Cardiff School of Mathematics

MAT012 Credit Risk Scoring

Assignment 2022/2023

Eshaq Rahmani

supervised by
Dr. Meirion Morgan

May 2, 2023

Part A

Answer to Q1:

Essay (Word Count: 1330)

Title:
Examining the Complexities of Building Credit Risk Scoring Models

1. Introduction

Credit risk scoring models play a vital role in financial institutions' lending decisions, enabling them to assess the likelihood of a borrower defaulting on their loan obligations (Hand and Henley, 1997). These models help lenders manage their risk exposure, price loans appropriately, and make informed decisions about extending credit. This essay will critically examine the factors that must be considered when developing a credit risk scoring model.

2. Overview of credit risk scoring models

Credit risk scoring models are designed to predict the likelihood of default for potential borrowers based on their credit history, demographic information, and other relevant factors (Thomas et al., 2005). These models have become an integral part of the lending industry, enabling lenders to make informed decisions about the creditworthiness of their clients. There are several types of credit scoring models, including traditional logistic regression, machine learning, and survival analysis models, such as the Cox proportional hazards model.

Traditional logistic regression models have been widely used for credit scoring due to their simplicity and ease of interpretation (Anderson, 2007). However, the increasing availability of data and computational power has resulted in the popularity of machine learning models such as decision trees, support vector machines, and neural networks (Yeh and Lien, 2009). These models can handle complex relationships and large datasets more effectively, potentially resulting in better predictive performance. For example, FICO, a leading credit scoring company, has been exploring alternative data and machine learning models, such as gradient boosting machines and neural networks, to enhance credit risk modelling and improve predictive performance (FICO, 2020).

In addition to machine learning models, survival analysis models like Cox's proportional hazards model estimate the time until an event such as a loan default occurs (Cox, 1972). This approach provides additional insights into the timing of defaults, which may be valuable in specific credit risk management contexts. With the use of these models, lenders can better understand the creditworthiness of their clients and make informed lending decisions.

3. CRISP-DM framework and its relevance to credit risk scoring

The Cross-Industry Standard Process for Data Mining (CRISP-DM) is a well-established framework for data mining projects, including credit risk scoring models (Shearer, 2000). The framework comprises six phases: business understanding, data understanding, data preparation, modelling, evaluation, and deployment (Wirth and Hipp, 2000).

In credit risk scoring, business understanding involves defining objectives such as predicting default probabilities or identifying high-risk borrowers (Lessmann et al., 2015). Data understanding requires identifying and assessing relevant data sources like credit bureaus, applications, and transactional data (Thomas et al., 2005). Data preparation involves cleaning, transforming, and enriching data, including handling missing values and feature engineering (Pyle, 1999). Modelling involves selecting suitable techniques and developing the credit risk model (Yeh and Lien, 2009). The evaluation assesses the model's performance using metrics like accuracy and the area under the ROC curve (Siddiqi, 2006).

The deployment phase focuses on integrating the model into the lender's operational systems for real-time credit risk assessment (Chen and Li, 2010). The following sections will examine factors to consider during each CRISP-DM phase and potential challenges and trade-offs when developing a credit risk scoring model.

1. **Business Understanding:** A clear understanding of the project's objectives is crucial in credit risk scoring (Lessmann et al., 2015). Objectives may include minimizing default rates, supporting risk-based pricing, or meeting regulatory requirements. Identifying the primary stakeholders and understanding their expectations and requirements

help align the project with the organization's strategic goals (Bensic et al., 2005). In this phase, it is essential to consider the trade-offs between model complexity, interpretability, and predictive performance, as well as the potential impact of the model on the lender's risk exposure and profitability (Bravo et al., 2018).

2. **Data Understanding:** Identifying relevant data sources and assessing their quality is critical in credit risk modelling (Thomas et al., 2005). Standard data sources include credit bureau reports, loan applications, and transactional data. Therefore, data quality, representativeness, and compliance with data protection regulations (e.g., GDPR) are essential (Martens et al., 2007). In addition, understanding the data's limitations, biases, and potential pitfalls can help prevent overfitting and improve model generalizability (Baesens et al., 2003). A real-world example can be seen in the case of LendingClub, an online peer-to-peer lending platform. LendingClub effectively combines data from credit bureaus, loan applications, and transactional data to create a comprehensive credit risk scoring model, showcasing the importance of using diverse data sources in the model-building process (Emekter et al., 2015).

3. **Data Preparation:** Data preparation is often the most time-consuming phase of the CRISP-DM process and is crucial for successful credit risk modelling (Pyle, 1999). Cleaning and transforming data, handling missing values, and encoding categorical variables are essential data preparation steps. Feature engineering, such as creating new variables (e.g., monthly-debt ratio) or aggregating transaction data, can help improve model performance (Brown and Mues, 2012). Balancing the trade-offs between the inclusion of potentially valuable predictors and the risk of overfitting or multicollinearity is a critical consideration (Bensic et al., 2005).

4. **Modeling:** Selecting appropriate techniques and tuning model parameters are vital for credit risk scoring (Yeh and Lien, 2009). Logistic regression, decision trees, support vector machines, and neural networks are commonly used methods. Comparing different models' performances using cross-validation can help identify the most suitable approach (Breiman, 2001). In addition, ensuring model interpretability and compliance with regulations, such as the "right to explanation"

under GDPR, may be essential factors to consider (Goodman and Flaxman, 2016).

5. **Evaluation:** Evaluating the credit risk model's performance using relevant metrics and validation techniques is crucial (Siddiqi, 2006). Standard performance metrics include accuracy, the area under the ROC curve, and the Kolmogorov-Smirnov statistic (Siddiqi, 2006). Validation techniques like out-of-time and cross-validation can help assess the model's generalizability and stability (Hastie et al., 2009). Considering the trade-offs between model performance, complexity, and interpretability is essential in this phase (Hand and Henley, 1997).

6. **Deployment:** Integrating the credit risk model into the lender's operational systems and processes is critical in ensuring its practical utility (Chen and Li, 2010). For example, ensuring seamless integration with existing systems, such as loan origination systems or credit management platforms, can help maximize the model's impact on lending decisions. In addition, monitoring the model's performance over time and updating it as necessary to account for changing economic conditions, borrower behaviour or regulatory requirements is essential for maintaining its effectiveness (Khandani et al., 2010). Capital One, a leading financial services company in the United States, has adopted machine learning models for credit risk assessment, which has improved its ability to make lending decisions and manage risk more effectively (Harvard Business School, 2020).

4. Alternative approaches and future trends

While the CRISP-DM framework is widely used for credit risk scorecard development, alternative methodologies, such as the Knowledge Discovery in Databases (KDD) process, can also be applied (Fayyad et al., 1996). Additionally, ensemble methods, which combine multiple models to improve prediction accuracy, have gained popularity in credit risk modelling (Zhou, 2012).

Future trends in credit risk scoring may include incorporating alternative data sources, such as social media data, geolocation data, or device usage patterns, to enhance predictive performance (Berg et al., 2020). These alternative data sources may provide additional insights into a borrower's creditworthiness and financial behaviour, allowing for a more accurate risk assessment. For instance, some financial institutions, particularly those in the microfinance industry, have started using alternative data sources, such as psychometric tests, social media, and other non-traditional data, to assess the creditworthiness of potential borrowers who lack traditional credit histories (Sánchez et al., 2019). However, using alternative data raises concerns regarding privacy, data protection, and potential biases in the model, which must be carefully addressed (Jagtiani and Lemieux, 2018).

Another future trend is the increased use of advanced machine learning techniques, such as deep learning and reinforcement learning, in credit risk modelling (Goodfellow et al., 2016). These techniques can capture complex patterns and relationships in the data, leading to better predictive performance. However, their increased complexity and computational requirements regarding interpretability, model validation, and regulatory compliance must be revised (Schapire, 2018).

Furthermore, the use of explainable artificial intelligence (XAI) methods is expected to grow in importance as regulators and stakeholders demand more transparency and interpretability in credit risk models (Adadi and Berrada, 2018). Techniques like LIME (Local Interpretable Model-agnostic Explanations) (Ribeiro et al., 2016) and SHAP (Shapley Additive Explanations) (Lundberg and Lee, 2017) can help provide insights into the model's decision-making process and enhance trust in the model's predictions.

5. Conclusion

In conclusion, developing a credit risk scoring model involves critically examining various factors and considerations across the CRISP-DM framework's phases. Ensuring a thorough understanding of the business objectives, selecting relevant and high-quality data, applying appropriate modelling techniques, evaluating model performance, and deploying the model are all essential. Alternative approaches and future trends, such as using alternative data sources, advanced machine learning techniques, and explainable AI methods, offer promising opportunities for improving credit risk modelling but also present challenges that must be carefully addressed. By considering these factors and adapting the process to the project's specific needs, a credit risk scoring model can be developed that effectively supports lending decisions and risk management.

6. References

Adadi, A. and Berrada, M., 2018. Peeking inside the black-box: A survey on explainable artificial intelligence (XAI). *IEEE Access*, 6, pp.52138-52160.

Anderson, R., 2007. The Credit Scoring Toolkit: Theory and Practice for Retail Credit Risk Management and Decision Automation. *Oxford University Press*.

Baesens, B., Van Gestel, T., Viaene, S., Stepanova, M., Suykens, J., and Vanthienen, J., 2003. Benchmarking state-of-the-art classification algorithms for credit scoring. *Journal of the Operational Research Society*, 54(6), pp.627-635.

Berg, T., Burg, V., Gombović, A., and Puri, M., 2020. On the rise of fintechs – credit scoring using digital footprints. *The Review of Financial Studies*, 33(7), pp.2845-2897.

Bensic, M., Sarlija, N., and Zekic-Susac, M., 2005. Modelling small-business credit scoring by using logistic regression, neural networks, and decision trees. *Intelligent Systems in Accounting, Finance and Management*, 13(3), pp.133-150.

Breiman, L., 2001. Random forests. *Machine learning*, 45(1), pp.5-32.

Brown, I. and Mues, C., 2012. An experimental comparison of classification algorithms for imbalanced credit scoring data sets. *Expert Systems with Applications*, 39(3), pp.3446-3453.

Chen, S. H., and Li, H. C., 2010. A two-stage credit scoring model using artificial neural networks and multivariate adaptive regression splines. *Expert Systems with Applications*, 37(7), pp.4898-4906.

Cox, D. R., 1972. Regression models and life-tables. *Journal of the Royal Statistical Society: Series B (Methodological)*, 34(2), pp.187-220.

Emekter, R., Tu, Y., Jirasakuldech, B. and Lu, M., 2015. Evaluating credit risk and loan performance in online Peer-to-Peer (P2P) lending. *Applied*

Economics, 47(1), pp.54-70.

Fayyad, U., Piatetsky-Shapiro, G., and Smyth, P., 1996. From data mining to knowledge discovery in databases. *AI magazine*, 17(3), pp.37-54.

FICO, 2020. Using Alternative Data in Credit Risk Modeling. [online] Available at: `https://www.fico.com/blogs/using-alternative-data-credit-risk-modelling` [Accessed 30 April 2023].

Goodman, B., and Flaxman, S., 2016. EU regulations on algorithmic decision-making and a" right to explanation". *arXiv preprint arXiv:1606.08813.*

Goodfellow, I., Bengio, Y., and Courville, A., 2016. *Deep Learning.* MIT Press.

Hand, D. J., and Henley, W. E., 1997. Statistical classification methods in consumer credit scoring: a review. *Journal of the Royal Statistical Society: Series A (Statistics in Society)*, 160(3), pp.523-541.

Harvard Business School, 2020. Machine Learning in Credit Assessment at Capital One. [online] Available at: `https://d3.harvard.edu/platform-rctom/submission/machine-learning-in-credit-assessment-at-capital-one/` [Accessed 30 April 2023].

Hastie, T., Tibshirani, R., and Friedman, J., 2009. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction.* Springer Science & Business Media.

Jagtiani, J., and Lemieux, C., 2018. Do fintech lenders penetrate areas that are underserved by traditional banks? Journal of Economics and Business, 100, pp.43-54.

Khandani, A. E., Kim, A. J., and Lo, A. W., 2010. Consumer credit-risk models via machine-learning algorithms. *Journal of Banking & Finance*, 34(11), pp.2767-2787.

Lessmann, S., Baesens, B., Seow, H. V., and Thomas, L. C., 2015. Benchmarking state-of-the-art classification algorithms for credit scoring: An update of research.

Lundberg, S. M., and Lee, S. I., 2017. A unified approach to interpreting model predictions. In *Advances in Neural Information Processing Systems* (pp. 4765-4774).

Martens, D., Baesens, B., Van Gestel, T., and Vanthienen, J., 2007. Comprehensible credit scoring models using rule extraction from support vector machines. *European Journal of Operational Research*, 183(3), pp.1466-1476.

Pyle, D., 1999. *Data Preparation for Data Mining*. Morgan Kaufmann.

Ribeiro, M. T., Singh, S., and Guestrin, C., 2016. "Why should I trust you?": Explaining the predictions of any classifier. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (pp. 1135-1144).

Sánchez, J.S., Muñoz, A. and Caro, R.M., 2019. Credit scoring models for the microfinance industry using neural networks: Evidence from Peru. *Expert Systems with Applications*, 116, pp.494-501.

Schapire, R. E., 2018. A brief introduction to boosting. In *Proceedings of the 16th International Joint Conference on Artificial Intelligence* (pp. 1401-1406).

Shearer, C., 2000. The CRISP-DM model: The new blueprint for data mining. *Journal of Data Warehousing*, 5(4), pp.13-22.

Siddiqi, N., 2006. *Credit Risk Scorecards: Developing and Implementing Intelligent Credit Scoring*. John Wiley & Sons.

Thomas, L. C., Oliver, R. W., and Hand, D. J., 2005. A survey of the issues in consumer credit modelling research. *Journal of the Operational Research Society*, 56(9), pp.1006-1015.

Wirth, R., and Hipp, J., 2000. CRISP-DM: Towards a standard process model for data mining. In *Proceedings of the 4th International Conference on the Practical Applications of Knowledge Discovery and Data Mining* (pp.

29-39).

Zhou, Z. H., 2012. *Ensemble Methods: Foundations and Algorithms.* CRC Press.

Answer to Q2:

Essay (Word Count: 1294)

Title:
Comparing Cox's Proportional Hazards Model and Logistic Regression for Credit Scorecard Construction: A Theoretical Perspective

1. Introduction

Credit risk scoring plays a vital role in the financial industry, as it helps lenders evaluate the risk associated with lending to potential borrowers (Thomas, 2000). Therefore, developing an accurate and reliable credit scoring model is crucial for lenders to make informed decisions about extending credit (Hand and Henley, 1997). Cox's proportional hazards (PH) model and logistic regression are two widely used techniques in credit risk modelling (Anderson, 2007). This essay aims to explain how, in theory, Cox's PH model can be used for constructing a credit risk scorecard and comment on the relative popularity of this model compared to logistic regression in scorecard construction.

2. Overview of Cox's Proportional Hazards Model

Cox's PH model, also known as the semi-parametric proportional hazards model, is a popular technique for survival analysis (Cox, 1972). Survival analysis focuses on the study of the time until the occurrence of an event of interest, which could be a borrower's default in the context of credit risk scoring (Altman and Saunders, 1997). The Cox PH model can handle time-to-event data, incorporating information about the event occurrence and the time at which the event occurs (Kleinbaum and Klein, 2005). The Cox PH model assumes that an individual's hazard function is proportional to a baseline hazard function, where the proportionality constant is a function of the individual's covariates (Cox, 1972). The hazard function is the instantaneous probability of the event occurring at a specific time, given that the individual has survived up to that time (Kleinbaum and Klein, 2005). The model can be represented as follows:

$$h(t, x) = h_0(t) \exp(\beta_1 x_1 + \beta_2 x_2 + \cdots + \beta_p x_p), \tag{1}$$

where $h(t, x)$ is the hazard function for an individual with covariates $x = (x_1, x_2, \ldots, x_p)$ at time t, $h_0(t)$ is the baseline hazard function, $\beta_1, \beta_2, \ldots, \beta_p$ are the regression coefficients, and $\exp(\cdot)$ denotes the exponential function (Cox, 1972).

In theory, the Cox PH model can be used for constructing a credit risk scorecard by modelling the hazard function of default for borrowers based on their relevant credit risk factors (Cox, 1972). The regression coefficients obtained from the model can be used to assign scores to the borrowers, reflecting their credit risk (Altman and Saunders, 1997). The higher the score, the lower the risk associated with the borrower, and vice versa. Using the Cox PH model, lenders can estimate each borrower's default probability and make more informed lending decisions (Anderson, 2007).

A study by Narain (1992) demonstrated the practical application of Cox's PH model in credit risk analysis. The researchers applied Cox's PH model to analyze mortgage loan delinquency data from a U.S. mortgage lender. They evaluated the impact of various borrowers and loan characteristics on the hazard of mortgage delinquency. The findings indicated that Cox's PH model offered valuable insights into the timing of delinquency events and the relative importance of different risk factors, ultimately helping the lender better manage its credit risk exposure.

3. Overview of Logistic Regression

Logistic regression is a widely used statistical method for binary classification problems, including credit risk scoring (Hosmer Jr et al., 2013). Logistic regression models the probability of an event occurring, such as a borrower's default, based on a linear combination of the covariates or risk factors (Hastie et al., 2009). The logistic regression model can be represented as follows:

$$P(Y = 1|x) = \frac{\exp(\beta_0 + \beta_1 x_1 + \beta_2 x_2 + \cdots + \beta_p x_p)}{1 + \exp(\beta_0 + \beta_1 x_1 + \beta_2 x_2 + \cdots + \beta_p x_p)}, \tag{2}$$

where $P(Y = 1|x)$ denotes the probability of the event (e.g., default) occurring for an individual with covariates $x = (x_1, x_2, \ldots, x_p)$, and $\beta_0, \beta_1, \ldots, \beta_p$

are the regression coefficients (Hosmer Jr et al., 2013).

Logistic regression can be used to construct a credit risk scorecard by estimating borrowers' default probability based on their credit risk factors (Anderson, 2007). The regression coefficients obtained from the model can be used to assign scores to borrowers, reflecting their credit risk (Thomas, 2000). Like the Cox PH model, a higher score indicates lower risk, and a lower score indicates higher risk. Logistic regression is a popular choice for credit risk modelling due to its simplicity and ease of interpretation (Anderson, 2007).

A study by Abdou et al. (2008) applied logistic regression to develop a credit scorecard for Egyptian banks. The researchers used logistic regression to model the probability of default based on various financial ratios and other borrower characteristics. The logistic regression model was accurate and robust, providing the banks with a practical tool for credit risk assessment and decision-making.

4. Comparison of Cox's PH Model and Logistic Regression

Comparing the two models regarding their assumptions, applicability, and flexibility reveals several key differences. The Cox PH model explicitly models the time-to-event data, providing insights into the timing of default events (Cox, 1972). This feature can benefit lenders, as it helps them better understand the dynamics of credit risk over time. However, the proportional hazards assumption may only sometimes hold in practice, limiting the model's applicability in certain situations (Kleinbaum and Klein, 2005).

On the other hand, logistic regression directly models the probability of an event occurring, making it more suitable for binary classification tasks like credit scoring (Hosmer Jr et al., 2013). Its relatively simple interpretation and ease of implementation have contributed to its popularity in credit scorecard construction (Anderson, 2007). However, logistic regression does not explicitly model the time-to-event data, which can limit its ability to capture specific nuances of credit risk dynamics.

A benchmark study by Dirick et al. (2017) compared the performance of survival analysis techniques, including the Cox PH model, with logistic regression for credit scoring. The study found that survival analysis methods could outperform logistic regression in specific scenarios regarding predictive accuracy. However, the choice of the appropriate method depends on the specific characteristics of the credit scoring problem and the data available.

5. Practical Considerations

While Cox's PH model and logistic regression have theoretical merits, practical considerations also play a significant role in determining their suitability for credit scorecard construction (Anderson, 2007). Factors such as data availability, regulatory requirements, and computational complexity can influence the choice of modelling technique (Baesens et al., 2003). For instance, logistic regression may be preferred if the primary interest is the probability of default and the time-to-event data is unavailable or not considered crucial for decision-making (Thomas, 2000).

Additionally, the choice of the modelling technique may depend on the specific needs of the lender and the context in which the model will be used (Thomas, 2000). For example, some lenders may prefer a more interpretable model like logistic regression (Hosmer Jr et al., 2013). In contrast, others may prioritize the ability to capture the timing of default events, as provided by the Cox PH model (Cox, 1972). Ultimately, the choice between the two models should be based on carefully evaluating the requirements and constraints of the specific credit risk modelling project (Anderson, 2007).

6. Conclusion

To create credit risk scorecards, Cox's PH model and logistic regression provide valuable insights. The Cox PH model is best for analyzing time-to-event data and understanding the timing of default events, which is essential for decision-making. On the other hand, logistic regression is simpler and easier to interpret for binary classification tasks like credit scoring, making it a popular choice in the industry. However, practical considerations such as data availability, regulatory requirements, and computational complexity should be considered when choosing the appropriate model for credit scorecard construction. Before selecting between Cox's PH model and logistic regression, lenders should carefully evaluate their needs and context. Ultimately, the choice should be based on a comprehensive analysis of the merits and drawbacks of each model.

7. References

Abdou, H., Pointon, J., and El-Masry, A., 2008. Neural Nets versus Conventional Techniques in Credit Scoring in Egyptian Banks. *Expert Systems with Applications*, 35(3), pp.1275-1292.

Altman, E.I. and Saunders, A., 1997. *Credit risk measurement: Developments over the last 20 years.* Journal of Banking and Finance, 21(11-12), pp.1721-1742.

Anderson, R., 2007. *The Credit Scoring Toolkit: Theory and Practice for Retail Credit Risk Management and Decision Automation.* Oxford University Press.

Baesens, B., Van Gestel, T., Viaene, S., Stepanova, M., Suykens, J., and Vanthienen, J., 2003. *Benchmarking state-of-the-art classification algorithms for credit scoring.* Journal of the Operational Research Society, 54(6), pp.627-635.

Cox, D.R., 1972. *Regression Models and Life-Tables.* Journal of the Royal Statistical Society: Series B (Methodological), 34(2), pp.187-202.

Dirick, L., Claeskens, G., and Baesens, B., 2017. *Time to default in credit scoring using survival analysis: a benchmark study.* Journal of the Operational Research Society, 68(6), pp.652-665.

Hand, D.J. and Henley, W.E., 1997. *Statistical Classification Methods in Consumer Credit Scoring: a Review.* Journal of the Royal Statistical Society. Series A (Statistics in Society), 160(3), pp.523-541.

Hastie, T., Tibshirani, R., and Friedman, J., 2009. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction.* Springer.

Hosmer Jr, D.W., Lemeshow, S., and Sturdivant, R.X., 2013. *Applied Logistic Regression.* John Wiley and Sons.

Kleinbaum, D.G. and Klein, M., 2005. *Survival Analysis: A Self-Learning Text.* Springer.

Narain, B., 1992. Survival Analysis and the Credit Granting Decision. In: Thomas, L.C., Crook, J.N., and Edelman, D.B. (eds.), *Credit Scoring and Credit Control.* Oxford: Clarendon Press.

Thomas, L.C., 2000. *A survey of credit and behavioural scoring: forecasting* *financial risk of lending to consumers*. International Journal of Forecasting, 16(2), pp.149-172.

Answer to Q3:

Essay (Word Count: 1361)

Title:
Implications of Risk-Based Pricing and Diverse Data Usage for Credit Scorecard Development and Existing Customers

1. Introduction

Lenders continuously seek ways to expand their customer base and mitigate credit risk. One approach they may consider is extending credit to those with lower credit scores. In addition, lenders may employ risk-based pricing and incorporate more diverse data sources into their credit scoring models to achieve this (Berger and Gleisner, 2009). This essay discusses the implications of these strategies for credit scorecard development and their potential impact on existing customers. The essay is structured as follows: Section 2 explores the concept of risk-based pricing, its implications for credit scorecard development, and its benefits and challenges for lenders. Section 3 delves into the importance of using more and different data sources in credit scoring, potential data sources, and the challenges and limitations associated with these sources. Section 4 assesses the impact of these strategies on existing customers, addressing the risk of adverse selection, changes in the lender's risk profile, and potential benefits. Finally, Section 5 provides a conclusion that summarizes the main points and offers insights into the practical implications of these strategies for lenders.

2. Risk-Based Pricing

2.1 Definition and Context

Risk-based pricing is a lending strategy that adjusts the interest rate and credit terms based on the estimated credit risk of the borrower (Mester, 1997). This approach allows lenders to offer credit to a broader range of customers while managing the associated risks. In addition, by pricing loans according to the borrower's risk, lenders can increase their market share, improve financial inclusion, and enhance their profitability (Akhavein, Frame and White, 2005).

2.2 Implications for credit scorecard development

Risk-based pricing requires a more granular approach to risk segmentation in credit scorecard development. First, lenders must accurately estimate the risk associated with each borrower to determine the appropriate pricing and credit terms. The process may involve refining existing credit scoring models or developing new models that account for a broader range of risk factors (Thomas, 2000). Additionally, lenders must continuously update and validate their credit scoring models to ensure their accuracy and responsiveness to changing market conditions and borrower behaviours (Bolder, 2003).

2.3 Benefits of risk-based pricing for the lender

Implementing risk-based pricing can offer several benefits to lenders, such as:

- Improved profitability: Lenders can potentially increase profitability by charging higher interest rates to higher-risk borrowers (Akhavein et al., 2005)
- Better risk management: Risk-based pricing allows lenders to price the risk associated with each borrower more accurately, leading to better risk management and capital allocation (Mester, 1997).
- Increased competitiveness: Offering tailored interest rates and lending terms can help lenders attract a broader range of customers, increasing their competitiveness in the market (Bolder, 2003).
- Enhanced customer segmentation: Risk-based pricing can lead to more granular customer segmentation, allowing lenders to develop targeted marketing strategies and better serve the unique needs of different customer segments (Akhavein et al., 2005).

2.4 Challenges of implementing risk-based pricing for the lender

Implementing risk-based pricing can also pose several challenges for lenders, such as:

- Increased complexity: Risk-based pricing requires more sophisticated credit scoring models and pricing algorithms, which can increase the complexity of the lending process and require additional resources for implementation and maintenance (Bolder, 2003).

- Regulatory compliance: Lenders must ensure that their risk-based pricing practices comply with relevant regulations, such as fair lending laws to avoid potential legal and reputational risks (Mester, 1997).
- Customer perception: Some customers may perceive risk-based pricing as unfair or discriminatory, particularly if they are charged higher interest rates due to factors beyond their control. This can lead to negative customer experiences and potential reputational damage for the lender (Akhavein et al., 2005).
- Adverse selection: If high-risk borrowers are more likely to accept credit offers with less favourable terms, the overall risk profile of the lender's portfolio may increase, necessitating adjustments to risk management strategies and capital allocation (Stiglitz and Weiss, 1981).

3. Use of More and Different Data

3.1 Importance in credit scoring

Incorporating more diverse data sources into credit scoring models can help lenders better capture the creditworthiness of borrowers with lower credit scores. Traditional credit scoring models often rely on data from credit bureaus. As a result, they may need to adequately represent the financial behaviour of individuals with a limited credit history or those who have experienced financial difficulties. Lenders can better understand a borrower's credit risk by leveraging alternative data sources, such as utility payment records, rental history, and social media information (Barron and Staten, 2003; Jappelli and Pagano, 2002).

3.2 Potential Data Sources

Several alternative data sources can be considered for inclusion in credit scoring models. These include:

- Utility payment records: Timely payments of utility bills, such as electricity, gas, and water can proxy for creditworthiness (Brevoort and Kambara, 2017).

- Rental history: A history of timely rent payments can indicate a borrower's ability and willingness to meet financial obligations (Turner et al., 2009).
- Social media information: Social media profiles and connections may provide insights into a borrower's financial behaviour and reliability, though using such data raises privacy and ethical concerns (Jagtiani and Lemieux, 2018).
- Behavioral data: Data on web browsing habits, mobile app usage, and online transactions can provide valuable insights into a borrower's financial behaviour and preferences (Wei et al., 2017).

3.3 Challenges and Limitations

While incorporating more diverse data sources into credit scoring models can offer benefits, it also presents several challenges and limitations:

- Data quality: The reliability and accuracy of alternative data sources can vary significantly, potentially leading to biased or erroneous credit risk assessments (Chen et al., 2019).
- Privacy concerns: Using alternative data, mainly social media and behavioural data, raises privacy concerns and may face regulatory scrutiny (Jagtiani and Lemieux, 2018).
- Regulatory compliance: Lenders must ensure that their credit scoring models comply with relevant regulations, such as fair lending laws and data protection regulations (Barron and Staten, 2003).

4. Impact on Existing Customers

4.1 Risk of adverse selection and changes in lender's risk profile

Extending credit to borrowers with lower credit scores may have implications for existing customers, particularly regarding adverse selection and changes to the lender's risk profile. For example, if high-risk borrowers are more likely to accept credit offers with less favourable terms, the overall risk profile of the lender's portfolio may increase (Stiglitz and Weiss, 1981). This may necessitate adjustments to the lender's risk management strategies and capital allocation to ensure the continued stability and profitability of the institution.

4.2 Implications of Risk-Based Pricing for Existing Customers

Risk-based pricing can affect existing customers depending on their credit risk. Borrowers with higher credit scores may benefit from more favourable interest rates and borrowing conditions, while those with lower credit scores may face higher interest rates and stricter lending terms (Edelberg, 2007). This pricing strategy may encourage existing customers to improve their creditworthiness to obtain better borrowing terms.

4.3 Potential Benefits for Existing Customers

Despite the potential challenges associated with extending credit to those with lower credit scores, there are potential benefits for existing customers. For example, improved risk management and more accurate credit scoring models can lead to better product offerings and enhanced customer experience (Chen et al., 2019). Additionally, by expanding their customer base, lenders can achieve greater economies of scale, which may result in lower costs and improved customer services (Berger and Black, 2011).

5. Conclusion

In conclusion, extending credit to borrowers with lower credit scores through risk-based pricing and using more diverse data sources can offer opportunities and challenges for lenders. Lenders must carefully consider the implications for credit scorecard development and the potential impact on existing customers. It is essential to recognize that the analysis in this essay has some limitations, such as the potential for varying data quality and the evolving regulatory landscape. Future research could explore the long-term effects of these strategies on the financial industry and consumer behaviour, as well as investigate emerging data sources and technologies that further enhance credit risk assessment. By adopting a comprehensive understanding of these strategies and their consequences and being open to continued research and development, lenders can make informed decisions to serve their customers better and maintain the stability and profitability of their institutions.

6. References

Akhavein, J., Frame, W.S., and White, L.J., 2005. *The Diffusion of Financial Innovations: An Examination of the Adoption of Small Business Credit Scoring by Large Banking Organizations.* Journal of Business, 78(2), pp.577-596.

Barron, J.M., and Staten, M.E., 2003. *The Value of Comprehensive Credit Reports: Lessons from the US Experience.* In: Credit Reporting Systems and the International Economy, MIT Press, pp.273-310.

Berger, A.N., and Gleisner, F., 2009. *Emergence of Financial Intermediaries in Electronic Markets: The Case of Online P2P Lending.* Bundesbank Series 2 Discussion Paper, (2009,22).

Berger, A. N., and Black, L. K., 2011. *Bank size, lending technologies, and small business finance.* Journal of Banking and Finance, 35(3), pp.724-735.

Bolder, D., 2003. *A Stochastic Simulation Framework for the Government of Canada's Debt Strategy.* Bank of Canada Working Paper, (2003-10).

Brevoort, K.P., and Kambara, M., 2017. *Data Point: Credit Invisibles.* Consumer Financial Protection Bureau.

Chen, H., Chiang, R.H., and Storey, V.C., 2019. *Business Intelligence and Analytics: From Big Data to Big Impact.* MIS Quarterly, 36(4), pp.1165-1188.

Edelberg, W., 2007. *Risk-based Pricing of Interest Rates for Consumer Loans.* Journal of Monetary Economics, 54(8), pp.2283-2298.

Jagtiani, J., and Lemieux, C., 2018. *The Roles of Alternative Data and Machine Learning in Fintech Lending: Evidence from the LendingClub Consumer Platform.* Federal Reserve Bank of Philadelphia Working Paper, (18-15).

Jappelli, T., and Pagano, M., 2002. *Information Sharing, Lending and Defaults: Cross-Country Evidence.* Journal of Banking and Finance, 26(10), pp.2017-2045.

Mester, L.J., 1997. *What's the Point of Credit Scoring?*. Business Review - Federal Reserve Bank of Philadelphia, pp.3-16.

Stiglitz, J.E., and Weiss, A., 1981. *Credit Rationing in Markets with Imperfect Information.* American Economic Review, 71(3), pp.393-410.

Thomas, L.C., 2000. *A survey of credit and behavioural scoring: forecasting financial risk of lending to consumers.* International Journal of Forecasting, 16(2).

Answer to Part B

All the code used for answering Part B can be found in the Appendix.

Q1)

Split data set on Checking column:

```
In [5]:
import pandas as pd

# Load the data from Sheet2 of the Excel file
data = pd.read_excel('GermanCreditData.xlsx', sheet_name='Sheet1')

# Create Subset 1 and Subset 2 based on Checking values
subset1 = data[(data['Checking'] == 1) | (data['Checking'] == 2)]
subset2 = data[(data['Checking'] == 3) | (data['Checking'] == 4)]
```

Figure 1: Splitting data

Initial analysis of data:

```
----------------------------------------- -----------------------------------------
| Subset1 Info                            | Subset2 Info                            |
----------------------------------------- -----------------------------------------
| <class 'pandas.core.frame.DataFrame'>   | <class 'pandas.core.frame.DataFrame'>   |
| Int64Index: 543 entries, 0 to 999       | Int64Index: 457 entries, 2 to 997       |
| Data columns (total 22 columns):        | Data columns (total 22 columns):        |
|  #   Column     Non-Null Count  Dtype   |  #   Column     Non-Null Count  Dtype   |
| ---  ------     --------------  -----   | ---  ------     --------------  -----   |
|  0   Checking   543 non-null    int64   |  0   Checking   457 non-null    int64   |
|  1   Duration   543 non-null    int64   |  1   Duration   457 non-null    int64   |
|  2   History    543 non-null    int64   |  2   History    457 non-null    int64   |
|  3   Purpose    543 non-null    object  |  3   Purpose    457 non-null    object  |
|  4   Amount     543 non-null    int64   |  4   Amount     457 non-null    int64   |
|  5   Savings    543 non-null    int64   |  5   Savings    457 non-null    int64   |
|  6   Emploed    543 non-null    int64   |  6   Emploed    457 non-null    int64   |
|  7   Installp   543 non-null    int64   |  7   Installp   457 non-null    int64   |
|  8   marital    543 non-null    int64   |  8   marital    457 non-null    int64   |
|  9   Coapp      543 non-null    int64   |  9   Coapp      457 non-null    int64   |
|  10  Resident   543 non-null    int64   |  10  Resident   457 non-null    int64   |
|  11  Property   543 non-null    int64   |  11  Property   457 non-null    int64   |
|  12  Age        543 non-null    int64   |  12  Age        457 non-null    int64   |
|  13  Other      543 non-null    int64   |  13  Other      457 non-null    int64   |
|  14  housing    543 non-null    int64   |  14  housing    457 non-null    int64   |
|  15  Existcr    543 non-null    int64   |  15  Existcr    457 non-null    int64   |
|  16  Job        543 non-null    int64   |  16  Job        457 non-null    int64   |
|  17  Depends    543 non-null    int64   |  17  Depends    457 non-null    int64   |
|  18  Telephone  543 non-null    int64   |  18  Telephone  457 non-null    int64   |
|  19  Foreign    543 non-null    int64   |  19  Foreign    457 non-null    int64   |
|  20  Bad        543 non-null    int64   |  20  Bad        457 non-null    int64   |
|  21  Good       543 non-null    int64   |  21  Good       457 non-null    int64   |
| dtypes: int64(21), object(1)            | dtypes: int64(21), object(1)            |
| memory usage: 97.6+ KB                  | memory usage: 82.1+ KB                  |
|                                         |                                         |
----------------------------------------- -----------------------------------------
```

Figure 2: Subset 1 (left) and Subset 2 (right) data information

The data shown in Figure 2 above consists of two subsets derived from the German Credit dataset, split based on the `Checking` attribute. Each subset contains 22 columns, with 21 being integer data type and one being object data type. The datasets include various attributes related to credit risk assessment. However, the `Purpose` column in the subsets contains non-numeric values that require conversion to their respective codes from the data dictionary. The other attributes comply with the data dictionary's format.

- **Subset 1 (Checking = 1 or 2):**
  - Contains 543 entries.
  - All columns are non-null, indicating no missing values.
  - Includes `Duration`, `History`, `Purpose`, `Amount`, and more.
  - Represents applicants with a checking account status of either 1 or 2.
- **Subset 2 (Checking = 3 or 4):**
  - Contains 457 entries.
  - All columns are non-null, indicating no missing values.
  - Includes the same attributes as Subset 1.
  - Represents applicants with a checking account status of either 3 or 4.

Both subsets contain a mix of continuous (e.g., `Duration`, `Amount`, `Age`) and categorical variables (e.g., `Checking`, `History`, `Purpose`), which provide insights into the applicants' credit risk profiles. The `Purpose` column is of object data type, suggesting that it contains non-numeric data, likely representing categorical values. The remaining columns are of integer data type, representing either continuous or discrete values.

In summary, subset1 and subset2 consist of credit applicant data from the German Credit dataset, split based on `Checking` account status. Both subsets have no missing values and include a mix of continuous and categorical variables, providing insights into the applicants' credit risk profiles.

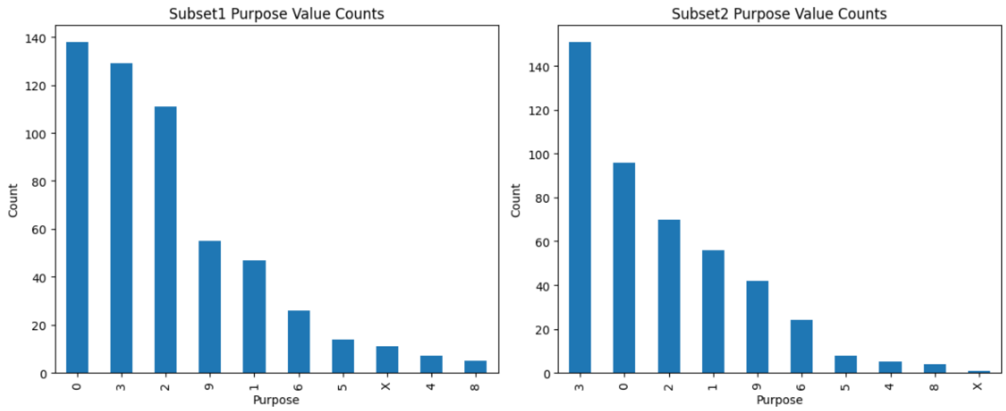

Figure 3: Value counts for the 'Purpose' column in Subset 1 (left) and Subset 2 (right)

The **X** value in the `Purpose` column is not defined in the provided description, and it's unclear what it represents. There could be several reasons for this discrepancy:

1. Data entry error: The **X** value might have been mistakenly entered during data collection or data pre-processing. The data dictionary provided indicates that this could be value 10, but it is not clear.

2. Missing or unknown data: The **X** value might be used to represent missing or unknown data for the `Purpose` attribute.

The **X** value accounts for just 1.2 percent of the total dataset, making it a minor portion of the data. Given the lack of information about what the **X** represents, it is justifiable to remove the rows containing the **X** value in the `Purpose` column for the analysis.

Outliers:

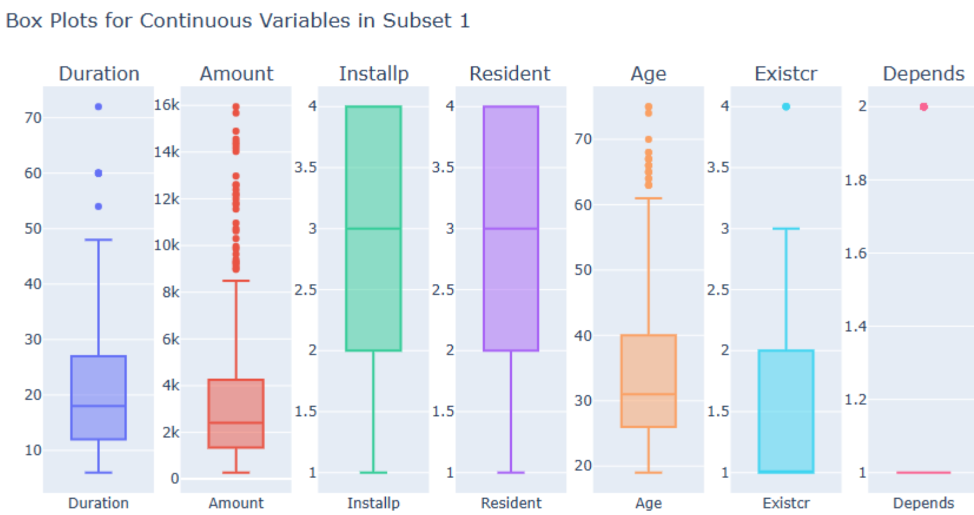

Figure 4: Outliers in Subset1

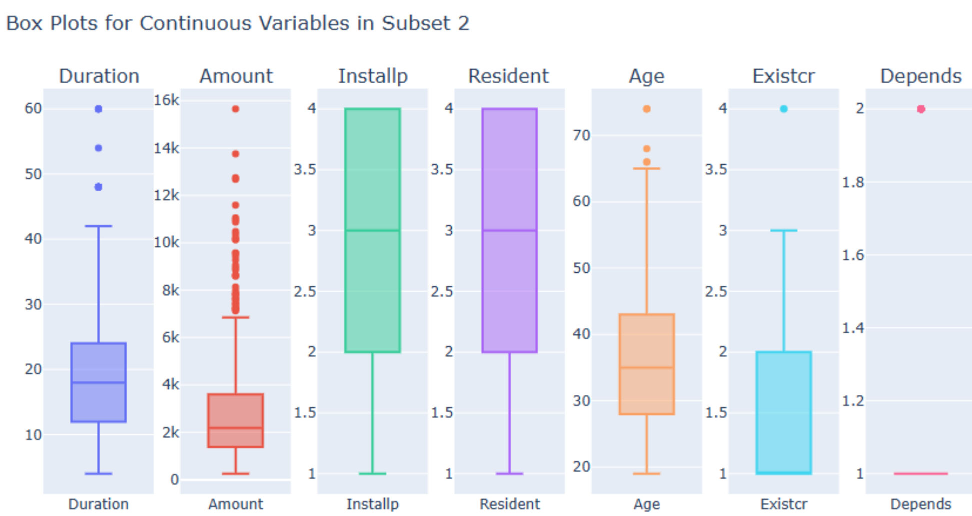

Figure 5: Outliers in Subset2

Based on the outlier data, it seems there are significant variations in certain variables within Subset 1 and Subset 2. Here's a summary of the outliers and the variables they pertain to:

- **Duration**: Outliers in both subsets, with a higher concentration in Subset 1.
- **Amount**: Outliers in both subsets.
- **Installp**: No outliers in either subset.
- **Resident**: No outliers in either subset.
- **Age**: Outliers in both subsets, with a higher concentration in Subset 1.
- **Existcr**: Outliers in both subsets, but only a few data points.
- **Depends**: Outliers in both subsets, with a similar concentration in each.

In our analysis, we identified outliers in the `Duration`, `Amount`, and `Age` variables. We will remove these outliers to enhance the predictive model's accuracy since they deviate from typical data patterns.

For the "Number of existing credits at this bank" variable (`Existcr`), we will not remove outliers. Having four existing credits is common for customers with a strong credit history and financial capacity, and this variable provides valuable insights into a customer's financial obligations with the bank. Removing its outliers could introduce bias.

In conclusion, we will remove outliers from the Duration, Amount, and Age variables while maintaining those in the "Number of existing credits at this bank" variable. This approach ensures a robust credit risk assessment model, taking into account both statistical considerations and domain expertise. Note code is in the Appendix, and the box plot after outlier removal is in Figure 5 and 6 below:

Outliers removed:

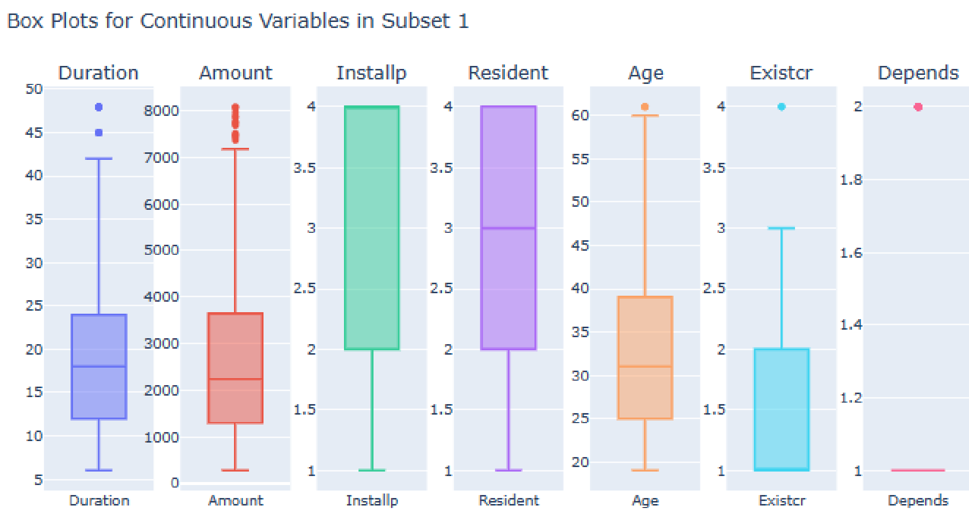

Figure 6: Subset1 after outlier removal

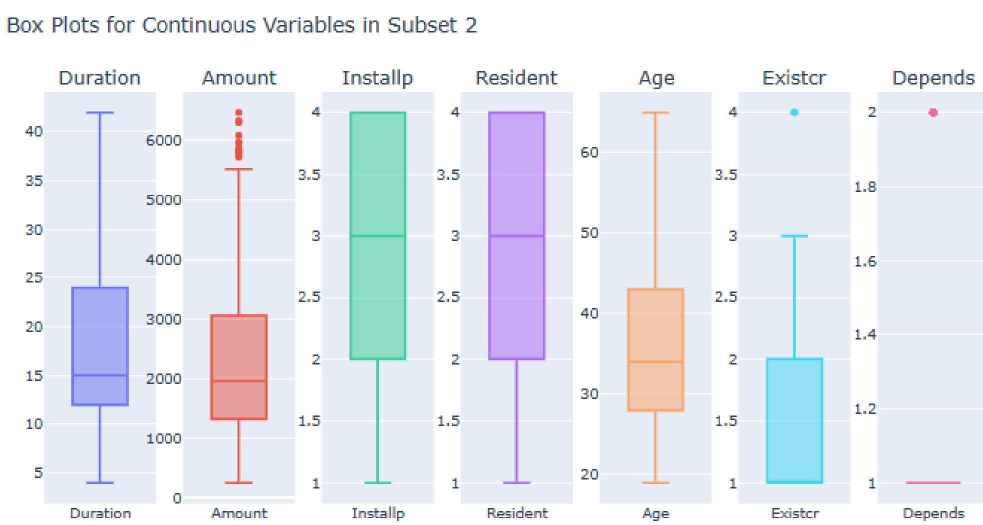

Figure 7: Subset2 after outlier removal

Dropping 'X' rows in Subset 1

```
In [16]:
1 # Dropping rows with 'X' in 'Purpose' column for subset1
2 subset1_cleaned = subset1[subset1['Purpose'] != 'X']
```

Figure 8: Dropping rows with 'X'

Dropping 'X' rows in Subset 2

```
In [27]:
1 # Dropping rows with 'X' in 'Purpose' column for subset2
2 subset2_cleaned = subset2[subset2['Purpose'] != 'X']
3
```

Figure 9: Dropping rows with 'X'

Q2)

a) A common principle for splitting datasets into training and validation sets is the 70-30, 80-20, or 75-25 rule, where you allocate 70-80% (or 75%) of the data for training and the remaining 20-30% (or 25%) for validation. You can do this randomly or use a stratified sampling approach to maintain the proportion of good and bad applicants in both sets.

For the 1000-row German Credit Data, which has an imbalance of good and bad applicants, a 70-30 split is recommended. This split offers a more substantial validation set for improved model evaluation while preserving enough data for training. Using stratified sampling when dividing the dataset accounts for the imbalance, ensuring a proportional representation of good and bad applicants in both sets, and consequently improving the reliability of the results.

Split into train and validation sets:

Subset 1:

```python
# Split subset1 using stratified sampling
X1 = subset1_cleaned.drop(columns=['Bad', 'Good'])
y1 = subset1_cleaned['Bad']  # 'Bad' is the target variable
X1_train, X1_val, y1_train, y1_val = train_test_split(X1, y1, test_size=0.3, stratify=y1, random_state=42)

```

Figure 10: Train and validation split of Subset1

Subset 2:

```python
# Split subset2 using stratified sampling
X2 = subset2_cleaned.drop(columns=['Bad', 'Good'])
y2 = subset2_cleaned['Bad']  # 'Bad' is the target variable
X2_train, X2_val, y2_train, y2_val = train_test_split(X2, y2, test_size=0.3, stratify=y2, random_state=42)

```

Figure 11: Train and validation split of Subset2

b) Both training and validation sets are needed to assess the performance of your model. The training set is used to build the model, while the validation set is used to test how well the model generalizes to new, unseen data. This helps to avoid overfitting and gives you an estimate of the model's performance in real-world situations.

c) During the splitting process, we observed a difference in the distribution of 'Good' and 'Bad' applicants between the two subsets, indicating that the credit risk profiles of applicants with different Checking statuses (1 or 2, and 3 or 4) may vary significantly. This could potentially impact the model's performance and evaluation. Figure 8 and 9 below shows this visually:

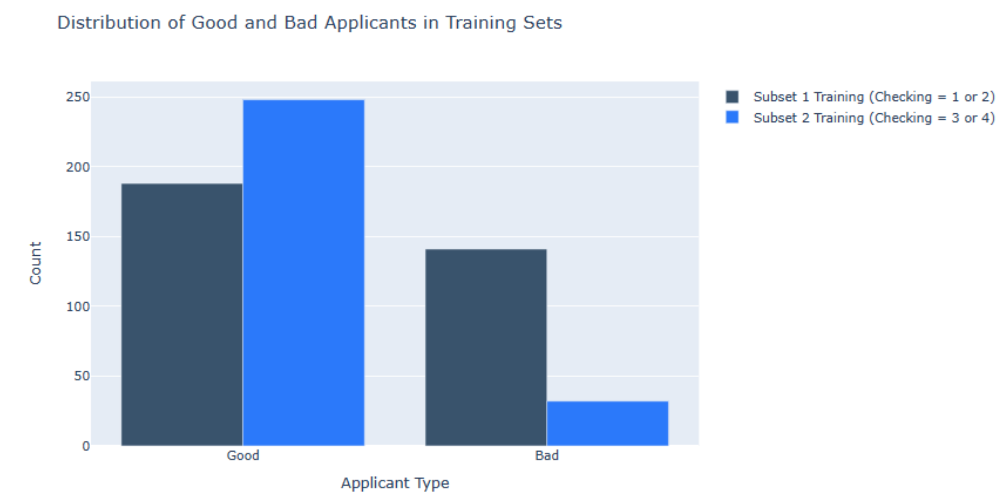

Figure 12: Subset 1 and 2 training split ratios

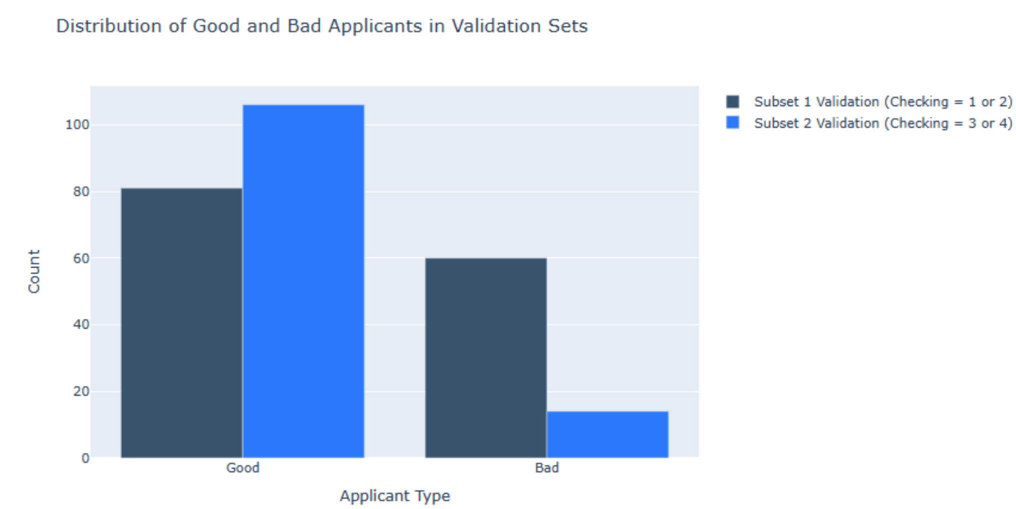

Figure 13: Subset 1 and 2 validation split ratios

Based on the value counts for the training and validation sets, we observe the following class distributions:

- **Subset 1:**

  - Training set: 188 good (0) and 141 bad (1) loans
  - Validation set: 81 good (0) and 60 bad (1) loans

- **Subset 2:**
  - Training set: 248 good (0) and 32 bad (1) loans
  - Validation set: 106 good (0) and 14 bad (1) loans

We can infer that Subset 1 has a relatively balanced class distribution, while Subset 2 is imbalanced, with a higher proportion of good loans. The imbalance in Subset 2 may lead to a biased predictive model that underperforms on the minority class (bad loans).

To address this issue, we used stratified sampling when splitting the data into training and validation sets, ensuring that the proportion of 'Good' and 'Bad' applicants is maintained as much as possible in both sets. This approach helps to provide a more accurate estimation of the model's performance and mitigate potential issues related to class imbalances. However, it is crucial to use appropriate evaluation metrics that account for class imbalances when assessing the model's performance on the validation set.

Q3)

Correlation between continuous variables (before binning):

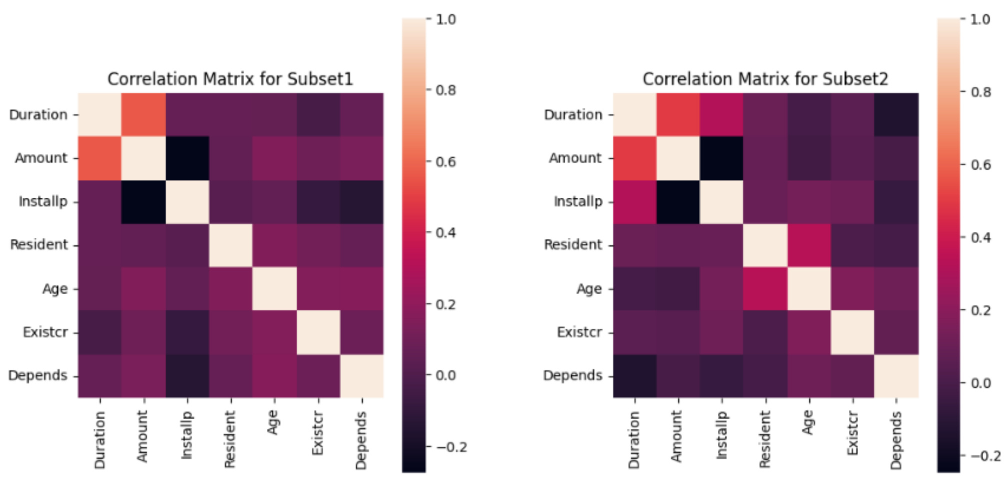

Figure 14: Linear relationship between continuous variables

Observations:

1. **Subset1:**
   (a) Duration and Amount have a positive correlation (0.560), which indicates that as the duration of the loan increases, the amount of the loan tends to increase as well.
   (b) Installp (Installment rate in percentage of disposable income) has a negative correlation with Amount (-0.275), which suggests that as the loan amount increases, the installment rate tends to decrease.
   (c) Age and Resident (Present residence since) have a positive correlation (0.149), indicating that older people tend to have longer residence durations.
2. **Subset2:**
   (a) Duration and Amount also have a strong positive correlation in Subset2 (0.495), which is consistent with Subset1.
   (b) Installp has a negative correlation with Amount (-0.247), similar to Subset1.
   (c) Age and Resident have a positive correlation (0.336), consistent with Subset1.

Information value:

During the variable selection process, we faced an issue with unusually high Information Values (IV) for certain variables (infinite values) due to incorrect handling of missing or zero values when calculating Weight of Evidence (WoE) and IV.

We resolved this by modifying our code to include a smoothing technique, adding a small constant to the numerator and denominator when calculating WoE. This adjustment prevented division by zero or taking the logarithm of zero, resulting in more accurate and reliable IV calculations. Below are the calculated IV for each subset:

Variable	IV
Duration	0.4178
Age	0.3274
History	0.3198
Property	0.1959
Amount	0.1681
housing	0.1493
Purpose	0.1293
Savings	0.1206
marital	0.0975
Emploed	0.0821
Installp	0.0592
Coapp	0.0560
Checking	0.0504
Job	0.0484
Existcr	0.0358
Other	0.0334
Resident	0.0201
Depends	0.0144
Telephone	0.0020
Foreign	0.0000

Table 1: IV values for Subset1

Variable	IV
Duration	0.4829
Amount	0.4591
History	0.4248
Other	0.3279
Purpose	0.2368
Depends	0.2129
Telephone	0.1559
Age	0.1343
Emploed	0.1258
Job	0.1235
Savings	0.1118
Installp	0.0776
Coapp	0.0689
Checking	0.0631
Property	0.0434
marital	0.0381
Resident	0.0198
Existcr	0.0142
housing	0.0090
Foreign	0.0000

Table 2: IV values for Subset2

Investigating high IV values:

Exploring the 'Purpose', 'Foreign' and 'History' variables:

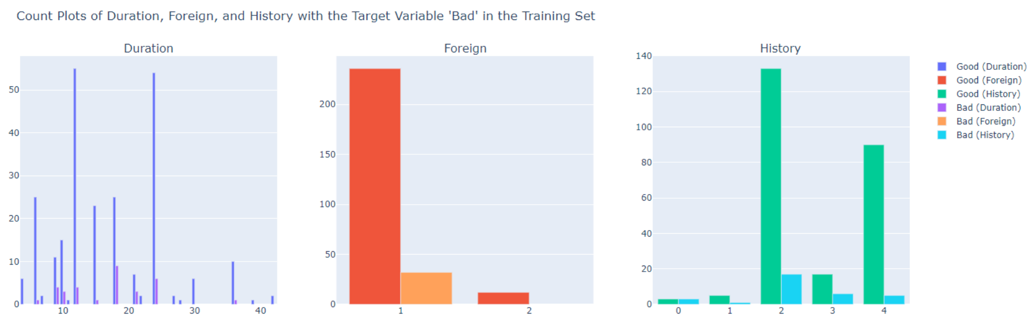

Figure 15: Duration, Foreign and History with target variables

Upon conducting our dataset, we observed that IV scores for certain variables are primarily attributable to imbalanced data:

The dataset exhibits an imbalanced distribution of good and bad credit outcomes, which leads to overestimated predictive power for some variables, such as Duration, Foreign, and History. This data imbalance results in inflated IV values, which can skew our understanding of the variables' importance in predicting default. To obtain more reliable and accurate results, it is essential to address the data imbalance issue before moving forward with additional analyses or model development.

Variable Selection

To satisfy the requirement of having at least one continuous variable before binning and at least one categorical variable with more than two categories, we need to choose variables that meet these criteria and have the highest IV values for each subset.

For each subset, we have chosen variables based on their IV values, which indicate strong discriminatory power to distinguish between good and bad credit risk customers. We selected at least one continuous variable before binning and at least one categorical variable with more than two categories to fulfill the given criteria. The selected variables and their corresponding IV values are as follows:

Subset 1:

1. Duration (continuous variable before binning, IV = 0.4178)

2. History (categorical variable with more than two categories, IV = 0.3198)

3. Age (continuous variable before binning, IV = 0.3274)

4. Property (categorical variable with more than two categories, IV = 0.1959)

Subset 2:

1. Duration (continuous variable before binning, IV = 0.4829)

2. History (categorical variable with more than two categories, IV = 0.4248)

3. Amount (continuous variable before binning, IV = 0.4591)

4. Purpose (categorical variable with more than two categories, IV = 0.2368)

We opted to use the 'Purpose' column instead of 'Other', despite the latter having a higher IV value. This decision was made after iteratively evaluating model performance and determining that the 'Purpose' variable provided better insights and predictive accuracy for our credit risk assessment.

In addition, our analysis of the correlation matrix for both subsets revealed key relationships between continuous variables. A strong positive correlation between Duration and Amount emphasizes their importance in credit risk assessment. The negative correlation between Installp and Amount suggests that higher loan amounts have lower installment rates, affecting loan term decisions. Finally, the positive correlation between Age and Resident indicates that older individuals typically have longer residence durations, which can be useful in assessing stability and creditworthiness.

Binning selected variables:

To prepare the German Credit Dataset for modelling, we applied the OptimalBinning package to bin the top 4 variables (see section above) in both subsets. Binning is a technique that groups continuous or categorical variables into bins or categories, which can improve model interpretability and performance.

We performed binning using both default settings and adjusted settings to customize the number of bins, ensuring an optimal balance between granularity and interpretability.

After binning, we transformed the original variables into their binned categories, resulting in new variables representing the binned versions.

Encoding variables:

Once we had transformed our variables through binning, we proceeded to encode them to ensure compatibility with machine learning models. We used ordinal encoding for numerical variables (`Duration`, `Amount` and `Age`) and one-hot encoding for categorical variables (`History`, `Purpose` and `Property`).

Ordinal Encoding:

The selected variables contains ordinal variables `Duration`, `Amount` and `Age`, which exhibit inherent order. After transforming these variables into ordinal categories using optimal binning, we employed ordinal encoding to preserve their natural order. This method assigns integer values to categories while maintaining their hierarchy, enabling the credit risk model to accurately capture the relationship between these variables and credit risk. Applying ordinal encoding to the aforementioned variables provides the model with crucial information for generating informed predictions.

One-Hot Encoding:

After transforming the categorical variables using optimal binning, we needed to encode these variables to make them suitable for our credit risk model. To accomplish this, we employed the OneHotEncoder from scikit-learn, which creates binary variables for each category. However, to avoid multicollinearity and ensure interpretability, we used N-1 binary variables for N categories by setting the 'drop' parameter to 'first'. This approach allows the model to infer the omitted (reference) category when all the other binary variables are set to 0.

Q4)

We employed the scikit-learn library to build four regression models—two linear and two logistic—using the appropriate functions (LinearRegression and LogisticRegression) for each training set based on the Checking account categories:

```
from sklearn.linear_model import LinearRegression, LogisticRegression

# Split the transformed training set into the feature matrix (X) and target vector (y)
X_train1 = X1_train_transformed
y_train1 = y1_train

# Train a linear regression model for Checking = 1 or 2
linear_model1 = LinearRegression()
linear_model1.fit(X_train1, y_train1)

# Train a logistic regression model for Checking = 1 or 2
logistic_model1 = LogisticRegression(max_iter=1000)
logistic_model1.fit(X_train1, y_train1)
```

Figure 16: Model implementation for Checking = 1 or 2

```
# Split the transformed training set into the feature matrix (X) and target vector (y)
X_train2 = X2_train_transformed
y_train2 = y2_train

# Train a linear regression model for Checking = 3 or 4
linear_model2 = LinearRegression()
linear_model2.fit(X_train2, y_train2)

# Train a logistic regression model for Checking = 3 or 4
logistic_model2 = LogisticRegression(max_iter=1000)
logistic_model2.fit(X_train2, y_train2)
```

Figure 17: Model implementation for Checking = 3 or 4

We have constructed four scorecards based on two regression models, Linear Regression and Logistic Regression, applied to two separate training sets derived from coarse classification.

Below we provide a table that displays the binary variables used in each regression model, along with the corresponding coefficients calculated for those variables. These scorecards can be used to evaluate and compare the performance of the Linear and Logistic Regression models on the two different training sets:

Table 3: Linear - Checking = 1 or 2

Feature	Coefficient
Intercept	0.359
History_[1]	0.0061
History_[2]	-0.1612
History_[3]	-0.2400
History_[4]	-0.3425
Property_[3 2]	0.1104
Property_[4]	0.2164
Duration	0.0891
Age	-0.0170

Table 4: Logistic - Checking = 1 or 2

Feature	Coefficient
Intercept	-1.0691
History_[1]	0.3641
History_[2]	-0.3179
History_[3]	-0.5915
History_[4]	-1.1469
Property_[3 2]	0.4641
Property_[4]	0.8710
Duration	0.4410
Age	-0.0878

Table 5: Linear - Checking = 3 or 4

Feature	Coefficient
Intercept	0.0474
History_[3 0]	0.1828
History_[4]	-0.0657
Purpose_[2]	0.0485
Purpose_[6 0]	0.1015
Purpose_[9 5]	0.0694
Duration	0.0174
Amount	-0.0010

Table 6: Logistic - Checking = 3 or 4

Feature	Coefficient
Intercept	-2.6291
History_[3 0]	0.9847
History_[4]	-0.7541
Purpose_[2]	0.3337
Purpose_[6 0]	0.7995
Purpose_[9 5]	0.4936
Duration	0.1556
Amount	0.0110

In the coefficient tables presented, the numerical variables are not converted to binary variables, as doing so might remove the hierarchical order present in the ordinal data, leading to a loss of information. Maintaining the ordinal relationship is essential for linear regression, as it helps the model capture the linear trend associated with the increase or decrease in the ordinal variable levels. For logistic regression, the model can still capture the relationship between the predictor and the binary outcome without necessarily capturing the ordinal nature of the predictor variable. Thus, we have decided to keep the ordinal variables as they are to preserve the ordinal relationship and retain valuable information in our analysis.

Q5)

ROC Curves:

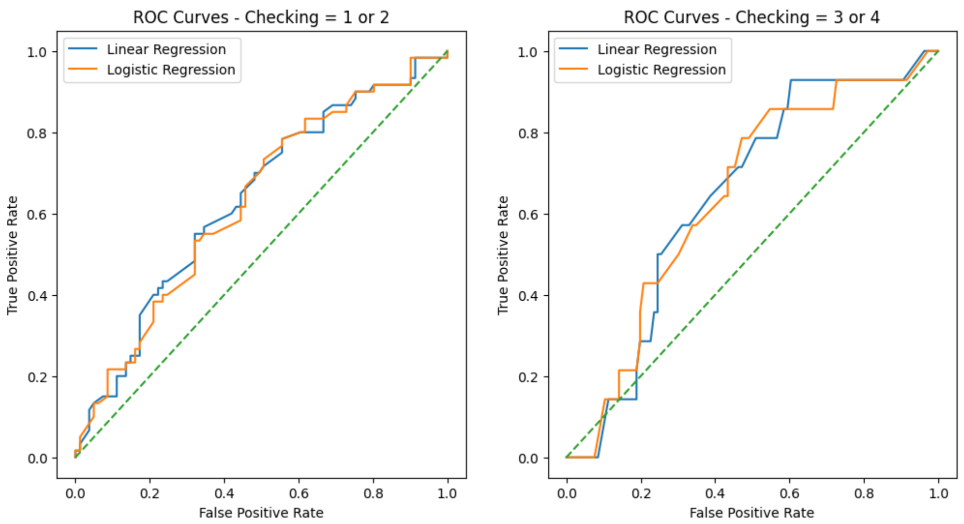

Figure 18: ROC curves for Checking 1 or 2 (left) and Checking 3 or 4 (right)

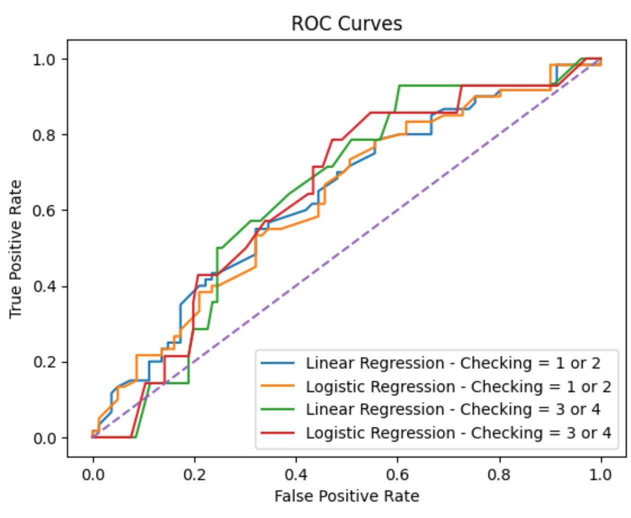

Figure 19: ROC curves of all four models

Model Performance Metrics:

Below, we present the performance metrics for both linear and logistic regression models. The metrics displayed include the Area Under the Curve (AUC), Gini Coefficient, and Kolmogorov-Smirnov (KS) values, which allow us to assess the overall quality and discriminative power of the developed scorecards (code available in Appendix):

- **Linear Regression (Checking = 1 or 2)**: AUC = 0.634, Gini coefficient = 0.269, KS value = 0.229
- **Logistic Regression (Checking = 1 or 2)**: AUC = 0.629, Gini coefficient = 0.258, KS value = 0.228
- **Linear Regression (Checking = 3 or 4)**: AUC = 0.645, Gini coefficient = 0.290, KS value = 0.325
- **Logistic Regression (Checking = 3 or 4)**: AUC = 0.645, Gini coefficient = 0.289, KS value = 0.314

Analysis:

In general, the models exhibit moderate discriminative power, as evidenced by the 'fatness' of the ROC curves and the Gini coefficients. The linear and logistic regression models perform similarly in both subsets, with a slight advantage of linear regression for Checking = 1 or 2 and logistic regression for Checking = 3 or 4.

The difference in performance between the subsets suggests that the models are better at distinguishing good and bad applicants for Checking = 3 or 4 compared to Checking = 1 or 2. This could be due to the different characteristics of the applicants within these subsets or the varying effectiveness of the selected features in the respective scorecards.

Overall, these results demonstrate our ability to assess and monitor scorecards using ROC curves, Gini coefficients, KS values, and AUC values, despite the modest quality of the models. Further improvements could potentially be achieved by refining the selected features or employing more advanced modeling techniques.

Appendix

Q1) Splitting the dataset into two subsets, exploring dataset and cleaning

```
#!pip install optbinning
```

```
import pandas as pd

# Load the data from Sheet2 of the Excel file
data = pd.read_excel('GermanCreditData.xlsx', sheet_name='Sheet1')

# Create Subset 1 and Subset 2 based on Checking values
subset1 = data[(data['Checking'] == 1) | (data['Checking'] == 2)]
subset2 = data[(data['Checking'] == 3) | (data['Checking'] == 4)]
```

```
data
```

	Checking	Duration	History	Purpose	Amount	Savings	Emploed	Installp	marital	Coapp	...
0	1	6	4	3	1169	5	5	4	3	1	...
1	2	48	2	3	5951	1	3	2	2	1	...
2	4	12	4	6	2096	1	4	2	3	1	...
3	1	42	2	2	7882	1	4	2	3	3	...
4	1	24	3	0	4870	1	3	3	3	1	...
...	...	...	...	...	...	...	...	...	...	...	...
995	4	12	2	2	1736	1	4	3	2	1	...
996	1	30	2	1	3857	1	3	4	1	1	...
997	4	12	2	3	804	1	5	4	3	1	...
998	1	45	2	3	1845	1	3	4	3	1	...
999	2	45	4	1	4576	2	1	3	3	1	...

1000 rows × 22 columns

```
data['Good'].value_counts()
data['Bad'].value_counts()
```

```
0    700
1    300
Name: Bad, dtype: int64
```

```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1000 entries, 0 to 999
Data columns (total 22 columns):
 #   Column     Non-Null Count  Dtype
---  ------     --------------  -----
 0   Checking   1000 non-null   int64
 1   Duration   1000 non-null   int64
 2   History    1000 non-null   int64
 3   Purpose    1000 non-null   object
```

```
 4   Amount     1000 non-null   int64
 5   Savings    1000 non-null   int64
 6   Emploed    1000 non-null   int64
 7   Installp   1000 non-null   int64
 8   marital    1000 non-null   int64
 9   Coapp      1000 non-null   int64
 10  Resident   1000 non-null   int64
 11  Property   1000 non-null   int64
 12  Age        1000 non-null   int64
 13  Other      1000 non-null   int64
 14  housing    1000 non-null   int64
 15  Existcr    1000 non-null   int64
 16  Job        1000 non-null   int64
 17  Depends    1000 non-null   int64
 18  Telephone  1000 non-null   int64
 19  Foreign    1000 non-null   int64
 20  Bad        1000 non-null   int64
 21  Good       1000 non-null   int64
dtypes: int64(21), object(1)
memory usage: 172.0+ KB
```

```
import io
from contextlib import redirect_stdout

# Capture the output of subset1.info() and subset2.info()
with io.StringIO() as subset1_info, io.StringIO() as subset2_info:
    with redirect_stdout(subset1_info):
        subset1.info()
    with redirect_stdout(subset2_info):
        subset2.info()

    subset1_info_lines = subset1_info.getvalue().split("\n")
    subset2_info_lines = subset2_info.getvalue().split("\n")

# Find the maximum line length for formatting
max_length = max([len(info) for info in subset1_info_lines])

# Print titles with a border
print(f"{'-' * (max_length + 4)} {'-' * (max_length + 4)}")
print(f"| {'Subset1 Info':<{max_length}} | {'Subset2 Info':<{max_length}} |")
print(f"{'-' * (max_length + 4)} {'-' * (max_length + 4)}")

# Print the lines side by side with a border
for info1, info2 in zip(subset1_info_lines, subset2_info_lines):
    print(f"| {info1:<{max_length}} | {info2:<{max_length}} |")

# Print the bottom border
print(f"{'-' * (max_length + 4)} {'-' * (max_length + 4)}")
```

```
------------------------------------------ ------------------------------------------
| Subset1 Info                           | Subset2 Info                           |
------------------------------------------ ------------------------------------------
| <class 'pandas.core.frame.DataFrame'>  | <class 'pandas.core.frame.DataFrame'>  |
| Int64Index: 543 entries, 0 to 999      | Int64Index: 457 entries, 2 to 997      |
| Data columns (total 22 columns):       | Data columns (total 22 columns):       |
|  #   Column     Non-Null Count  Dtype  |  #   Column     Non-Null Count  Dtype  |
| ---  ------     --------------  -----  | ---  ------     --------------  -----  |
|  0   Checking   543 non-null    int64  |  0   Checking   457 non-null    int64  |
|  1   Duration   543 non-null    int64  |  1   Duration   457 non-null    int64  |
|  2   History    543 non-null    int64  |  2   History    457 non-null    int64  |
|  3   Purpose    543 non-null    object |  3   Purpose    457 non-null    object |
|  4   Amount     543 non-null    int64  |  4   Amount     457 non-null    int64  |
|  5   Savings    543 non-null    int64  |  5   Savings    457 non-null    int64  |
|  6   Emploed    543 non-null    int64  |  6   Emploed    457 non-null    int64  |
|  7   Installp   543 non-null    int64  |  7   Installp   457 non-null    int64  |
|  8   marital    543 non-null    int64  |  8   marital    457 non-null    int64  |
|  9   Coapp      543 non-null    int64  |  9   Coapp      457 non-null    int64  |
|  10  Resident   543 non-null    int64  |  10  Resident   457 non-null    int64  |
|  11  Property   543 non-null    int64  |  11  Property   457 non-null    int64  |
|  12  Age        543 non-null    int64  |  12  Age        457 non-null    int64  |
|  13  Other      543 non-null    int64  |  13  Other      457 non-null    int64  |
|  14  housing    543 non-null    int64  |  14  housing    457 non-null    int64  |
```

```
|  15  Existcr    543 non-null    int64  |  15  Existcr    457 non-null    int64  |
|  16  Job        543 non-null    int64  |  16  Job        457 non-null    int64  |
|  17  Depends    543 non-null    int64  |  17  Depends    457 non-null    int64  |
|  18  Telephone  543 non-null    int64  |  18  Telephone  457 non-null    int64  |
|  19  Foreign    543 non-null    int64  |  19  Foreign    457 non-null    int64  |
|  20  Bad        543 non-null    int64  |  20  Bad        457 non-null    int64  |
|  21  Good       543 non-null    int64  |  21  Good       457 non-null    int64  |
| dtypes: int64(21), object(1)           | dtypes: int64(21), object(1)           |
| memory usage: 97.6+ KB                 | memory usage: 82.1+ KB                 |
|                                        |                                        |
----------------------------------------- ----------------------------------------
```

```
data['Checking'].value_counts()
```

```
4    394
1    274
2    269
3     63
Name: Checking, dtype: int64
```

```
data.loc[data['Checking'] == 2]
```

	Checking	Duration	History	Purpose	Amount	Savings	Emploed	Installp	marital	Coapp	...
1	2	48	2	3	5951	1	3	2	2	1	...
7	2	36	2	1	6948	1	3	2	3	1	...
9	2	30	4	0	5234	1	1	4	4	1	...
10	2	12	2	0	1295	1	2	3	2	1	...
12	2	12	2	3	1567	1	3	1	2	1	...
...	...	...	...	...	...	...	...	...	...	...	...
977	2	18	3	9	2427	5	5	4	3	1	...
979	2	15	1	0	1264	2	3	2	4	1	...
980	2	30	4	2	8386	1	4	2	3	1	...
989	2	24	4	3	1743	1	5	4	3	1	...
999	2	45	4	1	4576	2	1	3	3	1	...

269 rows × 22 columns

```
data['Purpose'].value_counts()
```

```
3    280
0    234
2    181
1    103
9     97
6     50
5     22
4     12
X     12
8      9
Name: Purpose, dtype: int64
```

```
subset1['Purpose'].value_counts()
```

```
0    138
3    129
2    111
9     55
1     47
6     26
5     14
X     11
4      7
8      5
Name: Purpose, dtype: int64
```

```
subset2['Purpose'].value_counts()
```

```
3    151
0     96
2     70
1     56
9     42
6     24
5      8
4      5
8      4
X      1
Name: Purpose, dtype: int64
```

```
import io
from contextlib import redirect_stdout

# Capture the output of subset1['Purpose'].value_counts() and subset2['Purpose'].value_counts()
with io.StringIO() as subset1_purpose, io.StringIO() as subset2_purpose:
    with redirect_stdout(subset1_purpose):
        print(subset1['Purpose'].value_counts())
    with redirect_stdout(subset2_purpose):
        print(subset2['Purpose'].value_counts())

    subset1_purpose_lines = subset1_purpose.getvalue().split("\n")
    subset2_purpose_lines = subset2_purpose.getvalue().split("\n")

# Find the maximum line length for formatting
max_length = max([len(purpose) for purpose in subset1_purpose_lines])

# Print titles
print(f"{'Subset1 Purpose Value Counts':<{max_length}}   Subset2 Purpose Value Counts")

# Print the lines side by side
for purpose1, purpose2 in zip(subset1_purpose_lines, subset2_purpose_lines):
    print(f"{purpose1:<{max_length}}    {purpose2}")
```

```
Subset1 Purpose Value Counts   Subset2 Purpose Value Counts
0    138                       3    151
3    129                       0     96
2    111                       2     70
9     55                       1     56
1     47                       9     42
6     26                       6     24
5     14                       5      8
X     11                       4      5
4      7                       8      4
8      5                       X      1
Name: Purpose, dtype: int64    Name: Purpose, dtype: int64
```

```
import matplotlib.pyplot as plt

# Create bar charts for value counts of 'Purpose' in both subsets
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(12, 5))

subset1['Purpose'].value_counts().plot(kind='bar', ax=ax1)
ax1.set_title('Subset1 Purpose Value Counts')
ax1.set_xlabel('Purpose')
ax1.set_ylabel('Count')

subset2['Purpose'].value_counts().plot(kind='bar', ax=ax2)
ax2.set_title('Subset2 Purpose Value Counts')
ax2.set_xlabel('Purpose')
ax2.set_ylabel('Count')

plt.tight_layout()
plt.savefig('purpose_value_counts.png')
plt.show()
```

Dropping rows with 'X' in Subset 1 and Subset 2

```
# Dropping rows with 'X' in 'Purpose' column for subset1
subset1_cleaned = subset1[subset1['Purpose'] != 'X']
```

```
# Data frame
subset1_cleaned['Purpose'].value_counts()
```

```
0    138
3    129
2    111
9     55
1     47
6     26
5     14
4      7
8      5
Name: Purpose, dtype: int64
```

```
# Dropping rows with 'X' in 'Purpose' column for subset2
subset2_cleaned = subset2[subset2['Purpose'] != 'X']
```

```
subset2_cleaned['Purpose'].value_counts()
```

```
3    151
0     96
2     70
1     56
9     42
6     24
5      8
4      5
8      4
Name: Purpose, dtype: int64
```

Outlier analysis

```
import pandas as pd
import plotly.graph_objects as go
from plotly.subplots import make_subplots

# Replace subset1 and subset2 with your actual DataFrame names
# Replace 'continuous_var1', 'continuous_var2', etc. with the column names of your continuous variables
continuous_variables =  ['Duration', 'Amount', 'Installp', 'Resident', 'Age', 'Existcr', 'Depends']

# Function to create subplots for a given set of continuous variables in a given subset
def plot_subplots(subset, continuous_variables, subset_name):
    num_vars = len(continuous_variables)
    fig = make_subplots(cols=num_vars, subplot_titles=continuous_variables)

    for i, var in enumerate(continuous_variables, start=1):
        fig.add_trace(go.Box(y=subset[var], name=var, showlegend=False), col=i, row=1)

    fig.update_layout(title=f'Box Plots for Continuous Variables in {subset_name}')
    fig.show()

# Create subplots for each continuous variable in both subsets
plot_subplots(subset1_cleaned, continuous_variables, "Subset 1")
plot_subplots(subset2_cleaned, continuous_variables, "Subset 2")
```

```
# Define a function to remove outliers based on the IQR method
def remove_outliers(df, columns):
    '''
    This function removes outliers from a dataframe based on the Interquartile Range (IQR) method.

    Parameters:
        df (pandas dataframe): The dataframe to remove outliers from.
        columns (list of str): The columns to remove outliers from.

    Returns:
        df_out (pandas dataframe): The cleaned dataframe with outliers removed.
    '''
    df_out = df.copy()
    for col in columns:
        Q1 = df_out[col].quantile(0.25) # First quartile
        Q3 = df_out[col].quantile(0.75) # Third quartile
        IQR = Q3 - Q1 # Interquartile range
        lower_bound = Q1 - 1.5 * IQR # Lower bound for outliers
        upper_bound = Q3 + 1.5 * IQR # Upper bound for outliers

        df_out = df_out[(df_out[col] >= lower_bound) & (df_out[col] <= upper_bound)] # Remove outliers

    return df_out

# List of columns to remove outliers from
columns_to_remove_outliers = ['Duration', 'Amount', 'Age']

# Remove outliers from subset1_cleaned and subset2_cleaned
subset1_cleaned = remove_outliers(subset1_cleaned, columns_to_remove_outliers)
subset2_cleaned = remove_outliers(subset2_cleaned, columns_to_remove_outliers)
```

```
# Replace subset1 and subset2 with your actual DataFrame names
# Replace 'continuous_var1', 'continuous_var2', etc. with the column names of your continuous variables
continuous_variables =  ['Duration', 'Amount', 'Installp', 'Resident', 'Age', 'Existcr', 'Depends']

# Function to create subplots for a given set of continuous variables in a given subset
def plot_subplots(subset, continuous_variables, subset_name):
    num_vars = len(continuous_variables)
    fig = make_subplots(cols=num_vars, subplot_titles=continuous_variables)

    for i, var in enumerate(continuous_variables, start=1):
        fig.add_trace(go.Box(y=subset[var], name=var, showlegend=False), col=i, row=1)

    fig.update_layout(title=f'Box Plots for Continuous Variables in {subset_name}')
    fig.show()

# Create subplots for each continuous variable in both subsets
plot_subplots(subset1_cleaned, continuous_variables, "Subset 1")
plot_subplots(subset2_cleaned, continuous_variables, "Subset 2")
```

Q2) Establishing Training set and Validation set

Train and test split

```
import pandas as pd
from sklearn.model_selection import train_test_split


# Split subset1 using stratified sampling
```

```
X1 = subset1_cleaned.drop(columns=['Bad', 'Good'])
y1 = subset1_cleaned['Bad']  # 'Bad' is the target variable
X1_train, X1_val, y1_train, y1_val = train_test_split(X1, y1, test_size=0.3, stratify=y1, random_state=

# Split subset2 using stratified sampling
X2 = subset2_cleaned.drop(columns=['Bad', 'Good'])
y2 = subset2_cleaned['Bad']  # 'Bad' is the target variable
X2_train, X2_val, y2_train, y2_val = train_test_split(X2, y2, test_size=0.3, stratify=y2, random_state=
```

```
y1_train.value_counts()
```

```
0    188
1    141
Name: Bad, dtype: int64
```

```
y2_train.value_counts()
```

```
0    248
1     32
Name: Bad, dtype: int64
```

```
y1_val.value_counts()
```

```
0    81
1    60
Name: Bad, dtype: int64
```

```
y2_val.value_counts()
```

```
0    106
1     14
Name: Bad, dtype: int64
```

```
subset1['Good'].value_counts()
subset1['Bad'].value_counts()
```

```
0    303
1    240
Name: Bad, dtype: int64
```

```
subset1_cleaned['Good'].value_counts()
subset1_cleaned['Bad'].value_counts()
```

```
0    269
1    201
Name: Bad, dtype: int64
```

```
subset2['Good'].value_counts()
subset2['Bad'].value_counts()
```

```
0    397
1     60
Name: Bad, dtype: int64
```

```
subset2_cleaned['Good'].value_counts()
subset2_cleaned['Bad'].value_counts()
```

```
0    354
1     46
Name: Bad, dtype: int64
```

```
import plotly.graph_objects as go
```

```
# Count the number of Good and Bad applicants in each subset
subset1_train_counts = y1_train.value_counts()
subset2_train_counts = y2_train.value_counts()


# Create the bar chart for Subset 1
trace1 = go.Bar(
    x=['Good', 'Bad'],
    y=subset1_train_counts,
    name='Subset 1 Training (Checking = 1 or 2)',
    marker_color='rgb(55, 83, 109)'
)

# Create the bar chart for Subset 2
trace2 = go.Bar(
    x=['Good', 'Bad'],
    y=subset2_train_counts,
    name='Subset 2 Training (Checking = 3 or 4)',
    marker_color='rgb(26, 118, 255)'
)

# Combine the bar charts and set the layout options
data = [trace1, trace2]
layout = go.Layout(
    title='Distribution of Good and Bad Applicants in Training Sets',
    xaxis=dict(title='Applicant Type'),
    yaxis=dict(title='Count'),
    barmode='group'
)

# Create the final figure and display it
fig = go.Figure(data=data, layout=layout)
fig.show()
```

```
# Count the number of Good and Bad applicants in the training and validation sets for both subsets
subset1_val_counts = y1_val.value_counts()
subset2_val_counts = y2_val.value_counts()

# Create the bar charts for the training and validation sets
trace1_train = go.Bar(
    x=['Good', 'Bad'],
    y=subset1_train_counts,
    name='Subset 1 Train (Checking = 1 or 2)',
    marker_color='rgb(55, 83, 109)',
    opacity=0.6
)

trace1_val = go.Bar(
    x=['Good', 'Bad'],
    y=subset1_val_counts,
    name='Subset 1 Validation (Checking = 1 or 2)',
    marker_color='rgb(55, 83, 109)',
    opacity=1
)

trace2_train = go.Bar(
    x=['Good', 'Bad'],
    y=subset2_train_counts,
    name='Subset 2 Train (Checking = 3 or 4)',
    marker_color='rgb(26, 118, 255)',
    opacity=0.6
)

trace2_val = go.Bar(
    x=['Good', 'Bad'],
    y=subset2_val_counts,
    name='Subset 2 Validation (Checking = 3 or 4)',
    marker_color='rgb(26, 118, 255)',
    opacity=1
```

```
)

# Combine the bar charts and set the layout options
data = [trace1_val, trace2_val]
layout = go.Layout(
    title='Distribution of Good and Bad Applicants in Validation Sets',
    xaxis=dict(title='Applicant Type'),
    yaxis=dict(title='Count'),
    barmode='group'
)

# Create the final figure and display it
fig = go.Figure(data=data, layout=layout)
fig.show()
```

Correlation Matrix

```
# Select the numerical variables in subset1
numerical_columns = ['Duration', 'Amount', 'Installp', 'Resident', 'Age', 'Existcr', 'Depends']
numerical_subset1 = X1_train[numerical_columns]

# Calculate the correlation matrix for subset1
correlation_matrix_subset1 = numerical_subset1.corr()
print("Correlation matrix for Subset1:\n", correlation_matrix_subset1)
```

```
Correlation matrix for Subset1:
           Duration    Amount  Installp  Resident       Age   Existcr   Depends
Duration  1.000000  0.560094  0.070830  0.065675  0.060329 -0.027796  0.068620
Amount    0.560094  1.000000 -0.275284  0.056177  0.149398  0.097159  0.130448
Installp  0.070830 -0.275284  1.000000  0.025610  0.055960 -0.089120 -0.143198
Resident  0.065675  0.056177  0.025610  1.000000  0.149161  0.111709  0.066619
Age       0.060329  0.149398  0.055960  0.149161  1.000000  0.154306  0.170292
Existcr  -0.027796  0.097159 -0.089120  0.111709  0.154306  1.000000  0.087835
Depends   0.068620  0.130448 -0.143198  0.066619  0.170292  0.087835  1.000000
```

```
# Select the numerical variables in subset2
numerical_subset2 = X2_train[numerical_columns]

# Calculate the correlation matrix for subset2
correlation_matrix_subset2 = numerical_subset2.corr()
print("\nCorrelation matrix for Subset2:\n", correlation_matrix_subset2)
```

```
Correlation matrix for Subset2:
           Duration    Amount  Installp  Resident       Age   Existcr   Depends
Duration  1.000000  0.494617  0.320752  0.102949 -0.016334  0.057801 -0.138760
Amount    0.494617  1.000000 -0.247300  0.089167 -0.030396  0.044734 -0.008450
Installp  0.320752 -0.247300  1.000000  0.095760  0.136242  0.109012 -0.060813
Resident  0.102949  0.089167  0.095760  1.000000  0.335953  0.008813 -0.013590
Age      -0.016334 -0.030396  0.136242  0.335953  1.000000  0.162378  0.108470
Existcr   0.057801  0.044734  0.109012  0.008813  0.162378  1.000000  0.072210
Depends  -0.138760 -0.008450 -0.060813 -0.013590  0.108470  0.072210  1.000000
```

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

# Create the subplots with 1 row and 2 columns
fig, axes = plt.subplots(1, 2, figsize=(12, 6))

# Create the first heatmap for Subset1
sns.heatmap(correlation_matrix_subset1, annot=False, square=True, ax=axes[0])
axes[0].set_title('Correlation Matrix for Subset1')

# Create the second heatmap for Subset2
```

```
sns.heatmap(correlation_matrix_subset2, annot=False, square=True, ax=axes[1])
axes[1].set_title('Correlation Matrix for Subset2')

# Adjust the spacing between the subplots
fig.subplots_adjust(wspace=0.4)

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

Q3) Variable selection and Binning

Cleaning data

```
import pandas as pd
from sklearn.preprocessing import LabelEncoder
from sklearn.preprocessing import OrdinalEncoder
from sklearn.preprocessing import OneHotEncoder
from optbinning import OptimalBinning
```

```
# convert 'Purpose' column to int64 data type in both subsets
X1_train['Purpose'] = X1_train['Purpose'].astype('int64')
X2_train['Purpose'] = X2_train['Purpose'].astype('int64')
```

OptimalBinning function

```
import pandas as pd
from optbinning import OptimalBinning
from sklearn.preprocessing import OrdinalEncoder

def perform_optimal_binning(X, y, continuous_vars, categorical_vars):
    X_binned = X.copy()
    iv_values = {}

    for variable in continuous_vars:
        optb = OptimalBinning(name=variable, dtype="numerical", prebinning_method="cart", solver="cp")
        optb.fit(X[variable].values, y.values)
        X_binned[variable] = optb.transform(X[variable].values, metric="indices")
        binning_table = optb.binning_table.build()
        iv_values[variable] = binning_table.loc["Totals", "IV"]

        print("NUMERICAL BINNING: ")
        print(f"Binning table for {variable}:")
        print(binning_table)
        print("\n")

    for variable in categorical_vars:
        optb = OptimalBinning(name=variable, dtype="categorical", solver="mip", cat_cutoff=None)
        optb.fit(X[variable].values, y.values)
        X_binned[variable] = optb.transform(X[variable].values, metric="indices")
        binning_table = optb.binning_table.build()
        iv_values[variable] = binning_table.loc["Totals", "IV"]

        print("CATEGORICAL BINNING: ")
        print(f"Binning table for {variable}:")
        print(binning_table)
        print("\n")

    return X_binned, iv_values
```

```
continuous_vars = ['Duration', 'Amount', 'Installp', 'Resident', 'Age', 'Existcr', 'Depends']
categorical_vars = [col for col in X1_train.columns if col not in continuous_vars]
```

Optimal Binning of X1_train

```
X1_train_binned, iv_values_X1 = perform_optimal_binning(X1_train, y1_train, continuous_vars, categorica
```

Optimal Binning of X2_train

```
X2_train_binned, iv_values_X2 = perform_optimal_binning(X2_train, y2_train, continuous_vars, categorica
```

Sorted IV's without Custom Binning

```
print("Information Value for Subset1:")
sorted_iv_values_X1 = sorted(iv_values_X1.items(), key=lambda x: x[1], reverse=True)
for variable, iv in sorted_iv_values_X1:
    print(f"{variable}: {iv}")

print("\nInformation Value for Subset2:")
sorted_iv_values_X2 = sorted(iv_values_X2.items(), key=lambda x: x[1], reverse=True)
for variable, iv in sorted_iv_values_X2:
    print(f"{variable}: {iv}")
```

```
Information Value for Subset1:
Duration: 0.4178489133645281
Age: 0.3274238864183101
History: 0.3197574550699738
Property: 0.1959232008502502
Amount: 0.16805023062228486
housing: 0.1493033198743211
Purpose: 0.12927103896433442
Savings: 0.12061556913976491
marital: 0.09745041681263165
Emploed: 0.08206587445302474
Installp: 0.05924117611193566
Coapp: 0.05596615242565124
Checking: 0.05044658012181499
Job: 0.04844828406310453
Existcr: 0.035820184424735585
Other: 0.033378873482336674
Resident: 0.02011687610608317
Depends: 0.014404628197682982
Telephone: 0.0020486642436627935
Foreign: 0.0

Information Value for Subset2:
Duration: 0.4828683748484717
Amount: 0.45914162410514436
History: 0.4248002895744063
Other: 0.32791173795221573
Purpose: 0.2368400811339722
Depends: 0.2129056651108976
Telephone: 0.15592834715762222
Age: 0.13426287511844748
Emploed: 0.12576038278569646
Job: 0.12348305128489563
Savings: 0.11180765359191974
```

```
    Installp: 0.07763495298793169
    Coapp: 0.06885914571488531
    Checking: 0.06308854436647741
    Property: 0.04339234818747909
    marital: 0.03811710898853561
    Resident: 0.01976005251637155
    Existcr: 0.014169270895199312
    housing: 0.009027561094449752
    Foreign: 0.0
```

```
import pandas as pd
from optbinning import OptimalBinning
from sklearn.preprocessing import OrdinalEncoder

def perform_optimal_binning2(X, y, continuous_vars, categorical_vars):
    X_binned = X.copy()
    iv_values = {}

    for variable in continuous_vars:
        if variable == "Duration":
            user_splits = [7.5, 11.5]
        elif variable == "Installp":
            user_splits = [1.5]
        elif variable == "Age":
            user_splits = [24.5, 25.5]
        else:
            user_splits = None

        optb = OptimalBinning(name=variable, dtype="numerical", prebinning_method="cart", solver="cp",
        optb.fit(X[variable].values, y.values)
        X_binned[variable] = optb.transform(X[variable].values, metric="indices")
        binning_table = optb.binning_table.build()
        iv_values[variable] = binning_table.loc["Totals", "IV"]

        print("NUMERICAL BINNING: ")
        print(f"Binning table for {variable}:")
        print(binning_table)
        print("\n")

    for variable in categorical_vars:
        optb = OptimalBinning(name=variable, dtype="categorical", solver="mip", cat_cutoff=None)
        optb.fit(X[variable].values, y.values)
        X_binned[variable] = optb.transform(X[variable].values, metric="indices")
        binning_table = optb.binning_table.build()
        iv_values[variable] = binning_table.loc["Totals", "IV"]

        print("CATEGORICAL BINNING: ")
        print(f"Binning table for {variable}:")
        print(binning_table)
        print("\n")

    return X_binned, iv_values

continuous_vars = ['Duration', 'Amount', 'Installp', 'Resident', 'Age', 'Existcr', 'Depends']
categorical_vars = [col for col in X1_train.columns if col not in continuous_vars]

# Optimal Binning of X1_train
X1_train_binned2, iv2_values_X1 = perform_optimal_binning2(X1_train, y1_train, continuous_vars, categor
```

Investigating high IV values

```
import pandas as pd
import plotly.graph_objects as go
from plotly.subplots import make_subplots

# Combine the training variables and target variable into a single DataFrame
train_df = pd.concat([X2_train, y2_train], axis=1)
```

```
# Create a subplot with 1 row and 3 columns
fig = make_subplots(rows=1, cols=3, subplot_titles=("Duration", "Foreign", "History"), specs=[[{'type'

# Calculate counts for each category and the target variable 'Bad'
purpose_counts = train_df.groupby(['Duration', 'Bad']).size().reset_index(name='Count')
foreign_counts = train_df.groupby(['Foreign', 'Bad']).size().reset_index(name='Count')
history_counts = train_df.groupby(['History', 'Bad']).size().reset_index(name='Count')

# Add bar charts to the subplot for Good credit risk (Bad == 0)
fig.add_trace(go.Bar(x=purpose_counts[purpose_counts['Bad'] == 0]['Duration'], y=purpose_counts[purpose
fig.add_trace(go.Bar(x=foreign_counts[foreign_counts['Bad'] == 0]['Foreign'], y=foreign_counts[foreign_
fig.add_trace(go.Bar(x=history_counts[history_counts['Bad'] == 0]['History'], y=history_counts[history_

# Add bar charts to the subplot for Bad credit risk (Bad == 1)
fig.add_trace(go.Bar(x=purpose_counts[purpose_counts['Bad'] == 1]['Duration'], y=purpose_counts[purpose
fig.add_trace(go.Bar(x=foreign_counts[foreign_counts['Bad'] == 1]['Foreign'], y=foreign_counts[foreign_
fig.add_trace(go.Bar(x=history_counts[history_counts['Bad'] == 1]['History'], y=history_counts[history_

# Update the layout
fig.update_layout(barmode='group', title_text="Count Plots of Duration, Foreign, and History with the T

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

Subset 1 Binning (Top 4 variables)

Binning: History

```
# Optimal binning for History
optb_history = OptimalBinning(name='History', dtype='categorical', solver='cp')
optb_history.fit(X1_train['History'], y1_train)
binning_table_history = optb_history.binning_table.build()
print("Optimal binning for History:")
print(binning_table_history)
```

```
Optimal binning for History:
             Bin  Count  Count (%)  Non-event  Event  Event rate       WoE  \
0            [4]     75   0.227964         58     17    0.226667  0.939548
1            [3]     33   0.100304         19     14    0.424242    0.0177
2            [2]    180   0.547112         98     82    0.455556 -0.109434
3            [1]     22   0.066869          7     15    0.681818 -1.049822
4            [0]     19   0.057751          6     13    0.684211 -1.060872
5        Special      0   0.000000          0      0    0.000000       0.0
6        Missing      0   0.000000          0      0    0.000000       0.0
Totals              329   1.000000        188    141    0.428571

              IV        JS
0       0.176582  0.021295
1       0.000031  0.000004
2       0.006597  0.000824
3       0.072594  0.008679
4       0.063953  0.007639
5       0.000000  0.000000
6       0.000000  0.000000
Totals  0.319757  0.038442
```

Customized Binning

```
from optbinning import OptimalBinning

# Adjusting parameters for OptimalBinning
optb_history_adjusted = OptimalBinning(name='History', dtype='categorical', solver='cp', max_n_bins=3)
```

```
# Fitting the adjusted OptimalBinning object
optb_history_adjusted.fit(X1_train['History'], y1_train)

# Building and displaying the adjusted binning table
binning_table_history_adjusted = optb_history_adjusted.binning_table.build()
print("Adjusted optimal binning for History:")
print(binning_table_history_adjusted)
```

```
Adjusted optimal binning for History:
            Bin  Count  Count (%)  Non-event  Event  Event rate       WoE  \
0           [4]     75   0.227964         58     17    0.226667  0.939548
1        [3, 2]    213   0.647416        117     96    0.450704 -0.089856
2        [1, 0]     41   0.124620         13     28    0.682927 -1.054937
3       Special      0   0.000000          0      0    0.000000       0.0
4       Missing      0   0.000000          0      0    0.000000       0.0
Totals             329   1.000000        188    141    0.428571

              IV        JS
0       0.176582  0.021295
1       0.005258  0.000657
2       0.136543  0.016318
3       0.000000  0.000000
4       0.000000  0.000000
Totals  0.318382  0.038270
```

Binning: Age

```
# Optimal binning for Age
optb_age = OptimalBinning(name='Age', dtype='numerical', solver='cp')
optb_age.fit(X1_train['Age'], y1_train)
binning_table_age = optb_age.binning_table.build()
print("Optimal binning for Age:")
print(binning_table_age)
```

```
Optimal binning for Age:
                    Bin  Count  Count (%)  Non-event  Event  Event rate  \
0         (-inf, 24.50)     63   0.191489         26     37    0.587302
1        [24.50, 25.50)     19   0.057751         10      9    0.473684
2        [25.50, 34.50)    113   0.343465         66     47    0.415929
3        [34.50, 36.50)     24   0.072948         20      4    0.166667
4        [36.50, 39.50)     20   0.060790         17      3    0.150000
5        [39.50, 49.50)     55   0.167173         34     21    0.381818
6          [49.50, inf)     35   0.106383         15     20    0.571429
7               Special      0   0.000000          0      0    0.000000
8               Missing      0   0.000000          0      0    0.000000
Totals                     329   1.000000        188    141    0.428571

             WoE        IV        JS
0      -0.640503  0.079495  0.009770
1      -0.182322  0.001940  0.000242
2       0.051825  0.000919  0.000115
3       1.321756  0.103116  0.012026
4       1.446919  0.100053  0.011518
5       0.194156  0.006196  0.000773
6      -0.575364  0.035705  0.004403
7            0.0  0.000000  0.000000
8            0.0  0.000000  0.000000
Totals            0.327424  0.038848
```

Customized Binning

```
# Optimal binning for Age
optb_age_adjusted = OptimalBinning(name='Age', dtype='numerical', solver='cp', max_n_bins=5)
optb_age_adjusted.fit(X1_train['Age'], y1_train)
binning_table_age_adjusted = optb_age_adjusted.binning_table.build()
print("Optimal binning for Age:")
print(binning_table_age_adjusted)
```

```
Optimal binning for Age:
                  Bin  Count  Count (%)  Non-event  Event  Event rate  \
0       (-inf, 24.50)     63   0.191489         26     37    0.587302
1      [24.50, 34.50)    132   0.401216         76     56    0.424242
2      [34.50, 39.50)     44   0.133739         37      7    0.159091
3      [39.50, 49.50)     55   0.167173         34     21    0.381818
4        [49.50, inf)     35   0.106383         15     20    0.571429
5             Special      0   0.000000          0      0    0.000000
6             Missing      0   0.000000          0      0    0.000000
Totals                   329   1.000000        188    141    0.428571

             WoE        IV        JS
0      -0.640503  0.079495  0.009770
1         0.0177  0.000126  0.000016
2       1.377326  0.202692  0.023506
3       0.194156  0.006196  0.000773
4      -0.575364  0.035705  0.004403
5            0.0  0.000000  0.000000
6            0.0  0.000000  0.000000
Totals            0.324214  0.038468
```

Binning: Duration

```
# Optimal binning for Duration
optb_duration = OptimalBinning(name='Duration', dtype='numerical', solver='cp')
optb_duration.fit(X1_train['Duration'], y1_train)
binning_table_duration = optb_duration.binning_table.build()
print("Optimal binning for Duration:")
print(binning_table_duration)
```

```
Optimal binning for Duration:
                  Bin  Count  Count (%)  Non-event  Event  Event rate  \
0        (-inf, 7.50)     24   0.072948         21      3    0.125000
1       [7.50, 11.50)     31   0.094225         22      9    0.290323
2      [11.50, 22.50)    145   0.440729         91     54    0.372414
3      [22.50, 31.50)     79   0.240122         36     43    0.544304
4      [31.50, 43.50)     31   0.094225         14     17    0.548387
5        [43.50, inf)     19   0.057751          4     15    0.789474
6             Special      0   0.000000          0      0    0.000000
7             Missing      0   0.000000          0      0    0.000000
Totals                   329   1.000000        188    141    0.428571

             WoE        IV        JS
0       1.658228  0.149946  0.016854
1       0.606136  0.032241  0.003970
2       0.234193  0.023668  0.002952
3      -0.465363  0.052807  0.006542
4      -0.481838  0.022212  0.002750
5      -1.609438  0.136973  0.015484
6            0.0  0.000000  0.000000
7            0.0  0.000000  0.000000
Totals            0.417849  0.048551
```

Customized Binning

```
# Optimal binning for Duration
optb_duration_adjusted = OptimalBinning(name='Duration', dtype='numerical', solver='cp', max_n_bins=4)
optb_duration_adjusted.fit(X1_train['Duration'], y1_train)
binning_table_duration_adjusted = optb_duration_adjusted.binning_table.build()
print("Optimal binning for Duration:")
print(binning_table_duration_adjusted)
```

```
Optimal binning for Duration:
                 Bin  Count  Count (%)  Non-event  Event  Event rate  \
0        (-inf, 7.50)     24   0.072948         21      3    0.125000
1       [7.50, 22.50)    176   0.534954        113     63    0.357955
2      [22.50, 43.50)    110   0.334347         50     60    0.545455
3        [43.50, inf)     19   0.057751          4     15    0.789474
4             Special      0   0.000000          0      0    0.000000
5             Missing      0   0.000000          0      0    0.000000
Totals                   329   1.000000        188    141    0.428571

             WoE        IV        JS
0       1.658228  0.149946  0.016854
1       0.296571  0.045748  0.005698
2      -0.470004  0.075001  0.009290
3      -1.609438  0.136973  0.015484
4            0.0  0.000000  0.000000
5            0.0  0.000000  0.000000
Totals            0.407668  0.047325
```

Binning: Property

```
# Optimal binning for Property
optb_property = OptimalBinning(name='Property', dtype='categorical', solver='cp')
optb_property.fit(X1_train['Property'], y1_train)
binning_table_property = optb_property.binning_table.build()
print("Optimal binning for Property:")
print(binning_table_property)
```

```
Optimal binning for Property:
            Bin  Count  Count (%)  Non-event  Event  Event rate       WoE  \
0           [1]     90   0.273556         65     25    0.277778  0.667829
1        [3, 2]    195   0.592705        106     89    0.456410 -0.112879
2           [4]     44   0.133739         17     27    0.613636 -0.750306
3       Special      0   0.000000          0      0    0.000000       0.0
4       Missing      0   0.000000          0      0    0.000000       0.0
Totals             329   1.000000        188    141    0.428571

              IV        JS
0       0.112489  0.013806
1       0.007605  0.000950
2       0.075829  0.009262
3       0.000000  0.000000
4       0.000000  0.000000
Totals  0.195923  0.024018
```

Customized Binning

```
# Optimal binning for Property
optb_property_adjusted = OptimalBinning(name='Property', dtype='categorical', solver='cp', max_n_bins=
optb_property_adjusted.fit(X1_train['Property'], y1_train)
binning_table_property_adjusted = optb_property_adjusted.binning_table.build()
print("Optimal binning for Property:")
print(binning_table_property_adjusted)
```

```
Optimal binning for Property:
              Bin  Count  Count (%)  Non-event  Event  Event rate       WoE  \
0             [1]     90   0.273556         65     25    0.277778  0.667829
1       [3, 2, 4]    239   0.726444        123    116    0.485356 -0.229088
2         Special      0   0.000000          0      0    0.000000       0.0
3         Missing      0   0.000000          0      0    0.000000       0.0
Totals               329   1.000000        188    141    0.428571

              IV        JS
0       0.112489  0.013806
1       0.038588  0.004813
2       0.000000  0.000000
3       0.000000  0.000000
Totals  0.151076  0.018618
```

Transform and Encode Training Set

```
# Transform the original variables into their binned categories
X1_train_transformed = X1_train.copy()
X1_train_transformed['History'] = optb_history.transform(X1_train_transformed['History'], metric='bins
X1_train_transformed['Duration'] = optb_duration.transform(X1_train_transformed['Duration'], metric='bi
X1_train_transformed['Age'] = optb_age.transform(X1_train_transformed['Age'], metric='bins')
X1_train_transformed['Property'] = optb_property.transform(X1_train_transformed['Property'], metric='bi



## Ordinal Encoding

# Obtain the bin labels for the binned Age variable from the binning table
bin_labels = binning_table_age['Bin'].tolist()
# Create an instance of the OrdinalEncoder with categories set to the sorted bin labels
encoder = OrdinalEncoder(categories=[bin_labels], dtype=int)
# Encode the Age variable in the X1_train_transformed dataframe using the OrdinalEncoder
X1_train_transformed['Age'] = encoder.fit_transform(X1_train_transformed[['Age']])


# Obtain the bin labels for the binned Duration variable from the binning table
bin_labels = binning_table_duration['Bin'].tolist()
# Create an instance of the OrdinalEncoder with categories set to the sorted bin labels
encoder = OrdinalEncoder(categories=[bin_labels], dtype=int)
# Encode the Duration variable in the X1_train_transformed dataframe using the OrdinalEncoder
X1_train_transformed['Duration'] = encoder.fit_transform(X1_train_transformed[['Duration']])


# One-hot Encoding
from sklearn.preprocessing import OneHotEncoder

# Instantiate the OneHotEncoder, specify the columns to encode, and set the drop parameter to 'first'
encoder = OneHotEncoder(sparse=False, drop='first')
X1_train_encoded = encoder.fit_transform(X1_train_transformed[['History', 'Property']])

# Create a DataFrame with the encoded columns and their respective names
encoded_columns = encoder.get_feature_names_out(['History', 'Property'])
X1_train_encoded_df = pd.DataFrame(X1_train_encoded, columns=encoded_columns, index=X1_train_transforme

# Drop the original 'History' and 'Property' columns and concatenate the encoded DataFrame
X1_train_transformed.drop(['History', 'Property'], axis=1, inplace=True)
X1_train_transformed = pd.concat([X1_train_transformed, X1_train_encoded_df], axis=1)


X1_train_transformed = X1_train_transformed[['History_[1]', 'History_[2]', 'History_[3]', 'History_[4]
```

Transform and Encode Validation Set

```
# Transform the original variables into their binned categories
X1_val_transformed = X1_val.copy()
X1_val_transformed['History'] = optb_history.transform(X1_val_transformed['History'], metric='bins')
X1_val_transformed['Duration'] = optb_duration.transform(X1_val_transformed['Duration'], metric='bins'
X1_val_transformed['Age'] = optb_age.transform(X1_val_transformed['Age'], metric='bins')
X1_val_transformed['Property'] = optb_property.transform(X1_val_transformed['Property'], metric='bins'


## Ordinal Encoding
from sklearn.preprocessing import OrdinalEncoder
# Obtain the bin labels for the binned Age variable from the binning table
```

```
bin_labels = binning_table_age['Bin'].tolist()
# Create an instance of the OrdinalEncoder with categories set to the sorted bin labels
encoder = OrdinalEncoder(categories=[bin_labels], dtype=int)
# Encode the Age variable in the X1_train_transformed dataframe using the OrdinalEncoder
X1_val_transformed['Age'] = encoder.fit_transform(X1_val_transformed[['Age']])


# Obtain the bin labels for the binned Duration variable from the binning table
bin_labels = binning_table_duration['Bin'].tolist()
# Create an instance of the OrdinalEncoder with categories set to the sorted bin labels
encoder = OrdinalEncoder(categories=[bin_labels], dtype=int)
# Encode the Duration variable in the X1_train_transformed dataframe using the OrdinalEncoder
X1_val_transformed['Duration'] = encoder.fit_transform(X1_val_transformed[['Duration']])



# One-hot Encoding
from sklearn.preprocessing import OneHotEncoder

# Instantiate the OneHotEncoder, specify the columns to encode, and set the drop parameter to 'first'
encoder = OneHotEncoder(sparse=False, drop='first')
X1_val_encoded = encoder.fit_transform(X1_val_transformed[['History', 'Property']])

# Create a DataFrame with the encoded columns and their respective names
encoded_columns = encoder.get_feature_names_out(['History', 'Property'])
X1_val_encoded_df = pd.DataFrame(X1_val_encoded, columns=encoded_columns, index=X1_val_transformed.inde

# Drop the original 'History' and 'Property' columns and concatenate the encoded DataFrame
X1_val_transformed.drop(['History', 'Property'], axis=1, inplace=True)
X1_val_transformed = pd.concat([X1_val_transformed, X1_val_encoded_df], axis=1)


X1_val_transformed = X1_val_transformed[['History_[1]', 'History_[2]', 'History_[3]', 'History_[4]', 'P

X1_val_transformed
```

Binary Variables Subset 1

```
X1_train_transformed
```

	History_[1]	History_[2]	History_[3]	History_[4]	Property_[32]	Property_[4]	Duration	Age
227	0.0	1.0	0.0	0.0	0.0	1.0	2	6
351	0.0	1.0	0.0	0.0	1.0	0.0	1	2
892	0.0	0.0	0.0	1.0	1.0	0.0	2	4
103	0.0	0.0	0.0	1.0	1.0	0.0	1	3
207	0.0	0.0	0.0	1.0	1.0	0.0	2	2
...	...	...	...	...	...	...	...	...
765	0.0	1.0	0.0	0.0	0.0	0.0	2	5
631	1.0	0.0	0.0	0.0	1.0	0.0	2	5
525	0.0	1.0	0.0	0.0	1.0	0.0	3	2
507	1.0	0.0	0.0	0.0	1.0	0.0	2	2
766	0.0	1.0	0.0	0.0	1.0	0.0	3	2

Subset 2 Binning (Top 4 variables)

Binning: Duration

```
# Optimal binning for Duration
optb_duration2 = OptimalBinning(name='Duration', dtype='numerical', solver='cp')
optb_duration2.fit(X2_train['Duration'], y2_train)
binning_table_duration2 = optb_duration2.binning_table.build()
print("Optimal binning for Duration:")
print(binning_table_duration2)
```

```
Optimal binning for Duration:
                    Bin  Count  Count (%)  Non-event  Event  Event rate  \
0         (-inf, 8.00)     34   0.121429         33      1    0.029412   
1        [8.00, 16.50)    117   0.417857        105     12    0.102564   
2       [16.50, 21.50)     44   0.157143         32     12    0.272727   
3       [21.50, 25.50)     62   0.221429         56      6    0.096774   
4         [25.50, inf)     23   0.082143         22      1    0.043478   
5              Special      0   0.000000          0      0    0.000000   
6              Missing      0   0.000000          0      0    0.000000   
Totals                    280   1.000000        248     32    0.114286   

             WoE        IV        JS  
0       1.448815  0.147510  0.016978  
1       0.121361  0.005872  0.000734  
2      -1.066864  0.262414  0.031330  
3       0.185899  0.007121  0.000889  
4        1.04335  0.059951  0.007171  
5            0.0  0.000000  0.000000  
6            0.0  0.000000  0.000000  
Totals            0.482868  0.057102  
```

Customized Binning

```
# Optimal binning for Duration
optb_duration2_adjusted = OptimalBinning(name='Duration', dtype='numerical', solver='cp', max_n_bins=4
optb_duration2_adjusted.fit(X2_train['Duration'], y2_train)
binning_table_duration2_adjusted = optb_duration2_adjusted.binning_table.build()
print("Optimal binning for Duration:")
print(binning_table_duration2_adjusted)
```

```
Optimal binning for Duration:
                    Bin  Count  Count (%)  Non-event  Event  Event rate  \
0         (-inf, 8.00)     34   0.121429         33      1    0.029412   
1        [8.00, 16.50)    117   0.417857        105     12    0.102564   
2       [16.50, 21.50)     44   0.157143         32     12    0.272727   
3         [21.50, inf)     85   0.303571         78      7    0.082353   
4              Special      0   0.000000          0      0    0.000000   
5              Missing      0   0.000000          0      0    0.000000   
Totals                    280   1.000000        248     32    0.114286   

             WoE        IV        JS  
0       1.448815  0.147510  0.016978  
1       0.121361  0.005872  0.000734  
2      -1.066864  0.262414  0.031330  
3       0.363106  0.034773  0.004323  
4            0.0  0.000000  0.000000  
5            0.0  0.000000  0.000000  
Totals            0.450570  0.053365  
```

Binning: History

```
# Optimal binning for History
optb_history2 = OptimalBinning(name='History', dtype='categorical', solver='cp')
optb_history2.fit(X2_train['History'], y2_train)
binning_table_history2 = optb_history2.binning_table.build()
print("Optimal binning for History (Subset 2):")
print(binning_table_history2)
```

```
Optimal binning for History (Subset 2):
            Bin  Count  Count (%)  Non-event  Event  Event rate       WoE  \
0           [4]     95   0.339286         90      5    0.052632  0.842679
1        [2, 1]    156   0.557143        138     18    0.115385 -0.010811
2        [3, 0]     29   0.103571         20      9    0.310345 -1.249185
3       Special      0   0.000000          0      0    0.000000       0.0
4       Missing      0   0.000000          0      0    0.000000       0.0
Totals             280   1.000000        248     32    0.114286

              IV        JS
0       0.174142  0.021146
1       0.000065  0.000008
2       0.250593  0.029434
3       0.000000  0.000000
4       0.000000  0.000000
Totals  0.424800  0.050588
```

Customized Binning

```
# Optimal binning for History
optb_history2_adjusted = OptimalBinning(name='History', dtype='categorical', solver='cp', max_n_bins=2
optb_history2_adjusted.fit(X2_train['History'], y2_train)
binning_table_history2_adjusted = optb_history2_adjusted.binning_table.build()
print("Optimal binning for History (Subset 2):")
print(binning_table_history2_adjusted)
```

```
Optimal binning for History (Subset 2):
              Bin  Count  Count (%)  Non-event  Event  Event rate       WoE  \
0       [4, 2, 1]    251   0.896429        228     23    0.091633  0.246159
1          [3, 0]     29   0.103571         20      9    0.310345 -1.249185
2         Special      0   0.000000          0      0    0.000000       0.0
3         Missing      0   0.000000          0      0    0.000000       0.0
Totals               280   1.000000        248     32    0.114286

              IV        JS
0       0.049381  0.006157
1       0.250593  0.029434
2       0.000000  0.000000
3       0.000000  0.000000
Totals  0.299973  0.035591
```

Binning: Amount

```
# Optimal binning for Amount
optb_amount = OptimalBinning(name='Amount', dtype='numerical', solver='cp')
optb_amount.fit(X2_train['Amount'], y2_train)
binning_table_amount = optb_amount.binning_table.build()
print("Optimal binning for Amount (Subset 2):")
print(binning_table_amount)
```

```
Optimal binning for Amount (Subset 2):
                         Bin  Count  Count (%)  Non-event  Event  Event rate  \
0           (-inf, 953.50)     33   0.117857         28      5    0.151515
1        [953.50, 1561.50)     72   0.257143         63      9    0.125000
2       [1561.50, 2803.00)     89   0.317857         79     10    0.112360
3       [2803.00, 4152.00)     54   0.192857         53      1    0.018519
4           [4152.00, inf)     32   0.114286         25      7    0.218750
5                  Special      0   0.000000          0      0    0.000000
6                  Missing      0   0.000000          0      0    0.000000
Totals                        280   1.000000        248     32    0.114286

             WoE        IV        JS
0      -0.324926  0.014085  0.001753
1      -0.101783  0.002770  0.000346
2        0.01917  0.000116  0.000014
3       1.922599  0.350797  0.038140
4      -0.774727  0.091374  0.011144
5            0.0  0.000000  0.000000
```

```
6           0.0  0.000000  0.000000
Totals           0.459142  0.051398
```

Customized Binning

```
# Optimal binning for Amount
optb_amount_adjusted = OptimalBinning(name='Amount', dtype='numerical', solver='cp', max_n_bins=3)
optb_amount_adjusted.fit(X2_train['Amount'], y2_train)
binning_table_amount_adjusted = optb_amount_adjusted.binning_table.build()
print("Optimal binning for Amount (Subset 2):")
print(binning_table_amount_adjusted)
```

```
Optimal binning for Amount (Subset 2):
                     Bin  Count  Count (%)  Non-event  Event  Event rate  \
0         (-inf, 2803.00)    194   0.692857        170     24    0.123711
1      [2803.00, 4152.00)     54   0.192857         53      1    0.018519
2          [4152.00, inf)     32   0.114286         25      7    0.218750
3                 Special      0   0.000000          0      0    0.000000
4                 Missing      0   0.000000          0      0    0.000000
Totals                       280   1.000000        248     32    0.114286

             WoE        IV        JS
0      -0.089948  0.005803  0.000725
1       1.922599  0.350797  0.038140
2      -0.774727  0.091374  0.011144
3            0.0  0.000000  0.000000
4            0.0  0.000000  0.000000
Totals            0.447974  0.050010
```

Binning: Purpose

```
# Optimal binning for Purpose
optb_purpose = OptimalBinning(name='Purpose', dtype='categorical', solver='cp')
optb_purpose.fit(X2_train['Purpose'], y2_train)
binning_table_purpose = optb_purpose.binning_table.build()
print("Optimal binning for Purpose:")
print(binning_table_purpose)
```

```
Optimal binning for Purpose:
                Bin  Count  Count (%)  Non-event  Event  Event rate  \
0      [1, 4, 8, 3]    132   0.471429        123      9    0.068182
1               [2]     44   0.157143         39      5    0.113636
2               [6]     14   0.050000         12      2    0.142857
3               [0]     61   0.217857         51     10    0.163934
4            [9, 5]     29   0.103571         23      6    0.206897
5           Special      0   0.000000          0      0    0.000000
6           Missing      0   0.000000          0      0    0.000000
Totals                 280   1.000000        248     32    0.114286

             WoE        IV            JS
0       0.567267  0.121802  1.502437e-02
1       0.006431  0.000006  8.103426e-07
2      -0.255933  0.003612  4.502671e-04
3      -0.418452  0.044714  5.548782e-03
4      -0.703958  0.066706  8.170200e-03
5            0.0  0.000000  0.000000e+00
6            0.0  0.000000  0.000000e+00
Totals            0.236840  2.919443e-02
```

Customized Binning

```
# Optimal binning for Purpose
optb_purpose_adjusted = OptimalBinning(name='Purpose', dtype='categorical', solver='cp', max_n_bins=4)
optb_purpose_adjusted.fit(X2_train['Purpose'], y2_train)
binning_table_purpose_adjusted = optb_purpose_adjusted.binning_table.build()
```

```
print("Optimal binning for Purpose:")
print(binning_table_purpose_adjusted)
```

```
Optimal binning for Purpose:
                 Bin  Count  Count (%)  Non-event  Event  Event rate  \
0       [1, 4, 8, 3]    132   0.471429        123      9    0.068182
1                [2]     44   0.157143         39      5    0.113636
2             [6, 0]     75   0.267857         63     12    0.160000
3             [9, 5]     29   0.103571         23      6    0.206897
4            Special      0   0.000000          0      0    0.000000
5            Missing      0   0.000000          0      0    0.000000
Totals                  280   1.000000        248     32    0.114286

             WoE        IV            JS
0       0.567267  0.121802  1.502437e-02
1       0.006431  0.000006  8.103426e-07
2      -0.389465  0.047113  5.852144e-03
3      -0.703958  0.066706  8.170200e-03
4            0.0  0.000000  0.000000e+00
5            0.0  0.000000  0.000000e+00
Totals            0.235627  2.904753e-02
```

Transform and Encode Training Set

```
# Transform the original variables into their binned categories
X2_train_transformed = X2_train.copy()
X2_train_transformed['History'] = optb_history2.transform(X2_train_transformed['History'], metric='bins
X2_train_transformed['Purpose'] = optb_purpose_adjusted.transform(X2_train_transformed['Purpose'], metr
X2_train_transformed['Amount'] = optb_amount_adjusted.transform(X2_train_transformed['Amount'], metric=
X2_train_transformed['Duration'] = optb_duration2_adjusted.transform(X2_train_transformed['Duration'],



# Ordinal Encoding

# Obtain the bin labels for the binned Duration variable from the binning table
bin_labels = binning_table_duration2_adjusted['Bin'].tolist()
# Create an instance of the OrdinalEncoder with categories set to the sorted bin labels
encoder2 = OrdinalEncoder(categories=[bin_labels], dtype=int)
# Encode the Duration variable in the X1_train_transformed dataframe using the OrdinalEncoder
X2_train_transformed['Duration'] = encoder2.fit_transform(X2_train_transformed[['Duration']])



# Obtain the bin labels for the binned Duration variable from the binning table
bin_labels = binning_table_amount_adjusted['Bin'].tolist()
# Create an instance of the OrdinalEncoder with categories set to the sorted bin labels
encoder2 = OrdinalEncoder(categories=[bin_labels], dtype=int)
# Encode the Duration variable in the X1_train_transformed dataframe using the OrdinalEncoder
X2_train_transformed['Amount'] = encoder2.fit_transform(X2_train_transformed[['Amount']])




# One-hot Encoding
from sklearn.preprocessing import OneHotEncoder

# Instantiate the OneHotEncoder, specify the columns to encode, and set the drop parameter to 'first'
encoder = OneHotEncoder(sparse=False, drop='first')
X2_train_encoded = encoder.fit_transform(X2_train_transformed[['History', 'Purpose']])

# Create a DataFrame with the encoded columns and their respective names
encoded_columns = encoder.get_feature_names_out(['History', 'Purpose'])
X2_train_encoded_df = pd.DataFrame(X2_train_encoded, columns=encoded_columns, index=X2_train_transforme

# Drop the original 'History', 'Purpose', and 'Foreign' columns and concatenate the encoded DataFrame
X2_train_transformed.drop(['History', 'Purpose'], axis=1, inplace=True)
X2_train_transformed = pd.concat([X2_train_transformed, X2_train_encoded_df], axis=1)
```

```
# Select the desired columns from X2_train_transformed
selected_columns = ['History_[3 0]', 'History_[4]', 'Purpose_[2]', 'Purpose_[6 0]', 'Purpose_[9 5]', 'D
X2_train_transformed = X2_train_transformed[selected_columns]
```

Transform and Encode Validation Set

```
# Transform the original variables into their binned categories
X2_val_transformed = X2_val.copy()
X2_val_transformed['History'] = optb_history2.transform(X2_val_transformed['History'], metric='bins')
X2_val_transformed['Purpose'] = optb_purpose_adjusted.transform(X2_val_transformed['Purpose'], metric=
X2_val_transformed['Duration'] = optb_duration2_adjusted.transform(X2_val_transformed['Duration'], metr
X2_val_transformed['Amount'] = optb_amount_adjusted.transform(X2_val_transformed['Amount'], metric='bin




# Ordinal Encoding

# Obtain the bin labels for the binned Duration variable from the binning table
bin_labels = binning_table_duration2_adjusted['Bin'].tolist()
# Create an instance of the OrdinalEncoder with categories set to the sorted bin labels
encoder2 = OrdinalEncoder(categories=[bin_labels], dtype=int)
# Encode the Duration variable in the X1_train_transformed dataframe using the OrdinalEncoder
X2_val_transformed['Duration'] = encoder2.fit_transform(X2_val_transformed[['Duration']])



# Obtain the bin labels for the binned Duration variable from the binning table
bin_labels = binning_table_amount_adjusted['Bin'].tolist()
# Create an instance of the OrdinalEncoder with categories set to the sorted bin labels
encoder2 = OrdinalEncoder(categories=[bin_labels], dtype=int)
# Encode the Duration variable in the X1_train_transformed dataframe using the OrdinalEncoder
X2_val_transformed['Amount'] = encoder2.fit_transform(X2_val_transformed[['Amount']])


# One-hot Encoding
from sklearn.preprocessing import OneHotEncoder

# Instantiate the OneHotEncoder, specify the columns to encode, and set the drop parameter to 'first'
encoder = OneHotEncoder(sparse=False, drop='first')
X2_val_encoded = encoder.fit_transform(X2_val_transformed[['History', 'Purpose']])

# Create a DataFrame with the encoded columns and their respective names
encoded_columns = encoder.get_feature_names_out(['History', 'Purpose'])
X2_val_encoded_df = pd.DataFrame(X2_val_encoded, columns=encoded_columns, index=X2_val_transformed.inde

# Drop the original 'History', 'Purpose', and 'Foreign' columns and concatenate the encoded DataFrame
X2_val_transformed.drop(['History', 'Purpose'], axis=1, inplace=True)
X2_val_transformed = pd.concat([X2_val_transformed, X2_val_encoded_df], axis=1)

# Select the desired columns from X2_val_transformed
X2_val_transformed = X2_val_transformed[selected_columns]

X2_val_transformed
```

Binary variables Subset 2

```
X2_train_transformed
```

	History_[3 0]	History_[4]	Purpose_[2]	Purpose_[6 0]	Purpose_[9 5]	Duration	Amount
231	0.0	0.0	0.0	1.0	0.0	1	0
190	0.0	0.0	0.0	0.0	1.0	3	2
232	0.0	0.0	0.0	0.0	0.0	1	0
786	0.0	0.0	0.0	0.0	0.0	3	0
314	0.0	0.0	0.0	1.0	0.0	0	0
...	...	...	...	...	...	...	...
263	0.0	1.0	0.0	1.0	0.0	1	0
660	0.0	0.0	0.0	0.0	0.0	1	0
210	0.0	1.0	0.0	0.0	0.0	1	1

Q4) Model implementation

280 rows × 7 columns

```
from sklearn.linear_model import LinearRegression, LogisticRegression

# Split the transformed training set into the feature matrix (X) and target vector (y)
X_train1 = X1_train_transformed
y_train1 = y1_train

# Train a linear regression model for Checking = 1 or 2
linear_model1 = LinearRegression()
linear_model1.fit(X_train1, y_train1)

# Train a logistic regression model for Checking = 1 or 2
logistic_model1 = LogisticRegression(max_iter=1000)
logistic_model1.fit(X_train1, y_train1)


# Split the transformed training set into the feature matrix (X) and target vector (y)
X_train2 = X2_train_transformed
y_train2 = y2_train

# Train a linear regression model for Checking = 3 or 4
linear_model2 = LinearRegression()
linear_model2.fit(X_train2, y_train2)

# Train a logistic regression model for Checking = 3 or 4
logistic_model2 = LogisticRegression(max_iter=1000)
logistic_model2.fit(X_train2, y_train2)
```

```
LogisticRegression
LogisticRegression(max_iter=1000)
```

Coefficents

```
def display_coefficients(model, feature_names, model_name, dataset_name):
    coef = model.coef_
    intercept = model.intercept_
    print(f"Model: {model_name} - {dataset_name}")
    print("Intercept:", intercept)
    print("Coefficients:")
    if len(coef.shape) == 2:
        coef = coef.flatten()
    for feature, coeff in zip(feature_names, coef):
        print(f"{feature}: {coeff}")
    print("\n")
```

```
# Display coefficients for all models
display_coefficients(linear_model1, X1_train_transformed.columns, "Linear Regression", "Checking = 1 or
display_coefficients(logistic_model1, X1_train_transformed.columns, "Logistic Regression", "Checking =
display_coefficients(linear_model2, X2_train_transformed.columns, "Linear Regression", "Checking = 3 or
display_coefficients(logistic_model2, X2_train_transformed.columns, "Logistic Regression", "Checking =
```

```
Model: Linear Regression - Checking = 1 or 2
Intercept: 0.3591633723403901
Coefficients:
History_[1]: 0.006072493418712775
History_[2]: -0.1612201612875368
History_[3]: -0.24001529547002642
History_[4]: -0.3425250588515163
Property_[3 2]: 0.11044157061144544
Property_[4]: 0.21635410752524742
Duration: 0.08912687111327311
Age: -0.01697970183041357


Model: Logistic Regression - Checking = 1 or 2
Intercept: [-1.06911445]
Coefficients:
History_[1]: 0.3641174172701093
History_[2]: -0.3178785900685967
History_[3]: -0.5914775410846044
History_[4]: -1.1468535144273795
Property_[3 2]: 0.4641011673229343
Property_[4]: 0.8709754236222877
Duration: 0.44101727461791385
Age: -0.08779297549529017


Model: Linear Regression - Checking = 3 or 4
Intercept: 0.047438761341134214
Coefficients:
History_[3 0]: 0.18282654551665628
History_[4]: -0.065717289663977
Purpose_[2]: 0.048480317592384053
Purpose_[6 0]: 0.10145717106573528
Purpose_[9 5]: 0.06942370203986799
Duration: 0.017447480326388636
Amount: -0.0010443270399388901


Model: Logistic Regression - Checking = 3 or 4
Intercept: [-2.62910557]
Coefficients:
History_[3 0]: 0.9846737794542854
History_[4]: -0.7540548498542532
Purpose_[2]: 0.33369614789284796
Purpose_[6 0]: 0.7994972043335203
Purpose_[9 5]: 0.4935544253195002
Duration: 0.15561495756397017
Amount: 0.011007869672980307
```

Q5) Performance Evaluation - ROC, GINI, KS

```
import numpy as np
from sklearn.metrics import roc_curve, roc_auc_score, confusion_matrix, auc
import numpy as np
from scipy.special import expit

# Define a function to transform linear regression output to probabilities using the sigmoid function
def lr_to_proba(y_pred):
    return expit(y_pred)

def calc_gini(auc_score):
    return (2 * auc_score) - 1
```

```
def calc_ks(y_true, y_pred_prob):
    fpr, tpr, thresholds = roc_curve(y_true, y_pred_prob)
    return np.max(tpr - fpr)

# Train and obtain predicted probabilities for the validation set
# Use models trained in previous steps
y1_pred_lr = linear_model1.predict(X1_val_transformed)
y1_pred_prob_lr = lr_to_proba(y1_pred_lr)
y1_pred_prob_logreg = logistic_model1.predict_proba(X1_val_transformed)[:, 1]
y2_pred_lr = linear_model2.predict(X2_val_transformed)
y2_pred_prob_lr = lr_to_proba(y2_pred_lr)
y2_pred_prob_logreg = logistic_model2.predict_proba(X2_val_transformed)[:, 1]

# Calculate the sensitivity, specificity, and thresholds for each scorecard
fpr1_lr, tpr1_lr, thresholds1_lr = roc_curve(y1_val, y1_pred_prob_lr)
fpr1_logreg, tpr1_logreg, thresholds1_logreg = roc_curve(y1_val, y1_pred_prob_logreg)
fpr2_lr, tpr2_lr, thresholds2_lr = roc_curve(y2_val, y2_pred_prob_lr)
fpr2_logreg, tpr2_logreg, thresholds2_logreg = roc_curve(y2_val, y2_pred_prob_logreg)

# Plot the ROC curves for each scorecard
plt.figure()
plt.plot(fpr1_lr, tpr1_lr, label="Linear Regression - Checking = 1 or 2")
plt.plot(fpr1_logreg, tpr1_logreg, label="Logistic Regression - Checking = 1 or 2")
plt.plot(fpr2_lr, tpr2_lr, label="Linear Regression - Checking = 3 or 4")
plt.plot(fpr2_logreg, tpr2_logreg, label="Logistic Regression - Checking = 3 or 4")
plt.plot([0, 1], [0, 1], linestyle="--")
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.title("ROC Curves")
plt.legend()
plt.show()

# Calculate the AUC, Gini coefficient, and KS values for each scorecard
auc1_lr = roc_auc_score(y1_val, y1_pred_prob_lr)
auc1_logreg = roc_auc_score(y1_val, y1_pred_prob_logreg)
auc2_lr = roc_auc_score(y2_val, y2_pred_prob_lr)
auc2_logreg = roc_auc_score(y2_val, y2_pred_prob_logreg)

gini1_lr = calc_gini(auc1_lr)
gini1_logreg = calc_gini(auc1_logreg)
gini2_lr = calc_gini(auc2_lr)
gini2_logreg = calc_gini(auc2_logreg)

ks1_lr = calc_ks(y1_val, y1_pred_prob_lr)
ks1_logreg = calc_ks(y1_val, y1_pred_prob_logreg)
ks2_lr = calc_ks(y2_val, y2_pred_prob_lr)
ks2_logreg = calc_ks(y2_val, y2_pred_prob_logreg)

# Print the AUC, Gini coefficient, and KS values for each scorecard
print("Linear Regression - Checking = 1 or 2")
print("AUC:", auc1_lr)
print("Gini Coefficient:", gini1_lr)
print("KS:", ks1_lr)
print("\nLogistic Regression - Checking = 1 or 2")
print("AUC:", auc1_logreg)
print("Gini Coefficient:", gini1_logreg)
print("KS:", ks1_logreg)
print("\nLinear Regression - Checking = 3 or 4")
print("AUC:", auc2_lr)
print("Gini Coefficient:", gini2_lr)
print("KS:", ks2_lr)
print("\nLogistic Regression - Checking = 3 or 4")
print("AUC:", auc2_logreg)
print("Gini Coefficient:", gini2_logreg)
print("KS:", ks2_logreg)
```

```
import matplotlib.pyplot as plt


fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(12, 6))

# Plot the ROC curves for Checking = 1 or 2
ax1.plot(fpr1_lr, tpr1_lr, label="Linear Regression")
ax1.plot(fpr1_logreg, tpr1_logreg, label="Logistic Regression")
ax1.plot([0, 1], [0, 1], linestyle="--")
ax1.set_xlabel("False Positive Rate")
ax1.set_ylabel("True Positive Rate")
ax1.set_title("ROC Curves - Checking = 1 or 2")
ax1.legend()

# Plot the ROC curves for Checking = 3 or 4
ax2.plot(fpr2_lr, tpr2_lr, label="Linear Regression")
ax2.plot(fpr2_logreg, tpr2_logreg, label="Logistic Regression")
ax2.plot([0, 1], [0, 1], linestyle="--")
ax2.set_xlabel("False Positive Rate")
ax2.set_ylabel("True Positive Rate")
ax2.set_title("ROC Curves - Checking = 3 or 4")
ax2.legend()

plt.show()
```