Tinbergen Institute
Machine Learning | Block Two

Risk at First Sight

Default-Risk Prediction & Screening at Loan Origination in P2P Consumer Lending, with a Double Machine Learning Extension of the Effects of Longer Terms and High Interest Rates

Student Number: **756486**

December 15, 2025

Disclosure

In the preparation of this research project, AI has been used for the following purposes:

1. **Idea development and discussion:** to bounce off ideas, explore alternative formulations, and stress-test the logic behind the research design.
2. **Validity checks (reasoning & implementation):** to help check the consistency of mathematical steps, identify possible mistakes, and diagnose issues in code implementation (e.g., debugging suggestions and sanity checks).
3. **Clarity and presentation:** to improve English phrasing, structure, and readability, and to refine or simplify LaTeX and mathematical notation when needed.
4. **Code quality and maintainability:** to improve code structure, readability, and organization (e.g., refactoring, clearer naming, comments, and modularization) without changing the intended analysis.

All written content, methodological choices, analyses, and interpretations reflect my own reasoning and final decisions. AI outputs were treated as suggestions, reviewed critically, and edited or discarded when necessary. Any remaining errors are my responsibility. AI was not used to fabricate data, results, or references; all sources and claims are verified by the author.

1 Introduction

Online peer-to-peer (P2P) lending has emerged as a significant alternative to traditional banking, directly connecting borrowers and lenders and shifting credit risk from intermediaries to investors (Lenz, 2016). Because these loans are typically unsecured and investors often lack the expertise to evaluate creditworthiness, P2P markets face severe information asymmetry and a heightened risk of adverse selection (Byanjankar et al., 2015; Emekter et al., 2015; Jin & Zhu, 2015). In practice, default-risk assessment becomes a *screening* problem under limited attention: investors and platforms must decide which applications to flag first, not merely estimate probabilities. Moreover, evidence suggests that charging higher interest rates to riskier borrowers may be insufficient to offset expected losses, reinforcing the need for effective screening at origination (Emekter et al., 2015).

In this paper, we ask: *How well can we predict default risk at loan origination using borrower and loan characteristics, and what does that imply for screening decisions?* We study LendingClub loans and restrict the feature set strictly to origination-time variables to prevent data leakage. We benchmark standard statistical approaches against commonly used machine learning methods that may better capture non-linear patterns in borrower characteristics (Byanjankar et al., 2015; Jin & Zhu, 2015). Unlike much of the existing P2P credit-scoring literature, which focuses mainly on generic predictive metrics, we explicitly link model performance to screening outcomes and to the distribution of access to credit. Our evaluation therefore emphasizes not only predictive performance but also ranking quality, which determines how effectively a model supports screening decisions.

As a complementary extension, we examine how default risk differs across two salient contract features: loan maturity (60 vs. 36 months) and interest rates. Because these terms may be endogenous to borrower risk, we first report naive differences in default rates as descriptive benchmarks and then use double machine learning (DML) to estimate *conditional* treatment effects while flexibly controlling for origination-time covariates, interpreting these estimates as adjusted associations and remaining cautious about causal claims (Emekter et al., 2015; Serrano-Cinca et al., 2015).

The remainder of the paper is organized as follows: Section 2 describes the data, Section 3 outlines the modeling approach, Section 4 presents the results, and Section 5 concludes.

2 Data

To study default-risk prediction and screening *at loan origination*, we use a widely used LendingClub loan-level dataset distributed via Kaggle. LendingClub is a major U.S. online marketplace for consumer credit in which investors fund unsecured installment loans; it was incorporated in 2006 and launched its marketplace in 2007 (LendingClub Corporation, 2014). The raw file contains 2,260,701 loan records with 151 variables spanning issuance years 2007–2018, covering borrower attributes, contract terms, platform risk measures, and loan outcomes.

Because our question is explicitly about *screening at origination*, we restrict features to variables observable at issuance and construct a clean default outcome from the loan-status field. Table 4 summarizes the processing steps (create issuance year, harmonize status labels, drop invalid records). To avoid right-censoring, we focus on loans issued in 2012–2015; as Figure 5 shows, later cohorts contain many active loans, which would mechanically bias default rates downward and contaminate screening evaluation.

Even after this restriction, the data remain high-dimensional (over 100 variables), and one-hot expansion of categorical features would exceed local computational limits. We therefore use a stratified sample (by issuance year and default outcome) and literature-guided feature selection, remove identifiers and post-origination leakage fields (e.g., realized payments), standardize factor levels, and engineer a parsimonious origination-time predictor set. Our binary outcome equals one

for loans that end in default and zero otherwise under the harmonized status definition. Predictors are origination-time borrower and contract characteristics used in screening, including credit-quality proxies (e.g., credit grade and FICO band), pricing (interest rate), maturity (36 vs. 60 months), and affordability measures (e.g., income and debt-to-income).

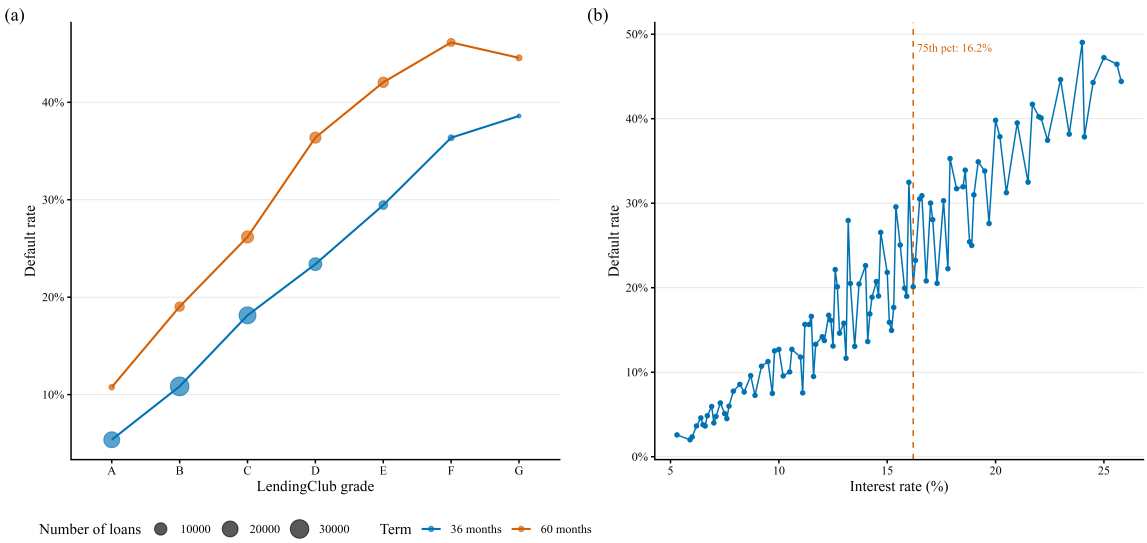

Figure 1: *Default rates by LendingClub grade and loan term, and by interest rate.*

The cohort exhibits strong monotone risk gradients in the platform's internal risk signals and in observed pricing. Figure 1(a) shows default rates rising sharply from grade A to grade G, with 60-month loans defaulting more often than 36-month loans within each grade, consistent with borrower-risk sorting and a maturity-risk component. Figure 1(b) shows a steep but noisy positive relationship between interest rates and default rates, consistent with risk-based pricing; because rates are assigned to observably riskier applicants, raw differences across rate levels are not immediately causal.

Risk also varies along joint credit dimensions. Figure 4 reports default rates by credit grade and FICO band separately for 36- and 60-month loans: within each term, default generally rises as grade worsens and FICO weakens, and the 60-month panel is uniformly riskier, especially in lower-grade segments. Together, these descriptives establish two empirical facts that guide the remainder of the paper: (i) default risk is highly predictable from origination-time information in economically meaningful ways, which makes screening a natural objective; and (ii) maturity and interest rate are strongly related to default in the raw data, but are also plausibly chosen *because* of borrower risk, which motivates a cautious, adjustment-based treatment-effect extension.

3 Methods

Prediction problem and information set. Our goal is to support *screening at loan origination*. For each loan i we observe origination-time features $\mathbf{x}_i$ and a binary default outcome $y_i \in \{0, 1\}$ (default $= 1$) from the harmonized status field. With $\mathcal{Z} = \{(\mathbf{x}_i, y_i)\}_{i=1}^{n}$, models output scores

$$\hat{p}_i = \hat{p}(\mathbf{x}_i) \approx \Pr(Y_i = 1 \mid \mathbf{x}_i),$$

interpreted as estimated default probabilities and used to rank borrowers by risk. A threshold rule at $\tau \in (0, 1)$ is $\hat{y}(\tau) = \mathbb{I}(\hat{p} \geq \tau)$.

Training, testing, and progressive evaluation logic. All models are compared on a common held-out test set. Any data-dependent preprocessing and tuning (including cross-validation and bootstrap aggregation) is performed on the training set only; the final fitted model is then applied once to the test set to obtain out-of-sample scores $\hat{p}_i$. We proceed from linear benchmarks (logit/elastic net) to single trees, then to bagging and random forests, and end with a DML extension. Diagnostics in Section 4 are defined once and applied uniformly; for bagging and forests we also use out-of-bag (OOB) diagnostics as an internal training check. All procedures are implemented in R and documented in the code appendix; the bagging routine and the DML estimator are coded from scratch (no high-level implementations).

Linear probability models as stable baselines: logit and elastic net. Logistic regression links default risk to a linear index,

$$\Pr(Y_i = 1 \mid \mathbf{x}_i) = \sigma(\eta_i), \qquad \eta_i = \beta_0 + \mathbf{x}_i^\top \boldsymbol{\beta}, \qquad \sigma(t) = \frac{1}{1 + e^{-t}},$$

so features enter additively in log-odds. This is interpretable and stable, but can underfit with many correlated predictors and nonlinearities.

We therefore also fit elastic-net penalized logistic regression. Writing $\hat{p}_i(\beta_0, \boldsymbol{\beta}) = \sigma(\beta_0 + \mathbf{x}_i^\top \boldsymbol{\beta})$, the estimator solves

$$\min_{\beta_0, \boldsymbol{\beta}} \frac{1}{n} \sum_{i=1}^{n} \big[-y_i \log \hat{p}_i - (1 - y_i) \log(1 - \hat{p}_i)\big] + \lambda\Big(\alpha \|\boldsymbol{\beta}\|_1 + \tfrac{1-\alpha}{2} \|\boldsymbol{\beta}\|_2^2\Big),$$

where $\lambda \geq 0$ controls shrinkage and $\alpha \in [0, 1]$ mixes ℓ_1 (sparsity/selection) with ℓ_2 (stability under collinearity). Predictors are standardized prior to estimation. Elastic net is typically effective when signals are spread across many correlated variables and the decision boundary is approximately linear.

Classification trees (CART) as flexible but high-variance learners. Trees partition the feature space into regions $\{R_m\}_{m=1}^{M}$ and predict a region-specific default probability (the within-node default rate). Splits are chosen greedily to reduce impurity, e.g. the Gini index

$$G = 1 - p_0^2 - p_1^2 = 2p_1(1 - p_1),$$

where p_1 is the node default share. Trees capture nonlinearities and interactions and handle mixed variable types, but are high-variance: small data changes can alter early splits and the entire tree.

Bagging (bootstrap aggregating). Bagging reduces variance by averaging many trees. Fix B bootstraps. For $b = 1, \ldots, B$: (i) draw a bootstrap sample $\mathcal{Z}_b^*$ of size n from $\mathcal{Z}$; (ii) fit a classification tree to obtain $\hat{p}_b^*(\mathbf{x})$; (iii) store bootstrap membership indicators for OOB prediction. The bagged probability (soft voting) is

$$\hat{p}_{\text{bag}}(\mathbf{x}) = \frac{1}{B} \sum_{b=1}^{B} \hat{p}_b^*(\mathbf{x}), \qquad \hat{y}_{\text{bag}}(\tau) = \mathbb{I}\big(\hat{p}_{\text{bag}}(\mathbf{x}) \geq \tau\big),$$

which preserves probabilistic information and typically improves ranking/calibration relative to hard voting.

For observation i, let $\mathcal{B}_i$ be the set of trees where i is out-of-bag. The OOB score is

$$\hat{p}_i^{\text{OOB}} = \frac{1}{|\mathcal{B}_i|} \sum_{b \in \mathcal{B}_i} \hat{p}_b^*(\mathbf{x}_i),$$

providing an internal performance estimate without a separate validation set. We also track OOB error as a function of the number of trees to confirm stabilization.

Random forests. If a few strong predictors dominate, bagged trees can become too similar. Random forests inject additional randomness by selecting m candidate predictors at each split (rather than all p). With B bootstrap trees, the forest probability is

$$\hat{p}_{\mathrm{rf}}(\mathbf{x}) = \frac{1}{B}\sum_{b=1}^{B} \hat{p}^{*}_{b,\mathrm{rf}}(\mathbf{x}),$$

with OOB diagnostics defined as above. For screening, forests retain tree flexibility while typically improving variance reduction under correlated predictors, at the cost of interpretability.

Diagnostics and performance measures. Let $\mathcal{T}$ be the common test set of size n_{test}. For threshold τ, define $\hat{y}_i(\tau) = \mathbb{I}(\hat{p}_i \geq \tau)$ and confusion-matrix counts $\mathrm{TP}(\tau), \mathrm{FP}(\tau), \mathrm{TN}(\tau), \mathrm{FN}(\tau)$ in the usual way. We report:

- *Misclassification error* and *accuracy*,

$$\mathrm{Error}(\tau) = \frac{\mathrm{FP}(\tau) + \mathrm{FN}(\tau)}{n_{\mathrm{test}}}, \qquad \mathrm{Accuracy}(\tau) = 1 - \mathrm{Error}(\tau),$$

  which are intuitive but can mislead under class imbalance.

- *Brier score* for probability accuracy and calibration,

$$\mathrm{Brier} = \frac{1}{n_{\mathrm{test}}}\sum_{i\in\mathcal{T}} (y_i - \hat{p}_i)^2,$$

  which penalizes both poor ranking and miscalibration (confidently wrong scores are costly).

- *ROC curve and AUC*. For $\tau \in (0,1)$,

$$\mathrm{TPR}(\tau) = \frac{\mathrm{TP}(\tau)}{\mathrm{TP}(\tau) + \mathrm{FN}(\tau)}, \qquad \mathrm{FPR}(\tau) = \frac{\mathrm{FP}(\tau)}{\mathrm{FP}(\tau) + \mathrm{TN}(\tau)}.$$

  The ROC curve is $\{(\mathrm{FPR}(\tau), \mathrm{TPR}(\tau))\}$ and

$$\mathrm{AUC}_{\mathrm{ROC}} = \int_0^1 \mathrm{TPR}(u)\, du,$$

  interpretable as the probability a randomly chosen default receives a higher score than a randomly chosen non-default.

- *Precision–Recall (PR) curve*. At threshold τ:

$$\mathrm{Precision}(\tau) = \frac{\mathrm{TP}(\tau)}{\mathrm{TP}(\tau) + \mathrm{FP}(\tau)}, \qquad \mathrm{Recall}(\tau) = \mathrm{TPR}(\tau),$$

  useful under imbalance because precision answers: among flagged loans, what fraction actually default?

- *Screening curves (cumulative gains and lift).* Sort test loans by decreasing $\hat{p}_i$ and let $u \in (0,1]$ be the screened population share:

$$G(u) = \frac{\sum_{j=1}^{\lfloor u n_{\mathrm{test}} \rfloor} y_{(j)}}{\sum_{j=1}^{n_{\mathrm{test}}} y_{(j)}}, \qquad L(u) = \frac{G(u)}{u},$$

  which quantify how many defaults are captured by screening the top u fraction and how much this improves over random screening.

Double/Debiased Machine Learning (DML). As a causal add-on, we study how default risk is associated with binary origination-time treatments D (60-month vs. 36-month term; and a high-interest indicator) after flexibly adjusting for remaining covariates $\mathbf{X}$. Because contract terms may be endogenous to borrower risk, naive differences can be misleading (Emekter et al., 2015; Serrano-Cinca et al., 2015). Following (Chernozhukov et al., 2016, 2018), we use the partially linear model

$$Y = \theta_0 D + g_0(\mathbf{X}) + \varepsilon, \qquad D = m_0(\mathbf{X}) + v,$$

where $g_0(\mathbf{X})$ captures baseline default risk and $m_0(\mathbf{X})$ the propensity of treatment given $\mathbf{X}$; θ_0 summarizes the average adjusted association between D and Y.

The key idea is to estimate θ_0 by "regressing residuals on residuals." We estimate g_0 and m_0 via cross-fitted elastic-net logistic regression: split the sample into K folds; for each fold k, fit nuisance models on I_k^c and predict $\hat{g}^{(-k)}(\mathbf{X}_i)$ and $\hat{m}^{(-k)}(\mathbf{X}_i)$ for $i \in I_k$. This yields orthogonalized residuals

$$\tilde{Y}_i = Y_i - \hat{g}(\mathbf{X}_i), \qquad \tilde{D}_i = D_i - \hat{m}(\mathbf{X}_i),$$

where each observation is predicted by models trained without that observation. The DML estimator is the least-squares slope

$$\hat{\theta}_{\text{DML}} = \frac{\sum_{i=1}^{n} \tilde{D}_i \tilde{Y}_i}{\sum_{i=1}^{n} \tilde{D}_i^2}.$$

Because $\tilde{D}_i$ is approximately uncorrelated with functions of $\mathbf{X}_i$, first-stage errors affect $\hat{\theta}_{\text{DML}}$ only to second order (Neyman orthogonality), reducing regularization bias relative to naive plug-in approaches.

For comparison we also report the naive difference in default rates,

$$\hat{\theta}_{\text{naive}} = \bar{Y}_{D=1} - \bar{Y}_{D=0},$$

so the gap between $\hat{\theta}_{\text{naive}}$ and $\hat{\theta}_{\text{DML}}$ indicates how strongly observable characteristics mediate the relationship between contract terms and default.

4 Results

Having set up the prediction problem and modelling framework, we now turn to the empirical performance of the six classifiers and to the treatment-effect extension on contract terms.

Overall predictive performance. Table 1 reports test-set error, accuracy, AUC, and Brier scores on the common hold-out sample. All models achieve similar accuracy (around 82%) in a setting where the unconditional default rate is roughly 18.5%. The carefully tuned logit and elastic-net models provide a very strong baseline: they attain the highest or near-highest AUC (about 0.73) and the lowest Brier scores. Random forests perform comparably, while both bagging variants and especially CART lag somewhat, with lower AUC and slightly higher error. Thus, there is clear signal in origination-time characteristics, but moving from a well-specified penalized linear model to more complex tree ensembles yields only modest gains in standard predictive metrics.

These patterns align with prior findings that logistic regression can match more complex techniques in P2P lending and that model performance is often similar across methods (Jin & Zhu, 2015; Serrano-Cinca et al., 2015).

Ranking quality and screening implications. Because screening is fundamentally about ranking rather than choosing a single threshold, Figure 2 and Figure 3 focus on ordering performance. The ROC and precision–recall curves in Figure 2 are broadly similar for logit, elastic net, and random forest, which dominate bagging and CART over most thresholds.

Table 1: *Test-set performance comparison (common test set)*

Model	Error	Accuracy	AUC	Brier	TP	FP	TN	FN
Logit	0.182	0.818	0.727	0.135	486	322	28,969	6,223
Elastic net	0.182	0.818	0.727	0.135	488	314	28,977	6,221
Random forest	0.182	0.818	0.725	0.136	415	273	29,018	6,294
Bagging (ipred)	0.183	0.817	0.704	0.152	526	394	28,897	6,183
Bagging (custom)	0.183	0.817	0.690	0.138	402	288	29,003	6,307
CART	0.184	0.816	0.666	0.140	552	484	28,807	6,157

This means that for a given tolerance for false positives, these models recover a larger share of eventual defaulters and maintain higher precision when targeting the riskiest slice of applicants.

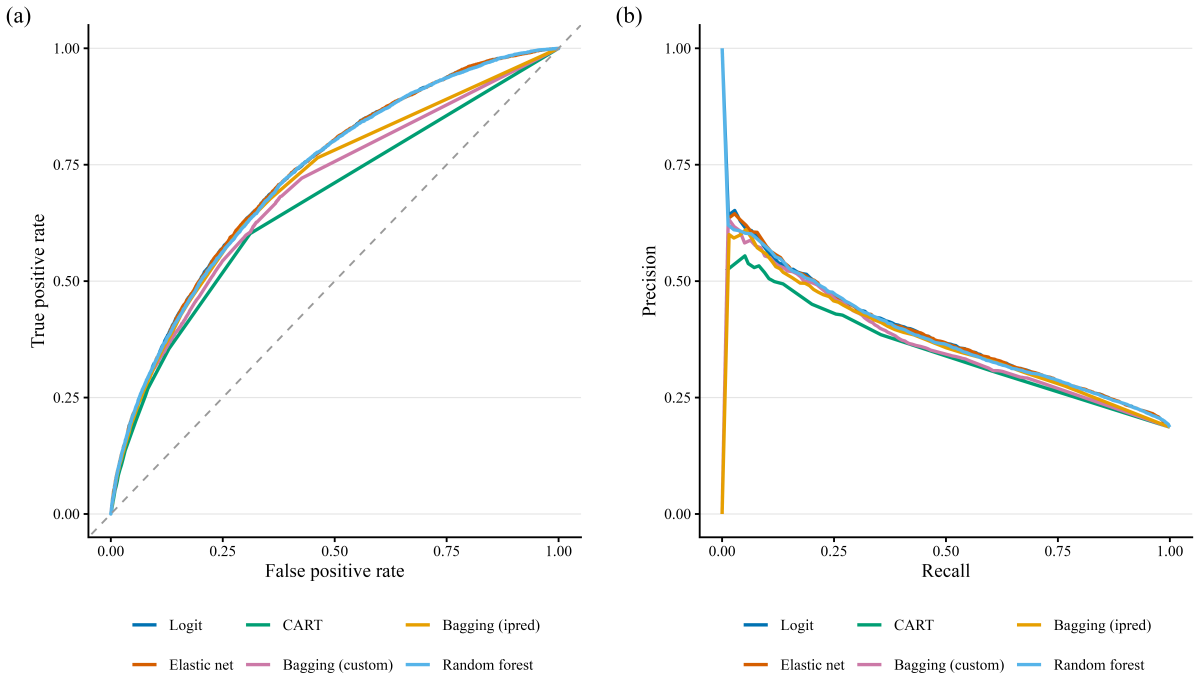

Figure 2: *ROC and precision–recall curves by model.*

The cumulative gains curve in Figure 3 translates this into a screening scenario. If an investor can manually review only the top 20–30% of applicants, the best-performing models capture roughly half of all future defaults, versus about 20–30% under random screening (the 45-degree line). Random forests sit slightly above logit and elastic net, but the gap is small; the main contrast is between any of these models and naive screening. The practical message is therefore twofold: even an interpretable, well-tuned linear model substantially improves who to flag first, while tree ensembles offer at most incremental gains in this dataset.

Predicted risk distributions. Figure 6 plots predicted default probabilities by true outcome for each classifier. Under logit and elastic net, non-defaults concentrate at low predicted risk while defaults shift toward the right tail, yielding clear but overlapping separation. Tree-based models produce more irregular or multi-modal shapes because they average discrete terminal-node probabilities, but

the qualitative ranking remains: defaulters are systematically assigned higher scores. The bagging models place more mass at very low probabilities, consistent with conservative predictions and their slightly worse Brier scores. Overall, the distributions confirm meaningful risk sorting, with calibration differences across algorithms.

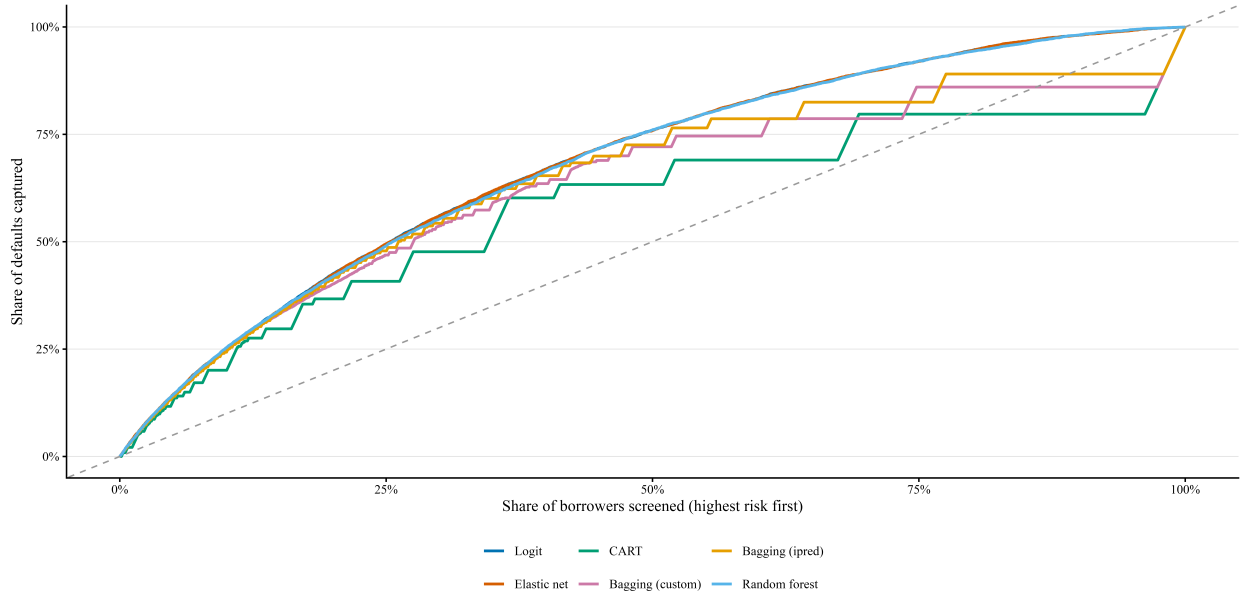

Figure 3: *Cumulative gains / lift curves by model.*

Tree-ensemble diagnostics. Table 2 and the out-of-bag (OOB) error path in Figure 7 provide additional checks for the ensemble methods. For the custom bagging implementation, OOB misclassification error declines with the number of trees and stabilizes around 18.4% at roughly 70 trees, slightly improving on the single-value OOB error of the off-the-shelf `ipred` bagging model. Random forests attain the lowest OOB error and highest OOB AUC among the tree-based methods, consistent with their held-out test performance. These diagnostics support our modelling choices: the custom bagging routine behaves as expected and the random forest is a strong nonlinear benchmark, even though its gains over simpler models remain limited.

Table 2: *Out-of-bag (OOB) performance summary*

Method	Number of Trees	OOB error (misclassification)	OOB AUC	OOB Brier
Bagging (custom)	80	0.184	0.677	0.140
Bagging (ipred)	80	0.185		
Random forest (ranger)	300	0.137		

Double machine learning extension. Finally, we examine how default risk differs across contract terms once we adjust for observed borrower characteristics. In Table 3, the naive difference-in-means implies that 60-month loans default about 18.3 percentage points more often than 36-month loans, with a tight confidence interval. By contrast, the orthogonalised DML estimator is numerically unstable: the point estimate becomes very large and the 95% confidence interval is extremely wide, spanning negative values and implausibly large positives. The reason is overlap: conditional on the rich origination covariates, term length is almost perfectly predictable from X, leaving too little

residual variation in treatment. Econometrically, this near-violation of overlap means the term-length DML estimate is effectively not identified in our design, so we avoid causal claims about moving from 36 to 60 months.

Table 3: *Naive vs DML estimates for two treatments*

Treatment	Estimator	Effect (pp) [95% CI]
60-month term (vs 36-month)	Naive diff-in-means	18.28 [17.59, 18.96]
60-month term (vs 36-month)	DML (elastic net, K-fold)	67.69 [-343.62, 479.01]
High interest (>= 75th pct)	Naive diff-in-means	18.19 [17.50, 18.87]
High interest (>= 75th pct)	DML (elastic net, K-fold)	1.74 [-4.87, 8.34]

The picture is clearer for pricing. High-interest loans (above the 75th percentile) have about 18.2 percentage points higher default rates in the raw data, consistent with the interest-rate gradient in Figure 1. After flexible adjustment via DML, the estimated effect shrinks to roughly 1.7 percentage points and the confidence interval includes zero. Hence most of the naive association appears driven by selection on observables: borrowers who look riskier are both more likely to receive high rates and more likely to default. Put differently, conditional on observables, LendingClub appears primarily to *price* risk rather than *create* it through high interest rates.

5 Conclusion

This paper examined how well default risk can be predicted *at loan origination* in a large P2P consumer lending platform, and what this implies for screening and contract terms. Using LendingClub loan-level data with an origination-only information set, we compared logistic regression and elastic net with tree-based methods. Across Table 1 and Figure 2, default risk is highly predictable from standard borrower and loan characteristics: a well-tuned (penalized) logit already attains strong accuracy, AUC, and calibration. Operationally, Figure 3 shows that reviewing the riskiest 20–30% of applicants captures about half of all subsequent defaults, far above random selection, so simple, transparent models already provide substantial *"risk at first sight"* signals.

Tree-based models behave as expected but add only modest improvements. Bagging and random forests deliver slightly better ranking (and stable OOB diagnostics in Table 2), yet their gains over elastic net are small relative to the increase in model complexity, implying only marginal benefits from moving to more opaque ensembles.

The double machine learning extension highlights the limits of causal interpretation. While naive comparisons link 60-month terms and high interest rates to much higher default, DML shrinks the high-interest association to a small, statistically inconclusive estimate and yields an unstable, very wide interval for term length (Table 3). This is consistent with strong selection on observables: riskier borrowers are more likely to receive longer terms or higher rates and to default, so contract terms largely *price* risk rather than clearly *cause* it. Our analysis is limited to one platform, cohort, and observables; future work could use richer data and quasi-experimental variation in pricing or term policies to sharpen causal claims. Within these bounds, the main message is that in P2P consumer lending, much of default risk is visible at origination, and exploiting this information through well-calibrated, transparent models can meaningfully improve screening performance while keeping distributional impacts on access to credit more interpretable.

References

Byanjankar, A., Heikkilä, M., & Mezei, J. (2015). Predicting credit risk in peer-to-peer lending: A neural network approach. *2015 IEEE Symposium Series on Computational Intelligence*, 719–725.

Chernozhukov, V., Chetverikov, D., Demirer, M., Duflo, E., Hansen, C., Newey, W., & Robins, J. (2016). Double/debiased machine learning for treatment and causal parameters. https://arxiv.org/abs/1608.00060

Chernozhukov, V., Chetverikov, D., Demirer, M., Duflo, E., Hansen, C., Newey, W. K., & Robins, J. (2018). Double/debiased machine learning for treatment and structural parameters. *The Econometrics Journal*, *21*(1), C1–C68. https://doi.org/10.1111/ectj.12097

Emekter, R., Tu, Y., Jirasakuldech, B., & Lu, M. (2015). Evaluating credit risk and loan performance in online peer-to-peer (p2p) lending. *Applied Economics*, *47*(1), 54–70.

Jin, Y., & Zhu, Y. (2015). A data-driven approach to predict default risk of loan for online peer-to-peer (p2p) lending [Also appearing in 2015 Fifth International Conference on Communication Systems and Network Technologies]. *Procedia Computer Science*.

LendingClub Corporation. (2014). Form s-1 registration statement [As filed Aug. 27, 2014. U.S. Securities and Exchange Commission].

Lenz, R. (2016). Peer-to-peer lending: Opportunities and risks. *European Journal of Risk Regulation*, *7*(4), 688–700.

Serrano-Cinca, C., Gutiérrez-Nieto, B., & López-Palacios, L. (2015). Determinants of default in p2p lending. *PloS one*, *10*(10), e0139427.

Appendix

A Figures

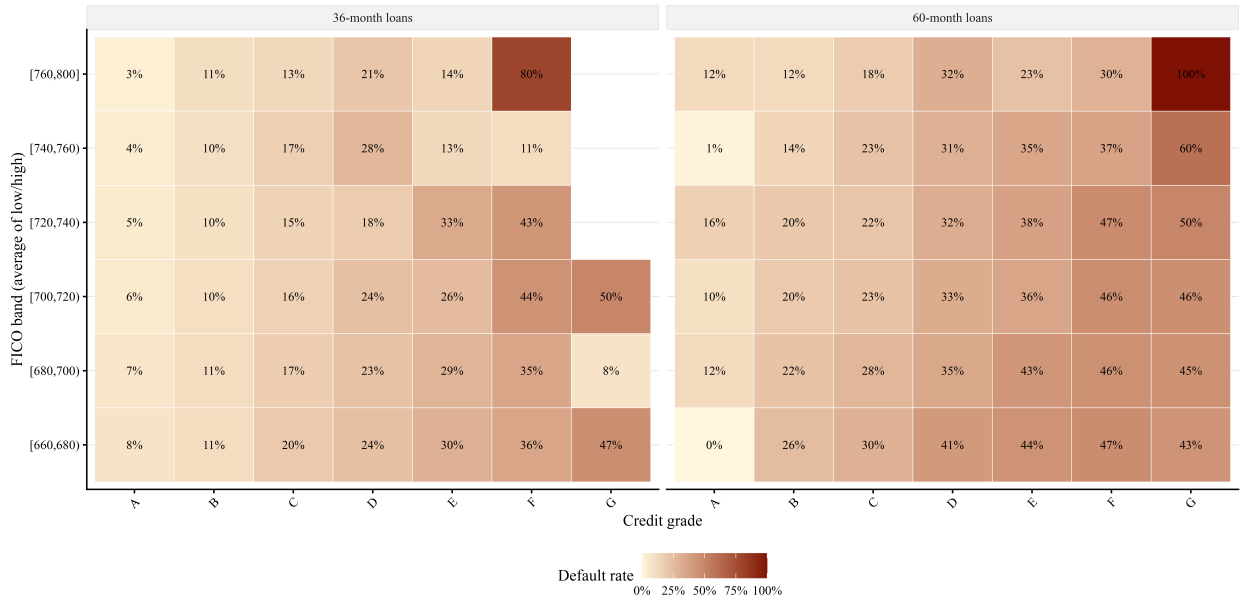

Figure 4: *Default-rate heatmaps by credit grade and FICO band, shown for 36- and 60-month loans.*

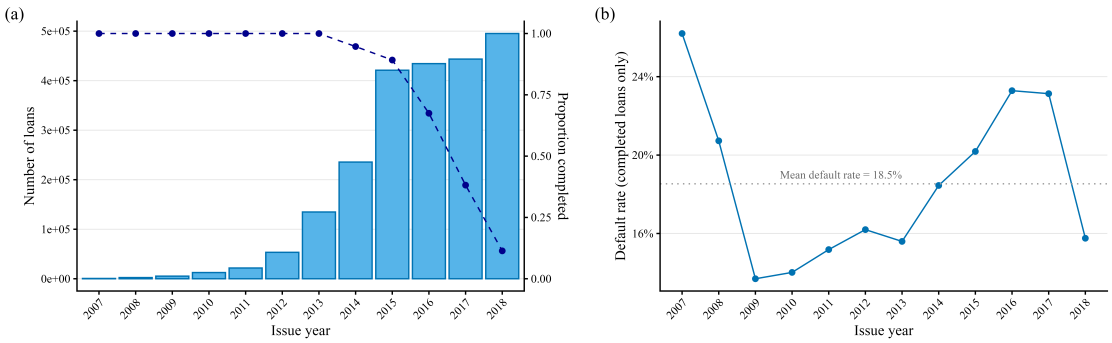

Figure 5: *Loan issuance volume and performance trends (2007-2018)*

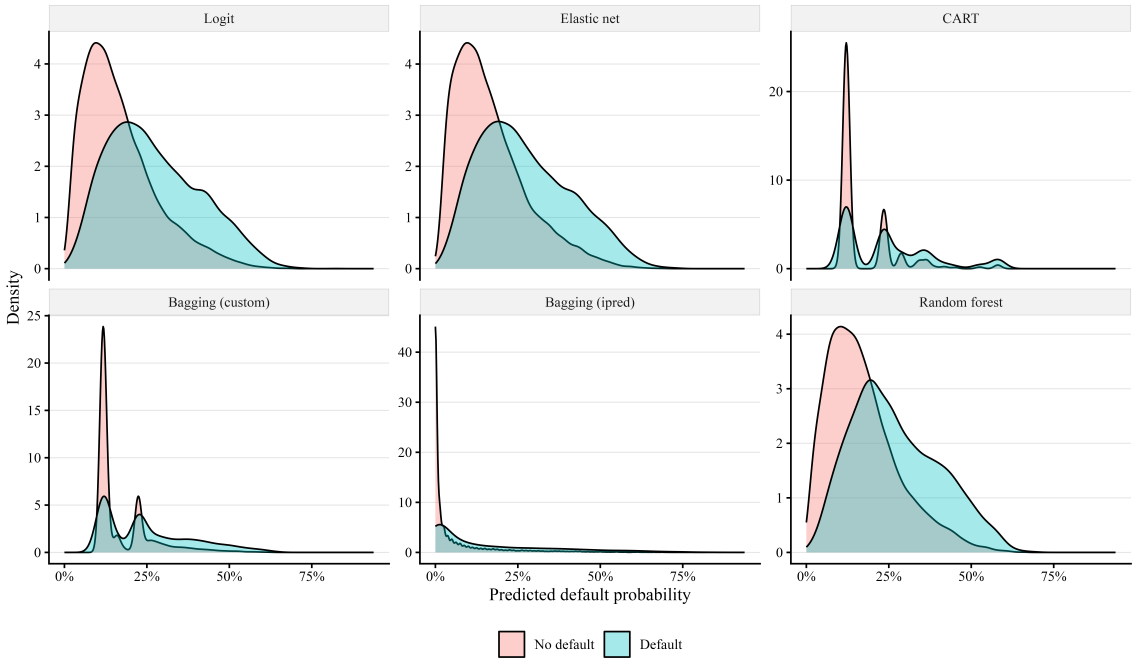

Figure 6: *Predicted default-probability distributions by true outcome across models*

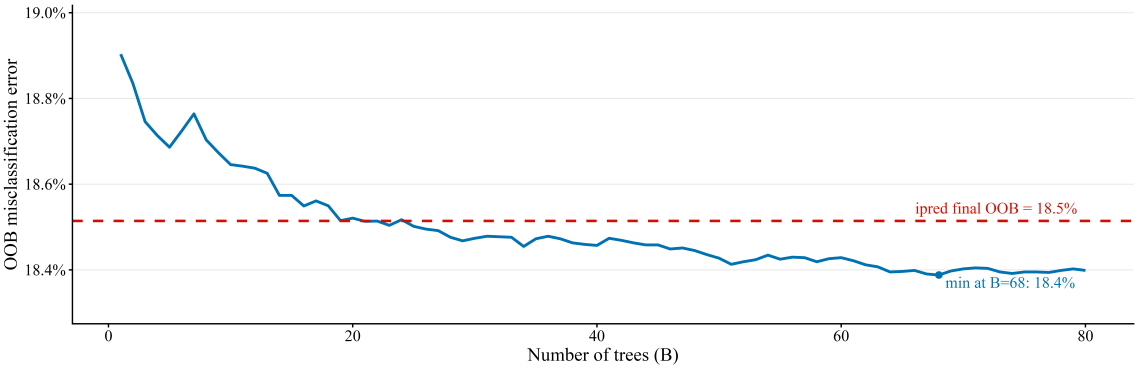

Figure 7: *Custom bagging out-of-bag misclassification error versus number of trees*

B Tables

Table 4: *Data preparation audit trail*

Action	Rows	Rows removed	Cols	Cols dropped	Cols added
Load raw data	2,260,701		151		
Create issue year + status	2,260,668	33	154	0	3
Cohort filter (2012–2015)	786,818	1,473,850	156	0	2
Stratified sample by (year × default)	119,995	666,823	156	0	0
Drop ID/text/leakage vars	119,995	0	112	44	0
Feature engineering + factor cleanup	119,995	0	113	0	1
Final modelling table	119,995	0	32	81	0

C Full R Implementation

Below is the complete R code used in this study. It imports the raw LendingClub data, constructs the origination-only modelling cohort (including cleaning the loan-status outcome, restricting to 2012–2015 completed loans to mitigate right-censoring, dropping identifiers and post-origination leakage fields, and applying a stratified downsample for computational tractability), and builds the final design matrices used for estimation. The code then fits and evaluates the full set of classifiers reported in the paper (logit, elastic net, CART, bagging, and random forest), reproducing all performance tables and screening-focused figures (ROC, precision–recall, cumulative gains, and predicted-risk distributions), and implements the double machine learning extensions for maturity and high-interest treatments alongside naive benchmarks. The script is organised in clearly labelled blocks that mirror the paper's workflow (data preparation, descriptives, modelling, diagnostics, and DML), so results can be replicated or extended with minimal modification.

```r
## ========================================================================
## Risk at First Sight - ML Final Assignment
## ------------------------------------------------------------------------
## Student Number: 756486 - Dec 2025
## ========================================================================

## ========================================================================
## 0. Set Up
## ========================================================================

## Packages used
## stargazer : exporting LaTeX tables
## readr     : fast CSV import (read_csv)
## dplyr     : data wrangling (mutate/filter/summarise)
## tidyr     : reshaping (pivot_longer/pivot_wider/unnest)
## stringr   : string handling (str_detect/str_sub/str_to_upper)
## forcats   : factor utilities (lump levels, NA level)
## tibble    : tidy tables (tibble)
## ggplot2   : plotting
## patchwork : combine ggplots (plot1 / plot2)
## scales    : axis formatting (percent_format)
## glmnet    : elastic net / ridge / lasso (cv.glmnet)
## pROC      : ROC/AUC + DeLong tests (roc, auc, ci.auc, roc.test)
## rpart     : CART trees
## ipred     : bagging benchmark (ipred::bagging)
## ranger    : random forest (probability=TRUE + OOB error)
```

```r

pkgs <- c(                                                  # list of packages required for the full analysis
  "readr","dplyr","tidyr","stringr","forcats","tibble",
  "ggplot2","patchwork","scales",
  "glmnet","pROC","rpart","ipred","ranger"
)

to_install <- pkgs[!pkgs %in% rownames(installed.packages())]         # check which required packages are not yet installed
if (length(to_install) > 0) install.packages(to_install)              # install missing packages so the script runs on a clean machine

invisible(lapply(pkgs, library, character.only = TRUE))               # load all packages quietly for use throughout the script

base_seed <- 1363                                                     # set the master seed for reproducible results across the project
set.seed(base_seed)                                                   # fix the random number generator state using the master seed

theme_academic <- function(base_size = 11, base_family = "Times New Roman") {  # define a consistent ggplot theme for all figures
  theme_classic(base_size = base_size, base_family = base_family) +            # use a clean classic theme as the base style
    theme(
      plot.title      = element_text(face = "bold", size = base_size + 1, hjust = 0),    # left-aligned bold title
      plot.subtitle   = element_text(size = base_size, hjust = 0),                       # left-aligned subtitle
      plot.caption    = element_text(size = base_size - 1, hjust = 0, color = "grey40"),# small caption
      axis.title      = element_text(face = "plain"),                                    # standard axis titles
      axis.text       = element_text(color = "black"),                                   # readable axis labels
      axis.ticks      = element_line(color = "black"),                                   # visible axis ticks
      panel.grid.major.y = element_line(color = "grey90", linewidth = 0.3),             # subtle horizontal grid for readability
      panel.grid.major.x = element_blank(),                                              # remove vertical grid lines
      panel.grid.minor   = element_blank(),                                              # remove minor grid lines
      legend.position = "bottom",                                                        # place legend under the plot
      legend.title    = element_text(face = "plain"),                                    # keep legend title simple
      legend.key      = element_blank(),                                                 # remove legend key background
      strip.background = element_rect(fill = "grey95", color = "grey80", linewidth = 0.3), # light facet strip background
      strip.text       = element_text(face = "plain"),                                   # simple facet labels
      plot.margin      = margin(10, 10, 10, 10)                                          # add consistent outer margins
    )
}

theme_set(theme_academic())                                           # apply the academic theme globally to all ggplot figures
```

```r

## ====================================================================
## 1. Load full Kaggle LendingClub file
## ====================================================================

data_dir <- "data"                                                  # folder name where the raw Kaggle file is stored
if (!dir.exists(data_dir)) dir.create(data_dir)                     # create the folder if it does not already exist

lc_file <- file.path(data_dir, "accepted_2007_to_2018Q4.csv")       # full path to the expected LendingClub CSV file

lc_raw <- readr::read_csv(lc_file, show_col_types = FALSE)          # read the full dataset and suppress column type printing

cat("\nLoaded LendingClub data\n")                                  # print a small header so logs are easy to scan
cat("Rows   :", nrow(lc_raw), "\n")                                 # print number of rows in the raw dataset
cat("Columns:", ncol(lc_raw), "\n\n")                               # print number of columns in the raw dataset

# cat("Top 20 distinct loan_status values:\n")                       # print a header for the loan status frequency table
# print(head(sort(table(lc_raw$loan_status), decreasing = TRUE), 20)) # show the most common loan status labels for auditing

## ====================================================================
## 2. Derive issue year and coarse outcme categories
## ====================================================================

status_completed <- c(                                              # loan status labels treated as completed outcomes
  "Fully Paid",
  "Charged Off",
  "Default",
  "Does not meet the credit policy. Status:Fully Paid",
  "Does not meet the credit policy. Status:Charged Off"
)

status_in_progress <- c(                                            # loan status labels treated as still active or delinquent
  "Current",
  "In Grace Period",
  "Late (16-30 days)",
  "Late (31-120 days)"
)

lc_year <- lc_raw %>%                                               # start from the raw LendingClub file and create key cohort variables
  mutate(
    issue_year = str_sub(issue_d, -4L),                             # extract the calendar year from the issue date string
    issue_year = na_if(issue_year, ""),                             # convert empty year strings to missing values
    status_group = case_when(                                       # map detailed loan_status into completed, in_progress, or other
      loan_status %in% status_completed   ~ "completed",
      loan_status %in% status_in_progress ~ "in_progress",
      TRUE                                ~ "other"
```

```r
    ),
    default_prelim = case_when(                                  # set a preliminary default indicator for completed outcomes only
      loan_status %in% c(
        "Charged Off",
        "Default",
        "Does not meet the credit policy. Status: Charged Off",
        "Does not meet the credit policy. Status:Charged Off"
      ) ~ 1L,                                                    # default equals one for charge-offs and defaults
      loan_status %in% c(
        "Fully Paid",
        "Does not meet the credit policy. Status:Fully Paid"
      ) ~ 0L,                                                    # default equals zero for fully paid loans
      TRUE ~ NA_integer_                                         # keep missing for non-completed statuses
    )
  ) %>%
  filter(!is.na(issue_year))                                     # drop rows with missing or malformed issue dates

## ====================================================================
## 3. Year-by-year summary: volume, censoring, default rates
## --------------------------------------------------------------------
## I have done this part and the following to see what years to keep
## ====================================================================

year_summary <- lc_year %>%                                      # start from the dataset with issue_year and status_group
  group_by(issue_year) %>%                                       # compute summaries separately for each issuance year
  summarise(
    n_total     = n(),                                           # total number of loans issued in that year
    n_completed = sum(status_group == "completed"),              # number of loans with completed outcomes
    n_in_prog   = sum(status_group == "in_progress"),            # number of loans still active or delinquent
    n_other     = sum(status_group == "other"),                  # number of loans with other or ambiguous statuses
    prop_completed = n_completed / n_total,                      # fraction of loans with completed outcomes
    prop_in_prog   = n_in_prog   / n_total,                      # fraction of loans still in progress
    default_rate_completed = mean(                               # default rate among completed loans only
      default_prelim[status_group == "completed"],               # restrict to completed outcomes for comparability
      na.rm = TRUE                                               # ignore missing values from non-completed statuses
    ),
    .groups = "drop"                                             # return an ungrouped data frame after summarising
  ) %>%
  arrange(issue_year)                                            # order rows chronologically by issuance year


## ====================================================================
## 4. Visual inspection: volume & default rate by year
```

```r
## ====================================================================

year_summary <- year_summary %>%                                    # start from the year-level summary table
  mutate(issue_year = factor(issue_year, levels = issue_year))      # convert issue_year to an ordered factor for a clean x-axis

mean_default_rate <- mean(year_summary$default_rate_completed, na.rm = TRUE)  # compute the average completed-loan default rate across years
x_label_year <- year_summary$issue_year[ceiling(nrow(year_summary) / 2)]      # choose a mid-sample year for placing the reference-line label

## --------------------------------------------------------------------
## 4.1 Total loans per year + proportion completed (dual axis)
## --------------------------------------------------------------------

plot_year_counts <- year_summary %>%                                # use the year-level table to plot annual volumes and completion share
  ggplot(aes(x = issue_year)) +                                     # set issue year on the x-axis
  geom_col(                                                         # draw bars for total number of loans per year
    aes(y = n_total),
    fill = "#56B4E9", color = "#0072B2"
  ) +
  geom_line(                                                        # overlay the completed share scaled to the count axis
    aes(y = prop_completed * max(n_total), group = 1),
    linetype = "dashed",
    color = "#00008B"
  ) +
  geom_point(                                                       # add points to highlight each yearly completed share
    aes(y = prop_completed * max(n_total)),
    size = 1.7,
    color = "#00008B"
  ) +
  scale_y_continuous(                                               # define the primary count axis and a secondary proportion axis
    name = "Number of loans",
    sec.axis = sec_axis(
      ~ . / max(year_summary$n_total),                              # rescale back to proportions using the maximum annual volume
      name = "Proportion completed"
    )
  ) +
  labs(
    x = "Issue year"                                                # label the x-axis
  ) +
  theme(
    axis.text.x = element_text(angle = 45, hjust = 1)               # rotate year labels for readability
  )

## --------------------------------------------------------------------
## 4.2 Default rate among completed loans by year
## --------------------------------------------------------------------

plot_year_default <- year_summary %>%                               # use the year-level table to plot default rates among completed loans
```

```r
  ggplot(aes(x = issue_year, y = default_rate_completed, group = 1)) +# connect points in
  ↪  chronological order
  geom_line(color = "#0072B2") +                                      # draw the default-rate
  ↪  trend line
  geom_point(size = 1.7, color = "#0072B2") +                         # add points at each year
  ↪  for visibility
  geom_hline(                                                         # add a reference line at
  ↪  the mean completed-loan default rate
    yintercept = mean_default_rate,
    linetype   = "dotted",
    color      = "#999999"
  ) +
  annotate(                                                           # label the reference line
  ↪  directly on the plot
    "text",
    x      = x_label_year,
    y      = mean_default_rate,
    label  = paste0(
      "Mean default rate = ",
      scales::percent(mean_default_rate, accuracy = 0.1)
    ),
    vjust  = -0.7,
    hjust  = 0.7,
    size   = 3,
    color  = "#666666"
  ) +
  scale_y_continuous(                                                 # format the y-axis as
  ↪  percentages
    labels = scales::percent_format(accuracy = 1),
    name   = "Default rate (completed loans only)"
  ) +
  labs(
    x = "Issue year"                                                  # label the x-axis
  ) +
  theme(
    axis.text.x = element_text(angle = 45, hjust = 1)                 # rotate year labels for
    ↪  readability
  )

## ------------------------------------------------------------------------
## 4.3 Combine as horizontal subplots (a) and (b)
## ------------------------------------------------------------------------

plot_year_panel <- plot_year_counts + plot_year_default +             # combine the two plots into
↪  a single panel
  patchwork::plot_layout(ncol = 2) +                                  # arrange the plots
  ↪  side-by-side
  patchwork::plot_annotation(                                         # add panel tags for figure
  ↪  referencing in the paper
    tag_levels = "a",
    tag_prefix = "(",
    tag_suffix = ")"
  )

print(plot_year_panel)                                                # display the combined panel
↪  in the plotting device

ggplot2::ggsave(                                                      # save the combined panel to
↪  disk at high resolution
  filename = "plot_year_panel.png",
  plot     = plot_year_panel,
```

```r
  width    = 12, height = 4, units = "in",
  dpi      = 1000
)

## =====================================================================
## 5. Cohort selection: 2012-2015, completed loans, individual apps
## =====================================================================

years_keep <- c("2012", "2013", "2014", "2015")                    # restrict to years with large volumes and moderate censoring

cat("\napplication_type by year (for 2012-2015):\n")               # print application_type counts to confirm the prevalence of joint loans
print(
  lc_year %>%
    dplyr::filter(issue_year %in% years_keep) %>%                  # keep only the candidate cohort years
    dplyr::count(issue_year, application_type) %>%                 # tabulate application types by year
    dplyr::arrange(issue_year, dplyr::desc(n))                     # sort within each year by frequency
)

lc_cohort <- lc_year %>%                                           # build the main modeling cohort from the cleaned year-level dataset
  dplyr::filter(
    issue_year %in% years_keep,                                    # keep loans issued in the target years
    loan_status %in% status_completed,                             # keep only completed outcomes for comparable default labeling
    is.na(application_type) |                                      # keep missing application types rather than dropping observations
      stringr::str_to_upper(application_type) != "JOINT"           # exclude clearly joint applications based on case-insensitive matching
  ) %>%
  dplyr::mutate(
    default = default_prelim,                                      # set the final binary default outcome used in modeling
    term_60 = ifelse(stringr::str_detect(term, "60"), 1L, 0L)      # create a 60-month term indicator from the term string
  ) %>%
  dplyr::filter(
    !is.na(default),                                               # require a valid default label
    !is.na(loan_amnt),                                             # require loan amount at origination
    !is.na(int_rate),                                              # require interest rate at origination
    !is.na(annual_inc),                                            # require annual income at origination
    !is.na(dti)                                                    # require debt-to-income at origination
  )

# cat("\nCohort (2012-2015, completed, non-joint) summary:\n")      # print a cohort summary for manual verification
# cat("Rows in lc_cohort:", nrow(lc_cohort), "\n")                  # print number of observations in the final cohort
# cat("Default distribution:\n")                                   # print default counts and shares for a quick sanity check
# print(table(lc_cohort$default))                                  # show default counts
```

```r
# print(prop.table(table(lc_cohort$default)))                    # show default proportions
# cat("\nCohort by year and default (counts):\n")                # show year-by-default counts for stratified sampling checks
# print(
#   lc_cohort %>%
#     dplyr::count(issue_year, default) %>%                      # count observations by year and default
#     tidyr::pivot_wider(
#       names_from   = default,
#       values_from  = n,
#       values_fill  = 0,
#       names_prefix = "default_"
#     )
# )

## ====================================================================
## 6. Stratified sample (by issue_year × default)
## ====================================================================

target_n <- 120000                                                 # target working sample size for computational tractability

N_cohort    <- nrow(lc_cohort)                                     # total number of observations in the full modeling cohort
sample_frac <- target_n / N_cohort                                 # sampling fraction to apply within each year by default stratum

# cat("\nTotal cohort size:", N_cohort, "\n")                      # print the cohort size for auditing
# cat("Target N:", target_n, "\n")                                 # print the intended sample size
# cat("Sampling fraction (per issue_year by default stratum):",    # print the per-cell sampling fraction used in stratified sampling
#     round(sample_frac, 4), "\n")

set.seed(base_seed + 1)                                            # fix the random seed for a reproducible stratified draw

lc_sample <- lc_cohort %>%                                         # draw a working sample while preserving cohort composition
  dplyr::group_by(issue_year, default) %>%                         # stratify by issuance year and default outcome
  dplyr::slice_sample(prop = sample_frac) %>%                      # sample the same fraction from each stratum
  dplyr::ungroup()                                                 # return to an ungrouped data frame for downstream steps

# cat("\nActual lc_sample size:\n")                                # print the realized sample size after stratification
# print(nrow(lc_sample))                                           # show number of rows in the working sample
#
# cat("\nDefault distribution in lc_sample:\n")                    # verify that default prevalence is close to the cohort rate
# print(table(lc_sample$default))                                  # show default counts
# print(prop.table(table(lc_sample$default)))                      # show default proportions
#
# cat("\nYear distribution in lc_sample:\n")                       # verify that year composition is preserved
# print(table(lc_sample$issue_year))                               # show year counts
# print(prop.table(table(lc_sample$issue_year)))                   # show year proportions
```

```r

## ====================================================================
## 7. From lc_sample → drop ID/leakage → concept-driven lc_model_base
## ====================================================================

## --------------------------------------------------------------------
## 7.1 Flag ID/text and leakage variables using full lc_raw
## --------------------------------------------------------------------

all_vars <- names(lc_raw)                                   # store all raw variable names from the full Kaggle file

id_text_vars <- c(                                          # define obvious identifiers and free-text fields to exclude from modeling
  "id",
  "member_id",
  "url",
  "desc",
  "title",
  "emp_title"
)

quasi_id_vars <- c("zip_code")                              # define quasi-identifiers to exclude to reduce re-identification risk

leakage_patterns <- c(                                      # define name patterns for variables that reflect post-origination information
  "^total_pymnt",
  "^total_pymnt_inv",
  "^total_rec_",
  "^recoveries",
  "^collection_recovery_fee",
  "^last_pymnt_",
  "^next_pymnt_",
  "^last_credit_pull_d",
  "^settlement_",
  "^hardship_",
  "^debt_settlement_"
)

is_id_text <- all_vars %in% c(id_text_vars, quasi_id_vars)  # flag variables that are identifiers, URLs, or free-text fields

is_leakage <- vapply(                                       # flag variables that match any leakage pattern in their name
  X   = all_vars,
  FUN = function(v) any(stringr::str_detect(v, leakage_patterns)),
  FUN.VALUE = logical(1L)
)

var_flags <- tibble::tibble(                                # create a table summarizing which raw variables are flagged for exclusion
  variable   = all_vars,
  id_or_text = is_id_text,
  leakage    = is_leakage
)

# cat("\nFlag summary on full lc_raw (ID/text vs leakage):\n")   # print a two-way table of ID/text flags and leakage flags
# print(table(
#   id_or_text = var_flags$id_or_text,
```

```r
#   leakage    = var_flags$leakage
# ))
#
# cat("\nVariables flagged as ID, text, or quasi ID:\n")               # list all variables flagged as identifiers or free-text fields
# print(var_flags$variable[var_flags$id_or_text])
#
# cat("\nVariables flagged as potential leakage:\n")                   # list all variables flagged as potential post-origination leakage
# print(var_flags$variable[var_flags$leakage])

## ----------------------------------------------------------------------
## 7.2 Build lc_model_base from lc_sample (drop ID / leakage / labels)
## ----------------------------------------------------------------------

id_text_vars_full <- intersect(                                          # keep only ID and text variables that actually exist in the working sample
  c(id_text_vars, quasi_id_vars),
  names(lc_sample)
)

leakage_vars_pattern <- intersect(                                       # keep only pattern-flagged leakage variables that exist in the working sample
  var_flags$variable[var_flags$leakage],
  names(lc_sample)
)

extra_leakage_vars <- intersect(                                         # add known post-origination variables that may not match the patterns
  c("out_prncp", "out_prncp_inv",
    "last_fico_range_low", "last_fico_range_high"),
  names(lc_sample)
)

outcome_label_vars <- intersect(                                         # exclude outcome labels and helper fields used during cleaning
  c("loan_status", "default_prelim", "status_group"),
  names(lc_sample)
)

drop_vars <- unique(c(                                                   # combine all exclusion sets into a single list of variables to drop
  id_text_vars_full,
  leakage_vars_pattern,
  extra_leakage_vars,
  outcome_label_vars
))

cat("\nNumber of variables in lc_sample:", ncol(lc_sample), "\n")      # report the number of columns available before dropping anything
cat("Number of variables to drop (ID/text/leakage/labels):", length(drop_vars), "\n")  # report the number of excluded variables

cat("\nDropping these ID/text/quasi ID vars:\n")                        # print the ID and text variables being removed
print(id_text_vars_full)

cat("\nDropping these leakage vars (pattern-based plus explicit extras):\n")  # print the leakage variables being removed
print(sort(unique(c(leakage_vars_pattern, extra_leakage_vars))))

```

```r
cat("\nDropping these outcome/label/helper vars:\n")                 # print the label variables being removed
print(outcome_label_vars)

lc_model_base <- lc_sample %>%                                       # create the modeling table after removing identifiers and leakage fields
  dplyr::select(-dplyr::all_of(drop_vars))                           # drop excluded variables to retain origination-time predictors only

# cat(                                                               # print the final number of columns after exclusions as a quick check
#   "\nNumber of variables in lc_model_base",
#   "(incl. default, term_60, issue_year):",
#   ncol(lc_model_base), "\n"
# )


## ====================================================================
## 8. Concept-driven feature set: lc_model_orig
## --------------------------------------------------------------------
## I needed to further reduce the dimension for computation sake
## ====================================================================

## --------------------------------------------------------------------
## 8.1 Feature shortlists by economic block
## --------------------------------------------------------------------

feat_terms_pricing_num <- c(                                         # numeric contract terms capturing loan size and price of credit
  "loan_amnt",
  "funded_amnt",
  "funded_amnt_inv",
  "int_rate",
  "installment"
)

feat_terms_pricing_cat <- c(                                         # categorical contract terms capturing platform risk buckets and listing status
  "term",                                                            # keep raw term; term_60 is the binary indicator built earlier
  "grade",
  "sub_grade",
  "initial_list_status"
)

feat_income_emp_num <- c(                                            # numeric borrower affordability measures
  "annual_inc",
  "dti"
)

feat_income_emp_cat <- c(                                            # categorical borrower employment and verification information
  "emp_length",
  "home_ownership",
  "verification_status"
)

feat_credit_num <- c(                                                # numeric credit history and revolving debt indicators
  "fico_range_low",
```

```r
  "fico_range_high",
  "open_acc",
  "total_acc",
  "revol_bal",
  "revol_util",
  "inq_last_6mths",
  "delinq_2yrs",
  "mths_since_last_delinq",
  "mths_since_last_record",
  "mths_since_recent_inq",
  "pub_rec",
  "pub_rec_bankruptcies",
  "mort_acc",
  "collections_12_mths_ex_med",
  "num_rev_tl_bal_gt_0"
)

feat_credit_cat <- character(0)                                    # no additional categorical credit-history variables in this shortlist

feat_purpose_region_cat <- c(                                      # categorical purpose and application metadata
  "purpose",
  "application_type"                                               # mostly individual and may become near-constant after filtering
)

feat_numeric_cand <- c(                                            # stack numeric candidates from the economic blocks
  feat_terms_pricing_num,
  feat_income_emp_num,
  feat_credit_num
)

feat_categorical_cand <- c(                                        # stack categorical candidates from the economic blocks
  feat_terms_pricing_cat,
  feat_income_emp_cat,
  feat_credit_cat,
  feat_purpose_region_cat
)

feat_all_cand <- c(feat_numeric_cand, feat_categorical_cand)        # combine all shortlisted variables into one candidate set

feat_present <- intersect(feat_all_cand, names(lc_model_base))      # keep only shortlisted features that actually exist in the data

# cat("\nNumber of candidate features in shortlist (present in data):",# print the number of shortlisted features available in the dataset
#     length(feat_present), "\n")
# cat("Candidate numeric vars present:\n")                          # print the numeric shortlist that exists in the dataset
# print(intersect(feat_numeric_cand, feat_present))
# cat("\nCandidate categorical vars present:\n")                    # print the categorical shortlist that exists in the dataset
# print(intersect(feat_categorical_cand, feat_present))

## ---------------------------------------------------------------------
## 8.2 Create FICO average and refresh shortlists
## ---------------------------------------------------------------------
```

```r

lc_model_features <- lc_model_base                                    # create a working copy before engineering additional features

if (all(c("fico_range_low", "fico_range_high") %in% names(lc_model_features))) {  # proceed only if both FICO bounds are available
  lc_model_features <- lc_model_features %>%                          # update the working table by adding a single FICO summary measure
    dplyr::mutate(
      fico_avg = 0.5 * (fico_range_low + fico_range_high)             # compute the midpoint of the reported FICO band
    )
  cat("\nCreated fico_avg from fico_range_low and fico_range_high.\n")# log feature creation for reproducibility

  feat_numeric_cand <- setdiff(                                       # remove the raw FICO bounds from the numeric candidate list
    feat_numeric_cand,
    c("fico_range_low", "fico_range_high")
  )
  feat_numeric_cand <- union(feat_numeric_cand, "fico_avg")           # add fico_avg to the numeric candidate list
}

feat_numeric_present <- intersect(feat_numeric_cand, names(lc_model_features))      # keep only numeric candidates present after the FICO update
feat_categorical_present <- intersect(feat_categorical_cand, names(lc_model_features))# keep only categorical candidates present after the FICO update

# cat("\nNumeric candidates present after FICO adjustment:\n")         # list numeric candidates that remain after the FICO transformation
# print(feat_numeric_present)
# cat("\nCategorical candidates present:\n")                           # list categorical candidates that remain after the FICO transformation
# print(feat_categorical_present)

## ----------------------------------------------------------------------
## 8.3 Tidy categorical variables (factor + lump rare purpose levels)
## ----------------------------------------------------------------------

purpose_min_count <- round(0.02 * nrow(lc_model_features))            # set the minimum count for keeping a named purpose level

lc_model_features <- lc_model_features %>%                            # coerce shortlisted categorical variables to factor for modeling
  dplyr::mutate(
    dplyr::across(
      dplyr::all_of(feat_categorical_present),
      ~ as.factor(.)
    )
  )

if ("purpose" %in% feat_categorical_present) {                        # lump rare purpose levels only if the purpose variable is present
  lc_model_features <- lc_model_features %>%                          # collapse low-frequency purpose categories into an other group
    dplyr::mutate(
      purpose = forcats::fct_lump_min(
        purpose,
        min         = purpose_min_count,                              # enforce the minimum cell size threshold
```

```
        other_level = "other"                                    # name for the pooled
        ↪  rare-category level
      )
    )
}

## -----------------------------------------------------------------
## 8.4 Drop high-missing or near-constant features within shortlist
## -----------------------------------------------------------------

vars_candidate <- intersect(                                     # restrict predictors to
↪  shortlisted variables that exist in the dataset
  c(feat_numeric_present, feat_categorical_present),
  names(lc_model_features)
)

tmp_candidate <- lc_model_features %>%                           # create a temporary
↪  candidate-only table for diagnostics
  dplyr::select(dplyr::all_of(vars_candidate))

na_summary_short <- tmp_candidate %>%                            # compute missingness rates
↪  for each candidate predictor
  dplyr::summarise(dplyr::across(
    dplyr::everything(),
    ~ mean(is.na(.))
  )) %>%
  tidyr::pivot_longer(
    dplyr::everything(),
    names_to  = "variable",
    values_to = "prop_na"
  )

high_na_vars <- na_summary_short %>%                             # identify predictors with
↪  high missingness to exclude from modeling
  dplyr::filter(prop_na > 0.40) %>%                              # drop predictors with more
  ↪  than 40 percent missing values
  dplyr::pull(variable)

# cat("\nWithin shortlist: variables with more than 40 percent missingness:\n")  # print variables
↪  dropped due to high missingness
# print(high_na_vars)

num_sd_short <- tmp_candidate %>%                                # compute standard deviations
↪  for numeric predictors to detect near-constants
  dplyr::select(where(is.numeric)) %>%
  dplyr::summarise(dplyr::across(
    dplyr::everything(),
    sd,
    na.rm = TRUE
  )) %>%
  tidyr::pivot_longer(
    dplyr::everything(),
    names_to  = "variable",
    values_to = "sd"
  )

near_const_num <- num_sd_short %>%                               # identify numeric predictors
↪  with negligible variation
  dplyr::filter(sd < 1e-6) %>%                                   # treat extremely small
  ↪  standard deviation as near-constant
  dplyr::pull(variable)
```

```r

# cat("\nWithin shortlist: near-constant numeric vars:\n")            # print numeric variables dropped due to near-constant values
# print(near_const_num)

fac_maxprop_short <- tmp_candidate %>%                                # compute the dominant category share for each factor predictor
  dplyr::select(where(is.factor)) %>%
  dplyr::summarise(
    dplyr::across(
      dplyr::everything(),
      ~ {
        tab <- table(., useNA = "no")                                 # tabulate observed factor levels excluding missing values
        max(prop.table(tab))                                          # compute the share of the most frequent level
      }
    )
  ) %>%
  tidyr::pivot_longer(
    dplyr::everything(),
    names_to  = "variable",
    values_to = "max_prop"
  )

near_const_fac <- fac_maxprop_short %>%                               # identify factor predictors dominated by a single category
  dplyr::filter(max_prop > 0.99) %>%                                  # drop predictors with more than 99 percent mass in one level
  dplyr::pull(variable)

# cat("\nWithin shortlist: near-constant factor vars:\n")            # print factor variables dropped due to near-constant levels
# print(near_const_fac)

vars_drop_short <- unique(c(                                          # combine all predictors flagged for removal within the shortlist
  high_na_vars,
  near_const_num,
  near_const_fac
))

vars_predictors_final <- setdiff(vars_candidate, vars_drop_short)     # define the final predictor set after removing problematic variables

# cat("\nFinal predictor set size:\n")                                # print the final number of predictors kept for modeling
# print(length(vars_predictors_final))
# print(vars_predictors_final)

lc_model_orig <- lc_model_features %>%                                # construct the final modeling dataset with outcome and predictors
  dplyr::select(
    default,                                                          # include the binary default outcome
    term_60,                                                          # include the 60-month term indicator for later analyses
    issue_year,                                                       # include issuance year for stratification and controls
    dplyr::all_of(vars_predictors_final)                              # include the final set of origination-time predictors
```

```r
  )

# cat("\nDimensions of lc_model_orig:\n")                                  # print dataset dimensions for a final sanity check
# print(dim(lc_model_orig))
#
# cat("\nFirst 25 variable names in lc_model_orig:\n")                    # print the first variable names to confirm ordering and content
# print(names(lc_model_orig)[1:min(25, ncol(lc_model_orig))])

## ====================================================================
## 9. Descriptives on lc_model_orig
## ====================================================================

## --------------------------------------------------------------------
## 9.1 Data cleaning and preparation audit trail (row and column counts)
## --------------------------------------------------------------------

data_audit <- tibble::tibble(                                           # collect a step-by-step audit trail of the data pipeline
  step = 1:7,                                                           # numeric step index used for ordering in the table
  action = c(                                                           # short description of what happens at each pipeline step
    "Load raw data",
    "Create issue year + status (drop missing year)",
    "Cohort filter (2012-2015, completed, non-JOINT, key vars)",
    "Stratified sample (~120k) by (year × default)",
    "Drop ID/text/leakage vars",
    "Feature engineering + factor cleanup",
    "Final modelling table"
  ),
  n_rows = c(                                                           # row counts at each step to track sample reduction
    nrow(lc_raw),
    nrow(lc_year),
    nrow(lc_cohort),
    nrow(lc_sample),
    nrow(lc_model_base),
    nrow(lc_model_features),
    nrow(lc_model_orig)
  ),
  n_cols = c(                                                           # column counts at each step to track feature reduction
    ncol(lc_raw),
    ncol(lc_year),
    ncol(lc_cohort),
    ncol(lc_sample),
    ncol(lc_model_base),
    ncol(lc_model_features),
    ncol(lc_model_orig)
  ),
  notes = c(                                                            # short notes that clarify the main reason for each step
    "Loaded full CSV from Kaggle",
    "Define default/status; keep non-missing issue_year",
    "Keep years 2012-2015; completed only; exclude JOINT; require non-missing key vars; add term_60",
    "Downsample for compute tractability (stratified)",
    "Remove identifiers + leakage + helper/outcome label vars",
```

```
    "Create fico_avg; convert to factors; lump rare purpose; handle high-missing/near-constant vars",
    "Keep: default, term_60, issue_year + final predictor shortlist"
  )
) %>%
  dplyr::mutate(                                                    # add derived columns that quantify how much was removed at each step
    rows_removed = dplyr::if_else(                                  # compute rows removed relative to the previous step
      is.na(dplyr::lag(n_rows)), NA_integer_,
      pmax(0L, dplyr::lag(n_rows) - n_rows)
    ),
    cols_dropped = dplyr::if_else(                                  # compute columns dropped relative to the previous step
      is.na(dplyr::lag(n_cols)), NA_integer_,
      pmax(0L, dplyr::lag(n_cols) - n_cols)
    ),
    cols_added = dplyr::if_else(                                    # compute columns added relative to the previous step
      is.na(dplyr::lag(n_cols)), NA_integer_,
      pmax(0L, n_cols - dplyr::lag(n_cols))
    )
  )

data_audit_keep <- data_audit %>%                                   # keep only the columns you want to display in the audit table
  dplyr::select(step, action, n_rows, rows_removed, n_cols, cols_dropped, cols_added, notes)

data_audit_pretty <- data_audit_keep %>%                            # create a console-friendly version with comma formatting
  dplyr::mutate(
    dplyr::across(
      c(n_rows, rows_removed, n_cols, cols_dropped, cols_added),
      ~ ifelse(is.na(.), NA, scales::comma(.))
    )
  )

print(data_audit_pretty)                                            # print the audit table for a quick visual check in the console

data_audit_tex <- data_audit_keep %>%                               # convert the audit table into a LaTeX-friendly data frame
  dplyr::mutate(
    dplyr::across(c(step, n_rows, rows_removed, n_cols, cols_dropped, cols_added), as.integer)
  ) %>%
  as.data.frame()

colnames(data_audit_tex) <- c(                                      # set display column names for the exported LaTeX table
  "Step", "Action", "Rows", "Rows removed", "Cols", "Cols dropped", "Cols added", "Notes"
)

stargazer::stargazer(                                               # export the audit trail as a LaTeX table for the paper
  data_audit_tex,
  type = "latex",
  summary = FALSE,
  rownames = FALSE,
  digits = 0,
  no.space = TRUE,
  table.placement = "!htbp",
```

```r
  title = "Data preparation audit trail",
  label = "tab:data_audit",
  out = "table_data_audit.tex"
)

## ----------------------------------------------------------------------
## 9.2 Default rate by grade and term length
## ----------------------------------------------------------------------

plot_def_by_grade_term <- lc_model_orig %>%                          # start from the final modeling table
  dplyr::group_by(grade, term_60) %>%                                # compute default rates by grade separately for each term group
  dplyr::summarise(
    def_rate = mean(default, na.rm = TRUE),                          # default rate within each grade and term cell
    n        = dplyr::n(),                                           # cell size used for point sizing in the plot
    .groups  = "drop"
  ) %>%
  dplyr::mutate(
    term_60 = factor(                                                # map the binary term indicator to readable labels
      term_60,
      levels = c(0, 1),
      labels = c("36 months", "60 months")
    )
  ) %>%
  ggplot(aes(x = grade, y = def_rate, group = term_60, colour = term_60)) +  # plot default rate by grade with a separate line per term
  geom_line(linewidth = 0.7) +                                       # connect grade points to show monotone risk gradients
  geom_point(aes(size = n), alpha = 0.65) +                          # size points by number of loans in each cell
  scale_y_continuous(labels = scales::percent_format(accuracy = 1)) +# express default rates as percentages on the y-axis
  scale_colour_manual(values = c("36 months" = "#0072B2", "60 months" = "#D55E00")) + # set consistent colors for term groups
  labs(
    x      = "LendingClub grade",
    y      = "Default rate",
    colour = "Term",
    size   = "Number of loans"
  ) +
  theme_academic() +                                                 # apply the project theme
  theme(legend.position = "bottom")                                  # place legend below the plot for readability

## ----------------------------------------------------------------------
## 9.3 Interest rate versus default
## ----------------------------------------------------------------------

int_rate_thr_75 <- stats::quantile(lc_model_orig$int_rate, 0.75, na.rm = TRUE)  # compute the seventy-fifth percentile interest-rate threshold

df_int_default <- lc_model_orig %>%                                  # compute default rates by rounded interest-rate bins
  dplyr::mutate(int_rate_round = round(int_rate, 1)) %>%             # round interest rates to create stable bins
  dplyr::group_by(int_rate_round) %>%                                # group by the interest-rate bin
```

```r
  dplyr::summarise(
    def_rate = mean(default, na.rm = TRUE),                    # default rate within each interest-rate bin
    n        = dplyr::n(),                                     # bin size used to remove very small bins
    .groups  = "drop"
  ) %>%
  dplyr::filter(n > 200)                                       # keep bins with enough observations to avoid noisy rates

y_max <- max(df_int_default$def_rate, na.rm = TRUE)            # store the maximum default rate for placing the annotation label

plot_int_default <- df_int_default %>%                         # plot default rate as a function of interest rate
  ggplot(aes(x = int_rate_round, y = def_rate)) +
  geom_line(color = "#0072B2") +                               # draw the default-rate curve across interest-rate bins
  geom_point(size = 1.2, color = "#0072B2") +                  # add points to show bin-level estimates
  geom_vline(                                                  # mark the seventy-fifth percentile threshold used for the high-interest treatment
    xintercept = int_rate_thr_75,
    linetype   = 2,
    color      = "#D55E00"
  ) +
  annotate(                                                    # label the threshold directly on the plot for interpretation
    "text",
    x     = as.numeric(int_rate_thr_75),
    y     = y_max,
    label = paste0("75th pct: ", round(int_rate_thr_75, 2), "%"),
    hjust = -0.05,
    vjust = 1.2,
    size  = 3,
    color = "#D55E00"
  ) +
  scale_y_continuous(labels = scales::percent_format(accuracy = 1)) +# express default rates as percentages
  labs(
    x = "Interest rate (%)",
    y = "Default rate"
  ) +
  theme_academic()                                             # apply the project theme

plot_def_grade_int_panel <- plot_def_by_grade_term + plot_int_default +  # combine the grade-term plot and the interest-default plot
  patchwork::plot_layout(ncol = 2) +                           # place the two plots side-by-side
  patchwork::plot_annotation(                                  # add panel tags for referencing in the paper
    tag_levels = "a",
    tag_prefix = "(",
    tag_suffix = ")"
  )

print(plot_def_grade_int_panel)                                # display the combined panel in the plotting device

ggplot2::ggsave(                                               # save the combined panel at high resolution
```

```r
  filename = "plot_def_grade_int_panel.png",
  plot     = plot_def_grade_int_panel,
  width    = 12, height = 6, units = "in",
  dpi      = 1000
)

## ----------------------------------------------------------------------
## 9.4 Risk surface: grade by FICO by term (heatmap)
## ----------------------------------------------------------------------

if (!"fico_bin" %in% names(lc_model_orig)) {                    # create a FICO bin only if it does not already exist
  lc_model_orig <- lc_model_orig %>%
    dplyr::mutate(
      fico_bin = cut(                                           # discretize fico_avg into standard bands for a readable heatmap
        fico_avg,
        breaks = c(600, 640, 660, 680, 700, 720, 740, 760, 800),
        include.lowest = TRUE,
        right = FALSE
      )
    )
}

plot_heat_grade_fico <- lc_model_orig %>%                       # compute cell-level default rates and plot them as a heatmap
  dplyr::filter(!is.na(fico_bin)) %>%                           # drop observations without a valid FICO bin
  dplyr::group_by(grade, fico_bin, term_60) %>%                 # group by grade, FICO band, and term group
  dplyr::summarise(
    def_rate = mean(default, na.rm = TRUE),                     # cell default rate
    n        = dplyr::n(),                                      # cell size kept for diagnostics if needed
    .groups  = "drop"
  ) %>%
  dplyr::mutate(
    term_60_lab = factor(                                       # build readable facet labels for term groups
      term_60,
      levels = c(0, 1),
      labels = c("36-month loans", "60-month loans")
    )
  ) %>%
  ggplot(aes(x = grade, y = fico_bin, fill = def_rate)) +       # map grade and FICO band to heatmap axes and default rate to fill
  geom_tile(color = "white") +                                  # draw heatmap tiles with white borders for separation
  geom_text(                                                    # print default rates inside tiles for quick interpretation
    aes(label = scales::percent(def_rate, accuracy = 1)),
    size = 3
  ) +
  facet_wrap(~ term_60_lab) +                                   # show separate heatmaps for 36-month and 60-month loans
  scale_fill_gradient(                                          # map default rate to a continuous color scale
    name   = "Default rate",
    labels = scales::percent_format(accuracy = 1),
    low    = "#FFF6DB",
    high   = "#7F1204"
```

```r
  ) +
  labs(
    x = "Credit grade",
    y = "FICO band (average of low and high)"
  ) +
  theme(
    axis.text.x = element_text(angle = 45, hjust = 1),               # rotate grade labels for readability
    panel.grid  = element_blank()                                    # remove grid lines to keep the heatmap clean
  )

print(plot_heat_grade_fico)                                          # display the heatmap in the plotting device

ggplot2::ggsave(                                                     # save the heatmap to disk at high resolution
  filename = "plot_heat_grade_fico.png",
  plot     = plot_heat_grade_fico,
  width    = 12, height = 6, units = "in",
  dpi      = 1000
)

## =====================================================================
## 10. ML dataset: split, impute, design matrices
## =====================================================================

## ---------------------------------------------------------------------
## 10.1 Start from lc_model_orig and enforce types
## ---------------------------------------------------------------------

ml_data <- lc_model_orig %>%                                         # start from the final modeling dataset
  dplyr::mutate(
    default    = as.integer(default),                                # ensure the outcome is stored as 0 or 1 integer
    term_60    = as.integer(term_60),                                # ensure the term indicator is stored as 0 or 1 integer
    issue_year = factor(issue_year)                                  # treat issuance year as a categorical control
  ) %>%
  dplyr::mutate(
    dplyr::across(                                                   # convert selected categorical predictors to factors if present
      .cols = intersect(
        c("term", "grade", "sub_grade", "emp_length", "home_ownership",
          "verification_status", "purpose", "application_type", "initial_list_status"),
        names(.)
      ),
      .fns  = ~ as.factor(.x)                                        # coerce to factor to enable stable one-hot encoding later
    )
  )

stopifnot(all(ml_data$default %in% c(0L, 1L)))                       # verify that the outcome contains only 0 and 1

## ---------------------------------------------------------------------
## 10.2 Stratified train/test split (70/30 within default)
## ---------------------------------------------------------------------

```

```r
train_frac <- 0.70                                                    # fraction of observations assigned to the training set
set.seed(base_seed + 3)                                               # set a seed so the split is reproducible

ml_data <- ml_data %>%                                                # add a row identifier so sampled rows can be tracked
  dplyr::mutate(row_id = dplyr::row_number())

train_ids <- ml_data %>%                                              # sample training rows within each outcome class
  dplyr::group_by(default) %>%                                        # stratify by the default outcome to preserve class balance
  dplyr::slice_sample(prop = train_frac) %>%                          # sample the training fraction from each class
  dplyr::ungroup() %>%
  dplyr::pull(row_id)                                                 # store the selected training row identifiers

ml_data_split <- ml_data %>%                                          # create a split indicator and remove the helper id
  dplyr::mutate(is_train = row_id %in% train_ids) %>%                 # flag whether each row is in the training split
  dplyr::select(-row_id)                                              # drop the helper identifier after the split is defined

train_ml <- ml_data_split %>% dplyr::filter(is_train)                 # training subset used for fitting and preprocessing
test_ml  <- ml_data_split %>% dplyr::filter(!is_train)                # test subset held out for final evaluation

stopifnot(abs(mean(train_ml$default) - mean(test_ml$default)) < 0.01)  # confirm that outcome prevalence is similar across splits

## ----------------------------------------------------------------------
## 10.3 Median and Unknown imputation (fit on train, apply to test)
## ----------------------------------------------------------------------

num_vars <- train_ml %>%                                              # identify numeric predictors using training data only
  dplyr::select(-default, -is_train) %>%                              # exclude outcome and split flag from predictor lists
  dplyr::select(where(is.numeric)) %>%
  names()

fac_vars <- train_ml %>%                                              # identify factor predictors using training data only
  dplyr::select(-default, -is_train) %>%                              # exclude outcome and split flag from predictor lists
  dplyr::select(where(is.factor)) %>%
  names()

num_impute_vals <- train_ml %>%                                       # compute training medians for numeric imputation
  dplyr::summarise(dplyr::across(dplyr::all_of(num_vars), ~ median(.x, na.rm = TRUE))) # store one median per numeric variable

train_ml_imp <- train_ml %>%                                          # apply imputation rules to the training set
  dplyr::mutate(
```

```r
    dplyr::across(dplyr::all_of(fac_vars),                                  # replace missing factor values with an explicit level
                  ~ forcats::fct_na_value_to_level(.x, level = "Unknown")),
    dplyr::across(dplyr::all_of(num_vars),                                  # replace missing numeric values using training medians
                  ~ replace(.x, is.na(.x), num_impute_vals[[dplyr::cur_column()]]))
  )

test_ml_imp <- test_ml %>%                                                  # apply the same imputation rules to the test set
  dplyr::mutate(
    dplyr::across(dplyr::all_of(fac_vars),                                  # replace missing factor values with the same explicit level
                  ~ forcats::fct_na_value_to_level(.x, level = "Unknown")),
    dplyr::across(dplyr::all_of(num_vars),                                  # replace missing numeric values using training medians
                  ~ replace(.x, is.na(.x), num_impute_vals[[dplyr::cur_column()]]))
  )

for (v in fac_vars) {                                                       # align test factor levels with training levels for stable one-hot encoding
  lvls_train <- levels(train_ml_imp[[v]])                                   # store the training levels for this factor
  x_chr      <- as.character(test_ml_imp[[v]])                              # convert test values to character for safe recoding
  x_chr[is.na(x_chr) | !(x_chr %in% lvls_train)] <- "Unknown"                # map missing and unseen test levels to Unknown
  test_ml_imp[[v]] <- factor(x_chr, levels = lvls_train)                    # rebuild the factor using the training level set
}

stopifnot(!anyNA(train_ml_imp %>% dplyr::select(-default, -is_train)))      # confirm there are no missing predictor values in the training set
stopifnot(!anyNA(test_ml_imp  %>% dplyr::select(-default, -is_train)))      # confirm there are no missing predictor values in the test set

## ----------------------------------------------------------------------
## 10.4 Design matrices (glmnet-ready one-hot encoding)
## ----------------------------------------------------------------------

mm_form <- default ~ . - is_train                                           # specify the model formula and drop the split flag

mf_train <- stats::model.frame(mm_form, data = train_ml_imp)                # build the training model frame and store factor level information
x_train  <- stats::model.matrix(mm_form, data = mf_train)                   # create the training design matrix with one-hot encoded factors

mf_test  <- stats::model.frame(                                             # build the test model frame using the training factor levels
  mm_form,
  data = test_ml_imp,
  xlev = .getXlevels(stats::terms(mf_train), mf_train)
)
x_test   <- stats::model.matrix(mm_form, data = mf_test)                    # create the test design matrix with the same columns as training

y_train <- as.numeric(train_ml_imp$default)                                 # extract training labels as numeric 0 or 1
y_test  <- as.numeric(test_ml_imp$default)                                  # extract test labels as numeric 0 or 1
```

```r

stopifnot(ncol(x_train) == ncol(x_test))                                  # verify that train and test have identical feature columns

## ----------------------------------------------------------------------
## 10.5 Metrics helper
## ----------------------------------------------------------------------

if (!exists("metrics_binary")) {                                          # define the helper only once to avoid overwriting in later runs
  if (!requireNamespace("pROC", quietly = TRUE)) {                        # check that the ROC and AUC package is available
    install.packages("pROC")                                              # install the package if it is missing
  }
  library(pROC)                                                           # attach pROC for ROC and AUC computation

  metrics_binary <- function(y_true, p_hat, threshold = 0.5) {            # compute standard classification metrics from labels and probabilities
    stopifnot(length(y_true) == length(p_hat))                            # confirm that labels and predictions have the same length
    y_true <- as.integer(y_true)                                          # store labels as 0 or 1 integers
    p_hat  <- as.numeric(p_hat)                                           # store predicted probabilities as numeric values

    p_hat <- pmin(pmax(p_hat, 1e-6), 1 - 1e-6)                            # clamp probabilities away from zero and one for stability

    pred_class <- ifelse(p_hat >= threshold, 1L, 0L)                      # convert probabilities to class predictions using the threshold

    accuracy <- mean(pred_class == y_true)                                # compute accuracy as the share of correctly classified observations
    error    <- 1 - accuracy                                              # compute misclassification error as one minus accuracy

    tp <- sum(pred_class == 1L & y_true == 1L)                            # count true positives at the chosen threshold
    fp <- sum(pred_class == 1L & y_true == 0L)                            # count false positives at the chosen threshold
    tn <- sum(pred_class == 0L & y_true == 0L)                            # count true negatives at the chosen threshold
    fn <- sum(pred_class == 0L & y_true == 1L)                            # count false negatives at the chosen threshold

    roc_obj <- pROC::roc(response = y_true, predictor = p_hat, quiet = TRUE)      # compute the ROC curve object from labels and probabilities
    auc_val <- as.numeric(pROC::auc(roc_obj))                             # extract the scalar AUC value

    brier <- mean((p_hat - y_true)^2)                                     # compute the Brier score as mean squared probability error

    dplyr::tibble(                                                        # return a one-row tibble for easy table binding
      error    = error,
      accuracy = accuracy,
      auc      = auc_val,
      brier    = brier,
```

```
      tp = tp, fp = fp, tn = tn, fn = fn
    )
  }
}

## ====================================================================
## 11. Regularised regression: logit & elastic net (block (a))
## ====================================================================

## --------------------------------------------------------------------
## 11.1 Unpenalised logistic regression (GLM)
## --------------------------------------------------------------------

logit_formula <- default ~ . - is_train                                  # define the model formula and exclude the split indicator

logit_model <- glm(                                                      # fit a baseline logistic regression on the training data
  formula = logit_formula,                                               # use the shared predictor set used elsewhere in the ML section
  family  = binomial(link = "logit"),                                    # specify a Bernoulli likelihood with a logit link
  data    = train_ml_imp                                                 # use the imputed training dataset
)

stopifnot(isTRUE(logit_model$converged))                                 # verify that the optimization converged successfully

logit_p_test <- predict(                                                 # generate predicted default probabilities on the test set
  logit_model,                                                           # fitted logistic regression model
  newdata = test_ml_imp,                                                 # imputed test dataset
  type    = "response"                                                   # return probabilities rather than linear predictors
)

logit_metrics <- metrics_binary(y_test, logit_p_test) %>%                # compute test-set metrics for the logistic regression
  dplyr::mutate(
    model  = "logit",                                                    # label used when combining results across models
    alpha  = NA_real_,                                                   # keep placeholder fields for consistent output structure
    lambda = NA_real_                                                    # keep placeholder fields for consistent output structure
  )

## --------------------------------------------------------------------
## 11.2 Elastic-net logistic regression (glmnet; tune alpha and lambda by CV)
## --------------------------------------------------------------------

alpha_grid <- c(0, 0.25, 0.5, 0.75, 1)                                    # define the alpha grid from ridge to lasso

set.seed(base_seed + 4)                                                  # set a seed so cross-validation folds are reproducible

```

```r
K <- 5                                                                  # set the number of cross-validation folds
foldid <- sample(rep(seq_len(K), length.out = nrow(x_train)))           # assign each training row to a fold using a fixed fold id vector

glmnet_results_list <- vector("list", length(alpha_grid))               # preallocate a list to store performance metrics by alpha
glmnet_fits         <- vector("list", length(alpha_grid))               # preallocate a list to store fitted cv.glmnet objects by alpha

for (i in seq_along(alpha_grid)) {                                      # iterate over all alpha values in the grid
  a <- alpha_grid[i]                                                    # set the current alpha value
  cat("\n[glmnet] Fitting cv.glmnet with alpha =", a, "...\n")          # print progress for long runs

  cv_fit <- glmnet::cv.glmnet(                                          # fit an elastic-net logistic regression with cross-validation
    x            = x_train,                                           # training design matrix
    y            = y_train,                                           # training labels
    family       = "binomial",                                        # logistic regression loss
    alpha        = a,                                                 # elastic-net mixing parameter
    nfolds       = K,                                                 # number of folds used in cross-validation
    foldid       = foldid,                                            # reuse identical folds across alpha values
    standardize  = TRUE,                                              # standardize predictors within glmnet
    type.measure = "deviance"                                         # select lambda by cross-validated deviance
  )

  idx_min    <- which(cv_fit$lambda == cv_fit$lambda.min)[1]          # locate the index of the selected lambda within the lambda path
  cv_err_min <- cv_fit$cvm[idx_min]                                   # store the cross-validated deviance at the selected lambda

  p_hat_test <- predict(                                              # predict test-set default probabilities at lambda.min
    cv_fit,                                                           # fitted cross-validation object
    newx = x_test,                                                    # test design matrix
    s    = "lambda.min",                                              # choose the lambda that minimizes CV deviance
    type = "response"                                                 # return probabilities
  )[, 1]

  glmnet_results_list[[i]] <- metrics_binary(y_test, p_hat_test) %>%  # evaluate the model on the common test set
    dplyr::mutate(
      model    = "glmnet",                                            # model identifier used for downstream tables and plots
      alpha    = a,                                                   # alpha value used in this fit
      lambda   = cv_fit$lambda.min,                                   # selected lambda value
```

```r
      cv_error = cv_err_min                                          # cross-validated
      ↪  deviance used for selecting among alpha values
    )

  glmnet_fits[[i]] <- cv_fit                                         # store the fitted
  ↪  object so the best model can be retrieved later
}

glmnet_metrics_all <- dplyr::bind_rows(glmnet_results_list)          # combine metrics
↪  across all alpha values into one table

# cat("\n[glmnet] Test-set performance (all alphas):\n")              # print the full
↪  alpha grid results for manual inspection
# print(glmnet_metrics_all)

glmnet_best_metrics <- glmnet_metrics_all %>%                        # select the best
↪  elastic-net model across alpha values
  dplyr::arrange(cv_error, dplyr::desc(auc)) %>%                     # prioritize lowest
  ↪  CV deviance and break ties by higher test AUC
  dplyr::slice(1)

alpha_best  <- glmnet_best_metrics$alpha                             # extract the best
↪  alpha value
lambda_best <- glmnet_best_metrics$lambda                            # extract the
↪  corresponding selected lambda value

best_idx <- which(alpha_grid == alpha_best)[1]                       # find the position
↪  of the best alpha in the stored fit list
glmnet_best_fit <- glmnet_fits[[best_idx]]                           # retrieve the fitted
↪  cv.glmnet object for the best alpha

glmnet_best_p_test <- predict(                                       # compute and store
↪  test-set probabilities for the selected elastic-net fit
  glmnet_best_fit,                                                   # best elastic-net
  ↪  fit object
  newx = x_test,                                                     # test design matrix
  s    = lambda_best,                                                # selected lambda
  ↪  value
  type = "response"                                                  # return
  ↪  probabilities
)[, 1]

## ---------------------------------------------------------------
## 11.3 Regression-family summary table
## ---------------------------------------------------------------

regression_family_metrics <- dplyr::bind_rows(                       # combine logistic
↪  regression and elastic-net results
  logit_metrics,
  glmnet_best_metrics
) %>%
  dplyr::select(model, alpha, lambda, error, accuracy, auc, brier,   # keep only the
  ↪  columns needed for the summary table
                tp, fp, tn, fn, cv_error)

cat("\nRegression-family models: summary metrics:\n")                 # print a header for
↪  the regression-family metrics table
print(regression_family_metrics)                                     # print the
↪  regression-family metrics table to the console

## ===============================================================
```

```r
## 12. Tree-based models: CART, Bagging (custom + benchmark), Random Forest
## ======================================================================

## ----------------------------------------------------------------------
## 12.0 Data objects (same predictors; no new split)
## ----------------------------------------------------------------------

tree_formula <- default ~ . - is_train                                  # use the same predictor set as in the regression models

train_tree <- train_ml_imp                                              # training data after imputation
test_tree  <- test_ml_imp                                               # test data after imputation

train_tree$default <- factor(train_tree$default, levels = c(0, 1))       # convert outcome to factor for classification trees
test_tree$default  <- factor(test_tree$default,  levels = c(0, 1))       # keep test levels identical to the training levels

y_train_vec <- as.integer(as.character(train_tree$default))              # store training labels as integer zero or one for metric functions
y_test_vec  <- as.integer(as.character(test_tree$default))               # store test labels as integer zero or one for metric functions

get_prob1 <- function(prob_obj) {                                       # extract the probability for class one from different predict outputs
  if (is.null(dim(prob_obj))) return(as.numeric(prob_obj))              # return directly when predict already gives a vector
  cn <- colnames(prob_obj)                                              # read the column names that represent class labels
  if (!is.null(cn) && "1" %in% cn) return(prob_obj[, "1"])              # take the column labeled one when it exists
  if (!is.null(cn) && "0" %in% cn) return(rep(0, nrow(prob_obj)))       # return zeros when a stump predicts only class zero
  prob_obj[, ncol(prob_obj)]                                            # fall back to the last column if labels are missing
}

## ----------------------------------------------------------------------
## 12.1 CART tree (rpart) with CV cp and stump safeguard
## ----------------------------------------------------------------------

cat("\n[CART] Fitting CART tree...\n")                                   # print a progress message

set.seed(base_seed + 5)                                                 # set a seed so the CART fit is reproducible

cart_grow <- rpart(                                                     # first grow a reasonably flexible tree
  formula = tree_formula,                                               # use the shared tree formula
  data    = train_tree,                                                 # fit on the training data
  method  = "class",                                                    # fit a classification tree
  control = rpart.control(
    cp        = 0.0005,                                                 # allow growth so pruning can be meaningful
```

```r
    minbucket = 200,                                                    # enforce a minimum leaf size for stability
    maxdepth  = 10                                                      # cap depth to avoid overly complex trees
  )
)

ctab   <- cart_grow$cptable                                             # extract the cross validation table produced by rpart
cp_min <- ctab[which.min(ctab[, "xerror"]), "CP"]                       # choose the complexity parameter with minimum cross validated error

cart_final <- prune(cart_grow, cp = cp_min)                             # prune the grown tree using the selected complexity parameter

if (!is.null(cart_final$cptable) && cart_final$cptable[1, "nsplit"] == 0) {  # detect the case where pruning removes all splits
  cat("[CART] Warning: pruned tree became a stump. Falling back to grown tree.\n") # report the fallback choice
  tree_fit     <- cart_grow                                             # use the grown tree as a fallback model
  cart_cp_used <- cp_min                                                # keep the selected complexity parameter for reporting
} else {
  tree_fit     <- cart_final                                            # use the pruned tree when it still contains splits
  cart_cp_used <- cp_min                                                # store the complexity parameter used for pruning
}

tree_prob_test <- get_prob1(predict(tree_fit, newdata = test_tree, type = "prob")) # compute test probabilities and extract the class one column

tree_metrics <- metrics_binary(y_test_vec, tree_prob_test) %>%          # evaluate CART performance on the common test set
  dplyr::mutate(
    model  = "cart_tree",                                               # model identifier used in later tables and plots
    alpha  = NA_real_,                                                  # placeholder to match the common metric table structure
    lambda = cart_cp_used                                               # store the chosen complexity parameter for reference
  )

cat("\n[CART] Test-set performance:\n")                                  # print a header for the CART metrics
print(tree_metrics)                                                     # print CART metrics

## ----------------------------------------------------------------------
## 12.2 Custom bagging with rpart and OOB tracking
## ----------------------------------------------------------------------

bagging_rpart_oob_trace <- function(train_df, test_df, formula, y_true,
                                    B = 80,
                                    minbucket = 50,
                                    maxdepth  = 20,
                                    cp        = 0.0005) {

  n_train <- nrow(train_df)                                             # number of training observations
```

```r
n_test  <- nrow(test_df)                                                    # number of test observations

test_sum  <- numeric(n_test)                                                # running sum of predicted probabilities on the test set
oob_sum   <- numeric(n_train)                                               # running sum of out of bag probabilities on the training set
oob_count <- integer(n_train)                                               # number of out of bag predictions per training observation

err_path  <- numeric(B)                                                     # store out of bag misclassification error after each tree

for (b in seq_len(B)) {                                                     # loop over the number of bootstrap trees
  if (b %% 10 == 0) cat("[Bagging] Tree", b, "of", B, "...\n")              # print progress every ten trees

  idx_b <- sample.int(n = n_train, size = n_train, replace = TRUE)          # draw a bootstrap sample of training indices

  inbag_flag <- logical(n_train)                                            # initialize in-bag indicator for this tree
  inbag_flag[idx_b] <- TRUE                                                 # mark in-bag rows for this bootstrap draw
  oob <- which(!inbag_flag)                                                 # identify out of bag rows for this tree

  train_b <- train_df[idx_b, , drop = FALSE]                                # create the bootstrap training set for this tree

  fit_b <- rpart(                                                           # fit a base CART learner on the bootstrap sample
    formula = formula,
    data    = train_b,
    method  = "class",
    control = rpart.control(                                                # skip internal cross validation to speed up bagging
      cp = cp, minbucket = minbucket, maxdepth = maxdepth, xval = 0
    )
  )

  p_test <- get_prob1(predict(fit_b, newdata = test_df, type = "prob"))     # compute predicted probabilities on the test set
  test_sum <- test_sum + p_test                                             # accumulate test probabilities for bagged averaging

  if (length(oob) > 0) {                                                    # update out of bag predictions when out of bag rows exist
    p_oob <- get_prob1(
      predict(fit_b, newdata = train_df[oob, , drop = FALSE], type = "prob")
    )
    oob_sum[oob]   <- oob_sum[oob] + p_oob                                  # accumulate out of bag probability sums for those rows
    oob_count[oob] <- oob_count[oob] + 1L                                   # increment the number of out of bag votes for those rows
  }

  has_oob <- oob_count > 0L                                                 # identify training rows with at least one out of bag prediction so far
  if (any(has_oob)) {
```

```
      curr_probs <- oob_sum[has_oob] / oob_count[has_oob]                    # compute current
      ↪  average out of bag probabilities
      curr_preds <- ifelse(curr_probs > 0.5, 1, 0)                           # convert
      ↪  probabilities to classes using threshold one half
      err_path[b] <- mean(curr_preds != y_true[has_oob])                     # compute out of bag
      ↪  misclassification error at this step
    } else {
      err_path[b] <- NA_real_                                                # keep missing when
      ↪  no out of bag predictions exist yet
    }
  }

  test_prob <- test_sum / B                                                  # compute the final
  ↪  bagged test probabilities

  oob_prob <- rep(NA_real_, n_train)                                         # initialize the out
  ↪  of bag probability vector
  has_oob  <- oob_count > 0L                                                 # identify
  ↪  observations with at least one out of bag prediction
  oob_prob[has_oob] <- oob_sum[has_oob] / oob_count[has_oob]                 # compute final out
  ↪  of bag probabilities by averaging votes

  oob_idx <- which(has_oob)                                                  # store indices of
  ↪  observations with out of bag probabilities

  list(test_prob = test_prob, oob_prob = oob_prob, oob_idx = oob_idx, err_path = err_path) #
  ↪  return test and out of bag outputs for reporting
}

set.seed(base_seed + 6)                                                      # set a seed so
↪  bagging is reproducible

cat("\n[Bagging] Fitting custom rpart bagging with OOB...\n")                  # print a progress
↪  message

bag_res <- bagging_rpart_oob_trace(
  train_df  = train_tree,                                                    # training data used
  ↪  to fit each bootstrap tree
  test_df   = test_tree,                                                     # test data used to
  ↪  compute bagged test predictions
  formula   = tree_formula,                                                  # formula shared
  ↪  across tree models
  y_true    = y_train_vec,                                                   # training truth used
  ↪  for out of bag error tracking
  B         = 80,                                                            # number of bootstrap
  ↪  trees
  minbucket = 50,                                                            # minimum leaf size
  ↪  for each base tree
  maxdepth  = 20,                                                            # maximum depth for
  ↪  each base tree
  cp        = 0.0005                                                         # complexity parameter
  ↪  controlling base tree growth
)

bagging_prob_test <- bag_res$test_prob                                       # store bagged test
↪  probabilities

bagging_test_metrics <- metrics_binary(y_test_vec, bagging_prob_test) %>%    # compute test-set
↪  metrics for custom bagging
  dplyr::mutate(model = "bagging_rpart_test", alpha = NA_real_, lambda = NA_real_)

```

```r
cat("\n[Bagging custom] Test-set performance:\n")                                  # print a header for custom bagging test metrics
print(bagging_test_metrics)                                                      # print custom bagging test metrics

y_oob <- y_train_vec[bag_res$oob_idx]                                            # store out of bag truth labels for observations with OOB predictions

bagging_oob_metrics <- metrics_binary(                                           # compute metrics on the out of bag predictions only
  y_true = y_oob,
  p_hat  = bag_res$oob_prob[bag_res$oob_idx]
) %>%
  dplyr::mutate(model = "bagging_rpart_oob", alpha = NA_real_, lambda = NA_real_)

cat("\n[Bagging custom] OOB performance:\n")                                       # print a header for custom bagging out of bag metrics
print(bagging_oob_metrics)                                                       # print custom bagging out of bag metrics

## -----------------------------------------------------------------------
## 12.3 Benchmark bagging (ipred::bagging) and its OOB error
## -----------------------------------------------------------------------

set.seed(base_seed + 7)                                                          # set a seed so benchmark bagging is reproducible

cat("\n[Bagging] Fitting ipred::bagging (benchmark)...\n")                        # print a progress message

bag_std <- ipred::bagging(
  formula = tree_formula,                                                        # formula shared with other tree models
  data    = train_tree,                                                          # training data used for fitting
  nbagg   = 80,                                                                  # number of bootstrap trees
  coob    = TRUE,                                                                # request out of bag error reporting
  control = rpart.control(xval = 0, cp = 0.0, minbucket = 50)                    # speed up by skipping internal CV inside each tree
)

if (!is.null(bag_std$err)) {                                                     # ipred stores an out of bag error path in the err slot
  oob_err_ipred <- tail(bag_std$err, 1)                                          # store the final out of bag misclassification error
  cat("\n[ipred::bagging] Final OOB misclassification error:\n")                 # print a header for the final out of bag error
  print(oob_err_ipred)                                                           # print the final out of bag error
}

bag_std_prob_test <- get_prob1(predict(bag_std, newdata = test_tree, type = "prob")) # compute benchmark bagging probabilities on the test set

bagging_ipred_test_metrics <- metrics_binary(y_test_vec, bag_std_prob_test) %>%     # compute test metrics for benchmark bagging
  dplyr::mutate(model = "bagging_ipred_test", alpha = NA_real_, lambda = NA_real_)

```

```r
cat("\n[ipred::bagging] Test-set performance:\n")                    # print a header for benchmark bagging test metrics
print(bagging_ipred_test_metrics)                                    # print benchmark bagging test metrics

## ---------------------------------------------------------------------
## 12.4 Random forest (ranger)
## ---------------------------------------------------------------------

set.seed(base_seed + 8)                                              # set a seed so the random forest fit is reproducible

cat("\n[RF] Fitting random forest (ranger)...\n")                     # print a progress message

rf_best_fit <- ranger(
  formula         = tree_formula,                                    # formula shared across models
  data            = train_tree,                                      # training data used for fitting
  num.trees       = 300,                                             # number of trees in the forest
  mtry            = floor(sqrt(ncol(train_tree) - 1)),               # default heuristic for candidate variables per split
  min.node.size   = 100,                                             # minimum terminal node size for regularization
  probability     = TRUE,                                            # request class probability predictions
  importance      = "impurity"                                       # compute impurity-based variable importance
)

rf_pred_test <- predict(rf_best_fit, data = test_tree)$predictions[, "1"]    # extract predicted probabilities for class one on the test set

rf_best_metrics <- metrics_binary(y_test_vec, rf_pred_test) %>%      # compute test metrics for the random forest
  dplyr::mutate(model = "random_forest_best", alpha = NA_real_, lambda = NA_real_)

cat("\n[RF] Test-set performance:\n")                                 # print a header for random forest test metrics
print(rf_best_metrics)                                               # print random forest test metrics

## ---------------------------------------------------------------------
## 12.5 Collect tree-family test-set metrics (same test set)
## ---------------------------------------------------------------------

tree_family_metrics <- dplyr::bind_rows(                             # combine test-set metrics across all tree-based models
  tree_metrics,
  bagging_test_metrics,
  bagging_ipred_test_metrics,
  rf_best_metrics
)

cat("\nTree-based models: test-set performance summary (same test set):\n")  # print a header for the combined tree metrics
print(tree_family_metrics)                                           # print the combined tree metrics

```

```r
cat("\nCustom bagging OOB metric (for discussion):\n")                    # print a header for the out of bag metrics
print(bagging_oob_metrics)                                                # print the out of bag metrics for custom bagging

## ====================================================================
## 12.6 OOB reporting outputs (tables plus optional OOB curve plot)
## ====================================================================

## --------------------------------------------------------------------
## 12.6.1 OOB summary table (final + best + extra metrics where possible)
## --------------------------------------------------------------------

stopifnot(exists("bag_res"), exists("bag_std"), exists("rf_best_fit"),
          exists("bagging_oob_metrics")) # ensure required objects exist before reporting

custom_err_path <- bag_res$err_path                                       # extract the custom bagging out of bag error path
custom_err_path <- custom_err_path[!is.na(custom_err_path)]               # remove missing entries from early iterations if present

summarise_oob_path <- function(err_path) {                                # summarise an out of bag error path into a few reporting statistics
  tibble::tibble(
    n_trees_final   = length(err_path),                                   # total number of trees reported in the error path
    oob_error_final = as.numeric(tail(err_path, 1)),                      # final out of bag misclassification error
    oob_error_min   = as.numeric(min(err_path)),                          # minimum out of bag misclassification error achieved
    n_trees_at_min  = as.integer(which.min(err_path))                     # tree index where the minimum out of bag error occurs
  )
}

oob_custom_sum <- summarise_oob_path(custom_err_path)                     # compute the custom bagging out of bag summary statistics

oob_ipred_sum <- tibble::tibble(                                          # create a comparable summary row for ipred bagging
  n_trees_final   = 80L,                                                  # number of trees used in ipred bagging
  oob_error_final = as.numeric(oob_err_ipred),                            # final out of bag error from ipred
  oob_error_min   = NA_real_,                                             # not available because a full path is not used here
  n_trees_at_min  = NA_integer_                                           # not available because a full path is not used here
)

oob_rf_sum <- tibble::tibble(                                             # create a comparable summary row for random forest
  n_trees_final   = as.integer(rf_best_fit$num.trees),                    # number of trees used in the forest
  oob_error_final = as.numeric(rf_best_fit$prediction.error),             # built-in out of bag misclassification error from ranger
  oob_error_min   = NA_real_,                                             # not available because a full path is not stored by default
  n_trees_at_min  = NA_integer_                                           # not available because a full path is not stored by default
)
```

```r

oob_report_tbl <- dplyr::bind_rows(                                    # bind all out of bag summaries into one reporting table
  dplyr::mutate(oob_custom_sum, method = "Bagging custom",       extra_note = "Full OOB path and OOB probabilities"),
  dplyr::mutate(oob_ipred_sum,  method = "Bagging ipred",        extra_note = "Only final OOB misclassification available"),
  dplyr::mutate(oob_rf_sum,     method = "Random forest ranger", extra_note = "Built-in OOB error as a single value")
) %>%
  dplyr::select(method, n_trees_final, oob_error_final, oob_error_min, n_trees_at_min, extra_note) %>%
  dplyr::mutate(
    oob_auc   = dplyr::if_else(method == "Bagging custom", bagging_oob_metrics$auc,   NA_real_),   # add OOB AUC only where OOB probabilities exist
    oob_brier = dplyr::if_else(method == "Bagging custom", bagging_oob_metrics$brier, NA_real_)   # add OOB Brier only where OOB probabilities exist
  )

oob_report_tex <- oob_report_tbl %>%                                   # convert the OOB report into a LaTeX-ready data frame
  dplyr::transmute(
    Method      = method,                                              # method label shown in the paper table
    `Number of Trees`    = as.integer(n_trees_final),                   # number of trees used by each method
    `OOB error (misclassification)` = as.numeric(oob_error_final),      # out of bag misclassification error
    `OOB AUC`    = as.numeric(oob_auc),                                 # OOB AUC when available
    `OOB Brier` = as.numeric(oob_brier)                                 # OOB Brier when available
  ) %>%
  as.data.frame()

stargazer::stargazer(                                                  # export the OOB report table to LaTeX
  oob_report_tex,
  type = "latex",
  summary = FALSE,
  rownames = FALSE,
  digits = 3,
  no.space = TRUE,
  table.placement = "!htbp",
  title = "Out-of-bag performance summary",
  label = "tab:oob_report",
  out = "table_oob_report.tex"
)

## ---------------------------------------------------------------------
## 12.6.2 OOB curve plot
## ---------------------------------------------------------------------

ipred_oob_final <- as.numeric(oob_err_ipred)                           # store the final ipred out of bag misclassification error

oob_plot_df <- tibble::tibble(                                         # build a plotting data frame for the custom bagging error path
  B       = seq_along(custom_err_path),                                # tree index from one to the final number of trees
```

```r
  oob_err = custom_err_path                                    # out of bag misclassification error at each tree index
)

B_star   <- oob_custom_sum$n_trees_at_min                      # tree index where the custom bagging OOB error is minimized
oob_star <- oob_custom_sum$oob_error_min                       # minimum custom bagging OOB error value

y_lo <- min(oob_plot_df$oob_err, ipred_oob_final) - 0.0008     # lower plot limit with a small margin
y_hi <- max(oob_plot_df$oob_err, ipred_oob_final) + 0.0008     # upper plot limit with a small margin

plot_oob_bagging <- ggplot2::ggplot(oob_plot_df, ggplot2::aes(x = B, y = oob_err)) +
  ggplot2::geom_line(linewidth = 1.0, colour = "#0072B2") +    # plot the custom bagging OOB error path
  ggplot2::geom_point(                                         # mark the minimum point on the custom OOB curve
    data = tibble::tibble(B = B_star, oob_err = oob_star),
    ggplot2::aes(x = B, y = oob_err),
    size = 2.0, colour = "#0072B2"
  ) +
  ggplot2::geom_hline(                                         # add a horizontal reference line for the ipred final OOB error
    yintercept = ipred_oob_final,
    linetype = 2, linewidth = 0.9, colour = "#D30E00"
  ) +
  ggplot2::annotate(                                           # label the ipred reference line directly on the plot
    "text",
    x = max(oob_plot_df$B),
    y = ipred_oob_final,
    label = paste0("ipred final OOB = ", scales::percent(ipred_oob_final, accuracy = 0.1)),
    hjust = 1.05, vjust = -0.7, size = 4.2, colour = "#D30E00"
  ) +
  ggplot2::annotate(                                           # label the minimum point on the custom OOB curve
    "text",
    x = B_star,
    y = oob_star,
    label = paste0("min at B = ", B_star, ": ", scales::percent(oob_star, accuracy = 0.1)),
    hjust = -0.05, vjust = 1.2, size = 4.2, colour = "#0072B2"
  ) +
  ggplot2::scale_y_continuous(
    limits = c(y_lo, y_hi),
    labels = scales::percent_format(accuracy = 0.1)
  ) +
  ggplot2::labs(x = "Number of trees", y = "OOB misclassification error") + # label axes for interpretation
  theme_academic() +                                           # apply the project theme
  theme(
    axis.title.x = ggplot2::element_text(size = 14),           # slightly larger axis titles for paper readability
    axis.title.y = ggplot2::element_text(size = 14),
    axis.text.x  = ggplot2::element_text(size = 12),
    axis.text.y  = ggplot2::element_text(size = 12)
  )

```

```r
print(plot_oob_bagging)                                                    # display the OOB curve plot

ggplot2::ggsave(                                                           # save the OOB curve plot at high resolution
  filename = "plot_oob_bagging.png",
  plot     = plot_oob_bagging,
  width    = 12, height = 4, units = "in",
  dpi      = 1000
)

## =====================================================================
## 13. Model comparison
## =====================================================================

## ---------------------------------------------------------------------
## 13.1 Sanity checks: required objects from Parts 11 and 12
## ---------------------------------------------------------------------

need_objs <- c(                                                            # objects that must exist before running Part 13
  "y_test", "metrics_binary",                                              # common test labels and metrics helper
  "logit_p_test", "glmnet_best_p_test",                                    # Part 11 test probabilities
  "tree_prob_test", "bagging_prob_test", "bag_std_prob_test", "rf_pred_test", # Part 12 test probabilities
  "bagging_oob_metrics", "rf_best_fit"                                     # Part 12 out of bag info and fitted random forest object
)

missing <- need_objs[!vapply(need_objs, exists, logical(1L))]              # identify which required objects are missing
if (length(missing) > 0) {                                                 # stop early with a clear message when prerequisites are not met
  stop("Part 13 missing objects (run Parts 11 and 12 first):\n  ",
       paste(missing, collapse = ", "))
}

y_test_use <- as.integer(y_test)                                           # ensure test labels are stored as 0 or 1 integers

## ---------------------------------------------------------------------
## 13.2 Collect predictions in one place (same test set)
## ---------------------------------------------------------------------

pred_tbl <- tibble::tibble(                                                # build a single table with true labels and model probabilities
  y            = y_test_use,                                               # true test label stored as 0 or 1
  p_logit      = as.numeric(logit_p_test),                                 # logit predicted probability of default equals one
  p_enet       = as.numeric(glmnet_best_p_test),                           # elastic net predicted probability of default equals one
  p_cart       = as.numeric(tree_prob_test),                               # CART predicted probability of default equals one
  p_bag_custom = as.numeric(bagging_prob_test),                            # custom bagging predicted probability of default equals one
  p_bag_ipred  = as.numeric(bag_std_prob_test),                            # ipred bagging predicted probability of default equals one
```

```r
  p_rf         = as.numeric(rf_pred_test)                                  # random forest predicted probability of default equals one
)

pred_long <- pred_tbl %>%                                                  # reshape to long format for plotting and grouped calculations
  tidyr::pivot_longer(
    cols      = dplyr::starts_with("p_"),                                  # gather all model probability columns
    names_to  = "model",
    values_to = "p_hat"
  ) %>%
  dplyr::mutate(
    model = dplyr::recode(                                                 # map internal column names to paper-ready model names
      model,
      p_logit      = "Logit",
      p_enet       = "Elastic net",
      p_cart       = "CART",
      p_bag_custom = "Bagging (custom)",
      p_bag_ipred  = "Bagging (ipred)",
      p_rf         = "Random forest"
    ),
    model = factor(                                                        # fix the ordering for legends and tables
      model,
      levels = c("Logit", "Elastic net", "CART",
                 "Bagging (custom)", "Bagging (ipred)", "Random forest")
    )
  )

## ---------------------------------------------------------------------
## 13.3 Main comparison table (computed from stored predictions)
## ---------------------------------------------------------------------

metric_one <- function(y, p) {                                             # wrapper so metric computation reads cleanly below
  metrics_binary(y_true = y, p_hat = p)                                    # apply the same metric definition to every model
}

model_metrics_test <- tibble::tibble(                                      # map model names to the probability columns in pred_tbl
  model = levels(pred_long$model),                                         # model names in the fixed order
  p_col = c("p_logit", "p_enet", "p_cart", "p_bag_custom", "p_bag_ipred", "p_rf")
) %>%
  dplyr::mutate(
    met = purrr::map(p_col, ~ metric_one(pred_tbl$y, pred_tbl[[.x]]))      # compute metrics on the same holdout test set
  ) %>%
  tidyr::unnest(met) %>%                                                   # unpack the returned metric tibble columns
  dplyr::select(model, error, accuracy, auc, brier, tp, fp, tn, fn) %>%    # keep only reporting columns
  dplyr::arrange(dplyr::desc(auc))                                         # rank primarily by AUC

cat("\n=== Test-set performance (all models, same test set) ===\n")        # print a header for the comparison table
```

```r
print(model_metrics_test)                                                    # print the model comparison table

model_metrics_tex <- model_metrics_test %>%                                  # create a LaTeX-export version with cleaned model labels
  dplyr::mutate(
    Model = dplyr::case_when(                                                # standardize model naming for the paper table
      model %in% c("logit", "Logit", "Logistic") ~ "Logit",
      model %in% c("enet", "ENet", "ElasticNet") ~ "Elastic Net",
      model %in% c("cart", "CART") ~ "CART",
      model %in% c("bag_custom", "Bag (custom)") ~ "Bagging (custom)",
      model %in% c("bag_ipred", "Bag (ipred)") ~ "Bagging (ipred)",
      model %in% c("rf", "RF", "RandomForest") ~ "Random Forest",
      TRUE ~ as.character(model)
    )
  ) %>%
  dplyr::select(
    Model,
    Error = error,
    Accuracy = accuracy,
    AUC = auc,
    Brier = brier,
    TP = tp, FP = fp, TN = tn, FN = fn
  ) %>%
  as.data.frame()

stargazer::stargazer(
  model_metrics_tex,                                                         # LaTeX-ready data frame with performance metrics
  type = "latex",
  summary = FALSE,
  rownames = FALSE,
  digits = 3,                                                                # format metrics with a consistent number of decimals
  no.space = TRUE,
  table.placement = "!htbp",
  title = "Test-set performance comparison (common test set)",
  label = "tab:model_metrics_test",
  out = "table_model_metrics_test.tex"
)

## ----------------------------------------------------------------------
## 13.4 Plot A and B: ROC and PR curves (two-panel)
## ----------------------------------------------------------------------

roc_df <- pred_long %>%                                                      # build ROC curve points for each model
  dplyr::group_by(model) %>%
  dplyr::group_modify(~{
    r <- pROC::roc(response = .x$y, predictor = .x$p_hat, quiet = TRUE)      # compute the ROC curve from probabilities and true labels
    tibble::tibble(
      fpr = 1 - r$specificities,                                             # false positive rate across thresholds
      tpr = r$sensitivities                                                  # true positive rate across thresholds
    )
  }) %>%
  dplyr::ungroup()

```

```r
plot_roc <- ggplot2::ggplot(roc_df, ggplot2::aes(x = fpr, y = tpr, colour = model)) +
  ggplot2::geom_line(linewidth = 0.9) +                                    # plot ROC curves for each model
  ggplot2::geom_abline(slope = 1, intercept = 0, linetype = 2, colour = "grey60") + # add random screening baseline
  ggplot2::scale_colour_manual(values = c(                                 # apply a fixed palette so models match across figures
    "Logit"            = "#0072B2",
    "Elastic net"      = "#D55E00",
    "CART"             = "#009E73",
    "Bagging (custom)" = "#CC79A7",
    "Bagging (ipred)"  = "#E69F00",
    "Random forest"    = "#56B4E9"
  )) +
  ggplot2::labs(x = "False positive rate", y = "True positive rate", colour = NULL) +
  theme_academic() +
  ggplot2::theme(legend.position = "bottom")                               # place legend below the plot for paper layout

pr_curve_df <- function(y, p, model_name, n_points = 250L) {               # construct a precision recall curve by sweeping thresholds
  thr <- stats::quantile(p, probs = seq(0, 1, length.out = n_points), na.rm = TRUE) # build a grid of thresholds based on probability quantiles
  thr <- sort(unique(as.numeric(thr)), decreasing = TRUE)                  # enforce unique thresholds from high to low
  out <- lapply(thr, function(t) {
    pred <- as.integer(p >= t)                                             # classify as one when probability is at least the threshold
    tp <- sum(pred == 1L & y == 1L)                                        # true positives at this threshold
    fp <- sum(pred == 1L & y == 0L)                                        # false positives at this threshold
    fn <- sum(pred == 0L & y == 1L)                                        # false negatives at this threshold
    precision <- ifelse(tp + fp == 0L, NA_real_, tp / (tp + fp))          # precision defined as true positives divided by predicted positives
    recall    <- ifelse(tp + fn == 0L, NA_real_, tp / (tp + fn))          # recall defined as true positives divided by actual positives
    c(precision = precision, recall = recall)
  })
  mat <- do.call(rbind, out)                                               # stack the threshold results into a matrix
  tibble::tibble(
    recall    = mat[, "recall"],                                           # recall values across thresholds
    precision = mat[, "precision"],                                        # precision values across thresholds
    model     = model_name                                                 # model name used in the legend
  ) %>%
    dplyr::filter(!is.na(precision), !is.na(recall))                       # drop undefined points for stable plotting
}

pr_df <- dplyr::bind_rows(                                                 # build PR curve points for each model on the same test set
  pr_curve_df(pred_tbl$y, pred_tbl$p_logit,      "Logit"),
  pr_curve_df(pred_tbl$y, pred_tbl$p_enet,       "Elastic net"),
  pr_curve_df(pred_tbl$y, pred_tbl$p_cart,       "CART"),
  pr_curve_df(pred_tbl$y, pred_tbl$p_bag_custom, "Bagging (custom)"),
  pr_curve_df(pred_tbl$y, pred_tbl$p_bag_ipred,  "Bagging (ipred)"),
```

```r
  pr_curve_df(pred_tbl$y, pred_tbl$p_rf,         "Random forest")
) %>%
  dplyr::mutate(model = factor(model, levels = levels(pred_long$model)))     # enforce the same
  ↪ legend ordering as pred_long

plot_pr <- ggplot2::ggplot(pr_df, ggplot2::aes(x = recall, y = precision, colour = model)) +
  ggplot2::geom_line(linewidth = 0.9) +                                      # plot PR curves for
  ↪ each model
  ggplot2::scale_colour_manual(values = c(                                   # reuse the same
  ↪ palette as the ROC plot
    "Logit"            = "#0072B2",
    "Elastic net"      = "#D55E00",
    "CART"             = "#009E73",
    "Bagging (custom)" = "#CC79A7",
    "Bagging (ipred)"  = "#E69F00",
    "Random forest"    = "#56B4E9"
  )) +
  ggplot2::labs(x = "Recall", y = "Precision", colour = NULL) +
  theme_academic() +
  ggplot2::theme(legend.position = "bottom")                                 # keep the legend
  ↪ consistent with the ROC plot

plot_roc_pr <- (plot_roc | plot_pr) +
  patchwork::plot_annotation(
    tag_levels = "a",                                                        # label panels for
    ↪ referencing in the text
    tag_prefix = "(",
    tag_suffix = ")"
  )

print(plot_roc_pr)                                                           # print the combined
↪ ROC and PR panel

ggplot2::ggsave(
  filename = "plot_roc_pr.png",                                              # save the combined
  ↪ ROC and PR panel as a high-resolution figure
  plot     = plot_roc_pr,
  width    = 10, height = 6, units = "in",
  dpi      = 1000
)

## ---------------------------------------------------------------------
## 13.5 Plot C: Cumulative gains and lift
## ---------------------------------------------------------------------

lift_df <- pred_long %>%                                                     # compute gains curves
↪ by sorting predicted risk within each model
  dplyr::group_by(model) %>%
  dplyr::arrange(dplyr::desc(p_hat), .by_group = TRUE) %>%                   # order borrowers from
  ↪ highest predicted risk to lowest
  dplyr::mutate(
    frac_population        = dplyr::row_number() / dplyr::n(),               # share of the
    ↪ population screened at each cutoff
    cum_defaults           = cumsum(y),                                      # cumulative number of
    ↪ defaults captured as screening expands
    total_defaults         = sum(y),                                         # total defaults in
    ↪ the test set for this model group
    frac_defaults_captured = cum_defaults / total_defaults                   # share of defaults
    ↪ captured at each cutoff
  ) %>%
  dplyr::ungroup()
```

```r

plot_lift <- ggplot2::ggplot(lift_df, ggplot2::aes(x = frac_population, y = frac_defaults_captured, colour = model)) +
  ggplot2::geom_line(linewidth = 0.9) +                                    # plot gains curves for each model
  ggplot2::geom_abline(slope = 1, intercept = 0, linetype = 2, colour = "grey60") + # add random screening baseline
  ggplot2::scale_colour_manual(values = c(                                 # reuse the same palette for model consistency
    "Logit"            = "#0072B2",
    "Elastic net"      = "#D55E00",
    "CART"             = "#009E73",
    "Bagging (custom)" = "#CC79A7",
    "Bagging (ipred)"  = "#E69F00",
    "Random forest"    = "#56B4E9"
  )) +
  ggplot2::scale_x_continuous(labels = scales::percent_format(accuracy = 1)) +
  ggplot2::scale_y_continuous(labels = scales::percent_format(accuracy = 1)) +
  ggplot2::labs(
    x = "Share of borrowers screened (highest risk first)",
    y = "Share of defaults captured",
    colour = NULL
  ) +
  theme_academic() +
  ggplot2::theme(legend.position = "bottom")                               # keep legend placement consistent across figures

print(plot_lift)                                                           # print the lift plot

ggplot2::ggsave(
  filename = "plot_lift.png",                                              # save the lift plot as a high-resolution figure
  plot     = plot_lift,
  width    = 12, height = 6, units = "in",
  dpi      = 1000
)

## ----------------------------------------------------------------------
## 13.6 Plot D: Predicted risk distributions by class (faceted)
## ----------------------------------------------------------------------

plot_risk_dist <- ggplot2::ggplot(
  pred_long %>% dplyr::mutate(y_fac = factor(y, levels = c(0, 1), labels = c("No default", "Default"))), # label true outcomes for plotting
  ggplot2::aes(x = p_hat, fill = y_fac)
  # map predicted risk and outcome group
) +
  ggplot2::geom_density(alpha = 0.35, adjust = 1.1) +                       # plot overlapping densities to compare separation by outcome
  ggplot2::facet_wrap(~ model, ncol = 3, scales = "free_y") +               # facet by model to compare distributions side-by-side
  ggplot2::scale_x_continuous(labels = scales::percent_format(accuracy = 1)) +
  ggplot2::labs(x = "Predicted default probability", y = "Density", fill = NULL) +
  theme_academic() +
  ggplot2::theme(legend.position = "bottom")                               # place legend below to match other figures

print(plot_risk_dist)                                                      # print the risk distribution figure

ggplot2::ggsave(
```

```r
  filename = "plot_risk_dist.png",                                    # save the risk distribution figure as a high-resolution plot
  plot     = plot_risk_dist,
  width    = 10, height = 6, units = "in",
  dpi      = 1000
)

## ====================================================================
## 14. Double machine learning extension: Effect of 60-month term on default
## ====================================================================

## --------------------------------------------------------------------
## 14.1 DML dataset (train only; imputed; numeric outcome and treatment)
## --------------------------------------------------------------------

dml_df <- train_ml_imp %>%                                            # start from the imputed training sample
  dplyr::select(-is_train) %>%                                        # drop the split flag since all rows here are training rows
  dplyr::mutate(
    default = as.numeric(default),                                    # store the outcome as numeric zero or one for DML algebra
    term_60 = as.numeric(term_60)                                     # store the treatment as numeric zero or one for DML algebra
  )

n_dml <- nrow(dml_df)                                                 # store the DML training sample size
y     <- dml_df$default                                               # extract the outcome vector
d     <- dml_df$term_60                                               # extract the treatment vector

## --------------------------------------------------------------------
## 14.2 Cross-fitting setup (precompute X once; fixed lambdas for speed)
## --------------------------------------------------------------------

K <- 5                                                                # set the number of folds used for cross-fitting
set.seed(base_seed + 9)                                               # set a seed so fold assignment is reproducible

fold_id <- sample(rep(seq_len(K), length.out = n_dml))                # assign roughly balanced fold labels

X_full <- model.matrix(                                               # build the one-hot encoded covariate matrix and exclude outcome and treatment
  ~ . - default - term_60,
  data = dml_df
)

alpha_dml <- if (exists("alpha_best")) alpha_best else 0.75           # reuse the selected elastic-net mixing value when available

set.seed(base_seed + 10)                                              # set a seed so the global lambda selection is reproducible

cv_g_full <- glmnet::cv.glmnet(                                       # select one lambda for the outcome model g(X) on the full training sample
  x            = X_full,
  y            = y,
```

```r
  family       = "binomial",
  alpha        = alpha_dml,
  nfolds       = 5,
  standardize  = TRUE,
  type.measure = "deviance"
)

cv_m_full <- glmnet::cv.glmnet(                                    # select one lambda for the treatment model m(X) on the full training sample
  x            = X_full,
  y            = d,
  family       = "binomial",
  alpha        = alpha_dml,
  nfolds       = 5,
  standardize  = TRUE,
  type.measure = "deviance"
)

lambda_g <- cv_g_full$lambda.min                                   # store the selected lambda for g(X) equals expected outcome given X
lambda_m <- cv_m_full$lambda.min                                   # store the selected lambda for m(X) equals expected treatment given X

m_hat <- numeric(n_dml)                                            # allocate out-of-fold predicted propensities
g_hat <- numeric(n_dml)                                            # allocate out-of-fold predicted default risks

## ---------------------------------------------------------------------
## 14.3 Cross-fitted nuisance predictions: g_hat and m_hat
## ---------------------------------------------------------------------

for (k in seq_len(K)) {                                            # loop over folds for cross-fitting
  cat("[DML] Fold", k, "of", K, "...\n")                           # print progress so long runs are visible

  idx_tr <- which(fold_id != k)                                    # training indices for this fold
  idx_va <- which(fold_id == k)                                    # validation indices for this fold

  fit_m <- glmnet::glmnet(                                         # fit the treatment model on the training folds only
    x           = X_full[idx_tr, , drop = FALSE],
    y           = d[idx_tr],
    family      = "binomial",
    alpha       = alpha_dml,
    lambda      = lambda_m,
    standardize = TRUE
  )

  m_hat[idx_va] <- predict(                                        # predict propensities on the held-out fold
    fit_m,
    newx = X_full[idx_va, , drop = FALSE],
    s    = lambda_m,
    type = "response"
  )[, 1]

```

```r
  fit_g <- glmnet::glmnet(                                    # fit the outcome model on the training folds only
    x           = X_full[idx_tr, , drop = FALSE],
    y           = y[idx_tr],
    family      = "binomial",
    alpha       = alpha_dml,
    lambda      = lambda_g,
    standardize = TRUE
  )

  g_hat[idx_va] <- predict(                                   # predict default risks on the held-out fold
    fit_g,
    newx = X_full[idx_va, , drop = FALSE],
    s    = lambda_g,
    type = "response"
  )[, 1]
}

eps  <- 1e-4                                                  # choose a small value to avoid extreme predicted probabilities
m_hat <- pmin(pmax(m_hat, eps), 1 - eps)                      # clamp estimated propensities away from zero and one
g_hat <- pmin(pmax(g_hat, eps), 1 - eps)                      # clamp estimated risks away from zero and one

## ---------------------------------------------------------------------
## 14.4 Orthogonalized regression: theta via residual-on-residual
## ---------------------------------------------------------------------

tilde_y <- y - g_hat                                          # compute residualized outcome using cross-fitted g_hat
tilde_d <- d - m_hat                                          # compute residualized treatment using cross-fitted m_hat

dml_fit <- lm(tilde_y ~ tilde_d)                              # regress residualized outcome on residualized treatment

theta_dml      <- unname(coef(dml_fit)["tilde_d"])            # store the DML effect estimate in probability units
theta_dml_ci   <- confint(dml_fit)["tilde_d", ]               # store the ninety-five percent confidence interval
theta_dml_se   <- summary(dml_fit)$coefficients["tilde_d", "Std. Error"]     # store the standard error
theta_dml_pval <- summary(dml_fit)$coefficients["tilde_d", "Pr(>|t|)"]       # store the p-value for the residual-on-residual coefficient

cat("\n[DML] Estimated ATE of 60-month term on default (train sample):\n")     # print a report header
cat(sprintf("  theta_DML  = %.4f (%.2f percentage points)\n",                 # print the estimate in probability units and percentage points
            theta_dml, 100 * theta_dml))
cat(sprintf("  95%% CI     = [%.4f, %.4f]\n",                                 # print the confidence interval endpoints
            theta_dml_ci[1], theta_dml_ci[2]))
cat(sprintf("  Std. Error = %.4f,  p-value = %.4g\n",                        # print the standard error and p-value
            theta_dml_se, theta_dml_pval))

## ---------------------------------------------------------------------
## 14.5 Naive benchmark: raw difference in default rates (60m minus 36m)
```

```r
## ----------------------------------------------------------------------

mean_d1 <- mean(y[d == 1])                                          # compute the default rate among treated loans
mean_d0 <- mean(y[d == 0])                                          # compute the default rate among control loans
theta_naive <- mean_d1 - mean_d0                                    # compute the raw difference in means

se_naive <- sqrt(                                                   # compute a two-sample standard error under independence
  var(y[d == 1]) / sum(d == 1) +
    var(y[d == 0]) / sum(d == 0)
)

ci_naive <- theta_naive + c(-1, 1) * 1.96 * se_naive                # compute a ninety-five percent confidence interval

cat("\n[Naive] Difference in default rates (60m minus 36m):\n")      # print a report header
cat(sprintf("  theta_naive = %.4f (%.2f percentage points)\n",      # print the naive estimate in probability units and percentage points
            theta_naive, 100 * theta_naive))
cat(sprintf("  95%% CI      = [%.4f, %.4f]\n",                       # print the naive confidence interval endpoints
            ci_naive[1], ci_naive[2]))

dml_comparison <- dplyr::tibble(                                     # build a compact table comparing naive and DML estimates
  estimator = c("Naive diff-in-means", "DML (cross-fit elastic net)"),
  theta     = c(theta_naive, theta_dml),
  ci_low    = c(ci_naive[1], theta_dml_ci[1]),
  ci_high   = c(ci_naive[2], theta_dml_ci[2])
)

cat("\n=== DML vs naive effect of 60-month term on default (train sample) ===\n") # print a header for the comparison table
print(dml_comparison)                                               # print the comparison table

## ======================================================================
## 15. DML extension II: Effect of high interest rate on default
## ======================================================================

## ----------------------------------------------------------------------
## 15.1 Define high-rate treatment and build DML dataset (train only)
## ----------------------------------------------------------------------

high_rate_thr <- quantile(train_ml_imp$int_rate, 0.75, na.rm = TRUE)  # compute the seventy-fifth percentile of interest rate in the training sample
cat("\n[High-rate DML] 75th percentile threshold for int_rate:",
    round(high_rate_thr, 2), "%\n")                                 # report the threshold that defines the high-rate treatment

dml_hr_df <- train_ml_imp %>%                                       # start from the training sample to keep evaluation data separated
  dplyr::select(-is_train) %>%                                      # drop the split flag since it is not a covariate
  dplyr::mutate(
    default   = as.numeric(default),                                # store the outcome as numeric zero or one
```

```r
    high_rate = as.numeric(int_rate >= high_rate_thr)                    # store the treatment as numeric one when interest rate is above the threshold
  )

n_hr <- nrow(dml_hr_df)                                                  # store the DML sample size for the high-rate analysis
y_hr <- dml_hr_df$default                                                # extract the outcome vector
d_hr <- dml_hr_df$high_rate                                              # extract the treatment vector

## ----------------------------------------------------------------------
## 15.2 Cross-fitting setup (precompute X once; fixed lambdas for speed)
## ----------------------------------------------------------------------

K_hr <- 5                                                                # set the number of folds used for cross-fitting
set.seed(base_seed + 11)                                                 # set a seed so fold assignment is reproducible

fold_id_hr <- sample(rep(seq_len(K_hr), length.out = n_hr))              # assign roughly balanced fold labels

X_full_hr <- model.matrix(                                               # build the one-hot encoded covariate matrix and exclude outcome and treatment
  ~ . - default - high_rate,
  data = dml_hr_df
)

alpha_dml_hr <- if (exists("alpha_best")) alpha_best else 0.75           # reuse the selected elastic-net mixing value when available

set.seed(base_seed + 12)                                                 # set a seed so global lambda selection is reproducible

cv_g_hr_full <- glmnet::cv.glmnet(                                       # select one lambda for the outcome model on the full training sample
  x            = X_full_hr,
  y            = y_hr,
  family       = "binomial",
  alpha        = alpha_dml_hr,
  nfolds       = 5,
  standardize  = TRUE,
  type.measure = "deviance"
)

cv_m_hr_full <- glmnet::cv.glmnet(                                       # select one lambda for the treatment model on the full training sample
  x            = X_full_hr,
  y            = d_hr,
  family       = "binomial",
  alpha        = alpha_dml_hr,
  nfolds       = 5,
  standardize  = TRUE,
  type.measure = "deviance"
)

lambda_g_hr <- cv_g_hr_full$lambda.min                                   # store the selected lambda for the outcome model
lambda_m_hr <- cv_m_hr_full$lambda.min                                   # store the selected lambda for the treatment model
```

```r

m_hat_hr <- numeric(n_hr)                                   # allocate out-of-fold predicted propensities
g_hat_hr <- numeric(n_hr)                                   # allocate out-of-fold predicted default risks

## ----------------------------------------------------------------------
## 15.3 Cross-fitted nuisance predictions: m_hat_hr and g_hat_hr
## ----------------------------------------------------------------------

for (k in seq_len(K_hr)) {                                  # loop over folds for cross-fitting
  cat("[High-rate DML] Fold", k, "of", K_hr, "...\n")       # print progress for long runs

  idx_tr <- which(fold_id_hr != k)                          # training indices for this fold
  idx_va <- which(fold_id_hr == k)                          # validation indices for this fold

  fit_m_hr <- glmnet::glmnet(                               # fit the treatment model on training folds only
    x           = X_full_hr[idx_tr, , drop = FALSE],
    y           = d_hr[idx_tr],
    family      = "binomial",
    alpha       = alpha_dml_hr,
    lambda      = lambda_m_hr,
    standardize = TRUE
  )

  m_hat_hr[idx_va] <- predict(                              # predict propensities on the held-out fold
    fit_m_hr,
    newx = X_full_hr[idx_va, , drop = FALSE],
    s    = lambda_m_hr,
    type = "response"
  )[, 1]

  fit_g_hr <- glmnet::glmnet(                               # fit the outcome model on training folds only
    x           = X_full_hr[idx_tr, , drop = FALSE],
    y           = y_hr[idx_tr],
    family      = "binomial",
    alpha       = alpha_dml_hr,
    lambda      = lambda_g_hr,
    standardize = TRUE
  )

  g_hat_hr[idx_va] <- predict(                              # predict baseline risks on the held-out fold
    fit_g_hr,
    newx = X_full_hr[idx_va, , drop = FALSE],
    s    = lambda_g_hr,
    type = "response"
  )[, 1]
}

eps_hr  <- 1e-4                                             # choose a small value to avoid extreme predicted probabilities
m_hat_hr <- pmin(pmax(m_hat_hr, eps_hr), 1 - eps_hr)        # clamp propensities away from zero and one
```

```r
g_hat_hr <- pmin(pmax(g_hat_hr, eps_hr), 1 - eps_hr)                    # clamp risks away from zero and one

## ---------------------------------------------------------------------
## 15.4 Orthogonalised regression: DML estimate for high-rate effect
## ---------------------------------------------------------------------

tilde_y_hr <- y_hr - g_hat_hr                                           # compute residualized outcome using cross-fitted g_hat_hr
tilde_d_hr <- d_hr - m_hat_hr                                           # compute residualized treatment using cross-fitted m_hat_hr

dml_hr_fit <- lm(tilde_y_hr ~ tilde_d_hr)                               # regress residualized outcome on residualized treatment

theta_dml_hr      <- unname(coef(dml_hr_fit)["tilde_d_hr"])             # store the DML effect estimate in probability units
theta_dml_hr_ci   <- confint(dml_hr_fit)["tilde_d_hr", ]                # store the ninety-five percent confidence interval
theta_dml_hr_se   <- summary(dml_hr_fit)$coefficients["tilde_d_hr", "Std. Error"] # store the standard error
theta_dml_hr_pval <- summary(dml_hr_fit)$coefficients["tilde_d_hr", "Pr(>|t|)"]   # store the p-value

cat("\n[High-rate DML] Estimated ATE of high interest rate on default:\n")   # print a report header
cat(sprintf("  theta_DML_highrate  = %.4f (%.2f percentage points)\n",       # print the estimate in probability units and percentage points
            theta_dml_hr, 100 * theta_dml_hr))
cat(sprintf("  95%% CI              = [%.4f, %.4f]\n",                       # print the confidence interval endpoints
            theta_dml_hr_ci[1], theta_dml_hr_ci[2]))
cat(sprintf("  Std. Error          = %.4f,  p-value = %.4g\n",              # print the standard error and p-value
            theta_dml_hr_se, theta_dml_hr_pval))

## ---------------------------------------------------------------------
## 15.5 Naive benchmark: raw difference in default rates (high minus low)
## ---------------------------------------------------------------------

mean_hr1 <- mean(y_hr[d_hr == 1])                                       # compute the default rate among high-rate loans
mean_hr0 <- mean(y_hr[d_hr == 0])                                       # compute the default rate among low-rate loans

theta_naive_hr <- mean_hr1 - mean_hr0                                   # compute the raw difference in means

se_naive_hr <- sqrt(                                                    # compute a two-sample standard error under independence
  var(y_hr[d_hr == 1]) / sum(d_hr == 1) +
    var(y_hr[d_hr == 0]) / sum(d_hr == 0)
)

ci_naive_hr <- theta_naive_hr + c(-1, 1) * 1.96 * se_naive_hr           # compute a ninety-five percent confidence interval

cat("\n[High-rate naive] Difference in default rates (high minus low):\n")   # print a report header
cat(sprintf("  theta_naive_highrate = %.4f (%.2f percentage points)\n",     # print the naive estimate in probability units and percentage points
```

```
            theta_naive_hr, 100 * theta_naive_hr))
cat(sprintf("  95%% CI              = [%.4f, %.4f]\n",                    # print the naive confidence interval endpoints
            ci_naive_hr[1], ci_naive_hr[2]))

## ---------------------------------------------------------------------
## 15.6 Compact comparison table for reporting
## ---------------------------------------------------------------------

dml_highrate_comparison <- dplyr::tibble(                                # build a report-ready comparison table
  estimator = c("Naive diff-in-means (high vs low rate)",
                "DML (cross-fit elastic net)"),
  theta     = c(theta_naive_hr, theta_dml_hr),
  ci_low    = c(ci_naive_hr[1], theta_dml_hr_ci[1]),
  ci_high   = c(ci_naive_hr[2], theta_dml_hr_ci[2])
)

cat("\n=== Effect of high interest rate on default: naive vs DML ===\n")     # print a header for the comparison table
print(dml_highrate_comparison)                                           # print the comparison table

## =====================================================================
## 16. DML Combined comparison table
## =====================================================================

dml_all_prob <- tibble::tibble(                                           # build one table that stacks both treatments and both estimators
  treatment = c(                                                          # treatment label used in the paper table
    "60-month term (vs 36-month)",
    "60-month term (vs 36-month)",
    "High interest (>= 75th pct)",
    "High interest (>= 75th pct)"
  ),
  estimator = c(                                                          # estimator label used in the paper table
    "Naive diff-in-means",
    "DML (elastic net, K-fold)",
    "Naive diff-in-means",
    "DML (elastic net, K-fold)"
  ),
  theta   = c(theta_naive,      theta_dml,      theta_naive_hr,      theta_dml_hr),       # effect estimate in probability units
  ci_low  = c(ci_naive[1],      theta_dml_ci[1], ci_naive_hr[1],      theta_dml_hr_ci[1]), # confidence interval lower bound in probability units
  ci_high = c(ci_naive[2],      theta_dml_ci[2], ci_naive_hr[2],      theta_dml_hr_ci[2])  # confidence interval upper bound in probability units
) %>%
  dplyr::mutate(
    theta_pct   = 100 * theta,                                            # convert the effect estimate to percentage points
    ci_low_pct  = 100 * ci_low,                                           # convert the lower confidence bound to percentage points
    ci_high_pct = 100 * ci_high,                                          # convert the upper confidence bound to percentage points
    effect_ci   = sprintf("%.2f [%.2f, %.2f]", theta_pct, ci_low_pct, ci_high_pct) # create a compact string for reporting effect and interval
  )

```

```r
cat("\n=== DML extension: naive vs DML for both treatments ===\n")            # print a marker to separate output blocks in the console
print(                                                                        # print a compact view for a quick check before exporting
  dml_all_prob %>%
    dplyr::select(treatment, estimator, effect_ci)
)

dml_all_tex <- dml_all_prob %>%                                               # construct the exact table to export to LaTeX
  dplyr::transmute(
    Treatment = treatment,                                                    # treatment label column used in the manuscript
    Estimator = estimator,                                                    # estimator label column used in the manuscript
    `Effect (pp) [95% CI]` = effect_ci                                        # compact effect and confidence interval in percentage points
  ) %>%
  as.data.frame()                                                             # convert to a base data frame for stargazer

stargazer::stargazer(                                                         # export the LaTeX table to file
  dml_all_tex,                                                                # table object to export
  type            = "latex",                                                  # output format
  summary         = FALSE,                                                    # suppress summary statistics
  rownames        = FALSE,                                                    # omit row names
  no.space        = TRUE,                                                     # use tighter LaTeX formatting
  digits          = 2,                                                        # set digits option even though effect_ci is a string
  table.placement = "!htbp",                                                  # set the LaTeX float placement preference
  title           = "Naive vs DML estimates for two treatments",              # table caption shown in the manuscript
  label           = "tab:dml_naive_vs_dml",                                    # LaTeX label for referencing
  out             = "table_dml_naive_vs_dml.tex"                               # output file name
)

## ================================================================
## End of file!
## ================================================================
```