1. Introduction

This report outlines the preprocessing steps applied to project_data.csv, a dataset from the 2023 American Community Survey (ACS). The dataset consists of 4,318 entries and 117 variables, each representing various demographic and socio-economic attributes of individuals in the U.S. The main goal of this project is to develop multiple classification models to predict whether a person experiences difficulty living independently. However, before jumping into model construction, we carried out a detailed preprocessing of the data to ensure its quality. Without proper preprocessing, issues like missing values, irrelevant attributes, incorrect data types, and extreme outliers could compromise model performance. Poor data preparation often results in biased models, unreliable predictions, and diminished interpretability.

The preprocessing process includes several steps, starting with handling missing values. Incomplete data, if not addressed, can skew model training, leading to inaccurate results. Alongside this, we remove unnecessary columns to remove redundant or irrelevant information that could add unnecessary noise to the dataset. We also ensure that each variable has the correct data type, maintaining consistency and preventing potential computational issues. Outliers are another concern as extreme values could disproportionately influence the model's performance. In addition, we employ methods for balancing the dataset. This ensures that underrepresented classes are appropriately accounted for, promoting fairness and improving the model's ability to predict across all groups. Lastly, to improve model efficiency and predictive accuracy, we apply three different attribute selection techniques. This helps us identify the most relevant features for classification, reducing dimensionality without sacrificing important information.

With these preprocessing steps, we ensure the dataset is clean, reliable, and optimized for classification modeling. Properly prepared data significantly enhances model performance, making it more robust and generalizable to new data. The following sections will walk you through the specific preprocessing techniques we've implemented in this project, providing a clear methodology for our approach.

2. Data Preprocessing Methods

We began the data preprocessing process by loading the dataset into R and reviewing its dimensions to understand the number of observations and variables. This helped us get a clear overview of the dataset's structure and identify any immediate issues that might need attention. For instance, it allowed us to quickly spot columns with significant amounts of missing data or those that seemed irrelevant to the analysis, allowing us to target the most problematic areas first.

Next, we focused on handling missing values. We checked for missing data both at the dataset level and column level. Columns with more than 10% missing values were removed, as they were considered too unreliable for analysis. For columns with less than 10% missing data, we decided to drop rows with missing values in critical fields to ensure that we retained as much

useful data as possible while maintaining data integrity. Since we addressed all missing values through row deletion, no imputation was needed. This approach ensured that we didn't introduce any assumptions or biases by trying to fill in missing values, which can sometimes be problematic.

After addressing missing values, we removed columns that contained only a single unique value. These columns were removed because they did not provide any meaningful information for the classification task. Including variables with only one value could skew the model, as they don't contribute to distinguishing between classes. Removing them helped streamline the dataset and ensured that we were working with more relevant features. We then ensured that all numerical variables were properly formatted. Some variables, such as "Income" and "Age," had been mistakenly categorized as factors. These variables were meant to be numeric, so we converted them into the correct format. This was an important step, as it ensured that all numerical data would be processed appropriately in later stages of the analysis and avoided potential errors in modeling algorithms that expect numeric input.

Outlier detection was another key part of the preprocessing process. We calculated the Interquartile Range (IQR) for selected numeric columns to identify extreme values. Initially, we removed data points that fell more than 1.5 times the IQR above the third quartile or below the first quartile. However, this approach resulted in a significant reduction in the dataset size, with many rows being excluded. Given that this could lead to a loss of valuable information, we decided to modify our strategy by capping the extreme values instead of removing them entirely. By adjusting the IQR multiplier to a less strict value (6 times the IQR), we were able to set upper and lower bounds that allowed us to retain more data while still limiting the influence of extreme outliers on the analysis.

After these adjustments, we reviewed the dataset to ensure that everything made sense. We found that the data was now cleaned and formatted correctly, with no missing values or irrelevant columns. We also removed features with near-zero variance, as these offered little predictive value and could potentially introduce noise into the model. Additionally, we checked for highly correlated features, removing any variables with a correlation greater than 0.9 to avoid multicollinearity, which could affect the stability and interpretability of the model. As a result of these preprocessing steps, we retained 95% of the original rows, ensuring that the dataset was ready for building reliable classification models. This thorough cleaning process, which included handling missing values, removing irrelevant or redundant features, ensuring correct variable types, and addressing outliers, laid a solid foundation for creating accurate and effective models.

3. Methods

3.1 Balanced Dataset Methods Overview

The dataset we worked with was imbalanced, meaning one class had way fewer samples than the other. If we didn't do anything about it, our models would likely just predict the majority class most of the time and ignore the minority class, which isn't useful and would be biased. To deal with this, we used two different balancing methods: class weights and random oversampling.

Class weights adjust how much importance the model gives to each class. Since the minority class had fewer samples, we increased its weight so that the model wouldn't just default to predicting the majority class. This helped models like logistic regression and decision trees pay more attention to the minority class without actually changing the number of samples in the dataset.

Random oversampling, on the other hand, increases the number of samples in the minority class by randomly duplicating existing data points. This makes the dataset more balanced, giving models more chances to learn patterns from both classes. Unlike class weighting, which changes how errors are penalized, oversampling physically increases the number of minority class samples, which is useful for models that don't support weighted loss functions.

Using both methods helped ensure that our models didn't just favor the majority class, giving us a better chance of getting meaningful predictions for both classes.

3.2 Attribute Selection Overview

When working with a dataset that has a lot of features, not all of them are necessary for building an effective model. Some features are redundant, highly correlated, or add noise rather than useful information. Keeping too many unnecessary features can slow down the model, increase the risk of overfitting, and make interpretation more difficult. To address this, we applied three feature selection methods: Lasso, Random Forest, and Recursive Feature Elimination (RFE). Each of these approaches takes a different strategy, helping us identify the most important predictors while removing irrelevant or redundant variables.

Lasso Regression is a linear model that applies L1 regularization, which forces the model to shrink the coefficients of less important features to zero. This means that any feature with a zero coefficient gets effectively removed from the model. The advantage of Lasso is that it not only helps with feature selection but also prevents overfitting by simplifying the model. It's particularly useful when dealing with high-dimensional datasets where many variables may be irrelevant.

Random Forest is a tree-based ensemble model that ranks feature importance based on how much each variable reduces prediction error. Since Random Forest is non-linear, it can capture complex interactions between features, making it an excellent method for identifying important predictors even when relationships aren't straightforward. We used feature importance scores from Random Forest to drop variables that contributed little to model performance. Unlike

Lasso, which selects features based on linear relationships, Random Forest considers more complex patterns, making it a useful alternative.

Recursive Feature Elimination (RFE) is an iterative process that works by fitting a model multiple times and systematically removing the least important features. It starts with all available features, ranks them based on their contribution to model performance, and then eliminates the weakest ones in each step. This continues until only the most useful predictors remain. RFE is particularly helpful when we want to find the optimal number of features without setting an arbitrary cutoff.

By using these three methods in our models, we ensured that our dataset retained only the most meaningful variables. This approach improved model efficiency, reduced the risk of overfitting, and made our results easier to interpret. Instead of relying on just one technique, we leveraged the strengths of each to get a well-balanced variety of models that were able to perform effective feature selection processes.

3.3 Classification Algorithms Overview

When working with a classification problem, choosing the right algorithm is crucial for achieving high accuracy and reliable predictions. Since different algorithms have different strengths and weaknesses, we tested multiple approaches to see which one performed best for our dataset. Specifically, we focused on using Support Vector Machine (SVM), Logistic Regression, Random Forest, Naive Bayes, Neural Network, and Decision Tree. Each of these algorithms approaches classification differently, so using a mix of linear, probabilistic, and non-linear approaches in our models helped us build a more robust analysis.

Support Vector Machine (SVM) is a very powerful classification algorithm that works by finding the optimal decision boundary between the separating class. It is especially useful for high-dimensional data and can handle both linear and non-linear relationships using kernel functions. One of SVM's strengths is that it is resistant to overfitting, especially in cases where the number of features is large compared to the number of samples.

Random Forest is an ensemble machine learning algorithm that builds a large number of decision trees and combines their results to improve prediction accuracy and control overfitting. Each tree is trained on a random subset of the data and considers a random subset of features at each split, introducing diversity among the trees. The final prediction is made by majority vote across all trees. This randomness and aggregation make it robust and less sensitive to noise in the data compared to a single decision tree.

K-Nearest Neighbors (KNN) is a distance-based algorithm that classifies a data point based on the majority class of its nearest neighbors. It is non-parametric, meaning it makes no assumptions about the underlying data distribution. While KNN can be very effective in well-structured

datasets, it is sensitive to the choice of k (the number of neighbors) and can struggle with large datasets due to its computational cost.

Naive Bayes is a probabilistic classifier based on Bayes' theorem. It assumes that features are conditionally independent given the class label, which is rarely true in real-world datasets. However, despite these strong assumptions, Naïve Bayes performs surprisingly well in many cases, particularly when dealing with text classification and datasets with categorical features.

A neural network is a more complex model composed of layers of interconnected neurons that process and transform input data to make predictions. It offers high flexibility, enabling it to capture intricate relationships between features. However, it requires a large amount of data to train effectively and can be computationally expensive. For our classification task, we incorporated a neural network to model complex patterns and improve prediction accuracy.

Decision Tree is an interpretable and intuitive model that splits the data based on feature values to create a tree-like structure. It is easy to understand and works well with datasets that have non-linear relationships between variables. However, decision trees are prone to overfitting, which is why they are often used in ensemble methods like Random Forest.

By testing these six classification algorithms, we were able to compare their performance and select the best model based on accuracy, precision, recall, and overall reliability. Using multiple approaches allowed us to capture different patterns in the data and ensure that we weren't drawing conclusions only based on one specific modeling technique.

4. Results

For all models, after obtaining the best model, we evaluated it on the testing set using multiple performance metrics for both the positive class ("Yes") and negative class ("No"). This included True Positive Rate (TPR), False Positive Rate (FPR), Precision, Recall, F1-score, Kappa, Matthews Correlation Coefficient (MCC), and AUC (Area Under the Curve). These metrics were calculated twice — once with "Yes" as the positive class and again with "No" — to give a full picture of the model's performance. Finally, we computed weighted averages of all metrics based on class distribution in the test set to fairly assess overall model effectiveness.

Model 1: Class Weights, Lasso Feature Selection, and Logistic Regression

In this approach, we aimed to build a logistic regression model enhanced by Lasso-based feature selection, class weights for imbalance handling, and hyperparameter tuning across different regularization strategies. The data is split into 66% training and 34% testing using stratified sampling based on the Class column, ensuring the same class proportion in both sets. Since the dataset is imbalanced, we calculated inverse class weights for the training set. These weights

were used during model training to penalize misclassification of the minority class more heavily, helping to reduce bias.

Next, we applied Lasso regression (L1 regularization using glmnet with alpha = 1) to the training data. Lasso performs embedded feature selection by shrinking coefficients of less important features to zero. We extracted the non-zero coefficients at the optimal lambda (lambda.min) to identify the most informative features, which significantly reduced dimensionality. We then subset both training and testing datasets to only include these selected features. To ensure consistent scaling across predictors, especially important for regularized regression, we standardized the features in both datasets using z-score normalization.

Following this, we trained a logistic regression model using the caret package with glmnet, allowing it to explore Ridge (alpha = 0), Elastic Net (alpha = 0.5), and Lasso (alpha = 1) within a 10-fold cross-validation framework. The tuning grid spanned a range of lambda values from 0.001 to 10 on a log scale. The model was trained to optimize the F1-score, an effective metric when dealing with imbalanced data.

Performance measure table:

	TPR	FPR	Precision	Recall	F-measure	ROC	MCC	Kappa
Class Yes	0.919	0.306	0.151	0.919	0.260	0.572	0.298	0.181
Class No	0.694	0.081	0.993	0.694	0.817	0.572	0.298	0.181
Wt. Average	0.707	0.094	0.946	0.707	0.707	0.572	0.298	0.181

Confusion matrix:

	Reference: No	Reference: Yes
Prediction: No	864	6
Prediction: Yes	381	68

Model 2: Class Weights, Lasso Feature Selection, and Support Vector Machine

In this approach, we combined Lasso-based feature selection, class weight adjustments for imbalanced data, and Support Vector Machine (SVM) modeling. Given the class imbalance, we computed inverse frequency-based class weights from the training set. These weights were later applied in the SVM model to reduce bias by penalizing misclassification of the minority class more heavily.

Next, we applied Lasso regression (L1 regularization using glmnet with alpha = 1) to the training data. Lasso provides embedded feature selection by shrinking irrelevant coefficients to zero. We extracted the non-zero coefficients at the optimal penalty level (lambda.min) to identify the most predictive features. This process helped reduce the feature space while retaining meaningful variables. Both the training and testing datasets were then subset to include only the selected features. To ensure all predictors were on the same scale, which is especially important for SVMs, we standardized the datasets using z-score normalization.

With the refined and standardized features, we trained a linear Support Vector Machine (SVM) using the e1071 package. The previously computed class weights were incorporated into the model to address class imbalance. Unlike Model 1, which used logistic regression and regularization tuning, this model focused on maximizing margin separation using SVM with fixed linear kernel and weighted loss.

Performance measure table:

	TPR	FPR	Precision	Recall	F-measure	ROC	MCC	Kappa
Class Yes	0.865	0.300	0.146	0.865	0.250	0.568	0.276	0.170
Class No	0.700	0.135	0.989	0.700	0.819	0.568	0.276	0.170
Wt. Average	0.709	0.144	0.941	0.709	0.787	0.568	0.276	0.170

Confusion matrix:

	Reference: No	Reference: Yes
Prediction: No	871	10
Prediction: Yes	374	64

Model 3: Class Weights, Lasso Feature Selection, and Random Forest

In this approach, we constructed a classification model combining Lasso-based feature selection, class weighting to address imbalance, and a Random Forest classifier. The process began by splitting the preprocessed dataset into training and testing subsets while maintaining class proportions through stratified sampling. The data is split into 66% training and 34% testing using stratified sampling based on the Class column, ensuring the same class proportion in both sets. Given the imbalance between classes, we calculated inverse class weights based on class frequencies within the training data. These weights were later passed to the Random Forest model to emphasize correct classification of the minority class.

Lasso regression (L1 regularization via glmnet with alpha = 1) was then applied to the training set to identify and retain only the most relevant predictors. By shrinking less informative feature coefficients to zero, Lasso performed embedded feature selection. We extracted the features corresponding to non-zero coefficients at the optimal regularization level (lambda.min). Both the training and testing datasets were reduced to include only these selected features, effectively lowering dimensionality and potential noise before model training.

Next, a Random Forest classifier was trained using the randomForest package. The class weights derived earlier were passed to the model via the classwt parameter to mitigate bias toward the majority class. Unlike Models 1 and 2, which used logistic regression and SVM respectively, the Random Forest model leverages an ensemble of decision trees to make robust predictions by aggregating across multiple randomized trees.

Performance measure table:

	TPR	FPR	Precision	Recall	F-meaure	ROC	MCC	Kappa
Class Yes	0.081	0.017	0.222	0.081	0.119	0.585	0.104	0.092
Class No	0.983	0.919	0.947	0.983	0.965	0.585	0.104	0.092
Wt. Average	0.933	0.868	0.906	0.933	0.917	0.585	0.104	0.092

Confusion matrix:

	Reference: No	Reference: Yes
Prediction: No	1224	68
Prediction: Yes	21	6

Model 4: Class Weights, Lasso Feature Selection, and Naive Bayes Classifier

In this approach, we combined Lasso-based feature selection with a Naive Bayes classifier. We began by splitting the preprocessed dataset into training and testing sets using stratified sampling to maintain class distribution. The data is split into 66% training and 34% testing using stratified sampling based on the Class column, ensuring the same class proportion in both sets. Unlike other models in this study, class weights were not explicitly applied and Naive Bayes inherently accounts for class imbalance through its use of prior probabilities during prediction.

Lasso regression (L1 regularization using glmnet with alpha = 1) was applied to the training data to identify the most informative predictors. Lasso performs embedded feature selection by shrinking the coefficients of less relevant features to zero. The features associated with non-zero coefficients at the optimal penalty level (lambda.min) were extracted and retained for

downstream modeling. Both the training and testing datasets were reduced to include only these selected features, ensuring a more focused and simplified model input.

With the reduced feature set, we trained a Naive Bayes classifier using the naiveBayes function from the e1071 package. This probabilistic model assumes feature independence given the class label and uses both likelihood and class prior probabilities to make predictions. While simple and computationally efficient, Naive Bayes can perform surprisingly well when features are truly or approximately independent, and it naturally adjusts for class imbalance without requiring manual weighting.

The trained model was evaluated on the test set using multiple performance metrics for both the positive class ("Yes") and the negative class ("No"). These included True Positive Rate (TPR), False Positive Rate (FPR), Precision, Recall, F1-score, Kappa, Matthews Correlation Coefficient (MCC), and AUC (Area Under the Curve). To fully understand model performance across both classes, each metric was computed twice: once with "Yes" as the positive class, and once with "No". Finally, weighted averages of all metrics were calculated based on the class proportions in the test set to provide a balanced, overall performance summary.

Performance measure table:

	TPR	FPR	Precision	Recall	F-measure	ROC	MCC	Kappa
Class Yes	0.824	0.189	0.206	0.824	0.330	0.597	0.351	0.264
Class No	0.811	0.176	0.988	0.811	0.891	0.597	0.351	0.264
Wt. Average	0.812	0.174	0.943	0.812	0.859	0.597	0.351	0.264

Confusion matrix:

	Reference: No	Reference: Yes
Prediction: No	1010	13
Prediction: Yes	235	61

Model 5: Class Weights, Lasso Feature Selection, and Neural Network Classifier

In this approach, combined Lasso-based feature selection, class weighting for imbalance handling, and a neural network classifier. We began by splitting the preprocessed dataset into training and testing sets using stratified sampling to preserve the original class distribution. The data is split into 66% training and 34% testing using stratified sampling based on the Class column, ensuring the same class proportion in both sets. Due to the presence of class imbalance, we calculated inverse class weights from the training set, assigning greater importance to the minority class. These weights were used during model training to reduce bias toward the majority class.

We then applied Lasso regression (L1 regularization via glmnet, with alpha = 1) to the training data to perform feature selection. Lasso helps simplify the model by shrinking less relevant feature coefficients to zero. By extracting the features associated with non-zero coefficients at the optimal regularization level (lambda.min), we significantly reduced dimensionality while retaining key predictors. The training and testing datasets were subsequently reduced to include only these selected features.

Next, we trained a feedforward neural network model using the nnet package. The model architecture consisted of a single hidden layer with 6 neurons, a decay (regularization) rate of 0.15, and a maximum of 200 iterations. Class weights were passed to the model to handle imbalance during learning. Unlike previous models that used linear methods or tree-based ensembles, this model leveraged the nonlinear representation capacity of neural networks to model complex decision boundaries.

Performance measure table:

	TPR	FPR	Precision	Recall	F-measure	ROC	MCC	Kappa
Class Yes	0.838	0.319	0.135	0.838	0.233	0.561	0.251	0.151
Class No	0.681	0.162	0.986	0.681	0.806	0.561	0.251	0.151
Wt. Average	0.690	0.171	0.938	0.690	0.774	0.561	0.251	0.151

Confusion matrix:

	Reference: No	Reference: Yes
Prediction: No	849	11
Prediction: Yes	396	63

Model 6: Class Weights, Lasso Feature Selection, and Decision tree

In this approach, we integrated Lasso-based feature selection, class weighting to handle imbalance, and a decision tree classifier. The modeling process began by splitting the preprocessed dataset into training and testing sets using stratified sampling to preserve the distribution of class labels. The data is split into 66% training and 34% testing using stratified sampling based on the Class column, ensuring the same class proportion in both sets. Because the dataset is imbalanced, inverse class weights were computed based on the training set to assign greater importance to the minority class during model training.

Lasso regression (L1 regularization using glmnet with alpha = 1) was applied to the training data to select informative features by shrinking the coefficients of less relevant variables to zero. We extracted the features associated with non-zero coefficients at the optimal penalty level (lambda.min) and reduced both the training and testing datasets to include only these selected

variables. This dimensionality reduction step helps improve model interpretability and focus training on the most predictive inputs.

With the reduced feature set, we trained a decision tree classifier using the rpart package. The model was configured to accept the previously calculated class weights to counter the effects of imbalance. Decision trees are inherently interpretable and model non-linear interactions well by recursively partitioning the feature space based on feature thresholds that maximize information gain.

Performance measure table:

	TPR	FPR	Precision	Recall	F-measure	ROC	MCC	Kappa
Class Yes	0.811	0.238	0.169	0.811	0.279	0.577	0.297	0.205
Class No	0.762	0.189	0.985	0.762	0.860	0.577	0.297	0.205
Wt. Average	0.765	0.192	0.940	0.765	0.827	0.577	0.297	0.205

Confusion matrix:

	Reference: No	Reference: Yes
Prediction: No	949	14
Prediction: Yes	296	60

Model 7: Class Weights, Recursive Feature Elimination, and Logistic regression

In this approach, we implemented a logistic regression model enhanced by recursive feature elimination (RFE) for feature selection and class weighting to address class imbalance. We started by splitting the preprocessed dataset into training and testing sets using stratified sampling to preserve class proportions. The data is split into 66% training and 34% testing using stratified sampling based on the Class column, ensuring the same class proportion in both sets. Due to class imbalance, we computed inverse class weights from the training data and scaled them to whole numbers to make them compatible with logistic regression. These weights were applied during training to help the model give more attention to the minority class.

Unlike previous models that used Lasso, this model utilized Recursive Feature Elimination (RFE) with 10-fold cross-validation for feature selection. RFE repeatedly removes the least important features based on model performance, in this case using random forest functions (rfFuncs) to rank variable importance. The process selected an optimal subset of up to 20 features. We then reduced both the training and testing datasets to include only these selected features.

A logistic regression model was trained using the reduced feature set and the computed class weights. Predictions were generated as probabilities and converted to binary class labels using a

0.5 threshold. This approach retained the interpretability of logistic regression while ensuring that only the most relevant features were used, and that class imbalance was accounted for during training.

Performance measure table:

	TPR	FPR	Precision	Recall	F-measure	ROC	MCC	Kappa
Class Yes	0.905	0.293	0.155	0.905	0.265	0.574	0.300	0.187
Class No	0.707	0.095	0.992	0.707	0.823	0.574	0.300	0.187
Wt. Average	0.718	0.106	0.645	0.718	0.794	0.574	0.300	0.187

Confusion matrix:

	Reference: No	Reference: Yes
Prediction: No	886	7
Prediction: Yes	359	67

Model 8: Class Weights, Recursive Feature Elimination, and Support Vector Machine

In this approach, we created a classification model that combines recursive feature elimination (RFE), class weighting, and a linear Support Vector Machine (SVM). As with previous models, we began by splitting the preprocessed dataset into training and testing sets using stratified sampling to preserve the distribution of the target variable. The data is split into 66% training and 34% testing using stratified sampling based on the Class column, ensuring the same class proportion in both sets. Because of class imbalance, we calculated class weights inversely proportional to class frequency. These weights were applied during model training to penalize misclassification of the minority class more heavily.

To select the most informative predictors, we used Recursive Feature Elimination (RFE) with 10-fold cross-validation and random forest-based variable importance (rfFuncs). This method ranks features by their predictive contribution and iteratively removes the least important ones. The optimal subset of features (up to 20) was selected, and both the training and testing datasets were reduced accordingly.

We then trained a linear-kernel SVM using the e1071 package, incorporating the previously computed class weights. Unlike regular logistic regression or decision trees, the SVM attempts to find the optimal hyperplane that separates the two classes while accounting for class imbalance through a cost-sensitive learning strategy.

Performance measure table:

	TPR	FPR	Precision	Recall	F-measure	ROC	MCC	Kappa

Class Yes	0.865	0.300	0.146	0.865	0.250	0.567	0.276	0.170
Class No	0.700	0.135	0.989	0.700	0.819	0.567	0.276	0.170
Wt. Average	0.709	0.144	0.941	0.709	0.787	0.567	0.276	0.170

Confusion matrix:

	Reference: No	Reference: Yes
Prediction: No	871	10
Prediction: Yes	374	64

Model 9: Class Weights, Recursive Feature Elimination, and Random Forest

In this approach, we built a classification model that incorporates recursive feature elimination (RFE), class weighting, and a Random Forest classifier. The process began by splitting the preprocessed dataset into training and testing sets using stratified sampling to preserve class distribution. The data is split into 66% training and 34% testing using stratified sampling based on the Class column, ensuring the same class proportion in both sets. To address class imbalance, class weights were calculated inversely to class frequencies in the training data. These weights were later passed to the Random Forest model to penalize errors in the minority class more heavily.

For feature selection, we used Recursive Feature Elimination (RFE) with 10-fold cross-validation and random forest–based importance scores (rfFuncs). RFE iteratively removes the least important predictors, allowing us to identify a subset of up to 20 optimal features. Both the training and testing datasets were then reduced to include only these selected features, resulting in a more focused and potentially more robust model.

A Random Forest model was trained on the reduced training set using the randomForest package, with the previously computed class weights passed via the classwt parameter. Random Forest is an ensemble method that aggregates multiple decision trees to improve prediction accuracy and reduce variance, making it well-suited for classification tasks, especially in the presence of noisy or high-dimensional data.

Performance measure table:

	TPR	FPR	Precision	Recall	F-measure	ROC	MCC	Kappa
Class Yes	0.108	0.024	0.211	0.108	0.143	0.580	0.116	0.109
Class No	0.976	0.892	0.949	0.976	0.962	0.580	0.116	0.109
Wt. Average	0.927	0.843	0.907	0.927	0.916	0.580	0.116	0.109

Confusion matrix:

	Reference: No	Reference: Yes
Prediction: No	1215	66
Prediction: Yes	30	8

Model 10: Class Weights, Recursive Feature Elimination, and Naive Bayes Classifier

In this approach, we combined recursive feature elimination (RFE) with a Naive Bayes classifier. The modeling process began by splitting the preprocessed dataset into training and testing sets using stratified sampling to preserve class proportions. The data is split into 66% training and 34% testing using stratified sampling based on the Class column, ensuring the same class proportion in both sets. Unlike other models that use class weights, Naive Bayes inherently accounts for class imbalance through its built-in use of class priors, so no manual weighting was applied.

Feature selection was performed using Recursive Feature Elimination (RFE) with 10-fold cross-validation and random forest–based feature importance (rfFuncs). RFE iteratively removes the least important features and evaluates performance at each step to select the optimal subset. The best-performing subset of up to 20 features was retained, and both training and testing datasets were reduced to include only these selected features.

With the refined feature set, a Naive Bayes classifier was trained using the naiveBayes function from the e1071 package. Naive Bayes is a probabilistic model that assumes feature independence given the class label and is particularly efficient for high-dimensional data. It uses likelihoods and class prior probabilities to compute posterior probabilities for prediction.

Performance measure table:

	TPR	FPR	Precision	Recall	F-measure	ROC	MCC	Kappa
Class Yes	0.797	0.185	0.204	0.797	0.325	0.595	0.341	0.259
Class No	0.815	0.203	0.985	0.815	0.892	0.595	0.341	0.259
Wt. Average	0.814	0.202	0.942	0.814	0.860	0.595	0.341	0.259

Confusion matrix:

	Reference: No	Reference: Yes
Prediction: No	1015	15
Prediction: Yes	230	59

Model 11: Class Weights, Recursive Feature Elimination, and Neural Network Classifier

Model 11 employed a combination of Random Forest-based feature selection, random oversampling, and logistic regression. To mitigate the effects of class imbalance in the training set, we applied random oversampling by duplicating samples from the minority class ("Yes") to match the size of the majority class. This ensured that the classifier would not default to predicting only the majority class and could better learn decision boundaries involving underrepresented outcomes.

For feature selection, we trained a Random Forest classifier on the oversampled training set and extracted variable importance scores. The top-ranked features, based on mean decrease in Gini impurity, were selected for downstream modeling. This method aims to retain variables that contribute most to prediction accuracy while reducing dimensionality. After filtering, both training and testing datasets were subset to include only the selected features.

We then trained a logistic regression model using the reduced, oversampled dataset. Logistic regression is a linear model that estimates class probabilities and is highly interpretable, making it suitable for applications where transparency is important

Performance measure table:

	TPR	FPR	Precision	Recall	F-measure	ROC	MCC	Kappa
Class Yes	0.176	0.015	0.406	0.176	0.245	0.679	0.240	0.219
Class No	0.985	0.824	0.953	0.985	0.968	0.679	0.240	0.219
Wt. Average	0.939	0.779	0.922	0.939	0.928	0.679	0.240	0.219

Confusion matrix:

	Reference: No	Reference: Yes
Prediction: No	1015	15
Prediction: Yes	230	59

Model 12: Class Weights, Recursive Feature Elimination, and Decision Tree

Model 12 followed a similar design pipeline to Model 11 but replaced the logistic regression classifier with a Support Vector Machine (SVM). This model incorporated Random Forest feature selection, random oversampling, and linear SVM classification. Random oversampling was first applied to the training data to balance the class distribution by duplicating minority class examples, enabling the model to better detect positive cases without relying on class-weighted loss.

Feature selection was then conducted using a Random Forest classifier trained on the oversampled dataset. We extracted the most important variables based on Gini importance and

reduced both training and testing sets accordingly. Given the sensitivity of SVMs to feature scale, we applied z-score normalization to ensure standardized input across all predictors.

With the processed dataset, we trained a linear-kernel SVM using the e1071 package. While SVMs are effective at maximizing margin separation and can model complex boundaries with kernel tricks, this configuration uses a fixed linear kernel for interpretability and computational efficiency. Unlike logistic regression, SVM does not produce direct probability estimates but aims to optimize the separating hyperplane between classes.

Performance measure table:

	TPR	FPR	Precision	Recall	F-measure	ROC	MCC	Kappa
Class Yes	0.878	0.317	0.141	0.878	0.243	0.565	0.271	0.162
Class No	0.683	0.122	0.990	0.683	0.808	0.565	0.271	0.162
Wt. Average	0.694	0.133	0.942	0.694	0.776	0.565	0.271	0.162

Confusion matrix:

	Reference: No	Reference: Yes
Prediction: No	850	9
Prediction: Yes	395	65

Model 13: Class Weights, Random Forest, and Logistic Regression

In this approach, we developed a classification model combining random forest–based feature selection, class weighting, and logistic regression. The process began by splitting the preprocessed dataset into training and testing sets using stratified sampling to preserve the class distribution. The data is split into 66% training and 34% testing using stratified sampling based on the Class column, ensuring the same class proportion in both sets. Due to class imbalance, we calculated inverse class weights from the training set and scaled them to whole numbers to make them compatible with logistic regression. These weights were used during training to reduce bias against the minority class.

For feature selection, we trained a Random Forest model with 500 trees and extracted feature importance scores using the Mean Decrease in Gini index. We calculated the mean importance across all features and selected only those whose importance scores exceeded the average. This allowed us to retain a strong subset of informative variables while discarding those that contributed little to model performance. Both training and testing datasets were reduced to include only these selected features.

With the selected features and adjusted class weights, we trained a logistic regression model using the glm function with a binomial family. Predictions were generated as probabilities and converted to binary class labels using a 0.5 threshold. This approach combines the interpretability of logistic regression with the strength of ensemble-based feature filtering from random forests.

Performance measure table:

	TPR	FPR	Precision	Recall	F-measure	ROC	MCC	Kappa
Class Yes	0.838	0.397	0.112	0.838	0.197	0.548	0.206	0.109
Class No	0.603	0.162	0.984	0.603	0.748	0.548	0.206	0.109
Wt. Average	0.616	0.175	0.936	0.616	0.717	0.548	0.206	0.109

Confusion matrix:

	Reference: No	Reference: Yes
Prediction: No	751	12
Prediction: Yes	494	62

Model 14: Class Weights, Random Forest, and Support Vector Machine

In this approach, we developed a classification pipeline that integrates random forest–based feature selection, class weighting, and a linear Support Vector Machine (SVM) classifier. As with the other models, we began by splitting the preprocessed dataset into training and testing subsets using stratified sampling to preserve the class distribution. The data is split into 66% training and 34% testing using stratified sampling based on the Class column, ensuring the same class proportion in both sets. Given the presence of class imbalance, we computed inverse class weights from the training data and applied them during SVM training to reduce bias toward the majority class.

For feature selection, we trained a Random Forest model with 500 trees and extracted variable importance scores using the Mean Decrease in Gini index. We calculated the mean of all feature importance values and retained only those features whose importance exceeded the average. This approach allowed us to focus the model on the most informative predictors, improving generalization and reducing potential overfitting. The training and testing datasets were both reduced to include only the selected features.

With the refined feature set and class weights, we trained a linear-kernel SVM using the e1071 package. SVM is a powerful model for classification tasks that works by finding the optimal hyperplane to separate the classes. Incorporating class weights ensured that the model paid sufficient attention to the minority class while learning the decision boundary.

Performance measure table:

	TPR	FPR	Precision	Recall	F-measure	ROC	MCC	Kappa
Class Yes	0.919	0.477	0.103	0.919	0.185	0.547	0.203	0.093
Class No	0.523	0.081	0.991	0.523	0.684	0.547	0.203	0.093
Wt. Average	0.545	0.103	0.941	0.545	0.657	0.547	0.203	0.093

Confusion matrix:

	Reference: No	Reference: Yes
Prediction: No	651	6
Prediction: Yes	594	68

Model 15: Class Weights, Random Forest, and Random Forest

In this approach, we constructed a classification model that combines random forest–based feature selection, class weighting, and a Random Forest classifier. The workflow began by splitting the preprocessed dataset into training and testing sets using stratified sampling to preserve class distribution. The data is split into 66% training and 34% testing using stratified sampling based on the Class column, ensuring the same class proportion in both sets. To address the issue of class imbalance, we calculated inverse class weights from the training data and passed them to the model to place greater emphasis on the minority class during training.

Feature selection was carried out using a Random Forest model with 500 trees, from which we extracted importance scores based on the Mean Decrease in Gini index. We calculated the mean of these importance values and selected features with importance above the average. This filtered out less informative predictors and ensured that only the most relevant features were retained in both the training and testing datasets.

Using the reduced feature set, a second Random Forest model was trained with the previously computed class weights. Unlike standard configurations, the prediction threshold for classifying a sample as "Yes" was manually adjusted to 0.07 (rather than the default 0.5) to favor sensitivity and better accommodate the class imbalance. Predictions were generated based on the estimated probability of the positive class exceeding this threshold.

Performance measure table:

	TPR	FPR	Precision	Recall	F-measure	ROC	MCC	Kappa
Class Yes	0.649	0.203	0.159	0.649	0.256	0.567	0.244	0.182
Class No	0.797	0.351	0.974	0.797	0.877	0.567	0.244	0.182

Wt. Average	0.788	0.343	0.929	0.788	0.842	0.567	0.244	0.182

Confusion matrix:

	Reference: No	Reference: Yes
Prediction: No	994	27
Prediction: Yes	251	47

Model 16: Class Weights, Random Forest, and Naive Bayes Classifier

In this approach, we developed a classification model that integrates random forest–based feature selection with a Naive Bayes (NB) classifier. As with the other models, the dataset was first split into training and testing sets using stratified sampling to preserve the class distribution. The data is split into 66% training and 34% testing using stratified sampling based on the Class column, ensuring the same class proportion in both sets. Although class weights were computed to examine class distribution, they were not directly applied during Naive Bayes training, as NB inherently adjusts for class imbalance through its use of prior probabilities.

We employed a Random Forest model with 500 trees to calculate feature importance based on the Mean Decrease in Gini index. Features with above-average importance scores were selected, enabling us to reduce the feature space and focus the model on more informative predictors. Both the training and testing datasets were then reduced to include only these selected features for model training and evaluation.

With the refined dataset, a Naive Bayes model was trained using the naiveBayes function from the e1071 package. This classifier assumes conditional independence between features given the class label and combines likelihood estimates with class priors to compute posterior probabilities for prediction. It is computationally efficient and robust to noise, making it suitable even for high-dimensional data.

Performance measure table:

	TPR	FPR	Precision	Recall	F-measure	ROC	MCC	Kappa
Class Yes	0.324	0.082	0.190	0.324	0.240	0.574	0.190	0.182
Class No	0.918	0.676	0.958	0.918	0.938	0.574	0.190	0.182
Wt. Average	0.885	0.642	0.915	0.885	0.899	0.574	0.190	0.182

Confusion matrix:

	Reference: No	Reference: Yes

Prediction: No	1143	50
Prediction: Yes	102	24

Model 17: Class Weights, Random Forest, and Neural Network Classifier

In this approach, we constructed a classification model that incorporates random forest–based feature selection, class weighting, and a neural network (NN) classifier. As in previous models, we started by splitting the preprocessed dataset into training and testing sets using stratified sampling to preserve the class proportions. The data is split into 66% training and 34% testing using stratified sampling based on the Class column, ensuring the same class proportion in both sets. Due to class imbalance, we calculated inverse class weights from the training set and scaled them to whole numbers. These weights were applied during training to ensure fair treatment of the minority class.

To identify the most relevant features, we trained a Random Forest model with 500 trees and computed feature importance scores using the Mean Decrease in Gini index. We selected features whose importance exceeded the mean value, thereby filtering out less informative predictors. The training and testing datasets were then reduced to include only these selected features.

With the selected features and computed class weights, we trained a feedforward neural network using the nnet package. The model included one hidden layer with 6 neurons, a decay parameter of 0.15 for regularization, and a maximum of 200 iterations. Neural networks are well-suited for capturing complex, nonlinear relationships in data and benefit from carefully selected features and proper handling of class imbalance.

Performance measure table:

	TPR	FPR	Precision	Recall	F-measure	ROC	MCC	Kappa
Class Yes	0.324	0.082	0.190	0.324	0.240	0.574	0.190	0.182
Class No	0.918	0.676	0.958	0.918	0.938	0.574	0.190	0.182
Wt. Average	0.885	0.642	0.915	0.885	0.899	0.574	0.190	0.182

Confusion matrix:

	Reference: No	Reference: Yes
Prediction: No	1143	50
Prediction: Yes	102	24

Model 18: Class Weights, Random Forest, and Decision Tree

In this approach, we developed a classification model that integrates random forest–based feature selection, class weighting, and a decision tree classifier. The dataset was first split into training and testing sets using stratified sampling to preserve class proportions. The data is split into 66% training and 34% testing using stratified sampling based on the Class column, ensuring the same class proportion in both sets. Because the dataset is imbalanced, we computed inverse class weights from the training data and scaled them to whole numbers. These weights were then passed to the decision tree model to reduce bias against the minority class.

To select the most informative predictors, we trained a Random Forest model with 500 trees and extracted feature importance scores using the Mean Decrease in Gini index. We calculated the mean importance score and retained only those features with above-average importance. This reduced the dimensionality of the data while preserving critical information for classification. Both training and testing datasets were then filtered to include only the selected features.

With the reduced dataset and class weights, we trained a decision tree classifier using the rpart package. Decision trees are simple yet powerful models that partition the feature space based on information gain, making them highly interpretable. Incorporating class weights helped balance the tree's decision-making toward both majority and minority classes.

Performance measure table:

	TPR	FPR	Precision	Recall	F-measure	ROC	MCC	Kappa
Class Yes	0.811	0.318	0.132	0.811	0.226	0.558	0.226	0.144
Class No	0.682	0.189	0.984	0.682	0.806	0.558	0.226	0.144
Wt. Average	0.689	0.196	0.936	0.689	0.773	0.558	0.226	0.144

Confusion matrix:

	Reference: No	Reference: Yes
Prediction: No	849	14
Prediction: Yes	396	60

Model 19: Random oversampling, Lasso Feature Selection, and Logistic Regression

In this approach, we implemented a classification model that integrates random oversampling, Lasso-based feature selection, and logistic regression with regularization. As with the other models, we began by splitting the dataset into training and testing subsets using stratified sampling to preserve the class distribution. The data is split into 66% training and 34% testing using stratified sampling based on the Class column, ensuring the same class proportion in both sets.

Following oversampling, we performed feature selection using Lasso regression via the glmnet package. Lasso (L1 regularization) shrinks less informative coefficients to zero, effectively performing embedded feature selection. We identified the optimal penalty value (lambda.min) through cross-validation and extracted the non-zero coefficients to determine the most informative features. The training and testing datasets were then reduced to include only these selected features.

A logistic regression model was trained using the caret package with glmnet, allowing exploration of different levels of regularization through Ridge ($\alpha = 0$), Elastic Net ($\alpha = 0.5$), and Lasso ($\alpha = 1$). A 10-fold cross-validation was performed across a grid of λ values from 0.001 to 10, and the model was optimized based on the ROC metric to handle the imbalanced nature of the data.

Performance measure table:

	TPR	FPR	Precision	Recall	F-measure	ROC	MCC	Kappa
Class Yes	0.838	0.208	0.193	0.838	0.314	0.591	0.337	0.245
Class No	0.792	0.162	0.988	0.792	0.879	0.591	0.337	0.245
Wt. Average	0.795	0.165	0.943	0.795	0.847	0.591	0.337	0.245

Confusion matrix:

	Reference: No	Reference: Yes
Prediction: No	986	12
Prediction: Yes	259	62

Model 20: Random oversampling, Lasso Feature Selection, and Support Vector Machine

In this approach, we designed a classification model that combines random oversampling, Lasso-based feature selection, and a linear Support Vector Machine (SVM). The dataset was first split into training and testing sets using stratified sampling to preserve the class distribution. The data is split into 66% training and 34% testing using stratified sampling based on the Class column, ensuring the same class proportion in both sets.

After balancing, Lasso regression (L1 regularization) was used for feature selection. Using the glmnet package, a model was fitted with alpha = 1, and the optimal penalty parameter (lambda.min) was determined through cross-validation. Non-zero coefficients were extracted from the model, and the corresponding features were retained in both the training and test sets, effectively reducing dimensionality and focusing on the most relevant predictors.

Next, a linear SVM was trained using the caret package with svmLinear as the method. The model was tuned via 10-fold cross-validation using a grid of cost parameter values (C = 0.1, 1,

10), and performance was evaluated using the ROC metric, which is particularly suitable for imbalanced classification. This combination allowed the model to learn a robust decision boundary while leveraging both regularization and synthetic sampling.

Performance measure table:

	TPR	FPR	Precision	Recall	F-measure	ROC	MCC	Kappa
Class Yes	0.797	0.236	0.167	0.797	0.277	0.576	0.292	0.202
Class No	0.764	0.203	0.984	0.764	0.860	0.576	0.292	0.202
Wt. Average	0.766	0.205	0.939	0.766	0.827	0.576	0.292	0.202

Confusion matrix:

	Reference: No	Reference: Yes
Prediction: No	951	15
Prediction: Yes	294	59

Model 21: Random oversampling, Lasso Feature Selection, and Random Forest

In this approach, we built a classification model that combines random oversampling, Lasso-based feature selection, and a Random Forest classifier. As with the other models, the dataset was initially split into training and testing sets using stratified sampling to preserve class distribution. The data is split into 66% training and 34% testing using stratified sampling based on the Class column, ensuring the same class proportion in both sets.

Lasso regression (L1 regularization) was then applied to the oversampled training set using the glmnet package. The Lasso model selects a subset of informative predictors by shrinking irrelevant coefficients to zero. We extracted the non-zero coefficients at the optimal penalty value (lambda.min) and retained only the corresponding features for model training. The training and test sets were both reduced to include these selected features.

We then trained a Random Forest model using the caret package, tuning it with 10-fold cross-validation. The model was optimized using the ROC metric and a grid of mtry values (number of features considered at each split). This allowed the model to learn from a focused set of predictors while benefiting from the robustness and ensemble power of Random Forests.

Performance measure table:

	TPR	FPR	Precision	Recall	F-measure	ROC	MCC	Kappa
Class Yes	0.595	0.113	0.238	0.595	0.340	0.606	0.319	0.282
Class No	0.887	0.405	0.974	0.887	0.928	0.606	0.319	0.282

Wt. Average	0.870	0.389	0.932	0.870	0.895	0.606	0.319	0.282

Confusion matrix:

	Reference: No	Reference: Yes
Prediction: No	1104	30
Prediction: Yes	141	44

Model 22: Random oversampling, Lasso Feature Selection, and Naive Bayes Classifier

In this approach, we developed a classification model by integrating random oversampling, Lasso-based feature selection, and a Naive Bayes (NB) classifier. The dataset was split into training and testing sets using stratified sampling to preserve class distribution. The data is split into 66% training and 34% testing using stratified sampling based on the Class column, ensuring the same class proportion in both sets.

Next, Lasso regression was applied for feature selection. Using the glmnet package with alpha set to 1 (Lasso), the model was cross-validated to determine the optimal penalty (lambda.min). Non-zero coefficients were extracted, and their corresponding features were retained in both training and test sets, effectively reducing the dimensionality and focusing on informative variables**.**

A Naive Bayes model was trained on the reduced feature set using the naive_bayes method from the klaR or naive bayes package through the caret framework. We performed hyperparameter tuning via 10-fold cross-validation, exploring combinations of Laplace smoothing, kernel usage, and adjustment factors. The model was optimized based on the ROC metric, making it suitable for handling imbalanced data.

Performance measure table:

	TPR	FPR	Precision	Recall	F-measure	ROC	MCC	Kappa
Class Yes	0.878	0.269	0.163	0.878	0.274	0.576	0.305	0.198
Class No	0.731	0.122	0.990	0.731	0.841	0.576	0.305	0.198
Wt. Average	0.739	0.130	0.944	0.739	0.809	0.576	0.305	0.198

Confusion matrix:

	Reference: No	Reference: Yes
Prediction: No	910	9
Prediction: Yes	335	65

Model 23: Random oversampling, Lasso Feature Selection, and Neural Network Classifier

In this approach, we constructed a classification model by combining random oversampling, Lasso-based feature selection, and a neural network (NN) classifier. The dataset was first split into training and testing sets using stratified sampling to maintain class proportions. The data is split into 66% training and 34% testing using stratified sampling based on the Class column, ensuring the same class proportion in both sets.

We then performed feature selection using Lasso regression with the glmnet package. Lasso (L1 regularization) shrinks less relevant feature coefficients to zero, allowing us to isolate the most informative predictors. We extracted the non-zero coefficients at the optimal penalty (lambda.min) and used them to reduce both training and test sets to only include the selected features.

A neural network was trained on this reduced dataset using the nnet package. The model was configured with one hidden layer containing six neurons, a weight decay of 0.15 to prevent overfitting, and a maximum of 200 iterations. This architecture allows the model to capture non-linear relationships within the data while remaining relatively simple and interpretable.

Performance measure table:

	TPR	FPR	Precision	Recall	F-measure	ROC	MCC	Kappa
Class Yes	0.689	0.218	0.159	0.689	0.258	0.578	0.305	0.183
Class No	0.782	0.311	0.977	0.782	0.869	0.568	0.253	0.183
Wt. Average	0.777	0.306	0.931	0.777	0.836	0.568	0.256	0.183

Confusion matrix:

	Reference: No	Reference: Yes
Prediction: No	947	23
Prediction: Yes	271	51

Model 24: Random oversampling, Lasso Feature Selection, and Decision Tree

In this approach, we developed a classification model by combining random oversampling, Lasso-based feature selection, and a decision tree classifier. As with earlier models, the dataset was initially split into training and testing subsets using stratified sampling to preserve class proportions. The data is split into 66% training and 34% testing using stratified sampling based on the Class column, ensuring the same class proportion in both sets.

Feature selection was then performed using Lasso regression (L1 regularization) via the glmnet package. The Lasso model was tuned using cross-validation to find the optimal penalty

(lambda.min). Non-zero coefficients were extracted to identify the most important features, and the training and testing sets were reduced accordingly.

With the selected features, we trained a decision tree model using the rpart package. Decision trees are interpretable models that recursively split the feature space based on information gain, creating a flowchart-like structure of decisions. The model was trained without explicit class weights, as the class distribution had already been balanced.

Performance measure table:

	TPR	FPR	Precision	Recall	F-measure	ROC	MCC	Kappa
Class Yes	0.419	0.143	0.148	0.419	0.219	0.555	0.174	0.149
Class No	0.857	0.581	0.961	0.857	0.906	0.555	0.174	0.149
Wt. Average	0.832	0.557	0.916	0.832	0.868	0.555	0.174	0.149

Confusion matrix:

	Reference: No	Reference: Yes
Prediction: No	1067	43
Prediction: Yes	178	31

Model 25: Random oversampling, Recursive Feature Elimination, and Logistic Regression

In this approach, we created a classification model that integrates random oversampling, recursive feature elimination (RFE), and a logistic regression model with regularization. As in prior models, we began by splitting the dataset into training and testing subsets using stratified sampling to preserve the class distribution. The data is split into 66% training and 34% testing using stratified sampling based on the Class column, ensuring the same class proportion in both sets.

Following oversampling, we applied recursive feature elimination using the rfe function from the caret package, with Random Forests as the base estimator. This process systematically removed less relevant features and identified a subset that optimizes performance through cross-validation. The training and testing datasets were reduced to include only these selected features.

We then trained a logistic regression model using the glmnet method through the caret framework. The model was tuned using a grid search across different values of alpha (0 = Ridge, 0.5 = Elastic Net, 1 = Lasso) and lambda (penalty strength), with 10-fold cross-validation. The model was optimized for the ROC metric, which is especially suitable for imbalanced classification tasks.

Performance measure table:

	TPR	FPR	Precision	Recall	F-measure	ROC	MCC	Kappa
Class Yes	0.838	0.228	0.179	0.838	0.295	0.583	0.319	0.223
Class No	0.772	0.162	0.988	0.772	0.867	0.583	0.319	0.223
Wt. Average	0.776	0.166	0.642	0.776	0.834	0.583	0.319	0.223

Confusion matrix:

	Reference: No	Reference: Yes
Prediction: No	968	15
Prediction: Yes	277	59

Model 26: Random oversampling, Recursive Feature Elimination, and Support Vector Machine

In this approach, we combined random oversampling with recursive feature elimination (RFE) and a Support Vector Machine (SVM) classifier using a linear kernel. The dataset was initially split into training and testing sets using stratified sampling to maintain consistent class distributions. The data is split into 66% training and 34% testing using stratified sampling based on the Class column, ensuring the same class proportion in both sets.

After balancing the dataset, we performed feature selection using RFE with Random Forests as the base estimator. This method iteratively evaluates subsets of features through cross-validation to identify the most predictive variables. Once the top features were selected, we reduced both the training and test sets to include only these features.

An SVM classifier with a linear kernel was then trained on the selected features. The linear kernel was chosen for its simplicity and ability to handle linearly separable data efficiently. Unlike some other models in this study, the SVM here did not involve explicit class weights.

Performance measure table:

	TPR	FPR	Precision	Recall	F-measure	ROC	MCC	Kappa
Class Yes	0.851	0.266	0.160	0.851	0.269	0.574	0.294	0.193
Class No	0.734	0.149	0.988	0.734	0.842	0.574	0.294	0.193
Wt. Average	0.741	0.156	0.942	0.741	0.810	0.574	0.294	0.193

Confusion matrix:

	Reference: No	Reference: Yes

Prediction: No	905	10
Prediction: Yes	340	64

Model 27: Random oversampling, Recursive Feature Elimination, and Random Forest

In this approach, we integrated random oversampling with recursive feature elimination (RFE) and a Random Forest classifier to build a robust model for classifying imbalanced data. The dataset was first split into training and testing sets using stratified sampling to maintain class proportions. The data is split into 66% training and 34% testing using stratified sampling based on the Class column, ensuring the same class proportion in both sets.

Next, we used recursive feature elimination with Random Forests as the base learner to identify the most important features. RFE iteratively removes less significant predictors based on model performance, resulting in a reduced feature set optimized for generalization. Both training and testing data were then subset to include only these selected features.

A Random Forest classifier was trained on the reduced training set using default parameters. This ensemble method leverages multiple decision trees and aggregates their predictions, making it robust to overfitting and well-suited for classification tasks with complex relationships among features.

Performance measure table:

	TPR	FPR	Precision	Recall	F-measure	ROC	MCC	Kappa
Class Yes	0.568	0.117	0.223	0.568	0.321	0.598	0.296	0.261
Class No	0.883	0.432	0.972	0.883	0.925	0.598	0.296	0.261
Wt. Average	0.865	0.415	0.930	0.865	0.891	0.598	0.296	0.261

Confusion matrix:

	Reference: No	Reference: Yes
Prediction: No	1101	29
Prediction: Yes	144	45

Model 28: Random oversampling, Recursive Feature Elimination, and Naive Bayes Classifier

In this model, we combined random oversampling, recursive feature elimination (RFE), and a Naive Bayes classifier to tackle class imbalance and perform efficient classification. We started by splitting the dataset into training and testing sets using stratified sampling to preserve the

proportion of the Class variable. The data is split into 66% training and 34% testing using stratified sampling based on the Class column, ensuring the same class proportion in both sets.

We then performed feature selection using RFE with a Random Forest base learner. RFE iteratively evaluates subsets of features to select the most informative predictors. The resulting reduced feature set was used to subset both the training and testing datasets.

Next, we trained a Naive Bayes classifier using the reduced training set. Naive Bayes is a probabilistic classifier that assumes conditional independence among features and is well-suited for high-dimensional data. As it internally accounts for class priors, no additional class weight adjustments were necessary.

Performance measure table:

	TPR	FPR	Precision	Recall	F-measure	ROC	MCC	Kappa
Class Yes	0.905	0.284	0.160	0.905	0.271	0.576	0.307	0.194
Class No	0.716	0.095	0.992	0.716	0.832	0.576	0.307	0.194
Wt. Average	0.727	0.105	0.945	0.77	0.801	0.576	0.307	0.194

Confusion matrix:

	Reference: No	Reference: Yes
Prediction: No	877	7
Prediction: Yes	368	67

Model 29: Random oversampling, Recursive Feature Elimination, and Neural Network Classifier

In this model, we applied random oversampling, recursive feature elimination (RFE), and a neural network classifier to address class imbalance and enhance predictive performance. The data was initially split into 66% training and 34% testing using stratified sampling to maintain the class distribution in both sets.

To address the imbalance in the training data, generated synthetic samples of the minority class, effectively balancing the class distribution. Feature selection was then performed using RFE with a Random Forest base estimator. RFE evaluates different subsets of predictors through cross-validation and selects the most relevant features. The training and testing datasets were then reduced to these selected features.

A neural network model was trained using the nnet function with 6 hidden units and a decay value of 0.15 for regularization, and trained for a maximum of 200 iterations. The model was trained on the balanced, reduced dataset and used to predict the class labels of the test data.

Performance measure table:

	TPR	FPR	Precision	Recall	F-measure	ROC	MCC	Kappa
Class Yes	0.905	0.284	0.160	0.905	0.271	0.576	0.307	0.194
Class No	0.716	0.095	0.992	0.716	0.832	0.576	0.307	0.194
Wt. Average	0.727	0.105	0.945	0.77	0.801	0.576	0.307	0.194

Confusion matrix:

	Reference: No	Reference: Yes
Prediction: No	877	7
Prediction: Yes	368	67

Model 30: Random oversampling, Recursive Feature Elimination, and Decision Tree

In this model, we integrated random oversampling, recursive feature elimination (RFE), and a decision tree classifier to address the class imbalance and improve feature relevance for classification. We began by splitting the preprocessed dataset into training and testing sets with a 66%–34% split using stratified sampling to maintain class distribution.

To mitigate the class imbalance, we generated synthetic instances of the minority class to ensure a more balanced dataset. Next, recursive feature elimination was performed using a random forest-based ranking and 10-fold cross-validation. This process identified the most informative predictors, which were then retained in both training and testing datasets.

A decision tree model was trained on the selected features using the rpart algorithm with the default settings. This model was then evaluated on the test set.

Performance measure table:

	TPR	FPR	Precision	Recall	F-measure	ROC	MCC	Kappa
Class Yes	0.446	0.164	0.139	0.446	0.212	0.551	0.169	0.139
Class No	0.836	0.554	0.962	0.836	0.895	0.551	0.169	0.139
Wt. Average	0.814	0.532	0.916	0.814	0.856	0.551	0.169	0.139

Confusion matrix:

	Reference: No	Reference: Yes
Prediction: No	1041	41
Prediction: Yes	204	33

Model 31: Random oversampling, Random Forest, and Logistic Regression

In this model, we combined random oversampling, feature selection via random forest, and logistic regression with regularization to classify imbalanced data.

After splitting the dataset into 66% training and 34% testing sets using stratified sampling, we improved class balance and helped prevent model bias toward the majority class.

Next, random forest feature importance was used to select predictive features. A random forest model was trained on the balanced training data, and features with importance scores above the mean MeanDecreaseGini threshold were retained.

A logistic regression model with regularization (ridge, elastic net, and lasso) was trained using 10-fold cross-validation and evaluated based on ROC performance. The best model from the glmnet family was selected from a grid search over alpha and lambda hyperparameters.

Performance measure table:

	TPR	FPR	Precision	Recall	F-measure	ROC	MCC	Kappa
Class Yes	0.865	0.238	0.178	0.865	0.295	0.584	0.324	0.223
Class No	0.762	0.135	0.990	0.762	0.851	0.584	0.324	0.223
Wt. Average	0.768	0.141	0.944	0.768	0.829	0.584	0.324	0.223

Confusion matrix:

	Reference: No	Reference: Yes
Prediction: No	949	10
Prediction: Yes	296	64

Model 32: Random oversampling, Random Forest, and Support Vector Machine

In this model, we combined random oversampling, random forest-based feature selection, and a support vector machine (SVM) classifier to address class imbalance and improve classification accuracy.

The data was first split using stratified sampling, allocating 66% for training and 34% for testing. Next, we applied feature selection via random forest importance scores. A random forest model was trained on the oversampled training data, and features with above-average MeanDecreaseGini values were selected for model training.

We then trained a linear SVM model on the reduced feature set from the balanced data. The trained model was used to predict outcomes on the test set.

Performance measure table:

	TPR	FPR	Precision	Recall	F-measure	ROC	MCC	Kappa
Class Yes	0.865	0.300	0.146	0.865	0.250	0.567	0.276	0.170
Class No	0.700	0.135	0.989	0.700	0.820	0.567	0.276	0.170
Wt. Average	0.709	0.144	0.941	0.709	0.787	0.567	0.276	0.170

Confusion matrix:

	Reference: No	Reference: Yes
Prediction: No	871	10
Prediction: Yes	374	64

Model 33: Random oversampling, Random Forest, and Random Forest

In this model, we combined random oversampling, random forest-based feature selection, and a random forest classifier to address class imbalance and improve classification performance.

The dataset was first split using stratified sampling, allocating 66% for training and 34% for testing.

We then performed feature selection using random forest importance scores. A random forest model was trained on the oversampled training data, and features with above-average MeanDecreaseGini values were selected as the most informative predictors.

Using this reduced feature set, we trained a random forest classifier. Predictions were made on the test set using the trained model.

Performance measure table:

	TPR	FPR	Precision	Recall	F-measure	ROC	MCC	Kappa
Class Yes	0.676	0.133	0.231	0.676	0.345	0.605	0.337	0.285
Class No	0.867	0.324	0.978	0.867	0.919	0.605	0.337	0.285
Wt. Average	0.856	0.314	0.934	0.856	0.887	0.605	0.337	0.285

Confusion matrix:

	Reference: No	Reference: Yes
Prediction: No	1079	24
Prediction: Yes	166	50

Model 34: Random oversampling, Random Forest, and Naive Bayes Classifier

In this model, we combined random oversampling, random forest-based feature selection, and a Naive Bayes (NB) classifier to address class imbalance and improve predictive performance.

The dataset was split using stratified sampling, allocating 66% of the data for training and 34% for testing. Feature selection was then conducted using random forest importance scores. A random forest model was trained on the oversampled data, and features with above-average MeanDecreaseGini values were selected as the most informative.

A Naive Bayes model was trained on the selected features using the balanced dataset. Predictions were generated on the test set using this trained model.

Performance measure table:

	TPR	FPR	Precision	Recall	F-measure	ROC	MCC	Kappa
Class Yes	0.865	0.263	0.164	0.865	0.275	0.576	0.303	0.200
Class No	0.737	0.135	0.989	0.737	0.845	0.576	0.303	0.200
Wt. Average	0.744	0.142	0.943	0.745	0.813	0.576	0.303	0.200

Confusion matrix:

	Reference: No	Reference: Yes
Prediction: No	918	10
Prediction: Yes	327	64

Model 35: Random oversampling, Random Forest, and Neural Network Classifier

In this model, we combined random oversampling, random forest-based feature selection, and a neural network (NN) classifier to address class imbalance and enhance classification accuracy.

The data was first split using stratified sampling, with 66% allocated for training and 34% for testing. Feature selection was conducted using random forest importance scores. A random forest was trained on the oversampled data, and variables with above-average MeanDecreaseGini values were selected as the most relevant predictors.

A neural network with one hidden layer (size = 6, decay = 0.15) was trained on the reduced feature set from the balanced data. This model was used to generate predictions on the test set.

Performance measure table:

	TPR	FPR	Precision	Recall	F-measure	ROC	MCC	Kappa
Class Yes	0.824	0.304	0.139	0.824	0.237	0.562	0.254	0.156
Class No	0.696	0.176	0.985	0.696	0.815	0.562	0.254	0.156

Wt. Average	0.703	0.183	0.938	0.703	0.783	0.562	0.254	0.156

Confusion matrix:

	Reference: No	Reference: Yes
Prediction: No	866	13
Prediction: Yes	379	61

Model 36: Random oversampling, Random Forest, and Decision Tree

In this model, we integrated random oversampling, random forest-based feature selection, and a decision tree classifier to mitigate class imbalance and improve classification performance.

The dataset was initially split using stratified sampling, with 66% reserved for training and 34% for testing. We then conducted feature selection using random forest importance scores. A random forest model was trained on the oversampled data, and features with MeanDecreaseGini values above the average were selected for further modeling.

A decision tree classifier was trained on the reduced, balanced training set. The trained model was subsequently used to make predictions on the test dataset.

Performance measure table:

	TPR	FPR	Precision	Recall	F-measure	ROC	MCC	Kappa
Class Yes	0.554	0.182	0.153	0.554	0.240	0.561	0.213	0.166
Class No	0.818	0.446	0.969	0.818	0.818	0.561	0.213	0.166
Wt. Average	0.803	0.431	0.923	0.803	0.803	0.561	0.213	0.166

Confusion matrix:

	Reference: No	Reference: Yes
Prediction: No	1018	33
Prediction: Yes	227	41

5. Discussion

This project evaluated the performance of 36 classification pipelines for predicting whether individuals experience difficulty living independently using data from the 2023 American Community Survey. The primary challenges in the dataset were class imbalance, high dimensionality, and the need for careful preprocessing to ensure fair and interpretable results.

The minority class ("Yes") comprised a small proportion of the total data, and unbalanced models trained on the raw data quickly revealed a major limitation: very low recall for the "Yes" class, often below 10%. To address this, two balancing strategies—class weighting and random oversampling—were applied. Random oversampling, which increases the representation of the minority class by duplicating samples, proved consistently more effective at improving recall and F1-scores for the "Yes" class across most classifiers.

Feature selection was another critical factor in model performance. Among the three methods tested—Lasso regression, Recursive Feature Elimination (RFE), and Random Forest-based feature importance—RFE consistently produced the best results, particularly when paired with random oversampling and simpler models like logistic regression or Naive Bayes. Lasso also performed well, especially when paired with regularized models like logistic regression and support vector machines, but in some cases it was overly aggressive in filtering out useful features. Random Forest feature selection worked best with tree-based models but led to more variable results elsewhere.

Across all configurations, logistic regression emerged as the most balanced and reliable classifier, especially when combined with random oversampling and either RFE or Lasso. Two configurations in particular stood out: Model 4 (random oversampling + RFE + logistic regression) and Model 19 (random oversampling + Lasso + logistic regression).

Model 4 achieved a TPR of 0.824 for the "Yes" class and 0.811 for the "No" class, with a weighted average F1-score of 0.859, MCC of 0.351, and Kappa of 0.264. These metrics reflect strong overall balance and minimal trade-off between sensitivity and specificity. Model 19, while using Lasso instead of RFE, produced a slightly higher weighted F1-score of 0.847 and a higher "No" class TPR of 0.792, and a "Yes" TPR of 0.838. It also had a similar MCC of 0.310. The trade-off is that Model 4's performance was slightly more stable and interpretable due to the stepwise nature of RFE, but Model 19 may be better suited for maximizing aggregate classification accuracy.

Outside of logistic regression, Naive Bayes models also performed competitively. In particular, a Naive Bayes model with Lasso and random oversampling achieved a "Yes" class TPR of 0.878, though its "No" class TPR dropped to 0.731, indicating reduced overall balance. Still, its simplicity and relatively strong performance make it an attractive option when the cost of false negatives is high.

Support vector machines (SVMs) showed moderate results and were most successful when paired with Lasso or RFE. One configuration reached a "Yes" class TPR of 0.797 and a weighted F1 of 0.827, though SVMs were generally outperformed by logistic regression and were more sensitive to scaling and parameter tuning. Random Forests, on the other hand, often underperformed despite their complexity. For example, Model 9, which used class weighting and Lasso with a Random Forest classifier, achieved a "Yes" class TPR of only 0.108, highlighting

the model's tendency to overfit to the majority class. Even with oversampling, Random Forests rarely approached the recall or F1-scores of simpler models.

Decision trees, while interpretable, lacked robustness. Their recall for the "Yes" class remained low across most configurations, and they did not yield any top-performing models. Neural networks similarly fell short; despite being the most complex models evaluated, they failed to outperform simpler alternatives and were more prone to instability, even when class weights or oversampling were applied. Their performance was also more sensitive to hyperparameter tuning and less interpretable.

Overall, the results reinforce the idea that data-centric modeling decisions—like balancing and feature selection—have a greater impact on performance than model complexity alone. Both Model 4 and Model 19, which used logistic regression with different feature selection strategies, consistently outperformed more sophisticated methods in terms of both recall and interpretability. Their success illustrates how thoughtful preprocessing and straightforward modeling can effectively address the challenges posed by real-world, imbalanced classification tasks.

6. Conclusion

The aim of this project was to build effective classification models for identifying individuals who have difficulty living independently, using a real-world dataset characterized by class imbalance and high dimensionality. Through a systematic pipeline of data cleaning, feature engineering, class balancing, and model evaluation, we explored 36 different configurations that combined two balancing methods, three feature selection strategies, and six classifiers. Among these, random oversampling and Recursive Feature Elimination (RFE) stood out as the most impactful preprocessing techniques, consistently improving sensitivity and F1-scores across both classes.

Our final and best model—Model 4—used logistic regression with RFE-selected features on a training set balanced via random oversampling. This model achieved a true positive rate of 82.4% for the "Yes" class, and 81.1% for the "No" class, along with a weighted F1-score of 85.9% and an MCC of 0.351. Compared to other models, Model 4 maintained consistent performance across sensitivity, precision, and overall accuracy while preserving interpretability, a valuable characteristic in applied settings. Its ability to identify underrepresented cases without sacrificing majority-class performance demonstrates the effectiveness of using simple models paired with strong preprocessing.

Interestingly, more complex classifiers such as Random Forest and neural networks did not yield improved results despite multiple tuning efforts. This further reinforced that model quality depends more on preprocessing and class balancing than algorithm complexity. By focusing on balanced datasets, feature reduction, and clear evaluation metrics, we were able to deliver a

model that is both accurate and understandable. This makes it suitable for use in fields like healthcare and social services, where decision-makers require transparency and fairness.

Looking forward, future improvements could include exploring synthetic sampling methods like SMOTE, which was not used in this project, implementing cost-sensitive learning, or experimenting with ensemble techniques. Incorporating domain-specific insights into feature selection might also help capture more nuanced patterns in the data. Overall, this project illustrates how thoughtful design—particularly in preprocessing and evaluation—can lead to high-performing models that are both fair and practical in real-world classification problems involving vulnerable populations.