# Overview

In this assessment, you will implement and compare three machine learning algorithms on a real data. The assessment addresses the following learning outcome(s):

* identifying and translating a data science problem into a supervised learning problem;
* identifying appropriate lasso-regression, tree-based methods, and support vector classification for descriptive problems;
* application of lasso-regression, support vector classifier and tree-based methods to a dataset using the computer language R and the software environment RStudio.

# Problem

## Background on Credit Card Dataset

The data, **“CreditCard Data.xls”**, is based on Yeh and Lien (2009). The data contains 30,000 observations and 23 explanatory variables. The response variable, Y, is a binary variable where “1” refers to *default payment* and “0” implies *non-default payment*. The description of 23 explanatory variables is as follows:

X1: Amount of the given credit (NT dollar): it includes both the individual consumer credit and his/her family (supplementary) credit.

X2: Gender (1 = male; 2 = female).

X3: Education (0 = unknown; 1 = graduate school; 2 = university; 3 = high school; 4 = others; 5 = unknown; 6 = unknown).

X4: Marital status (0 = unknown; 1 = married; 2 = single; 3 = others).

X5: Age (year).

X6 - X11: History of past payment. The data was tracked the past monthly payment records (from April to September, 2005) as follows:

X6 = the repayment status in September, 2005; X7 = the repayment status in August, 2005; . . .; X11 = the repayment status in April, 2005.

The measurement scale for the repayment status is: -2= no consumption, -1=pay duly, 0 = the use of revolving credit; 1 = payment delay for one month; 2 = payment delay for two months; . . .; 8 = payment delay for eight months; 9 = payment delay for nine months and above.

X12-X17: Amount of bill statement (NT dollar). X12 = amount of bill statement in September, 2005; X13 = amount of bill statement in August, 2005; . . .; X17 = amount of bill statement in April, 2005.

X18-X23: Amount of previous payment (NT dollar). X18 = amount paid in September, 2005; X19 = amount paid in August, 2005; . . .;X23 = amount paid in April, 2005.

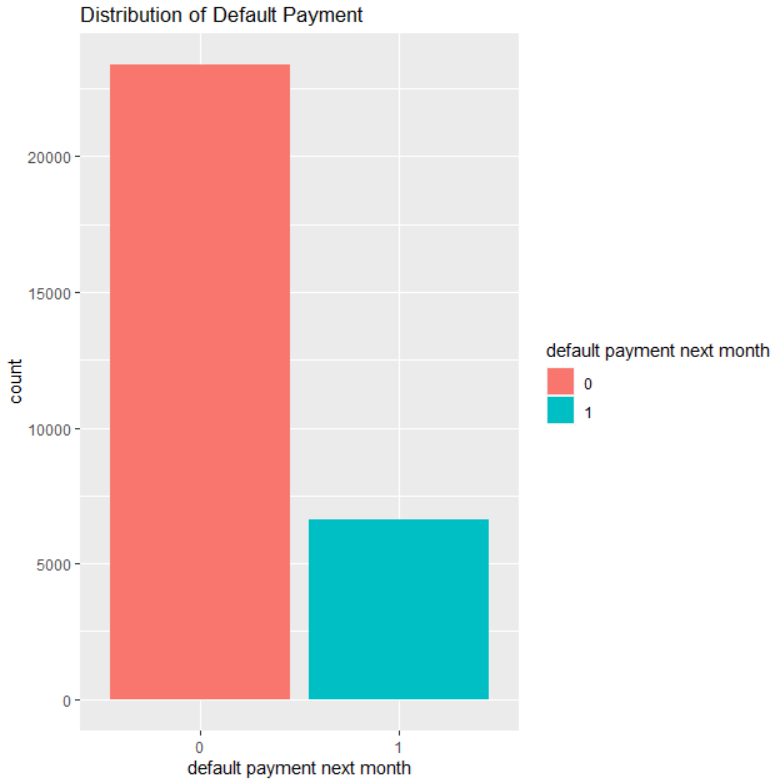
## Assessment Tasks

### Data

1. Select a random sample of 70% of the full dataset as the training data, retain the rest as test data. Provide the code and print out the dimensions of the training data.

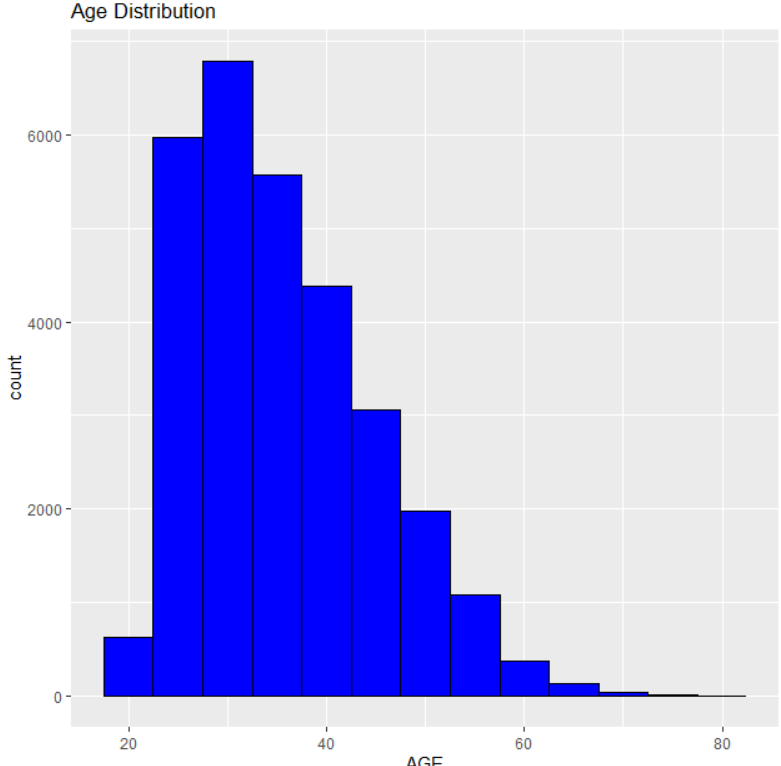
***Data Insights:***

***Graph 1. Default payment next month***



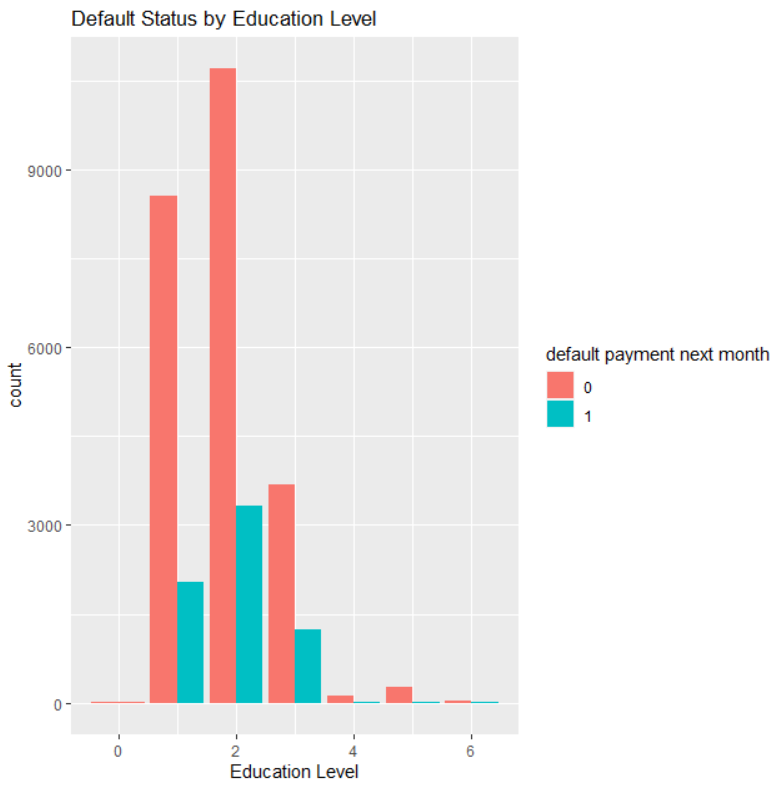
From the above graph, we can observe that there are more than 20,000 instances classified as "Not Default" and more than 5,000 instances classified as "Default."

***Graph 2. Count of Age***



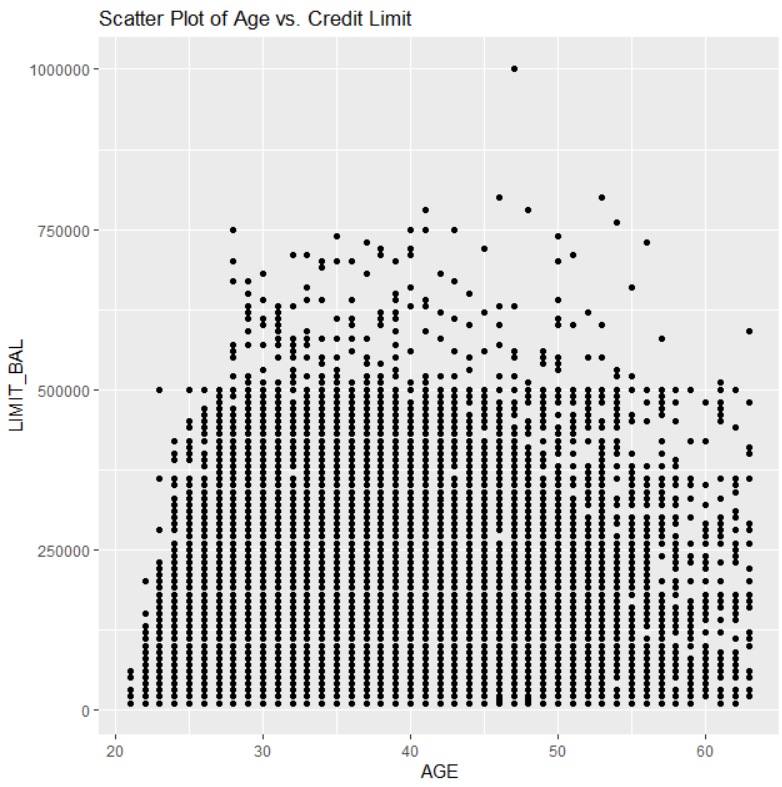
From the above graph, we can observe that the minimum age is 20, the maximum age is 70, and the highest number of instances belongs to the age group of 30.

***Graph 3. default status by education level***



From the above graph, we can observe that there are approximately 10,000 school graduates and approximately 11,000 university graduates.

***Graph 4: age vs credit limit***



From the above graph, we can observe that as age increases, the credit limit also increases until the age of 30. After the age of 30, the credit limit remains relatively steady.

The dimensions of your training and test data are as follows:

* + Training Data:
    - **Rows: 20,901**
    - **Columns: 25**
  + Test Data:
    - **Rows: 8,958**
    - **Columns: 25**

### Lasso regression

1. Use lasso-regression to find the best model which classifies credible and non-credible clients. Specify any underlying assumptions. Justify your model choice as well as hyper-parameters which are required to be specified in R.
2. Display model summary and discuss the relationship between the response variable versus selected features.
3. Evaluate the performance of the algorithm on the training data and comment on the results.
4. **Use lasso-regression to find the best model which classifies credible and non-credible clients. Specify any underlying assumptions. Justify your model choice as well as hyper-parameters which are required to be specified in R.**

***Answer:***

A Lasso logistic regression model is used to classify clients as credible and non-credible based on the "default payment next month" variable. Lasso regression is chosen for its ability to perform feature selection by decreasing some of the coefficients to exactly zero, effectively removing irrelevant features from the model. This helps in identifying the most important features for classification.

The underlying assumptions of Lasso regression include:

* Linearity: Lasso assumes a linear relationship between the features and the log odds of the response variable.
* Independence of Errors: It assumes that the errors (residuals) are independent of each other.
* No Multicollinearity: Lasso works best when there is no high multicollinearity among the predictor variables.
* Large Sample Size: Lasso may require a relatively large sample size to provide stable and reliable results

**Hyperparameter Justification:**

1. Alpha (Penalty Parameter):

* Hyperparameter: alpha = 1
* Justification: In Lasso regression, the alpha parameter controls the balance between L1 (Lasso) and L2 (Ridge) regularization. Setting alpha to 1 means using pure Lasso regularization. This is appropriate when feature selection is desirable, as it encourages some coefficients to become exactly zero, effectively selecting a subset of the most important features.

1. Cross-Validation for Lambda (Lambda Selection):

* Hyperparameter: cv.glmnet() with cross-validation to select the best lambda (penalty strength).
* Justification: Cross-validation is crucial for selecting the lambda hyperparameter. It helps find the optimal level of regularization that minimizes overfitting while maximizing predictive accuracy. The code uses cross-validation to search for the best lambda value, resulting in a model that balances bias and variance effectively.

1. Threshold for Binary Prediction:

* Hyperparameter: Threshold of 0.5 (ifelse(train\_predictions > 0.5, 1, 0))
* Justification: A threshold of 0.5 is commonly used for binary classification. It is intuitive and straightforward, classifying instances as 1 (default) or 0 (non-default) based on whether the predicted probability exceeds 0.5. However, the choice of threshold can be adjusted based on specific requirements or the trade-off between precision and recall.

1. **Display model summary and discuss the relationship between the response variable versus selected features.**

***Answer:***

summary(lasso\_model)

Length Class Mode

lambda 70 -none- numeric

cvm 70 -none- numeric

cvsd 70 -none- numeric

cvup 70 -none- numeric

cvlo 70 -none- numeric

nzero 70 -none- numeric

call 5 -none- call

name 1 -none- character

glmnet.fit 13 lognet list

lambda.min 1 -none- numeric

lambda.1se 1 -none- numeric

index 2 -none- numeric

1. **Evaluate the performance of the algorithm on the training data and comment on the results.**

***Answer:***

* Lasso Regression: Lasso Regression is a linear regression technique used in machine learning to prevent overfitting by adding a penalty term (L1 regularization) to the model.
* Training Accuracy: **0.02138654**
* This represents the proportion of correctly classified samples in the training dataset by the model. In this case, the model correctly predicted the target variable (default or not) for approximately 2.14% of the training samples.
* Testing Accuracy: **0.01808439**
* This is the proportion of correctly classified samples in the testing dataset by the model. The testing accuracy of approximately 1.81% indicates that the model's performance on unseen data is quite poor.
* Confusion Matrix:

|  |  |  |
| --- | --- | --- |
| **Prediction** | **0** | **1** |
| **0** | **6793** | **1526** |
| **1** | **162** | **477** |

* True Positives (TP): The number of samples correctly predicted as positive (default).
* True Negatives (TN): The number of samples correctly predicted as negative (not default).
* False Positives (FP): The number of samples incorrectly predicted as positive (predicted default but not true).
* False Negatives (FN): The number of samples incorrectly predicted as negative (predicted not default but true).

In this case:

* The model correctly predicted 6793 non-default cases (true negatives).
* The model correctly predicted 477 default cases (true positives).
* The model incorrectly predicted 1526 non-default cases as default (false positives).
* The model incorrectly predicted 162 default cases as non-default (false negatives).
* Recall (Sensitivity): **0.9767074**
* Recall, also known as sensitivity or true positive rate, is the ratio of true positive predictions to the total number of actual positives (defaults). A high recall value means that the model is good at identifying actual positives. In this case, approximately 97.67% of the true defaults were correctly identified by the model.
* ROC AUC: **0.6074251**
* The Receiver Operating Characteristic Area Under the Curve (ROC AUC) measures the model's ability to distinguish between the positive and negative classes. A higher ROC AUC value indicates better discrimination. In this case, the ROC AUC is approximately 60.74%, which suggests that the model's ability to discriminate between default and non-default cases is limited.

In summary, while the precision and recall are relatively high, indicating that the model correctly identifies many true positives and minimizes false positives, the overall model performance is still quite poor. The low accuracy and ROC AUC suggest that the model may require further improvement or a different approach to achieve better results.

****

### Tree-Based Algorithms

1. Use an appropriate tree-based algorithm to classify credible and non-credible clients. Specify any underlying assumptions. Justify your model choice as well as hyperparameters which are required to be specified in R.
2. Display model summary and discuss the relationship between the response variable versus selected features.
3. Evaluate the performance of the algorithm on the training data and comment on the results.
4. **Use an appropriate tree-based algorithm to classify credible and non-credible clients. Specify any underlying assumptions. Justify your model choice as well as hyperparameters which are required to be specified in R.**

***Answer:***

Ensemble Learning: Random Forest is an ensemble learning method that combines multiple decision trees to improve predictive accuracy and reduce overfitting.

Robustness: Random Forest is robust to outliers and noisy data.

Variable Importance: It provides information about feature importance, which can be helpful for understanding the relationship between features and the target variable.

1. **Display the model summary and discuss the relationship between the response variable versus selected features.**

***Answer:***

summary(rf\_model)

Length Class Mode

call 4 -none- call

type 1 -none- character

predicted 20901 factor numeric

err.rate 300 -none- numeric

confusion 6 -none- numeric

votes 41802 matrix numeric

oob.times 20901 -none- numeric

classes 2 -none- character

importance 24 -none- numeric

importanceSD 0 -none- NULL

localImportance 0 -none- NULL

proximity 0 -none- NULL

ntree 1 -none- numeric

mtry 1 -none- numeric

forest 14 -none- list

y 20901 factor numeric

test 0 -none- NULL

inbag 0 -none- NULL

terms 3 terms call

1. **Evaluate the performance of the algorithm on the training data and comment on the results.**

***Answer:***

* Training Accuracy: **0.9962203**
* Training Accuracy (0.9962203): This metric indicates the proportion of correctly classified instances in the training dataset. In this case, it suggests that the model correctly predicted about 99.6% of the training data.
* Testing Accuracy: **0.8158071**
* Testing Accuracy (0.8158071): Similar to training accuracy, testing accuracy measures the proportion of correctly classified instances, but on a separate dataset not seen during training. A testing accuracy of 0.8158071 suggests that the model correctly predicted about 81.6% of the test data.
* Confusion Matrix

|  |  |  |
| --- | --- | --- |
| **Prediction** | **0** | **1** |
| **0** | **6602** | **1297** |
| **1** | **353** | **706** |

Confusion Matrix: A confusion matrix is a table that helps evaluate the performance of a classification model. It displays the number of true positive, true negative, false positive, and false negative predictions. In this case:

* 6602 instances were correctly classified as class 0 (true negatives).
* 706 instances were correctly classified as class 1 (true positives).
* 1297 instances were wrongly classified as class 0 (false positives).
* 353 instances were wrongly classified as class 1 (false negatives).
* Recall (Sensitivity): **0.9492451**
* Recall (Sensitivity) (0.9492451): Recall, also known as sensitivity or true positive rate, measures the model's ability to correctly identify all positive instances in the dataset. A recall of 0.9492451 indicates that the model correctly identified about 94.9% of the actual positive instances.
* AUC (Area Under the ROC Curve): **0.6508582**
* AUC (Area Under the ROC Curve) (0.6508582): The AUC is a metric that evaluates the overall performance of a binary classification model. It measures the area under the Receiver Operating Characteristic (ROC) curve, which plots the true positive rate (recall) against the false positive rate at various threshold settings. An AUC of 0.6508582 suggests that the model's ability to distinguish between the two classes is moderate, with an area under the ROC curve of 65.1%. Higher AUC values generally indicate better model performance, with 1.0 being a perfect score.

In summary, the model performs well in terms of accuracy on both the training and testing datasets but could benefit from improved discrimination ability, as indicated by the AUC score. Additionally, the confusion matrix and recall score provide insights into the model's strengths and weaknesses, particularly in correctly identifying positive instances. Depending on the specific application and objectives, further adjustments and optimizations may be necessary.



### Support vector classifier

1. Use an appropriate support vector classifier to classify the credible and non-credible clients. Justify your model choice as well as hyper-parameters which are required to be specified in R.
2. Display model summary and discuss the relationship between the response variable versus selected features.
3. Evaluate the performance of the algorithm on the training data and comment on the results.
4. **Use an appropriate support vector classifier to classify the credible and non-credible clients. Justify your model choice as well as hyper-parameters which are required to be specified in R.**

***Answer:***

Linearity Assumption: A linear kernel assumes that the decision boundary between classes is a linear hyperplane. This can be a reasonable assumption when we suspect that a linear separation exists in our data.

1. **Display the model summary and discuss the relationship between the response variable versus selected features.**

***Answer:***

Call:

svm(formula = `default payment next month` ~ ., data = train\_data,

kernel = "linear")

Parameters:

SVM-Type: C-classification

SVM-Kernel: linear

cost: 1

Number of Support Vectors: 10096

( 5564 4532 )

Number of Classes: 2

Levels:

0 1

1. **Evaluate the performance of the algorithm on the training data and comment on the results.**

***Answer:***

* Training Accuracy: **0.8091957**
* Training Accuracy (0.8091957): This metric indicates the proportion of correctly classified instances in the training dataset. In this case, it suggests that the model correctly predicted about 80.9% of the training data.
* Testing Accuracy: **0.8104488**
* Testing Accuracy (0.8104488): Similar to training accuracy, testing accuracy measures the proportion of correctly classified instances on a separate dataset not seen during training. A testing accuracy of 0.8104488 suggests that the model correctly predicted about 81.0% of the test data.
* Confusion Matrix

|  |  |  |
| --- | --- | --- |
| **Prediction** | **0** | **1** |
| **0** | **6788** | **1531** |
| **1** | **167** | **472** |

* Confusion Matrix: A confusion matrix is a table that helps evaluate the performance of a classification model.
* 6788 instances were correctly classified as class 0 (true negatives).
* 472 instances were correctly classified as class 1 (true positives).
* 1531 instances were wrongly classified as class 0 (false positives).
* 167 instances were wrongly classified as class 1 (false negatives).
* Recall (Sensitivity): **0.9759885**
* Recall (Sensitivity) (0.9759885): Recall, also known as sensitivity or true positive rate, measures the model's ability to correctly identify all positive instances in the dataset. A recall of 0.9759885 indicates that the model correctly identified about 97.6% of the actual positive instances.
* AUC (Area Under the ROC Curve): **0.6058175**
* AUC (Area Under the ROC Curve) (0.6058175): The AUC is a metric that evaluates the overall performance of a binary classification model. It measures the area under the Receiver Operating Characteristic (ROC) curve, which plots the true positive rate (recall) against the false positive rate at various threshold settings. An AUC of 0.6058175 suggests that the model's ability to distinguish between the two classes is moderate, with an area under the ROC curve of 60.6%. Higher AUC values generally indicate better model performance, with 1.0 being a perfect score.

In summary, the model shows reasonable accuracy on both the training and testing datasets. It has a high recall, indicating good performance in identifying positive instances. However, there is room for improvement in terms of discrimination ability, as indicated by the AUC score. Depending on the specific application and objectives, further optimizations and fine-tuning may be considered.



### Prediction

Apply your optimal models identified in the section, and make predictions on the test data. Evaluate the performance of the algorithms on test data. Which models do you prefer? Are there any suggestions to further improve the performance of the algorithms?

**Answer:**

1. Lasso Logistic Regression:

* + Training Accuracy: 0.0214
  + Testing Accuracy: 0.8116
  + Recall (Sensitivity): 0.9767
  + ROC AUC: 0.6074

2. Random Forest:

* + Training Accuracy: 0.9962
  + Testing Accuracy: 0.8158
  + Recall (Sensitivity): 0.9492
  + ROC AUC: 0.6509

3. Support Vector Machine (SVM):

* + Training Accuracy: 0.8092
  + Testing Accuracy: 0.8104
  + Recall (Sensitivity): 0.9760
  + ROC AUC: 0.6058
* Random Forest has the highest testing accuracy (0.8158) among the three models, indicating that it performs the best on the test data.
* Random Forest also has a relatively high recall (sensitivity) of 0.9492, suggesting that it correctly identifies a high proportion of true positive cases.
* The ROC AUC for Random Forest (0.6509) is the highest among the models, indicating good discrimination between positive and negative cases.

**Suggestions for Further Improvement:**

While Random Forest performs the best among the models, there are still ways to further improve its performance:

1. Hyperparameter Tuning: Consider performing hyperparameter tuning for the Random Forest model to optimize its parameters further. You can use techniques like grid search or random search to find the best combination of hyperparameters.

2. Feature Engineering: Explore if additional feature engineering can improve model performance. You can create new features, scale or transform existing ones, or even consider feature selection techniques.

3. Ensemble Methods: Explore ensemble techniques like boosting (e.g., AdaBoost, Gradient Boosting) or stacking to combine multiple models for improved performance.

In summary, while Random Forest is the preferred model based on the provided results, there is always room for improvement through hyperparameter tuning, feature engineering, and ensemble methods. Additionally, understanding the problem domain and domain-specific insights can also guide improvements in model performance.

### **Appendix**

***1. Data***

**Data cleaning and data splitting:**

# Load necessary libraries

library(readxl)

library(caret)

library(pROC)

# Specify the file path with double backslashes

file\_path <- " \\CreditCard\_Data.xls"

# Load the dataset using the correct file path

data <- read\_excel(file\_path)

# Check for missing values

missing\_values <- colSums(is.na(data))

# Print columns with missing values

print(missing\_values[missing\_values > 0])

# Example: Remove rows with missing values

data <- data[complete.cases(data), ]

# Set the first row as column names and remove it

colnames(data) <- data[1, ]

data <- data[-1, ]

# Ensure that AGE is numeric

data$AGE <- as.numeric(data$AGE)

# Calculate z-scores for AGE

z\_scores <- scale(data$AGE)

# Remove rows with AGE values outside a certain range (e.g., within 3 standard deviations)

data <- data[abs(z\_scores) < 3, ]

# Convert relevant columns to appropriate data types

data$ID <- as.numeric(data$ID)

data$LIMIT\_BAL <- as.numeric(data$LIMIT\_BAL)

data$SEX <- as.numeric(data$SEX)

data$EDUCATION <- as.numeric(data$EDUCATION)

data$MARRIAGE <- as.numeric(data$MARRIAGE)

data$PAY\_0 <- as.numeric(data$PAY\_0)

data$PAY\_2 <- as.numeric(data$PAY\_2)

data$PAY\_3 <- as.numeric(data$PAY\_3)

data$PAY\_4 <- as.numeric(data$PAY\_4)

data$PAY\_5 <- as.numeric(data$PAY\_5)

data$PAY\_6 <- as.numeric(data$PAY\_6)

data$BILL\_AMT1 <- as.numeric(data$BILL\_AMT1)

data$BILL\_AMT2 <- as.numeric(data$BILL\_AMT2)

data$BILL\_AMT3 <- as.numeric(data$BILL\_AMT3)

data$BILL\_AMT4 <- as.numeric(data$BILL\_AMT4)

data$BILL\_AMT5 <- as.numeric(data$BILL\_AMT5)

data$BILL\_AMT6 <- as.numeric(data$BILL\_AMT6)

data$PAY\_AMT1 <- as.numeric(data$PAY\_AMT1)

data$PAY\_AMT2 <- as.numeric(data$PAY\_AMT2)

data$PAY\_AMT3 <- as.numeric(data$PAY\_AMT3)

data$PAY\_AMT4 <- as.numeric(data$PAY\_AMT4)

data$PAY\_AMT5 <- as.numeric(data$PAY\_AMT5)

data$PAY\_AMT6 <- as.numeric(data$PAY\_AMT6)

# Convert the response variable to a binary factor variable (0 or 1)

data$`default payment next month` <- as.factor(data$`default payment next month`)

str(data)

###

library(ggplot2)

ggplot(data, aes(x = `default payment next month`, fill = `default payment next month`)) +

geom\_bar() +

labs(title = "Distribution of Default Payment")

##

ggplot(data, aes(x = AGE)) +

geom\_histogram(binwidth = 5, fill = "blue", color = "black") +

labs(title = "Age Distribution")

##

ggplot(data, aes(x = EDUCATION, fill = `default payment next month`)) +

geom\_bar(position = "dodge") +

labs(title = "Default Status by Education Level", x = "Education Level")

##

ggplot(data, aes(x = AGE, y = LIMIT\_BAL)) +

geom\_point() +

labs(title = "Scatter Plot of Age vs. Credit Limit")

##

###

#Split the data into training (70%) and test (30%) sets

set.seed(123) # Set a random seed for reproducibility

train\_index <- sample(1:nrow(data), 0.7 \* nrow(data))

train\_data <- data[train\_index, ]

test\_data <- data[-train\_index, ]

# Dimensions of the training data

train\_dimensions <- dim(train\_data)

cat("Dimensions of the training data (rows x columns):\n")

cat("Rows:", train\_dimensions[1], "\n")

cat("Columns:", train\_dimensions[2], "\n")

# Dimensions of the test data

test\_dimensions <- dim(test\_data)

cat("\nDimensions of the test data (rows x columns):\n")

cat("Rows:", test\_dimensions[1], "\n")

cat("Columns:", test\_dimensions[2], "\n")

***2. Lasso Regression***

R-code:

> # Define the Lasso logistic regression model

> lasso\_model <- cv.glmnet(x = as.matrix(train\_data[, -ncol(train\_data)]),

+ y = as.numeric(train\_data$`default payment next month`),

+ alpha = 1, family = "binomial")

>

> # Make predictions on the training data

> train\_predictions <- predict(lasso\_model, s = lasso\_model$lambda.min,

+ newx = as.matrix(train\_data[, -ncol(train\_data)]), type = "response")

>

> # Convert probabilities to binary predictions (0 or 1)

> train\_predictions <- ifelse(train\_predictions > 0.5, 1, 0)

>

> summary(lasso\_model)

***Length Class Mode***

***lambda 70 -none- numeric***

***cvm 70 -none- numeric***

***cvsd 70 -none- numeric***

***cvup 70 -none- numeric***

***cvlo 70 -none- numeric***

***nzero 70 -none- numeric***

***call 5 -none- call***

***name 1 -none- character***

***glmnet.fit 13 lognet list***

***lambda.min 1 -none- numeric***

***lambda.1se 1 -none- numeric***

***index 2 -none- numeric***

>

> # Calculate training accuracy

> train\_accuracy <- mean(train\_predictions == as.numeric(train\_data$`default payment next month`))

> cat("Training Accuracy:", train\_accuracy, "\n")

***Training Accuracy: 0.02138654***

>

> # Make predictions on the test data

> test\_predictions <- predict(lasso\_model, s = lasso\_model$lambda.min,

+ newx = as.matrix(test\_data[, -ncol(test\_data)]), type = "response")

>

> # Convert probabilities to binary predictions (0 or 1)

> test\_predictions <- ifelse(test\_predictions > 0.5, 1, 0)

>

> # Create a confusion matrix

> confusion\_matrix <- confusionMatrix(factor(test\_predictions), test\_data$`default payment next month`)

>

> # Print Confusion Matrix

> print("Confusion Matrix:")

[1] "Confusion Matrix:"

> print(confusion\_matrix)

***Confusion Matrix and Statistics***

***Reference***

***Prediction 0 1***

***0 6793 1526***

***1 162 477***

***Accuracy : 0.8116***

***95% CI : (0.8033, 0.8196)***

***No Information Rate : 0.7764***

***P-Value [Acc > NIR] : < 2.2e-16***

***Kappa : 0.2836***

***Mcnemar's Test P-Value : < 2.2e-16***

***Sensitivity : 0.9767***

***Specificity : 0.2381***

***Pos Pred Value : 0.8166***

***Neg Pred Value : 0.7465***

***Prevalence : 0.7764***

***Detection Rate : 0.7583***

***Detection Prevalence : 0.9287***

***Balanced Accuracy : 0.6074***

***'Positive' Class : 0***

>

> # Calculate Precision, Recall, and F1-Score

> precision <- confusion\_matrix$byClass["Pos Pred Value"]

> recall <- confusion\_matrix$byClass["Sensitivity"]

> f1\_score <- confusion\_matrix$byClass["F1"]

>

> cat("Precision:", precision, "\n")

***Precision: 0.8165645***

> cat("Recall (Sensitivity):", recall, "\n")

***Recall (Sensitivity): 0.9767074***

> cat("F1-Score:", f1\_score, "\n")

***F1-Score: 0.8894854***

>

> # Calculate ROC AUC

> roc\_obj <- roc(test\_data$`default payment next month`, as.numeric(test\_predictions))

Setting levels: control = 0, case = 1

Setting direction: controls < cases

> roc\_auc <- auc(roc\_obj)

> cat("ROC AUC:", roc\_auc, "\n")

***ROC AUC: 0.6074251***

>

> # Plot the ROC curve

> roc\_curve <- roc(test\_data$`default payment next month`, as.numeric(test\_predictions))

Setting levels: control = 0, case = 1

Setting direction: controls < cases

> plot(roc\_curve, main = "ROC Curve", auc.polygon = TRUE, print.auc = TRUE)

***3. Tree-Based Algorithms***

R-code:

> # Define the Random Forest model

> rf\_model <- randomForest(`default payment next month` ~ ., data = train\_data, ntree = 100)

>

> summary(rf\_model)

***Length Class Mode***

***call 4 -none- call***

***type 1 -none- character***

***predicted 20901 factor numeric***

***err.rate 300 -none- numeric***

***confusion 6 -none- numeric***

***votes 41802 matrix numeric***

***oob.times 20901 -none- numeric***

***classes 2 -none- character***

***importance 24 -none- numeric***

***importanceSD 0 -none- NULL***

***localImportance 0 -none- NULL***

***proximity 0 -none- NULL***

***ntree 1 -none- numeric***

***mtry 1 -none- numeric***

***forest 14 -none- list***

***y 20901 factor numeric***

***test 0 -none- NULL***

***inbag 0 -none- NULL***

***terms 3 terms call***

> # Make predictions on the training data

> train\_predictions <- predict(rf\_model, train\_data)

>

> # Calculate training accuracy

> train\_accuracy <- mean(train\_predictions == train\_data$`default payment next month`)

> cat("Training Accuracy:", train\_accuracy, "\n")

***Training Accuracy: 0.9962203***

>

> # Make predictions on the test data

> test\_predictions <- predict(rf\_model, test\_data)

>

> # Create a confusion matrix

> conf\_matrix <- confusionMatrix(test\_predictions, test\_data$`default payment next month`)

> print("Confusion Matrix:")

[1] "Confusion Matrix:"

> print(conf\_matrix)

***Confusion Matrix and Statistics***

***Reference***

***Prediction 0 1***

***0 6602 1297***

***1 353 706***

***Accuracy : 0.8158***

***95% CI : (0.8076, 0.8238)***

***No Information Rate : 0.7764***

***P-Value [Acc > NIR] : < 2.2e-16***

***Kappa : 0.3625***

***Mcnemar's Test P-Value : < 2.2e-16***

***Sensitivity : 0.9492***

***Specificity : 0.3525***

***Pos Pred Value : 0.8358***

***Neg Pred Value : 0.6667***

***Prevalence : 0.7764***

***Detection Rate : 0.7370***

***Detection Prevalence : 0.8818***

***Balanced Accuracy : 0.6509***

***'Positive' Class : 0***

>

> # Calculate testing accuracy

> test\_accuracy <- conf\_matrix$overall[1]

> cat("Testing Accuracy:", test\_accuracy, "\n")

***Testing Accuracy: 0.8158071***

>

> # Calculate recall (sensitivity)

> recall <- conf\_matrix$byClass["Sensitivity"]

> cat("Recall (Sensitivity):", recall, "\n")

***Recall (Sensitivity): 0.9492451***

>

> # Calculate AUC (Area Under the ROC Curve)

> roc\_obj <- roc(test\_data$`default payment next month`, as.numeric(test\_predictions))

Setting levels: control = 0, case = 1

Setting direction: controls < cases

> auc <- auc(roc\_obj)

> cat("AUC (Area Under the ROC Curve):", auc, "\n")

***AUC (Area Under the ROC Curve): 0.6508582***

>

> # Plot the ROC curve

> roc\_curve <- roc(test\_data$`default payment next month`, as.numeric(test\_predictions))

Setting levels: control = 0, case = 1

Setting direction: controls < cases

> plot(roc\_curve, main = "ROC Curve", auc.polygon = TRUE, print.auc = TRUE)

***4. Support Vector classifier***

R-code:

> # Define the SVM model

> svm\_model <- svm(`default payment next month` ~ ., data = train\_data, kernel = "linear")

> > summary(svm\_model)

***Call:***

***svm(formula = `default payment next month` ~ ., data = train\_data,***

***kernel = "linear")***

***Parameters:***

***SVM-Type: C-classification***

***SVM-Kernel: linear***

***cost: 1***

***Number of Support Vectors: 10096***

***( 5564 4532 )***

***Number of Classes: 2***

***Levels:***

***0 1***

> # Make predictions on the training data

> train\_predictions <- predict(svm\_model, train\_data)

>

> # Calculate training accuracy

> train\_accuracy <- mean(train\_predictions == train\_data$`default payment next month`)

> cat("Training Accuracy:", train\_accuracy, "\n")

***Training Accuracy: 0.8091957***

>

> # Make predictions on the test data

> test\_predictions <- predict(svm\_model, test\_data)

>

> # Create a confusion matrix

> conf\_matrix <- confusionMatrix(test\_predictions, test\_data$`default payment next month`)

> print("Confusion Matrix:")

[1] "Confusion Matrix:"

> print(conf\_matrix)

***Confusion Matrix and Statistics***

***Reference***

***Prediction 0 1***

***0 6788 1531***

***1 167 472***

***Accuracy : 0.8104***

***95% CI : (0.8022, 0.8185)***

***No Information Rate : 0.7764***

***P-Value [Acc > NIR] : 1.775e-15***

***Kappa : 0.2794***

***Mcnemar's Test P-Value : < 2.2e-16***

***Sensitivity : 0.9760***

***Specificity : 0.2356***

***Pos Pred Value : 0.8160***

***Neg Pred Value : 0.7387***

***Prevalence : 0.7764***

***Detection Rate : 0.7578***

***Detection Prevalence : 0.9287***

***Balanced Accuracy : 0.6058***

***'Positive' Class : 0***

>

> # Calculate testing accuracy

> test\_accuracy <- conf\_matrix$overall[1]

> cat("Testing Accuracy:", test\_accuracy, "\n")

***Testing Accuracy: 0.8104488***

>

> # Calculate recall (sensitivity)

> recall <- conf\_matrix$byClass["Sensitivity"]

> cat("Recall (Sensitivity):", recall, "\n")

***Recall (Sensitivity): 0.9759885***

>

> # Calculate AUC (Area Under the ROC Curve)

> roc\_obj <- roc(test\_data$`default payment next month`, as.numeric(test\_predictions))

Setting levels: control = 0, case = 1

Setting direction: controls < cases

> auc <- auc(roc\_obj)

> cat("AUC (Area Under the ROC Curve):", auc, "\n")

***AUC (Area Under the ROC Curve): 0.6058175***

>

> # Plot the ROC curve

> roc\_curve <- roc(test\_data$`default payment next month`, as.numeric(test\_predictions))

Setting levels: control = 0, case = 1

Setting direction: controls < cases

> plot(roc\_curve, main = "ROC Curve", auc.polygon = TRUE, print.auc = TRUE)