Predictive Optimization of Phone Call Marketing

Zhaoyang Li, Jasper Ye, Ziyu Zhou, Chloe Pan

1 Executive Summary

This project focuses on enhancing the effectiveness and profitability of phone call-based marketing campaigns for a Portuguese banking institution through predictive modeling and optimization. The dataset, sourced from the UCI Machine Learning Repository, contains over 45,000 records of past marketing campaigns, including client demographics, contact history, and campaign outcomes.

Our work addresses two core objectives: (1) accurately predicting whether a client will subscribe to a term deposit, and (2) selecting an optimal subset of clients to recall under resource constraints to maximize expected net revenue. To achieve these goals, we implemented and compared three modeling approaches: Logistic Regression, XGBoost, and Deep Neural Networks (DNN).

The Logistic Regression model offers strong interpretability and decent performance, but struggles with low recall due to the imbalanced nature of the dataset. XGBoost provides a balanced trade-off between accuracy and interpretability, benefiting from advanced regularization and SHAP-based feature attributions. The Deep Neural Network achieved the highest predictive performance across all metrics, particularly excelling in recall and F1-score, although it introduces challenges in explainability.

To further support decision-making, we developed a binary integer optimization framework that leverages predicted probabilities, revenue estimates, and contact costs to select clients for recall. This optimization model aligns marketing decisions with business objectives by identifying clients with the highest expected profitability, subject to human resource constraints.

Implementation considerations such as class balancing, threshold tuning, real-time model integration, and ongoing performance monitoring are also addressed. The final solution delivers a comprehensive and scalable decision-support pipeline that improves marketing efficiency, supports ethical targeting, and enhances the return on investment for the bank's outreach campaigns.

2 Problem Statement

Based on data from direct marketing campaigns (phone calls) of a Portuguese banking institution, this project aims to achieve two objectives.

The first one is to predict each customer's likelihood of subscribing to a term deposit. By analyzing patterns in customer financial profiles and past campaign interactions using statistical and machine learning methods, the model helps identify and prioritize customers who are more likely to respond positively, enabling more targeted and effective outreach.

The second objective is maximizing the total expected revenue from term deposit subscriptions by selecting customers to recall under the same offer. Each customer is associated with a predicted subscription probability from the response model, an estimated revenue upon subscribing from another model trained on a separate dataset, and a predicted contact cost. Subject to a total contact limit within a period, the optimization model identifies a subset of customers who are more profitable to recall, thereby improving the overall efficiency and return of the campaign.

This problem is important in banking and finance since marketing is a crucial tool for financial institutions to engage and retain customers. Poorly timed or excessive phone calls could lead to unsubscribe requests and call rejections, reducing future engagement opportunities. An optimized outreach strategy builds trust,

improves customer experience, and encourages long-term relationships. Banks can also reduce marketing costs by optimizing marketing strategies.

3 Data Analyses

The database is provided by UCI Machine Learning Repository. The data is on direct marketing campaigns by a Portuguese bank that relied on phone calls to reach customers. It has a large sample size (45,211), ranging from 2008 to 2010. The variables of the data are listed below: `age`, `job`, `marital`, `education`, `contact` (contact communication type), `balance`, `default` (has credit in default), `housing` (has housing loan), `loan` (has personal loan), `month` (last contact month of year), `day` (last contact day of the month), `duration` (last contact duration), `campaign` (number of contacts in current campaign), `pdays` (days since last contact), `previous` (number of contacts before this campaign), `poutcome` (outcome of the previous campaign). Variables, such as job, marital and education, that contain privacy risks will be eliminated in the modeling.

3.1 Exploratory Data Analysis for Categorical Variables

We initiate the exploratory analysis by first examining the response variable `y`, which is a binary indicator of whether a client subscribed to a term deposit. As shown in Figure 1, approximately 11.7% of clients subscribed, while 88.3% did not. This substantial class imbalance suggests that special consideration is needed during model building.

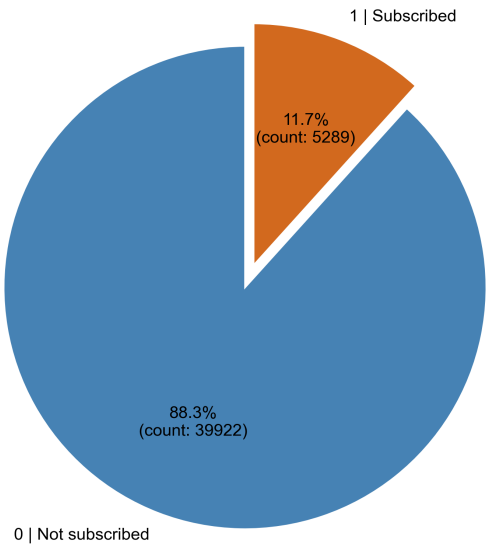


Figure 1: Proportions of subscription outcomes in the dataset.

Figure 2 presents the distributions of key categorical variables stratified by the target variable `y`. Several important patterns emerge from the analysis. For the `default` variable, individuals without a history of credit default have a noticeably higher subscription rate compared to those who have defaulted. Similarly, clients without a housing loan or a personal loan exhibit higher subscription proportions relative to those with such loans, suggesting that greater financial flexibility may be associated with a higher likelihood of subscribing.

For the `contact` method, clients who were contacted via cellular phones show significantly better subscription outcomes compared to those contacted via telephone or whose contact method is unknown. This may reflect greater responsiveness or convenience associated with mobile communication.

Seasonal trends are also observed in the `month` variable: subscription rates peak during March, September, October, and December, potentially aligning with specific marketing campaigns or seasonal financial decision patterns.

Finally, the `poutcome` variable (the outcome of the previous marketing campaign) reveals that clients with a previous successful outcome are far more likely to subscribe again, highlighting the importance of positive prior engagement in predicting future responses.

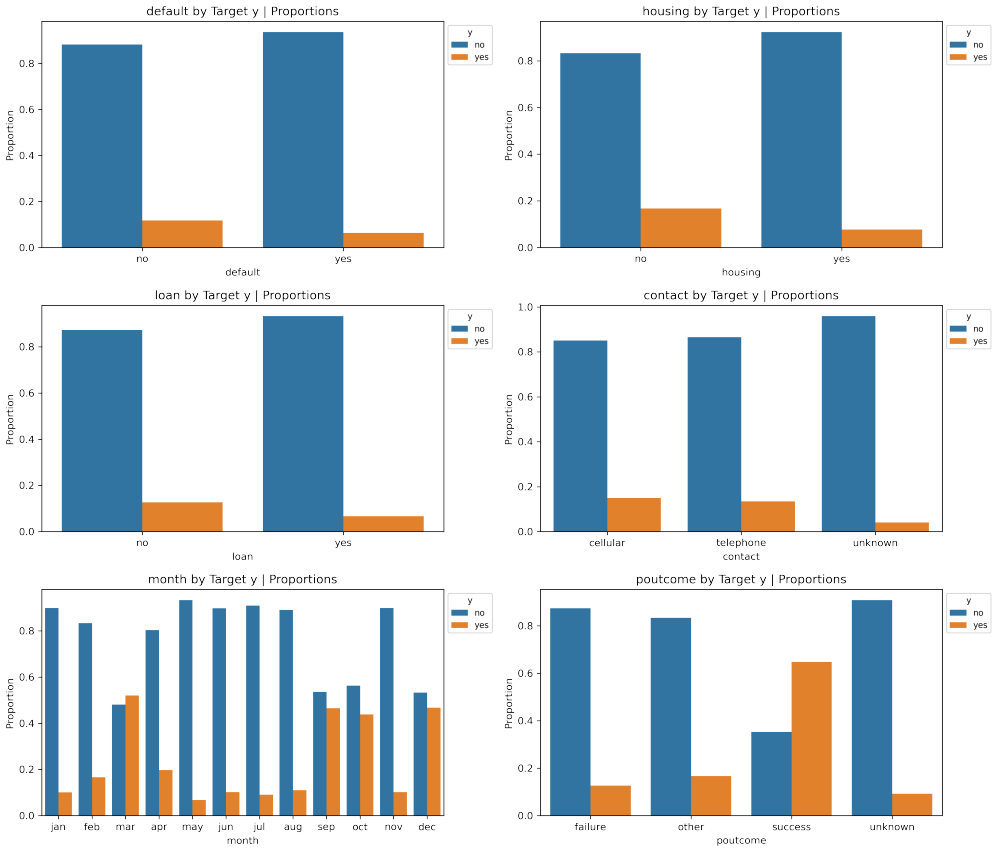


Figure 2: Distribution of categorical variables stratified by subscription status.

3.2 Exploratory Data Analysis for Numerical Variables

We next examine the distributions of the key numerical variables stratified by subscription status, as shown in Figure 3.

The `duration` variable, representing the last contact duration in seconds, is highly right-skewed, with most calls being short but a few lasting considerably longer. As expected, longer call durations are associated with a higher likelihood of subscription, suggesting that sustained client engagement significantly influences positive outcomes.

The `balance` variable, representing the client's account balance, also shows a heavily skewed distribution with many clients holding relatively small balances and a few outliers with exceptionally large balances. However, there is no strong visual distinction in balance between subscribers and non-subscribers, implying a weaker direct relationship.

The `age` variable exhibits a moderately right-skewed distribution, with most clients falling within middle-aged groups. Subscriptions appear slightly more common among older clients, hinting at a possible demographic influence on product interest.

The `campaign` variable, which captures the number of contacts performed during the campaign, is right-skewed with most clients contacted only a few times. Higher numbers of contacts do not appear to strongly improve subscription rates, suggesting diminishing returns after repeated attempts.

The `pdays` variable, recording the number of days since a client was last contacted, shows that a large proportion of values are clustered at -1, indicating no previous contact. For clients with valid previous contacts, there is a slight trend toward higher subscription rates with shorter `pdays` values, reinforcing the importance of timely follow-up.

Finally, the `day` variable, referring to the day of the month of the last contact, appears fairly uniformly distributed without strong seasonality or notable association with the subscription outcome.

3.3 Correlation Analysis

Figure 4 presents the correlation matrix among the key numerical variables, including the target variable `y`. Overall, the majority of the features exhibit very weak linear relationships with each other, as most correlation coefficients are close to zero.

The strongest observed correlation is between `pdays` and `previous` (correlation coefficient of 0.45), indicating that the number of days since the last contact is moderately associated with the number of previous contacts. This relationship is intuitive, as clients with more frequent past interactions may tend to have shorter gaps between contacts. Additionally, `duration` shows a moderate positive correlation of 0.39 with the response variable `y`, suggesting that longer conversations are moderately linked to a higher likelihood of subscription.

Other variable pairs, such as `age` with `balance` (0.10) and `campaign` with `day` (0.16), display very low correlations, implying that the numerical features are largely independent. This independence reduces concerns about multicollinearity and ensures that each feature can contribute uniquely to the predictive modeling process.

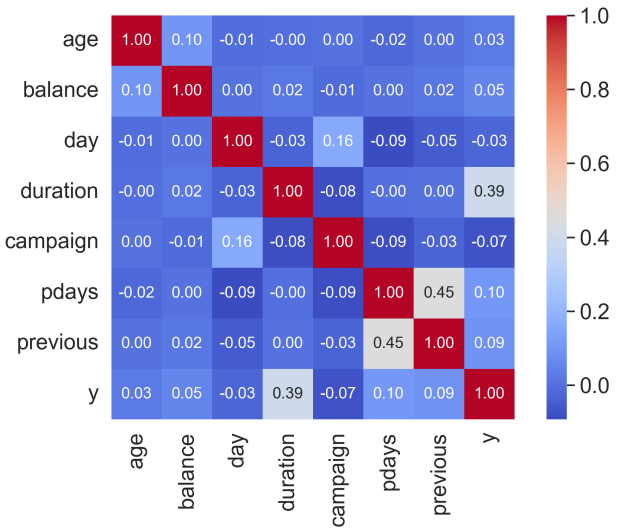
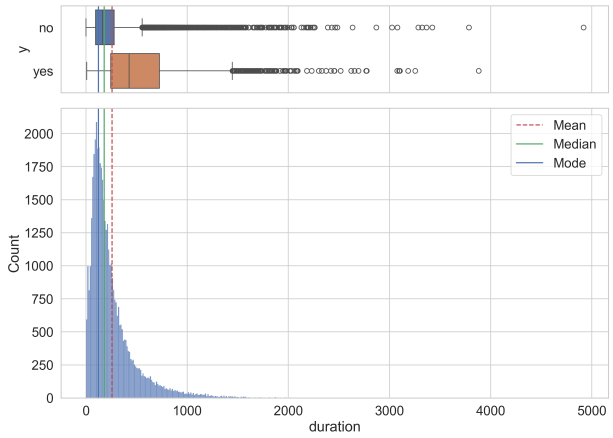
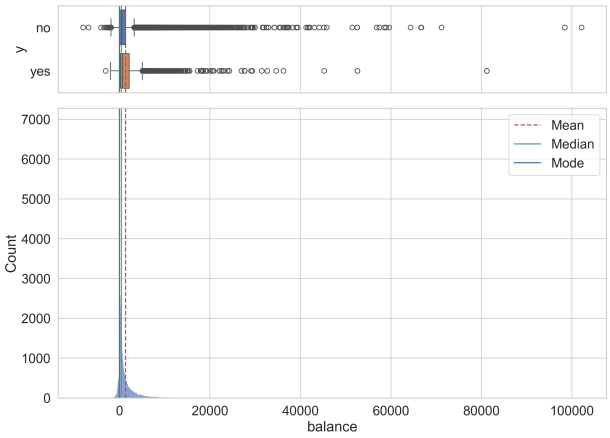


Figure 4: Correlation matrix among key numerical variables.

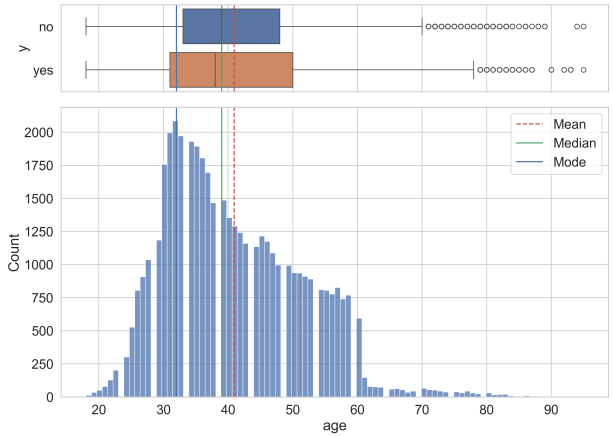
Based on the previous analysis, we proceed by dropping the `unknown` category from `contact` variable, as well as removing the `unknown` and `other` categories from the `poutcome` variable. In addition, we standardize the



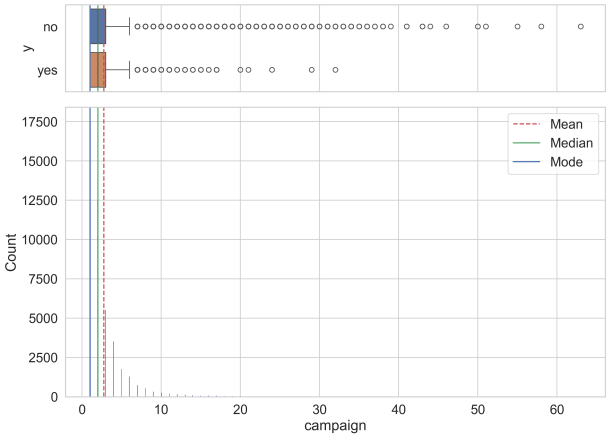
(a) Distribution of call duration.



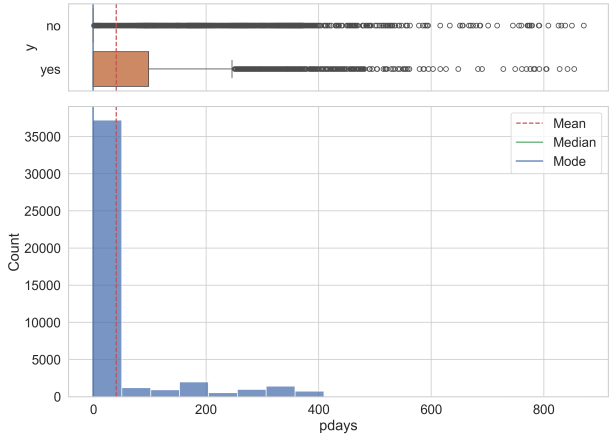
(b) Distribution of account balance.



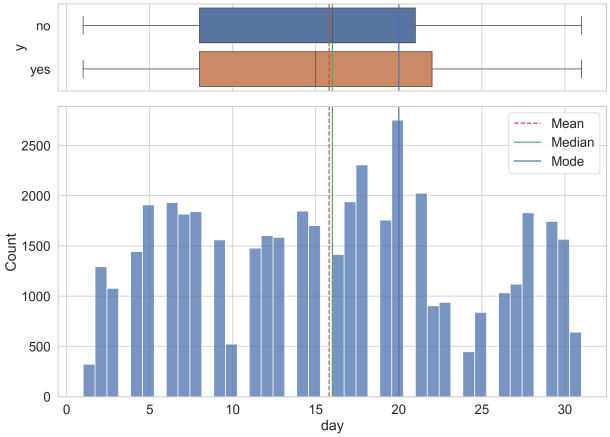
(c) Distribution of client age.



(d) Distribution of number of contacts in campaign.



(e) Distribution of days since last contact.



(f) Distribution of day of month of last contact.

Figure 3: Distribution and boxplots of key numerical variables stratified by subscription status.

duration, balance, campaign, and pdays features to ensure that all numerical variables are on a comparable scale for subsequent modeling.

4 Estimation Modeling

4.1 Logistic Regression

4.1.1 Model Methodology and Data Preprocessing

The objective of the estimation model is to predict the likelihood of a customer subscribing to a term deposit based on customer financial profiles and past campaign interactions. The outcome variable y represents whether a client has subscribed to a term deposit. Since the outcome variable y is binary, logistic regression is used for modeling. The general equation for logistic regression is as follows:

$$\log\left(\frac{p}{1-p}\right) = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \cdots + \beta_k X_k$$

Where: p is the Probability of the outcome (subscription = "yes"), $log(\frac{p}{1-p})$ is logit of the probability, β_0 is the intercept, $\beta_1, \beta_2, \ldots, \beta_k$ represent the change in the log-odds of the outcome for a one-unit increase in the predictor. The Odds Ratio will help interpret the results (Odds Ratio=e^{β}).

Logistic regression explicitly displays the relationships between the explanatory variables and the response variable, making the model interpretable. In this case, the logistic regression model helps to find the variables that could impact the probability of a client subscribing to a term deposit.

In the data preprocessing, variables like job, marital, education, and contact were dropped first due to client privacy protection. Since variables like default, housing, and loan were binary type and contained missing values, they must be converted into numeric types and drop the unknowns, which generated new variables (default_num, housing_num, and loan_num). Month was a categorical variable and needed to be converted into a numerical categorical variable, which created a new variable month_num. January(1) was set as a reference. For variable day, day 1 was set as a reference for the day of month. About 81.74% of pdays value was -1, which meaned 81.74% of the clients were not previously contacted. Since the main goal is to test whether or not a client was previously contacted would influence the probability of subscription, a new dummy variable contacted_before was created. If contacted_before = 1, it meaned the client was contacted in a prior campaign (pdays $\geq$ 0). If contacted_before = 0, the client was never contacted before (pdays = -1). poutcome was encoded in a new numeric variable poutcome_num. This encoding process assigned 1, 2, 3, 4 to failture, other, success, unknown respectively. Unknown(4) was set as a reference. Regarding the output variable y, since it was binary, a new dummy variable y_numeric was generated to help construct the model.

4.1.2 Model Training and Evaluation

To perform the backtest of the model, a random seed was set to ensure reproducibility, and the dataset was split into training and testing sets using a 70/30 split, stratified to maintain the balance of classes between training and validation. The model was a logistic regression model (binary classifier) with robust standard errors. The default cutoff threshold was set at 0.5.

The confusion matrix summarizes the performance of a classification model, comparing the predicted values with the true values. Based on the confusion matrix, we could calculate some key metrics to evaluate the performance of the model. In Table 2, the In-sample Validation results in an accuracy of 0.899 and an AUC-ROC score of 0.897, which indicates that the in-sample fits well.

Table 3 and Table 4 present the results of the out-of-sample evaluation on the held-out validation set. The accuracy is 0.9048 and the AUC-ROC score is 0.9061, showing strong predictive performance and discriminating power. The specificity is 0.9765, revealing that the model performs well in identifying the non-subscribers. And the area under the ROC curve is 0.8965, as shown in Figures 6.

However, both the in-sample and out-of-sample validation results reveal that the recall is low; recall is only 0.3373. The low recall suggests that the model fails to capture about 66% of clients who subscribed, shown by the high FN. Overall, the model is consistent and not overfitting, but it shows the inability to capture the actual subscribers. The reason for the low recall may be the class imbalance of the dataset, even after the stratification. The number of subscribers is overall rare compared to the large sample size. Besides, the threshold setting at 0.5 might be relatively high as it would prefer specificity over recall.

Table 1: In-Sample Confusion Matrix

	Predicted 0	**Predicted 1**	**Total**
Actual 0	27,255 (85.95%)	680 (2.14%)	27,935
Actual 1	2,524 (7.96%)	1,250 (3.94%)	3,774
Total	29,779	1,930	31,709

Table 2: In-sample Performance Metrics

Metric	Value
Accuracy	0.899
Recall (Sensitivity)	0.331
Specificity	0.976
Precision	0.648
F1 Score	0.438
AUC	0.897

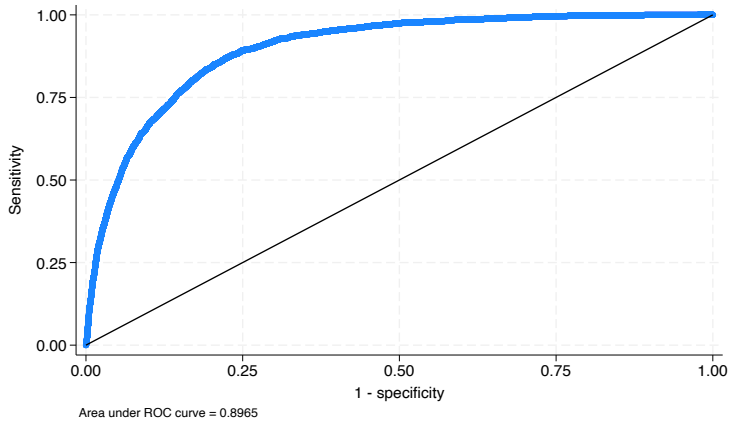


Figure 5: In-sample ROC Curve

Table 3: Out-of-sample Confusion Matrix

	Predicted 0	**Predicted 1**	**Total**
Actual 0	11,705 (86.69%)	282 (2.09%)	11,987
Actual 1	1,004 (7.44%)	511 (3.78%)	1,515
Total	12,709	793	13,502

	Accuracy	Precision	Recall	Specificity	F1-Score	AUC
Logit Model	.9047549	.6443884	.3372937	.9764745	.4428076	.9060501

Threshold = 0.5
Validation N = 13,502

Table 4: Out-of-sample Performance Matrix

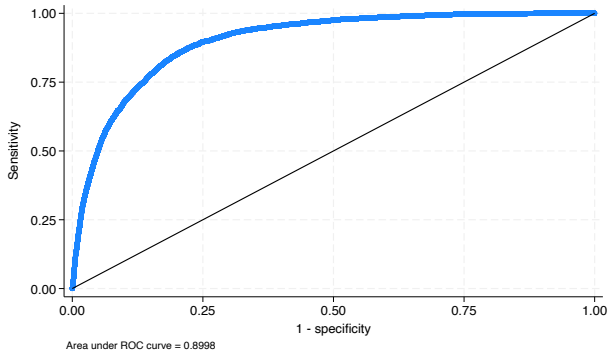


Figure 6: Out-of-sample ROC Curve

4.1.3 Model Interpretation

The model result is shown below in Table 6. To interpret the result, we need odds ratio to assist the analysis, which is shown in Table 7. The interpretation depends on whether the odds ratio is greater than, less than, or equal to 1.

The strongest positive drivers who have relatively high subscription odds are **seasoning effects**, **previous marketing campaign success**, and **duration**. As shown in Table 7, March (OR=17.2) has extremely high value of odds ratio. October and September have high OR values as well. They all have low p-values, indicating their statistical significance. The high impact on subscription might be due to seasoning reasons. March is normally tax return season in Europe, so clients might have more disposable income for investment and are more likely to subscribe a term deposit. September and October are post-summer periods. Clients are back from summer holidays and may want to reallocate their budgets. `poutcome_3` (previous marketing campaign success) and `duration` also has high OR, meaning it helps boost the probability of subscription.

The strongest negative drivers who have relatively low subscription odds are **loans**, **campaign**, and **housing loan**. As shown in Table 7, `loan_num` has OR of 0.61 and `housing_num` has OR of 0.42, indicating that clients who have a loan have about 39% lower odds to subscribe to a term deposit and housing loan holders have about 58% lower odds to subscribe. This is reasonable since clients who have loans or housing loans might experience financial constraints and they may want to pay their debts first. `campaign` means the

number of contacts performed during this campaign and for this client. Its OR has a value of 0.909. The negative effect of `campaign` on subscription might arise from diminishing returns. Clients might have fatigue after several contacts and further contact would create negative effects.

Based on the Odds Ratio analysis, the budgets for campaign and marketing could be reallocated, shifting more budgets on March, September and October to raise the subscription probability. And when contacting clients, the number of contacts could be controlled to avoid fatigue. As for the target clients, clients with loans and housing loans might not be first choices to contact with.

4.2 XGBoost

4.2.1 Model Methodology

XGBoost is a highly efficient and scalable machine learning algorithm based on the principle of gradient boosting, because it captures complex non-linear interactions between features, enabling it to model sophisticated relationships in the data. It is especially powerful for imbalanced datasets, as it offers a parameter called `scale_pos_weight` that increases the weight of the minority class during training, which is an important feature in applications such as marketing, where positive responses are rare. Additionally, XGBoost improves over traditional boosting methods by incorporating both L_1 (Lasso) and L_2 (Ridge) regularization terms into its objective function. This helps control model complexity, prevent overfitting, and enhance generalization performance.

Formally, the objective function at iteration t is given by:

$$\mathcal{L}^{(t)} = \sum_{i=1}^{n} l\left(y_i, \hat{y}_i^{(t-1)} + f_t(x_i)\right) + \Omega(f_t)$$

where l is a differentiable convex loss function (e.g., logistic loss for classification), $\hat{y}_i^{(t-1)}$ is the prediction from the previous iteration, and $f_t(x)$ represents the newly added tree at iteration t.

The regularization term $\Omega(f)$ is defined as:

$$\Omega(f) = \gamma T + \frac{1}{2}\lambda \sum_{j=1}^{T} w_j^2 + \alpha \sum_{j=1}^{T} |w_j|$$

where T is the number of leaves in the tree, w_j are the leaf weights, γ penalizes the number of leaves (controlling model complexity), λ controls the L_2 regularization strength, and α controls the L_1 regularization strength.

XGBoost uses a boosting framework, where trees are built sequentially. Each new tree is trained to correct the residual errors made by the previous ensemble, progressively reducing bias and improving prediction accuracy.

To efficiently optimize the regularized objective function, XGBoost employs a second-order Taylor series approximation of the loss function around the current predictions. Specifically, the loss function at iteration t is approximated as:

$$\mathcal{L}^{(t)} \approx \sum_{i=1}^{n} \left[g_i f_t(x_i) + \frac{1}{2} h_i f_t(x_i)^2 \right] + \Omega(f_t)$$

where the first and second derivatives of the loss function with respect to the previous prediction are:

$$g_i = \frac{\partial l(y_i, \hat{y}_i^{(t-1)})}{\partial \hat{y}_i^{(t-1)}} \quad \text{and} \quad h_i = \frac{\partial^2 l(y_i, \hat{y}_i^{(t-1)})}{\partial (\hat{y}_i^{(t-1)})^2}$$

Variables	Coefficient
age	-0.007***
	(0.000)
balance	0.000***
	(0.000)
default_num	-0.105
	(0.500)
housing_num	-0.857***
	(0.000)
loan_num	-0.495***
	(0.000)
month_num=1	0.000
	(.)
month_num=2	0.986***
	(0.000)
month_num=3	2.846***
	(0.000)
month_num=4	1.207***
	(0.000)
month_num=5	0.199
	(0.106)
month_num=6	0.536***
	(0.000)
month_num=7	0.404***
	(0.001)
month_num=8	0.532***
	(0.000)
month_num=9	1.990***
	(0.000)
month_num=10	2.034***
	(0.000)
month_num=11	0.305**
	(0.018)
month_num=12	1.876***
	(0.000)
Day of month (1–31)	0.004
	(0.134)
duration	0.004***
	(0.000)
campaign	-0.096***
	(0.000)
contacted_before=0	0.000
	(.)
contacted_before=1	0.830
	(0.382)
previous	0.010
	(0.314)
failure	-0.443
	(0.641)
other	-0.230
	(0.809)
success	1.862*
	(0.050)
unknown	0.000
	(.)
Constant	-3.428***
	(0.000)
Observations	45211

p-values in parentheses
* $p < 0.10$, ** $p < 0.05$, *** $p < 0.01$

Table 6: Logistic Regression Results

Variables	Odds Ratio
age	0.993***
	(0.002)
balance	1.000***
	(0.000)
default_num	0.900
	(0.141)
housing_num	0.424***
	(0.019)
loan_num	0.610***
	(0.036)
month_num=1	1.000
	(.)
month_num=2	2.680***
	(0.386)
month_num=3	17.224***
	(2.822)
month_num=4	3.342***
	(0.427)
month_num=5	1.220
	(0.150)
month_num=6	1.709***
	(0.224)
month_num=7	1.498**
	(0.186)
month_num=8	1.702***
	(0.216)
month_num=9	7.313***
	(1.188)
month_num=10	7.641***
	(1.181)
month_num=11	1.357*
	(0.175)
month_num=12	6.529***
	(1.480)
Day of month (1–31)	1.004
	(0.003)
duration	1.004***
	(0.000)
campaign	0.909***
	(0.010)
contacted_before=0	1.000
	(.)
contacted_before=1	2.294
	(2.179)
previous	1.010
	(0.010)
failure	0.642
	(0.610)
other	0.794
	(0.756)
success	6.439
	(6.126)
unknown	1.000
	(.)
Observations	45211

Exponentiated coefficients; Standard errors in parentheses
* $p < 0.05$, ** $p < 0.01$, *** $p < 0.001$

Table 7: Odds Ratio

This second-order approximation allows XGBoost to use both gradient and curvature information for more accurate updates, improving convergence speed compared to traditional first-order methods.

Given a tree structure $q(x)$ that maps each input x to a corresponding leaf j, and assigning a weight w_j to each leaf, the approximated objective function becomes:

$$\mathcal{L}^{(t)} \approx \sum_{j=1}^{T} \left[G_j w_j + \frac{1}{2} H_j w_j^2 \right] + \gamma T + \frac{1}{2} \lambda \sum_{j=1}^{T} w_j^2 + \alpha \sum_{j=1}^{T} |w_j|$$

where $G_j = \sum_{i \in I_j} g_i$ and $H_j = \sum_{i \in I_j} h_i$ are the aggregated first- and second-order derivatives for all samples assigned to leaf j, and I_j denotes the set of instances in leaf j.

The optimal weight w_j^* for each leaf can be derived analytically as:

$$w_j^* = -\frac{G_j + \alpha \cdot \text{sign}(G_j)}{H_j + \lambda}$$

Substituting the optimal weights back into the objective yields the minimized loss value:

$$\mathcal{L}_{\text{optimal}}^{(t)} = -\frac{1}{2} \sum_{j=1}^{T} \frac{(G_j + \alpha \cdot \text{sign}(G_j))^2}{H_j + \lambda} + \gamma T$$

During training, XGBoost greedily grows trees by selecting the split that maximizes the reduction in this regularized objective, a quantity known as the *gain*. The gain from a potential split is calculated as:

$$\text{Gain} = \frac{1}{2} \left(\frac{G_L^2}{H_L + \lambda} + \frac{G_R^2}{H_R + \lambda} - \frac{(G_L + G_R)^2}{H_L + H_R + \lambda} \right) - \gamma$$

where G_L and H_L are the sum of gradients and Hessians for the left child, and G_R and H_R are those for the right child after a split. A higher gain indicates a greater improvement in the objective function, making the split more desirable. By systematically maximizing the gain at each possible split, XGBoost builds trees that balance data fitting with model complexity, leading to superior predictive performance.

Additionally, it implements various system-level optimizations, including sparsity-aware split finding and histogram-based approximate algorithms, making it highly efficient for large-scale and sparse datasets.

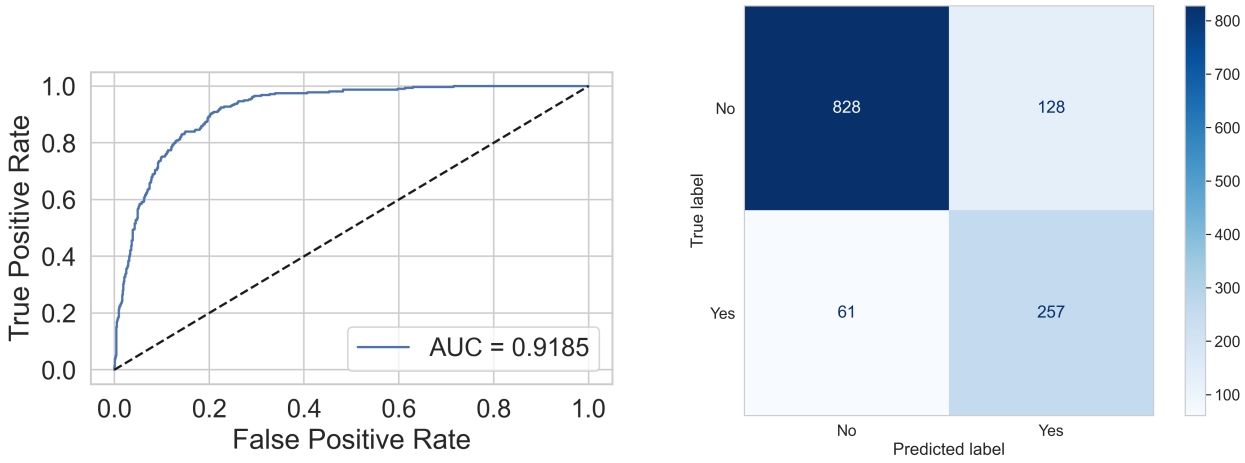
4.2.2 Model Training and Evaluation

After data preprocessing, the feature matrix X was constructed by dropping the target variable y from the dataset, while y was defined as the binary target variable. The dataset was then split into training and testing sets using an 80/20 split, stratified by the target to maintain class proportions. Due to class imbalance, the `scale_pos_weight` parameter was computed as the ratio of negative to positive instances in the training set, resulting in a value of approximately 3.01.

To optimize the model, a grid search with 5-fold cross-validation was performed over a defined hyperparameter grid. The base model was an XGBoost classifier (`XGBClassifier`) using the `hist` tree method for faster histogram-based splitting and `logloss` as the evaluation metric. The best model was selected based on the highest cross-validated accuracy.

The training performance of the best model yielded an accuracy of 0.8608 and an AUC-ROC score of 0.9276, indicating good in-sample fit. After training, the model was evaluated on the held-out test set to assess generalization performance. XGBoost achieved an accuracy of 0.8516 and an AUC-ROC of 0.9185 on the test set (Figures 7a), demonstrating strong overall predictive performance. The confusion matrix, shown in Figure 7b, indicates that out of 956 actual non-subscribers, 828 were correctly classified, while 128 were incorrectly predicted as subscribers. Among 318 actual subscribers, 257 were correctly identified, whereas 61

were missed by the model. Results from Table 8 show that precision for the positive class (subscribers) was 0.67, with a recall of 0.81 and an F1-score of 0.73. These results suggest that the model is highly effective at identifying actual subscribers and maintains strong performance in detecting non-subscribers, highlighting overall robustness. However, there may still be room for minor threshold adjustments or further model calibration to optimize the trade-off between precision and recall depending on specific business priorities.



(a) ROC curve on the test set.

(b) Confusion matrix on the test set.

Figure 7: Performance evaluation of the XGBoost model: (a) ROC curve and (b) confusion matrix.

Class	Precision	Recall	F1-score	Support
0 (No subscription)	0.93	0.87	0.90	956
1 (Subscription)	0.67	0.81	0.73	318
Accuracy			0.85	1274
Macro avg	0.80	0.84	0.81	1274
Weighted avg	0.87	0.85	0.86	1274

Table 8: Classification report for the XGBoost model on the test set.

4.2.3 Model Interpretation with SHAP

After training the XGBoost model, we applied SHapley Additive exPlanations (SHAP) to interpret how each feature contributed to the final predictions. Understanding feature attributions is crucial for model transparency, particularly in applications like marketing and finance where decision justification is essential.

SHAP is a unified framework based on cooperative game theory that provides consistent and locally accurate attributions for any machine learning model. It assigns each feature a contribution value by viewing the prediction task as a collaborative game where features cooperate to form the output. Specifically, for a model f and an input instance x, the SHAP value ϕ_j for feature j is defined as:

$$\phi_j(f,x) = \sum_{S \subseteq F \setminus \{j\}} \frac{|S|!(|F|-|S|-1)!}{|F|!} \left(f_{S\cup\{j\}}(x) - f_S(x)\right)$$

where F denotes the set of all features, S is a subset excluding j, $f_S(x)$ is the model prediction using only features in S, and $f_{S\cup\{j\}}(x)$ is the prediction after adding feature j. The combinatorial weighting ensures fair attribution across all possible coalitions of features.

Importantly, SHAP values satisfy three key properties:

- **Local Accuracy**: The sum of all SHAP values plus a base value exactly matches the model output.
- **Consistency**: If a model changes to increase the marginal contribution of a feature, its SHAP value does not decrease.
- **Missingness**: Features that have no effect on the output receive a SHAP value of zero.

For tree-based models such as XGBoost, SHAP leverages the efficient TreeSHAP algorithm, which allows exact computation of SHAP values in polynomial time without needing to enumerate all $2^{|F|}$ feature subsets. This makes SHAP practically feasible for large datasets and complex tree ensembles.

In our analysis, we used the `shap.Explainer` function to calculate SHAP values for the test set predictions. To visualize the results, we generated two types of plots: a bar plot showing the mean absolute SHAP values across all features, and a beeswarm plot displaying the distribution of SHAP values across individual samples.

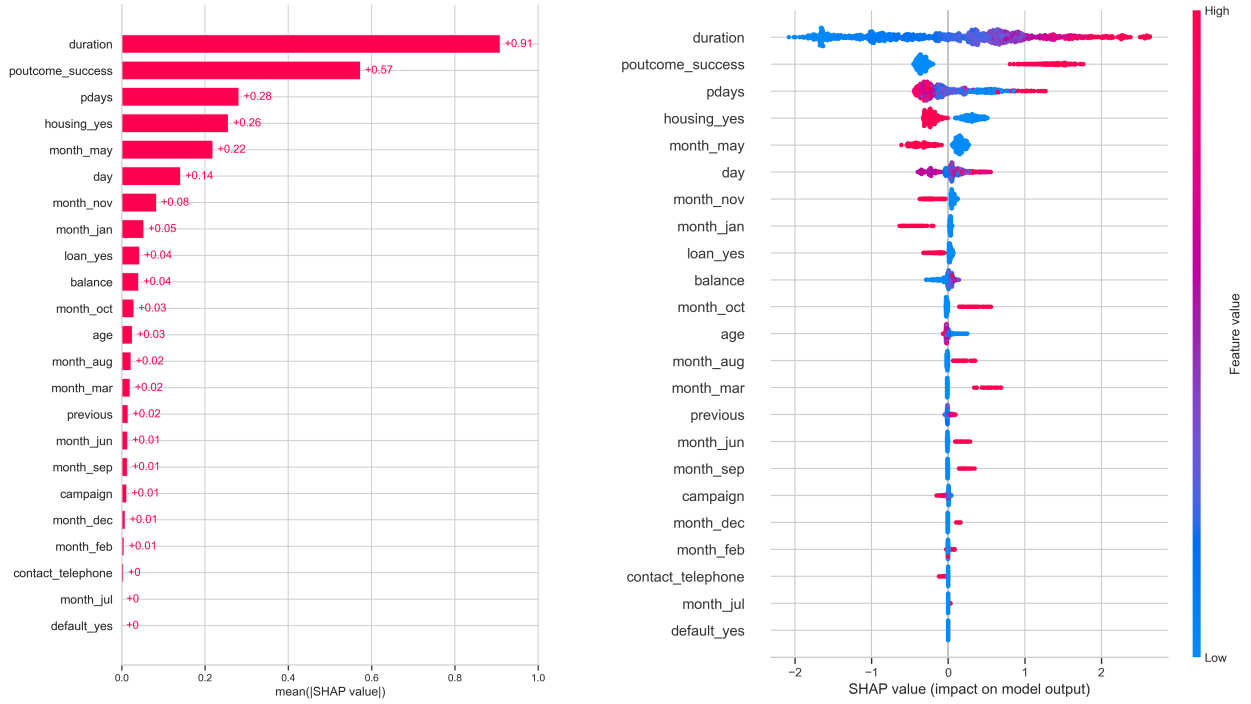
The SHAP interpretation results in Figure 8a indicate that **duration** is by far the most influential feature, exhibiting the largest mean absolute SHAP value and thus the greatest overall impact on model predictions. Longer call durations significantly increase the likelihood of subscription, likely reflecting stronger client engagement or interest, while short conversations may indicate early disinterest.

The second most important feature is **poutcome_success**, representing success in a previous marketing campaign. Clients with prior positive outcomes show a substantially higher predicted probability of subscribing again, highlighting the lasting effect of successful historical engagement. Following this, **pdays** (number of days since last contact) and **housing_yes** (presence of a housing loan) also exhibit strong contributions. More recent contacts (lower `pdays`) are associated with higher subscription likelihood, while clients without a housing loan are more likely to subscribe, suggesting greater financial flexibility as a positive factor.

Seasonal trends are also apparent: **month_may**, **month_nov**, and **month_jan** show noticeable influence. Clients contacted during May, November, and January tend to have slightly lower subscription probabilities, possibly due to external seasonal factors. In Portugal, May includes several public holidays such as Labor Day and municipal holidays, which may disrupt client availability and attention. November coincides with the end of the fiscal year for many businesses, potentially causing financial caution among clients. Similarly, January often involves the aftermath of holiday spending and financial planning for the new year, reducing client willingness to commit to new financial products.

By contrast, contacts made in months such as August, October, and March tend to capture relatively more favorable responses. August aligns with summer holidays in Portugal when individuals may have more leisure time and be more receptive to engagement, while October and March fall outside major holiday periods, representing more stable financial planning windows before year-end or the annual tax season. These seasonal patterns suggest that optimizing the timing of marketing campaigns to avoid holiday congestion and leverage quieter financial periods could significantly improve subscription outcomes.

Figure 8b further illustrates feature impacts at an individual level. Higher `duration` values consistently push predictions toward subscription, whereas higher `pdays` values (indicating longer periods without contact) push predictions toward non-subscription. Similarly, positive prior campaign outcomes (`poutcome_success`=1) increase the likelihood of a positive outcome.



(a) Bar plot of mean absolute SHAP values.

(b) SHAP beeswarm plot of feature impact distributions.

Figure 8: SHAP interpretation plots for the XGBoost model.

To further investigate the detailed relationship between features and their SHAP values, as well as potential threshold effects, we present SHAP dependence plots for the key variables influencing subscription probability.

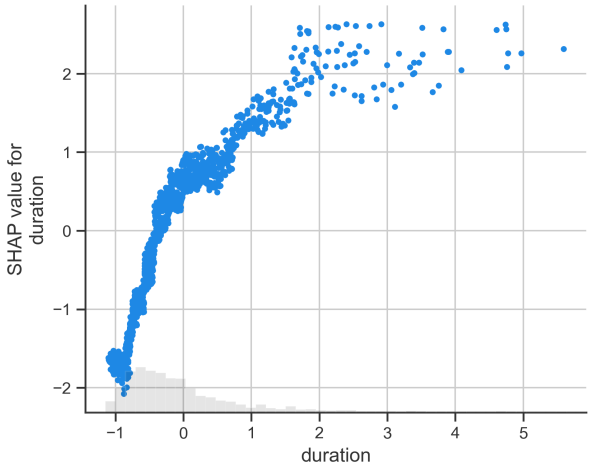
Figure 9a illustrates a strong positive relationship between `duration` and its SHAP value, indicating that longer call durations substantially increase the likelihood of subscription. Initially, even modest increases in call length lead to significant gains; however, beyond a certain threshold, the curve flattens, suggesting diminishing marginal returns. This threshold effect highlights that while sustained engagement is beneficial, excessively long conversations provide limited additional benefit.

Figure 9b depicts a more complex relationship between `pdays` (days since last contact) and its SHAP value. When `pdays` is close to -1 (indicating no prior contact), the SHAP impact is minimal. For nonnegative `pdays` values, recent contacts (low `pdays`) correspond to SHAP values clustered around zero, whereas longer intervals between contacts are associated with greater variability and increasingly positive SHAP values. This suggests that while recent engagement is important, reactivating long-dormant clients can also enhance subscription probability if approached appropriately.

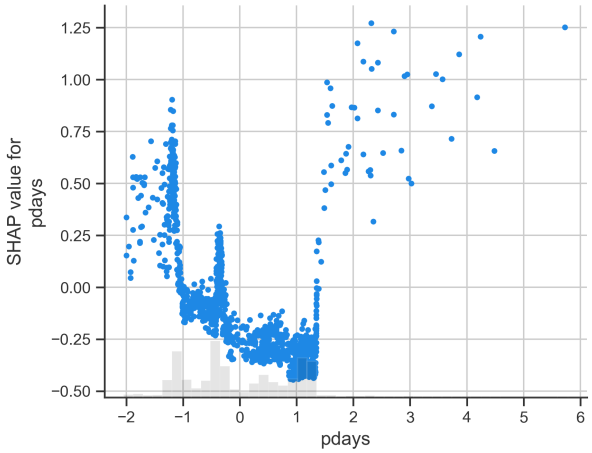
Figure 9c shows the SHAP dependence plot for `day`, representing the day of the month of client contact. A distinct nonlinear pattern emerges: early-month (days 1–15) and late-month (days 21–31) contacts are associated with higher subscription probabilities, whereas mid-month (days 16–20) contacts show a sharp decline in responsiveness. This trend likely reflects monthly financial cycles in Portugal, where salaries are typically disbursed at the end of the month. Clients contacted shortly after payday may feel more financially secure and receptive to savings offers, while mid-month budget constraints may reduce willingness to commit.

Figure 9d presents the SHAP dependence plot for `balance`. Clients with negative or very low balances exhibit slightly negative SHAP values, reflecting lower subscription likelihood. As `balance` increases toward zero, SHAP values improve sharply, suggesting that achieving a positive balance significantly boosts the probability of subscription. Beyond zero, however, additional increases in balance yield relatively stable SHAP values, indicating a saturation effect where further wealth accumulation has little marginal influence.

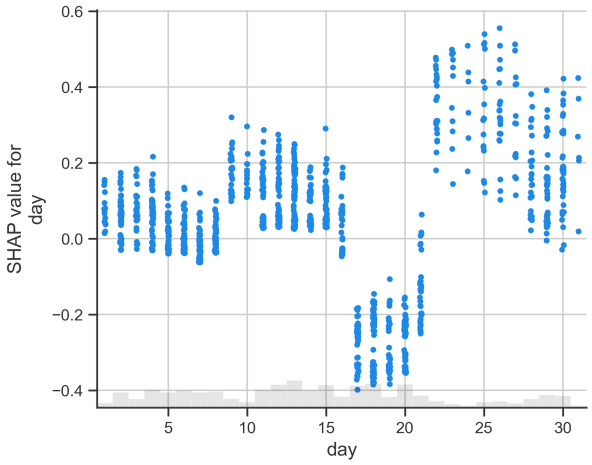
Overall, the SHAP analysis highlights that communication quality (particularly call duration), client history (such as prior campaign success), recent engagement (e.g., low `pdays`), and financial readiness (e.g., positive account balance) are critical drivers of subscription decisions. Additionally, the timing of interactions, such as contacting clients early or late in the month, emerges as an important factor, likely reflecting behavioral and financial cycles. Seasonal effects further reveal that subscription rates are lower around periods of financial strain and holidays (May, November, January), while responses are more favorable during months of greater financial stability and leisure (August, October, March). These findings suggest that marketing strategies could be significantly enhanced by fostering longer and more meaningful client interactions, maintaining frequent and well-timed outreach, and carefully aligning campaign timing with periods of heightened client receptiveness and financial well-being.



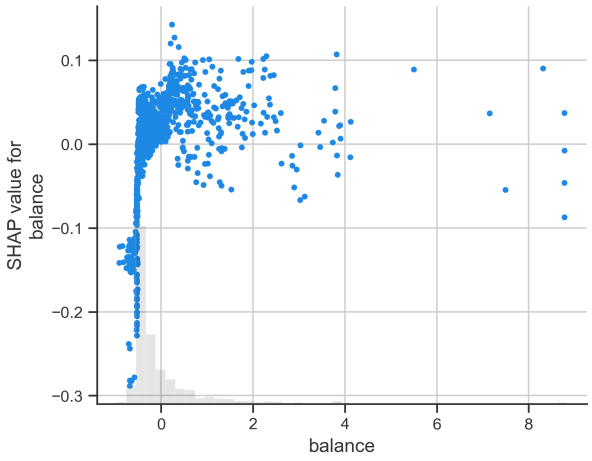
(a) SHAP dependence plot for `duration`.



(b) SHAP dependence plot for `pdays`.



(c) SHAP dependence plot for `day`.



(d) SHAP dependence plot for `balance`.

Figure 9: SHAP dependence plots for key features in the XGBoost model.

4.3 Deep Learning

4.3.1 Neural Networks and Deep Neural Networks

A neural network is a computational model inspired by the structure and function of the human brain. It consists of layers of interconnected units called neurons, where each neuron receives inputs, applies a transformation (typically a weighted sum followed by a non-linear activation), and passes the result to the

next layer. Neural networks are particularly powerful for modeling complex, non-linear relationships in data that traditional linear models cannot easily capture.

A deep neural network (DNN) specifically refers to a neural network that contains multiple hidden layers between the input and output layers. The additional layers allow the network to learn hierarchical feature representations: lower layers capture simple patterns, while deeper layers model more abstract and complex interactions. This hierarchical feature learning makes DNNs especially suitable for tasks with high-dimensional and complex feature spaces, such as our marketing subscription prediction problem.

4.3.2 Relevance to Our Problem and Dataset-Specific Modifications

The subscription prediction task involves subtle, non-linear relationships between customer characteristics, campaign interactions, and subscription behavior. For instance, the likelihood of subscription may depend not only on simple features like age or balance, but also on complex interactions among variables such as contact method, prior campaign success, and timing. Deep neural networks are well-suited to uncover these layered patterns that simpler models might miss.

However, several challenges needed to be addressed to adapt deep learning effectively to our specific dataset:

- **Class Weighting:** The original dataset exhibited severe class imbalance, with only about 11.7% of clients subscribing. To mitigate bias toward the majority class, we implemented class weighting in the binary cross-entropy loss function, penalizing errors on the minority (positive) class more heavily. This encouraged the model to correctly identify more positive cases without simply defaulting to predicting the majority class.
- **Resampling:** In addition to class weighting, we performed random oversampling of the minority class. By replicating positive examples until achieving a balanced dataset, we ensured that the model encountered sufficient instances of both classes during training, improving its ability to generalize to positive outcomes.
- **Threshold Tuning:** Neural network classifiers typically apply a probability threshold of 0.5 to decide between classes. However, in imbalanced problems, this default may not yield the best performance. We systematically searched across multiple thresholds and selected the one that maximized the F1-score, balancing precision and recall for the positive class. The optimal threshold (0.48) was then used during evaluation.

Through these targeted modifications — class weighting, resampling, and threshold tuning — the deep neural network was tailored to overcome the unique challenges of the bank marketing dataset and deliver strong predictive performance.

4.3.3 Modeling Assumptions

Applying a deep neural network to the subscription prediction task entails several key assumptions:

- **Nonlinearity of Relationships:** We assume that the relationship between customer features (e.g., age, balance, contact timing) and subscription behavior is inherently non-linear and complex. This justifies the use of a deep model over simpler linear classifiers.
- **Feature Interaction:** The model is expected to benefit from hierarchical feature learning, capturing interactions between variables that may not be evident through manual engineering or shallow models.
- **Sufficient Informative Signal:** Despite class imbalance, we assume the dataset contains enough discriminative patterns for the model to generalize well after training.
- **Representative Data:** The training data, though limited to campaigns conducted between 2008 and 2010, is assumed to be reasonably representative of similar future marketing scenarios.

- **Effective Preprocessing and Resampling:** One-hot encoding, standardization, and random oversampling are assumed to preserve the underlying data distributions and help mitigate imbalance without introducing bias or overfitting.
- **Trainability and Stability:** The selected model architecture, including use of dropout and early stopping, is assumed to be sufficient for effective training and generalization without requiring excessive tuning.
- **Decision Threshold Optimization:** The use of a tuned probability threshold (0.48) instead of the default 0.5 is based on the assumption that optimizing for F1-score leads to better decision boundaries under class imbalance.

4.3.4 Model Training and Evaluation

The dataset was first split into training and testing sets using an 80/20 stratified split to maintain the class distribution. After balancing the training set through oversampling and applying appropriate preprocessing (one-hot encoding for categorical variables and standard scaling for numerical variables), the deep neural network was trained using the Adam optimizer with a learning rate of 0.001 for 50 epochs. Dropout regularization was employed after each hidden layer to prevent overfitting, and early stopping was applied to halt training if validation loss did not improve, further enhancing model generalization.

Unlike traditional binary classification models that use a fixed 0.5 threshold, we performed threshold tuning after training. By evaluating different probability thresholds, we selected a threshold of 0.48 that maximized the F1-score, ensuring a balanced performance between precision and recall for the positive class.

The final deep learning model achieved the following results on the held-out test set:

- Accuracy: 92.2%
- Precision (Positive Class): 88%
- Recall (Positive Class): 98%
- F1-Score (Positive Class): 92.4%
- ROC-AUC Score: 0.9593

Comparison with Previous Models

Model	Accuracy	Positive Class Precision	Positive Class Recall	F1-Score	ROC-AUC
Logistic Regression	90.5%	64.4%	33.7%	44.3%	0.9061
XGBoost	85.2%	67%	81%	73%	0.9185
Deep Neural Network	92.2%	88%	98%	92.4%	0.9593

Compared to logistic regression, the deep learning model dramatically improved recall (from 33.7% to 98%) and F1-score (from 44.3% to 92.4%), highlighting its ability to capture complex patterns that simple linear models miss.

Compared to XGBoost, the deep learning model achieved higher recall and F1-score, while also improving overall accuracy and ROC-AUC. While XGBoost provided strong results with sophisticated tree ensembles, the deep neural network was able to further enhance performance, particularly for the minority class (subscribers).

4.3.5 Limitations and Practical Considerations

Despite the strong predictive performance of deep neural networks, their application in real-world financial and marketing settings is often constrained by a key limitation: lack of interpretability. Unlike logistic regression, where coefficients directly correspond to changes in log-odds, or XGBoost, which can be explained

using SHAP values, DNNs function as "black-box" models. Their complex architecture, consisting of multiple non-linear transformations across layers, makes it extremely difficult to trace how individual input features influence the final output.

This lack of transparency poses challenges in contexts where regulatory compliance, ethical responsibility, and stakeholder trust are critical. For example, financial institutions are often required to justify decisions—especially when offering or denying products—to customers or regulators. A model that cannot clearly articulate why a particular customer was predicted as a likely subscriber may face barriers to deployment, even if it performs exceptionally well.

Therefore, while the deep learning model in this study demonstrates robust performance and effectively captures complex patterns in the data, its practical adoption would require additional tools for model explainability, such as LIME or SHAP applied post hoc, or the development of more inherently interpretable architectures. In many production environments, this tradeoff between accuracy and interpretability must be carefully evaluated based on the specific use case and compliance requirements.

4.3.6 Model Interpretability with SHAP

While deep neural networks offer strong predictive power, they are often criticized for their lack of transparency. To enhance the interpretability of our deep learning model, we applied Permutation Feature Importances, a model-agnostic method that assigns an importance value to each feature for individual predictions.

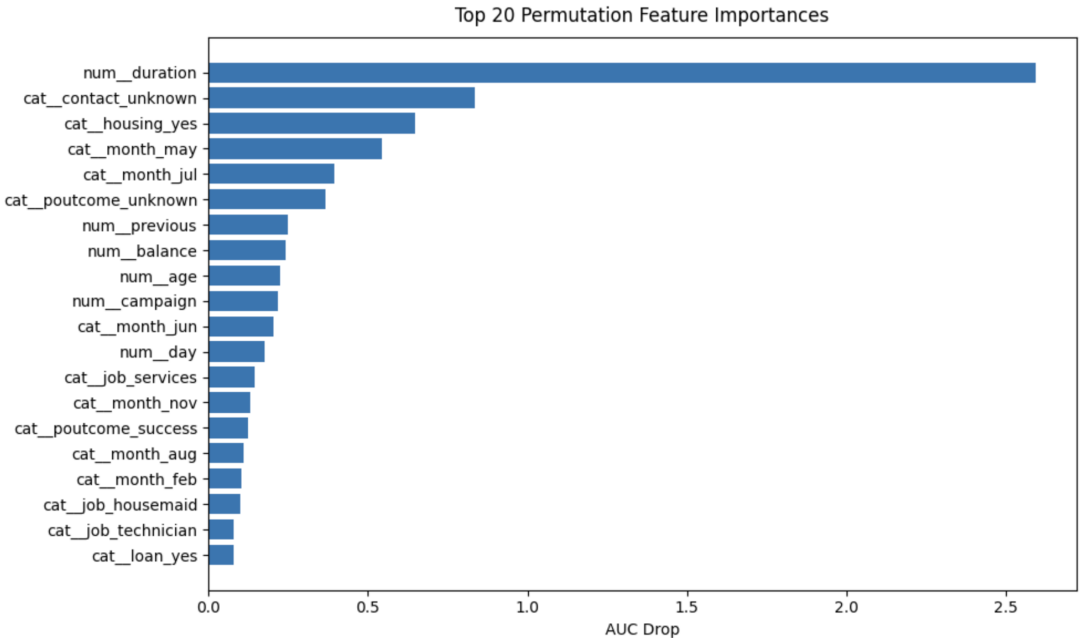


Figure 10: Top 20 Feature Importances as Measured by SHAP

Figure 10 shows that `duration` is the most influential feature by a substantial margin, indicating that longer phone calls are strongly associated with higher likelihoods of term deposit subscription. Categorical features such as contact method (e.g., unknown vs. cellular), housing loan status, and specific months (e.g., May, July) also demonstrate considerable predictive importance. These results align with domain intuition and offer partial transparency into the model's internal decision-making process.

Despite these insights, it is important to note that ermutation Feature Importances offer a post hoc explanation and do not fundamentally change the "black-box" nature of deep neural networks. For practical deployment, further work on model transparency or use of inherently interpretable models may be warranted.

5 Decision Optimization

Beforehand, we used statistical and machine learning models to estimate the likelihood that each customer would subscribe to a term deposit. However, the ultimate objective of bank marketing is not just to improve the subscription rate, but to generate the highest possible revenue at the lowest possible cost. In reality, even if two customers both choose to subscribe, the revenue they generate and the cost of reaching them can differ substantially. To better align with business goals, we develop a decision optimization framework that focuses on selecting the most cost-effective subset of previously contacted customers for recall, with the aim of maximizing net expected revenue.

5.1 Problem Formulation

The objective is to determine which previously contacted customers should be recalled in order to maximize the total net expected revenue. We formulate this as a binary optimization problem:

$$\begin{aligned} \text{Maximize} \quad & \sum_{i=1}^{n} [(r_i - c_i) \cdot p_i] \cdot x_i \\ \text{subject to} \quad & \sum_{i=1}^{n} x_i \leq H \\ & x_i \in \{0, 1\}, \quad \forall i = 1, 2, \ldots, n \end{aligned}$$

Where:

- r_i: Estimated revenue from customer i, based on their age, account balance, and personal and housing loan status.
- p_i: Predicted probability that customer i will subscribe to the term deposit.
- c_i: Estimated cost to recall customer i, modeled as a function of last contact duration, previous campaign outcome, days since last contact, etc.
- H: Human resource constraint, representing the maximum number of customers that can be contacted during one period.

5.2 Assumptions and Limitations

To simplify the modeling process, we assume that all revenue during the period is generated exclusively from term deposit subscriptions in the campaign. The effect of interest rate fluctuations on revenue is considered minimal and is therefore omitted from the objective function. The cost of recalling a customer is estimated based on observable features from previous campaigns, such as call duration, outcome of prior contact, and days since the last interaction. Since these features are among the most important in the machine learning model, the cost can be reasonably approximated using the predicted subscription probability p_i. Beyond basic resource constraints, several real-world factors may further limit customer selection, including behavioral risk, early withdrawal risk, and liquidity risk, all of which can affect the reliability and actual realization of expected returns.

5.3 Optimization Methodology

5.3.1 Revenue Estimation

To support revenue estimation, we introduce an additional dataset from the case study in Building Better Models with JMP® Pro (Grayson, Gardner & Stephens, SAS Press, 2015). The dataset is based on real banking operations but has been anonymized and adapted for instructional use. It includes key customer-level features such as total account balance, age, and indicators of personal loan and mortgage activity. The

prediction target, `Rev_Total`, represents the total revenue generated by each customer over a six-month period.

The dataset offers a moderate sample size of 7,420. While it includes rich features related to customer financial profiles, its overlap with the campaign result dataset is limited—only four features are shared between the two. In addition, the dataset has been preprocessed for teaching purposes, which may limit its generalizability. Since the original business context has been abstracted away, we assume that all revenue generated during the six-month period comes from term deposit subscriptions in the campaign.

To ensure that the external revenue dataset used for revenue estimation is representative of the original campaign dataset, we assess whether the two samples can reasonably be considered as drawn from the same population. We focus on the four features that overlap between the campaign dataset and the revenue dataset. Among these, `AGE` and `age`, as well as `Bal_Total` and `balance`, are numerical variables, while `LOAN` and `loan`, and `MORT` and `housing` are binary indicators.

For numerical features, we standardized the values to ensure consistency with the revenue estimation model and to place the features on a common scale for visualization and comparison. As shown in Figure 11, the standardized features do not follow normal distributions. Therefore, we use the Mann–Whitney U test at a significance level of $\alpha = 0.05$, which is appropriate for comparing samples with non-normal distributions. Since the Mann–Whitney U test is a nonparametric method that relies on the relative ranks of values rather than their absolute magnitudes, the standardization does not affect the validity of the test.

Given the test's sensitivity to large sample sizes, we implement a repeated subsampling approach: drawing samples of size 50 and running 1,000 trials to compute the average p-value. The resulting average p-values for standardized age and balance are 0.4953 and 0.0647, respectively, both above the significance threshold, suggesting no meaningful difference between the two distributions.

For binary features, we apply the Chi-square test at a significance level of $\alpha = 0.05$ to evaluate independence between two datasets, again using repeated subsampling with samples of size 50 and 1,000 trials to compute average p-value. The average p-values for personal loan and housing loan indicators are 0.4555 and 0.0725, respectively, indicating no statistically significant differences. Together, these results support the conclusion that the external revenue dataset is representative of the original campaign dataset.

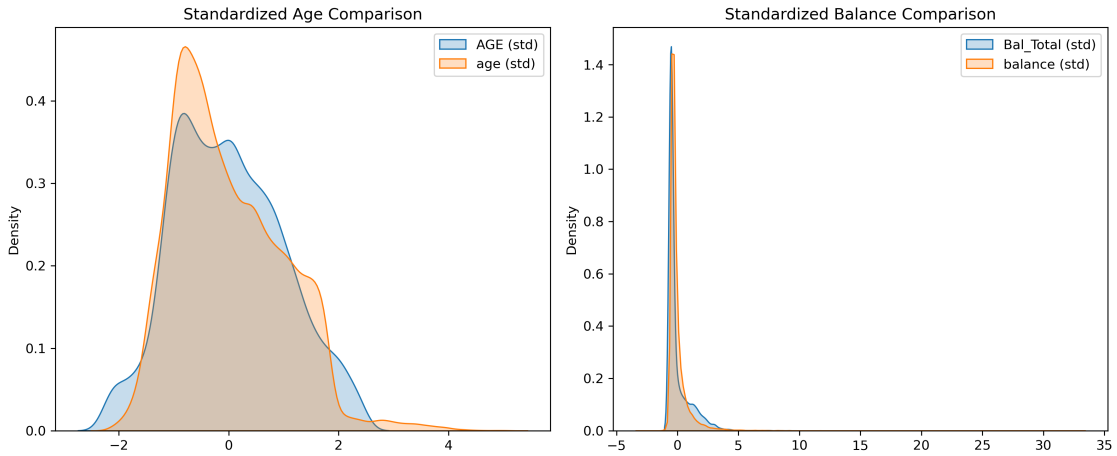


Figure 11: Distribution Comparison of Age and Account Balance

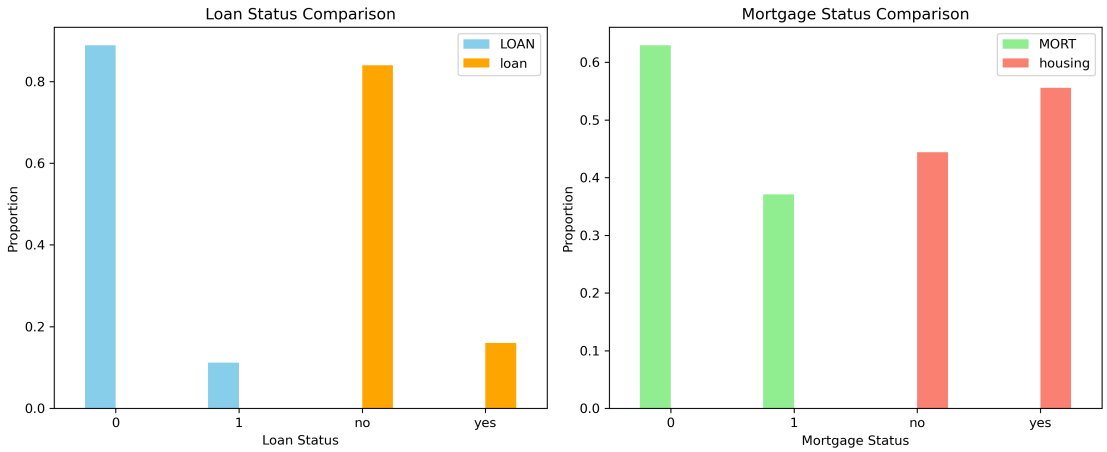


Figure 12: Distribution Comparison of Personal Loan and Housing Loan

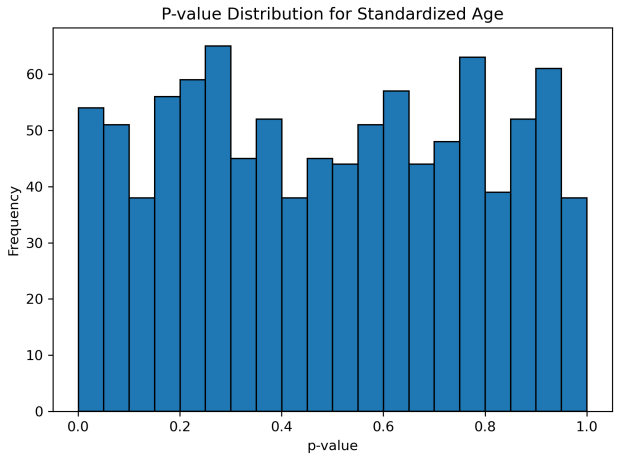


Figure 13: P-value Distribution for Standardized Age

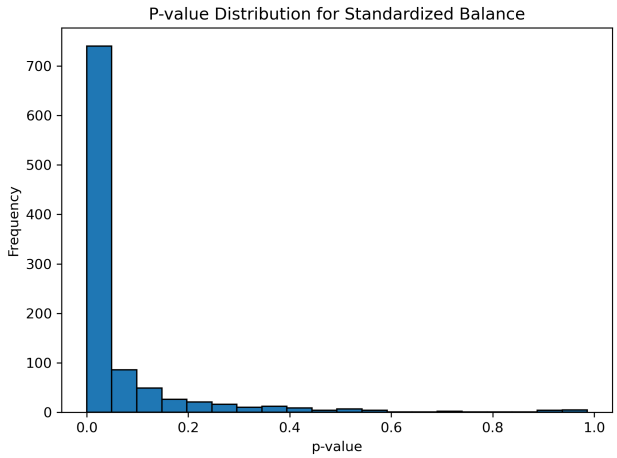


Figure 14: P-value Distribution for Standardized Balance

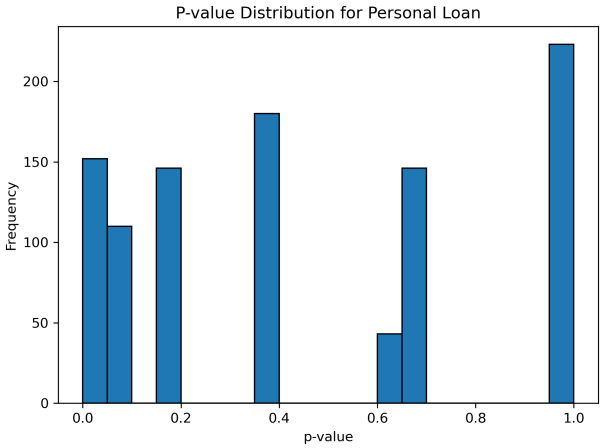


Figure 15: P-value Distribution for Personal Loan

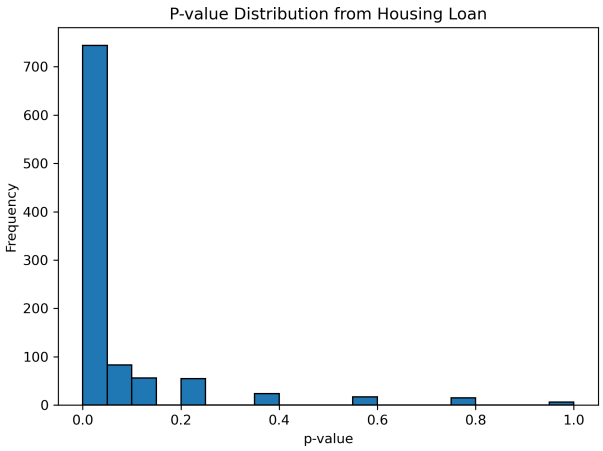


Figure 16: P-value Distribution for Housing Loan

To estimate customer-level revenue, we implemented a Random Forest regression model using features from the intersection of campaign dataset and revenue dataset. We selected Random Forest over XGBoost due to its simpler hyperparameter configuration and greater robustness to overfitting on moderate-sized datasets. We also did experiments and observed it had better generalization performance.

The model was trained on 80% of the data and evaluated on the remaining 20%. Our feature set included four raw predictors: `AGE`, `Bal_Total`, `LOAN`, and `MORT`; as well as four engineered features capturing nonlinear and interaction effects: $\texttt{AGE}^2$, $\texttt{Bal\_Total}^2$, $\texttt{Bal\_Total}/\texttt{AGE}$, and $\texttt{LOAN} \times \texttt{MORT}$. The final model used 300 decision trees with a maximum depth of 10, and hyperparameters were tuned via grid search with 5-fold cross-validation.

On the test set, the model achieved a mean squared error (MSE) of 6.2405 and an R^2 score of 0.2828. Given the limited number of available features and the relatively small size of the training set, this modest level of performance is reasonable and expected. In practical applications, collecting a larger and more diverse set of training samples, along with richer feature information, would likely improve the model's accuracy and robustness. Feature importance analysis revealed that `Bal_Total` and its squared term were the most influential predictors, followed by `MORT`. These results highlight the importance of total account balance and mortgage status in explaining revenue variation across customers.

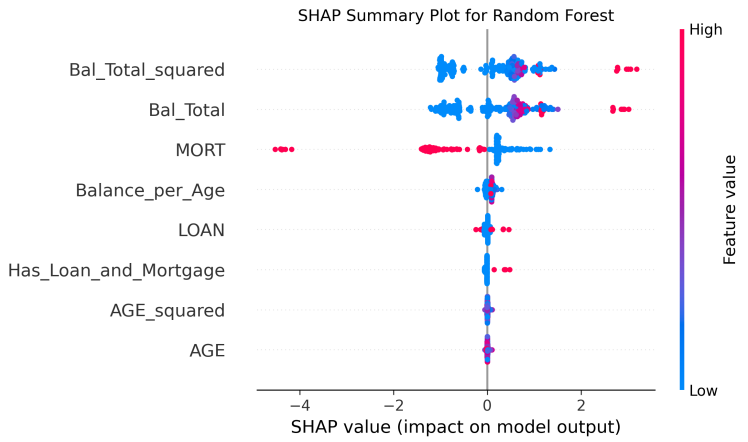


Figure 17: SHAP Summary Plot for Random Forest Regression

5.3.2 Optimization Strategy

Based on the problem formulation, our model is implemented as a binary integer linear program that maximizes the expected net revenue using customer-level predicted revenue and subscription probability. The contact cost is approximated as proportional to the predicted probability. A constant human resource constraint, set to the average number of customers contacted across different months (3767.68 $\approx$ 3768), is applied, and the optimization is solved using the PuLP solver.

To better illustrate the behavior of the optimal selection strategy, we performed distributional analyses using boxplots and barplots based on the top features identified by both the probability and revenue estimation models. We compared these key features across selected and unselected customer groups. The results show that the distribution patterns between the these groups closely mirror those observed in the campaign dataset between customers who subscribed and those who did not. This alignment suggests that the optimization model is effectively capturing behavioral patterns associated with subscription likelihood.

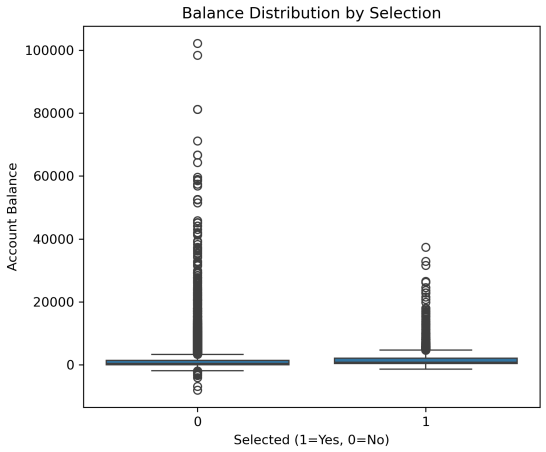


Figure 18: Balance Boxplot-based Distribution by Selection

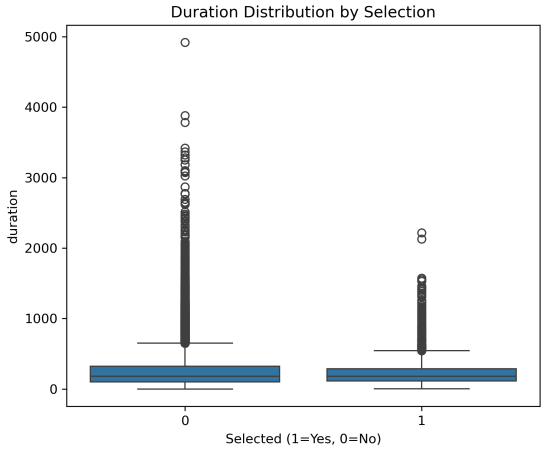


Figure 19: Duration Boxplot-based Distribution by Selection

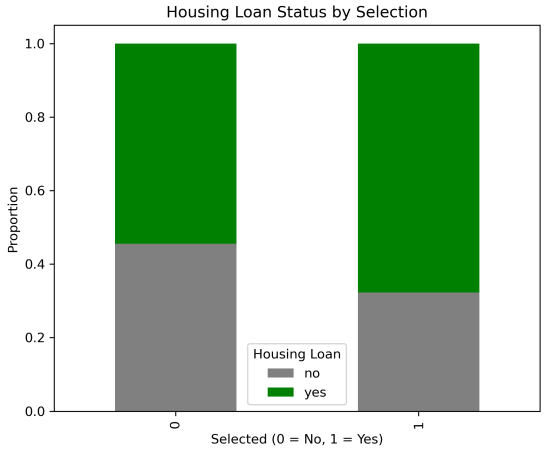


Figure 20: Housing Barplot-based Distribution by Selection

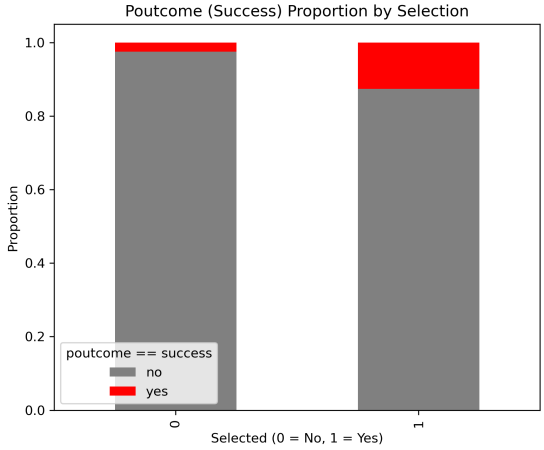


Figure 21: Poutcome Barplot-based Distribution by Selection

Based on insights from multiple modeling approaches, including odds ratios from logistic regression, SHAP values from XGBoost and deep neural networks, and patterns observed in the optimization strategy, we identified a set of intuitive principles behind the optimal selection strategy. These different methods point to consistent signals about which customer characteristics are most linked to successful outcomes, allowing us to summarize the key selection rules as follows.

- Customers who subscribed in previous campaigns exhibit a higher likelihood of subscribing again.
- The recency of customer contact is positively associated with the probability of subscription.
- Customers with existing loans tend to have lower subscription rates and generate less revenue, though this should be considered in combination with other features.
- Customers with higher account balances are generally more profitable; however, extremely high balances are associated with reduced subscription likelihood.
- Call duration has a non-linear effect: moderate durations are optimal, as longer calls may increase subscription rates but also incur higher costs.
- The timing of contact plays a role in campaign effectiveness, with improved responses observed at quarter-end periods, during the tax season (April to June in Portugal), and at the beginning or end of the month.

6 Implementation Guideline

To implement the subscription prediction and optimization strategy, a structured, multi-stage implementation plan is recommended. This approach ensures effective deployment, ongoing monitoring, and alignment with regulatory and business objectives.

6.1 System Integration

The deep learning model (DNN), which has demonstrated superior predictive performance, will be deployed within the bank's Customer Relationship Management (CRM) system. Each incoming or existing client will be scored in real time based on updated financial and behavioral features. The predicted probability of subscription will then feed into the optimization module.

The optimization model, which selects the most profitable subset of clients to recall under a fixed contact budget, will run on a regular schedule (e.g., monthly). This batch-mode execution ensures that updated probabilities, estimated contact costs, and potential revenues are taken into account for every decision window.

6.2 Monitoring and Maintenance

Ongoing monitoring of model performance is critical to ensure sustained accuracy and business relevance. The key evaluation metrics, such as accuracy, precision, recall, F1-score, and ROC-AUC, will be tracked on a weekly basis. A monthly retraining pipeline will be implemented using the most recent campaign data to reflect potential shifts in customer behavior.

For model transparency, SHAP analysis will be conducted regularly to identify the most influential features and to provide explanations to business stakeholders. This step ensures accountability and interpretability, especially in compliance-sensitive environments.

6.3 Operational Considerations

Several considerations are essential for successful implementation:

- **Class Imbalance:** The dataset contains a significant imbalance between subscribers and non-subscribers. Ongoing recalibration of class weights and thresholds is necessary to maintain fairness and predictive power.
- **Threshold Tuning:** Instead of using a fixed probability threshold (e.g., 0.5), we employ a dynamic threshold (e.g., 0.48) tuned to maximize the F1-score, and this should be re-evaluated quarterly.
- **Legal and Ethical Compliance:** Since the solution operates in the financial services domain, appropriate privacy, consent, and fairness reviews must be conducted before model deployment. Data governance practices must be rigorously followed.

Successful implementation requires not only accurate modeling and optimization, but also continuous validation, ethical safeguards, and seamless integration with existing business workflows.

7 Conclusion

7.1 Logistic Regression Conclusion

The logistic regression model explicitly presents the relationships between the explanatory variables and the outcome variable, which makes the model interpretable. Therefore, the model helps to find the variables that could influence the probability of a client subscribing to a term deposit. However, as shown in Table 2 and Table 4, the model displays a low recall compared to the XGBoost model and the Deep Learning model. The low recall reveals that the model fails to capture a large proportion of the true subscribers. And the reason behind the low recall value might be because of the class imbalance and relatively high threshold.

7.2 XGBoost Conclusion

The XGBoost model achieved strong predictive performance on the test set, and it effectively leveraged key predictors such as call duration, previous campaign success, recent engagement, and financial readiness. A major strength of the model is its ability to capture nonlinear feature interactions while remaining computationally efficient. SHAP analysis further enhanced model interpretability, offering actionable marketing insights. However, the model exhibited lower precision for the minority class (subscribers) and, being non-parametric, may limit transparency compared to simpler models. Future improvements could include more advanced hyperparameter tuning (e.g., Bayesian optimization), expanded feature engineering, alternative rebalancing techniques (such as SMOTE), and ensemble approaches to boost performance.

7.3 DNN Conclusion

The deep learning approach offered a substantial improvement in predictive performance over both logistic regression and XGBoost, particularly in identifying positive cases with high recall and F1-score. By incorporating class weighting, resampling, and threshold tuning, we were able to adapt the DNN to address the dataset's class imbalance and optimize decision boundaries. These modifications allowed the model to uncover complex interactions among features that traditional models may overlook.

However, this performance comes at a cost. The lack of interpretability and explainability inherent to deep neural networks limits their practical deployment in regulated industries like finance and marketing, where transparent decision-making is essential. While our DNN model demonstrates the potential of deep learning in subscription prediction, its real-world application would require either supplementary explainability tools or more interpretable alternatives, depending on the organizational and regulatory context.

7.4 Decision Optimization Conclusion

The decision optimization model effectively integrates estimated revenue, subscription probability, contact cost, and human resource constraints to identify the most profitable customers to target. It offers a clear and

interpretable framework that directly aligns with business objectives, ensuring that limited human resources are allocated to customers with the highest expected net return.

However, the model's performance is highly dependent on the quality of data and the accuracy of input predictions. Developling reliable predictive models requires access to large-scale, high-quality datasets with diverse and relevant features. This process depends on thoughtful data collection strategies to ensure the patterns learned are representative and avoids by bias or noise. Otherwise, errors in estimating revenue or subscription probability may lead to suboptimal selections.

Additionally, a one-time optimization model is limited in its ability to capture nonlinear feature interactions or respond to uncertainty in real time. Enhancing the robustness of upstream predictive models and incorporating stochastic optimization techniques could further improve its practical effectiveness.

8 Appendices

8.1 Primary Data Source

Grayson, J., Gardner, S., & Stephens, M. (2015). *Bank Revenue*[Dataset]. Building Better Models with JMP® Pro. SAS Press. https://www.jmp.com/en/academic/case-study-library/bank-revenue

Moro, S., Rita, P., & Cortez, P. (2014). *Bank Marketing* [Dataset]. UCI Machine Learning Repository. https://doi.org/10.24432/C5K306.

SAS Institute Inc. (n.d.). *Bank revenue and marketing campaign case study* [Dataset]. https://www.jmp.com/en/academic/case-study-library/bank-revenue

8.2 Key Codes

```
import delimited "/Users/chloepyx/Downloads/bank+marketing/bank/bank-full.csv",
    clear
drop job marital education contact

tab default, miss
gen default_num = .
replace default_num = 1 if default == "yes"
replace default_num = 0 if default == "no"
label define default_num 0 "No" 1 "Yes"
label values default_num default_num
drop if default == "unknown"

tab default_num, miss

//

tab housing, miss
gen housing_num = .
replace housing_num = 1 if housing == "yes"
replace housing_num = 0 if housing == "no"
label define housing_num 0 "No" 1 "Yes"
label values housing_num housing_num
drop if housing == "unknown"

tab housing housing_num, miss

//

tab loan, miss
gen loan_num = .
```

```
replace loan_num = 1 if loan == "yes"
replace loan_num = 0 if loan == "no"
label define loan_num 0 "No" 1 "Yes"
label values loan_num loan_num
drop if loan == "unknown"

tab loan loan_num, miss

//

tab month, miss
gen month_num = .
local months "jan feb mar apr may jun jul aug sep oct nov dec"
local i = 1
foreach m of local months {
    replace month_num = `i' if month == "`m'"
    local ++i
}

tab month month_num, miss
fvset base 1 month_num


//
tab day, nolabel
label variable day "Day of month (1  31  )"

//

tab pdays, missing
sum pdays, detail
replace pdays = -1 if pdays == 999
gen contacted_before = (pdays != -1)
//

tab poutcome, miss
encode poutcome, gen(poutcome_num)
tab poutcome_num, nolabel

//

tab y, missing
codebook y
gen y_numeric = (y == "yes")
label define y_lab 0 "no" 1 "yes"
label values y_numeric y_lab
tab y_numeric
codebook y_numeric

//

logit y_numeric age balance default_num housing_num loan_num ib1.month_num ib1.day
    duration campaign i.contacted_before previous ib4.poutcome_num, vce(robust)

//


logit, or

```

```
//

*backtesting

set seed 123

sort y_numeric
by y_numeric: gen rand = runiform()
by y_numeric: gen train = (rand <= 0.7)
gen validation = !train
drop rand

bysort y_numeric: tab train

tab y_numeric if train
if r(r) == 2 {
    logit y_numeric age balance default_num housing_num loan_num ib1.month_num day ///
          duration campaign i.contacted_before previous ib4.poutcome_num if train, vce(robust)

predict prob_in_sample if train, pr

gen pred_in_sample = (prob_in_sample >= 0.5) if train

tab y_numeric pred_in_sample if train, cell

gen actual = y_numeric if train
gen predicted = pred_in_sample if train

count if actual==1 & predicted==1
local TP_in = r(N)

count if actual==0 & predicted==1
local FP_in = r(N)

count if actual==0 & predicted==0
local TN_in = r(N)

count if actual==1 & predicted==0
local FN_in = r(N)

di _n "IN-SAMPLE PERFORMANCE METRICS:"
di "Accuracy: " %4.3f (`TP_in' + `TN_in') / (`TP_in' + `TN_in' + `FP_in' + `FN_in')
di "Sensitivity/Recall: " %4.3f `TP_in' / (`TP_in' + `FN_in')
di "Specificity: " %4.3f `TN_in' / (`TN_in' + `FP_in')
di "Precision: " %4.3f `TP_in' / (`TP_in' + `FP_in')
di "F1 Score: " %4.3f 2*(`TP_in'/(`TP_in'+`FP_in'))*(`TP_in'/(`TP_in'+`FN_in')) ///
                      (`TP_in'/(`TP_in'+`FP_in') + `TP_in'/(`TP_in'+`FN_in'))

lroc if train, nograph
di "In-sample AUC: " %4.3f r(area)


cap drop prob_default
predict prob_default if validation, pr
summarize prob_default if validation
```

```
143
144 cap drop pred_default
145 gen pred_default = (prob_default >= 0.5) if validation
146
147 tab y_numeric pred_default if validation, cell
148
149 local TN = 11705
150 local FP = 282
151 local FN = 1004
152 local TP = 511
153
154 di _n "MODEL PERFORMANCE METRICS:"
155 di "Accuracy: " %4.3f (`TP' + `TN') / (`TP' + `TN' + `FP' + `FN')
156 di "Sensitivity/Recall: " %4.3f `TP' / (`TP' + `FN')
157 di "Specificity: " %4.3f `TN' / (`TN' + `FP')
158 di "Precision: " %4.3f `TP' / (`TP' + `FP')
159 di "F1 Score: " %4.3f 2*(`TP'/(`TP'+`FP'))*(`TP'/(`TP'+`FN'))/(`TP'/(`TP'+`FP')+`
    TP'/(`TP'+`FN'))
160
161 lroc if validation, nograph
162 di "AUC: " %4.3f r(area)
163 }
164 else {
165     di "WARNING: Training set missing one class. Cannot estimate model."
166     di "Try oversampling or different stratification method."
167 }
```

Listing 1: Logistic Regression Code in Stata

XGBoost Model

In [ ]:
```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import xgboost as xgb
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import roc_auc_score, accuracy_score
from sklearn.metrics import classification_report, roc_auc_score, roc_curve
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay
import shap
import numpy as np
```

In [ ]:
```
df_orig = pd.read_csv("/Users/ziyuzhou/Downloads/bank-full.csv",
                      sep=';',
                      quotechar='"')
df = df_orig.copy()
df.drop(columns=["job", "marital", "education"], inplace=True)
print(df.head())
```

In [ ]:
```
print("Shape of dataset:", df.shape) # 45211,14
print("\nMissing values per column:")
print(df.isnull().sum()) # no missing value
```

In [ ]:
```
print(df['y'].value_counts(normalize=True)) # imbalanced, use scale_pos_weig
```

In [ ]:
```
# Prepare y variable counts and percentages
y_var = (
    df["y"]
    .value_counts()
    .rename_axis('y')
    .reset_index(name='counts')
    .assign(percent=lambda d: (d["counts"] / d["counts"].sum() * 100).round(
)

# Custom colors
target_color = sns.color_palette(["steelblue", "chocolate"])

# Labels for slices
labels = ["0 | Not subscribed", "1 | Subscribed"]

# Prepare combined labels for autopct
autopct_labels = [
    f"{row['percent']}%\n(count: {row['counts']})"
    for _, row in y_var.iterrows()
]

fig, ax = plt.subplots(figsize=(8, 6))

# Plot pie with soft shadow effect
wedges, texts, autotexts = ax.pie(
    x=y_var["percent"],
    labels=labels,
    colors=target_color,
    explode=[0.1, 0],
    shadow=False,  # keep basic shadow
    startangle=90,
    textprops={"fontsize": 11, "color": "#000000"},
    autopct=lambda pct: ''  # leave blank first
)


# Set autopct labels manually
for i, a in enumerate(autotexts):
    a.set_text(autopct_labels[i])
    a.set_fontsize(11)
    a.set_color('black')
    a.set_ha('center')
    a.set_va('center')

plt.tight_layout()
plt.savefig('pie_y.png', dpi=300, bbox_inches='tight')
plt.show()
```

In [ ]:
```
categorical_cols = ['default', 'housing', 'loan', 'contact', 'month', 'poutc
for col in categorical_cols:
    print(df[col].value_counts()) # consider drop unknown value
```

In [ ]:

```
#categorical_cols = ['default', 'housing', 'loan', 'contact', 'month', 'pout
categorical_cols = ['default', 'loan', 'month', 'poutcome']

# Month order manually defined
month_order = ['jan', 'feb', 'mar', 'apr', 'may', 'jun',
               'jul', 'aug', 'sep', 'oct', 'nov', 'dec']

n_cols = 2
n_rows = (len(categorical_cols) + 1) // n_cols
plt.figure(figsize=(14, 4 * n_rows))

for i, col in enumerate(categorical_cols, 1):
    plt.subplot(n_rows, n_cols, i)

    # Group and normalize manually
    counts = df.groupby([col, 'y']).size()
    total = counts.groupby(level=0).sum()
    proportion = (counts / total).reset_index(name='proportion')

    # Set specific order for month
    if col == 'month':
        order = month_order
    else:
        order = sorted(proportion[col].unique())  # default alphabetic for c

    # Plot
    sns.barplot(data=proportion, x=col, y='proportion', hue='y', order=order

    plt.title(f"{col} by Target y | Proportions")
    plt.xlabel(col)
    plt.ylabel("Proportion")
    plt.xticks(rotation=0)

    # Adjust legend: smaller font, and place outside
    plt.legend(
        title='y',
        loc='upper left',
        bbox_to_anchor=(1, 1),
        fontsize='small',
        title_fontsize='small'
    )

plt.tight_layout()
plt.savefig('bar_catag_ppt.png', dpi=300, bbox_inches='tight')
plt.show()
```

In [ ]:
```
categorical_cols = ['default', 'housing', 'loan', 'contact', 'month', 'poutc

n_cols = 2
n_rows = (len(categorical_cols) + 1) // n_cols
plt.figure(figsize=(12, 4 * n_rows))

for i, col in enumerate(categorical_cols, 1):
    plt.subplot(n_rows, n_cols, i)
    sns.countplot(data=df, x=col, hue='y')
    plt.title(f"{col} by Target (y)")
    plt.xlabel(col)
    plt.ylabel("Count")
    plt.xticks(rotation=45)
    plt.legend(title='y', loc='upper right')

plt.tight_layout()
plt.show()
```

In [ ]:
```
numeric_cols = ['duration', 'campaign', 'pdays', 'previous','age','balance',
print(df[numeric_cols].describe())

# consider standardize duration and campaign for stats model
```

In [ ]:

```
numeric_cols = ['duration', 'balance', 'age', 'campaign', 'pdays', 'previous

# Set style (match your sample exactly)
sns.set(rc={'figure.figsize': (12, 8)}, font_scale=1.5, style='whitegrid')

for col in numeric_cols:
    # Create subplots
    f, (ax_box, ax_hist) = plt.subplots(
        2,
        sharex=True,
        gridspec_kw={"height_ratios": (0.3, 1)},
        figsize=(11, 8)
    )

    # Calculate statistics
    feature_array = df[col].dropna().to_numpy()
    mean = np.mean(feature_array)
    median = np.median(feature_array)
    mode = pd.Series(feature_array).mode().values[0]

    # Boxplot (top)
    sns.boxplot(
        data=df, x=col, y = 'y', hue="y",
        ax=ax_box,
        order=df["y"].value_counts().index
    )
    ax_box.axvline(mean, color='r', linestyle='--', label="Mean")
    ax_box.axvline(median, color='g', linestyle='-', label="Median")
    ax_box.axvline(mode, color='b', linestyle='-', label="Mode")
    ax_box.set(xlabel='')
    #ax_box.set_title(f"{col} distribution by Target y")

    # Histogram (bottom)
    sns.histplot(data=df, x=col, ax=ax_hist)
    ax_hist.axvline(mean, color='r', linestyle='--', label="Mean")
    ax_hist.axvline(median, color='g', linestyle='-', label="Median")
    ax_hist.axvline(mode, color='b', linestyle='-', label="Mode")

    ax_hist.set_xlabel(col)
    ax_hist.set_ylabel("Count")
    ax_hist.legend()

    plt.tight_layout()

    # Save figure with its feature name
    plt.savefig(f'{col}_distr.png', dpi=300, bbox_inches='tight')
    plt.show()
```

In [ ]:
```
numeric_cols = ['duration', 'campaign', 'pdays', 'previous','age','balance',

plt.figure(figsize=(12, 8))

for i, col in enumerate(numeric_cols, 1):
    plt.subplot(4, 2, i)
    plt.hist(df[col].dropna(), bins=30, edgecolor='black')
    plt.title(f"Distribution of {col}")
    plt.xlabel(col)
    plt.ylabel("Count")

plt.tight_layout()
plt.show()
```

In [ ]:
```
numeric_cols = ['duration', 'campaign', 'pdays', 'previous','age','balance',

plt.figure(figsize=(12, 10))

for i, col in enumerate(numeric_cols, 1):
    plt.subplot(4, 2, i)
    sns.boxplot(data=df, x='y', y=col)
    plt.title(f"{col} by Target (y)")
    plt.xlabel("y")
    plt.ylabel(col)

plt.tight_layout()
plt.show()

# duration and previous may act as important feature
```

In [ ]:
```
# Convert target variable 'y' from 'yes'/'no' to 1/0
df['y'] = df['y'].map({'yes': 1, 'no': 0})

# Compute correlation matrix (only numeric columns)
corr_matrix = df.corr(numeric_only=True)

plt.figure(figsize=(8, 6))
sns.heatmap(
    corr_matrix,
    annot=True,
    fmt=".2f",
    cmap="coolwarm",
    square=True,
    cbar=True,
    annot_kws={"size": 12}
)
plt.tight_layout()
plt.savefig('corr_mat.png', dpi=300, bbox_inches='tight')
plt.show()
```

In [ ]:
```
df = df[df['contact'] != 'unknown']  # removes rows with 'unknown' in contac
df = df[df['poutcome'] != 'unknown']  # removes rows with 'unknown' in poutc
df = df[df['poutcome'] != 'other'] # removes rows with 'other' in poutcome
```

In [ ]:
```
categorical_cols = ['default', 'housing', 'loan', 'contact', 'month', 'poutc
for col in categorical_cols:
    print(df[col].value_counts()) # consider drop unknown value
```

In [ ]:
```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()

# standardization
df[['duration', 'campaign','balance','pdays']] = scaler.fit_transform(df[['d
```

In [ ]:
```
# convert categorical to numerical type
categorical_cols = ['default', 'housing', 'loan', 'contact', 'month', 'poutc
df = pd.get_dummies(df, columns=categorical_cols, drop_first=True)
# Convert boolean columns (if any) to integers
bool_cols = df.select_dtypes(include='bool').columns
df[bool_cols] = df[bool_cols].astype(int)


print(df.head())
```

In [ ]:
```
# Define features and target
X = df.drop(columns=['y'])
y = df['y']

# Train-test split
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, stratify=y, test_s

# Count class distribution
neg, pos = (y_train == 0).sum(), (y_train == 1).sum()
scale_pos_weight = neg / pos
print(f"scale_pos_weight: {scale_pos_weight:.2f}")
```

In [ ]:
```python
# Define parameter grid for tuning
param_grid = {
    'max_depth': [4,6,8,10],
    'min_child_weight': [10, 20, 30],
    'gamma': [5, 10],
    'subsample': [0.8],
    'colsample_bytree': [0.8],
    'learning_rate': [0.01,0.05],
    'scale_pos_weight': [2,4,6]  # tune this further
}


# Initialize model
xgb_clf = xgb.XGBClassifier(
    tree_method='hist',
    eval_metric='logloss',
    scale_pos_weight=scale_pos_weight,
    missing=np.nan,
    random_state=42
)

# Grid search with 5-fold cross-validation
grid_search = GridSearchCV(
    estimator=xgb_clf,
    param_grid=param_grid,
    cv=5,
    scoring='accuracy',
    n_jobs=-1,
    verbose=1
)

grid_search.fit(X_train, y_train)

# Best model
best_model = grid_search.best_estimator_
print("Best Parameters:", grid_search.best_params_)
```

In [ ]:
```python
# Predict on training data
y_train_pred = best_model.predict(X_train)
y_train_prob = best_model.predict_proba(X_train)[:, 1]

# Accuracy on training data
train_accuracy = accuracy_score(y_train, y_train_pred)
print(f"Training Accuracy: {train_accuracy:.4f}")

# AUC-ROC on training data
train_auc = roc_auc_score(y_train, y_train_prob)
print(f"Training AUC-ROC: {train_auc:.4f}")
```

In [ ]:
```
# Predict on test data
y_pred = best_model.predict(X_test)
y_prob = best_model.predict_proba(X_test)[:, 1]

# Accuracy
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy:.4f}")

# AUC-ROC
auc = roc_auc_score(y_test, y_prob)
print(f"AUC-ROC: {auc:.4f}")

# Full classification report
print("\nClassification Report:")
print(classification_report(y_test, y_pred))
```

In [ ]:
```
print(y_train_prob)
```

In [ ]:
```
# Save y_prob into a CSV file
pd.DataFrame({'y_test_prob': y_prob}).to_csv('y_test_prob_output.csv', index
pd.DataFrame({'y_train_prob': y_train_prob}).to_csv('y_train_prob_output.csv
```

In [ ]:
```
fpr, tpr, thresholds = roc_curve(y_test, y_prob)

plt.figure(figsize=(6, 4))
plt.plot(fpr, tpr, label=f'AUC = {auc:.4f}')
plt.plot([0, 1], [0, 1], 'k--')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.legend(loc='lower right')
plt.grid(True)
plt.tight_layout()
# Save the plot
plt.savefig('xgb_roc.png', dpi=300, bbox_inches='tight')

plt.show()
```

In [ ]:
```
# Predict labels on the test set
y_pred = best_model.predict(X_test)

# Compute confusion matrix
cm = confusion_matrix(y_test, y_pred)
print("Confusion Matrix:\n", cm)
```

In [ ]:
```
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=["No", "Ye
disp.plot(cmap="Blues")
plt.grid(False)

# Save the plot
plt.savefig('xgb_confu.png', dpi=300, bbox_inches='tight')
plt.show()
```

In [ ]:
```
# Get feature importances from the model
plt.figure(figsize=(10, 6))
booster = best_model.get_booster()
importance_dict = booster.get_score(importance_type='gain')

importance_series = pd.Series(importance_dict).sort_values(ascending=False)

xgb.plot_importance(best_model,
                    max_num_features=23,
                    importance_type='gain',
                    height=0.5,
                    show_values=False)

plt.tight_layout()

plt.savefig('gain_bar.png', dpi=300, bbox_inches='tight')

plt.show()

# Print top 23 feature importances with gain values
print("\nTop 20 Feature Importances by Gain:")
for feature, score in importance_series.head(23).items():
    print(f"{feature}: {score:.4f}")
```

In [ ]:
```
explainer = shap.Explainer(best_model, X_test)
shap_values = explainer(X_test)

plt.figure(figsize=(10, 6))

shap.plots.bar(shap_values,max_display=30, show = False)

plt.savefig('shap_bar.png', dpi=300, bbox_inches='tight')

plt.show()
```

In [ ]:
```
plt.figure(figsize=(10, 6))

shap.plots.beeswarm(shap_values, max_display=30, show = False)

plt.savefig('shap_beeswarm.png', dpi=300, bbox_inches='tight')

plt.show()
```

In [ ]:
```
plt.figure(figsize=(10, 6))
shap.plots.scatter(shap_values[:, "duration"], show = False)
plt.savefig('shap_dur.png', dpi=300, bbox_inches='tight')
```

In [ ]:
```
plt.figure(figsize=(10, 6))
shap.plots.scatter(shap_values[:, "day"], show = False)
plt.savefig('shap_day.png', dpi=300, bbox_inches='tight')
```

In [ ]:
```
plt.figure(figsize=(10, 6))
shap.plots.scatter(shap_values[:, "balance"], show = False)
plt.savefig('shap_balance.png', dpi=300, bbox_inches='tight')
```

In [ ]:
```
shap.plots.scatter(shap_values[:, "month_may"])
```

In [ ]:
```
plt.figure(figsize=(10, 6))
shap.plots.scatter(shap_values[:, "pdays"], show = False)
plt.savefig('shap_pdays.png', dpi=300, bbox_inches='tight')
```

In [ ]:
```
shap.plots.scatter(shap_values[:, "pdays"], color=shap_values[:, "previous"]
```

DNN Model

In [ ]:
```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler, OneHotEncoder
from sklearn.compose import ColumnTransformer
from sklearn.metrics import accuracy_score, classification_report, roc_auc_s
from sklearn.utils import resample
import matplotlib.pyplot as plt
import torch
import torch.nn as nn
import torch.optim as optim
from torch.utils.data import DataLoader, TensorDataset
```

In [ ]:
```
data = pd.read_csv("/bank-full.csv", sep=';')

data = data.drop(['marital', 'education'], axis=1)

X = data.drop('y', axis=1)
y = data['y'].map({'yes': 1, 'no': 0})

X['y'] = y
minority = X[X['y'] == 1]
majority = X[X['y'] == 0]
minority_upsampled = resample(minority, replace=True, n_samples=len(majority
X_balanced = pd.concat([majority, minority_upsampled])

X = X_balanced.drop('y', axis=1)
y = X_balanced['y']

categorical_cols = X.select_dtypes(include=['object']).columns.tolist()
numerical_cols = X.select_dtypes(include=['int64', 'float64']).columns.tolis

categorical_transformer = OneHotEncoder(drop='first')
numerical_transformer = StandardScaler()

preprocessor = ColumnTransformer(
    transformers=[
        ('num', numerical_transformer, numerical_cols),
        ('cat', categorical_transformer, categorical_cols)
    ])

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, rar

X_train = preprocessor.fit_transform(X_train)
X_test = preprocessor.transform(X_test)

X_train_tensor = torch.tensor(X_train.toarray() if hasattr(X_train, 'toarray
y_train_tensor = torch.tensor(y_train.values, dtype=torch.float32).view(-1,
X_test_tensor = torch.tensor(X_test.toarray() if hasattr(X_test, 'toarray')
y_test_tensor = torch.tensor(y_test.values, dtype=torch.float32).view(-1, 1)
```

```
train_dataset = TensorDataset(X_train_tensor, y_train_tensor)
test_dataset = TensorDataset(X_test_tensor, y_test_tensor)
train_loader = DataLoader(train_dataset, batch_size=32, shuffle=True)
test_loader = DataLoader(test_dataset, batch_size=32)

class DNN(nn.Module):
    def __init__(self, input_dim):
        super(DNN, self).__init__()
        self.model = nn.Sequential(
            nn.Linear(input_dim, 128),
            nn.ReLU(),
            nn.Dropout(0.3),
            nn.Linear(128, 64),
            nn.ReLU(),
            nn.Dropout(0.3),
            nn.Linear(64, 32),
            nn.ReLU(),
            nn.Dropout(0.3),
            nn.Linear(32, 1)
        )

    def forward(self, x):
        return self.model(x)

model = DNN(X_train_tensor.shape[1])

pos_weight = torch.tensor([(y_train == 0).sum() / (y_train == 1).sum()], dty
criterion = nn.BCEWithLogitsLoss(pos_weight=pos_weight)
optimizer = optim.Adam(model.parameters(), lr=0.001)

for epoch in range(50):
    model.train()
    epoch_loss = 0
    for batch_X, batch_y in train_loader:
        optimizer.zero_grad()
        outputs = model(batch_X)
        loss = criterion(outputs, batch_y)
        loss.backward()
        optimizer.step()
        epoch_loss += loss.item()
    print(f"Epoch {epoch+1}, Loss: {epoch_loss/len(train_loader):.4f}")

model.eval()
y_probs = []
y_true = []
with torch.no_grad():
    for batch_X, batch_y in test_loader:
        outputs = model(batch_X)
        probs = torch.sigmoid(outputs)
        y_probs.extend(probs.numpy())
        y_true.extend(batch_y.numpy())

y_probs = np.array(y_probs).flatten()
y_true = np.array(y_true).flatten()
```

```
best_f1 = 0
best_thresh = 0.5
for thresh in np.arange(0.1, 0.9, 0.01):
    preds = (y_probs >= thresh).astype(int)
    f1 = f1_score(y_true, preds)
    if f1 > best_f1:
        best_f1 = f1
        best_thresh = thresh

print(f"Best Threshold: {best_thresh:.2f} with F1: {best_f1:.4f}")

final_preds = (y_probs >= best_thresh).astype(int)
print("\nClassification Report:")
print(classification_report(y_true, final_preds))
print(f"ROC-AUC Score: {roc_auc_score(y_true, y_probs):.4f}")

fpr, tpr, _ = roc_curve(y_true, y_probs)
plt.figure(figsize=(8,6))
plt.plot(fpr, tpr, label=f'ROC curve (area = {roc_auc_score(y_true, y_probs)
plt.plot([0,1], [0,1], 'k--')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve')
plt.legend()
plt.grid()
plt.show()
```

Permutation Feature Importance

In [ ]:
```
from sklearn.metrics import roc_auc_score
import copy

baseline_score = roc_auc_score(y_true, y_probs)
importances = []

X_test_np = X_test_tensor.numpy()

for i in range(X_test_np.shape[1]):
    X_perm = copy.deepcopy(X_test_np)
    np.random.shuffle(X_perm[:, i])
    with torch.no_grad():
        preds = model(torch.tensor(X_perm, dtype=torch.float32))
        probs = torch.sigmoid(preds).numpy().flatten()
        score = roc_auc_score(y_true, probs)
    importances.append(baseline_score - score)
```

In [ ]:
```
feature_names = preprocessor.get_feature_names_out()
importance_df = pd.DataFrame({
    'Feature': feature_names,
    'Importance': importances
}).sort_values(by='Importance', ascending=False)
top_n = 20
top_features = importance_df.head(top_n)
```

```
plt.figure(figsize=(10, 6))
plt.barh(top_features['Feature'], top_features['Importance'])
plt.xlabel('AUC Drop')
plt.title(f'Top {top_n} Permutation Feature Importances', pad=10)
plt.gca().invert_yaxis()
plt.tight_layout()
plt.show()
```

Decision Optimization

In [ ]:
```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split, cross_val_score, KFold
from sklearn.preprocessing import StandardScaler
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_squared_error, r2_score
from joblib import load
from joblib import dump

df = pd.read_excel('bankrevenue.xlsx')  # Read the Excel file into a pandas
df.to_csv('bankrevenue.csv', index=False)  # Save the data as a CSV file

data = pd.read_csv('bankrevenue.csv')  # Load the data from the CSV file

X = data[['AGE', 'Bal_Total', 'LOAN', 'MORT']].copy()  # Features: Age, Bala
y = data['Rev_Total'].copy()  # Target: Revenue

# Featurization
X['AGE_squared'] = X['AGE'] ** 2  # Create a new feature: AGE squared
X['Bal_Total_squared'] = X['Bal_Total'] ** 2  # Create a new feature: Balanc
X['Balance_per_Age'] = X['Bal_Total'] / (X['AGE'] + 1)  # Create a new featu
X['Has_Loan_and_Mortgage'] = X['LOAN'] * X['MORT']  # Create a new feature:

feature_columns = X.columns.tolist()
dump(feature_columns, 'feature_columns.joblib')

# Split the data into training and testing sets (80% train, 20% test)
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42  # Randomly split the data with a f
)

# Standardize the features
scaler = StandardScaler()
num_features = ['AGE', 'Bal_Total', 'AGE_squared', 'Bal_Total_squared', 'Bal

X_train_scaled = X_train.copy()
X_test_scaled = X_test.copy()

X_train_scaled[num_features] = scaler.fit_transform(X_train[num_features])
X_test_scaled[num_features] = scaler.transform(X_test[num_features])  # Tran

dump(scaler, 'scaler.joblib')

# Define the parameter grid for Random Forest hyperparameter tuning
param_grid = {
    'n_estimators': [100, 200, 300],  # Number of trees in the forest
    'max_depth': [None, 5, 10],  # Maximum depth of the tree (None means no
    'min_samples_split': [2, 5, 10],  # Minimum samples required to split a
    'min_samples_leaf': [1, 2, 4]  # Minimum samples required to be at a lea
}

# Perform Grid Search with Cross-Validation to find the best hyperparameters
```

```
grid_search = GridSearchCV(
    RandomForestRegressor(random_state=42),  # RandomForest model with a fix
    param_grid,  # Parameter grid to search over
    scoring='neg_mean_squared_error',  # Use negative MSE as the scoring met
    cv=5,  # 5-fold cross-validation
    n_jobs=-1  # Use all available cores for parallel processing
)
grid_search.fit(X_train_scaled, y_train)  # Fit the grid search to the train

print("Best Parameters:", grid_search.best_params_)  # Print the best hyperp
print("Best CV MSE:", -grid_search.best_score_)  # Print the best CV MSE sco

# Get the best RandomForest model from grid search
rf_model = grid_search.best_estimator_  # Get the best model based on the gr

# Perform K-Fold Cross-Validation on the best model
kfold = KFold(n_splits=5, shuffle=True, random_state=42)  # 5-fold cross-val
cv_scores = cross_val_score(rf_model, X_train_scaled, y_train, cv=kfold, sco

# Print the cross-validation MSE scores and the average MSE
print(f"Cross-validation MSE scores: {-cv_scores}")  # Print MSE scores for
print(f"Average CV MSE: {-cv_scores.mean():.4f}")  # Print the average CV MS

# Train the model on the entire training set and make predictions on the tes
rf_model.fit(X_train_scaled, y_train)  # Fit the Random Forest model on the
y_pred = rf_model.predict(X_test_scaled)  # Predict on the test set

# Calculate and print the Mean Squared Error (MSE) and R-squared (R^2) score
mse = mean_squared_error(y_test, y_pred)  # Calculate the MSE on the test se
r2 = r2_score(y_test, y_pred)  # Calculate the R^2 score on the test set

print(f"Test MSE: {mse:.4f}")  # Print the test MSE
print(f"Test R^2 Score: {r2:.4f}")  # Print the test R^2 score

dump(rf_model, 'best_random_forest_model.joblib')

print("Model and scaler saved successfully.")
```

In [ ]:

```
import pandas as pd
from joblib import load

feature_columns = load('feature_columns.joblib')            # List of featur
rf_model = load('best_random_forest_model.joblib')          # Trained Random

importances = rf_model.feature_importances_
feat_imp = pd.Series(importances, index=feature_columns)
feat_imp = feat_imp.sort_values(ascending=False)

print("Feature importances:")
print(feat_imp)

import matplotlib.pyplot as plt

plt.figure(figsize=(8, 4))
feat_imp.plot(kind='bar')
plt.title("Feature Importances")
```

```
plt.ylabel("Importance")
plt.tight_layout()
plt.savefig('feature_importances.png', dpi=300, bbox_inches='tight')
plt.show()
```

In [ ]:
```
import pandas as pd
import joblib
import shap
import matplotlib.pyplot as plt

data = pd.read_csv('bankrevenue.csv')
feature_columns = joblib.load('feature_columns.joblib')
scaler = joblib.load('scaler.joblib')
rf_model = joblib.load('best_random_forest_model.joblib')

X = data[['AGE', 'Bal_Total', 'LOAN', 'MORT']].copy()
X['AGE_squared'] = X['AGE'] ** 2
X['Bal_Total_squared'] = X['Bal_Total'] ** 2
X['Balance_per_Age'] = X['Bal_Total'] / (X['AGE'] + 1)
X['Has_Loan_and_Mortgage'] = X['LOAN'] * X['MORT']

num_feats = ['AGE', 'Bal_Total', 'AGE_squared', 'Bal_Total_squared', 'Balanc
X_scaled = X.copy()
X_scaled[num_feats] = scaler.transform(X[num_feats])

X_sample = X_scaled.sample(n=200, random_state=42)

explainer = shap.TreeExplainer(rf_model)
shap_values = explainer.shap_values(X_sample)

plt.figure(figsize=(10, 6))
shap.summary_plot(shap_values, X_sample, feature_names=feature_columns, show
plt.title("SHAP Summary Plot for Random Forest")
plt.tight_layout()
plt.savefig('shap_summary.png', dpi=300, bbox_inches='tight')
plt.show()
```

In [ ]:
```
import pandas as pd
from joblib import load

# Load the saved Random Forest model
rf_model = load('best_random_forest_model.joblib')

# Load new prediction data (bank-full.csv) with correct delimiter
new_data = pd.read_csv('bank-full.csv', delimiter=';')

# Strip unwanted characters from column names
new_data.columns = new_data.columns.str.replace('"', '').str.strip()
```

In [ ]:
```
sns.histplot(new_data['balance'], kde=True)
plt.title('Balance Distribution (new_data)')
plt.xlabel('Balance')
plt.ylabel('Frequency')
plt.show()
```

```
stats.probplot(new_data['balance'], dist="norm", plot=plt)
plt.title('QQ Plot of Balance (new_data)')
plt.xlabel('Theoretical Quantiles')
plt.ylabel('Sample Quantiles')
plt.show()
```

In [ ]:
```
sns.histplot(new_data['age'], kde=True)
plt.title('Age Distribution (new_data)')
plt.xlabel('Age')
plt.ylabel('Frequency')
plt.show()

stats.probplot(new_data['age'], dist="norm", plot=plt)
plt.title('QQ Plot of Age (new_data)')
plt.xlabel('Theoretical Quantiles')
plt.ylabel('Sample Quantiles')
plt.show()
```

In [ ]:
```
import pandas as pd
import numpy as np
from scipy import stats
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.preprocessing import StandardScaler

data = pd.read_csv('bankrevenue.csv')
new_data = pd.read_csv('bank-full.csv', delimiter=';')
new_data.columns = new_data.columns.str.replace('"', '').str.strip()

data_std = StandardScaler().fit_transform(data[['AGE', 'Bal_Total']])
new_data_std = StandardScaler().fit_transform(new_data[['age', 'balance']])

plt.figure(figsize=(12, 5))

plt.subplot(1, 2, 1)
sns.kdeplot(data_std[:, 0], label='AGE (std)', fill=True)
sns.kdeplot(new_data_std[:, 0], label='age (std)', fill=True)
plt.title('Standardized Age Comparison')
plt.legend()

plt.subplot(1, 2, 2)
sns.kdeplot(data_std[:, 1], label='Bal_Total (std)', fill=True)
sns.kdeplot(new_data_std[:, 1], label='balance (std)', fill=True)
plt.title('Standardized Balance Comparison')
plt.legend()

plt.tight_layout()
plt.savefig('standardized_comparison.png', dpi=300, bbox_inches='tight')
plt.show()

u_stat_age_std, p_value_age_std = stats.mannwhitneyu(data_std[:, 0], new_dat
u_stat_bal_std, p_value_bal_std = stats.mannwhitneyu(data_std[:, 1], new_dat

print(f"\nMann-Whitney U Test (Age): p = {p_value_age_std:.4f}")
print(f"Mann-Whitney U Test (Balance): p = {p_value_bal_std:.4f}")
```

```
def repeated_mannwhitney_u(x, y, sample_size=500, n_trials=100, seed=42, fil
    np.random.seed(seed)
    p_values = []

    for _ in range(n_trials):
        x_sample = np.random.choice(x, size=sample_size, replace=False)
        y_sample = np.random.choice(y, size=sample_size, replace=False)
        _, p = stats.mannwhitneyu(x_sample, y_sample, alternative='two-sidec
        p_values.append(p)

    avg_p = np.mean(p_values)
    print(f"{title}")
    print(f"Sample size per trial = {sample_size}, Total trials = {n_trials}
    print(f"Average p-value: {avg_p:.4f}")

    plt.hist(p_values, bins=20, edgecolor='black')
    plt.title(title)
    plt.xlabel("p-value")
    plt.ylabel("Frequency")
    plt.tight_layout()
    plt.savefig(filename, dpi=300, bbox_inches='tight')
    plt.show()

    return p_values
p_vals_age = repeated_mannwhitney_u(
    data_std[:, 0], new_data_std[:, 0],
    sample_size=50, n_trials=1000,
    filename='pvalue_dist_age.png',
    title='P-value Distribution for Standardized Age'
)

p_vals_balance = repeated_mannwhitney_u(
    data_std[:, 1], new_data_std[:, 1],
    sample_size=50, n_trials=1000,
    filename='pvalue_dist_balance.png',
    title='P-value Distribution for Standardized Balance'
)
```

In [ ]:

```
import pandas as pd
from joblib import load

# Load the saved Random Forest model
rf_model = load('best_random_forest_model.joblib')

# Load new prediction data (bank-full.csv) with correct delimiter
new_data = pd.read_csv('bank-full.csv', delimiter=';')

# Strip unwanted characters from column names
new_data.columns = new_data.columns.str.replace('"', '').str.strip()

# Preprocessing: Rename and transform columns
new_data.rename(columns={'age': 'AGE', 'balance': 'Bal_Total', 'housing': 'M
new_data['MORT'] = new_data['MORT'].apply(lambda x: 1 if x == 'yes' else 0)
new_data['LOAN'] = new_data['LOAN'].apply(lambda x: 1 if x == 'yes' else 0)
```

```
# Feature engineering
new_data['AGE_squared'] = new_data['AGE'] ** 2
new_data['Bal_Total_squared'] = new_data['Bal_Total'] ** 2
new_data['Balance_per_Age'] = new_data['Bal_Total'] / (new_data['AGE'] + 1)
new_data['Has_Loan_and_Mortgage'] = new_data['LOAN'] * new_data['MORT']

# Select the exact feature columns used during training
X_new = new_data[['AGE', 'Bal_Total', 'LOAN', 'MORT', 'AGE_squared', 'Bal_To

# Load the scaler
scaler = load('scaler.joblib')

# Standardize only the numerical columns (NOT LOAN, MORT, Has_Loan_and_Mortg
num_features = ['AGE', 'Bal_Total', 'AGE_squared', 'Bal_Total_squared', 'Bal

X_new_scaled = X_new.copy()
X_new_scaled[num_features] = scaler.transform(X_new[num_features])

# Predict
predictions = rf_model.predict(X_new_scaled)

# Save predictions
predictions_df = pd.DataFrame(predictions, columns=['Predicted_Rev_Total'])
predictions_df.to_csv('predictions.csv', index=False)

print("Predictions saved to 'predictions.csv'.")
```

In [ ]:
```
import pandas as pd

bank_data = pd.read_csv('bank-full.csv', delimiter=';')

predictions = pd.read_csv('predictions.csv')

bank_data.columns = bank_data.columns.str.replace('"', '').str.strip()

# y == 'yes'
yes_count = (bank_data['y'] == 'yes').sum()
print(f"Number of rows with y == 'yes': {yes_count}")

yes_indices = bank_data.index[bank_data['y'] == 'yes'].tolist()

predictions_cleaned = predictions.drop(index=yes_indices)

predictions_cleaned.to_csv('predictions_cleaned.csv', index=True)
```

In [ ]:
```
bank_full_with_y_prob = pd.read_csv('bank_full_with_y_prob.csv', index_col=0
subset_indices = bank_full_with_y_prob.index

filtered_predictions = predictions_cleaned.loc[predictions_cleaned.index.int

filtered_predictions.to_csv('filtered_predictions.csv', index=True)
```

In [ ]:
```
filtered_predictions = pd.read_csv('filtered_predictions.csv', index_col=0)

common_indices = bank_full_with_y_prob.index.intersection(filtered_predictic
```

```
final_result = bank_full_with_y_prob.loc[common_indices]

final_result.to_csv('final_result.csv', index=True)
```

In [ ]:
```
import pandas as pd
from pulp import LpMaximize, LpProblem, LpVariable, lpSum

df_revenue = pd.read_csv('final_revenue.csv', index_col=0)
df_prob = pd.read_csv('final_probability.csv', index_col=0)

df = df_revenue.join(df_prob, how='inner')

r = df['Predicted_Rev_Total'].to_numpy()
p = df['y_prob'].to_numpy()
c = p.copy()
H = 3768
n = len(df)

model = LpProblem(name="customer-optimization", sense=LpMaximize)
x = [LpVariable(f"x_{i}", cat="Binary") for i in range(n)]

model += lpSum((r[i] - c[i]) * p[i] * x[i] for i in range(n)), "Total_Expect
model += lpSum(x) <= H, "Human_Resource_Limit"

model.solve()

selected_indices = [i for i in range(n) if x[i].value() == 1]
selected_customers = df.iloc[selected_indices]

selected_customers.to_csv('selected_customers_ilp.csv', index=True)
print(f"Selected {len(selected_customers)} customers saved to 'selected_cust
```

In [ ]:
```
import pandas as pd

df = pd.read_csv('bank-full.csv', delimiter=';')

df.columns = df.columns.str.strip().str.replace('"', '')

month_counts = df['month'].value_counts()

print(month_counts)
```

In [ ]:
```
import pandas as pd

month_counts = pd.Series({
    'may': 13766,
    'jul': 6895,
    'aug': 6247,
    'jun': 5341,
    'nov': 3970,
    'apr': 2932,
    'feb': 2649,
    'jan': 1403,
    'oct': 738,
```

```
    'sep': 579,
    'mar': 477,
    'dec': 214
})

average = month_counts.mean()

print(f"Average number of contacts per month: {average:.2f}")
```

In [ ]:
```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

full_df = pd.read_csv('bank-full.csv', delimiter=';')
selected_df = pd.read_csv('selected_customers.csv', index_col=0)

full_df.columns = full_df.columns.str.strip().str.replace('"', '')

full_df['selected'] = 0
full_df.loc[selected_df.index, 'selected'] = 1

plt.figure(figsize=(6, 5))
sns.boxplot(data=full_df, x='selected', y='duration')
plt.title('Duration Distribution by Selection')
plt.xlabel('Selected (1=Yes, 0=No)')
plt.ylabel('duration')

plt.tight_layout()
plt.savefig('duration_selected_boxplot.png', dpi=300)
plt.show()
```

In [ ]:
```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

full_df = pd.read_csv('bank-full.csv', delimiter=';')
selected_df = pd.read_csv('selected_customers.csv', index_col=0)

full_df.columns = full_df.columns.str.strip().str.replace('"', '')

full_df['selected'] = 0
full_df.loc[selected_df.index, 'selected'] = 1

plt.figure(figsize=(6, 5))
sns.boxplot(data=full_df, x='selected', y='balance')
plt.title('Balance Distribution by Selection')
plt.xlabel('Selected (1=Yes, 0=No)')
plt.ylabel('Account Balance')

plt.tight_layout()
plt.savefig('balance_selected_boxplot.png', dpi=300)
plt.show()
```

In [ ]:
```
import pandas as pd
import matplotlib.pyplot as plt
```

```
import seaborn as sns

full_df = pd.read_csv('bank-full.csv', delimiter=';')
selected_df = pd.read_csv('selected_customers.csv', index_col=0)

full_df.columns = full_df.columns.str.strip().str.replace('"', '')

full_df['selected'] = 0
full_df.loc[selected_df.index, 'selected'] = 1

plt.figure(figsize=(6, 5))
sns.boxplot(data=full_df, x='selected', y='pdays')
plt.title('Pdays Distribution by Selection')
plt.xlabel('Selected (1=Yes, 0=No)')
plt.ylabel('Days Since Last Contact')

plt.tight_layout()
plt.savefig('pdays_selected_boxplot.png', dpi=300)
plt.show()
```

In [ ]:

```
import pandas as pd
import matplotlib.pyplot as plt

full_df = pd.read_csv('bank-full.csv', delimiter=';')
selected_df = pd.read_csv('selected_customers.csv', index_col=0)

full_df.columns = full_df.columns.str.strip().str.replace('"', '')

full_df['selected'] = 0
full_df.loc[selected_df.index, 'selected'] = 1

full_df['poutcome_binary'] = full_df['poutcome'].apply(lambda x: 'yes' if x

poutcome_summary = (
    full_df.groupby('selected')['poutcome_binary']
    .value_counts(normalize=True)
    .unstack()
)

color_map = ['gray', 'red']
poutcome_summary[['no', 'yes']].plot(
    kind='bar',
    stacked=True,
    figsize=(6, 5),
    color=color_map
)

plt.title('Poutcome (Success) Proportion by Selection')
plt.xlabel('Selected (0 = No, 1 = Yes)')
plt.ylabel('Proportion')
plt.legend(title='poutcome == success')
plt.tight_layout()
plt.savefig('poutcome_success_barplot_colored.png', dpi=300)
plt.show()
```

In [ ]:
```
import pandas as pd
import matplotlib.pyplot as plt

full_df = pd.read_csv('bank-full.csv', delimiter=';')
selected_df = pd.read_csv('selected_customers.csv', index_col=0)

full_df.columns = full_df.columns.str.strip().str.replace('"', '')

full_df['selected'] = 0
full_df.loc[selected_df.index, 'selected'] = 1

housing_summary = (
    full_df.groupby('selected')['housing']
    .value_counts(normalize=True)
    .unstack()
)

color_map = ['gray', 'green']

housing_summary[['no', 'yes']].plot(
    kind='bar',
    stacked=True,
    figsize=(6, 5),
    color=color_map
)

plt.title('Housing Loan Status by Selection')
plt.xlabel('Selected (0 = No, 1 = Yes)')
plt.ylabel('Proportion')
plt.legend(title='Housing Loan')
plt.tight_layout()
plt.savefig('housing_selected_barplot_colored.png', dpi=300)
plt.show()
```

In [ ]:
```
import pandas as pd
import matplotlib.pyplot as plt

revenue_df = pd.read_csv('bankrevenue.csv')
full_df = pd.read_csv('bank-full.csv', delimiter=';')

revenue_df.columns = revenue_df.columns.str.strip().str.replace('"', '')
full_df.columns = full_df.columns.str.strip().str.replace('"', '')

revenue_df['LOAN'] = revenue_df['LOAN'].astype(str).str.lower().str.strip()
full_df['loan'] = full_df['loan'].astype(str).str.lower().str.strip()

loan_dist_revenue = revenue_df['LOAN'].value_counts(normalize=True)
loan_dist_full = full_df['loan'].value_counts(normalize=True)

loan_dist_df = pd.DataFrame({
    'LOAN': loan_dist_revenue,
    'loan': loan_dist_full
}).fillna(0)

loan_dist_df.plot(kind='bar', figsize=(6, 5), color=['skyblue', 'orange'])
```

```
plt.title('Loan Comparison')
plt.ylabel('Proportion')
plt.xlabel('Loan Status')
plt.xticks(rotation=0)
plt.legend()
plt.tight_layout()
plt.savefig('loan_distribution_comparison.png', dpi=300)
plt.show()
```

In [ ]:

```
import pandas as pd
import matplotlib.pyplot as plt

revenue_df = pd.read_csv('bankrevenue.csv')
full_df = pd.read_csv('bank-full.csv', delimiter=';')

revenue_df.columns = revenue_df.columns.str.strip().str.replace('"', '')
full_df.columns = full_df.columns.str.strip().str.replace('"', '')

revenue_df['LOAN'] = revenue_df['LOAN'].astype(str).str.lower().str.strip()
full_df['loan'] = full_df['loan'].astype(str).str.lower().str.strip()

revenue_df['MORT'] = revenue_df['MORT'].astype(str).str.lower().str.strip()
full_df['housing'] = full_df['housing'].astype(str).str.lower().str.strip()

loan_dist_revenue = revenue_df['LOAN'].value_counts(normalize=True)
loan_dist_full = full_df['loan'].value_counts(normalize=True)

mort_dist_revenue = revenue_df['MORT'].value_counts(normalize=True)
mort_dist_full = full_df['housing'].value_counts(normalize=True)

loan_dist_df = pd.DataFrame({'LOAN': loan_dist_revenue,
                             'loan': loan_dist_full}).fillna(0)

mort_dist_df = pd.DataFrame({'MORT': mort_dist_revenue,
                             'housing': mort_dist_full}).fillna(0)

fig, axes = plt.subplots(1, 2, figsize=(12, 5))

loan_dist_df.plot(kind='bar', ax=axes[0], color=['skyblue', 'orange'])
axes[0].set_title('Loan Status Comparison')
axes[0].set_ylabel('Proportion')
axes[0].set_xlabel('Loan Status')
axes[0].set_xticklabels(loan_dist_df.index, rotation=0)
axes[0].legend()

mort_dist_df.plot(kind='bar', ax=axes[1], color=['lightgreen', 'salmon'])
axes[1].set_title('Mortgage Status Comparison')
axes[1].set_ylabel('Proportion')
axes[1].set_xlabel('Mortgage Status')
axes[1].set_xticklabels(mort_dist_df.index, rotation=0)
axes[1].legend()

plt.tight_layout()
plt.savefig('loan_mort_distribution_comparison.png', dpi=300)
plt.show()
```

In [ ]:
```
import pandas as pd
import numpy as np
from scipy.stats import chi2_contingency
import matplotlib.pyplot as plt

revenue_df = pd.read_csv('bankrevenue.csv')
full_df = pd.read_csv('bank-full.csv', delimiter=';')

revenue_df.columns = revenue_df.columns.str.strip().str.replace('"', '')
full_df.columns = full_df.columns.str.strip().str.replace('"', '')

revenue_df['LOAN'] = revenue_df['LOAN'].astype(str).str.lower().str.strip().
full_df['loan'] = full_df['loan'].astype(str).str.lower().str.strip().replac

n_trials = 1000
sample_size = 50
p_values = []

for _ in range(n_trials):
    sampled = full_df.sample(n=sample_size, replace=False, random_state=None

    merged = pd.concat([
        revenue_df[['LOAN']].rename(columns={'LOAN': 'loan'}),
        sampled[['loan']]
    ], keys=['revenue', 'full']).reset_index(level=1, drop=True).reset_index

    table = pd.crosstab(merged['index'], merged['loan'])
    _, p, _, _ = chi2_contingency(table)
    p_values.append(p)

plt.hist(p_values, bins=20, edgecolor='black')
plt.title("P-value Distribution for Personal Loan")
plt.xlabel("p-value")
plt.ylabel("Frequency")
plt.tight_layout()
plt.savefig("pvalue_distribution_hist.png", dpi=300)
plt.show()

average_p = np.mean(p_values)
print(average_p)
```

In [ ]:
```
import pandas as pd
import numpy as np
from scipy.stats import chi2_contingency
import matplotlib.pyplot as plt

revenue_df = pd.read_csv('bankrevenue.csv')
full_df = pd.read_csv('bank-full.csv', delimiter=';')

revenue_df.columns = revenue_df.columns.str.strip().str.replace('"', '')
full_df.columns = full_df.columns.str.strip().str.replace('"', '')

revenue_df['MORT'] = revenue_df['MORT'].astype(str).str.lower().str.strip().
full_df['housing'] = full_df['housing'].astype(str).str.lower().str.strip().
```

```
n_trials = 1000
sample_size = 50
p_values = []

for _ in range(n_trials):
    sampled = full_df.sample(n=sample_size, replace=False, random_state=None

    merged = pd.concat([
        revenue_df[['MORT']].rename(columns={'MORT': 'housing'}),
        sampled[['housing']]
    ], keys=['revenue', 'full']).reset_index(level=1, drop=True).reset_index

    table = pd.crosstab(merged['index'], merged['housing'])
    _, p, _, _ = chi2_contingency(table)
    p_values.append(p)

plt.hist(p_values, bins=20, edgecolor='black')
plt.title("P-value Distribution from Housing Loan")
plt.xlabel("p-value")
plt.ylabel("Frequency")
plt.tight_layout()
plt.savefig("pvalue_distribution_mortgage_hist.png", dpi=300)
plt.show()

average_p = np.mean(p_values)
print(average_p)
```