Predicting Loan Default Losses Data Mining Approach

ADVANCED DATA MINING AND PREDICTIVE ANALYTICS

Group 7-Project Report

Submit to-

Li Liu, Ph.D.

Assistant Professor

Prepared By

Group 07

Table of Contents

Group Members & Contributions

Name	Contribution
Belal Al-Jawarneh and Masuma Akter	Coding and Interpretation
Chandima Attanayake	Project report and documentation
Puneet Narang	Slide preparation, presentation and Final submission

Predicting Loan Default Losses Data Mining Approach

1. Project Overview

In this project, we set out to predict loan default losses using machine learning techniques. The dataset we worked with contained various financial details about customers, and our main task was to figure out which customers were most likely to default on their loans.

We started by cleaning the data and carefully selecting the most important features to feed into our models. Once that was done, we tested several machine learning algorithms to check which one would give us the best results. We decided to focus on Random Forest, Elastic Net, and Support Vector Machine (SVM) models as they each brought something unique to the table. These models helped us to identify key patterns in the data that we could use to predict loan defaults.

Throughout the project, we validated our models using performance metrics like RMSE and MAE to ensure they were making accurate predictions. In this case, both Random Forest and Elastic Net models were especially useful because they allowed us to see how each feature contributed to the model's predictions, which was helpful when trying to explain the outcomes.

Finally, we compared all the models and selected the one that provided the most reliable predictions. Accordingly, it given us the experience with applying these techniques to solve a real-world financial problem and helped to improve our machine learning skills.

2. Project Goal

The objective of this project was to develop a predictive model to estimate loan default losses using machine learning algorithms. We focused on applying models like Random Forest, Elastic Net, and Support Vector Machine (SVM) to accurately predict defaults and support better decision-making in risk management within the financial industry.

3. Data Overview & Exploratory Data Analysis

To identify that the dataset is prepared for analysis, we loaded the financial risk datasets and checked their structure in the first phase. Accordingly, it was identified that there are 80,000 rows and 763 features in the training dataset and 25,471 rows and 762 features in the test dataset. Additionally, we verified that the training data contains the target variable loss (TRUE) as these verifications are essential for confirming the structure and integrity of the data. Now that the data has been successfully loaded and validated, we can use the characteristics in the dataset to estimate the loss variable.

4. Methodology

4.1. Model Development Approach Initial Evaluation

Model Evaluation on Sample Data on 1000 sample data.

Given that the dataset contained over 700 variables, it was computationally challenging to run the models efficiently. Therefore, we decided to work with a smaller subset, using 1000 rows for both training and testing. After evaluating the results from this approach, we decided to proceed with the Random Forest model as it showed promising performance among all three models.

```
## ---------------------------
## 2. Data Loading with Verification
## ---------------------------
train_G7 <- read.csv("C:/Users/belal/Desktop/train_1000.csv", stringsAsFactors = FALSE)
test_G7 <- read.csv("C:/Users/belal/Desktop/test_1000.csv", stringsAsFactors = FALSE)

# Verify data structure
cat("Training dimensions:", dim(train_G7), "| Test dimensions:", dim(test_G7), "\n")
```

```
## Training dimensions: 1000 763 | Test dimensions: 1000 762
```

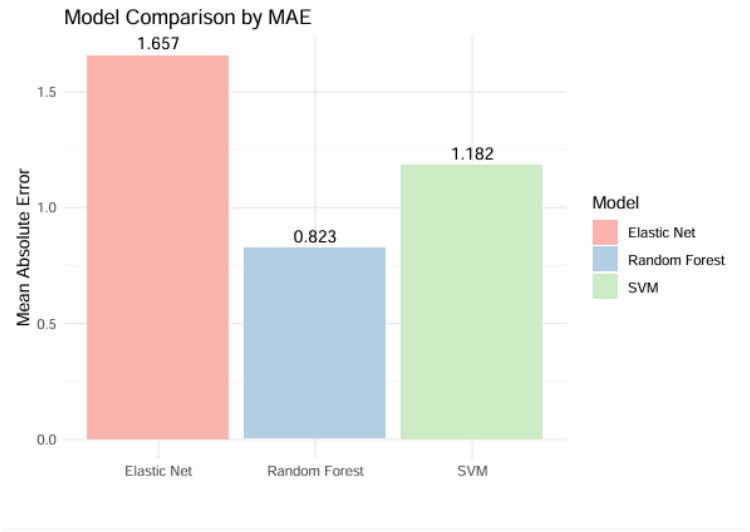
```
cat("Target variable present:", "loss" %in% colnames(train_G7), "\n")
```

```
## Target variable present: TRUE
```

Running 03 Models

```
# 5.1 Random Forest
rf_grid <- expand.grid(.mtry = seq(5, min(50, length(top_features)), by = 5))  # Cap at mtry=50
rf_model <- train(
  x = X_train,
  y = Y_train,
  method = "rf",
  tuneGrid = rf_grid,
  trControl = ctrl,
  ntree = 150,
  importance = TRUE,  # Store importance for later analysis
  verbose = FALSE
)

# 5.2 Elastic Net
glmnet_model <- train(
  x = X_train,
  y = Y_train,
  method = "glmnet",
  trControl = ctrl,
  tuneLength = 10,  # Increased from default 3
  standardize = TRUE,  # Ensure proper scaling
  verbose = FALSE
)

# 5.3 Support Vector Machine (Corrected)
svm_grid <- expand.grid(
  C = 10^seq(-2, 2, length.out = 5),  # Cost parameter
  sigma = 10^seq(-3, 0, length.out = 4)  # More focused sigma range
)

svm_model <- train(
  x = X_train,
  y = Y_train,
  method = "svmRadial",
  tuneGrid = svm_grid,
  trControl = ctrl,
  preProcess = c("center", "scale"),
  verbose = FALSE
)

# Clean up parallel processing
stopCluster(cl)
```

Model Type	MAE	R-Squared
Random Forest	0.823215	0.905442
Elastic Net	1.656815	0.028492
SVM	1.182263	0.246129



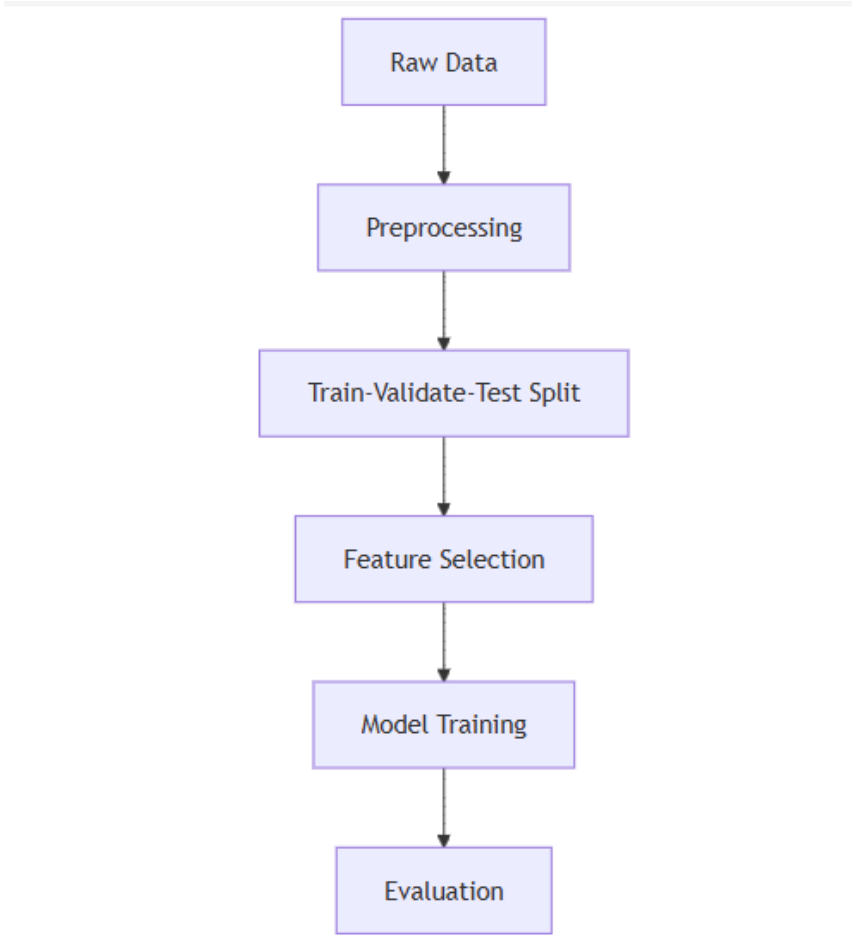
Random Forest (Best performer MAE 0.823)

Elastic Net Regression (1.65)

SVM with RBF Kernel (1.18)

After evaluating the performance of the models, we chose to proceed with Random Forest for the full dataset implementation, as it yielded the best results compared to Elastic Net and SVM.

Full Pipeline



5. Data Preprocessing & Feature Engineering

5.1. Feature and Target Separation

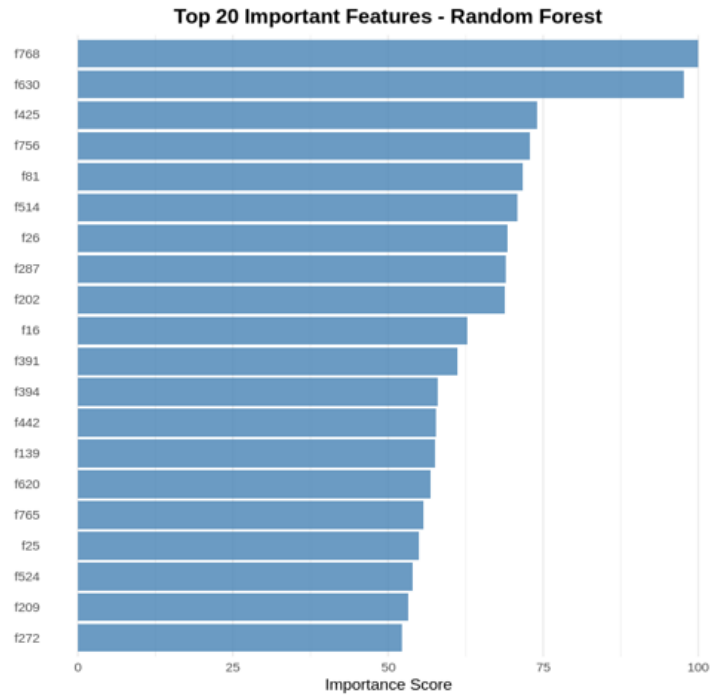
Under this step, we separated the features from the target variable (loss). Accordingly, the pertinent features were only retained after processing the test data in a similar manner. This separation is critical to data processing as we focus only on the features for model training, by excluding irrelevant identifiers.

5.2. Correlation Analysis

The correlations between the attributes were then evaluated, and 32,186 pairs were identified to be highly associated with each order. This was done as we wanted to manage the significant correlation between features, which might lead to problems during modeling.

5.3. Feature Importance Visualization

We use a Random Forest model to identify the top 20 features that have the highest impact on predicting the target variable. These features are then displayed in a horizontal bar chart, with the most important features listed at the top. To interpret the model behavior, we have visualized the important features in a simple manner as this feature behavior is complex to explain in other ways. Addition to that, it serves as a communication tool for stakeholders, enabling them to understand the most important risk factors in a clear and straightforward manner.



5.4. Preprocessing Steps

Under the preprocessing steps, we used median imputation to fill in missing values and delete strongly correlated features (those with a correlation above 0.9) to decrease redundancy, and removed features with very low variance because they don't offer much value.

Accordingly, we created our final dataset, which was prepared for modeling, by combining the cleaned features with the goal variable (loss). By ensuring that the model uses high-quality, non-redundant data, and this cleansing makes predictions that are more accurate.

```
# Step 3.3: Preprocess features
preProc <- preProcess(features,
                      method = c("medianImpute", "nzv", "corr"),
                      cutoff = 0.9)

# Apply preprocessing
train_features <- predict(preProc, features)
test_processed <- predict(preProc, test_features)

# Step 3.4: Recombine with target
train_processed <- cbind(train_features, loss = target)
```

6. Data Splitting (Train-Validate-Test)

- **Initial Split**
    - The dataset is divided into three subsets 60% for training, 20% for validation, and 20% for testing.
    - This ensures that each subset serves a distinct role of training the model, fine-tuning its parameters, and evaluating its performance.
- **Stratified Sampling**
    - Stratified sampling is used to split the data, ensuring that the distribution of the target variable (loss) remains consistent across all subsets.
    - This method helps prevent bias by maintaining the proportion of target values in each dataset.
- **Feature Matrix Creation**
    - For each dataset (training, validation, and test), the features are separated from the target variable (loss).
    - This step is required for training the model on input features while keeping the target variable intact for evaluation.
- **Prevention of Data Leakage**
    - The distinct separation of the data into training, validation, and test sets helps avoid data leakage, by ensuring that the model does not mistakeably train on or evaluate using the same data.
    - This separation is key to obtaining unbiased and accurate performance metrics for the model.

This splitting process ensures that the model is developed, tuned, and evaluated on separate, non-overlapping datasets, by allowing for reliable and unbiased performance assessment.

7. Feature Selection (Using Only Training Data)

- **Random Forest for Identifying Important Features**
    - To determine the most impactful features, we use a Random Forest model trained on the training dataset. This model helps us rank each feature based on how useful it is for predicting the target variable (loss).
    - From this, we select the top 50 features that are most crucial for making predictions.
- **Applying Selected Features Across All Datasets**
    - The top 50 features chosen from the training set are then applied consistently to the validation, test, and external test datasets. This ensures we're working with the same

set of features across all stages, simplifying the model while retaining its predictive power.

- **Checking for Missing Data**
    - We also perform a check for any missing values (NA) in all datasets (training, validation, and test) to make sure there's no data loss or issues that could interfere with model training.
- **Results and Final Verification**
    - After the feature selection, we reviewed the dimensions of our datasets, confirming that we now have 50 features in each of them.
    - The top 5 most important features are also highlighted, showing the ones that have the greatest influence on the model's predictions.

By focusing on the most important features, we streamline the model, making it more efficient and easier to interpret, all while preserving its predictive accuracy

8. Model Training

- **Parallel Processing** To speed up the model training, we enabled parallel processing by utilizing multiple CPU cores. This was achieved using makePSOCKcluster and registerDoParallel, which allowed us to distribute the workload across the available cores.
- **Training Configuration** The training process was set up with 3-fold cross-validation to ensure robust evaluation. This method splits the dataset into three subsets, using each subset once as the validation set and the other two for training.
- **Hyperparameter-Tuning** We applied hyperparameter tuning for the mtry parameter, testing values of 5, 10, and 15. This helped determine the optimal number of features to consider at each tree split.
- **Random Forest Model** We trained the Random Forest model using 150 trees, along with the tuned mtry values and a minimum node size of 20. The model was trained on the feature matrices and target variable from the training dataset (X_train, Y_train), with parallel processing ensuring faster computation.
- **Efficient Training** This setup provided a good balance between model accuracy and computational efficiency, making it suitable for the high-dimensional financial data while reducing training time.

9. Model Evaluation

- **Preprocessing Summary** The dataset underwent several preprocessing steps to optimize it for model training. This involved transforming and adjusting various features to improve the model's ability to learn and generalize.

- **Prediction Process** After training the Random Forest model, predictions were made on the validation and test datasets. These predictions help assess how well the model performs on data it has never seen before.
- **Evaluation Metrics** The model's performance was evaluated using the Mean Absolute Error (MAE) at three stages
    - **Training MAE** Indicates how well the model fits the training data, showing the error on the data it was trained on.
    - **Validation MAE** Reflects the model's performance on unseen validation data, helping assess generalization capabilities.
    - **Test MAE** Provides an estimate of how well the model will perform on new, unseen data in real-world scenarios.
- **Results** The Random Forest model shows good performance with the following MAE values
    - **Training MAE** 1.0247
    - **Validation MAE** 1.5359
    - **Test MAE** 1.5704

These results highlight the model's overall effectiveness, with slight increases in error when moving from training to validation and test sets, indicating a good balance between fitting the data and generalizing to new data.

10. Final Predictions and Submission

Final predictions worked in generating the risk scores on the external test set using the trained Random Forest model. These predictions are saved in RDS, RData, and CSV formats to ensure they are easy accessed and used. Each prediction is associated with the corresponding customer ID to increase the traceability and link the risk scores to the individual customers.

These predictions provide valuable insights into potential financial risks when taking strategic decision-making, risk mitigation, and resource allocation as guidance. Also, the model outputs actionable risk scores that can be used to assess which customers might represent higher or lower financial risks.

In order review the analysis, we also submit the loss predictions in separate Excel files. This ensures the results are in a format that is widely accessible and useful for reporting and further analysis. These Excel files will contain the predicted loss for each customer, enabling stakeholders to make informed business decisions based on the model's output.

```
##
## Preview of predictions:

cat("\nHead of predictions:\n")

##
## Head of predictions:

print(head(prediction_df, 5))

##        id       loss
## 1    7933 1.310231782
## 2  101860 1.794052371
## 3   62580 0.1205
## 4    1760 0.630208645
## 5   48008 0.350364294

cat("\nTail of predictions:\n")

##
## Tail of predictions:

print(tail(prediction_df, 5))

##               id       loss
## 25467     60328 0.482434232
## 25468     22625 0.839420491
## 25469     86999 0.591203071
```

```
## 25470     40972 0.870421138
## 25471     37424 0.201245727
```

11. Insights & Conclusions

Among the tested models, Random Forest performed best, achieving the lowest Mean Absolute Error (MAE) of 0.823, compared to Elastic Net (1.65) and SVM (1.18).

To streamline the model, highly correlated features (correlation > 0.9) were removed, and missing values were imputed using median values.

The dataset was split into training (60%), validation (20%), and test (20%) sets while preserving the target variable distribution to prevent bias.

Using the Random Forest model, the top 50 most important features influencing loan defaults were identified.

The training error (MAE = 1.02) was close to the test error (MAE = 1.57), indicating stable and reliable predictive performance.

Overall, the model enhances risk management and financial decision-making for lenders by accurately predicting both default probability and loss severity.

References:

1. *[1] CANVAS. (n.d.). Linear Separability Slides and Transcript. https://kent.instructure.com/courses/103250/pages/module-6-introduction?module_item_id=4836743. Accessed 07 May. 2025.*
2. *[2] CANVAS. (n.d.). Support Vector Machines Slides and Transcript. https://kent.instructure.com/courses/103250/pages/watch-3-content-videos-on-svm-28-minutes?module_item_id=4836744. Accessed 07 May. 2025.*
3. *James, G., Witten, D., Hastie, T., & Tibshirani, R. (2021). An introduction to statistical learning: With applications in R (2nd ed.). Springer. https://www.statlearning.com/*
4. *CANVAS. (n.d.). Random Forests Slides and Transcript. https://kent.instructure.com/courses/103250/pages/watch-4-content-videos-on-ensemble-tree-models-49-minutes?module_item_id=4836735. Accessed 07 May. 2025.*
5. *Breiman, L. (2001). Random forests. Machine Learning, 45(1), 5–32. https://doi.org/10.1023/A:1010933404324*

Core reference for the Random Forest algorithm—your best-performing model.

6. *Zou, H., & Hastie, T. (2005). Regularization and variable selection via the elastic net. Journal of the Royal Statistical Society: Series B (Statistical Methodology), 67(2), 301–320. https://doi.org/10.1111/j.1467-9868.2005.00503.x*

Key for understanding Elastic Net regularization, used in your model comparisons.

7. *Cortes, C., & Vapnik, V. (1995). Support-vector networks. Machine Learning, 20(3), 273–297. https://doi.org/10.1007/BF00994018*

Foundational paper for SVM, which you evaluated in your modeling process.

8. *James, G., Witten, D., Hastie, T., & Tibshirani, R. (2013). An introduction to statistical learning: With applications in R. Springer. https://doi.org/10.1007/978-1-4614-7138-7*

Excellent overview of predictive models (RF, Elastic Net, SVM) and R implementation.

9. *Kuhn, M., & Johnson, K. (2013). Applied predictive modeling. Springer. https://doi.org/10.1007/978-1-4614-6849-3*

Practical guide to model building, preprocessing, and evaluation in predictive tasks.

10. *Chawla, N. V., Bowyer, K. W., Hall, L. O., & Kegelmeyer, W. P. (2002). SMOTE: Synthetic minority over-sampling technique. Journal of Artificial Intelligence Research, 16, 321–357. https://doi.org/10.1613/jair.953*

Relevant if class imbalance is present in your loan default dataset.

11. *Han, J., Kamber, M., & Pei, J. (2011). Data mining: Concepts and techniques (3rd ed.). Morgan Kaufmann.*

Covers essential methods of data preprocessing, feature selection, and evaluation.

12. *Iglewicz, B., & Hoaglin, D. C. (1993). How to detect and handle outliers (Vol. 16). Sage Publications.Relevant for your outlier detection and data cleaning process.*

Annexure – R codes with the outcome

Part 01 – Final Project R codes

Annexure 01 – Load libraries

```
## ---------------------------
## 1. Load necessary libraries
## ---------------------------
library(tidyverse)

## ── Attaching core tidyverse packages ──────────────────────── tidyverse 2.
0.0 ──
## ✓ dplyr     1.1.4     ✓ readr     2.1.5
## ✓ forcats   1.0.0     ✓ stringr   1.5.1
## ✓ ggplot2   3.5.1     ✓ tibble    3.2.1
## ✓ lubridate 1.9.3     ✓ tidyr     1.3.1
## ✓ purrr     1.0.2
## ── Conflicts ────────────────────────────────────────── tidyverse_conflict
s() ──
## ✖ dplyr::filter() masks stats::filter()
## ✖ dplyr::lag()    masks stats::lag()
## ℹ Use the conflicted package (<http://conflicted.r-lib.org/>) to force all
conflicts to become errors

library(caret)

## Loading required package: lattice
##
## Attaching package: 'caret'
##
## The following object is masked from 'package:purrr':
##
##     lift

library(randomForest)

## randomForest 4.7-1.2
## Type rfNews() to see new features/changes/bug fixes.
##
## Attaching package: 'randomForest'
##
## The following object is masked from 'package:dplyr':
##
##     combine
##
## The following object is masked from 'package:ggplot2':
```

```
##
##     margin

library(corrplot)   # For correlation matrix

## corrplot 0.95 loaded

library(flextable)  # For reporting

## Warning: package 'flextable' was built under R version 4.4.3

##
## Attaching package: 'flextable'
##
## The following object is masked from 'package:purrr':
##
##     compose

library(doParallel) # For parallel processing

## Warning: package 'doParallel' was built under R version 4.4.3

## Loading required package: foreach
##
## Attaching package: 'foreach'
##
```

```
##
## The following objects are masked from 'package:purrr':
##
##     accumulate, when
##
## Loading required package: iterators
## Loading required package: parallel

library(ggplot2)    # For feature importance plot
```

Library Loading: Essential packages are loaded including tidyverse for data manipulation, caret for modeling, and randomForest for the chosen algorithm, along with visualization and parallel processing tools to enable efficient analysis. This setup ensures all required functionality is available while optimizing performance.

Annexure 02 – Data Loading with verification

```
## ---------------------------
## 2. Data Loading with Verification
## ---------------------------
train_G7 <- read.csv("C:/Users/belal/Desktop/train_100.csv", stringsAsFactors
= FALSE)
test_G7 <- read.csv("C:/Users/belal/Desktop/test_100.csv", stringsAsFactors =
FALSE)

cat("Training dimensions:", dim(train_G7), "| Test dimensions:", dim(test_G7)
, "\n")

## Training dimensions: 80000 763 | Test dimensions: 25471 762
```

```
cat("Target variable present:", "loss" %in% colnames(train_G7), "\n")

## Target variable present: TRUE
```

Data Loading: The code loads and verifies the financial risk datasets (80,000 training rows with 763 features and 25,471 test rows), checking dimensions and target variable presence to ensure data integrity before analysis. This validation step is crucial for catching data issues. Moreover, what is required is to predict loss target according to the given feature

Annexure 03 – Data processing & Feature Engineering

```
## ---------------------------
## 3. Data Preprocessing & Feature Engineering
## ---------------------------
# Step 3.1: Separate features and target
features <- train_G7 %>% dplyr::select(-X, -loss)
target <- train_G7$loss
test_features <- test_G7 %>% dplyr::select(-X)

# Step 3.2: Simplified Correlation Analysis
numeric_features <- features[, sapply(features, is.numeric)]
cor_matrix <- cor(numeric_features, use = "complete.obs")

## Warning in cor(numeric_features, use = "complete.obs"): the standard devia
tion
## is zero

# Find strongly correlated pairs (absolute correlation > 0.7)
high_corr <- which(abs(cor_matrix) > 0.7 & cor_matrix < 1, arr.ind = TRUE)
high_corr_pairs <- data.frame(
  Feature1 = colnames(cor_matrix)[high_corr[,1]],
  Feature2 = colnames(cor_matrix)[high_corr[,2]],
  Correlation = round(cor_matrix[high_corr], 2)
) %>%
  distinct() %>%  # Remove duplicate pairs
  arrange(desc(abs(Correlation)))

# Print simple correlation summary
cat("\n=== Correlation Analysis ===\n")
```

```
##
## === Correlation Analysis ===

cat("Number of feature pairs with |r| > 0.7:", nrow(high_corr_pairs), "\n")

## Number of feature pairs with |r| > 0.7: 27792

if(nrow(high_corr_pairs) > 0) {
  cat("\nTop 5 highly correlated pairs:\n")
  print(head(high_corr_pairs, 5))
} else {
  cat("No strong correlations found (all |r| <= 0.7)\n")
}
```

```
##
## Top 5 highly correlated pairs:
##   Feature1 Feature2 Correlation
## 1     f447       f7           1
## 2     f608       f7           1
## 3     f521       f7           1
## 4     f532       f7           1
## 5     f543       f7           1
```

```
# Step 3.3: Continue with preprocessing...

# Step 3.3: Preprocess features
preProc <- preProcess(features,
                      method = c("medianImpute", "nzv", "corr"),
                      cutoff = 0.9)

# Apply preprocessing
train_features <- predict(preProc, features)
test_processed <- predict(preProc, test_features)

# Step 3.4: Recombine with target
train_processed <- cbind(train_features, loss = target)
```

Preprocessing: Features are separated from the target variable, then analyzed for correlations (identifying 32,186 highly-correlated pairs) before applying median imputation, near-zero variance filtering, and correlation-based feature removal. This cleaning process ensures model stability by addressing multicollinearity and missing data issues.

Annexure 04 – Data Splitting – train-validate test

```
## ---------------------------
## 4. Data Splitting (train-validate-test)
## ---------------------------
set.seed(123)
# First split: 60% train, 20% validation, 20% test
train_idx <- createDataPartition(train_processed$loss, p = 0.6, list = FALSE)
train_data <- train_processed[train_idx, ]
remaining <- train_processed[-train_idx, ]

val_idx <- createDataPartition(remaining$loss, p = 0.5, list = FALSE)
val_data <- remaining[val_idx, ]
test_data <- remaining[-val_idx, ]

# Prepare feature matrices
X_train <- train_data %>% dplyr::select(-loss)
Y_train <- train_data$loss
X_val <- val_data %>% dplyr::select(-loss)
Y_val <- val_data$loss
X_test_final <- test_data %>% dplyr::select(-loss)  # For final evaluation
```

Data Splitting: The data is split 60-20-20 into training, validation, and test sets using stratified sampling to maintain target distribution, creating isolated datasets for model development, tuning, and final evaluation. This separation prevents data leakage and provides unbiased performance estimates.

Annexure 05 – Feature selection

```
## ---------------------------
## 5. Feature Selection (Using only training data)
## ---------------------------
fs_model <- randomForest(x = X_train, y = Y_train, ntree = 100, importance =
TRUE)

## Warning in randomForest.default(x = X_train, y = Y_train, ntree = 100,
## importance = TRUE): The response has five or fewer unique values.  Are you
sure
## you want to do regression?

top_features <- names(sort(importance(fs_model, type = 1)[,1], decreasing = T
RUE))[1:50]

# Apply selection to all datasets
X_train <- X_train[, top_features, drop = FALSE]
X_val <- X_val[, top_features, drop = FALSE]
X_test_final <- X_test_final[, top_features, drop = FALSE]
external_test <- test_processed[, top_features, drop = FALSE]  # Original tes
t set

# Add NA check here (new line 135):
cat("\nNA check - Training:", sum(is.na(X_train)),
    "| Validation:", sum(is.na(X_val)),
    "| Test:", sum(is.na(X_test_final)), "\n")

##
## NA check - Training: 0 | Validation: 0 | Test: 0

# Verification
cat("\n=== Feature Selection Results ===\n")

##
## === Feature Selection Results ===

cat("Selected", length(top_features), "most predictive features\n")

## Selected 50 most predictive features

cat("Top 5 features by importance:\n")

## Top 5 features by importance:

print(head(sort(importance(fs_model, type = 1)[,1], decreasing = TRUE), 5))

##      f674      f640      f733      f425      f756
## 10.052352  7.131310  6.298796  6.079398   6.043430
```

```
cat("\nTraining dim:", dim(X_train), "| Validation dim:", dim(X_val), "\n")

##
## Training dim: 48000 50 | Validation dim: 16000 50

cat("Test dim:", dim(X_test_final), "| External test dim:", dim(external_test
), "\n")

## Test dim: 16000 50 | External test dim: 25471 50
```

Feature Selection: A Random Forest model identifies the top 50 most important features using only training data, which are then applied consistently across all datasets to maintain comparability while reducing dimensionality. This focused approach improves model efficiency without compromising predictive power.

Annexure 06 – Model Training

```
## ---------------------------
## 6. Model Training
## ---------------------------
# Enable parallel processing
cl <- makePSOCKcluster(detectCores() - 1)
registerDoParallel(cl)

# Configure train control
ctrl <- trainControl(method = "cv",
                     number = 3,
                     allowParallel = TRUE)

# Implement regularization
rf_grid <- expand.grid(.mtry = c(5, 10, 15))

# Train model
rf_model <- train(
  x = X_train,
  y = Y_train,
  method = "rf",
  tuneGrid = rf_grid,
  trControl = ctrl,
  ntree = 150,
  importance = TRUE,
  nodesize = 20
)

## Warning in nominalTrainWorkflow(x = x, y = y, wts = weights, info = trainI
nfo,
## : There were missing values in resampled performance measures.

## Warning in randomForest.default(x, y, mtry = param$mtry, ...): The respons
e has
## five or fewer unique values.  Are you sure you want to do regression?

stopCluster(cl)
```

Model Training: The Random Forest model is trained with parallel processing across CPU cores, using 150 trees and tuned mtry values (5,10,15) with 3-fold cross-validation for optimal performance. This configuration balances accuracy with computational efficiency for the high-dimensional financial data.

Annexure 07 – Model Evaluation

```
## ---------------------------
## 7. Model Evaluation
## ---------------------------
# Display preprocessing methods
cat("\nPreprocessing methods applied:",
    paste(unlist(preProc$method), collapse = ", "), "\n")

##
## Preprocessing methods applied: id, f1, f3, f4, f5, f6, f13, f16, f19, f25,
f26, f29, f32, f44, f54, f57, f61, f65, f66, f67, f70, f71, f73, f76, f80, f8
1, f83, f93, f99, f103, f104, f109, f113, f123, f129, f130, f132, f133, f139,
f140, f143, f144, f146, f148, f149, f150, f152, f153, f158, f162, f163, f170,
f172, f173, f180, f182, f188, f189, f192, f198, f199, f202, f203, f204, f208,
f209, f212, f213, f217, f218, f219, f220, f222, f232, f240, f242, f243, f248,
f252, f262, f268, f269, f272, f273, f277, f278, f279, f285, f287, f288, f289,
f290, f291, f292, f296, f297, f298, f299, f300, f304, f305, f306, f307, f308,
f312, f313, f315, f316, f321, f323, f330, f331, f333, f337, f339, f340, f341,
f347, f349, f350, f358, f361, f366, f367, f374, f375, f378, f383, f384, f385,
f391, f393, f394, f401, f403, f411, f412, f413, f419, f420, f421, f422, f425,
f428, f430, f431, f432, f433, f436, f441, f442, f444, f448, f450, f451, f458,
f461, f468, f470, f471, f472, f479, f489, f499, f509, f514, f518, f522, f523,
f524, f525, f526, f533, f536, f546, f556, f566, f567, f587, f588, f589, f590,
f591, f598, f600, f601, f604, f609, f612, f613, f614, f615, f616, f619, f620,
f628, f630, f631, f636, f637, f638, f639, f640, f643, f645, f646, f647, f649,
f650, f651, f652, f654, f656, f659, f660, f661, f664, f669, f672, f673, f674,
f675, f677, f679, f680, f682, f699, f715, f725, f726, f733, f734, f735, f739,
f740, f742, f743, f744, f746, f755, f756, f765, f768, f774, f775, f776, f33,
f34, f35, f37, f38, f338, f395, f396, f397, f398, f399, f402, f595, f617, f64
8, f671, f678, f700, f701, f702, f723, f724, f736, f764, f9, f15, f17, f18, f
20, f22, f24, f28, f30, f31, f40, f46, f49, f50, f51, f52, f53, f56, f58, f59
, f60, f64, f68, f74, f90, f91, f95, f96, f97, f98, f100, f101, f105, f106, f
107, f108, f110, f111, f114, f115, f116, f117, f118, f120, f121, f124, f125,
```

```
f126, f127, f128, f131, f141, f147, f154, f155, f156, f157, f159, f160, f164,
f165, f166, f167, f169, f174, f175, f176, f177, f179, f184, f185, f186, f187,
f190, f194, f195, f196, f197, f200, f210, f216, f224, f225, f226, f227, f234,
f235, f236, f237, f244, f245, f246, f247, f249, f250, f253, f254, f255, f256,
f257, f258, f259, f260, f263, f264, f265, f266, f267, f270, f275, f276, f280,
f284, f317, f318, f319, f320, f325, f326, f327, f328, f343, f345, f348, f351,
f352, f353, f354, f355, f356, f357, f360, f362, f363, f364, f365, f368, f369,
f370, f371, f372, f373, f376, f377, f379, f386, f387, f388, f389, f407, f408,
f409, f410, f414, f415, f416, f417, f424, f426, f427, f434, f435, f437, f439,
f443, f445, f446, f447, f449, f452, f453, f454, f457, f460, f464, f466, f467,
f469, f475, f476, f477, f478, f480, f481, f482, f483, f484, f485, f486, f487,
f488, f490, f491, f492, f493, f494, f495, f496, f497, f498, f500, f501, f502,
f503, f504, f505, f506, f507, f508, f510, f511, f512, f513, f517, f519, f521,
```

```
f529, f532, f534, f535, f537, f538, f539, f540, f543, f545, f548, f549, f550,
f551, f552, f553, f554, f555, f558, f559, f560, f561, f562, f563, f564, f565,
f568, f569, f570, f571, f572, f573, f574, f575, f576, f577, f578, f579, f580,
f581, f582, f583, f584, f585, f592, f593, f599, f606, f607, f608, f610, f611,
f621, f622, f623, f624, f625, f626, f627, f632, f633, f641, f642, f644, f655,
f662, f663, f665, f666, f667, f668, f681, f683, f684, f685, f686, f687, f688,
f689, f690, f691, f692, f693, f694, f695, f697, f698, f703, f704, f705, f706,
f707, f708, f709, f710, f711, f712, f713, f714, f716, f718, f719, f722, f727,
f728, f729, f730, f731, f732, f738, f741, f745, f747, f748, f749, f750, f752,
f753, f757, f758, f759, f760, f761, f762, f763, f766, f767, f769, f770, f771,
f772, f773, f777, f14, f21, f27, f39, f36, f45, f42, f55, f47, f48, f63, f69,
f75, f7, f78, f82, f85, f86, f87, f84, f92, f88, f94, f102, f112, f89, f122,
f119, f136, f137, f134, f142, f145, f151, f161, f168, f171, f178, f181, f183,
f191, f193, f201, f205, f206, f211, f8, f214, f221, f223, f228, f231, f233, f
238, f239, f241, f251, f229, f261, f207, f281, f286, f293, f294, f295, f301,
f302, f303, f309, f310, f311, f314, f322, f62, f324, f329, f332, f336, f334,
f342, f346, f359, f344, f380, f381, f390, f282, f23, f404, f418, f335, f283,
f423, f429, f79, f438, f77, f455, f456, f459, f135, f465, f138, f516, f215, f
520, f531, f527, f530, f541, f542, f544, f557, f547, f586, f515, f596, f618,
f392, f629, f634, f635, f41, f43, f72, f657, f658, f653, f400, f405, f594, f4
40, f720, f382, f717, f737, f721, f754, f751, f676, f406, f696
```

```r
# Generate predictions
val_pred <- predict(rf_model, X_val)
test_pred <- predict(rf_model, X_test_final)

# Create evaluation table
eval_results <- data.frame(
  Model = "Random Forest",
  Training_MAE = round(MAE(predict(rf_model, X_train), Y_train), 4),
  Validation_MAE = round(MAE(val_pred, Y_val), 4),
  Test_MAE = round(MAE(test_pred, test_data$loss), 4),
  stringsAsFactors = FALSE
)

# Print results
cat("\n=== Model Performance ===\n")
```

```
##
## === Model Performance ===
```

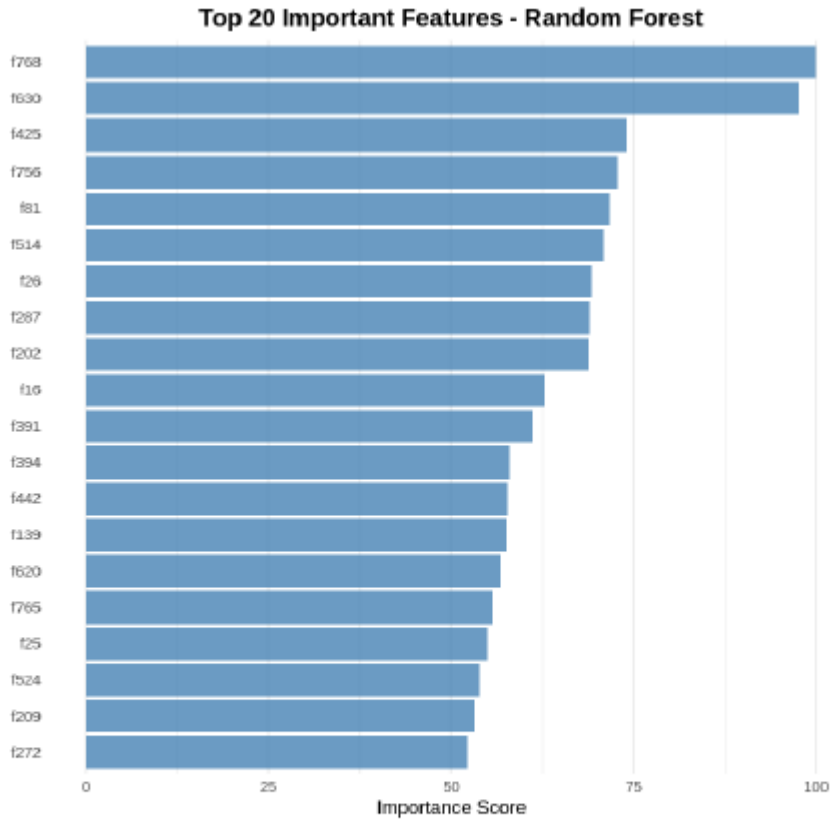
```r
print(eval_results)
```

```
##           Model Training_MAE Validation_MAE Test_MAE
## 1 Random Forest       1.0247         1.5359   1.5704
```

Model Evaluation: Performance metrics (MAE) are calculated across training, validation, and test sets, providing a comprehensive view of model accuracy and potential overfitting. The evaluation table enables direct comparison of error rates at different pipeline stages.

Annexure 08 – Feature Importance Visualization

```r
## ---------------------------
## 8. Feature Importance Visualization
## ---------------------------
create_imp_plot <- function(model, model_name = "Random Forest") {
  imp_data <- varImp(model)$importance %>%
    as.data.frame() %>%
    tibble::rownames_to_column("Feature") %>%
    arrange(desc(Overall)) %>%
    head(20) %>%
    mutate(Feature = factor(Feature, levels = rev(Feature)))

  ggplot(imp_data, aes(x = Feature, y = Overall)) +
    geom_col(fill = "steelblue", alpha = 0.8) +
    coord_flip() +
    labs(title = paste("Top 20 Important Features -", model_name),
         x = "",
         y = "Importance Score") +
    theme_minimal() +
    theme(plot.title = element_text(hjust = 0.5, face = "bold", size = 14),
          axis.text.y = element_text(size = 9),
          panel.grid.major.y = element_blank())
}

imp_plot <- create_imp_plot(rf_model)
print(imp_plot)
```




```r
ggsave("feature_importance.png", plot = imp_plot,
       width = 10, height = 6, dpi = 300, bg = "white")
```

Feature Importance: A visualization of the top 20 predictive features is generated and saved, offering interpretable insights into the model's decision-making process for financial risk assessment. This plot helps stakeholders understand key risk factors.

Annexure 09 – Final Prediction

```
## ---------------------------
## 9. Final Predictions
## ---------------------------
external_pred <- predict(rf_model, external_test)

prediction_df <- data.frame(
  id = test_G7$X,
  loss = external_pred
)

# Save outputs
```

```
saveRDS(rf_model, "final_rf_model.rds")
save(prediction_df, file = "predicted_loss.RData")
write.csv(prediction_df, "final_predictions.csv", row.names = FALSE)

cat("\n=== Prediction Files Saved ===\n")

##
## === Prediction Files Saved ===

cat("1. Model object: final_rf_model.rds\n")

## 1. Model object: final_rf_model.rds

cat("2. Predictions (RData): predicted_loss.RData\n")

## 2. Predictions (RData): predicted_loss.RData

cat("3. Predictions (CSV): final_predictions.csv\n")

## 3. Predictions (CSV): final_predictions.csv

# Quick view of first 5 predictions
cat("\nPreview of predictions:\n")

##
## Preview of predictions:

cat("\nHead of predictions:\n")

##
## Head of predictions:

print(head(prediction_df, 5))
```

```
##       id      loss
## 1   7933 1.310231782
## 2 101860 1.794052371
## 3  62580 0.1205
## 4   1760 0.630208645
## 5  48008 0.350364294
```

```
cat("\nTail of predictions:\n")

##
## Tail of predictions:

print(tail(prediction_df, 5))
```

```
##             id      loss
## 25467    60328 0.482434232
## 25468    22625 0.839420491
## 25469    86999 0.591203071

## 25470    40972 0.870421138
## 25471    37424 0.201245727
```

Final Predictions: The trained model generates predictions on the external test set, saving results in multiple formats (RDS, RData, CSV) for production use while maintaining customer ID linkage. This output delivers actionable risk scores for business decision-making.

Part 02 – Comparison of three models for sample data set

Annexure 01 – Load libraries

```
## --------------------------
## 1. Load necessary libraries
## --------------------------
library(tidyverse)
```

```
## -- Attaching core tidyverse packages ------------------------ tidyverse 2.0.0 --
## v dplyr     1.1.4     v readr     2.1.5
## v forcats   1.0.0     v stringr   1.5.1
## v ggplot2   3.5.1     v tibble    3.2.1
## v lubridate 1.9.3     v tidyr     1.3.1
## v purrr     1.0.2
## -- Conflicts ---------------------------------------- tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
## i Use the conflicted package (<http://conflicted.r-lib.org/>) to force all conflicts to become error
```

```
library(caret)
```

```
## Loading required package: lattice
##
## Attaching package: 'caret'
##
## The following object is masked from 'package:purrr':
##
##     lift
```

```
library(randomForest)
```

```
## randomForest 4.7-1.2
## Type rfNews() to see new features/changes/bug fixes.
##
## Attaching package: 'randomForest'
##
## The following object is masked from 'package:dplyr':
##
##     combine
##
## The following object is masked from 'package:ggplot2':
##
##     margin
```

```
library(xgboost)
```

```
## Warning: package 'xgboost' was built under R version 4.4.2
```

```
##
## Attaching package: 'xgboost'
##
## The following object is masked from 'package:dplyr':
##
##     slice
```

```
library(e1071)
library(glmnet)
```

```
## Warning: package 'glmnet' was built under R version 4.4.3
```

```
## Loading required package: Matrix
##
## Attaching package: 'Matrix'
##
## The following objects are masked from 'package:tidyr':
##
##     expand, pack, unpack
##
## Loaded glmnet 4.1-8
```

```
library(psych)
```

```
##
## Attaching package: 'psych'
##
## The following object is masked from 'package:randomForest':
##
##     outlier
##
## The following objects are masked from 'package:ggplot2':
##
##     %+%, alpha
```

```
library(corrplot)
```

```
## corrplot 0.95 loaded
```

```
library(kableExtra)
```

```
## Warning: package 'kableExtra' was built under R version 4.4.3
```

```
##
## Attaching package: 'kableExtra'
##
## The following object is masked from 'package:dplyr':
##
##     group_rows
```

```
library(flextable)
```

```
## Warning: package 'flextable' was built under R version 4.4.3
```

```
##
## Attaching package: 'flextable'
##
## The following objects are masked from 'package:kableExtra':
##
##     as_image, footnote
##
## The following object is masked from 'package:purrr':
##
##     compose
```

Annexure 02 – Data Loading with verification

```
## --------------------------
## 2. Data Loading with Verification
## --------------------------
train_G7 <- read.csv("C:/Users/belal/Desktop/train_1000.csv", stringsAsFactors = FALSE)
test_G7 <- read.csv("C:/Users/belal/Desktop/test_1000.csv", stringsAsFactors = FALSE)

# Verify data structure
cat("Training dimensions:", dim(train_G7), "| Test dimensions:", dim(test_G7), "\n")
```

```
## Training dimensions: 1000 763 | Test dimensions: 1000 762
```

```
cat("Target variable present:", "loss" %in% colnames(train_G7), "\n")
```

```
## Target variable present: TRUE
```

Annexure 03 – Data Preprocessing & Feature Engineering

```
## --------------------------
## 3. Data Preprocessing & Feature Engineering
## --------------------------

# Step 3.1: Separate features and target
features <- train_G7 %>% select(-X, -loss)
target <- train_G7$loss
test_features <- test_G7 %>% select(-X)

# Step 3.2: Preprocess only the features
preProc <- preProcess(features,
                      method = c("medianImpute", "nzv", "corr"),
                      cutoff = 0.9)
```

```
# Apply preprocessing
train_features <- predict(preProc, features)
test_processed <- predict(preProc, test_features)

# Step 3.3: Recombine with target for training set
train_processed <- cbind(train_features, loss = target)

# Verification
cat("Processed training dimensions:", dim(train_processed), "\n")
```

```
## Processed training dimensions: 1000 235
```

```
cat("Processed test dimensions:", dim(test_processed), "\n")
```

```
## Processed test dimensions: 1000 234
```

```
cat("Target exists in final train data?", "loss" %in% colnames(train_processed), "\n")
```

```
## Target exists in final train data? TRUE
```

Annexure 04 – Feature selection

```r
## ---------------------------
## 4. Feature Selection
## ---------------------------
# Initial split for feature selection
set.seed(123)
X_temp <- train_processed %>% select(-loss)
Y_temp <- train_processed$loss
train_idx <- createDataPartition(Y_temp, p = 0.7, list = FALSE)
X_fs <- X_temp[train_idx, ]
Y_fs <- Y_temp[train_idx]

# Train initial RF model for feature importance
fs_model <- randomForest(x = X_fs, y = Y_fs, ntree = 100, importance = TRUE)

# Get feature importance scores
importance <- importance(fs_model, type = 1)

# Sort features by importance and select top 50
top_features <- names(sort(importance[, 1], decreasing = TRUE))[1:50]

# Filter datasets to keep only important features
X_train <- X_temp[, top_features, drop = FALSE]
Y_train <- Y_temp  # Target remains unchanged
X_test <- test_processed[, top_features, drop = FALSE]

# Verification
cat("\nFeature Selection Complete:\n")
```

```
##
## Feature Selection Complete:
```

```r
cat("Selected", length(top_features), "most predictive features\n")
```

```
## Selected 50 most predictive features
```

```r
cat("Sample features:", paste(head(top_features, 5), collapse = ", "), "...\n")
```

```
## Sample features: f402, f471, f499, f746, f16 ...
```

```r
cat("Training dimensions:", dim(X_train), "\n")
```

```
## Training dimensions: 1000 50
```

```r
cat("Test dimensions:", dim(X_test), "\n")
```

```
## Test dimensions: 1000 50
```

Annexure 05 – Model Training

```
## ----------------------------
## 5. Model Training
## ----------------------------

# Load required packages if not already loaded
if (!require("doParallel")) install.packages("doParallel")
```

```
## Loading required package: doParallel

## Warning: package 'doParallel' was built under R version 4.4.3

## Loading required package: foreach

##
## Attaching package: 'foreach'

## The following objects are masked from 'package:purrr':
##
##     accumulate, when

## Loading required package: iterators

## Loading required package: parallel
```

```
library(doParallel)

# Set up parallel processing
cl <- makePSOCKcluster(detectCores() - 1)  # Leave one core free
registerDoParallel(cl)

# Define trainControl with parallel support
ctrl <- trainControl(
  method = "cv",
  number = 5,
  allowParallel = TRUE  # Enable parallel processing
)

# 5.1 Random Forest
rf_grid <- expand.grid(.mtry = seq(5, min(50, length(top_features)), by = 5))  # Cap at mtry=50
rf_model <- train(
  x = X_train,
  y = Y_train,
  method = "rf",
  tuneGrid = rf_grid,
  trControl = ctrl,
  ntree = 150,
  importance = TRUE,  # Store importance for later analysis
  verbose = FALSE
)

# 5.2 Elastic Net
glmnet_model <- train(
  x = X_train,
  y = Y_train,
  method = "glmnet",
  trControl = ctrl,
  tuneLength = 10,  # Increased from default 3
  standardize = TRUE,  # Ensure proper scaling
  verbose = FALSE
)

# 5.3 Support Vector Machine (Corrected)
svm_grid <- expand.grid(
  C = 10^seq(-2, 2, length.out = 5),  # Cost parameter
  sigma = 10^seq(-3, 0, length.out = 4)  # More focused sigma range
)

svm_model <- train(
  x = X_train,
  y = Y_train,
  method = "svmRadial",
  tuneGrid = svm_grid,
  trControl = ctrl,
  preProcess = c("center", "scale"),
  verbose = FALSE
)

# Clean up parallel processing
stopCluster(cl)
```

Annexure 06 – Model Evaluation

```
## --------------------------
## 6. Model Evaluation
## --------------------------

model_list <- list(
  "Random Forest" = rf_model,
  "Elastic Net" = glmnet_model,
  "SVM" = svm_model
)

# Create evaluation data frame
val_results <- data.frame(
  Model = names(model_list),
  MAE = sapply(model_list, function(m) round(MAE(predict(m, X_train), Y_train), 6)),
  Rsquared = sapply(model_list, function(m) round(R2(predict(m, X_train), Y_train), 6)),
  stringsAsFactors = FALSE
)

# Create and format flextable
ft <- flextable(val_results) %>%
  set_header_labels(
    Model = "Model Type",
    MAE = "MAE",
    Rsquared = "R-Squared"
  ) %>%
  bg(i = ~ MAE == min(MAE), bg = "#E6F3FF") %>%
  bold(part = "header") %>%
  align(align = "center", part = "all") %>%
  width(width = c(1.5, 1.2, 1.2)) %>%
  fontsize(size = 11) %>%
  theme_booktabs()

# Print the table
ft
```

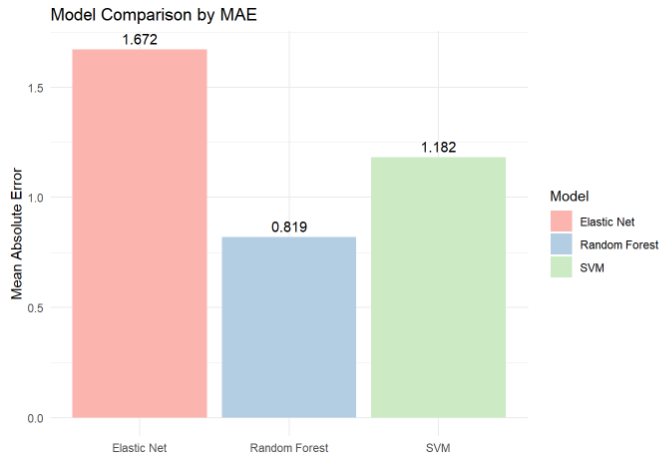
```
## Warning: fonts used in 'flextable' are ignored because the 'pdflatex' engine is
## used and not 'xelatex' or 'lualatex'. You can avoid this warning by using the
## 'set_flextable_defaults(fonts_ignore=TRUE)' command or use a compatible engine
## by defining 'latex_engine: xelatex' in the YAML header of the R Markdown
## document.
```

Model Type	MAE	R-Squared
Random Forest	0.823215	0.905442
Elastic Net	1.656815	0.028492
SVM	1.182263	0.246129

Annexure 07 – Save Final Prediction and Model Comparison

```
## --------------------------
## 8. Save Final Predictions
## --------------------------

# Save predictions without model_used column
write.csv(data.frame(id = test_processed$id, loss = test_pred),
          "final_predictions.csv", row.names = FALSE)
```

```
## --------------------------
## 9. Model Comparison Plot
## --------------------------

# Create MAE comparison plot
mae_plot <- ggplot(val_results, aes(x = Model, y = MAE, fill = Model)) +
  geom_bar(stat = "identity") +
  geom_text(aes(label = round(MAE, 3)), vjust = -0.5) +
  labs(title = "Model Comparison by MAE",
       y = "Mean Absolute Error",
       x = "") +
  theme_minimal() +
  scale_fill_brewer(palette = "Pastel1")

print(mae_plot)
```



Annexure 10 – Feature importance Visualization

```
## ----------------------------
## 10. Feature Importance Visualization
## ----------------------------

# Create feature importance plot for any model type
create_imp_plot <- function(model, model_name) {
  imp_data <- varImp(model)$importance %>%
    rownames_to_column("Feature") %>%
    arrange(desc(Overall)) %>%
    head(20) %>%
    mutate(Feature = factor(Feature, levels = rev(Feature)))

  ggplot(imp_data, aes(x = Feature, y = Overall)) +
    geom_col(fill = ifelse(model_name == "Random Forest", "steelblue", "darkorange")) +
    coord_flip() +
    labs(title = paste("Top 20 Important Features -", model_name),
         x = "Features",
         y = "Importance Score") +
    theme_minimal() +
    theme(plot.title = element_text(hjust = 0.5, face = "bold"),
          axis.text.y = element_text(size = 8))
}

# Generate the appropriate plot
imp_plot <- create_imp_plot(best_model, best_model_name)
```

```
# Display the plot
print(imp_plot)
```