Predicting the Success of Crowdfunding Projects in Turkey with Supervised Machine Learning

Seminar Paper

submitted to
Prof. Dr. Kevin Bauer
Chair of Game-Theoretic and Causal Artificial Intelligence
in Business and Economics
Faculty of Economics and Business Administration
Goethe University Frankfurt am Main

Supervisor: Prof. Dr. Kevin Bauer

Submitted by: First Name Last Name: Marc Günter Dörsam
Student-ID: 8021533
E-Mail: s8840817@stud.uni-frankfurt.de

First Name Last Name: Ngoc Khanh Uyen Tran
Student-ID: 8518013
E-Mail: s0657641@stud.uni-frankfurt.de

Contents

Preface

This work was developed collaboratively by

Marc Günter Dörsam and Ngoc Khanh Uyen Tran.

Parts written by Marc Günter Dörsam:

4. Methods
5. Performance Evaluation
7. Conclusion

Parts written by Ngoc Khanh Uyen Tran:

1. Introduction
2. Literature Review
3. Data Description and Preprocessing
6. Business Implications

List of figures

List of tables

List of abbreviations

LogReg:	Logistic Regression
SVM:	Support Vector Machine
DT:	Decision Tree
RF:	Random Forests
XGB:	XGBoost
AUC:	Area under the curve

1 Introduction

Crowdfunding refers to a fundraising mechanism that enables project owners' to raise capital by collecting funds or donations from a large number of individuals rather than specialized investors, which is commonly done through online platforms. This practice has transformed the entrepreneurial landscape by democratizing access to financial resources and bypassing traditional financial approaches to funding (Mollick, 2014). The crowdfunding ecosystem offers benefits to both entrepreneurs seeking capital and investors looking to participate in projects, creating new opportunities for financial venturing that were previously unavailable through conventional channels, and letting the "crowd" be involved in these projects, either as consumers or investors, or both (Belleflamme et al., 2014).

Over the last decade, the crowdfunding market has experienced significant growth and is considered as a revolution for fundraising approaches. According to Fortune Business Insights, it is projected to grow from 1.83 billion dollars in 2025 to 4.45 billion dollars by 2032 with a compound annual growth rate (CAGR) of 13.5%. However, in contrast to the rapid expansion, the number of actually successful projects remain low across platforms (Kilinc & Aydin, 2023). For instance, Kickstarter's success rate was approximately 37.45% as of December 2019 (Yeh & Chen, 2022). Specifically, with regard to the Turkish crowdfunding market, the most common platforms such as Fongogo, CrowdFon, and FonlaBeni have witnessed declining success rates despite increasing project volumes. Fongogo's success rate decreased from 30% in 2017 to 26.6% in 2021, even though the number of projects increased approximately fivefold during that time (Kilinc & Aydin, 2023). This declining trend underscores the critical need for predictive mechanisms to effectively identify and understand the factors that contribute to a successful crowdfunding project before launching (Greenberg et al., 2013).

The research in this paper aims to address two fundamental questions: (i) *"Is it possible to predict crowdfunding project outcomes before being launched using supervised machine learning algorithms?"* and (ii) *"What are the most relevant factors that influence project success?"*. To answer these questions, the research is structured as follows: Firstly, section 2 presents the existing studies on crowdfunding success factors and machine learning applications. In section 3, the chosen dataset is described and preprocessed using relevant techniques for redundancy and data leakage, data skewness, scaling numerical variables and encoding categorical variables. This preprocessed dataset is used with the in section 4 described supervised learning algorithms, including logistic regression, support vector machine, decision tree, random forests and XGBoost for binary classification of the crowdfunding projects success. In section 5, the results are evaluated and compared across models to identify the best-performing one. Section 6 and 7 discuss the business implications for stakeholders involved in the crowdfunding ecosystem, as well as giving the outlook on the predictive model.

2 Literature Review

Towards the topic of predicting the success of crowdfunding projects, the existing studies have predominantly focused on binary classification, treating project outcomes as binary variables (successful vs. unsuccessful). Early studies implemented traditional statistical methods, with Li et al. (2016) utilizing logistic regression to identify key factors of Kickstarter project success. Etter et al. (2013), who pioneered the implementation of ensemble methods, has demonstrated that random forest algorithms could effectively predict the crowdfunding success outcome and factors better than other approaches. Greenberg et al. (2013) extended the methodologies by implementing support vector machines and neural networks, showing that different algorithms exhibit varying strengths depending on the specific characteristics of the dataset. Recent studies have favored towards deep learning algorithms, with Cheng et al. (2019) indicating that neural networks can effectively work with complex textual and visual features extracted from project descriptions and media. However, one of the fundamental issues with classification-based approach in predictive modelling is class imbalance in target variable, which within the scope of this research means the majority of projects failing to reach their funding targets, especially with the small dataset acquired from crowdfunding platform (Kilinc & Aydin, 2023). It is characterized by Zhang et al. (2014) as the context where successful projects represent the less frequent but more valuable outcome to predict. This imbalance necessitates specialized approaches in machine learning applications, as traditional classification algorithms tend to exhibit bias toward the majority class, resulting in models that may achieve high overall accuracy by simply predicting the majority class, while failing to identify the minority class of successful projects (David and Goadrich, 2006; He and Garcia, 2009).

On scientific level, forecasting crowdfunding outcomes based on project features have been researched in multiple disciplines, including entrepreneurship theory, social psychology, and information systems. For instance, Courtney et al. (2017) provides a framework regarding how project characteristics influence the observation of project quality and viability to potential backers. According to this research, features such as project descriptions, visual content, and project owners' credentials are considered as signals that reduce information asymmetry between project owners' and backers. On the other hand, social proof theory argues that early funding incentives and backer engagement could determine subsequent decisions (Cialdini, 2009). From resource-based perspective, creator characteristics and network effects are recognized as predictors for projects outcomes (Barney, 1991). When it comes to applied machine learning algorithms, a critical challenge in crowdfunding success prediction involves data leakage, i.e. the use of data that are only available only after launching projects for model training. Many approaches often incorporate features such as number of backers, funding achieved, or social media engagement metrics without carefully consider their

availability at which stage of the project, which forms a severe issue in terms of the credibility of predictive model for decision-making in pre-launch phase (Kaufman et al., 2011). To address this fundamental issue, researchers have developed two-phase modelling approaches that separate pre-launch features from dynamic project features. The proposed framework differentiates between static project characteristics available at launch and evolving metrics that emerge during the project lifecycle (Silva et el., 2020). The methodology demonstrates that, even though incorporating features that are available after launching provide superior predictive power, they offer very limited values for project owners' in pre-launch phase. Meanwhile, static feature models offer more practical application for pre-launch decision-making in actual business context. This tension between feature availability and predictive accuracy has driven recent research toward developing predictive modelling frameworks that account for information availability constraints.

To evaluate the performance of predictive models for crowdfunding success, recent studies have shown that results vary significantly based on dataset characteristics, feature engineering approaches, and evaluation methodologies. Yuan et al. (2016) reported F1-scores ranging from 0.67 to 0.95 using ensemble methods on dataset acquired from two crowdfunding platforms, while incorporating textual sentiment analysis and temporal dynamics. Gradient boosting methods have particularly demonstrated effectiveness in handling the complex features of crowdfunding data. Chen and Guestrin (2016) showed that XGBoost algorithms could achieve significant performance compared to traditional methods while providing interpretable feature importance. Their approach demonstrated that XGBoost has the ability to handle over 1 billion data with less than five machines on a dataset that has been preprocessed for handling sparse data. Deep learning algorithms have also shown the potential in integrating diverse data modalities, with convolutional neural networks effectively processing visual project elements and recurrent neural networks capturing temporal dynamics. However, these sophisticated methods often require advanced computational resources and may suffer from overfitting when applied to relatively small crowdfunding datasets (Krizhevsky et al., 2012).

3 Data Description and Preprocessing

The dataset used in this study contains data on crowdfunding projects in Turkey between 2014 and 2020. Dataset provides information on 1628 projects, with 38 features collected regarding project owners' characteristics and project elements in both pre-launch and post-launch phases. Initial data preprocessing involves removing variables that provide no predictive information (project ID and name, project owner's name, project description) and eliminating features that would introduce data leakage by containing information unavailable before the launch of project (target amount percentage achieved, collected target amount, FAQ, number of comments or updates and social media followers).

Additionally, features with predominantly zero values (number of projects the owner has subscribed to, number of projects owned by the project owner) were removed to reduce noise in the dataset.

The dataset appears to have a significant class imbalance in target variable, with only 376 successful projects and 1252 projects failing to reach their targets, respectively, around 23% and 77%. However, in this study, techniques to address class imbalance were not applied as considering: First, on average, features show low significance towards target variable in terms of correlation and mutual information for classification, with approximately 0.08 average correlation coefficient and 0.012 average mutual information score across all features. Second, given that the dataset is relatively small and consist mostly categorical features, implementing oversampling techniques such as SMOTE could introduce unrealistic feature combinations that do not exist in real-life crowdfunding context. This issue is especially more severe in small dataset, like the one used in this study, as new data points might misrepresent the actual data patterns, which potentially leads to overfitting and poor model generalization. Therefore, the predictive model in this study was built relying on balanced class weights during model training to adjust the learning algorithm's sensitivity to minority class instances without manipulating the data distribution, as well as combined with hyperparameter tuning and cross-validation to ensure robust performance evaluation while preserving the characteristics of the original dataset.

Missing values in this dataset were identified as belonging mostly to project start and end dates. Since project duration was already available as a feature, the start and end date variables were removed from the analysis to avoid redundancy. Additionally, there was also missing value in the project location attribute. However, this attribute is considered to be closely overlapped with the region variable, which might lead to multicollinearity. To maintain dimensionality and avoid unnecessary complexity of the dataset, the location feature was removed.

Preprocessing also handles the huge imbalance between instances among information by dividing them into category groups that share the same characteristics, aiming to support the model performance with balanced distribution. For instance, among 17 project categories, “creative” group was created to include categories such as photography, culture art, design; meanwhile, “lifestyle” group was created combining sports, tourism, health and beauty categories. For crowdfunding platforms, besides the major values such as Fongogo, Crowdfon, Fonbulucu, the rest of the funding platforms were grouped into “other” category. Accordingly, the same grouping method was also applied to region feature, where the most dominant regions remained as separate categories, while the rest was grouped as “other” to reduce data sparsity (Table I in Appendix shows the final features selection).

Predictive models in this study are developed by building classification rules for the target variable “success”. In this research, five classification algorithms were chosen to represent a wide range of

approaches to predict the outcome of projects. In order to train classifiers, select their parameters and evaluate their performance, the dataset was randomly divided into two parts: 80% of the projects are selected as the training set, and the other 20% as the validation set to evaluate the "out-of-sample" performance of the models. This separation was done before data transformation to prevent data leakage issue, ensuring the training and test sets remain independent and have no mutual information.

Numerical features of the dataset appear to have significant skewness; however, logarithmic transformations could not be applied due to the presence of zero values. Instead, the Yeo-Johnson power transformation was applied to address this issue, as it accommodates both positive and negative values, making it suitable for datasets containing zero values where alternative approach such as log transformation would fail (Yeo & Johnson, 2000). The transformation was applied only to features with absolute skewness values exceeding 1, because existing studies identify such values as indicative of substantial departure from normality requiring transformation (Yeo & Johnson, 2000; Osborne, 2010). Binary features were excluded from transformation as they follow a Bernoulli distribution, which are not suitable for data transformation (Kuhn & Johnson, 2013). Excluding binary variables, numerical features were also standardized using StandardScaler to ensure comparable scales across features. This preprocessing step is important for distance-based algorithms such as logistic regression and support vector machine, which are sensitive to feature magnitudes. While tree-based algorithms are scale-invariant, standardization does not negatively impact their performance yet ensures consistency across different model types. Fundamentally, this transformation centers the distribution at zero with unit variance, facilitating optimal convergence for gradient-based optimization algorithms by improving numerical stability (Jain et al., 2000; Géron, 2017, Lemaitre et al., 2017).

For categorical variables, they were encoded using one-hot encoding, which is also referred to as dummy encoding, with the first category used as the reference level. This approach was chosen over ordinal encoding as distance-based algorithms interpret numerical relationships between categories, which could introduce artificial ordinality where none exists (Kuhn & Johnson, 2013). One-hot encoding creates binary indicator variables for each category level, transforming a categorical variable with k categories into k-1 binary variables, with the omitted category serving as the baseline (James et al., 2022). This encoding is appropriate, especially for support vector machine model, as it maintains the categorical nature of the data without introducing false numerical relationships.

Overall, after preprocessing and feature engineering processes, the dataset in the final has 29 features, compared to nearly 40 features at the beginning. For model training and evaluation, the training set contains of 1239 samples and the test set contains of 326 samples. Duplications and multicollinearity have also been checked and returned with no results on the training data.

4 Methods

4.1 Supervised Machine Learning Models

In the following section 4.1 the five supervised machine learning models are presented which are suitable for binary classification. They are widely used in the literature, not only in the field of crowdfunding (Kilinc & Aydin, 2023; Oduro et al., 2022; Raflesia et al., 2023), but also in the fields of fraud detection (Afriyie et al., 2023; Shen et al., 2007) or medicine (Podgorelec et al., 2002).

4.1.1 Logistic Regression

Logistic regression (LogReg) is a widely used model, which directly models the probability of classes through a simple, interpretable and linear decision boundary using the sigmoid function (James et al., 2022), shown in equation (1):

$$P(Y = 1 \mid X) = \frac{e^{\beta_0 + \sum_{j=1}^{p} \beta_j X_j}}{1 + e^{\beta_0 + \sum_{j=1}^{p} \beta_j X_j}} \quad (1)$$

if $P(Y = 1 \mid X) \geq 0.5$, the predicted outcome is $\hat{y} = 1$ and otherwise $\hat{y} = 0$ (Géron, 2017).

The parameters β are estimated by minimizing the negative log-likelihood function, shown in equation (2), where the penalized terms L1: $-\lambda \sum_{j=1}^{p} |\beta_j|$ (LASSO) or L2: $-\lambda \sum_{j=1}^{p} \beta_j^2$ (Ridge) can be added to shrink or eliminate irrelevant coefficients to prevent overfitting (Hastie et al., 2009).

$$\mathcal{L}(\beta) = -\sum_{i=1}^{N} \{ y_i \log P(Y = 1 \mid X = x_i) + (1 - y_i) \log\big(1 - P(Y = 1 \mid X = x_i)\big) \} \quad (2)$$

The coefficients can be used through the sign and size of β for the interpretation of feature importance. Additionally, LogReg is computationally efficient and works well with small datasets. Since it is a strictly linear model, one limitation is that it has poor performance on complex non-linear data and then requires feature engineering like adding polynomials (Hosmer & Lemeshow, 2000).

4.1.2 Support Vector Machines

In comparison, the support vector machine (SVM) can not only capture linear but also non-linear structures using the so-called kernel trick. The kernel trick enables SVMs to handle non-linear data by computing relationships in a higher-dimensional space without explicit transformation. SVMs classifies by finding the optimal hyperplane that maximizes the margin between two classes using equation (3) and then uses the resulting parameters to make a prediction with equation (4) (Hastie et al., 2009):

$$\max_{\alpha} \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{i'=1}^{n} \alpha_i \alpha_{i'} y_i y_{i'} K(x, x') \quad \text{subject to } 0 \leq \alpha_i \leq C, \sum_{i=1}^{n} \alpha_i y_i = 0 \tag{3}$$

where α are Lagrange multipliers, y are the class labels, $K(x, x')$ is the kernel function and C regularizes the margin width against the classification error.

$$\hat{y}_{SVM} = \sum_{i=1}^{N} \alpha_i y_i K(x, x') + \beta_0 \tag{4}$$

SVMs perform particularly well when classes are clearly separable with a large margin. As the model wants to maximize the margin between classes, it is a distance-based model. Distance-based models require essential preprocessing steps like feature scaling and outlier handling to avoid misfitting. Additionally, the results are often difficult to interpret if there is no linear hyperplane (Géron, 2017).

4.1.3 Tree Models

Compared to SVMs, decision trees (Breiman et al., 1984) offer a much higher interpretability. A decision tree (DT) consists of a hierarchical structure with a root node at the top, followed by internal split nodes and ending with leaf nodes that represent class predictions. In a classification problem the optimal split at each node is commonly calculated with impurity measures such as the. $GINI\ index = 1 - \sum_{i=1}^{N} (p_i)^2$ (Breiman et al., 1984) or the $Entropy = -\sum_{i=1}^{N} p_i * \log_2(p_i)$ (Quinlan, 1986), where p_i is the proportion belonging to class i. A lower value of the impurity measure indicates a split that better separates classes, with zero being the best and all samples belong to a single class. Classifications are made by assigning the majority class in a leaf node as shown in equation (5) by Hastie et al. (2009):

$$\hat{y}_{DT} = \arg \max_{k \in \{0,1\}} \left(\frac{1}{N_m} \sum_{x_i \in R_m} I(y_i = k) \right) \tag{5}$$

where x is the input sample, R_m is the leaf node region containing x, N_m is the number of training samples in the leaf node region R_m and $\hat{y}_{DT}$ is the predicated class label.

Despite their simplicity, DTs offer several advantages. DTs are easy to compute, visualize and interpret, even for people without technical knowledge. Moreover, they can handle numerical as well as categorical features and no scaling or data transformation is required, because trees are built by split criteria. Through these split criteria a direct feature selection is also included. However, trees are sensitive to small input changes which can lead to a completely different tree. They also tend to overfit

and be difficult to interpret with larger size, e.g. if there are categorical features with many levels, the model wants to represent these structures leading to overfitting (Géron, 2017; Hastie et al., 2009).

Random forests (Breiman, 2001) effectively mitigate this overfitting tendency of decision trees. Random forests (RF) is an ensemble method that constructs multiple independent decision trees using bootstrapped subsets of the data with a technique known as 'bagging'. Hastie et al. (2009) present the classification determined by a majority vote of the trees with the highest mode by equation (6):

$$\hat{y}_{RF} = \arg\max_{k\in\{0,1\}} \left(\sum_{t=1}^{T} I\left(\widehat{y_{DT}^{t}} = k\right) \right) \tag{6}$$

where T is total number of trees in the ensemble, $\widehat{y_{DT}^{t}}$ is the prediction of the t-th tree as defined in equation (5) and $\widehat{y_{\mathrm{RF}}}$ is the predication based on majority vote.

Breiman (2001) emphasizes that RF benefit in classifications from the law of large numbers, since the bagging technique with many trees introduces randomness and ensures that individual trees are mostly uncorrelated. More randomness can be achieved by reducing the number of features and/or the subsample size. This helps that individual errors cancel out through averaging, leading to more stable predictions and reduced overfitting. Additional advantages are that trees can be trained in parallelization for more efficiency and feature importance can be analysed. A limitation of RF is that, due to the increased complexity, it is more difficult to interpret than a single DT. Moreover, Hastie et al. (2009) point out that RF have problems with datasets with many irrelevant features. Irrelevant features reduce the probability of selecting informative features resulting in a limited performance.

Another ensemble method which combines several weak leaners is gradient boosting. Here eXtreme Gradient Boosting that is also called XGBoost (T. Chen & Guestrin, 2016) is a more optimized version of this algorithm. XGBoost (XGB) starts with a very simple decision tree and builds a classification model by iteratively adding new decision trees. Each new tree corrects the prediction errors of the previous tree by fitting the negative gradient of the loss function and is built to minimize the regularized loss function, which is a second-order Taylor approximation, shown in equation (7):

$$\mathcal{L} = \sum_{i=1}^{n} \left[g_i f_t(x_i) + \frac{1}{2} h_i f_t^2(x_i) \right] + \gamma T + \frac{1}{2} \lambda \sum_{j=1}^{T} w_j^2 \tag{7}$$

where g_i is the gradient, h_i is the Hessian, γT is the tree complexity and $\frac{1}{2}\lambda \sum_{j=1}^{T} w_j^2$ is the regularization on the leaf weights.

A prediction is made using the entire ensemble, like in RF. The input is passed through each tree and lands in exactly in one leaf. Each leaf has an associated score, which was calculated while training and represents how strongly this path contributes to the prediction outcome. Explicitly, the score represents log-odds $= log\left(\frac{P(y=1)}{1-P(y=1)}\right)$. A positive log-odds score indicates a tendency to the positive class while a negative value indicates a tendency to the negative class, where a larger value is a stronger tendency. The final prediction is obtained by summing up each score of each leaf node across all trees and passing the total log-odds score through the sigmoid function. The sigmoid function converts the total score into a probability between 0 and 1 (Hastie et al., 2009; Wade & Glynn, 2020)

Chen & Guestrin (2016) explain that through this process the XGBoost model has a very high accuracy and overfitting can be reduced through the integrated regularization. Additionally, it is robust against outliers, because weak learners are combined to an ensemble. Furthermore, as with RF, the feature importance can be explained but with a lower interpretability of the model due to the higher complexity. Hyperparameters also have to be carefully selected to avoid overfitting.

4.2 Implementation and Hyperparameter Tuning

In this section the process of model implementation and hyperparameter tuning followed by the instructions of Géron (2017) is described. Firstly, the dataset was divided into a training set to train the models and a test set for testing the model performance. Typical splits are 70% or 80% for training and 30% or 20% for testing. Géron (2017) showed a stratified split of 80:20, which was also used in the model development of this study. Moreover, the stratification guaranteed that in both subsets the target variable ‘success’ had the same distribution. This is important due to the low number of successful projects. Additionally, a bit more of training data (80% vs. 70%) ensured that the models had more successful cases to learn patterns while training.

Stratified cross-validation was used while hyperparameter tuning. Hyperparameter tuning is an important step while developing a machine learning model. Models often cannot catch specific structures in the data if they are trained with default parameters which leads to over-/underfitting. The stratified cross-validation divides the training set in k equal distributed folds. K-1 folds are used for training and one fold is used for the validation of the model performance, with each set being the validation set once. This method makes it possible to create more stable models by using data more efficiently, which is essential with a small dataset. Typical values for k are three, five or ten whereby five folds were chosen in this study to ensure that enough successful projects were in the validation sets. Ten folds would have been too few successful cases per fold and three folds tend to reduce result stability.

Standard methods for hyperparameter tuning presented by Géron (2017) are RandomizedSearch or GridSearch. For example, Gridsearch tests every possible combination of the specified parameter region which often makes it very slow. A more efficient method for hyperparameter tuning used in this study is Optuna (Akiba et al., 2019). Optuna is an open-source library with a highly efficient algorithm that searches through a defined hyperparameter region using Bayes optimization. Optuna continuously updates the probability distribution of the search space based on previous results on the validation set to focus only on the most promising regions. During a predefined number of trials, a model was trained with different parameter and evaluated on the validation set. Here, 200 trials were chosen to find optimal parameters because more trials showed no improvement. Within Optuna an objective function was maximized. In this study, the mean F1-score across all validation folds was maximized, as it represents the harmonic mean of precision and recall making it a more suitable metric to identify positive cases in an imbalanced dataset than, e.g. accuracy (Kilinc & Aydin, 2023; Oduro et al., 2022).

The hyperparameter spaces were set as recommended by Géron (2017) to obtain a regularized model. LogReg and SVM have a C parameter which is the reciprocal of the regularization strength. This C parameter was set for both models to a range of 0.01 to 1, because a smaller value leads to more regularization and less overfitting. Within the tree models, the depth of the tree and the minimum number of samples per leaf or for a split were defined to control the model complexity. For DT and XGB a max_depth of 3 to 5 was selected to not allow for too much complexity. A depth of 8 to 12 was selected for RF, because RF can handle more complexity, as classification is based on a majority vote. Furthermore, RF included parameters such as the number of trees (350 to 500), the number of features (0.3 to 0.5) and the proportion of the subsamples (0.8 to 0.9) to increase randomness. For XGB the learning rate (0.05 to 0.15), the number of trees (50 to 150), the subsampling parameters (subsample 0.5 to 0.7 and colsample_bytree 0.5 to 0.7) and the regularization terms (reg_alpha 0.1 to 1 and reg_lamba 5 to 20) were set to balance bias and variance. Dataset imbalance was addressed across all models by setting the class_weight parameter to ‘balanced’ (or in the case of XGB, scale_pos_weight to the ratio between the negative and positive class). This approach assigns weights that are inversely proportional to class frequencies so that minority classes have stronger influence while training (Asundi et al., 2022).

The final model was trained with the most promising parameters from Optuna on the entire training set and evaluated on the test set. The evaluation of performance on the test set was based on standard metrics used in the literature, whereby more attention was paid to F1, precision and recall, due to the imbalanced dataset and to identify positive (successful) cases (Géron, 2017; Kilinc & Aydin, 2023; Oduro et al., 2022).

5 Performance Evaluation

Table 1 presents the performances of the different models. LogReg and SVM achieved the same accuracy of 80%. An accuracy of 80% means that 80% of successful and unsuccessful projects were classified correctly. Similarly, both models reached a specificity of 80%, meaning that 80% of the unsuccessful projects were correctly classified as unsuccessful. Comparing recall, precision and F1-score of these both models, SVM performed slightly better. Both models realized a recall of 81%, showing that 81% of the successful projects were classified correctly as successful. Precision gives an insight into the correctness of the positive predictions. Here, SVM slightly outperformed LogReg with a precision of 55% vs. 54%, indicating that 55% of the projects classified as successful were indeed successful. This resulted in a higher F1-score of 0.66 vs. 0.65 for SVM.

	LogReg	SVM	DT	RF	XGB
Accuracy	0.80	0.80	0.74	0.83	0.83
Specificity	0.80	0.80	0.73	0.86	0.84
Recall/Sensitivity	0.81	0.81	0.76	0.73	0.79
Precision	0.54	0.55	0.46	0.61	0.60
F1-Score	0.65	0.66	0.57	0.67	0.68
AUC	0.87	0.88	0.82	0.89	0.90

Table 1: Results of the different supervised machine learning methods.

The area under the curve (AUC) tells how good a model separates between classes. Both models reached a score above 0.80 (LogReg: 0.87; SVM: 0.88) which is an excellent performance (Hosmer & Lemeshow, 2000). The lowest AUC of 0.82, shown in Table 1, was recorded for DT. This can also be seen in Figure 1, where the ROC curve clearly stands out from the others.

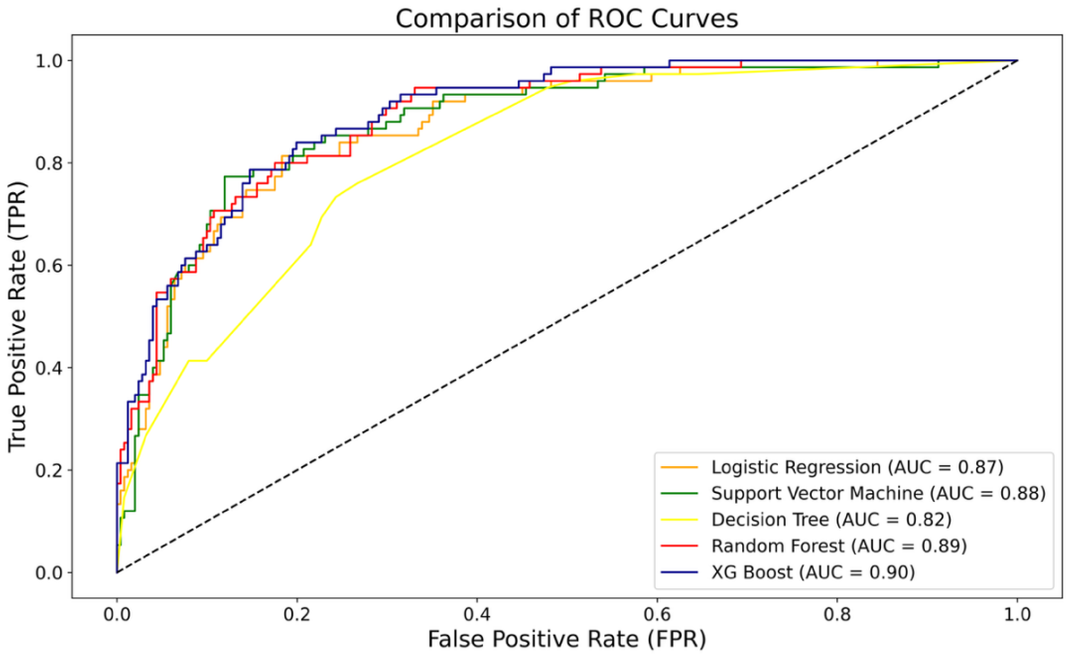

Figure 1: ROC curves comparison of the different models.

The other metrics also showed that simple DT had the worst performance metrices compared to the other models. DT only had an accuracy of 74%, which is worse than if the model had always predicted unsuccessful (77%), due to the imbalanced data. The other metrics were also worse than both LogReg and SVM with specificity: 73%, recall: 76%, precision: 46% and the lowest F1-score of 0.57.

Among all models, both ensemble methods had the best performances. Both models achieved the highest accuracy of 83%. Furthermore, RF achieved a higher F1-score of 0.67 than the previous three models and XGB had even reached the highest F1-score of 0.68. This F1 score for XGB is based on a slightly lower precision of 60% compared to 61% for RF and a higher recall of 79% compared to RF's recall of 73%. Regarding to specificity, RF was slightly better able to identify unsuccessful projects with 86% compared to XGB with 84%. XGB, otherwise, was better at separating the classes and achieved an outstanding result of an AUC of 0.90 vs 0.89 for RF (Hosmer & Lemeshow, 2000).

Based on the presented results, it can be said that XGB was the best of the models due to its high accuracy and both the highest F1 score as well as the highest AUC. Therefore, it is analysed in more detail below. Compared to other results in the literature, Kilinc & Aydin (2023) analysed almost the same dataset, but the performance is quite different. In their analyses they had much higher F1-, recall- and precision-scores. One explanation for this is, they included post-launch features, e.g. number of social media followers or number of updates. These might predict the outcome, because they are collected after start and can change over time which resulted in higher scores. Raflesia et al. (2023) analysed a much larger and balanced dataset with largely the same models. Their results are consistent with those of this analysis. XGB had in their analysis an accuracy of 86% and a F1-score of 67%. However, it should be mentioned here that the XGB of Raflesia et al. (2023) achieved a much higher precision of 77% compared with the precision of the XGB in this study with 60%.

The confusion matrix in Figure 2 shows that XGB struggled to correctly identify positive cases.

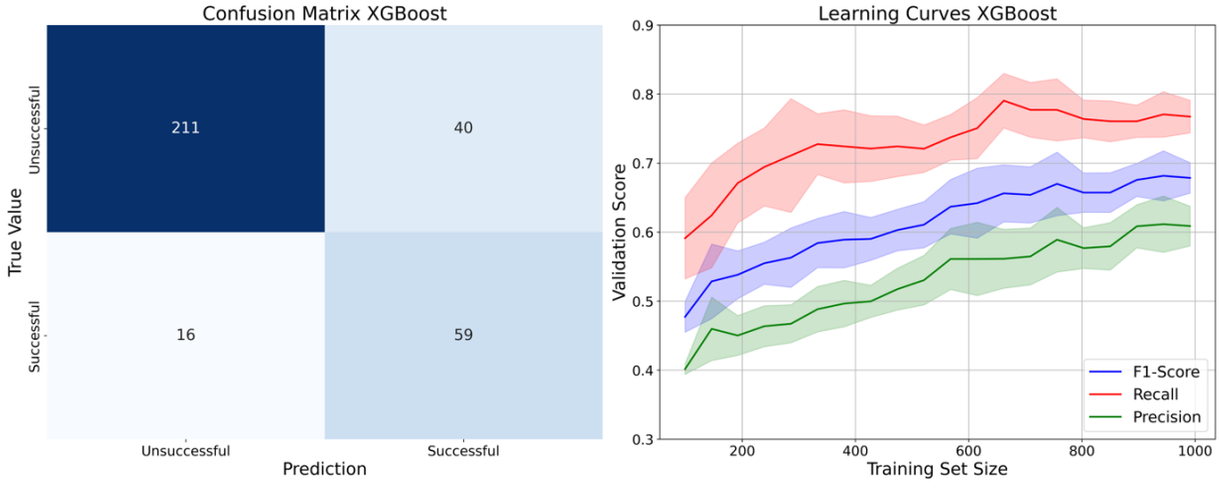

Figure 1: Confusion matrix and the learning curves of the XGBoost model.

The model predicted 40 cases falsely and 59 correctly as successful. The learning curves on the mean validation scores of the tuned model in Figure 2 show that especially precision and the F1 score increased significantly with an increasing training size, whereas recall tends to stagnate and flatten out slightly. From this it can be concluded that performance would increase significantly with a larger amount of data. Furthermore, the mean validation F1-score had a low standard deviation of just under 2% with a mean of 0.68. This is close to the test F1-score of 0.68, reflecting the stability of the model.

Figure 3 shows the SHAP feature importance of the XGB model. Higher values of number of backed projects and promotion video length had a high impact on the prediction outcome. A one-sided t-test confirmed that the average promotion video length for successful projects is significantly larger than for unsuccessful projects (t-value = 7.60, p-value < 0.01). Additionally, it is important if there is a website or promotion video. This also confirms the results of K. Chen et al. (2013) and Kamath, R. S., & Kamat, R. K. (2016), who state that a promotion video has an impact on the project success.

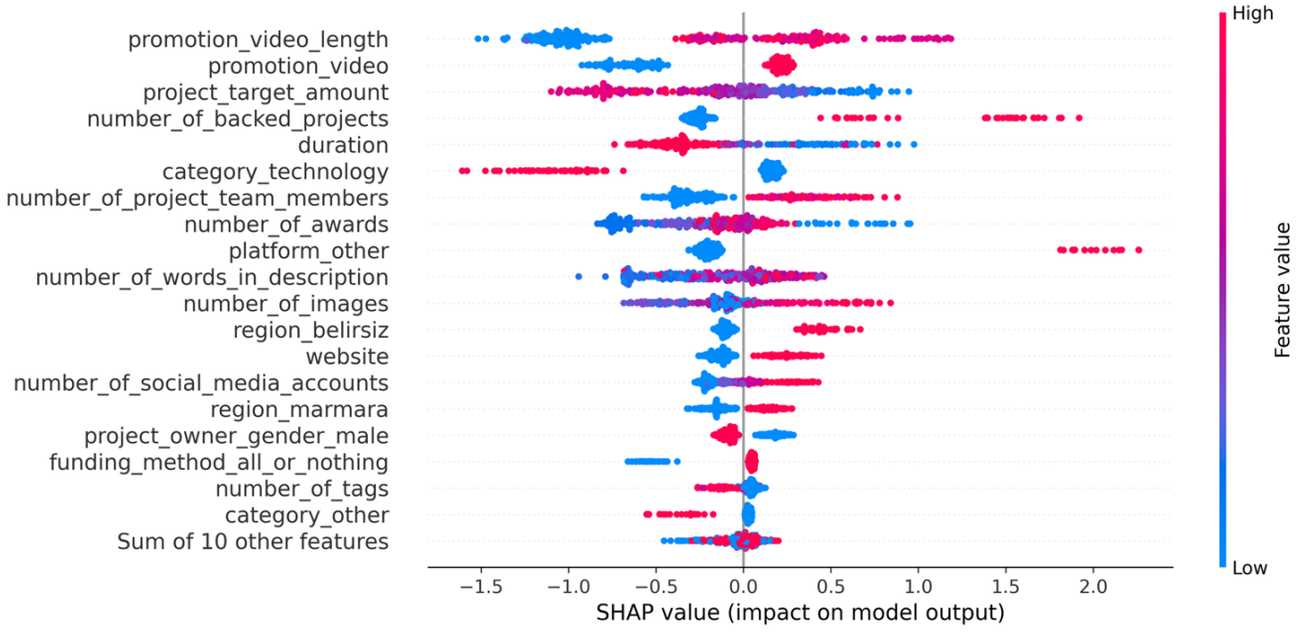

Figure 3: SHAP feature importance of the XGBoost model.

On the other hand, the model predicted for lower values of the project target amount or duration a successful outcome. A one-sided t-test showed that successful projects had a significantly lower average duration (t-value = -2.99; p-value < 0.01) and a significantly lower project target amount (t-value = -1.44; p-value < 0.1) than unsuccessful projects.

An analysis to identify the causes of misclassification showed that the model had problems with the promotion video. The model incorrectly predicts success of the crowdfunding project when there is a promotion video and especially if there is a very long promotion video. This is underlined by the fact that the promotion_video feature also had the highest weight within the model with 0.22.

6 Business Implications

The predictive model developed in this study provides decision-making support by identifying key factors that influence project success, introducing potential trade-offs from business perspectives, and suggesting practical considerations for stakeholders in the Turkish crowdfunding ecosystem.

Considering the feature importance analysis using SHAP values, promotion video length and its presence are the most influential factors determining project success. Both the presence and the length of a promotional video are associated with higher predicted success, underlining the importance of professional visual communication in crowdfunding. This can be understood as projects with longer and more professional promotional videos have a higher likelihood of success. However, looking from the perspective of project owners, creating high-quality promotion videos usually requires professional videographers, scriptwriters, editors and media equipments, which lead to significant upfront costs. These costs may, in turn, lead to higher funding targets that reduce the likelihood of success, because projects with lower targets generally have better success rate. Similarly, maintaining a decent promotion website demands ongoing investment in web development and maintenance, as well as requiring technical expertise and personnel that many early-stage or low-budget projects lack access to. The positive impact of having a website must be weighed against the operational costs to maintain an effective online presence, which determine the success of a project. The results suggest owners either adjust the budget to invest in professional marketing activities, or accepting lower success probabilities with more modest promotional approaches. Therefore, there is a significant need for strategic planning for media presence and coverage, in terms of both human and financial capital. On the other hand, crowdfunding platforms might consider offering video production or website creation services as additional packages for project owners.

From the investors' perspective, due diligence become a quality indicator for assessing project potential. As evaluating promotion videos, review websites and checking media presence are time-consuming, it accordingly raises the standard of whether a project is worth to invest in. While features like number of backed projects and team size provide social proof, the effectiveness of online communication from projects, even though requires significant research effort, often returns with better trust and credibility. However, the confusion matrix for XGBoost have 40 false positives and 16 false negatives, indicating that some projects with promotional videos were predicted successful but ultimately failed, highlighting the importance of substantive and meaningful. As superficial promotion materials without solid usefulness and real value can mislead the investors, emphasizing the importance of deeper analysis beyond production quality metrics.

7 Conclusion

This study showed that the success of crowdfunding projects in Turkey can be predicted before being launched using supervised machine learning algorithms. Building upon existing research that focused on binary classification approaches, this study addressed the issue of class imbalance within target variable, while also avoiding data leakage problem with using data that is not available during pre-launch phase. The dataset of 1628 Turkish crowdfunding projects from 2014 to 2020 was preprocessed with features removing, skewness transformation, standardization and dummy encoding, making it suitable for binary classification. Among the five models (logistic regression, support vector machine, decision trees and two ensemble models random forests and XGBoost), XGBoost achieved the highest accuracy of 83%, F1-score of 0.68 and AUC of 0.90.

The feature importance analysis with SHAP identified promotion video as well as the promotion video length, funding targets, project duration, number of backed projects and the presence of a website as influential success factors. Thus, the model offers for both investors and project owners concrete recommendations for more successful crowdfunding projects. For investors, the model enables data-driven decisions by focusing on projects with professional public presence, such as professional promotion videos or attractive website and realistic project targets. Therefore, project owners should prioritize high-quality video production, strategic target and optimal project durations. The study also highlights a paradox in the crowdfunding market: success is tied to the ability to invest in online visibility and professionalism, but on the other hand, media efforts also come with a significant cost for project owners.

However, all models struggled with precision issues due to many false positive predictions, with XGBoost particularly affected by the presence and length of a promotion video. The learning curve of the XGBoost showed that with increasing data size both precision and F1-Score drastically improved while reducing false positives. This indicates that the model is far from reaching its full potential due to the small data set in this analysis. Similarly, other models such as e.g. neural nets were not suitable due to the small data set and therefore not tested.

Further research has to analyse how the models perform with more data and how other more complex models perform, which might capture other or more complex patterns. This may bring enormous advantage due to the detection of complex patterns in the data, resulting in more reliable and powerful models, from which both, investors and project owner, would significantly benefit. Moreover, it is essential to predict not only the success in the Turkish crowdfunding market, but also for global crowdfunding ecosystem.

References

Afriyie, J. K., Tawiah, K., Pels, W. A., Addai-Henne, S., Dwamena, H. A., Owiredu, E. O., Ayeh, S. A., & Eshun, J. (2023). A supervised machine learning algorithm for detecting and predicting fraud in credit card transactions. *Decision Analytics Journal*, *6*, 100163.

Akiba, T., Sano, S., Yanase, T., Ohta, T., & Koyama, M. (2019). Optuna: A Next-generation Hyperparameter Optimization Framework. In *Proceedings of the 25th ACM SIGKDD international conference on knowledge discovery and data mining.*

Asundi, R. V., Prakash, R., Kumar, K., & India, B. (2022). Class weight technique for handling class imbalance. *BMS Institute of Technology and Management*.

Barney, J. (1991). Firm resources and sustained competitive advantage. *Journal of Management*, 17(1), 99-120.

Belleflamme, P., Lambert, T., & Schwienbacher, A. (2014). Crowdfunding: Tapping the right crowd. *Journal of Business Venturing*, 29(5), 585–609.

Breiman, L. (2001). Random Forests. *Machine Learning*, *45*(1), 5–32.

Breiman, L., Friedman, J., Olshen, R. A., & Stone, C. J. (1984). *CART: Classification and Regression Trees*. Wadsworth.

Chen, K., Jones, B., Kim, I., & Schlamp, B. (2013). KickPredict: Predicting Kickstarter Success. *Technical report, California Institute of Technology*.

Chen, T., & Guestrin, C. (2016). XGBoost: A Scalable Tree Boosting System. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 785–794.

Cheng, C., Tan, F., Hou, X., & Wei, Z. (2019, August). Success prediction on crowdfunding with multimodal deep learning. In *Proceedings of International Joint Conference on Artificial Intelligence*, 2158-2164.

Cialdini, R. B. (2009). *Influence: Science and practice*. Pearson Education.

Courtney, C., Dutta, S., & Li, Y. (2017). Resolving information asymmetry: Signaling, endorsement, and crowdfunding success. *Entrepreneurship Theory and Practice*, 41(2), 265-290.

Davis, J., & Goadrich, M. (2006). The relationship between Precision-Recall and ROC curves. *Proceedings of the 23rd International Conference on Machine Learning*, 233-240.

Etter, V., Grossglauser, M., & Thiran, P. (2013). Launch hard or go home! Predicting the success of Kickstarter campaigns. In *Proceedings of the First ACM Conference on Online Social Networks*, 177-182.

Fortune Business Insights. (2024). *Crowdfunding Market Size, Share & Industry Analysis, By Type (Equity-based, Debt-based, Blockchain-based, and Others), By End-user (Startups, NGOs, and Individuals), and Regional Forecast, 2025-2032.* Retrieved June 17, 2025, from https://www.fortunebusinessinsights.com/crowdfunding-market-107129

Géron, A. (2017). *Hands-On Machine Learning with Scikit-Learn and TensorFlow: Concepts, tools, and techniques to build intelligent systems*. O'Reilly Media, Inc.

Goodfellow, I., Bengio, Y., & Courville, A. (2016). Deep Learning. MIT Press.

Greenberg, M. D., Pardo, B., Hariharan, K., & Gerber, E. (2013). Crowdfunding support tools: Predicting success & failure. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. 1815–1820.

Hastie, T., Tibshirani, R., & Friedman, J. (2009). *The Elements of Statistical Learning*. Springer.

He, H., & Garcia, E. A. (2009). Learning from imbalanced data. *IEEE Transactions on Knowledge and Data Engineering*, 21(9), 1263-1284.

Hosmer, D. W., & Lemeshow, S. (2000). *Applied Logistic Regression*. Wiley-Interscience Publication.

Jain, A. K., Duin, R. P. W., & Mao, J. (2000). Statistical pattern recognition: A review. *IEEE Transactions on Pattern Analysis and Machine Intelligence, 22*(1), 4–37.

James, G., Witten, D., Hastie, T., & Tibshirani, R. (2022). *An Introduction to Statistical Learning with Applications in R.* Springer.

Kamath, R. S., & Kamat, R. K. (2016). Supervised learning model for kickstarter campaigns with R mining. *International Journal of Information Technology, Modeling and Computing (IJITMC), 4*(1).

Kaufman, S., Rosset, S., Perlich, C., & Stitelman, O. (2011). Leakage in data mining: Formulation, detection, and avoidance. *ACM Transactions on Knowledge Discovery from Data*, 6(4), 1-21.

Kilinc, M., & Aydin, C. (2023). Feature selection for Turkish Crowdfunding projects with using filtering and wrapping methods. *Electronic Commerce Research and Applications*, *62*, 101340.

Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems*, *25*, 1097-1105.

Kuhn, M., & Johnson, K. (2013). *Applied predictive modeling*. Springer.

Lemaitre, G., Nogueira, F., & Aridas, C. K. (2017). Imbalanced-learn: A Python toolbox to tackle the curse of imbalanced datasets in machine learning. *Journal of Machine Learning Research, 18*(17), 1–5.

Li, Y., Rakesh, V., & Reddy, C. K. (2016. Project success prediction in crowdfunding environments. In *Proceedings of the Ninth ACM International Conference on Web Search and Data Mining*, 247-256.

Mollick, E. (2014). The dynamics of crowdfunding: An exploratory study. *Journal of Business Venturing*, 29(1), 1–16.

Oduro, M. S., Yu, H., & Huang, H. (2022). Predicting the Entrepreneurial Success of Crowdfunding Campaigns Using Model-Based Machine Learning Methods. *International Journal of Crowd Science*, *6*(1), 7–16.

Osborne, J. W. (2010). Improving your data transformations: Applying the Box-Cox transformation. *Practical Assessment, Research, and Evaluation, 15*(12), 1–9.

Podgorelec, V., Kokol, P., Stiglic, B., & Rozman, I. (2002). Decision trees: An overview and their use in medicine. *Journal of medical systems*, *26*, 445-463.

Quinlan, J. R. (1986). Induction of decision trees. *Machine Learning*, *1*(1), 81–106.

Raflesia, S. P., Lestarini, D., Kurnia, R. D., & Hardiyanti, D. Y. (2023). Using machine learning approach towards successful crowdfunding prediction. *Bulletin of Electrical Engineering and Informatics*, *12*(4), 2438–2445.

Shen, A., Tong, R., & Deng, Y. (2007). Application of Classification Models on Credit Card Fraud Detection. In *International Conference on Service Systems and Service Management*, 1–4.

Silva, L., Silva, N. F., & Rosa, T. (2020). Success prediction of crowdfunding campaigns: a two-phase modeling. *International Journal of Web Information Systems*, *16*(4), 387-412.

Wade, C., & Glynn, K. (2020). *Hands-On Gradient Boosting with XGBoost and scikit-learn: Perform accessible machine learning and extreme gradient boosting with Python*. Packt Publishing Ltd.

Yeh, J.-Y., & Chen, C.-H. (2022). A machine learning approach to predict the success of crowdfunding fintech project. *Journal of Enterprise Information Management*, *35*(6), 1678-1696.

Yeo, I. K., & Johnson, R. A. (2000). A new family of power transformations to improve normality or symmetry. *Biometrika*, 87(4), 954-959.

Yuan, H., Lau, R. Y., & Xu, W. (2016). The determinants of crowdfunding success: A semantic text analytics approach. *Decision Support Systems*, 91, 67-76.

Zhang, Y., Fu, P., Liu, W., & Chen, G. (2014). Imbalanced data classification based on scaling kernel-based support vector machine. *Neural Computing and Applications*, *25*, 927-935.

Appendix

Feature	Type	Description
platform_adi	Categorical	The crowdfunding platform where the project is hosted
kitle_fonlamasi_turu	Categorical	Type of crowdfunding
kategori	Categorical	Category of the project
fon_sekli	Categorical	Funding method
proje_sahibi_cinsiyet	Categorical	Gender of the project owner
kac_proje_destekledi	Integer	Number of projects the owner has backed
kac_projenin_sahibi	Integer	Number of projects owned by the project owner
bolge	Categorical	Region of the project
gun_sayisi	Numeric	Duration of the project in days
tanitim_videosu	Categorical	Whether the project has a promotional video
video_uzunlugu	Numeric	Length of the promotional video
gorsel_sayisi	Categorical	Number of images related to the project
odul_sayisi	Numeric	Number of rewards offered in the project
ekip_kisi_sayisi	Numeric	Number of people in the project team
web_sitesi	Categorical	Whether the project has a website
sosyal_medya	Categorical	Whether the project has social media accounts
sm_sayisi	Numeric	Number of social media accounts for the project
etiket_sayisi	Numeric	Number of tags used in the project description
icerik_kelime_sayisi	Numeric	Number of words in the project description
hedef_miktari	Numeric	Target amount of funding for the project
basari_durumu	Categorical	Success status of the project (Target variable)

Table I: Selected features to perform supervised machine learning algorithms.

Python Code

```
# Import relevant libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from scipy.stats import ttest_ind

from sklearn.svm import SVC
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.tree import DecisionTreeClassifier
from xgboost import XGBClassifier

from sklearn.metrics import roc_curve, roc_auc_score, confusion_matrix,
make_scorer, f1_score, precision_score, recall_score
from sklearn.model_selection import train_test_split, StratifiedKFold,
cross_val_score, learning_curve
from sklearn.preprocessing import StandardScaler, PowerTransformer
from statsmodels.stats.outliers_influence import variance_inflation_factor
from sklearn.feature_selection import mutual_info_classif
import optuna
import shap

#%%

random_seed = 42
metric = 'f1'

# Import data
#data_raw = pd.read_csv('/Users/tracie/Desktop/academic/goethe/2-SoSe25/Seminar-
AADS/turkishCF.csv', sep=';')
data_raw = pd.read_csv('/Users/marc/Dropbox/6. Semester/Advanced Applied Data
Science/Data/turkishCF.csv', sep=';')

data = data_raw.copy()

print(data.duplicated().sum())
# No duplicated rows
```

```
# Rename feature names from turkish to english
data.rename(columns = {
    'platform_adi': 'platform',
    'kitle_fonlamasi_turu': 'crowdfunding_type',
    'kategori': 'category',
    'fon_sekli': 'funding_method',
    'proje_adi': 'project_name',
    'proje_sahibi': 'project_owner',
    'proje_sahibi_cinsiyet': 'project_owner_gender',
    'kac_proje_destekledi': 'number_of_backed_projects',
    'kac_projeye_abone': 'number_of_subscribed_projects',
    'kac_projenin_sahibi': 'number_of_projects_owners',
    'kac_proje_takiminda': 'number_teams_project_owner',
    'konum': 'location',
    'bolge': 'region',
    'yil': 'year',
    'proje_baslama_tarihi': 'start_date',
    'proje_bitis_tarihi': 'end_date',
    'gun_sayisi': 'duration',
    'tanitim_videosu': 'promotion_video',
    'video_uzunlugu': 'promotion_video_length',
    'gorsel_sayisi': 'number_of_images',
    'sss': 'faq',
    'guncellemeler': 'updates',
    'yorumlar': 'comments',
    'destekci_sayisi': 'number_of_supporters',
    'odul_sayisi': 'number_of_awards',
    'ekip_kisi_sayisi': 'number_of_project_team_members',
    'web_sitesi': 'website',
    'sosyal_medya': 'social_media',
    'sm_sayisi': 'number_of_social_media_accounts',
    'sm_takipci': 'social_media_followers',
    'etiket_sayisi': 'number_of_tags',
    'icerik_kelime_sayisi': 'number_of_words_in_description',
    'proje_aciklamasi': 'project_description',
    'hedef_miktari': 'project_target_amount',
    'toplanan_tutar': 'project_amount_collected',
    'destek_orani': 'percentage_amount_achieved',
    'basari_durumu': 'success'}, inplace = True)

print(data.head())
print(data.info())
print(data.isna().sum().sort_values(ascending=False))
print(data.describe(include='all').T)

#%% Categorical features

# Check distribution of binary features
categorical_cols = data.select_dtypes(include='object').columns
print(data[categorical_cols].nunique().sort_values(ascending=True))

binary_cols = data[categorical_cols].nunique()
binary_cols = binary_cols[binary_cols == 2].index.tolist()
binary_cols.remove('success')

# Crowdfunding type
print(data['crowdfunding_type'].value_counts())
# Delete because of no information
data = data.drop(columns=['crowdfunding_type'])
binary_cols.remove('crowdfunding_type')

# Funding method
print(data['funding_method'].value_counts()
```

```
# Rename from turkish to english
data['funding_method'] = data['funding_method'].map({
    'ya hep ya hiç': 'all_or_nothing',
    'hepsi kalsın': 'keep_it_all'})

data['funding_method_all_or_nothing'] = data['fund-
ing_method'].map({'all_or_nothing': 1, 'keep_it_all': 0})
data = data.drop(columns=['funding_method'])
binary_cols.append('funding_method_all_or_nothing')
binary_cols.remove('funding_method')

# Promotion video
print(data['promotion_video'].value_counts())
# Keep and translate
data['promotion_video'] = data['promotion_video'].map({'var': 1,'yok': 0})

# Website
print(data['website'].value_counts())
# Keep and translate
data['website'] = data['website'].map({'var': 1,'yok': 0})

# Social media
print(data['social_media'].value_counts())
# Keep and translate
data['social_media'] = data['social_media'].map({'var': 1,'yok': 0})

# Target
print(data['success'].value_counts())
# We have an imbalanced data set -> need to address this
data['success'] = data['success'].map({'başarılı': 1, 'başarısız': 0})

# New check
categorical_cols = data.select_dtypes(include='object').columns
print(data[categorical_cols].nunique().sort_values(ascending=True))

print(data['region'].value_counts())
print(data['location'].value_counts())
# Delete location because much more values, that leads to much more dummies
# that have perfect multicollinearity with region

# Drop these columns because there are irrelevant or known after start
data = data.drop(columns=[
    'project_description', # irrelevant
    'project_name', # irrelevant
    'project_owner', # irrelevant
    'end_date', # we have project duration
    'start_date', # we have project duration
    'percentage_amount_achieved', # leakage
    'location']) # multicollinearity

# Category
print(data['category'].value_counts())
# Massive imbalance between instances, have to create larger groups that
# models can work with information
```

```
# Translate from turkish to english
data['category'] = data['category'].map({
    'film-video-fotoğraf': 'film-video-photography',
    'teknoloji': 'technology',
    'kültür-sanat': 'culture-art',
    'eğitim': 'education',
    'diğer': 'other',
    'çevre': 'environment',
    'müzik': 'music',
    'sağlık-güzellik': 'health-beauty',
    'tasarım': 'design',
    'yayıncılık': 'publishing',
    'gıda-yeme-içme': 'food-eating-drinking',
    'spor': 'sports',
    'hayvanlar': 'animals',
    'moda': 'fashion',
    'sosyal sorumluluk': 'social_responsibility',
    'dans-performans': 'dance-performance',
    'turizm': 'tourism'})

# Group to reduce outliers
data['category'] = data['category'].map({
    'film-video-photography': 'creative',
    'culture-art': 'creative',
    'design': 'creative',
    'music': 'creative',
    'fashion': 'creative',
    'dance-performance': 'creative',
    'publishing': 'creative',
    'technology': 'technology',
    'other': 'other',
    'education': 'social',
    'social_responsibility': 'social',
    'environment': 'social',
    'food-eating-drinking': 'lifestyle',
    'sports': 'lifestyle',
    'animals': 'lifestyle',
    'tourism': 'lifestyle',
    'health-beauty': 'lifestyle'})

print(data['category'].value_counts())
# Much better distribution

# Gender
print(data['project_owner_gender'].value_counts())
# Good and translate
data['project_owner_gender'] = data['project_owner_gender'].map({
    'belirsiz': 'unknown',
    'kadın': 'female',
    'erkek':'male'})

# Region
print(data['region'].value_counts())
# Imbalanced distribution ->  group to other
```

```
data['region'] = data['region'].map({
    'marmara': 'marmara',
    'belirsiz': 'belirsiz',
    'iç anadolu': 'iç_anadolu',
    'ege': 'ege',
    'akdeniz': 'akdeniz',
    'genel': 'other',
    'karadeniz': 'other',
    'güneydoğu': 'other',
    'doğu': 'other'})

print(data['region'].value_counts())
# Better distribution

# Platform
print(data['platform'].value_counts())
# Imbalance, group to other

data['platform'] = data['platform'].map({
    'fongogo': 'fongogo',
    'crowdfon': 'crowdfon',
    'fonbulucu': 'fonbulucu',
    'arıkovanı': 'other',
    'buluşum': 'other',
    'ideanest': 'other'})

print(data['platform'].value_counts())

#%% Numerical data
numerical_cols = data.select_dtypes(include=['number']).columns
print((data[numerical_cols] != 0).sum().sort_values(ascending=True))

# Check correlations
correlations = data[numerical_cols].corrwith(data['success']).abs().sort_val-
ues(ascending=False)
print(correlations)
# No high correlations

print(data.groupby('year')['success'].mean())
# Only year 2011 was very successfull -> imbalance and also
# year has a very very low correlation with success and might result in many
# dummies or implies a wrong order, therefore delete it

data = data.drop(columns=[
    'number_of_subscribed_projects', # low number of values != 0
    'number_teams_project_owner', # low number of values != 0
    'id', # irrelevant
    'year', # correlation
    'faq', # leakage
    'comments', # leakage
    'updates', # leakage
    'number_of_supporters', # leakage
    'project_amount_collected', # leakage
    'social_media_followers' ]) # leakage

#%% Handle missing values
data.isna().sum().sort_values(ascending=False)



# Drop the row where the missing value in region is
data.drop(data[pd.isnull(data.region)].index,
          inplace=True)
```

```
#%% Feature engineering

# 80% training and 20% test data to have enough information in the train data
# Split into train and test data

X_train_raw, X_test_raw, y_train, y_test = train_test_split(
    data.drop(columns='success'),
    data['success'],
    test_size=0.2,
    stratify=data['success'],
    random_state=random_seed)

# Check distribution of target
print(data['success'].value_counts(normalize=True))
print(y_train.value_counts(normalize=True))
print(y_test.value_counts(normalize=True))
# Distribution is the same

X_train = X_train_raw.copy()
X_test = X_test_raw.copy()

#%% Preprocesing
numerical_cols = X_train.select_dtypes(include=['number']).columns
print(numerical_cols)

numerical_cols = pd.Index(numerical_cols).difference(binary_cols).tolist()
# Do not scale binary features

skewness_features = X_train[numerical_cols].skew().sort_values(ascending=False)
print(skewness_features)
# heavy skewed features
skewed_features = skewness_features[abs(skewness_features) > 1].index.tolist()

(X_train[skewed_features] == 0).sum()
# There are null values -> simple log does not work

# Adjust skewness
pt = PowerTransformer(method='yeo-johnson')
X_train[skewed_features] = pt.fit_transform(X_train[skewed_features])
X_test[skewed_features] = pt.transform(X_test[skewed_features])

print(X_train[numerical_cols].skew().sort_values(ascending=False))
# Skewness is much better

# Scale numerical continous features
scaler = StandardScaler()

X_train_scaled = pd.DataFrame(scaler.fit_transform(X_train[numerical_cols]),
                              columns=numerical_cols,
                              index=X_train.index)

X_test_scaled = pd.DataFrame(scaler.transform(X_test[numerical_cols]),
                             columns=numerical_cols,
                             index=X_test.index)

X_train_scaled.hist(bins=100, figsize=(20,15))
plt.show()

X_test_scaled.hist(bins=100, figsize=(20,15))
plt.show()
```

```
# Check multicollinearity
vif = pd.DataFrame()
vif['feature'] = X_train_scaled.columns
vif['vif'] = [variance_inflation_factor(X_train_scaled.values, i)
              for i in range(X_train_scaled.shape[1])]

print(vif)
# No multicollinearity

#%% Dummy encoding
categorical_cols = X_train.select_dtypes(include='object').columns

X_train_dummies = pd.get_dummies(X_train[categorical_cols],
                                 drop_first='True').astype(int)


X_test_dummies = pd.get_dummies(X_test[categorical_cols],
                                drop_first='True').astype(int)

X_test_dummies = X_test_dummies.reindex(columns=X_train_dummies.columns,
                                        fill_value=0)

#%% Get final feature data after preprocessing
X_train_binary = X_train_raw[binary_cols]
X_test_binary = X_test_raw[binary_cols]

X_train = pd.concat([X_train_scaled, X_train_dummies, X_train_binary],
                    axis=1)

X_test = pd.concat([X_test_scaled, X_test_dummies, X_test_binary],
                    axis=1)

X_train = X_train.reindex(sorted(X_train.columns),
                          axis=1)

X_test = X_test.reindex(sorted(X_test.columns),
                          axis=1)

print(data.duplicated().mean())
print(X_train.duplicated().mean())
print(X_test.duplicated().mean())

# Before data handling there were no duplicates,
# delete the duplicates in training data because it can lead
# to overfitting besides in the raw data are no duplicates
duplicated_rows =  X_train.duplicated()
X_train = X_train[~duplicated_rows]
y_train = y_train[~duplicated_rows]



#%% Checking correlation with target variable
print(X_train[numerical_cols].corrwith(y_train))

# Checking mutual information with target variable
mi = mutual_info_classif(X_train[numerical_cols], y_train)
mi_series = pd.Series(mi, index=numerical_cols)
print(mi_series.sort_values(ascending=False))
```

```
#%% Model builder
def model_build(ml_type, X_train=X_train, X_test=X_test, pred_score=metric):

    random_seed = 42

    # Function to get the model
    # A function because we need this twice, one time in the tuning and then to
fit
    # Set class_weight = 'balanced' within the models where this is possible to
address
    # imbalance and also a seed to make results reproductable
    def get_model(ml_type, params):

        random_seed = 42

        if ml_type == 'logreg':
            return LogisticRegression(**params,
                                      class_weight='balanced',
                                      random_state=random_seed)

        if ml_type == 'svm':
            return SVC(**params,
                       probability=True,
                       class_weight='balanced',
                       random_state=random_seed)

        if ml_type == 'dt':
            return DecisionTreeClassifier(**params,
                                          class_weight='balanced',
                                          random_state=random_seed)

        if ml_type == 'rf':
            return RandomForestClassifier(**params,
                                          class_weight='balanced',
                                          random_state=random_seed)

        if ml_type == 'xgb':
            return XGBClassifier(**params,
                                 random_state=random_seed)


    # Function for hyperparameter tuning
    def objective(trial, ml_type, X_train, y_train, folds):

        # Set parameters search spaces for each model
        if ml_type == 'logreg':
            params = {
                'C': trial.suggest_float('C', 0.01, 1),
                'penalty': trial.suggest_categorical('penalty', ['l1', 'l2']),
                'solver': trial.suggest_categorical('solver', ['liblinear']),
                'max_iter': trial.suggest_int('max_iter', 1000, 5000)}

        if ml_type == 'svm':
            params = {
                'C': trial.suggest_float('C', 0.01, 1),
                'kernel': trial.suggest_categorical('kernel', ['rbf', 'line-
ar']),
                'gamma': trial.suggest_categorical('gamma', ['scale', 'auto']),
                'degree': trial.suggest_int('degree', 2, 4)}
```

```
        if ml_type == 'dt':
            params = {
                'criterion': trial.suggest_categorical('criterion', ['entropy',
'gini']),
                'max_depth': trial.suggest_int('max_depth', 3, 5),
                'min_samples_leaf': trial.suggest_int('min_samples_leaf', 20,
50),
                'min_samples_split': trial.suggest_int('min_samples_split', 30,
80)}

        if ml_type == 'rf':
            params = {
                'n_estimators': trial.suggest_int('n_estimators', 350, 500),
                'max_depth': trial.suggest_int('max_depth', 8, 12),
                'min_samples_leaf': trial.suggest_int('min_samples_leaf', 8,
15),
                'min_samples_split': trial.suggest_int('min_samples_split', 15,
25),
                'max_features': trial.suggest_float('max_features', 0.3, 0.5),
                'max_samples': trial.suggest_float('max_samples', 0.8, 0.9),
                'bootstrap': True}

        if ml_type == 'xgb':
            params = {
                'n_estimators': trial.suggest_int('n_estimators', 50, 150),
                'learning_rate': trial.suggest_float('learning_rate', 0.05,
0.15),
                'max_depth': trial.suggest_int('max_depth', 3, 5),
                'min_child_weight': trial.suggest_int('min_child_weight', 10,
25),
                'subsample': trial.suggest_float('subsample', 0.5, 0.7),
                'colsample_bytree': trial.suggest_float('colsample_bytree', 0.5,
0.7),
                'reg_alpha': trial.suggest_float('reg_alpha', 0.1, 1.0),
                'reg_lambda': trial.suggest_float('reg_lambda', 5, 20)}


        model = get_model(ml_type=ml_type,
                          params=params)

        # Get F1 values of each fold
        score = cross_val_score(estimator=model,
                                X=X_train,
                                y=y_train,
                                cv=folds,
                                scoring='f1',
                                n_jobs=-1)

        # Maximize the mean F1-score from the validation folds
        return score.mean()


    # Split train set into five splits means there is every time 80% training
    # and 20% validation, save the folds to have them in a later calculation
    # for the mean score of the folds
    k_folds = StratifiedKFold(n_splits=5,
                              shuffle=True,
                              random_state=random_seed)
    folds = list(k_folds.split(X_train, y_train))

    # Optuna hyperparameter tuning
    sampler = optuna.samplers.TPESampler(seed=random_seed)
```

```
    pruner = optuna.pruners.MedianPruner(n_warmup_steps=1)

    study = optuna.create_study(direction='maximize',
                                sampler=sampler,
                                pruner=pruner)

    # High number of trials to find optimal values
    study.optimize(lambda trial: objective(trial=trial,
                                           ml_type=ml_type,
                                           X_train=X_train,
                                           y_train=y_train,
                                           folds=folds),
                   n_trials=200,
                   timeout=300)

    # Get optimal values
    optimal_params = study.best_params

    # If model is XGB set extra parameter to address imbalance
    if ml_type == 'xgb':
        scale_pos_weight_value = (y_train == 0).sum() / (y_train == 1).sum()
        optimal_params['scale_pos_weight'] = scale_pos_weight_value

    # Tune model with best parameters
    tuned_model = get_model(ml_type = ml_type,
                            params=optimal_params)

    tuned_model.fit(X_train, y_train)

    y_pred_train = tuned_model.predict(X_train)
    train_score = f1_score(y_true=y_train,
                           y_pred=y_pred_train)


    cv_scores = cross_val_score(estimator=tuned_model,
                                X=X_train,
                                y=y_train,
                                cv=folds,
                                scoring=pred_score,
                                n_jobs=-1)

    return tuned_model, cv_scores, folds, train_score

#%% Build predict funtion
def model_pred(tuned_model, X_test=X_test):

    y_pred = tuned_model.predict(X_test)
    y_prob = tuned_model.predict_proba(X_test)[:, 1]

    return y_pred, y_prob
```

```
#%%
def evaluate_model(fitted_model, y_test=y_test):

    y_pred = fitted_model[0]
    y_prob = fitted_model[1]

    tn, fp, fn, tp = confusion_matrix(y_true=y_test,
                                      y_pred=y_pred).ravel()

    metrics = pd.DataFrame.from_dict(
        {'Accurracy': (tp+tn)/(tp+tn+fp+fn),
         'Specificity': tn/(tn+fp),
         'Recall/Sensitivity': tp/(tp+fn),
         'Precision': tp/(tp+fp),
         'F1': (2*tp)/(2*tp+fp+fn),
         'AUC': roc_auc_score(y_test, y_prob)},
        orient='index',
        columns=['value'])

    return metrics

#%% Build models and predict
logreg_model = model_build(ml_type='logreg')
dt_model = model_build(ml_type='dt')
rf_model = model_build(ml_type='rf')
xgb_model = model_build(ml_type='xgb')


svm_model = model_build(ml_type='svm')
logreg_pred = model_pred(tuned_model=logreg_model[0])
svm_pred = model_pred(tuned_model=svm_model[0])
dt_pred = model_pred(tuned_model=dt_model[0])
rf_pred = model_pred(tuned_model=rf_model[0])
xgb_pred = model_pred(tuned_model=xgb_model[0])

#%% Evaluations
evaluation_logreg = evaluate_model(fitted_model=logreg_pred)
evaluation_svm = evaluate_model(fitted_model=svm_pred)
evaluation_dt = evaluate_model(fitted_model=dt_pred)
evaluation_rf = evaluate_model(fitted_model=rf_pred)
evaluation_xgb = evaluate_model(fitted_model=xgb_pred)

evaluation_logreg.columns = ['LOGREG']
evaluation_svm.columns = ['SVM']
evaluation_dt.columns = ['DT']
evaluation_rf.columns = ['RF']
evaluation_xgb.columns = ['XGB']

evaluation_models = pd.concat([evaluation_logreg,
                               evaluation_svm,
                               evaluation_dt,
                               evaluation_rf,
                               evaluation_xgb],
                              axis=1)

evaluation_models.iloc[0:] = evaluation_models.iloc[0:].round(2)

# Export to Excel
evaluation_models.to_excel('evaluation_models.xlsx', index=True)
```

```
#%% Over-/ Underfitting check

score = 'F1'

logreg_model[3]
np.mean(logreg_model[1])
np.std(logreg_model[1])
evaluation_logreg.loc[score, 'LOGREG']
logreg_model[0].get_params()

svm_model[3]
np.mean(svm_model[1])
np.std(svm_model[1])
evaluation_svm.loc[score, 'SVM']
svm_model[0].get_params()

dt_model[3]
np.mean(dt_model[1])
np.std(dt_model[1])
evaluation_dt.loc[score, 'DT']
dt_model[0].get_params()

rf_model[3]
np.mean(rf_model[1])
np.std(rf_model[1])
evaluation_rf.loc[score, 'RF']
rf_model[0].get_params()

xgb_model[3]
np.mean(xgb_model[1])
np.std(xgb_model[1])
evaluation_xgb.loc[score, 'XGB']
xgb_model[0].get_params()

#%% ROC curve
model_dict = {
    'Logistic Regression': logreg_pred[1],
    'Support Vector Machine': svm_pred[1],
    'Decision Tree': dt_pred[1],
    'Random Forest': rf_pred[1],
    'XGBoost': xgb_pred[1]}

color_dict = {
    'Logistic Regression': 'orange',
    'Support Vector Machine': 'green',
    'Decision Tree': 'yellow',
    'Random Forest': 'red',
    'XGBoost': 'darkblue'}

linestyle_dict = {
    'Logistic Regression': '-',
    'Support Vector Machine': '-',
    'Decision Tree': '-',
    'Random Forest': '-',
    'XGBoost': '-'}
```

```
fig, ax = plt.subplots(figsize=(14, 8))

for name, y_prob in model_dict.items():
    fpr, tpr, _ = roc_curve(y_test, y_prob)
    auc = roc_auc_score(y_test, y_prob)
    ax.plot(
        fpr, tpr,
        label=f'{name} (AUC = {auc:.2f})',
        color=color_dict.get(name),
        linestyle=linestyle_dict.get(name, '-'))

ax.plot([0, 1], [0, 1], linestyle='--', color='black')
ax.set_xlabel('False Positive Rate (FPR)',
              fontsize=18)
ax.set_ylabel('True Positive Rate (TPR)',
              fontsize=18)
ax.tick_params(axis='both',
               labelsize=14)
ax.set_title('Comparison of ROC Curves',
             fontsize=20)
ax.legend(loc='lower right',
          fontsize=14)
ax.grid(False)

plt.savefig('roc_curve.png',
            dpi=500,
            bbox_inches='tight')
plt.show()

#%% Feature imporatnce
feature_importance = pd.Series(xgb_model[0].feature_importances_,
                               index=X_train.columns).sort_values(ascend-
ing=False)
print(feature_importance)

#%% SHAP
explainer = shap.TreeExplainer(xgb_model[0])
shap_values = explainer(X_test)

shap.plots.beeswarm(shap_values,
                    max_display=20,
                    show = False)

fig = plt.gcf()
fig.set_size_inches(10, 6)

plt.savefig('shap_importance.png',
            dpi=500,
            bbox_inches='tight')

plt.show()

#%% Statistical test

# Less values
success = X_test[y_test == 1]['duration']
fail = X_test[y_test == 0]['duration']
t_value, p_value = ttest_ind(success, fail, equal_var=False, alternative='less')
print(t_value, p_value )
```

```
success = X_test[y_test == 1]['project_target_amount']
fail = X_test[y_test == 0]['project_target_amount']
t_value, p_value = ttest_ind(success, fail, equal_var=False, alternative='less')
print(t_value, p_value )


# Greater values
success = X_test[y_test == 1]['promotion_video_length']
fail = X_test[y_test == 0]['promotion_video_length']
t_value, p_value = ttest_ind(success, fail, equal_var=False, alternative='great-
er')
print(t_value, p_value )

#%% Misclassifications
pred = xgb_pred[0]

# Searching for differences in shap values of correct and falsely predicted
misclassifications_test = (y_test!=pred).values

error_shap_comparison = pd.DataFrame({
    'feature': X_test.columns,
    'correct': np.mean(shap_values[~misclassifications_test].values, axis=0),
    'misclassified': np.mean(shap_values[misclassifications_test].values,
axis=0)})

error_shap_comparison['diff'] = abs(error_shap_comparison['correct']) - abs(er-
ror_shap_comparison['misclassified'])
error_shap_comparison = error_shap_comparison.sort_values('diff', ascend-
ing=False)
print(error_shap_comparison)

#%% Confusion matrix and learning curve
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(20, 8))

# Confusion Matrix
confusion_mat_data = confusion_matrix(y_true=y_test,
                                      y_pred=xgb_pred[0])

sns.heatmap(confusion_mat_data, annot=True, fmt='d', cmap='Blues',
            xticklabels=['Unsuccessful', 'Successful'],
            yticklabels=['Unsuccessful', 'Successful'],
            cbar=False,
            annot_kws={"size": 18},
            ax=ax1)

ax1.set_xlabel('Prediction', fontsize=20)
ax1.set_ylabel('True Value', fontsize=20)
ax1.tick_params(axis='both', which='major', labelsize=16)
ax1.set_title('Confusion Matrix XGBoost', fontsize=22)

model = xgb_model[0]
folds = xgb_model[2]
```

```
# F1-Score Learning Curve
train_sizes_f1, train_scores_f1, validation_scores_f1 = learning_curve(
    estimator=model,
    X=X_train,
    y=y_train,
    cv=folds,
    scoring=make_scorer(f1_score),
    train_sizes=np.linspace(0.1, 1.0, 20),
    n_jobs=-1,
    random_state=random_seed)

validation_f1_mean = validation_scores_f1.mean(axis=1)
validation_f1_std = validation_scores_f1.std(axis=1)

# Recall Learning Curve
train_sizes_recall, train_scores_recall, validation_scores_recall = learn-
ing_curve(
    estimator=model,
    X=X_train,
    y=y_train,
    cv=folds,
    scoring=make_scorer(recall_score),
    train_sizes=np.linspace(0.1, 1.0, 20),
    n_jobs=-1,
    random_state=random_seed)

validation_recall_mean = validation_scores_recall.mean(axis=1)
validation_recall_std = validation_scores_recall.std(axis=1)

# Precision Learning Curve
train_sizes_precision, train_scores_precision, validation_scores_precision =
learning_curve(
    estimator=model,
    X=X_train,
    y=y_train,
    cv=folds,
    scoring=make_scorer(precision_score),
    train_sizes=np.linspace(0.1, 1.0, 20),
    n_jobs=-1,
    random_state=random_seed)

validation_precision_mean = validation_scores_precision.mean(axis=1)
validation_precision_std = validation_scores_precision.std(axis=1)

# Learning curves
# F1
ax2.plot(train_sizes_f1,
         validation_f1_mean,
         color='blue',
         label='F1-Score',
         linewidth=2)

ax2.fill_between(train_sizes_f1,
                 validation_f1_mean - validation_f1_std,
                 validation_f1_mean + validation_f1_std,
                 color='blue',
                 alpha=0.2)
```

```
# Recall
ax2.plot(train_sizes_recall,
         validation_recall_mean,
         color='red',
         label='Recall',
         linewidth=2)

ax2.fill_between(train_sizes_recall,
                 validation_recall_mean - validation_recall_std,
                 validation_recall_mean + validation_recall_std,
                 color='red',
                 alpha=0.2)

# Precision
ax2.plot(train_sizes_precision,
         validation_precision_mean,
         color='green',
         label='Precision',
         linewidth=2)

ax2.fill_between(train_sizes_precision,
                 validation_precision_mean - validation_precision_std,
                 validation_precision_mean + validation_precision_std,
                 color='green',
                 alpha=0.2)



ax2.set_title('Learning Curves XGBoost', fontsize=22)
ax2.set_xlabel('Training Set Size', fontsize=20)
ax2.set_ylabel('Validation Score', fontsize=20)
ax2.tick_params(axis='both', which='major', labelsize=16)
ax2.grid(True)
ax2.legend(loc='lower right', fontsize=18)
ax2.set_ylim(0.3, 0.9)

plt.tight_layout()
plt.savefig('confusion_matrix_and_learning_curve.png',
            dpi=500,
            bbox_inches='tight')
plt.show()
```

Statutory Declaration

“We herewith declare that we have composed the present thesis ourself and without use of any other than the cited sources and aids. Sentences or parts of sentences quoted literally are marked as such; other references with regard to the statement and scope are indicated by full details of the publications concerned. The thesis in the same or similar form has not been submitted to any examination body and has not been published. This thesis was not yet, even in part, used in another examination or as a course performance. Furthermore we declare that the submitted written (bound) copies of the present thesis and the version submitted on a data carrier are consistent with each other in contents.”

Place, Date: Frankfurt am Main, 22.06.2025.

Signature: Marc Günter Dörsam Ngoc Khanh Uyen Tran