribution of features, which might affect the interpretation of their importance and relationships in credit risk analysis.

Therefore, Standardization is chosen.

```
from sklearn.preprocessing import StandardScaler
# Initialize StandardScaler
scaler = StandardScaler()
new_train_df.iloc[:, :-1] = scaler.fit_transform(new_train_df.iloc[:, :-1])
test_df.iloc[:, :-1] = scaler.transform(test_df.iloc[:, :-1])
```

Need to save the Scaler object because it is needed to perform feature scaling on user input entered in the deployment environment.

```
from pickle import dump
dump(scaler, open('human_standard_scaler.pkl', 'wb'))
```

19. Perform Machine Learning Modelling

The ML algorithms I have selected for this Binary Classification problem of predicting Loan Default are:

1. Logistic Regression
2. Decision Tree
3. Random Forest
4. Gradient Boosting Machine
5. Multilayer Perceptron Neural Network
6. Deep Learning with TensorFlow Keras Classifier

Although my dataset is quite small (1000 rows), I decided to experiment with Neural Networks in the hope that they can produce as good or better results than the other algorithms traditionally used for binary classification.

I will also be integrating MLFlow into the ML modelling process because it can track each of my experiments. Once the data has been stored in MLFlow, later I can go back and see the hyperparameter values used in a modelling experiment and also compare the results of different experiments.

The following are the import statements for modelling:

```
import time
import mlflow
import sklearn
from sklearn.model_selection import RepeatedStratifiedKFold, GridSearchCV
from scikeras.wrappers import KerasClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.neural_network import MLPClassifier
from tensorflow.keras import models, layers, regularizers, optimizers
from tensorflow.keras.constraints import MaxNorm
from keras.callbacks import EarlyStopping
from keras.metrics import AUC
import pickle
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, roc_auc_score, make_scorer
```

To integrate Keras models with scikit-learn, I need to use the KerasClassifier wrapper from the SciKeras module. This class accepts a function that constructs and returns a neural network

model, along with additional arguments like the number of epochs and batch size, which are passed to the fit() method during training.

KerasClassifier

We will start by defining a function for the KerasClassifier model:

```
def create_keras_model(neurons1, neurons2, l2_reg, dropout_rate, weight_constraint):
    model = models.Sequential()
    model.add(layers.Dense(neurons1, activation=None, input_shape=(X_train_resampled.shape[1]),
                          kernel_initializer='he_normal', kernel_regularizer=regularizers.l2(l2_reg)))
    model.add(layers.BatchNormalization())
    model.add(layers.Activation('relu'))
    model.add(layers.Dropout(dropout_rate))
    model.add(layers.Dense(neurons2, activation=None, kernel_initializer='he_normal',
                          kernel_regularizer=regularizers.l2(l2_reg)))
    model.add(layers.BatchNormalization())
    model.add(layers.Activation('relu'))
    model.add(layers.Dropout(dropout_rate))
    model.add(layers.Dense(1, activation='sigmoid', kernel_initializer="glorot_normal"))
    #model.compile(optimizer=opt, loss='binary_crossentropy', metrics=['accuracy'])
    return model
```

This neural network has 2 *Dense* Hidden Layers:

```
model.add(layers.Dense(neurons1, activation=None, input_shape=(X_train_resampled.shape[1]),
                      kernel_initializer='he_normal', kernel_regularizer=regularizers.l2(l2_reg)))

model.add(layers.Dense(neurons2, activation=None, kernel_initializer='he_normal',
                      kernel_regularizer=regularizers.l2(l2_reg)))
```

Each hidden layer is followed by batch normalization, activation, and dropout. The final layer, a *Dense* layer with a sigmoid activation, is the output layer.

A *Dense* layer means that each neuron in the layer is connected to every neuron in the previous layer. It performs a linear transformation of the input data, followed by an activation function

A *Sequential* model is a linear stack of layers, where each layer has exactly one input tensor and one output tensor. This means that the layers are arranged in a sequence, with each layer feeding its output as the input to the next layer.

To prevent *Overfitting*, the following block of code will halt training when the model's performance stops improving. When the AUC ('auc_2') plateaus or starts to degrade, training will be stopped, preventing overfitting and saving computational resources. The model will stop after 5 epochs of no significant improvement (change less than 0.01) and will restore the weights from the best-performing epoch. This **Regularization** technique is called **Early Stopping**.

```
early_stopping = EarlyStopping(
    monitor='auc_2',          # Monitor validation auc
    min_delta=0.01,            # Minimum change to qualify as an improvement
    patience=5,                # Number of epochs to wait for improvement
    verbose=1,                  # Print message when stopping is triggered
    mode='max',                  # Mode for the metric (maximizing accuracy)
    baseline=None,              # No baseline in this example
    restore_best_weights=True # Restore best weights when stopping
)
```

Next, the 6 models and their hyperparameters (to be used in grid search) are defined:

```
classifiers = {
    'LogisticRegression': {
        'model': LogisticRegression(solver='lbfgs'),
        'params': {
            'C': [0.1, 1, 10],
            'penalty': ['l1', 'l2'],
            'max_iter': [100, 200, 300]
        }
    }
}
```

```

        }
    },
    'DecisionTree': {
        'model': DecisionTreeClassifier(criterion='entropy', random_state=42),
        'params': {
            'max_depth': [5, 10, 15, None],
            'min_samples_split': [2, 5, 10],
            'min_samples_leaf': [1, 2, 4],
            'max_features': [None, 'sqrt', 'log2'],
            'ccp_alpha': [0.0, 0.01, 0.1]
        }
    },
    'RandomForest': {
        'model': RandomForestClassifier(),
        'params': {
            'n_estimators': [50, 100, 200],
            'max_depth': [5, 10, 15, None],
            'min_samples_split': [2, 5, 10],
            'min_samples_leaf': [1, 2, 4],
            'max_features': [None, 'sqrt', 'log2'],
            'bootstrap': [True, False]
        }
    },
    'GradientBoosting': {
        'model': GradientBoostingClassifier(),
        'params': {
            'n_estimators': [50, 100, 200],
            'learning_rate': [0.01, 0.1],
            'max_depth': [3, 4, 5],
            'min_samples_split': [2, 5, 10],
            'min_samples_leaf': [1, 2, 4],
            'max_features': [None, 'sqrt', 'log2'],
            'subsample': [0.6, 0.8, 1.0]
        }
    },
    'MLP': {
        'model': MLPClassifier(max_iter=1000),
        'params': {
            'hidden_layer_sizes': [(50,), (100,), (50, 50)],
            'activation': ['relu', 'elu', 'logistic', 'tanh'],
            'solver': ['adam', 'sgd', 'lbfgs'],
            'alpha': [0.0001, 0.001, 0.01],
            'learning_rate': ['constant', 'invscaling', 'adaptive'],
            'learning_rate_init': [0.0001, 0.001, 0.01, 0.1],
            'batch_size': [32, 64, 128], # Mini-batch size
            'early_stopping': [True],
            'max_iter': [1000], # Maximum number of iterations (epochs)
        }
    },
    'Keras': {
        'model': KerasClassifier(model=create_keras_model, loss="binary_crossentropy", optimizer="SGD", metrics=[AUC()],
        callbacks=[early_stopping], verbose=0),
        'params': {
            'model__dropout_rate': [0.3, 0.6],
            'model__weight_constraint': [2.0, 4.0],
            'model__l2_reg': [0.1, 0.01],
            'model__neurons1': [8, 13],
            'model__neurons2': [8, 13],
            'optimizer__learning_rate': [0.01, 0.1],
            'optimizer__momentum': [0.5, 0.9],
            'epochs': [30, 50],
            'batch_size': [16, 32]
        }
    }
}

```

Next, a special form of **Cross Validation** is defined. *RepeatedStratifiedKFold* is a cross-validation method that combines stratified k-fold cross-validation with repeated cross-validation. It is specifically designed for classification tasks where the data is imbalanced or where you want to ensure that each fold maintains the class distribution. *Repeated Cross-Validation* runs the entire k-fold process multiple times (*n_repeats*) with different random splits of the data. The data is first stratified and split into *n_splits* folds. The process of splitting into

folds is repeated *n_repeats* times, each time with a different random split but still maintaining the stratified class distribution. This improve the robustness of cross-validation by repeating the process multiple times and averaging the results, hence obtain a more reliable performance measure. The code is as follows:

```
# Configure cross-validation
cv = RepeatedStratifiedKFold(n_splits=5, n_repeats=3, random_state=42)
```

```
# Custom scoring function
scoring = {
    'accuracy': 'accuracy',
    'precision': make_scorer(precision_score),
    'recall': make_scorer(recall_score),
    'f1': make_scorer(f1_score),
    'roc_auc': make_scorer(roc_auc_score)}
```

Next, a function is defined for **Training** a model. The model name and its set of hyperparameters are passed into a function. Grid search is then performed with cross validation (defined above). Inside this function, MLFlow is used to record the model name, its hyperparameters, and its scores. It also saves the best model so far into a pickle file along with its best hyperparameters .

```
def train_classifier(clf_name, clf_info):
    with mlflow.start_run(run_name=clf_name):
        model = clf_info['model']
        params = clf_info['params']
        start_time = time.time()
        grid_search = GridSearchCV(estimator=model, param_grid=params, cv=cv, scoring=scoring, refit='accuracy', n_jobs=1,
                                   verbose=10)
        grid_search.fit(X_train_resampled, y_train_resampled)
        end_time = time.time()
        duration = end_time - start_time

        mlflow.log_param("best_params", grid_search.best_params_)
        mlflow.log_metric("best_score", grid_search.best_score_)
        mlflow.log_metric("training_duration", duration)
        print(f"Training {clf_name}")
        print(f"Training duration: {duration:.2f} seconds")
        print("Best Params : {}".format(grid_search.best_params_))
        best_model = grid_search.best_estimator_
        mlflow.sklearn.log_model(best_model, "model")

        with open(f"best_model_{clf_name}.pkl", 'wb') as f:
            pickle.dump(best_model, f)

        mean_accuracy = grid_search.cv_results_['mean_test_accuracy'][grid_search.best_index_]
        mean_precision = grid_search.cv_results_['mean_test_precision'][grid_search.best_index_]
        mean_recall = grid_search.cv_results_['mean_test_recall'][grid_search.best_index_]
        mean_f1 = grid_search.cv_results_['mean_test_f1'][grid_search.best_index_]
        mean_roc_auc = grid_search.cv_results_['mean_test_roc_auc'][grid_search.best_index_]

        print(f"Mean training accuracy: {mean_accuracy}")
        print(f"Mean training precision: {mean_precision}")
        print(f"Mean training recall: {mean_recall}")
        print(f"Mean training F1-score: {mean_f1}")
        print(f"Mean training ROC-AUC score: {mean_roc_auc}")
        print()

        mlflow.log_metric("mean_train_accuracy", mean_accuracy)
        mlflow.log_metric("mean_train_precision", mean_precision)
        mlflow.log_metric("mean_train_recall", mean_recall)
        mlflow.log_metric("mean_train_f1", mean_f1)
        mlflow.log_metric("mean_train_roc_auc", mean_roc_auc)
```

To train a model, below is the code to use. This code trains the first classifier (index 0) which is Logistic Regression. To train another classifier, just change the index.

```
clf_name = list(classifiers.keys())[0]
clf_info = classifiers[clf_name]
train_classifier(clf_name, clf_info)
```

The function to **Test** a model (not applicable to Keras model) is as follows:

```
def test_classifier(clf_name):
    with open(f'best_model_{clf_name}.pkl', 'rb') as f:
        best_model = pickle.load(f)

    y_pred = best_model.predict(X_test)

    accuracy = accuracy_score(y_test, y_pred)
    precision = precision_score(y_test, y_pred)
    recall = recall_score(y_test, y_pred)
    f1 = f1_score(y_test, y_pred)
    roc_auc = roc_auc_score(y_test, y_pred)

    print(f"Testing {clf_name}")
    print(f"Test accuracy: {accuracy}")
    print(f"Test precision: {precision}")
    print(f"Test recall: {recall}")
    print(f"Test F1-score: {f1}")
    print(f"Test ROC-AUC score: {roc_auc}")
    print()
```

The function to test the Keras classifier is:

```
def test_keras_classifier(best_model):
    y_pred = best_model.predict(X_test)

    accuracy = accuracy_score(y_test, y_pred)
    precision = precision_score(y_test, y_pred)
    recall = recall_score(y_test, y_pred)
    f1 = f1_score(y_test, y_pred)
    roc_auc = roc_auc_score(y_test, y_pred)

    print(f"Testing Keras")
    print(f"Test accuracy: {accuracy}")
    print(f"Test precision: {precision}")
    print(f"Test recall: {recall}")
    print(f"Test F1-score: {f1}")
    print(f"Test ROC-AUC score: {roc_auc}")
    print()
```

After all the 6 classifiers have been trained, let's obtain their test scores:

```
clf_name = list(classifiers.keys())[0]
test_classifier(clf_name)
```

a) Testing LogisticRegression

Test accuracy: 0.72

Test precision: 0.5277777777777778

Test recall: 0.6333333333333333

Test F1-score: 0.5757575757575758

Test ROC-AUC score: 0.6952380952380952

```
clf_name = list(classifiers.keys())[1]
test_classifier(clf_name)
```

b) Testing DecisionTree

Test accuracy: 0.665

Test precision: 0.4606741573033708

Test recall: 0.6833333333333333

Test F1-score: 0.5503355704697988

Test ROC-AUC score: 0.6702380952380952

```
clf_name = list(classifiers.keys())[2]
test_classifier(clf_name)
```

c) Testing RandomForest

Test accuracy: 0.705

Test precision: 0.5076923076923077

Test recall: 0.55

Test F1-score: 0.528

Test ROC-AUC score: 0.6607142857142857

```
clf_name = list(classifiers.keys())[3]
test_classifier(clf_name)
```

d) Testing GradientBoosting

Test accuracy: 0.73

Test precision: 0.5441176470588235

Test recall: 0.6166666666666667

Test F1-score: 0.578125

Test ROC-AUC score: 0.6976190476190477

```
clf_name = list(classifiers.keys())[4]
test_classifier(clf_name)
```

e) Testing MLP

Test accuracy: 0.745

Test precision: 0.5692307692307692

Test recall: 0.6166666666666667

Test F1-score: 0.5920000000000001

Test ROC-AUC score: 0.7083333333333334

f) Special Steps for the Keras Model

For the Keras Classifier, the steps to test the model are different. This is because if I turn the model into a pickle file and try to load into Spyder, there will be an error. Therefore, I have to recreate the model.

```
early_stopping = EarlyStopping(
    monitor='auc_2', # Monitor validation auc
    min_delta=0.01, # Minimum change to qualify as an improvement
    patience=5, # Number of epochs to wait for improvement
    verbose=1, # Print message when stopping is triggered
    mode='max', # Mode for the metric (maximizing accuracy)
    baseline=None, # No baseline in this example
    restore_best_weights=True # Restore best weights when stopping
)
best_keras_model = KerasClassifier(
```

```

model = create_keras_model,
batch_size = 32,
epochs = 50,
loss="binary_crossentropy",
model_dropout_rate = 0.3,
model_l2_reg = 0.01,
model_neurons1 = 8,
model_neurons2 = 8,
model_weight_constraint = 4.0,
optimizer="SGD",
optimizer_learning_rate = 0.01,
optimizer_momentum = 0.9,
metrics=[AUC()],
callbacks=[early_stopping])

```

Because I am starting from scratch, I have to fit the model.

```
best_keras_model.fit(X_train_resampled, y_train_resampled)
```

OUTPUT:

```

Epoch 1/50
36/36 [=====] - 1s 3ms/step - loss: 1.0478 - auc_2: 0.5560
Epoch 2/50
36/36 [=====] - 0s 2ms/step - loss: 0.8911 - auc_2: 0.6507
Epoch 3/50
36/36 [=====] - 0s 2ms/step - loss: 0.8372 - auc_2: 0.6860
Epoch 4/50
36/36 [=====] - 0s 2ms/step - loss: 0.7798 - auc_2: 0.7404
Epoch 5/50
36/36 [=====] - 0s 2ms/step - loss: 0.7395 - auc_2: 0.7706
Epoch 6/50
36/36 [=====] - 0s 3ms/step - loss: 0.6912 - auc_2: 0.7936
Epoch 7/50
36/36 [=====] - 0s 3ms/step - loss: 0.6515 - auc_2: 0.8038
Epoch 8/50
36/36 [=====] - 0s 3ms/step - loss: 0.6268 - auc_2: 0.8081
Epoch 9/50
36/36 [=====] - 0s 6ms/step - loss: 0.6305 - auc_2: 0.8092
Epoch 10/50
36/36 [=====] - 0s 7ms/step - loss: 0.5999 - auc_2: 0.8210
Epoch 11/50
36/36 [=====] - 0s 7ms/step - loss: 0.5864 - auc_2: 0.8214
Epoch 12/50
36/36 [=====] - 0s 7ms/step - loss: 0.5829 - auc_2: 0.8168
Epoch 13/50
36/36 [=====] - 0s 5ms/step - loss: 0.5719 - auc_2: 0.8274
Epoch 14/50
36/36 [=====] - 0s 2ms/step - loss: 0.5676 - auc_2: 0.8221
Epoch 15/50
36/36 [=====] - 0s 7ms/step - loss: 0.5367 - auc_2: 0.8396
Epoch 16/50
36/36 [=====] - 0s 8ms/step - loss: 0.5353 - auc_2: 0.8381
Epoch 17/50
36/36 [=====] - 0s 9ms/step - loss: 0.5323 - auc_2: 0.8411
Epoch 18/50
36/36 [=====] - 0s 8ms/step - loss: 0.5381 - auc_2: 0.8421
Epoch 19/50
36/36 [=====] - 0s 6ms/step - loss: 0.5158 - auc_2: 0.8486
Epoch 20/50
31/36 [======>....] - ETA: 0s - loss: 0.5303 - auc_2: 0.8374Restoring model weights from the end of
the best epoch: 15.
36/36 [=====] - 0s 7ms/step - loss: 0.5285 - auc_2: 0.8385
Epoch 20: early stopping

```

To obtain the performance metrics results of the model, run the code below:

```
test_keras_classifier(best_keras_model)
```

Testing Keras

Test accuracy: 0.705
 Test precision: 0.5057471264367817
 Test recall: 0.7333333333333333
 Test F1-score: 0.598639455782313
 Test ROC-AUC score: 0.7130952380952381

To save the Keras model, use the save method and the model attribute:

```
best_keras_model.model_.save('Keras_Best_Model')
```

```
INFO:tensorflow:Assets written to: Keras_Best_Model\assets
INFO:tensorflow:Assets written to: Keras_Best_Model\assets
```

To load the saved Keras model:

```
from keras.models import load_model
keras2 = load_model('Keras_Best_Model')
scikeras2 = KerasClassifier(
    model = keras2,
    batch_size = 32,
    epochs = 50,
    loss="binary_crossentropy",
    optimizer="SGD",
    optimizer_learning_rate = 0.01,
    optimizer_momentum = 0.9,
    metrics=[AUC()],
    callbacks=[early_stopping])
```

After creating the scikeras model, I have to initialize it so that it can be used to make predictions:

```
scikeras2.initialize(X_train_resampled, y_train_resampled)
test_keras_classifier(scikeras2)
```

Testing Keras

Test accuracy: 0.705
 Test precision: 0.5057471264367817
 Test recall: 0.7333333333333333
 Test F1-score: 0.598639455782313
 Test ROC-AUC score: 0.7130952380952381

20. Select the Best Model

To determine the best model to be deployed, I will pay more attention to the Recall and ROC-AUC scores.

Recall is important for Loan Default because the financial cost of False Negatives (failing to identify a defaulter) is very high. Therefore, False Negatives should be minimized which means a high Recall score.

AUC-ROC is also somewhat important because it helps in understanding how well the model can separate defaulters from non-defaulters, although the financial impact of failing

to identify a defaulter is much higher than that of identifying non-defaulters as defaulters (False Positives).

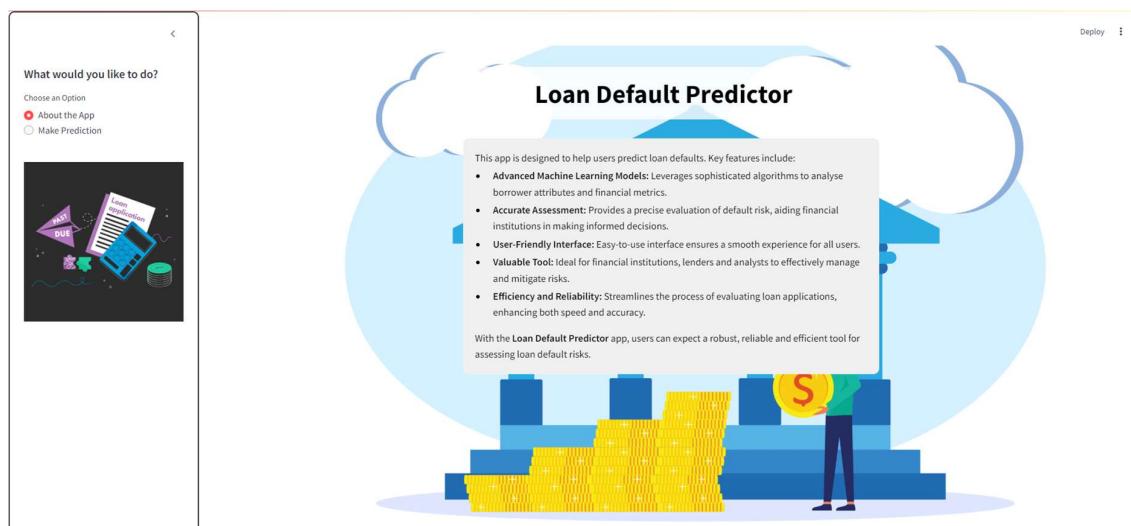
On the training dataset, all the models performed well. However, on the testing dataset, their performance generally dropped. Keras Classifier has the highest recall of 73.33% and the highest AUC-ROC score of 71.31%. MLP Classifier has the second highest AUC-ROC score of 70.83% and the second lowest recall of 61.67%.

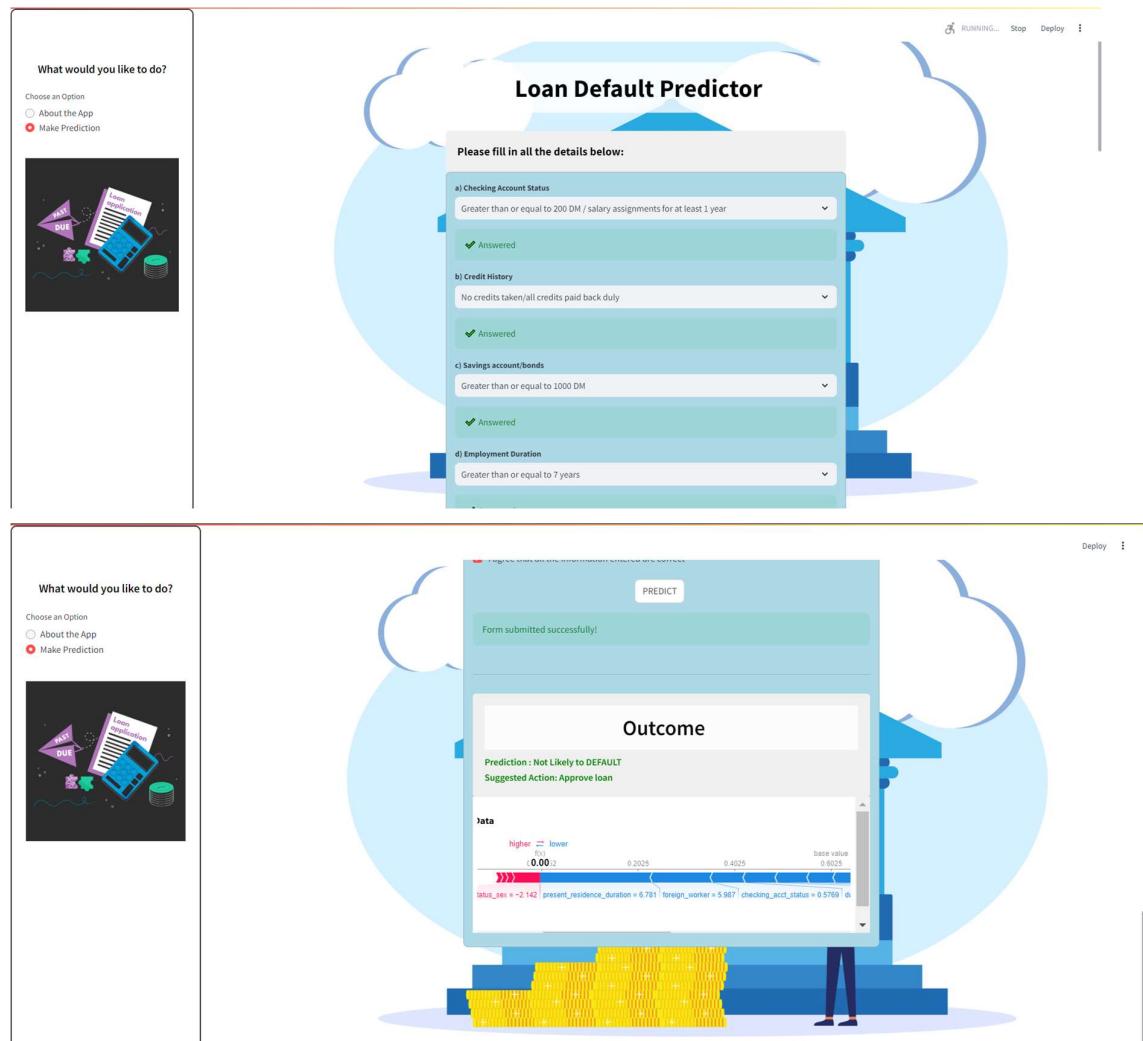
I select the Keras Classifier as the best model because it has the highest Recall, the most important metric for this case study. Another reason for selecting the Keras Classifier because it has the highest AUC-ROC score, which is the second most important metric.

5.3.2 User Interface & API Integration Implementation

The user interface (UI) for the Human Data Scientist approach, which allows an end user to enter loan information, is almost identical to that for AutoML so I will not describe it again. The API backend logic is a little different. The source code for the Streamlit and FastAPI components can be found in “uiHuman.py” and “fastapiHuman.py” respectively.

a) Actual User Interface





Snippets 5.54

The pseudo code for the FastAPI backend component is as follows:

b) FastAPI Backend Logic

- Import necessary modules (FastAPI, Pydantic, Scikit-learn, TensorFlow, SHAP, Pandas, Numpy, etc.).
- Create a FastAPI application instance.
- Load the best model (TensorFlow Keras Classifier) using a custom function `configure_model()`.
- Load the StandardScaler object from a file (`human_standard_scaler.pkl`).
- Define a data model `LoanApplication` using a schema with attributes like `checking_acct_status`, `credit_history`, `employment_duration`, etc.
- Define label encoding mappings for categorical variables (e.g., `checking_acct_status`, `credit_history`).

- Define a function `apply_label_encoding()` that replaces categorical values with their encoded labels using the provided mappings.
- When a POST request is sent to `/predict`, receive the loan application data.
- Convert the input data into a Pandas DataFrame.
- Define preprocessing steps using `ColumnTransformer` and `FunctionTransformer` to apply label encoding and transformations to the input data.
- Apply the preprocessing to transform the input data.
- Use the previously loaded `StandardScaler` object to scale the preprocessed data.
- Use the `best_model` to make predictions on the scaled data.
- Return the prediction as a response.
- Process Endpoint: `/show-shap`
- Load resampled training data from a CSV file (`X_train_resampled.csv`).
- Initialize the SHAP explainer using the trained model and training data.
- Calculate SHAP values for the scaled input data.
- Generate and save a SHAP force plot to an HTML file.
- Return the saved SHAP plot as an HTML file response.

5.3.3 System Integration Testing

ID	Test Objective	Test Data	Precondition	Steps	Expected Result	Actual Result	Status
1	View the Application	None	The Frontend and Backend servers must be running.	1. Open conda prompt & navigate to the directory containing streamlit script using <code>cd</code> 2. Enter the command “ <code>streamlit run <filename></code> ” on conda prompt 3. Wait for a message to be displayed on conda prompt 4. View Streamlit app on the browser 5. Open another conda prompt & navigate to the directory	A message is displayed on the conda prompt stating that the Streamlit app can be viewed on the browser. The app can be viewed on the browser. A message is displayed on the conda prompt stating that the model has been built and that the FastAPI server has started.	A message is displayed on the conda prompt stating that the Streamlit app can be viewed on the browser. The app can be viewed on the browser. A message is displayed on the conda prompt stating that the model has been built and that the FastAPI server has started.	Pass

				containing fastapi script using cd 6. Enter the command “uvicorn <filename>:app – reload” 7. Once there is a message saying the server has started, you can make predictions on the application.			
2	To navigate to the “Make Prediction” page	None	Must be on the “About the App” page	1. Click on the “Make Prediction” radio button	The user is directed to the “Make Prediction” page	The user is directed to the “Make Prediction” page	Pass
3	To navigate to the “About the App” page	None	Must be on the “Make Prediction” page	1. Click on the “About the App” radio button	The user is directed to the “About the App” page	The user is directed to the “About the App” page	Pass
4	To check whether the form will be submitted if no value is entered but the check box is ticked	None	Must be on the “Make Prediction” page and the input fields are blank	1. Scroll to the bottom of the page and tick the check box 2. Click the ‘Predict’ button	The page displays an error message asking the user to answer all the input fields and an error message at each input field.	The page displays an error message asking the user to answer all the input fields and an error message at each input field.	Pass
5	To check whether the form will be submitted if all input fields are entered but the check box is not ticked	1. Checking Account Status = No checking account 2. Credit History = Delay in paying off in the past 3. Savings account/bonds = Less than 100 DM 4. Employment Duration = Unemployed 5. Personal Status and Sex = Male: single	All input fields must be answered but the check box is not ticked. Must be on the “Make Prediction” page	1. Scroll to the bottom of the page and click the ‘Predict’ button	The page displays an error message asking the user to confirm the details before submitting the form.	The page displays an error message asking the user to confirm the details before submitting the form.	Pass

		6. Other Instalment Plans = Bank 7. Housing = Rent 8. Has Telephone? = None 9. Duration = Long term 10. Purpose = Car (new) 11. Property = Unknown / no property 12. Present Residence Duration = 1.00 13. Is Foreign Worker? = Yes 14. Existing Credits at Bank = 2.00				
6	To predict that a defaulter will default	1. Checking Account Status = No checking account 2. Credit History = Delay in paying off in the past 3. Savings account/bonds = Less than 100 DM 4. Employment Duration = Unemployed 5. Personal Status and Sex = Male: single 6. Other Instalment Plans = Bank 7. Housing = Rent 8. Has Telephone? = None	Must be on the “Make Prediction” page	1. Enter the test data into the input fields 2. Scroll to the bottom of the page and tick the check box. 3. Click the ‘Predict’ button	The page’s response is that the person is likely to default and the suggested action is not to approve the loan. A SHAP force plot will be displayed after the prediction.	The page’s response is that the person is likely to default and the suggested action is not to approve the loan. A SHAP force plot will be displayed after the prediction. Pass

		9. Duration = Long term 10. Purpose = Car (new) 11. Property = Unknown / no property 12. Present Residence Duration = 1.00 13. Is Foreign Worker? = Yes 14. Existing Credits at Bank = 2.00					
7	To predict that a defaulter will default	1. Checking Account Status = Less than 0 DM 2. Credit History = Delay in paying off in the past 3. Savings account/bonds = Less than 100 DM 4. Employment Duration = Less than 1 year 5. Personal Status and Sex = Female: single 6. Other Instalment Plans = None 7. Housing = Rent 8. Has Telephone? = Yes 9. Duration = Long term 10. Purpose = Car (used)	Must be on the “Make Prediction” page	1. Enter the test data into the input fields 2. Scroll to the bottom of the page and tick the check box. 3. Click the ‘Predict’ button	The page’s response is that the person is likely to default and the suggested action is not to approve the loan. A SHAP force plot will be displayed after the prediction.	The page’s response is that the person is likely to default and the suggested action is not to approve the loan. A SHAP force plot will be displayed after the prediction.	Pass

		11. Property = Unknown / no property 12. Present Residence Duration = 2.00 13. Is Foreign Worker? = No 14. Existing Credits at Bank = 1.00					
8	To predict that a non-defaulter will not default	1. Checking Account Status = Greater than or equal to 200 DM / salary assignments for at least 1 year 2. Credit History = All credits at this bank paid back duly 3. Savings account/bonds = Greater than or equal to 1000 DM 4. Employment Duration = Greater than or equal to 7 years 5. Personal Status and Sex = Female: single 6. Other Instalment Plans = None 7. Housing = Rent 8. Has Telephone? = Yes 9. Duration = Medium term 10. Purpose = Domestic appliances 11. Property = Car or other	Must be on the “Make Prediction” page	1. Enter the test data into the input fields 2. Scroll to the bottom of the page and tick the check box. 3. Click the ‘Predict’ button	The page’s response is that the person is not likely to default and the suggested action is to approve the loan. A SHAP force plot will be displayed after the prediction.	The page’s response is that the person is not likely to default and the suggested action is to approve the loan. A SHAP force plot will be displayed after the prediction.	Pass

		12. Present Residence Duration = 4.00 13. Is Foreign Worker? = No 14. Existing Credits at Bank = 1.00					
9	To predict that a non-defaulter will not default	1. Checking Account Status = Greater than or equal to 200 DM / salary assignments for at least 1 year 2. Credit History = Existing credits paid back duly till now 3. Savings account/bonds = Greater than or equal to 1000 DM 4. Employment Duration = Greater than or equal to 4 but less than 7 years 5. Personal Status and Sex = Male; married/widowed 6. Other Instalment Plans = None 7. Housing = Own 8. Has Telephone? = Yes 9. Duration = Short term 10. Purpose = Car (used) 11. Property = Car or other 12. Present Residence Duration = 5.00	Must be on the “Make Prediction” page	1. Enter the test data into the input fields 2. Scroll to the bottom of the page and tick the check box. 3. Click the ‘Predict’ button	The page’s response is that the person is not likely to default and the suggested action is to approve the loan. A SHAP force plot will be displayed after the prediction.	The page’s response is that the person is not likely to default and the suggested action is to approve the loan. A SHAP force plot will be displayed after the prediction.	Pass

		13. Is Foreign Worker? = No 14. Existing Credits at Bank = 1.00					
--	--	--	--	--	--	--	--

Table 5.3. Test Cases Conducted for AutoML Approach

Chapter 6

System Deployment

6 System Deployment

This chapter focuses on the deployment of the best model selected from each of the three approaches: Human Data Scientist, AutoML, and Generative AI. The 3 applications/systems are deployed locally, using Streamlit as the client and FastAPI as the server within a client-server methodology.

6.1 Overall View of Deployment

This section covers the resources/components needed for deployment and provides an overall view of how the applications/systems will be deployed.

6.1.1 Resources/Components Needed

For each of the 3 approaches, there needs to be 1 **Streamlit .py** file, 1 **FastAPI .py** file, 1 file that represents the **Machine Learning model** and 1 file that represents the **Scaler object**. Since this is a local deployment, only **1 laptop or desktop computer** will be needed for the demonstration. For each approach, **2 windows of Anaconda Prompt** Command Line Interface (CLI) are needed. **1 window** will be used to launch the Streamlit server and open the application in the default web browser. The **other window** will be used to start the FastAPI application using the Uvicorn ASGI server with auto-reloading enabled to automatically update the server on code changes.

6.1.2 How the Components Work Together

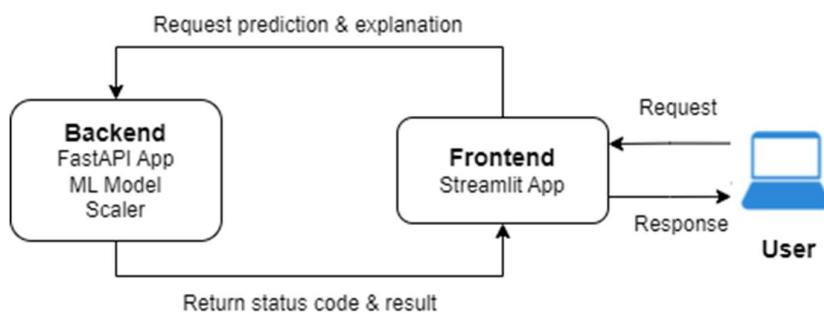


Figure 6.1. Deployment

Figure 6.1 illustrates how each of the 3 applications/systems will be deployed. In this setup, Streamlit provides a user interface for users to input data, submit it and view predictions and explanations. On the other hand, FastAPI receives data from the frontend, processes it and returns predictions & explanations. The Scaler object performs feature scaling on the user's input data and the ML Model object performs the actual prediction on the scaled data. If everything runs smoothly, the prediction and explanation will be returned to the frontend.

Given that there is a possibility of internal server error, the FastAPI app will also return the status code of the HTTP response to indicate the result of the server's attempt to handle the request. If

the status code is 200, indicating a successful request, the response data will be processed and vice versa. In my case, if the status code is 200, I will process the prediction returned and request the explanation plot from the FastAPI app to be displayed on Streamlit. However, if the status code is not 200, I will not process the prediction returned, and I will not request the explanation plot from the FastAPI app. The Streamlit app will also display a message saying that the server is down.

6.2 Reasons for Local Deployment

There are several reasons for deploying locally. Firstly, it allows me to develop and test my applications in a controlled environment, identify and fix bugs and ensure all features work as intended. Moreover, since this project is for a Final Year Project, this approach is convenient for demonstrating my work and ensuring reliable access without relying on external servers. Lastly, handling IP addresses and network configurations can be complex. Local deployment helps me manage these aspects more easily. Managing IP addresses involves configuring my application to listen on the correct address and port, handling firewall settings, and dealing with network security—challenges that can become more complex when transitioning from a local setup to a public or more intricate network environment.

6.3 How Communication between Streamlit and FastAPI is Achieved

This consists of a series of steps to get the prediction and explanation plot from the FastAPI app.

6.3.1 Initialising a FastAPI Application Instance (FastAPI)

The app variable represents an instance of a FastAPI application. This instance is used to define routes, handle requests and configure various aspects of my API.

```
app = FastAPI()
```

Figure 6.2. For Initialising FastAPI

6.3.2 Requesting for Prediction (Streamlit)

This task involves sending data to a local server endpoint (/predict) and receiving a prediction and a status code from that server. The code snippet below is from uiHuman.py.

```
def get_prediction(data):
    url = 'http://localhost:8000/predict'
    response = requests.post(url, json=data)
    print(response.json())
    #if response.ok():
    #    st.write("OK")
    #else:
    #    st.write("Something is not right")
    return response.json(), response.status_code
```

Figure 6.3. Requesting Prediction

6.3.3 Handling Prediction Request (FastAPI)

The `@app.post('/predict')` decorator from FastAPI specifies that the function `predict_loan_default` should handle HTTP POST requests sent to the `/predict` endpoint. This endpoint is designed to handle requests where users submit loan applications and receive predictions about loan defaults. The function finally returns a JSON response, which holds the prediction result, to the client that made the POST request. The code snippet below is from uiHuman.py.

```
@app.post('/predict')
def predict_loan_default(application: LoanApplication):
    data = [
        'checking_acct_status': [application.checking_acct_status],
        'credit_history': [application.credit_history],
        'savings_acct_bonds': [application.savings_acct_bonds],
        'employment_duration': [application.employment_duration],
        'personal_status_sex': [application.personal_status_sex],
        'other_installment_plans': [application.other_installment_plans],
        'housing': [application.housing],
        'telephone': [application.telephone],
        'duration_category': [application.duration_category],
        'purpose': [application.purpose],
        'property': [application.property],
        'present_residence_duration': [application.present_residence_duration],
        'foreign_worker': [application.foreign_worker],
        'existing_credits_at_bank': [application.existing_credits_at_bank]
    ]

    # Convert to DataFrame
    input_df = pd.DataFrame(data)
    print(input_df)
    # Create the ColumnTransformer
    preprocessing_steps = [
        ('checking_acct_status', FunctionTransformer(lambda x: apply_label_encoding(x, label_mappings['checking_acct_status']), validate=False), ['checking_acct_status']),
        ('credit_history', FunctionTransformer(lambda x: apply_label_encoding(x, label_mappings['credit_history']), validate=False), ['credit_history']),
        ('savings_acct_bonds', FunctionTransformer(lambda x: apply_label_encoding(x, label_mappings['savings_acct_bonds']), validate=False), ['savings_acct_bonds']),
        ('employment_duration', FunctionTransformer(lambda x: apply_label_encoding(x, label_mappings['employment_duration']), validate=False), ['employment_duration']),
        ('personal_status_sex', FunctionTransformer(lambda x: apply_label_encoding(x, label_mappings['personal_status_sex']), validate=False), ['personal_status_sex']),
        ('other_installment_plans', FunctionTransformer(lambda x: apply_label_encoding(x, label_mappings['other_installment_plans']), validate=False), ['other_installment_plans']),
        ('housing', FunctionTransformer(lambda x: apply_label_encoding(x, label_mappings['housing']), validate=False), ['housing']),
        ('telephone', FunctionTransformer(lambda x: apply_label_encoding(x, label_mappings['telephone']), validate=False), ['telephone']),
        ('duration_category', FunctionTransformer(lambda x: apply_label_encoding(x, label_mappings['duration_category']), validate=False), ['duration_category']),
        ('purpose', FunctionTransformer(lambda x: apply_label_encoding(x, label_mappings['purpose']), validate=False), ['purpose']),
        ('property', FunctionTransformer(lambda x: apply_label_encoding(x, label_mappings['property']), validate=False), ['property']),
        ('present_residence_duration', FunctionTransformer(lambda x: x, validate=False), ['present_residence_duration']),
        ('foreign_worker', FunctionTransformer(lambda x: apply_label_encoding(x, label_mappings['foreign_worker']), validate=False), ['foreign_worker']),
        ('existing_credits_at_bank', FunctionTransformer(lambda x: x, validate=False), ['existing_credits_at_bank'])
    ]
    preprocessor = ColumnTransformer(transformers=preprocessing_steps)

    # Preprocess the data
    preprocessed_data = preprocessor.fit_transform(input_df)
    print(len(preprocessed_data))
    # Convert preprocessed data to DataFrame
    preprocessed_df = pd.DataFrame(preprocessed_data, columns=['checking_acct_status', 'credit_history', 'savings_acct_bonds', 'employment_duration', 'personal_status_sex', 'other_installment_plans'])
    for column in preprocessed_df.columns:
        print(f'{column} in column {len(column)}')
        for value in preprocessed_df[column]:
            print(value)
    print(preprocessed_df.info())
    global scaled_df
    scaled_df = scaler_obj.transform(preprocessed_df)
    scaled_df = pd.DataFrame(scaled_df, columns=preprocessed_df.columns, index=preprocessed_df.index)
    print(scaled_df.info())
    global prediction
    prediction = best_model.predict(scaled_df)
    print(prediction.tolist())
    return {
        'prediction': prediction.tolist()
    }
```

Figure 6.4. How Prediction is made

6.3.4 Requesting for Explanation Plot (Streamlit)

There are a few ways to retrieve the explanation plot from the FastAPI server. Here, I discuss one method that is used in the AutoML approach. On the AutoML streamlit script, the `get_shap` function is designed to request a SHAP summary plot from the local FastAPI server and display it in a Streamlit application. `response.content` contains the raw byte data of the response, which is expected to be an image. `BytesIO(response.content)` creates an in-memory binary stream from the byte data. `Image.open()` from the PIL (Pillow) library opens this binary stream as an image. `st.image(image, caption='SHAP Summary Plot')` is used to display the image in the web application. The code snippet below is from `uiAutoML.py`.

```
def get_shap():
    url = 'http://localhost:8000/show-shap'
    response = requests.post(url)
    if response.status_code == 200:
        # Display the image
        image = Image.open(BytesIO(response.content))
        st.image(image, caption='SHAP Summary Plot')
    else:
        st.write("Failed to fetch the SHAP summary plot.")
```

Figure 6.5. Requesting SHAP Plot

6.3.5 Returning Explanation Plot (FastAPI)

The `@app.post("/show-shap")` decorator indicates that the `show_shap` function will handle POST requests sent to the `/show-shap` endpoint. `buf = io.BytesIO()` creates an in-memory binary stream (BytesIO object) to store the plot image. `plt.savefig(buf, format="png")` saves the current figure to the BytesIO object in PNG format. `buf.seek(0)` moves the cursor of the BytesIO object to the beginning of the stream, so the image data can be read from the start. `plt.close()` closes the current figure to free up resources and avoid memory leaks. `return Response(content=buf.getvalue(), media_type="image/png")` returns the contents of the BytesIO object (`buf.getvalue()`) as the HTTP response and indicates that the response content is a PNG image. The code snippet is from `fastapiAutoMLv2.py`.

```
@app.post("/show-shap")
async def show_shap():
    plt.figure()
    model.shap_explain_row_plot(user_input, row_index=0)
    plt.title("Explanation for Prediction", fontsize=16)
    plt.xlabel("SHAP Value", fontsize=14)
    plt.ylabel("Features", fontsize=14)
    plt.xticks(fontsize=10)
    plt.yticks(fontsize=10)
    # Save plot to a BytesIO object

    buf = io.BytesIO()
    plt.savefig(buf, format="png")
    buf.seek(0)
    plt.close()

    return Response(content=buf.getvalue(), media_type="image/png")
```

Figure 6.6. Returning SHAP Plot

6.4 Deployment Procedures

The following procedures are applicable for all 3 approaches.

- 1) Open 1 Anaconda prompt window, navigate to the directory that the FastAPI app is in and start the FastAPI server. Below is a sample:

```
(ForChatGPT) C:\Users\ryank>cd C:\Users\ryank\Desktop\FYP\FYP Deployment\Generative AI Deployment
(ForChatGPT) C:\Users\ryank\Desktop\FYP\FYP Deployment\Generative AI Deployment>uvicorn fastapiGenAIv3:app --reload
```

- 2) Check that the FastAPI app is working properly by looking at the messages displayed in the CLI.

```
-[32mINFO-[0m: Will watch for changes in these directories: ['C:\\Users\\ryank\\Desktop\\FYP\\FYP Deployment\\Generative AI Deployment']
-[32mINFO-[0m: Uvicorn running on <[1mhttp://127.0.0.1:8000-</m (Press CTRL+C to quit)
-[32mINFO-[0m: Started reloader process <[36m-[1m3068-</m] using <[36m-[1mStatReload-</m
INFO: Started server process [2436]
INFO: Waiting for application startup.
INFO: Application startup complete.
```

- 3) Open another Anaconda prompt window, navigate to the directory that the Streamlit app is in and start the Streamlit server. Below is a sample:

```
(ForChatGPT) C:\Users\ryank>cd C:\Users\ryank\Desktop\FYP\FYP Deployment\Generative AI Deployment
(ForChatGPT) C:\Users\ryank\Desktop\FYP\FYP Deployment\Generative AI Deployment>streamlit run uiGenAIv4.py
```

- 4) Check the browser to ensure the Streamlit app is displayed properly.

```
You can now view your Streamlit app in your browser.

Local URL: http://localhost:8501
Network URL: http://192.168.100.7:8501
```

- 5) To stop using Streamlit, just close the CLI window. To stop FastAPI, use Ctrl+C in the CLI.

6.5 Challenges Faced during Deployment

1. Environmental Setup

I had to ensure that all necessary dependencies for both Streamlit and FastAPI are properly installed. Besides that, since I am using the same virtual environment for all the codes related to the Human Data Scientist and AutoML approaches, I had to ensure that the dependencies associated with all the packages installed were compatible with each other. Lastly, I had to make sure that both Streamlit and FastAPI are compatible with the Python version that I am using.

2. Communication between Streamlit and FastAPI

Streamlit and FastAPI are being run as separate services. Therefore, every time the applications are deployed, I had to ensure that FastAPI is running on a local server (e.g.,

<http://localhost:8000>) and that the Streamlit app correctly handles these requests. Note that Streamlit is the one calling the FastAPI endpoints directly using the requests library.

3. Testing and Debugging

Initially, I faced issues with processing the input values entered by the user and returning the model prediction and SHAP plot from the FastAPI app to the Streamlit app. To debug these issues, I had to ensure both Streamlit and FastAPI were running and to analyse the causes of the FastAPI internal server error by reading the error messages displayed in the command line interface. Moreover, I had to perform both frontend and backend testing to ensure that the user interface remains responsive, especially when interacting with the FastAPI backend.

4. Deployment of the KerasClassifier Model

There was not that much documentation on how to deploy the KerasClassifier model. Somehow, even if I could save the model as a pickle file, I could not load the model later on when using the Spyder application to write the FastAPI app script. Therefore, I had to use an unconventional method, which involves saving the Keras model using the save method and the model_.attribute. To load the Keras model into the FastAPI app, my FastAPI app had to call a function from a python script to load and initialize the Keras model so that it is able to make predictions. In a future enhancement, to serve the KerasClassifier model I will use TensorFlow Serving because it is supposed to be more robust and is tightly integrated with TensorFlow Keras models.

Chapter 7

Discussions and Conclusion

7 Discussions and Conclusion

7.1 Summary

The goal of this project is to determine whether or not Generative AI and AutoML can replace the Human Data Scientist. The proposed solution is to perform a comparison study whereby the 3 approaches (Generative AI, AutoML, Human Data Scientists) are used to develop a Credit Risk Assessment (a.k.a. Loan Default Prediction) model using a realistic dataset (German Credit dataset). The tools selected to represent Generative AI and AutoML are ChatGPT Data Analyst (included in the paid ChatGPT Plus account) and H2O AutoML respectively. From its name, you can tell that OpenAI is positioning ChatGPT Data Analyst as a replacement for the human Data Analyst. This product was formerly named “Advanced Data Analysis”. We all know how capable the base ChatGPT base product is. H2O AutoML was chosen because it is the most feature-rich among the open source AutoML products. Unfortunately, I am unable to include the proprietary DataRobot in our study as I don’t have a license for it. I chose CRISP-DM (Cross-Industry Standard Process for Data Mining) as the data science process methodology because it provides a structured approach to the development and execution of data science projects. It is flexible and generally applicable across industries. Moreover, to ensure that the models created have substance and are of good quality, all 3 approaches are required to implement a set of Machine Learning Best Practices.

7.2 Achievements

My overall project objective is to determine whether or not Generative AI and AutoML can replace the Human Data Scientist. But before I can achieve this final objective, first of all I need to study the tools which are being used to represent Generative AI and AutoML as well as various Python frameworks used by the Human Data Scientist. Secondly, after I have attained a sufficient level of competency with these tools, only then can I evaluate the strengths and weaknesses of each of these tools in developing and implementing ML models for the selected case study. Finally, I will compare the strengths and weaknesses of Generative AI and AutoML to those of the Human Data Scientist and then come to a conclusion as to whether or not either of these technologies can replace the Human Data Scientist. Therefore, below I have broken my overall project objective into its constituent steps and I will evaluate the achievement of each sub-goal individually.

Project Objective 1: To *explore and study how to use the tools* that represent the 3 approaches (Human Data Scientist, AutoML, Generative AI) of machine learning.

For implementation, the Human Data Scientist (in our situation) uses Python, Scikit-Learn, TensorFlow Keras and other Python frameworks (e.g. Pandas, NumPy, Matplotlib, Seaborn, MLFlow, Imblearn, SHAP) to develop and implement the case study. In the process of implementing the ML models for the selected case study, I managed to expand my knowledge and sharpen my skills in Python, Scikit-Learn, TensorFlow Keras, and the various supporting frameworks. Before I started this project, I had no experience of using an AutoML tool at all. My impression, mainly from its name, was that AutoML would automate almost everything and that the human operating it would merely need to “push a few buttons”. How wrong that impression was! The amount of automation in AutoML was much less than what I thought. Major areas like data preprocessing and feature engineering for the most part still require manual coding. Even if I had used the proprietary DataRobot (which is mostly GUI-driven) instead of H2O AutoML, I expect in some situations (e.g. data cleaning), it would still be quite tedious despite the use of GUI. There are still decisions to be made unless AutoML already made default decisions for you, which is not ideal because the defaults may not be appropriate for your situation. That is why I opted to use the programmatic approach rather than the GUI in H2O AutoML. Although I have to learn some of H2O AutoML’s Python API commands, this approach is more flexible and furthermore, it is reproducible. Although H2O AutoML has many features, I felt I have learnt its important core features and understand its nature and characteristics. For ChatGPT Data Analyst (CDA), it was a little different. Prior to using CDA, I had already used the ChatGPT base product for generating code. CDA with its Runtime Environment seem to promise to make life so much more convenient for the Data Analyst and Data Scientist. No longer do we need to: (1) run the code on our own computer and (2) upload results and visualizations for ChatGPT to interpret. CDA with its Runtime Environment is supposed to be able do everything itself i.e. run the code and immediately interpret the results without human intervention. For data preprocessing and descriptive statistics (EDA), the CDA runtime is capable of carrying out these types of processing. However, I discovered that when it comes to modelling, CDA was only capable of occasionally running the baseline models which use default hyperparameter values. When I asked it to run grid search, CDA runtime will return “Analyzed Error”. Even when this is scaled down to random grid search, CDA still cannot run the code. When I started using CDA, I first tried using a “minimal guidance” approach but it did not work out well as CDA would omit important ML tasks or perform tasks in the incorrect order. After that, I switched to a “guided” approach where I would explicitly tell CDA what to do. This guided approach worked out much better.

To summarize, in the process of developing and implementing the case study, I managed to attain a competent proficiency in using the tools (Python frameworks, AutoML, ChatGPT Data Analyst) used to develop and implement the case study. Obviously, there are still gaps in my

knowledge, for e.g. there are other useful Python frameworks that I would like to learn and additional features in H2O AutoML I did not have enough time to look into. As for ChatGPT Data Analyst, I would have liked to dig deeper into prompt engineering to see if that has significant impact on the quality of work produced by CDA.

Project Objective 2: To evaluate the strengths and weaknesses of each of the 3 approaches in developing, evaluating and implementing the selected case study, including machine learning best practices.

All 3 approaches (Human Data Scientist, AutoML, Generative AI) were able to implement all or most of the ML best practices but in some situations there are differences. In order to determine whether or not Generative AI or AutoML can replace the Human Data Scientist, we need to know their strengths and weaknesses. Below, I list out the strengths and weaknesses of each approach. These strengths and weaknesses were derived from my experiences during the implementation phase. Please refer to Chapter 5 (Implementation and Testing) for more details.

Approach 1: Generative AI (ChatGPT Data Analyst)

STRENGTHS

- CDA's greatest strength is its ability to rapidly generate new code and make code changes (in seconds most of the time).
- CDA is able to generate visualizations for EDA and results in short period of time.
- CDA is able to provide insights into dataset, visualizations, and results.
- CDA is to provide advice on ML Best Practices and other domain issues
- CDA generates documentation for the code it generated, increasing understandability.
- With the guidance of a Human Data Scientist, CDA is able to implement ML tasks in the correct order.

WEAKNESSES

- Without the guidance of a Human Data Scientist, CDA can perform ML tasks in the wrong order.
- Without the guidance of a Human Data Scientist, CDA can omit important ML tasks.
- When a script goes through several iterations of change, existing code can go missing (ChatGPT memory issue).
- When a code change is made, CDA may not make adjustments to all other code which are affected. Need to manually check.

- CDA is built for a special technical purpose and therefore does not have the general management and decision-making skills which are required to manage a ML project.
- The CDA runtime environment cannot run grid search or even random grid search. It may be able to run baseline versions of models but even this is not guaranteed. CDA recommended to run the code in a local environment or in Colab.
- The CDA runtime environment does not allow new packages to be installed in it. This is a severe restriction as many popular packages do not exist in the CDA runtime environment.

Approach 2: Human Data Scientist

STRENGTHS

- Better at human-related activities e.g. stakeholders engagement and collaboration
- A Data Scientist usually has organization-specific knowledge which AI and automation do not possess.
- A Data Scientist usually has deeper domain knowledge (domain expert)
- A Data Scientist can be more skilled than AI in a particular area if he/she is a subject expert
- A Data Scientist can supervise and validate Generative AI which may make mistakes
- A Data Scientist can intervene when CDA and AutoML cannot proceed (e.g. exceptions)
- A Data Scientist has management and decision-making skills which AI and automation do not possess.
- A Data Scientist's code does not have the reliability issue that CDA's code has

WEAKNESSES

- Human coding speed is several orders of magnitudes slower than that of ChatGPT
- Human Data Scientist's coding skills is not as extensive as that of ChatGPT
- Human Data Scientist's knowledge of ML may not be as broad as that of ChatGPT

Approach 3: AutoML (H2O)

STRENGTHS

- Model Selection is automated and simplified with Leaderboard
- Ensembles Models (stacked, best-in-family) are automatically selected
- Cross-Validation is automatically applied by default to reduce overfitting
- Built-in Model Explainability and Feature Importance
- Predefined parameters for Flow GUI make it easier for “Citizen Data Scientists”
- Results produced by AutoML are more extensive and detailed
- Scalable platform with support for big data and distributed, parallel, in-memory processing
- MLOps Monitoring platform and Automatic Documentation are available but proprietary

WEAKNESSES

- No support for Business Understanding phase (requirements, analysis)
- Little support for Exploratory Data Analysis (EDA). Need to be augmented with other visualization frameworks.
- Little automation for Data Preprocessing (programmatic)
- Web Flow GUI offers limited functionality and lack reproducibility
- No automation for easing deployment of models and integration with other applications

In summary, this project objective achieves the sub-goal of finding the strengths and weaknesses of each of the 3 approaches (Human Data Scientist, AutoML, Generative AI).

Project Objective 3: Based on the strengths and weaknesses of the 3 approaches, *provide a conclusion* as to whether or not Generative AI or AutoML can replace the Human Data Scientist.

With the advent of ChatGPT and in particular the “Data Analyst” plugin, it has been said that a business executive or manager does not need a Data Analyst to provide answers to analytics questions. All the business executive or manager need to do is issue prompts to ChatGPT Data Analyst (CDA) in natural language and get the results, visualizations, insights, and insights that he/she sought for. How much of this is true? If we are talking about Data Analysts who only deal with descriptive analytics, perhaps there is some substance to some of the claims made as CDA is capable of analysing data, generating visualizations from it, providing insights into the trends and patterns in the data, and making recommendations.

However, my research topic is about Generative AI and AutoML replacing Data Scientists. The term “Data Science” is often meant to cover Predictive Analytics (Machine Learning) as well as Descriptive Analytics. AutoML has often been said to democratize data science for non-technical people, meaning a person perhaps from e.g. the operations department of an organization can use AutoML to perform predictive analytics without much assistance. There is no need for a trained Data Scientist. So, can Generative AI or AutoML replace the Human Scientist? If not completely, will Generative AI or AutoML take away most of a Data Scientist’s role in a machine learning project? In the preceding objective (Project Objective 2), I listed out the strengths and weaknesses of Generative AI, AutoML and the Human Data Scientist. Let’s look at AutoML first. It is quite apparent that most AutoMLs have been designed for a particular role, which is to focus on certain technical aspects of ML. They are not meant to play any role in the business understanding phase of a ML project. Although H2O AutoML does not provide much automation for the data preprocessing phase, it is possible to automate them but most of these preprocessing tasks still require decisions to be taken (e.g. imputation methods, scaling methods, balancing methods, outlier handling methods, etc.). Most of the time, making the correct decision requires some training in statistics. It might be easy to just apply defaults, but the default may not always be the right choice and this can impact the performance of the model created. So, back to the question: “Can AutoML replace the Human Data Scientist?”. My answer is: “It depends”. Since AutoML tools can vary widely, I would say that an AutoML tool like DataRobot (which is feature-rich and GUI-driven) can make things easier for a non-technical person (citizen data scientist) to operate it. But even with a point-and-click user interface, the citizen data scientist will still need to have some statistical knowledge in order to make the right decisions. Later on, when the citizen data scientist gets into modelling, he/she may encounter further complexities. Take for example the Loan Default case study which is a binary classification problem. He/she may encounter this term “Decision Threshold” which has an impact on how the model makes predictions. Let’s say even after running grid search for a set of hyperparameters, the model performance is still not good enough. In this situation, the citizen data scientist will need to tune the hyperparameters as the defaults set by AutoML are not ideal. However, in order to choose which hyperparameters to use and what values to set them to, the citizen data scientist need to understand the concepts behind the hyperparameters which are usually statistical and mathematical in nature. Where does all this lead to? The answer is that it is not so easy to put a citizen data scientist to operate an AutoML. That person must be sufficiently well-trained in statistics in order to make good decisions at various points in the ML life cycle. In order to be able to create high-performing models, the citizen data scientist may need to acquire even DL (deep Learning) knowledge to be able to understand what hyperparameters such as *epoch* and *mini-batch size* mean. In addition, the citizen data scientist must also need to understand performance metrics and know which ones are more important

for the problem being solved. AutoML will no doubt use a default metric to rank the performances of the difference algorithms. But is this default metric the ideal one to use for the problem at hand? Similarly, for a binary classification problem like Loan Default, AutoML will likely default the Decision Threshold to 0.5 but maybe the organisation is risk averse and want to use a lower threshold so as to minimize false negatives (actual defaulters predicted as not) at the expense of having more false positives. This means priority is given to the Recall score. Given that AutoML does not really shield the user from technical matters (even with a GUI), maybe the question organizations want to know the answer to is: “Is it easier and cheaper to train a non-technical person to be a data scientist rather than hiring a real data scientist?” Even if a citizen data scientist is put in charge of AutoML, he/she will still need technical support especially in setup, troubleshooting and also deployment which requires software engineering skills. If a data scientist is in place, he/she may have all these additional skills. Lastly, since few organizations can afford to pay for DataRobot licenses, those organizations that cannot will have to resort to using Open Source AutoMLs. Many open source AutoMLs do not have a GUI and as for the ones that have do have, the GUI is limited and only certain features can only be accessed through the product’s API (this is case with H2O AutoML). This implication of this is that in order for the citizen data scientist to operate such AutoMLs, he/she will need to have a technical aptitude, basically be a programmer. Therefore, my answer to the question: “Can AutoML replace the Human Data Scientist?” is: *“It depends on the organization’s strategy and available resources”*.

Note that the arguments above also apply to Generative AI even though it is very different from AutoML. A novice will definitely feel more comfortable using a organised and structured tool like DataRobot. Generative AI, on the other hand, is free form. You can do anything you like and, as a result, land in more trouble if you not sufficiently skilled enough with the Gen AI tool and the subject matter. As mentioned earlier, if you don’t give ChatGPT sufficient guidance, it can perform ML tasks in the wrong order and miss out important ML tasks if you don’t explicitly tell it to perform them. Code generation reliability is also a big issue. In short, the current version of ChatGPT is still not sufficiently well trained in ML and it needs guidance and careful monitoring to make sure it does its job well. Therefore, *at this moment in time, Generative AI like ChatGPT cannot replace a Human Data Scientist* (this answer does not take into account the CDA runtime environment’s problems mentioned earlier). Having said that, I believe Generative AI like ChatGPT has tremendous potential. Once it has been properly pre-trained in creating ML models, it can be a threat to the Data Scientist’s job. Take for example, ChatGPT is used to create a binary classification model like Loan Default (like this project). If in the future, ChatGPT is sufficiently advanced such that it can use its domain and ML knowledge to make intelligent decisions like what metrics to use, what decision threshold to

use as default and which features are more important, then it certainly will challenge the expertise of the human data scientist. However, currently given the technical nature of ML, it will still be a challenge for a citizen data scientist to take up the role as coming up with high-performing models requires sufficient statistical, mathematical and algorithmic knowledge.

The way forward is a hybrid approach with the automation technology (Gen AI or AutoML) complementing the Human Data Scientist, with each approach focussing on the areas they are strong in. The human data scientist takes care of issues requiring human collaboration and contributes his/her domain and technical expertise. The high productivity of AI and automation taking over most of the coding tasks.

To summarize, the final objective of this project has been achieved as I have answered the question: “Can Generative AI or AutoML replace the Human Data Scientist?”

7.3 Contributions

1) Summary of the project’s contributions

While previous research explored the impacts of AutoML and Generative AI on data science, this study investigated the differences and synergy between these two technologies and human data scientists to enhance the data science process. It examined their individual strengths and weaknesses, how they complement each other, and how their combined use can improve data analysis, decision-making, and productivity. With Generative AI's recent advancements, such as the seamless integration of advanced multimodal capabilities with real-time adaptive learning in Gemini AI, this study is highly relevant in a rapidly evolving area.

2) Creativity and Innovativeness

The study explores a cutting-edge intersection of Automated Machine Learning (AutoML) and Generative AI, evaluating their potential to replace human data scientists. Being both innovative and timely, it addresses a critical question in the evolving field of data science. By systematically comparing the capabilities of these advanced technologies against the traditional roles of human data scientists, this study offers a unique perspective on how automation and AI are reshaping the landscape of data science. This research is creative in its integration of diverse AI methodologies to tackle a central question about the future of the profession.

3) Contribution of the Proposed Study

The primary contribution of the proposed study is its detailed analysis of AutoML and Generative AI technologies in the context of replacing the human data scientist. By investigating these systems' ability to replicate or even surpass the functions of human data scientists, the research provides valuable insights into their strengths and limitations. This analysis is useful for organizations considering the adoption of these technologies, as it offers a clear understanding of their practical capabilities and the extent to which they can automate data science tasks. The study's findings will inform both academic and industry perspectives on the role of human expertise in data science versus the capabilities of emerging technologies.

4) Necessity of the Proposed Study

The necessity of this study stems from the rapid advancements in Generative AI and Machine Learning technologies, which are transforming various industries, including data science. As AutoML and Generative AI tools become more sophisticated, there is an urgent need to evaluate their impact on the traditional role of data scientists. This evaluation will help organizations make informed decisions about integrating these technologies into their operations and will contribute to understanding the future role of human expertise in a data-driven world. The study addresses a gap in current research by providing a structured analysis of whether these technologies can genuinely replace human data scientists or if there are aspects of data science that remain uniquely human.

5) Marketability of the Study

The study's insights into these technologies' roles and benefits make it highly marketable, as it addresses the needs and interests of various stakeholders and enhances the accessibility and efficiency of data science practices. The following describes the significant value that the study's contributions offer across multiple stakeholders:

- **Aspiring and Practicing Data Scientists/Analysts:** AutoML, Generative AI, and MLOps enhance productivity by automating routine tasks, enabling data scientists to concentrate on higher-value activities like critical thinking and complex problem-solving. Proficiency in these technologies provides a competitive edge in the job market, making professionals more attractive to employers.
- **IT Management:** By understanding how Generative AI, AutoML, and MLOps can improve data science workflows, IT management can make informed decisions about resource allocation and integration. This knowledge leads to more efficient utilization

of data science staff, optimized workflows, reduced costs and a stronger competitive position.

- **Business Executives and Decision Makers:** Insight into how these technologies contribute to business objectives, such as increased efficiency and innovation, helps executives align technology initiatives with strategic goals and evaluate the potential return on investment, fostering informed decision-making.
- **Data Science Educators and Trainers:** Updating curricula to include Generative AI, AutoML, and MLOps ensures that students are equipped with cutting-edge knowledge and skills. This preparation aligns educational outcomes with industry demands, enhancing students' readiness for the workforce.
- **Business End Users and the Community:** By simplifying data science tools and processes, AutoML and Generative AI democratize access to data, allowing a broader audience to engage in data-driven decision-making. This increased accessibility empowers individuals to make informed choices in personal, professional, and civic areas, promoting greater overall participation in data science activities.

7.4 Limitations and Future Improvements

1. Case Study Evaluation Limitation

a) Limitation:

Relying on a single case study ("Credit Risk Assessment") might limit the generalizability of my findings. Different domains have unique challenges and requirements, which could influence the effectiveness of each approach (Human Data Scientist, AutoML, Generative AI). For instance, the effectiveness of these methods can vary significantly in fields such as healthcare, finance, marketing or manufacturing.

b) Future Improvements:

- **Diverse Case Studies:** To address this limitation, one could incorporate a range of case studies from various industries. This would help in assessing the generalizability of the findings and understanding how each approach performs under different scenarios. For example, including case studies on customer segmentation in marketing, anomaly detection in cybersecurity or predictive maintenance in manufacturing could provide a broader perspective. A comparative analysis across various case studies will provide a richer understanding of where each approach excels or falls short, allowing for a more nuanced conclusion.

- **Diverse Machine Learning Problems:** Currently, my final year project only focuses on Supervised ML specifically, Binary Classification. The study should cover not only Regression but also Unsupervised ML and Reinforcement Learning.
- **Diverse Dataset Types:** The current study focusses only on Structured data. Given more time, the study should also be done using Time-Series and Unstructured data types such as images, text (emails, social media posts, documents).

2. Budget Constraints

a) Limitation:

Due to budget constraints, accessing commercial AutoML and Generative AI tools is challenging. Many advanced features and capabilities of these tools are often locked behind expensive licenses or subscription fees, which can be prohibitive for a student final year project. The subscription to ChatGPT Plus to access ChatGPT Data Analyst costs around USD 20 per month, which is quite close to RM 100 per month.

b) Future Improvements:

The following improvements could be applied if I am given more time and certain conditions are met.

- **Seek Partial Funding from University Institution:** If my university has funds available for student research, I could request partial funding or support from my university to cover the costs of commercial tools or additional resources needed for my project if they think that my project is innovative or has potential academic value.
- **Collaborate with Research Centers or Industry Partners:** I could seek to partner with university research centers or industry collaborators to gain access to commercial tools and resources. This collaboration could provide additional support and expertise, which can enhance the quality of my project.
- **Request University License from Vendor:** I could request an academic license from vendors for commercial tools and emphasize that the usage is for non-profit, academic purposes. This could potentially reduce costs and provide access to essential features needed for my project.
- **Broaden Tool Selection:** If I am able to access additional funding from any of the previous 3 future improvements, I could consider comparing against another AutoML like DataRobot, which can provide almost all the

functionalities via GUI. Additionally, I could consider comparing against other Generative AIs, such as Gemini, Bing Copilot and Bard.

3. Scalability and Performance Constraints

a) Limitation:

AutoML and Generative AI solutions may face performance issues with very large datasets or highly complex models. While they can handle a broad range of tasks, their efficiency might decrease as data size or complexity increases, compared to custom solutions developed by human data scientists. For this project, I only used a small credit risk assessment dataset (the German Credit dataset), which consists of only 1000 rows. H2O AutoML is highly memory-intensive and it does not allow the XGBoost algorithm to be selected if it detects that the platform does not have GPU. I have already mentioned earlier that the CDA Runtime is very awful and model training, testing and grid search need to be run on an alternate platform. Moreover, the CDA Runtime does not allow installation of new packages.

b) Future Improvements:

- **Benchmarking with Large Dataset:** If a large dataset is used for this project, I could include performance metrics and scalability tests with large and complex dataset to understand the limitations and strengths of each approach.
- **Optimizations and Enhancements:** At the same time, I could investigate how each solution can be optimized or scaled up for larger problems or datasets.
- **Using Google Colab or CUDA:** In this project, all the computations and modeling were done locally on Jupyter Notebook. In the future, I could consider using Google Colab or a machine with NVIDIA GPU to perform computationally intensive tasks such as Deep Learning and XGBoost. The TensorFlow deep learning framework is already CUDA-enabled.

4. Monitoring Phase

a) Limitation:

In this project, I made the decision to omit this phase of the CRISP-DM process. Monitoring is important for ensuring that the deployed model continues to perform well over time, especially in changing environments. Tracking model performances involves detecting data drift (statistical properties of the input data change over time, which can reduce the model's predictive accuracy) and concept drift (changes in the underlying relationships between input features and the target variable). The reason for omitting this phase is because I felt

the effort to build the infrastructure for logging, metrics collection, and drift detection is too large for an individual project.

7.5 Issues and Solutions

Issue 1: Problems with installing packages due to incompatible dependencies

One of the key challenges encountered was managing package dependencies. Different packages in the machine learning or AI ecosystem often rely on specific versions of underlying libraries. Using the latest versions of all packages can lead to conflicts, as some libraries may require older or incompatible versions of shared dependencies.

Solution 1

To solve this issue, I had to strike a balance between using the latest features and maintaining compatibility, which required manual intervention in managing these dependencies. I had to take note of the versions of the dependencies that were conflicting with one another and downgrade the appropriate packages. One of the scenarios in which this problem happened was when I installed tensorflow, FastAPI and MLFlow in the same virtual environment.

Issue 2: Inability to perform hyperparameter tuning using RandomizedSearchCV and GridSearchCV on ChatGPT Data Analyst Runtime

When using ChatGPT Data Analyst, efforts to tune model hyperparameters using techniques like RandomizedSearchCV and GridSearchCV were hindered by its runtime environmental constraints. The computational resources of the environment, likely CPU-limited, and network issues, made it infeasible to run these processes. This made it impossible to optimize machine learning models. As a result, these processing has to be offloaded to my laptop.

Solution 2

Many attempts were made to prompt ChatGPT Data Analyst to run the codes related to RandomizedSearchCV and GridSearchCV but the runtime returns an error status “Analyzed Error” everytime. However, because ChatGPT Data Analyst constantly indicated that there was a network issue or that the environment did not have enough resources to perform hyperparameter tuning, I had to run the hyperparameter tuning codes on my local computer to obtain the results. Fortunately, the hyperparameter tuning codes did not take long to execute.

Issue 3: Cannot pip install packages into ChatGPT runtime environment

The ChatGPT environment does not allow users to directly install Python packages via pip. This limitation restricts the ability to customize the environment with additional libraries or tools that may be required for specific tasks, making it difficult to fully leverage certain machine learning or data analysis workflows without access to all necessary tools. For example, after ChatGPT Data Analyst generated some code using the imblearn package to balance the training dataset, the runtime could not execute the code as its environment did not have the imblearn package. Therefore, I told it to pip install the package but it replied that it was not able to do so.

Solution 3

I instructed ChatGPT Data Analyst to provide an alternative package (sklearn) for balancing the dataset without the use of the imblearn package. FYI, the imblearn is a well known and widely used package. It is ironic that CDA generated code that use packages which it does not allowed to be installed in its runtime.

Issue 4: KerasClassifier modelling difficulty

The TensorFlow/Keras library previously included a wrapper that allowed deep learning models to be used as classification or regression estimators in scikit-learn. However, this wrapper was recently separated and made into its own standalone Python module.

Solution 4

I had to use SciKeras. SciKeras is a Python library that provides a seamless interface between TensorFlow/Keras and scikit-learn, allowing Keras models to be used as scikit-learn estimators for tasks like classification and regression.

Issue 5: KerasClassifier deployment difficulty

There was not that much documentation on how to deploy the KerasClassifier model. Somehow, even if I could save the model as a pickle file, I could not load the model later on when using the Spyder application to write the FastAPI app script.

Solution 5

Therefore, I had to use an unconventional method, which involves saving the Keras model using the save method and the model_ attribute. To load the Keras model into into the FastAPI app, my FastAPI app had to call a function from a python script to load and initialize the Keras model so that it is able to make predictions.

Issue 6: Subscription to ChatGPT Plus to use ChatGPT Data Analyst

For leveraging the advanced capabilities of Generative AI for data analysis, a subscription (USD20 per month) to ChatGPT Plus is required. This provides access to features like the ChatGPT Data Analyst, for various data analysis and machine learning tasks. However, subscribing added a financial cost to the project and reliance on the capabilities provided by a third-party platform. The minimum charge is 1 month's subscription even if one cancels the service after 1 day of use.

Solution 6

I had to be selective when to subscribe to ChatGPT Plus which is only when I had time to interact with it. After subscribing, I spent the next few days using it to complete the ChatGPT Data Analyst implementation of the case study.

References

- Acharya, N. (2023, September 26). *Why is it called “logistic regression” and not “logistic classification”?* Medium. <https://medium.com/@nirajan.acharya666/why-is-it-called-logistic-regression-and-not-logistic-classification-c201889d268c>
- Adam, F. M. (2023, May 26). *Classification algorithms with their solver parameters.* Medium. <https://medium.com/@fateemamohdadam2/classification-algorithms-with-their-solver-parameters-ce7828599611>
- Alam, S., Ayub, M. S., Arora, S., & Khan, M. A. (2023). An investigation of the imputation techniques for missing values in ordinal data enhancing clustering and classification analysis validity. *Decision Analytics Journal*, 9, 1–4. <https://doi.org/10.1016/j.dajour.2023.100341>
- Ali, M. (2023, June 28). Home - PyCaret. PyCaret. <https://pycaret.org/>
- Ajaymehta. (2023, June 8). *Exploring the benefits of embedded methods for feature selection*. Medium. <https://medium.com/@dancerworld60/exploring-the-benefits-of-embedded-methods-for-feature-selection-d7c9fb246eeb>
- Ajgaonkar, S. (2022). *Different clients communicating with the same H2O server* [Diagram]. From Salil Ajgaonkar, Practical Automated Machine Learning Using H2O.ai (p. 124). Livery Place, Birmingham: Packt, 2022.
- Ajgaonkar, S. (2022). *H2O AI high-level architecture* [Diagram]. From Salil Ajgaonkar, Practical Automated Machine Learning Using H2O.ai (p. 122). Livery Place, Birmingham: Packt, 2022.
- Amanpriyanshusms. (2024, January 3). *What is Automl in machine learning?* GeeksforGeeks. <https://www.geeksforgeeks.org/what-is-automl-in-machine-learning/>
- Amiyaranjanrout. (2023, January 10). *Advantages and disadvantages of logistic regression.* <https://www.geeksforgeeks.org/advantages-and-disadvantages-of-logistic-regression/>
- Arasteh, S. T., Han, T., Lotfinia, M., Kuhl, C., Kather, J. N., Truhn, D., & Nebelung, S. (2023). Large Language Models Streamline Automated Machine Learning for Clinical Studies, 12–13. https://www.researchgate.net/publication/373450909_Large_Language_Models_Streamline_Automated_Machine_Learning_for_Clinical_Studies
- Ashtari, H. (2024, February 21). *XGBoost vs. Random Forest vs. Gradient Boosting: Key Differences.* Spiceworks.com. <https://www.spiceworks.com/tech/artificial-intelligence/articles/xgboost-vs-random-forest-vs-gradient-boosting/>
- Attard, M. (2024, May 1). *8 key advantages and disadvantages of decision trees.* Inside Learning Machines. https://insidelearningmachines.com/advantages_and_disadvantages_of_decision_trees/
- AutoML. (n.d.). AutoML. <https://www.automl.org/automl/>
- Awan, A. A. (2023, August 30). *What are neural networks?* DataCamp. <https://www.datacamp.com/blog/what-are-neural-networks>

Project Title	References
---------------	------------

- AWS. (2024). *ML lifecycle phase - business goal - machine learning lens*. Amazon Web Services.
<https://docs.aws.amazon.com/wellarchitected/latest/machine-learning-lens/ml-lifecycle-phase-business-goal.html>
- Bahmani, M. (2023, August 23). *A quickstart guide to auto-sklearn (automl) for Machine Learning Practitioners*. neptune.ai. <https://neptune.ai/blog/a-quickstart-guide-to-auto-sklearn-automl-for-machine-learning-practitioners>
- Badr, W. (2019, February 22). *Having an imbalanced dataset? here is how you can fix it*. Medium.
<https://towardsdatascience.com/having-an-imbalanced-dataset-here-is-how-you-can-solve-it-1640568947eb>
- Bhupathiraju, S. (2023, May 15). *Columntransformer and pipeline ...they are here to make our life easy...* Medium. <https://medium.com/@shubavarma/columntransformer-and-pipeline-they-are-here-to-make-our-life-easy-b1fd5ecfbe5b>
- BIAŁOZOR, Ł. (2023, June 15). *Hallucinations of CHATGPT-4. Flying Bisons*. <https://flyingbisons.com/blog/hallucinations-of-chatgpt-4-even-the-most-powerful-tool-has-a-weakness>
- Bismi, I. (2023, February 2). *How to perform hyper-parameter tuning in artificial neural networks*. Medium.
<https://medium.com/@iqra.bismi/how-to-perform-hyper-parameter-tuning-in-artificial-neural-networks-2bed80903b48>
- Brownlee, J. (2020, August 26). *Repeated k-fold cross-validation for model evaluation in Python*. MachineLearningMastery.com. <https://machinelearningmastery.com/repeated-k-fold-cross-validation-with-python/>
- Brownlee, J. (2021, March 16). *Smote for imbalanced classification with python*. MachineLearningMastery.com. <https://machinelearningmastery.com/smote-oversampling-for-imbalanced-classification/>
- Brownlee, J. (2022, August 4). *How to grid search hyperparameters for deep learning models in python with keras*. MachineLearningMastery.com. <https://machinelearningmastery.com/grid-search-hyperparameters-deep-learning-models-python-keras/>
- Btd. (2024, January 20). *25 techniques and applications in feature engineering for Enhancing predictive performance*. Medium. <https://baotramduong.medium.com/machine-learning-14-feature-engineering-techniques-f97040e4f5b5>
- ChenDataBytes. (2024, January 26). *Master the art of Feature Engineering and feature selection*. Medium.
<https://medium.com/@chenycy/master-the-art-of-feature-engineering-and-feature-selection-e6e87f76f89b>
- Chouliaras, G. C., Kiełczewski, K., Beka, A., Konopnicki, D., & Bernardi, L. (2023). Best Practices for Machine Learning Systems: An Industrial Framework for Analysis and Optimization, 1–20. <http://dx.doi.org/10.13140/RG.2.2.31255.14249>
- CodingTheSmartWay. (2023, July 16). CHATGPT's code interpreter: The limitations you must be aware of.
<https://www.codingthesmartway.com/chatgpt-code-interpreter-the-limitations-you-must-be-aware-of/>
- Coursera Staff. (2023, November 30). What is a data scientist? salary, skills, and how to become one. Coursera.
<https://www.coursera.org/articles/what-is-a-data-scientist>

Project Title	References
---------------	------------

- DataScientest. (2024, January 21). Automl and Machine Learning Automation: A threat to data scientists? <https://datascientest.com/en/automl-and-machine-learning-automation-a-threat-to-data-scientists>
- Davenport, T. H., & Patil, D. (2022, July 15). *Is data scientist still the sexiest job of the 21st century?*. Harvard Business Review. <https://hbr.org/2022/07/is-data-scientist-still-the-sexiest-job-of-the-21st-century>
- DigitalOcean. (2024). Understanding and mitigating AI hallucination. <https://www.digitalocean.com/resources/article/ai-hallucination>
- Drpa. (2023, June 29). *Feature scaling-normalization, standardization*. Medium. <https://medium.com/@drpa/feature-scaling-normalization-standardization-eddbf9ec395>
- Emmanuel, T., Maupong, T., Mpoeleng, D., Semong, T., Mphago, B., & Tabona, O. (2021). A survey on missing data in machine learning. *Journal of Big Data*, 8(1), 1–31. <https://doi.org/10.1186/s40537-021-00516-9>
- FastAPI. (n.d.). *FASTAPI*. <https://fastapi.tiangolo.com/>
- FasterCapital. (n.d.). Boosting model accuracy with gradient boosting. <https://fastercapital.com/topics/boosting-model-accuracy-with-gradient-boosting.html/5>
- Favilla, V. (2023, May 30). *Regularization in logistic regression*. Medium. <https://medium.com/@vincefav/regularization-in-logistic-regression-14b50d7cc31>
- Ferreira, L., Pilastri, A., Martins, C. M., Pires, P. M., & Cortez, P. (2021). A comparison of AutoML Tools for machine learning, Deep Learning and XGBoost. 2021 International Joint Conference on Neural Networks (IJCNN), 7–7. <https://doi.org/10.1109/ijcnn52387.2021.9534091>
- Fleiss, A. (2023, February 19). *What are the advantages and disadvantages of Random Forest?*. Rebellion Research. <https://www.rebellionresearch.com/what-are-the-advantages-and-disadvantages-of-random-forest>
- Frenzel. (2023, October 24). *#KB resampling methods-part 2*. Medium. <https://prof-frenzel.medium.com/kb-resampling-methods-part-2-6d78ecc591fc>
- Ghosh, S. (2024, January 19). *The ultimate guide to evaluation and selection of models in ML*. neptune.ai. <https://neptune.ai/blog/ml-model-evaluation-and-selection>
- Gillett, T. (2023, July 15). *Exploring the impact of CHATGPT code interpreter on Data Science* [Post]. LinkedIn. [https://www.linkedin.com/pulse/exploring-impact-chatgpt-code-interpreter-data-science-tony-gillett/?trk=article\(ssr-frontend-pulse_more-articles_related-content-card](https://www.linkedin.com/pulse/exploring-impact-chatgpt-code-interpreter-data-science-tony-gillett/?trk=article(ssr-frontend-pulse_more-articles_related-content-card)
- Gudivada, V. N., Apon, A., & Ding, J. (2017). Data Quality Considerations for Big Data and Machine Learning: Going Beyond Data Cleaning and Transformations. *International Journal on Advances in Software*, 10(1 & 2), 1–18. https://doi.org/https://personales.upv.es/thinkmind/dl/journals/soft/soft_v10_n12_2017/soft_v10_n12_2017_1.pdf
- H2O. (2022). *H2O MLOPS*. H2O MLOps | H2O.ai. <https://h2o.ai/ja/resources/product-brief/h2o-mlops/>

Project Title	References
---------------	------------

- H2O. (2023, October 24). *H2O-autodoc*. PyPI. <https://pypi.org/project/h2o-autodoc/>
- H2O. (2024a, March 13). *Balance_classes*. balance_classes - H2O 3.46.0.1 documentation. https://docs.h2o.ai/h2o/latest-stable/h2o-docs/data-science/algo-params/balance_classes.html
- H2O. (2024b, March 13). H2O AUTOML: Automatic Machine learning. H2O AutoML: Automatic Machine Learning - H2O 3.46.0.1 documentation. <https://docs.h2o.ai/h2o/latest-stable/h2o-docs/automl.html>
- H2O. (2024c, March 13). *Model explainability*. Model Explainability - H2O 3.46.0.1 documentation. <https://docs.h2o.ai/h2o/latest-stable/h2o-docs/explain.html>
- H2O. (2024d, April 8). *Pysparkling*. PySparkling - H2O Sparkling Water 3.46.0.1-1-2.3 documentation. <https://docs.h2o.ai/sparkling-water/2.3/latest-stable/doc/pysparkling.html>
- Heymann, H., Kies, A. D., Frye, M., Schmitt, R. H., & Boza, A. (2022). Guideline for deployment of machine learning models for predictive quality in production. *Procedia CIRP*, 107, 815–820. <https://doi.org/10.1016/j.procir.2022.05.068>
- Hofmann, H. (1994, November 16). *Statlog (German credit data)* [Data set]. UCI Machine Learning Repository. <https://archive.ics.uci.edu/dataset/144/statlog+german+credit+data>
- IBM. (n.d.). What is automl? <https://www.ibm.com/topics/automl>
- Ioffe, S., & Szegedy , C. (2015). Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. *Google Research*, 1–5. <https://doi.org/10.48550/arXiv.1502.03167>
- Irvine, D. J., Halloran, L. J., & Brunner, P. (2023). Opportunities and limitations of the ChatGPT Advanced Data Analysis plugin for Hydrological Analyses. *Hydrological Processes*, 37(10), 1–5. <https://doi.org/10.1002/hyp.15015>
- Ivy, M. (2023, October 30). *Machine learning model: Decision trees part 1*. Medium. <https://medium.com/@marjorieivy/machine-learning-model-decision-trees-part-1-adc97a2abdf6>
- J., P. K. (2023, August 30). *Empowering intelligence: Automated machine learning (automl) unveiled - making machine learning accessible to all* [Post]. LinkedIn. <https://www.linkedin.com/pulse/empowering-intelligence-automated-machine-learning-automl-jha/>
- JavaTpoint. (2021). GBM in Machine Learning - JavaTpoint. [www.javatpoint.com](http://www.javatpoint.com/gbm-in-machine-learning). <https://www.javatpoint.com/gbm-in-machine-learning>
- Jin, H., Chollet , F., Song, Q., & Hu, X. (2024). AutoKeras: an AutoML library for deep learning. *The Journal of Machine Learning Research*, 24(1), 169–174. <https://www.jmlr.org/papers/volume24/20-1355/20-1355.pdf>
- Kashaboina, M. (2023, July 7). *Matrix and vector operations in logistic regression*. Medium. <https://towardsdatascience.com/matrix-and-vector-operations-in-logistic-regression-e35714c4810f>
- Kaushik, S. (2024, April 30). *Introduction to feature selection methods with an example*. Analytics Vidhya. <https://www.analyticsvidhya.com/blog/2016/12/introduction-to-feature-selection-methods-with-an-example-or-how-to-select-the-right-variables/>

- Kharkar, D. (2023, July 16). *About boosting and gradient boosting algorithm*. Medium. <https://medium.com/@dishantkharkar9/about-boosting-and-gradient-boosting-algorithm-98dd4081ec18>
- Kılıç, iyurek. (2023, September 23). *Gradient Boosting Machines (GBM) with python example*. Medium. <https://medium.com/@ilyurek/gradient-boosting-machines-gbm-with-python-example-b65421d10f2a>
- Kotsiantis, S., Kanellopoulos, D., & Pintelas, P. (2005). Handling imbalanced datasets: A review. *GESTS International Transactions on Computer Science and Engineering*, 30, 1–9. https://www.researchgate.net/publication/228084509_Handling_imbalanced_datasets_A_review
- Kotsilieris, T., Anagnostopoulos, I., & Livieris, I. E. (2022). Special issue: Regularization techniques for machine learning and their applications. *Electronics*, 11(4), 1–3. <https://doi.org/10.3390/electronics11040521>
- Krauß, J., Pacheco, B. M., Zang, H. M., & Schmitt, R. H. (2020). Automated Machine Learning for predictive quality in production. *Procedia CIRP*, 93, 443–448. <https://doi.org/10.1016/j.procir.2020.04.039>
- Lin, W. W. K. (2023). *How Can You Handle Inconsistent Data Formatting in Machine Learning?*, 1–2. https://www.researchgate.net/publication/375770573_How_can_you_handle_inconsistent_data_formatting_in_machine_learning
- Luna, Z. (2021). CRISP-DM [Online image]. Medium. <https://medium.com/analytics-vidhya/understanding-crisp-dm-and-its-importance-in-data-science-projects-91c8742c9f9b>.
- Lyu, Q. (2021, March 26). *Implementing random forest*. Medium. <https://towardsdatascience.com/implementing-random-forest-26dd3e4f55c3>
- Marr, B. (2023, October 5). *Will CHATGPT put data analysts out of work?*. Forbes. <https://www.forbes.com/sites/bernardmarr/2023/02/07/will-chatgpt-put-data-analysts-out-of-work/?sh=aac58214030d>
- McKinsey & Company. (2023, January 19). What is Generative Ai? <https://www.mckinsey.com/featured-insights/mckinsey-explainers/what-is-generative-ai>
- Medhi, D. (2023, May 18). *Elevate your machine learning workflow: How to use mlflow for experiment tracking and model...* Medium. <https://dipankarmedhl.medium.com/elevate-your-machine-learning-workflow-how-to-use-mlflow-for-experiment-tracking-and-model-419c2a700ec5>
- MIT. (2024, February 2). How to use CHATGPT's Advanced Data Analysis Feature. MIT Sloan Teaching & Learning Technologies. <https://mitsloanedtech.mit.edu/ai/tools/data-analysis/how-to-use-chatgpts-advanced-data-analysis-feature/>
- MSOE. (2024, April 4). *The importance of feature engineering in Machine Learning*. Online Degree Programs | Milwaukee School of Engineering. <https://online.msoe.edu/engineering/blog/importance-of-feature-engineering-in-machine-learning>
- Naderalvojoud, B., & Hernandez-Boussard, T. (2024). Improving machine learning with ensemble learning on observational healthcare data. AMIA ... Annual Symposium proceedings. AMIA Symposium, 2023, 521–529. <https://www.ncbi.nlm.nih.gov/pmc/articles/PMC10785929/>

- Nair, G. (2023, August 23). *The spark your neural network needs: Understanding the significance of activation functions*. Medium. <https://medium.com/@gauravnair/the-spark-your-neural-network-needs-understanding-the-significance-of-activation-functions-6b82d5f27fbf>
- Nassar, O. (2023). *Data Leakage in Machine Learning*, 1–3. <http://dx.doi.org/10.13140/RG.2.2.27468.59528>
- OpenAI. (2023, August 28). Introducing ChatGPT Enterprise. <https://openai.com/blog/introducing-chatgpt-enterprise>
- OpenAI. (2024a). ChatGPT (Feb 13 version) [Large language model]. <https://chat.openai.com/chat>
- OpenAI. (2024b). How do I use CHATGPT browse with Bing to search the web? | openai help center. <https://help.openai.com/en/articles/8077698-how-do-i-use-chatgpt-browse-with-bing-to-search-the-web>
- Pargent, F., Schoedel, R., & Stachl, C. (2023). Best practices in supervised machine learning: A tutorial for psychologists. *Advances in Methods and Practices in Psychological Science*, 6(3). <https://doi.org/10.1177/25152459231162559>
- Pascual, A. L. (2021, August 24). *Can automl replace data scientists?*. Medium. <https://medium.com/analytics-vidhya/can-automl-replace-data-scientists-cf1f939a82ba>
- Paterson, Katie (2024, January 8). *What can the ChatGPT data analysis chatbot do?*. Zapier. <https://zapier.com/blog/chatgpt-data-analysis/>
- Piltch, Avram. (2023, November 14). *ChatGPT's New Code Interpreter Has Giant Security Hole, Allows Hackers to Steal Your Data*. <https://www.tomshardware.com/news/chatgpt-code-interpreter-security-hole>
- Pramod, O. (2023, January 29). *Decision trees*. Medium. <https://medium.com/@ompramod9921/decision-trees-8e2391f93fa7>
- Purwanto, R., Pal, A., Blair, A., & Jha, S. (2021). Man versus Machine: AutoML and Human Experts' Role in Phishing Detection. Arxiv, 1–28. <https://doi.org/10.48550/arXiv.2108.12193>
- Pushkar. (2023, March 21). *How to choose the right model: A guide to model selection in machine learning*. Medium. <https://medium.com/codersarts/how-to-choose-the-right-model-a-guide-to-model-selection-in-machine-learning-d253ad98e619>
- Python Programmer. (2023, July 11). *Exploring decision tree algorithms: A powerful tool for machine learning*. Medium. <https://medium.com/@impythonprogrammer/exploring-decision-tree-algorithms-a-powerful-tool-for-machine-learning-6a34b982e461>
- Sachinsoni. (2023, August 20). *Mastering random forests: Unraveling the magic of ensemble learning*. Medium. <https://medium.com/@sachinsoni600517/mastering-random-forests-unraveling-the-magic-of-ensemble-learning-e80472723cee>
- Sagar, A. (2019, December 6). *5 techniques to prevent overfitting in neural networks*. KDnuggets. <https://www.kdnuggets.com/2019/12/5-techniques-prevent-overfitting-neural-networks.html>
- Sahoo, K., Samal, A. K., Pramanik, J., & Pani, S. K. (2019). Exploratory Data Analysis using Python. *International Journal of Innovative Technology and Exploring Engineering*, 8(12), 4727–4734. https://www.researchgate.net/publication/341121348_Exploratory_Data_Analysis_using_Python

- Saini, A. (2024, January 10). *Gradient boosting algorithm: A complete guide for beginners*. Analytics Vidhya. <https://www.analyticsvidhya.com/blog/2021/09/gradient-boosting-algorithm-a-complete-guide-for-beginners/>
- Salehin, I., Islam, Md. S., Saha, P., Noman, S. M., Tuni, A., Hasan, Md. M., & Baten, Md. A. (2024). AUTOML: A systematic review on Automated Machine Learning with Neural Architecture Search. *Journal of Information and Intelligence*, 2(1), 52–81. <https://doi.org/10.1016/j.jiixd.2023.10.002>
- Sangfor Technologies. (2023, October 3). What is SOC 2 Compliance? <https://www.sangfor.com/glossary/cybersecurity/what-is-soc-2-compliance>
- Sarita, P. (2023, October 3). Basic understanding of neural network structure. Medium. https://medium.com/@sarita_68521/basic-understanding-of-neural-network-structure-eecc8f149a23
- scikit-learn. (2024). 3.1. cross-validation: Evaluating estimator performance. scikit. https://scikit-learn.org/stable/modules/cross_validation.html
- scikit-learn. (2024). 10. common pitfalls and recommended practices. scikit. https://scikit-learn.org/stable/common_pitfalls.html
- Selvaraj, N. (2023a, July 27). CHATGPT code interpreter: Do data science in minutes. KDnuggets. <https://www.kdnuggets.com/2023/07/chatgpt-code-interpreter-data-science-minutes.html>
- Selvaraj, N. (2023b, September 16). Will CHATGPT take data science jobs?. Medium. <https://towardsdatascience.com/will-chatgpt-take-data-science-jobs-73e4fbb706b8>
- Sha, Arjun. (2023, July 12). ChatGPT Code Interpreter: What Is It and How It Works. Beebom. <https://beebom.com/chatgpt-code-interpreter/>
- Sharma, N. (2021, August 3). Ways to detect and remove the outliers. Medium. <https://towardsdatascience.com/ways-to-detect-and-remove-the-outliers-404d16608dba>
- Shivanipickl. (2023, November 16). What is feature scaling and why does machine learning need it?. Medium. <https://medium.com/@shivanipickl/what-is-feature-scaling-and-why-does-machine-learning-need-it-104eedebb1c9>
- Shu, C. (2019, July 27). It's time to think more about the pipeline. Medium. <https://towardsdatascience.com/its-time-to-think-more-about-the-pipeline-5346c039a803>
- Singh, N. (2023, August 28). How outliers can skew your insights and what to do about it. Medium. <https://pub.aimind.so/how-outliers-can-skew-your-insights-and-what-to-do-about-it-2f8e987a4fda>
- Ski, M. C. (2024). An Overview of Outlier Detection Methods. *London Journal of Engineering Research*, 24(2), 37–72. https://journalspress.com/LJER_Volume24/An-Overview-of-Outlier-Detection-Methods.pdf
- S, T. S. (2020, May 29). Types of transformations for better normal distribution. Medium. <https://towardsdatascience.com/types-of-transformations-for-better-normal-distribution-61c22668d3b9>

Project Title	References
---------------	------------

- Statistics Solutions. (2024, April 17). *Assumptions of logistic regression.* <https://www.statisticssolutions.com/free-resources/directory-of-statistical-analyses/assumptions-of-logistic-regression/>
- Thanmai, B. T., Srinivas, T. A. S., Donald, A. D., Thippanna, G., Madileddy, C., & Srihith, I. V. D. (2023). Data Detective: Hunting Outliers with Python. *Advanced Innovations in Computer Programming Languages*, 5(3), 28–33. <https://doi.org/10.5281/zenodo.8207454>
- The Investopedia Team. (2024, February 28). *Credit risk: Definition, role of ratings, and examples.* Investopedia. <https://www.investopedia.com/terms/c/creditrisk.asp>
- Theng, D., & Bhoyar, K. K. (2023). Feature selection techniques for Machine Learning: A Survey of more than two decades of research. *Knowledge and Information Systems*, 66(3), 1575–1637. <https://doi.org/10.1007/s10115-023-02010-5>
- Thevapalan, A., & Le, J. (2023, June 1). R decision trees tutorial: Examples & code in R for Regression & Classification. DataCamp. <https://www.datacamp.com/tutorial/decision-trees-R>
- Thulo, C. (2023, October 20). *Most important interview question of Decision Tree Algorithm.* Medium. <https://medium.com/@codethulo/most-important-interview-question-of-decision-tree-algorithm-586f21812269>
- Tuychiev, B. (2023, December 27). *A guide to the gradient boosting algorithm.* DataCamp. <https://www.datacamp.com/tutorial/guide-to-the-gradient-boosting-algorithm>
- Uber. (2019, February 11). Introducing Ludwig, a code-free deep learning toolbox - uber engineering blog. Uber Blog. <https://eng.uber.com/introducing-ludwig/>
- Udilač, A.-I. (2023). Encoding Methods for Categorical Data: A Comparative Analysis for Linear Models, Decision Trees, and Support Vector Machines. *EEMCS Faculty Delft University of Technology*, 2–4. <https://repository.tudelft.nl/islandora/object/uuid:10b91b99-2685-4a45-b44e-48fbbf808ce2/datastream/OBJ/download>
- Vivek4x59. (2024, March 26). *Dropout regularization in Deep learning.* GeeksforGeeks. <https://www.geeksforgeeks.org/dropout-regularization-in-deep-learning/>
- West Virginia University (2023, August 25). WVU researchers weigh pros and cons of Code Interpreter, await upgrades. <https://wvutoday.wvu.edu/stories/2023/08/25/wvu-researchers-weigh-pros-and-cons-of-code-interpreter-await-upgrades>
- Yin, X. (2022, March 17). *AutoGluon: Easy-to-use and high-performing automl.* Medium. https://medium.com/@xiangyu_y/autogluon-easy-to-use-and-high-performing-automl-40d7408ba8b6
- YoavAvneon. (2023, February 16). *Regularize it!.* Medium. <https://medium.com/@yoavyeledteva/regularize-it-9f020cb14e90>
- Yosifova, A. (2024, January 8). *ChatGPT Code Interpreter: What It Is and How It Works.* <https://365datascience.com/trending/chatgpt-code-interpreter-what-it-is-and-how-it-works/>

Appendices

User Guide:

1. Download and Install Anaconda Navigator and ensure it has Jupyter Notebook and Spyder. You may get it from here: <https://docs.anaconda.com/navigator/>.
2. Download and Install H2O. You may get it from here: <https://docs.h2o.ai/h2o/latest-stable/h2o-docs/downloading.html>
3. In Anaconda Navigator, create 2 environments. One environment is for both AutoML and Human Data Scientist approaches. Another environment is for the Generative AI approach.
4. Get all the necessary packages for the Generative AI environment from cdarequirements.txt.
5. Get all the necessary packages for the AutoML and Human Data Scientist approaches from fyprequirements.txt.
6. Run the .ipynbs in Jupyter Notebook.
7. To run the Streamlit and FastAPI applications, use the deployment procedures listed in Chapter 6.

This page is intentionally left blank to indicate the back cover. Ensure that the back cover is black in color.