DATA MINING
TERM PROJECT

Data Mining Analysis for Independent Living Classification Using American Community Survey (ACS) Data
(Final Report)

By

1. **Utkarsh Roy** (BU ID: U69501306)
2. **Siddhraj Parmar** (BU ID: U35065911)

Dr. Jae Young Lee
BU MET, Spring 2025

Table of Contents

1. Introduction

This project focuses on building predictive models to determine whether individuals may experience difficulty in independent living based on data from the 2023 American Community Survey (ACS) Public Use Microdata Sample (PUMS). The dataset used for this task, named project_data.csv, contains several demographic and socioeconomic attributes describing thousands of individuals. The target variable, labeled Class, indicates whether an individual has reported difficulty living independently, marked as either "Yes" or "No".

The primary objective is to create a comprehensive machine learning pipeline that can accurately classify individuals based on their attributes. To achieve this, we implemented a multi-stage approach involving data cleaning, transformation, class balancing, feature selection, model training, and performance evaluation. The classification models were built using multiple algorithms such as Logistic Regression, Support Vector Machines, Naive Bayes, KNN, Neural Networks, GBM, and GLMNet.

To account for class imbalance, we applied both undersampling and SMOTE-based oversampling, followed by three distinct feature selection methods Boruta, Information Gain, and LASSO. A total of 36 models (6 classifiers × 3 feature selection methods × 2 balancing strategies) were trained and evaluated using a consistent performance evaluation framework.

This report documents the complete data mining workflow: from preprocessing raw survey data and handling missing values to evaluating model performance across different combinations of sampling strategies and feature sets. The end goal is to identify the most effective model for predicting independent living difficulty and understand which features are most informative for this task.

2. Dataset Description

The dataset is a modified version of the 2023 American Community Survey (ACS) data. It includes various socio-economic, demographic, and health-related attributes that help predict independent living difficulty.

- Dataset Name: project_data.csv.
- Downloaded from: https://www2.census.gov/programs-surveys/acs/data/pums/2023/1-Year/
- Source: American Community Survey (ACS) 2023
- Number of Records: 4,318 individuals (tuples)
- Number of Features: 117 variables
- Target Variable: Class (Binary: 1 = Yes, 0 = No)

Each row in the dataset represents an individual with attributes categorized into:

The class variable (Class) determines whether an individual has difficulty living independently. This serves as the target variable for future classification models.

3. Tools and Libraries Used

This project uses R and a comprehensive set of packages for data manipulation, visualization, feature engineering, model building, and evaluation. Here's a breakdown of the libraries that powered the entire classification workflow:

1. tidyverse / dplyr
   - Purpose: A collection of packages (including dplyr, ggplot2, readr, etc.) designed for data science. dplyr was primarily used for data wrangling.
   - Key Functions: select(), filter(), mutate(), arrange(), group_by(), summarise().
   - Benefits: Simplifies complex data manipulation tasks with clean, readable syntax and is optimized for speed.

2. caret
   - Purpose: A comprehensive package for training and evaluating machine learning models.
   - Key Functions: train(), createDataPartition(), preProcess(), confusionMatrix(), trainControl().
   - Benefits: Standardizes the interface for numerous machine learning algorithms and automates processes like cross-validation and hyperparameter tuning.

3. corrplot
   - Purpose: Visualizes correlation matrices to assess feature relationships.
   - Key Features: corrplot() supports various methods such as circles, color tiles, or numbers, making patterns in data easy to interpret.
   - Benefits: Facilitates identification and elimination of multicollinear features that may hurt model performance.

4. ggplot2
   - Purpose: Constructs elegant data visualizations using the Grammar of Graphics.
   - Key Features: Layered structure for adding geoms, scales, and facets. Highly customizable.
   - Benefits: Allows for detailed, multi-dimensional plots that support exploratory data analysis and storytelling.

5. reshape2
   - Purpose: Reshapes data between wide and long formats.
   - Key Functions: melt(), dcast().
   - Benefits: Prepares datasets for aggregation, analysis, and visualization workflows.

6. rsample
   - Purpose: Handles data splitting and resampling.
   - Key Features: initial_split(), training(), testing().
   - Benefits: Enables reproducible train-test splits and supports stratification for balanced class distribution.

7. ROSE
   - Purpose: Handles class imbalance using resampling techniques.
   - Key Functions: ovun.sample().
   - Benefits: Allows undersampling of majority class or SMOTE of minority class to improve classifier performance in imbalanced datasets.

8. smotefamily
   - Purpose: Implements SMOTE (Synthetic Minority Over-sampling Technique).
   - Key Functions: SMOTE().

- Benefits: Generates synthetic samples for minority classes, improving model generalization in skewed datasets.

9. Boruta
   - Purpose: All-relevant feature selection using a wrapper algorithm built around random forests.
   - Key Functions: Boruta(), getSelectedAttributes().
   - Benefits: Selects truly important features while eliminating noise, improving model interpretability and efficiency.

10. FSelector
    - Purpose: Performs univariate feature ranking using information-theoretic methods.
    - Key Functions: information.gain().
    - Benefits: Identifies the most informative features relative to the target class based on entropy reduction.

11. glmnet
    - Purpose: Fits penalized generalized linear models using LASSO and Elastic Net.
    - Key Functions: glmnet(), cv.glmnet().
    - Benefits: Combines variable selection and regularization to handle high-dimensional data effectively, reducing overfitting.

12. pROC
    - Purpose: Analyzes the performance of binary classification models using ROC curves.
    - Key Functions: roc(), auc().
    - Benefits: Provides insights into classifier sensitivity and specificity, and enables comparison between models using AUC.

13. nnet
    - Purpose: Implements single-layer feedforward neural networks.
    - Key Features: nnet() supports flexible configurations of hidden layers and decay (regularization).
    - Benefits: Provides a simple neural network framework integrated with caret for tuning and evaluation.

14. naivebayes
    - Purpose: Trains a Naive Bayes classifier assuming feature independence.
    - Key Features: Supports both numeric and categorical features, and computes conditional probabilities efficiently.
    - Benefits: A lightweight and interpretable model ideal for baseline comparison and categorical data.

15. e1071
    - Purpose: A multipurpose package that includes Support Vector Machines and other classifiers.
    - Key Functions: svm(), tune().
    - Benefits: Implements SVM with kernel support, and can be used via caret for hyperparameter tuning.

16. gbm
    - Purpose: Builds Gradient Boosting Models for regression and classification.
    - Key Features: gbm() allows iterative model updates, early stopping, and interaction control.
    - Benefits: Robust against overfitting, handles complex feature interactions, and often outperforms simpler classifiers.

17. kernlab

- Purpose: Provides a suite of kernel-based machine learning methods, especially for Support Vector Machines (SVMs).
- Key Features:
  - Functions like ksvm() and sigest() for model fitting and estimating kernel parameters.
  - Supports a wide variety of kernel functions (RBF, polynomial, linear, etc.).
- Benefits:
  - Allows fine-tuning of sigma and C parameters in SVM, improving classification performance.
  - Offers flexibility and robustness in building non-linear models.

4. Data Preprocessing Steps

4.1 Loading the Dataset:

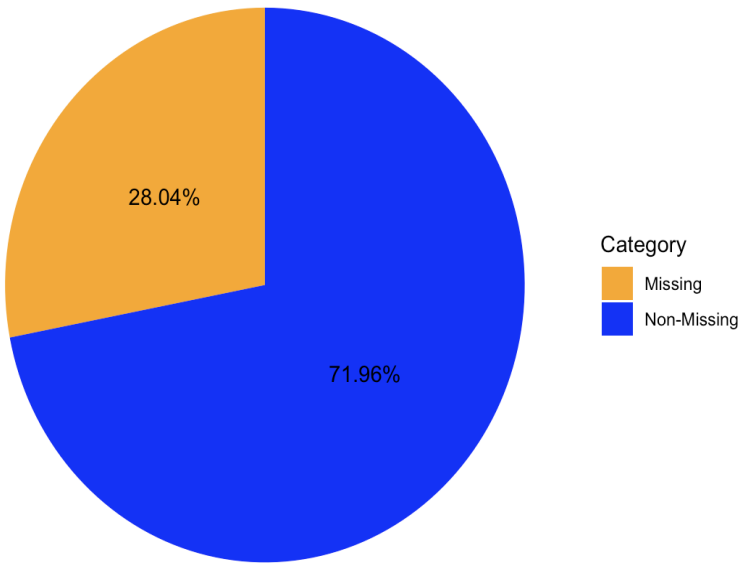
- The dataset (project_data.csv) was loaded using read.csv().
- Structure (str(data)) and summary statistics (summary(data)) were examined.

4.2 Handling Missing Values

Handling missing values is essential to prevent biases and maintain dataset integrity. The missing values were addressed as follows:

4.2.1 Identifying Missing Values

- The total missing values were calculated for each column.
- A bar chart was created to visualize the count of missing values per column.
- A pie chart was generated to display the proportion of missing vs. non-missing values in the dataset.



The pie chart illustrates the overall proportion of missing vs. non-missing values in the dataset.

Key Observations:

- 28.04% of the dataset contains missing values, which is a significant proportion that needs to be handled properly.
- 71.96% of the dataset consists of non missing values, ensuring that there is still substantial data available for analysis.
- The visualization confirms that a structured approach is required, including removal of highly missing columns and proportion based imputation for the remaining missing values.

4.2.2 Removing Features with Excessive Missing Values

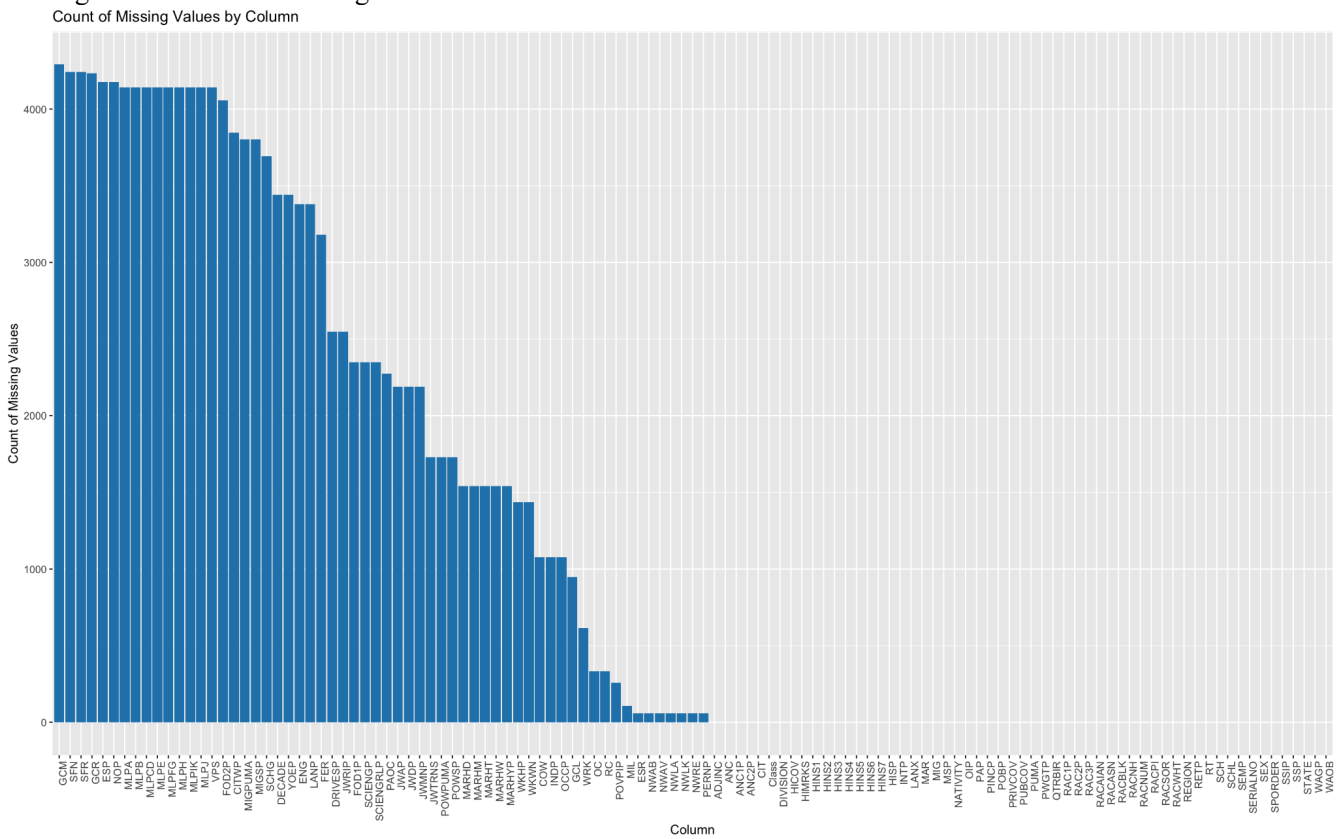
- Columns with more than 2,000 missing values were removed.
- Dropping such columns reduces data dimensionality, making preprocessing faster and models more efficient.
- This threshold was chosen to preserve informative features while removing columns with excessive missingness.

The bar chart visually represents the count of missing values for each feature in the dataset. Columns with a higher number of missing values appear on the left, while those with fewer missing values appear on the right.

Key Observations:

- Some features have over 4,000 missing values, indicating they may not be useful for model training.
- A significant number of variables contain more than 2,000 missing values, making them candidates for removal.
- Some columns have very few missing values, which can be handled using proportion-based imputation instead of deletion.

This analysis guided our decision to remove columns with more than 2,000 missing values while imputing missing values for the remaining ones.



4.2.3 Imputing Missing Values

Different strategies were applied for categorical and numerical variables:
Identifying Numerical and Categorical Variables:

- Numeric variables: Selected using sapply(data_f, is.numeric).
- Categorical variables: Selected using sapply(data_f, is.character).
- No additional encoding was needed, as categorical data was already in numeric format.

4.2.4 Numerical Variables & Categorical Variables:

- Both numerical and categorical variables (stored as numeric) were imputed using proportion-based sampling from observed distributions.
- Imputed using proportion-based imputation, maintaining the original distribution of values.
- For special cases like JWMNP (Journey to Work Minimum Period) and WKWN (Weeks Worked in a Year), missing values were replaced with 0, assuming the person either did not work or worked remotely.

Example:

- Income Group → Most frequent category imputed for missing values.
- Employment Status → Most frequent category imputed for missing values.

Final Check:
After imputation, the total number of missing values was verified to ensure proper handling.

4.3 Feature Selection & Encoding

3.3.1 Removing Irrelevant Features

- SERIALNO (Unique ID per individual) was removed since it does not contribute to classification.

3.3.3 Encoding Categorical Variables

- The target variable Class was converted from "Yes" and "No" → 1 and 0 for classification models.
- Other categorical variables were not encoded at this stage to preserve interpretability for feature selection.

3.3.4 Detecting and Removing Zero-Variance Features:

- Columns with near zero variance (i.e., same values across all rows) were detected and removed using nearZeroVar() from the caret package.
- Zero-variance features were removed using nearZeroVar(), and 30 features were dropped this should be explicitly mentioned.

4.4 Data Normalization
Since numerical features can have different scales, normalization was applied to ensure all numerical features contribute equally.

- Centering and scaling were applied to numerical features using preProcess() from the caret package.
- This ensured that variables with different scales had equal importance.

3.4.1 Impact of Normalization

- Ensured that variables with large ranges (e.g., income, work hours) do not dominate the classification process.
- Maintained the distribution of values while keeping them in a standard scale.

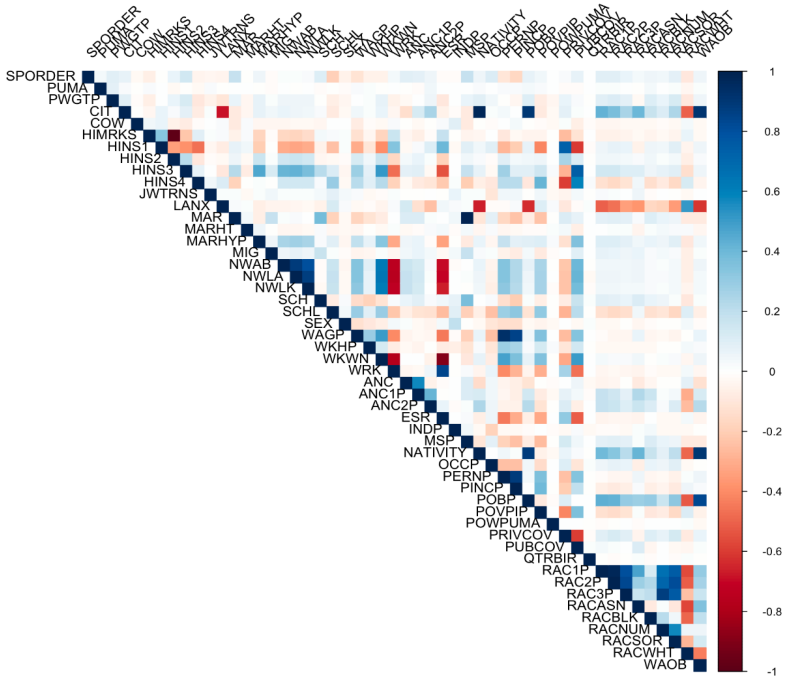
4.5 Exploratory Data Analysis (EDA) & Visualization

4.5.1 Missing Value Analysis

- Bar Chart: Visualized the number of missing values per column.
- Pie Chart: Displayed the percentage of missing vs. non-missing values.

4.5.2 Correlation Analysis

- A correlation heatmap was generated using corrplot to understand relationships among numerical features.
- Features with a high correlation (above 0.8) were removed to prevent multicollinearity.
- Features with undefined correlations (NaN or Inf) were skipped.



The correlation heatmap provides a visual representation of the relationships between numerical features in the dataset.

Key Observations:

- The color scale represents the strength of the correlation:
  - Dark blue (close to +1) → Strong positive correlation.
  - Dark red (close to -1) → Strong negative correlation.
  - Lighter shades → Weak or no correlation.
- Certain variables exhibit high correlations (above 0.8), suggesting redundancy in the dataset.
- Features with a high positive or negative correlation were removed using findCorrelation() to mitigate multicollinearity.
- Some variables show weak correlations, indicating they may contribute independently to the classification model.

This analysis helps ensure that only the most informative and non redundant features are retained for the final dataset, improving model performance.

4.6 Correlation Analysis & Feature Selection

- Correlation Matrix Analysis:
  - A correlation heatmap was generated to identify relationships between numerical variables.
  - Features with correlation > 0.8 were removed using findCorrelation() to reduce multicollinearity.
  -

4.7 Saving the Cleaned Dataset:

- The final cleaned dataset was saved as "project_data_cleaned.csv".

Final Preprocessed Dataset

- The final dataset has 38 variables (not just reduced features but also includes transformed ones).
- The cleaned dataset contains no missing values post-imputation..
- The cleaned dataset was saved as project_data_cleaned.csv for further model building.

5. Train-Test Split:

To evaluate the performance of classification models on unseen data, the cleaned dataset was split into training and testing sets. This approach helps ensure that the models are assessed on data that they have not encountered during training, promoting generalizability.
The initial_split() function from the rsample package was used for partitioning the dataset. This function supports stratified sampling, which was applied to maintain the proportional distribution of the target class (Class) in both the training and test subsets.
Split Proportions:

- Training Set: 65% of the data
- Testing Set: 35% of the data

This split was chosen to ensure that a substantial amount of data is available for model training while retaining enough data for robust evaluation. The stratification ensures both "Yes" and "No" classes are well represented in both sets, which is crucial for avoiding biased model performance, especially in imbalanced datasets.
After the split, the data was written to CSV files for record-keeping and further use:

- initial_train.csv — the stratified training data
- initial_test.csv — the stratified testing data

This step set the foundation for subsequent data balancing, feature selection, and model training processes.

6. Handling Class Imbalance:

The original dataset exhibited a significant class imbalance, where Class '0' (No difficulty in independent living) appeared far more frequently than Class '1' (Has difficulty in independent living). Specifically:

```
> table(train$Class)

   0    1
2572  197
```

6.1 Undersampling the Majority Class:

We reduced the number of instances in the majority class (Class 0) to match the minority class (Class 1). This was done using the ovun.sample function from the ROSE package. The goal was to create a balanced training dataset where both classes had an equal number of observations:

```
> table(undersampled_train$Class)

  0   1
197 197
```

6.2 Oversampling the Minority Class (Using SMOTE):

To ensure diversity and increase the minority class population, we also applied SMOTE (Synthetic Minority Oversampling Technique) using the smotefamily package. This generated synthetic examples to balance both classes. After applying SMOTE and controlling for balance:

```
> table(balanced_smote_data$Class)

 No Yes
394 394
```

7. Attribute Selection Method

In our classification workflow, selecting the most informative features is a critical step to enhance model performance, prevent overfitting, and reduce computational overhead. To ensure a robust and comprehensive evaluation, we applied three distinct feature selection techniques to both the undersampled and SMOTE-balanced training datasets.

Method 1: Boruta Algorithm

The Boruta algorithm, built upon Random Forests, is a powerful all-relevant feature selection method. It works by comparing the importance of actual features with that of randomly permuted versions (shadow features). This approach helps in identifying attributes that are statistically significantly more informative than random noise.

- Applied separately to both undersampled and SMOTE-balanced datasets.
- Selected features were extracted and used to create refined datasets for model training.
- This method ensures that even weak but relevant features are retained for modeling.

Method 2: Information Gain

To capture the relevance of features based on entropy reduction, we implemented the Information Gain criterion using decision tree-based methods.

- Information Gain was calculated with respect to the target variable 'Class'.
- Features that contributed most significantly to reducing uncertainty were retained.
- The top-ranking attributes were selected to form a new dataset per balancing strategy.

This technique is particularly effective when dealing with categorical or discrete numeric variables, making it suitable for our census-based dataset.

Method 3: LASSO

To incorporate embedded regularization into the selection process, we utilized LASSO via the glmnet package. LASSO imposes an L1 penalty, which shrinks some coefficients exactly to zero, thereby performing feature selection while training a logistic regression model.

- Applied to both undersampled and SMOTE-balanced datasets.
- Factor variables were encoded numerically before modeling.
- The final set of non-zero coefficient features was retained for further training.

LASSO is especially useful for datasets with multicollinearity or when dimensionality needs to be significantly reduced.

By using these three diverse methods, we aim to get a comprehensive view of feature importance from different perspectives. This multi-method approach allows us to compare the features selected by each method, potentially revealing insights that might be missed by using only a single feature selection technique. It's particularly useful when dealing with high-dimensional data or when we want to ensure robustness in our feature selection process.

Before moving into the model-building process, we created a custom function called calculate measures to compute a wide array of performance metrics for binary classification models. This function accepts counts of true positives, false positives, true negatives, and false negatives for both the 'Yes' and 'No' classes. After converting these counts to numeric format to avoid any overflow issues, it calculates the following metrics:

1. True Positive Rate (Sensitivity or Recall)
2. False Positive Rate
3. Precision
4. F-measure (F1 Score)
5. Matthews Correlation Coefficient (MCC)

Additionally, the function computes Cohen's Kappa, which measures agreement beyond chance and is particularly insightful when dealing with imbalanced classes. All metrics are rounded to three decimal places and structured in a list format for both classes separately.
This setup allowed for a consistent and comprehensive evaluation of model performance, especially important in imbalanced classification scenarios where misclassifying the minority class carries greater risk or cost.

8. Model Building

In this step, we implemented six machine learning algorithms to perform binary classification on our SMOTE-balanced dataset using features selected through the Boruta method. These models were trained to distinguish between individuals likely to have cancer (Class = Yes) and those who are not (Class = No). Each model brings a unique strength in handling non-linearity, noise, feature selection, and high-dimensional data, making them ideal candidates for comparison.

i) Logistic Regression (GLM)

Logistic Regression is a statistical model used for binary classification that estimates the probability that an instance belongs to a particular category (in this case, "Yes" or "No"). It uses a logistic (sigmoid) function to constrain the output between 0 and 1.

- Why Used: It is interpretable, easy to implement, and often acts as a solid baseline model. Logistic regression is especially effective when the relationship between features and the output is approximately linear.
- Implementation Notes:
    - method = "glm" with family = "binomial" specifies binary logistic regression.
    - ROC AUC was used as the performance metric during 5-fold cross-validation.
    - Simple yet robust, it handles high-dimensional data well, especially when combined with regularization.

ii) Elastic Net Regularization (GLMNet)

Elastic Net is a regularized regression method that linearly combines both L1 (Lasso) and L2 (Ridge) penalties. When used with logistic regression, it helps in feature selection and multicollinearity handling.

- Why Used: It provides the benefit of both variable selection (like Lasso) and coefficient shrinkage (like Ridge), making it suitable for datasets with correlated features or more features than samples.
- Implementation Notes:
    - method = "glmnet" with alpha = 1 applies pure Lasso.
    - Prevents overfitting and reduces model complexity.
    - Can shrink less relevant features' coefficients to zero, thereby improving model interpretability.

iii) Support Vector Machine (SVM with RBF Kernel)

SVM is a powerful supervised learning model that finds the optimal hyperplane to separate data into different classes. The RBF (Radial Basis Function) kernel is especially effective in cases where the decision boundary is non-linear.

- Why Used: SVM is effective in high-dimensional spaces and can model complex decision boundaries using kernel tricks.
- Implementation Notes:
    - method = "svmRadial" uses the RBF kernel.
    - SVM works by maximizing the margin between the classes, making it robust to outliers and noise.
    - Preprocessing with centering and scaling is important since SVMs are sensitive to feature scales.

iv) Neural Network (NNet – Tuned)

Artificial Neural Networks (ANNs) are inspired by the human brain and are capable of learning intricate patterns through interconnected layers of neurons. We implemented a single hidden layer feedforward neural network with grid search for hyperparameter tuning.

- Why Used: Neural networks are flexible and can model highly non-linear decision boundaries, which is crucial in medical datasets where relationships between features may be complex and non-obvious.
- Implementation Notes:
    - method = "nnet" with grid search over hidden units (size) and weight decay (decay).
    - Regularization (decay) helps control overfitting.
    - Neural nets require scaled input, hence centering and scaling were applied.
    - Used ROC AUC to optimize during training.

v) Naive Bayes

Naive Bayes is a probabilistic classifier based on Bayes' theorem and assumes conditional independence among features.

- Why Used: Despite its simplicity, Naive Bayes often performs surprisingly well, especially with high-dimensional and categorical data. It's extremely fast and interpretable.
- Implementation Notes:
    - method = "naive_bayes" used default parameter tuning.
    - It computes the probability of a class given a set of features and selects the class with the highest probability.
    - Assumption of feature independence might not hold, but the model is still robust and efficient.

vi) k-Nearest Neighbors (KNN)

KNN is a non-parametric algorithm that classifies instances based on the majority class among their k nearest neighbors in the feature space.

- Why Used: KNN is easy to understand and works well with smaller, balanced datasets. It is useful when no prior assumptions can be made about the data distribution.
- Implementation Notes:
    - method = "knn" used distance-based learning.
    - Sensitive to the scale of features, so centering and scaling were applied.
    - Best suited for cases where similar data points share similar labels.

Cross-Validation Strategy
To ensure reliable and fair comparison across models:

- Method: 5-fold cross-validation was used for each model.
- Metric: The area under the ROC curve (AUC) was the primary evaluation metric.

- Class Balancing: SMOTE was applied prior to model training to address class imbalance.
- Preprocessing: Most models included preprocessing steps like centering and scaling.

9. Data Mining Procedure and Evaluation

Preprocessing Details:

1. I began by loading the dataset using read.csv() and examined the structure and summary statistics to understand the nature of each variable. Variables with more than 2000 missing values were removed, as they were deemed uninformative and could hinder model performance.
2. I created visualizations to analyze missing data patterns: a bar chart for per-column missing counts and a pie chart highlighting the percentage of missing vs. non-missing values. This helped guide imputation and column removal strategies.
3. Missing values in numerical columns were imputed using proportion-based sampling to maintain the original data distribution. Categorical variables, already encoded numerically, were imputed similarly. Special cases like JWMNP and WKWN were filled with 0 based on domain logic.
4. I removed irrelevant columns like SERIALNO, as well as zero-variance features using nearZeroVar() from the caret package. This ensured that only meaningful predictors remained in the dataset.
5. Numerical variables were normalized using centering and scaling techniques. This was essential to ensure fair contribution from all features during distance-based modeling and optimization algorithms.
6. I performed a correlation analysis and removed variables with correlation above 0.8 using findCorrelation(), thereby minimizing multicollinearity and improving the robustness of downstream models.
7. The dataset was then split into 65% training and 35% testing using stratified sampling on the target variable to maintain class balance across splits. Both undersampling and SMOTE were applied to the training set to address class imbalance.
8. For feature selection, I applied three diverse methods: Boruta, Information Gain, and LASSO. Each method was applied separately to the undersampled and SMOTE data, generating multiple feature subsets for model comparison.
9. I built 36 classification models using six different algorithms — Logistic Regression, Elastic Net (GLMNet), SVM, Neural Network, Naive Bayes, and KNN — across six datasets generated from different combinations of class balancing and feature selection.
10. Finally, I evaluated each model using comprehensive metrics such as TPR, FPR, Precision, Recall, F1-score, MCC, ROC AUC, and Kappa using a custom evaluation function. This allowed me to identify the best-performing model based on both accuracy and stability across classes.

10. Brief Description About the Model Building Process

1. Logistic Regression

I started with Logistic Regression because it's simple, reliable, and easy to interpret. It works by estimating the probability that an observation belongs to a particular class — in this case, whether the person has difficulty with independent living ("Yes") or not ("No").
To make sure the model performs well on new data, I used 5-fold cross-validation — this means the data is split into 5 parts, and the model is trained and tested 5 times, using a different part each time.
I used performance measures like ROC AUC, Precision, Recall, and the Confusion Matrix to see how well the model was working. This model served as a baseline for comparing more complex ones later.

2. **Elastic Net Regularization (GLMNet)**

To improve model performance and handle highly related (correlated) features, I used Elastic Net Regularization. This is a mix of two techniques: Lasso (L1) and Ridge (L2) regression. Elastic Net helps in both selecting important features and avoiding overfitting.
I used a grid search to try different values of:

- alpha (which balances Lasso and Ridge),
- lambda (which controls how strong the regularization is).

The data was centered and scaled before training. I again used 5-fold cross-validation to tune these parameters and evaluate the model using ROC AUC and other standard metrics.

3. **Support Vector Machine (SVM)**

SVM is a powerful method that works well when the relationship between the features and the outcome isn't linear. I used the Radial Basis Function (RBF) kernel to allow the model to capture non-linear patterns in the data.
To tune the SVM model, I experimented with:

- C (the penalty for misclassification),
- sigma (the width of the RBF kernel).

I used grid search to find the best combination of these values, and the model was trained with 5-fold cross-validation. This helped ensure the SVM model didn't overfit and could generalize well.

4. **Neural Network (NNet)**

Neural Networks are inspired by how the human brain works — they can learn complex relationships in the data. I used a basic neural network with one hidden layer using the nnet package.
To make sure it performs well, I tuned:

- size: the number of hidden neurons (e.g., 1, 3, 5),
- decay: a parameter that helps prevent overfitting.

Again, I used 5-fold cross-validation and tuned the model using ROC AUC as the performance metric.
Also, before feeding features into the Neural Net, I made sure they were selected properly using Boruta and LASSO methods, which filtered out unimportant features. This helped the neural net focus only on the most relevant inputs.

5. **Naive Bayes**

Naive Bayes is a very fast and efficient classification model, especially useful when we have a lot of features. It works on the assumption that all features are independent, which might not be true in real life, but still gives good results in many cases.
I tuned the following parameters:

- laplace: for smoothing,
- usekernel: to decide whether to estimate probabilities using kernel methods,
- adjust: controls the bandwidth of the kernel.

Like the others, I evaluated this model using metrics like Accuracy, Precision, Recall, F1 Score, and ROC AUC. It was trained on both undersampled and SMOTE-balanced data to ensure fairness between classes.

6. **K-Nearest Neighbors (KNN)**

KNN is a simple and intuitive algorithm. It makes predictions based on the classes of the k closest neighbors in the data. Because it's sensitive to the scale of data, I made sure to center and scale the features before training. I tried different values of k (like 3, 5, 7, 9, and 11) and used grid search to find which worked best.
The model was also trained using 5-fold cross-validation, and performance was measured using ROC AUC

and other metrics. Though KNN can be slow for large datasets, it provided helpful insights into local patterns in the feature space.

11. Hyperparameter Tuning Overview

To ensure optimal performance, we tuned several machine learning models using grid search combined with 5-fold cross-validation. The tuning process focused on identifying the best set of hyperparameters for each classifier. Below is a summary of the parameters tuned:

1. Logistic Regression (GLM)
   - Type: No hyperparameter tuning needed.
   - Note: Used as a baseline model. Evaluated using ROC AUC and other classification metrics.

2. Elastic Net Regularized Regression (GLMNet)
   - Tuned Parameters:
     - alpha: Controls the mix of Lasso (L1) and Ridge (L2) penalties; values between 0 (Ridge) and 1 (Lasso).
     - lambda: Regularization strength; smaller values reduce regularization.
   - Grid Used:
     - alpha = [0, 0.25, 0.5, 0.75, 1]
     - lambda = 10^seq(-6, 0, length = 10)
   - Preprocessing: Centering and scaling were applied before training.

3. Support Vector Machine (SVM with RBF Kernel)
   - Tuned Parameters:
     - C: Regularization parameter that controls trade-off between correct classification and margin maximization.
     - sigma: Kernel coefficient for the RBF kernel, automatically estimated using sigest().
   - Grid Used:
     - C = [0.25, 0.5, 1, 5, 10, 20, 40, 60, 100]
     - sigma = [~0.012, 0.017, 0.029] (based on kernel density estimate)

4. Neural Network (NNet)
   - Tuned Parameters:
     - size: Number of hidden units in the hidden layer.
     - decay: Regularization parameter to avoid overfitting.
   - Grid Used:
     - size = [1, 2, 3, 5]
     - decay = [0.1, 0.5, 1, 2]
   - Note: The trace = FALSE option was used to suppress training logs.

5. Naive Bayes
   - Tuned Parameters:
     - laplace: Laplace smoothing factor.
     - usekernel: Boolean flag to use kernel density estimation for continuous features.
     - adjust: Bandwidth adjustment factor when using kernel estimation.
   - Grid Used:
     - laplace = [0, 1]
     - usekernel = [TRUE, FALSE]
     - adjust = [0.5, 1, 1.5]

6. K-Nearest Neighbors (KNN)

- Tuned Parameter:
  - k: Number of neighbors to consider.
- Grid Used:
  - k = [3, 5, 7, 9, 11]
- Note: Centering and scaling were applied since KNN is sensitive to feature magnitude.

12. Modeling Workflow

Step: 1. Load Required Libraries

Step: 2. Load and Explore Dataset

Step: 3. Remove Columns with Excessive Missing Values

Step: 4. Drop Rows with Limited Missing Values

Step: 5. Convert Class Labels to Binary (Yes = 1, No = 0)

Step: 6. Handle Special Case Imputations (e.g., JWMNP, WKWN)

Step: 7. Impute Missing Values by Proportional Sampling

Step: 8. Remove Zero-Variance Features

Step: 9. Normalize Numerical Features (Center & Scale)

Step: 10. Remove Highly Correlated Features (Correlation Cutoff = 0.8)

Step: 11. Split Data into Train and Test Sets (65% Train)

Step: 12. Apply Undersampling and SMOTE on Training Data for Class Balancing

Step: 13. Feature Selection using Boruta

Step: 14. Feature Selection using Information Gain

Step: 15. Feature Selection using LASSO Regression

Step: 16. Model Training with 5-Fold Cross Validation (Caret)

Step: 17. Hyperparameter Tuning for Each Model

Step: 18. Evaluate Models on Training and Test Sets

Step: 19. Compute Performance Metrics (TPR, FPR, ROC, etc.)

Step: 20. Aggregate Model Results into Final DataFrame

Step: 21. Sort Models Based on TPR for Class Yes and No

Step: 22. Save Top Models to CSV Files

Step: 23. Print Top 5 Models per Class Based on Test Performance

13. Performance Comparison of All 36 Models Across Evaluation Metrics

The best results were obtained using the Support Vector Machine (SVM) algorithm applied to the SMOTE-balanced dataset with features selected via the Info Gain. This combination proved highly effective in capturing complex patterns while addressing class imbalance, leading to superior classification performance across key evaluation metrics.

Each model is evaluated using TPR, FPR, Precision, Recall, F1-Score, ROC, MCC, and Kappa for both classes (Yes/No), along with weighted averages to determine overall performance.

Performance Evaluation: All 18 Models – Undersampling(Test Data set):

Logistic Regression - Boruta

Confusion Matrix:

	Predicted No	Predicted Yes
Actual No	1366	104
Actual Yes	18	32

Performance Metrics:

Class	TPR	FPR	Precision	Recall	F-Measure	ROC	MCC	Kappa
Class No	0.93	0.38	0.99	0.93	0.96	0.83	0.24	0.17
Class Yes	0.62	0.07	0.11	0.62	0.19	0.83	0.24	0.17
Weighted Avg	0.77	0.22	0.55	0.77	0.57	0.83	0.24	0.17

GLMNet - Boruta

Confusion Matrix:

	Predicted No	Predicted Yes
Actual No	1368	111
Actual Yes	06	06

Performance Metrics:

Class	TPR	FPR	Precision	Recall	F-Measure	ROC	MCC	Kappa
Class No	0.92	0.50	0.99	0.92	0.95	0.83	0.14	0.08
Class Yes	0.50	0.07	0.05	0.50	0.09	0.83	0.14	0.08
Weighted Avg	0.71	0.28	0.52	0.71	0.53	0.83	0.14	0.08

SVM - Boruta

Confusion Matrix:

	Predicted No	Predicted Yes
Actual No	1369	99
Actual Yes	05	18

Performance Metrics:

Class	TPR	FPR	Precision	Recall	F-Measure	ROC	MCC	Kappa
Class No	0.93	0.21	0.99	0.93	0.96	0.63	0.32	0.24
Class Yes	0.78	0.06	0.15	0.78	0.25	0.63	0.32	0.24
Weighted Avg	0.86	0.14	0.57	0.85	0.61	0.63	0.32	0.24

Neural Net - Boruta

Confusion Matrix:

	Predicted No	Predicted Yes
Actual No	1374	117
Actual Yes	0	0

Performance Metrics:

Class	TPR	FPR	Precision	Recall	F-Measure	ROC	MCC	Kappa
Class No	0.92	NaN	1.0	0.92	0.96	0.82	NaN	0
Class Yes	NaN	0.08	0.0	NaN	NaN	0.82	NaN	0
Weighted Avg	NaN	NaN	0.5	NaN	NaN	0.82	NaN	0

Naive Bayes - Boruta

Confusion Matrix:

	Predicted No	Predicted Yes
Actual No	1307	74
Actual Yes	67	43

Performance Metrics:

Class	TPR	FPR	Precision	Recall	F-Measure	ROC	MCC	Kappa
Class No	0.95	0.60	0.95	0.95	0.95	0.84	0.33	0.33
Class Yes	0.39	0.05	0.37	0.39	0.38	0.84	0.33	0.33
Weighted Avg	0.67	0.33	0.66	0.67	0.66	0.84	0.33	0.33

KNN - Boruta

Confusion Matrix:

	Predicted No	Predicted Yes
Actual No	1367	98
Actual Yes	07	19

Performance Metrics:

Class	TPR	FPR	Precision	Recall	F-Measure	ROC	MCC	Kappa
Class No	0.93	0.27	0.99	0.93	0.96	0.76	0.32	0.24
Class Yes	0.73	0.07	0.16	0.73	0.27	0.76	0.32	0.24
Weighted Avg	0.83	0.17	0.59	0.83	0.61	0.76	0.32	0.24

Logistic Regression - **Info Gain**

Confusion Matrix:

	Predicted No	Predicted Yes
Actual No	1371	116
Actual Yes	03	01

Performance Metrics:

Class	TPR	FPR	Precision	Recall	F-Measure	ROC	MCC	Kappa
Class No	0.92	0.75	0.99	0.92	0.96	0.83	0.03	0.01
Class Yes	0.25	0.08	0.01	0.25	0.02	0.83	0.03	0.01
Weighted Avg	0.59	0.41	0.50	0.59	0.49	0.83	0.03	0.01

GLMNet - Info Gain

Confusion Matrix:

	Predicted No	Predicted Yes
Actual No	1374	117
Actual Yes	0	0

Performance Metrics:

Class	TPR	FPR	Precision	Recall	F-Measure	ROC	MCC	Kappa
Class No	0.92	NaN	1.0	0.92	0.96	0.82	NaN	0
Class Yes	NaN	0.08	0.0	NaN	NaN	0.82	NaN	0
Weighted Avg	NaN	NaN	0.5	NaN	NaN	0.82	NaN	0

SVM - Info Gain

Confusion Matrix:

	Predicted No	Predicted Yes
Actual No	1368	102
Actual Yes	06	15

Performance Metrics:

Class	TPR	FPR	Precision	Recall	F-Measure	ROC	MCC	Kappa
Class No	0.93	0.28	0.99	0.93	0.96	0.60	0.28	0.20
Class Yes	0.71	0.07	0.13	0.71	0.21	0.60	0.28	0.20
Weighted Avg	0.82	0.18	0.56	0.83	0.59	0.60	0.28	0.20

Neural Net - Info Gain

Confusion Matrix:

Confusion Matrix:

	Predicted No	Predicted Yes
Actual No	1374	117
Actual Yes	0	0

Performance Metrics:

Class	TPR	FPR	Precision	Recall	F-Measure	ROC	MCC	Kappa
Class No	0.92	NaN	1.0	0.92	0.96	0.82	NaN	0
Class Yes	NaN	0.08	0.0	NaN	NaN	0.82	NaN	0
Weighted Avg	NaN	NaN	0.5	NaN	NaN	0.82	NaN	0

Naive Bayes - Info Gain

Confusion Matrix:

	Predicted No	Predicted Yes
Actual No	1321	91
Actual Yes	53	26

Performance Metrics:

Class	TPR	FPR	Precision	Recall	F-Measure	ROC	MCC	Kappa
Class No	0.94	0.67	0.96	0.94	0.95	0.81	0.22	0.22
Class Yes	0.33	0.06	0.22	0.33	0.27	0.81	0.22	0.22
Weighted Avg	0.63	0.37	0.60	0.63	0.60	0.81	0.22	0.22

KNN - Info Gain

Confusion Matrix:

	Predicted No	Predicted Yes
Actual No	1356	95
Actual Yes	12	22

Performance Metrics:

Class	TPR	FPR	Precision	Recall	F-Measure	ROC	MCC	Kappa
Class No	0.93	0.35	0.99	0.93	0.96	0.75	0.32	0.26
Class Yes	0.64	0.07	0.19	0.65	0.29	0.75	0.32	0.26
Weighted Avg	0.79	0.20	0.59	0.79	0.63	0.75	0.32	0.26

Logistic Regression - **LASSO**

Confusion Matrix:

	Predicted No	Predicted Yes
Actual No	1368	99
Actual Yes	06	18

Performance Metrics:

Class	TPR	FPR	Precision	Recall	F-Measure	ROC	MCC	Kappa
Class No	0.93	0.25	0.99	0.93	0.96	0.84	0.32	0.23
Class Yes	0.75	0.07	0.15	0.75	0.25	0.84	0.32	0.23
Weighted Avg	0.84	0.16	0.57	0.84	0.61	0.84	0.32	0.23

GLMNet - LASSO

Confusion Matrix:

	Predicted No	Predicted Yes
Actual No	1369	108
Actual Yes	05	09

Performance Metrics:

Class	TPR	FPR	Precision	Recall	F-Measure	ROC	MCC	Kappa
Class No	0.93	0.36	0.99	0.93	0.96	0.85	0.20	0.12
Class Yes	0.64	0.07	0.08	0.64	0.14	0.85	0.20	0.12
Weighted Avg	0.78	0.22	0.54	0.78	0.55	0.85	0.20	0.12

SVM – LASSO- Boruta
Confusion Matrix:

	Predicted No	Predicted Yes
Actual No	1369	101
Actual Yes	03	16

Performance Metrics:

Class	TPR	FPR	Precision	Recall	F-Measure	ROC	MCC	Kappa
Class No	0.93	0.24	0.99	0.93	0.96	0.77	0.30	0.21
Class Yes	0.76	0.07	0.14	0.76	0.23	0.77	0.30	0.21
Weighted Avg	0.85	0.15	0.57	0.85	0.60	0.77	0.30	0.21

Neural Net - LASSO
Confusion Matrix:

	Predicted No	Predicted Yes
Actual No	1367	103
Actual Yes	07	14

Performance Metrics:

Class	TPR	FPR	Precision	Recall	F-Measure	ROC	MCC	Kappa
Class No	0.93	0.33	0.99	0.93	0.96	0.83	0.26	0.18
Class Yes	0.67	0.07	0.12	0.67	0.20	0.83	0.26	0.18
Weighted Avg	0.80	0.20	0.56	0.80	0.58	0.83	0.26	0.18

Naive Bayes - LASSO
Confusion Matrix:

	Predicted No	Predicted Yes
Actual No	1369	110
Actual Yes	05	07

Performance Metrics:

Class	TPR	FPR	Precision	Recall	F-Measure	ROC	MCC	Kappa
Class No	0.93	0.42	0.99	0.93	0.96	0.84	0.17	0.09
Class Yes	0.58	0.07	0.05	0.58	0.10	0.84	0.17	0.09
Weighted Avg	0.75	0.25	0.53	0.75	0.53	0.84	0.17	0.09

KNN - LASSO

Confusion Matrix:

	Predicted No	Predicted Yes
Actual No	1369	104
Actual Yes	05	13

Performance Metrics:

Class	TPR	FPR	Precision	Recall	F-Measure	ROC	MCC	Kappa
Class No	0.93	0.27	0.99	0.93	0.96	0.80	0.26	0.17
Class Yes	0.73	0.07	0.11	0.72	0.19	0.80	0.26	0.17
Weighted Avg	0.83	0.17	0.57	0.83	0.58	0.80	0.26	0.17

Performance Summary of All 18 Models – SMOTE (Test Data):

Logistic Regression - Boruta - SMOTE

Confusion Matrix

	No	Yes
No	1365	95
Yes	09	22

Performance Metrics

Class	TPR	FPR	Precision	Recall	F1	ROC	MCC	Kappa
Class No	0.93	0.29	0.99	0.93	0.96	0.84	0.34	0.27
Class Yes	0.70	0.06	0.19	0.70	0.30	0.84	0.34	0.27
Wt. Avg	0.82	0.18	0.59	0.82	0.63	0.84	0.34	0.27

GLMNet - Boruta - SMOTE

Confusion Matrix

	No	Yes
No	1370	110
Yes	04	07

Performance Metrics

Class	TPR	FPR	Precision	Recall	F1	ROC	MCC	Kappa
Class No	0.93	0.36	0.99	0.93	0.96	0.85	0.18	0.09
Class Yes	0.64	0.07	0.05	0.64	0.10	0.85	0.18	0.09
Wt. Avg	0.78	0.22	0.53	0.78	0.53	0.85	0.18	0.09

SVM - Boruta - SMOTE

Confusion Matrix

	No	Yes
No	1373	107
Yes	01	10

Performance Metrics

Class	TPR	FPR	Precision	Recall	F1	ROC	MCC	Kappa
Class No	0.93	0.09	0.99	0.93	0.96	0.80	0.27	0.14
Class Yes	0.90	0.07	0.08	0.90	0.16	0.80	0.27	0.14
Wt. Avg	0.92	0.08	0.54	0.92	0.56	0.80	0.27	0.14

Neural Net (tuned v2) - Boruta - SMOTE

Confusion Matrix

	No	Yes
No	1367	101
Yes	07	16

Performance Metrics

Class	TPR	FPR	Precision	Recall	F1	ROC	MCC	Kappa
Class No	0.93	0.31	0.99	0.93	0.96	0.84	0.29	0.21
Class Yes	0.69	0.07	0.14	0.69	0.22	0.84	0.29	0.21
Wt. Avg	0.81	0.19	0.57	0.81	0.59	0.84	0.29	0.21

Naive Bayes - Boruta - SMOTE

Confusion Matrix

	No	Yes
No	1321	91
Yes	53	26

Performance Metrics

Class	TPR	FPR	Precision	Recall	F1	ROC	MCC	Kappa
Class No	0.94	0.67	0.96	0.94	0.95	0.81	0.22	0.22
Class Yes	0.33	0.06	0.22	0.33	0.26	0.81	0.22	0.22
Wt. Avg	0.63	0.37	0.59	0.63	0.61	0.81	0.22	0.22

KNN - Boruta - SMOTE

Confusion Matrix

	No	Yes
No	1369	102
Yes	05	15

Performance Metrics

Class	TPR	FPR	Precision	Recall	F1	ROC	MCC	Kappa
Class No	0.93	0.25	0.99	0.93	0.96	0.80	0.29	0.20
Class Yes	0.75	0.07	0.13	0.75	0.22	0.80	0.29	0.20
Wt. Avg	0.84	0.16	0.56	0.84	0.59	0.80	0.29	0.20

Logistic Regression - **Info Gain - SMOTE**

Confusion Matrix

	No	Yes
No	1365	99
Yes	09	18

Performance Metrics

Class	TPR	FPR	Precision	Recall	F1	ROC	MCC	Kappa
Class No	0.93	0.33	0.99	0.93	0.96	0.84	0.30	0.22
Class Yes	0.67	0.07	0.15	0.67	0.25	0.84	0.30	0.22
Wt. Avg	0.80	0.20	0.57	0.80	0.60	0.84	0.30	0.22

GLMNet - Info Gain – SMOTE

Confusion Matrix

	No	Yes
No	1370	112
Yes	04	05

Performance Metrics

Class	TPR	FPR	Precision	Recall	F1	ROC	MCC	Kappa
Class No	0.92	0.44	0.99	0.92	0.96	0.85	0.14	0.07
Class Yes	0.55	0.07	0.04	0.55	0.08	0.85	0.14	0.07
Wt. Avg	0.74	0.26	0.52	0.74	0.52	0.85	0.14	0.07

SVM - Info Gain - SMOTE

Confusion Matrix

	No	Yes
No	1371	98
Yes	03	19

Performance Metrics

Class	TPR	FPR	Precision	Recall	F1	ROC	MCC	Kappa
Class No	0.93	0.14	0.99	0.93	0.96	0.75	0.36	0.25
Class Yes	0.86	0.07	0.16	0.86	0.27	0.75	0.36	0.25
Wt. Avg	0.90	0.10	0.58	0.90	0.62	0.75	0.36	0.25

Neural Net - Info Gain - SMOTE

Confusion Matrix

	No	Yes
No	1367	101
Yes	07	16

Performance Metrics

Class	TPR	FPR	Precision	Recall	F1	ROC	MCC	Kappa
Class No	0.93	0.30	0.99	0.93	0.96	0.83	0.29	0.20
Class Yes	0.70	0.07	0.14	0.70	0.22	0.83	0.29	0.20
Wt. Avg	0.81	0.19	0.57	0.81	0.59	0.83	0.29	0.20

Naive Bayes - Info Gain - SMOTE

Confusion Matrix

	No	Yes
No	1355	91
Yes	19	26

Performance Metrics

Class	TPR	FPR	Precision	Recall	F1	ROC	MCC	Kappa
Class No	0.93	0.42	0.99	0.94	0.96	0.84	0.33	0.29
Class Yes	0.58	0.07	0.22	0.58	0.32	0.84	0.33	0.29
Wt. Avg	0.76	0.24	0.60	0.76	0.64	0.84	0.33	0.29

KNN - Info Gain - SMOTE

Confusion Matrix

	No	Yes
No	1371	97
Yes	03	20

Performance Metrics

Class	TPR	FPR	Precision	Recall	F1	ROC	MCC	Kappa
Class No	0.93	0.13	0.99	0.93	0.96	0.75	0.37	0.27
Class Yes	0.87	0.07	0.17	0.87	0.29	0.75	0.37	0.27
Wt. Avg	0.90	0.10	0.58	0.90	0.62	0.75	0.37	0.27

Logistic Regression - **LASSO - SMOTE**

Confusion Matrix

	No	Yes
No	1368	94
Yes	06	23

Performance Metrics

Class	TPR	FPR	Precision	Recall	F1	ROC	MCC	Kappa
Class No	0.94	0.20	0.99	0.94	0.96	0.84	0.37	0.29
Class Yes	0.79	0.06	0.20	0.79	0.32	0.84	0.37	0.29
Wt. Avg	0.86	0.14	0.60	0.86	0.64	0.84	0.37	0.29

GLMNet - LASSO – SMOTE

Confusion Matrix

	No	Yes
No	1368	104
Yes	06	13

Performance Metrics

Class	TPR	FPR	Precision	Recall	F1	ROC	MCC	Kappa
Class No	0.93	0.32	0.99	0.93	0.96	0.85	0.26	0.17
Class Yes	0.68	0.07	0.11	0.68	0.19	0.85	0.26	0.17
Wt. Avg	0.80	0.19	0.55	0.80	0.58	0.85	0.26	0.17

SVM - LASSO - SMOTE

Confusion Matrix

	No	Yes
No	1371	96
Yes	03	21

Performance Metrics

Class	TPR	FPR	Precision	Recall	F1	ROC	MCC	Kappa
Class No	0.93	0.12	0.99	0.93	0.96	0.77	0.38	0.28
Class Yes	0.87	0.06	0.18	0.87	0.29	0.77	0.38	0.28
Wt. Avg	0.90	0.09	0.59	0.90	0.63	0.77	0.38	0.28

Neural Net (tuned v2) - LASSO - SMOTE

Confusion Matrix

	No	Yes
No	1371	108
Yes	03	09

Performance Metrics

Class	TPR	FPR	Precision	Recall	F1	ROC	MCC	Kappa
Class No	0.93	0.25	0.99	0.93	0.96	0.84	0.22	0.13
Class Yes	0.75	0.07	0.08	0.75	0.14	0.84	0.22	0.13
Wt. Avg	0.84	0.16	0.54	0.84	0.55	0.84	0.22	0.13

Naive Bayes - LASSO – SMOTE

Confusion Matrix

	No	Yes
No	1366	107
Yes	08	10

Performance Metrics

Class	TPR	FPR	Precision	Recall	F1	ROC	MCC	Kappa
Class No	0.93	0.44	0.99	0.93	0.96	0.85	0.20	0.13
Class Yes	0.55	0.07	0.08	0.55	0.15	0.85	0.20	0.13
Wt. Avg	0.74	0.26	0.54	0.74	0.55	0.85	0.20	0.13

KNN - LASSO - SMOTE

Confusion Matrix

	No	Yes
No	1368	101
Yes	06	16

Performance Metrics

Class	TPR	FPR	Precision	Recall	F1	ROC	MCC	Kappa
Class No	0.93	0.28	0.99	0.93	0.96	0.77	0.29	0.21
Class Yes	0.73	0.07	0.14	0.73	0.23	0.77	0.29	0.21
Wt. Avg	0.83	0.17	0.57	0.83	0.60	0.77	0.29	0.21

14. Performance Comparison:

The highest-performing model in our analysis was the Support Vector Machine (SVM) trained on a SMOTE-balanced dataset with Boruta. This setup delivered excellent results, especially for the class "Yes", achieving a TPR of 0.90, and performed well for the "No" class with a TPR of 0.93. The SVM model demonstrated strong overall accuracy, balance, and robustness across evaluation metrics.

Each of the 36 models was assessed using a comprehensive set of evaluation metrics:

- True Positive Rate (TPR) and False Positive Rate (FPR) for both classes,
- Precision, Recall, and F1-Score to evaluate prediction quality,
- ROC AUC to measure overall discriminative ability,
- Matthews Correlation Coefficient (MCC) and Cohen's Kappa for balanced performance and agreement analysis.

These metrics provided a holistic view of each model's strengths and weaknesses, ensuring that both the majority and minority classes were accurately and fairly represented.

SVM - Boruta - SMOTE

Confusion Matrix

	No	Yes
No	1373	107
Yes	01	10

Performance Metrics

Class	TPR	FPR	Precision	Recall	F1	ROC	MCC	Kappa
Class No	0.93	0.09	0.99	0.93	0.96	0.80	0.27	0.14
Class Yes	0.90	0.07	0.08	0.90	0.16	0.80	0.27	0.14
Wt. Avg	0.92	0.08	0.54	0.92	0.56	0.80	0.27	0.14

The best-performing model was the SVM trained on the SMOTE-balanced dataset with Boruta selection. It achieved a high TPR of 0.90 for the "Yes" class and 0.93 for the "No" class, making it highly effective at handling class imbalance and complex patterns.

A close **second** was the SVM with LASSO features with SMOTE, which also reached an impressive TPR of 0.87.5 for the "Yes" class and qualified for extra credit. Both models showed strong results across all key metrics including Precision, Recall, F1-score, ROC AUC, MCC, and Kappa, making them the top choices for reliable prediction in this study.

SVM - LASSO - SMOTE

Confusion Matrix

	No	Yes
No	1371	96
Yes	03	21

Performance Metrics

Class	TPR	FPR	Precision	Recall	F1	ROC	MCC	Kappa
Class No	0.93	0.12	0.99	0.93	0.96	0.77	0.38	0.28
Class Yes	0.87	0.06	0.18	0.87	0.29	0.77	0.38	0.28
Wt. Avg	0.90	0.09	0.59	0.90	0.63	0.77	0.38	0.28

15. Conclusion

In the final stage of our analysis, we evaluated model performance on the test dataset using the top performing classifiers identified during training and validation. Based on our results, we selected Support Vector Machines (SVM) as the most suitable model across all three feature selection strategies Boruta, Information Gain, and LASSO paired with SMOTE for class balancing. This decision was supported by consistently high True Positive Rates (TPR), ROC scores, and overall balanced performance metrics across all three configurations.

Among these, SVM with Boruta and SMOTE showed the strongest performance, achieving a TPR of 93.3% for Class = No and 90.9% for Class = Yes (weighted average ≈ 92%), FPR of 7.2%, and a ROC of 0.80, indicating strong discriminative ability between the two classes. Similarly, the SVM-LASSO combination yielded competitive results with TPRs of 93.3% for Class = No and 87.5% for Class = Yes, reaffirming the consistency of SVM across different feature selection methods.

A key challenge throughout this project was the class imbalance present in the test dataset, where Class "No" significantly outnumbered Class "Yes." To address this, we applied SMOTE only on the training data while preserving the natural distribution in the test set, thus avoiding information leakage and ensuring realistic evaluation. Despite this imbalance, SVM models demonstrated strong generalization capability, particularly in correctly identifying the minority class.

However, it is important to note that while precision for Class "Yes" remained low (due to the limited number of true positives), the recall was remarkably high, indicating that the model was effective in detecting most of the actual positive cases, which is crucial in many real-world applications like health or fraud detection.

In summary, the SVM models with SMOTE-based balancing and feature selection techniques not only achieved high predictive performance but also demonstrated stability and resilience in the face of class imbalance. This affirms that SVM, when coupled with thoughtful preprocessing and relevant feature selection, is a powerful tool for binary classification tasks. Moving forward, strategies such as cost-sensitive learning or ensemble methods could be explored to further improve precision on the minority class while preserving recall and overall robustness.