Exercise 6 - Advanced Methods for Regression and Classification

12433732 - Stefan Merdian

2024-11-27

```r
library(ROCit)
```

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

```r
data("Loan", package = "ROCit")
```

```r
indicator <- Loan$Status

Loan$Status <- ifelse(Loan$Status == "CO", 0, 1)

#data.frame(Status = indicator, NumericStatus = Loan$Status)
```

```r
str(Loan)
```

```
## 'data.frame':    900 obs. of  9 variables:
##  $ Amount : num  67.6 23 54 24.3 43.2 ...
##  $ Term   : int  36 36 36 36 36 36 36 36 36 36 ...
##  $ IntRate: num  0.184 0.12 0.117 0.173 0.172 ...
##  $ ILR    : num  0.035 0.032 0.032 0.034 0.034 0.033 0.035 0.03 0.031 0.034 ...
##  $ EmpLen : Factor w/ 5 levels "A","B","C","D",..: 4 4 4 1 1 2 4 4 2 4 ...
##  $ Home   : Factor w/ 3 levels "MORTGAGE","OWN",..: 3 3 1 3 1 3 3 1 1 1 ...
##  $ Income : num  126400 30900 111900 66000 71900 ...
##  $ Status : num  0 0 1 1 0 1 1 1 1 1 ...
##  $ Score  : num  201 180 162 197 203 ...
```

```r
head(Loan)
```

```
##   Amount Term IntRate   ILR EmpLen     Home Income Status    Score
## 1  67.57   36  0.1838 0.035      D     RENT 126400      0 200.9581
## 2  22.97   36  0.1198 0.032      D     RENT  30900      0 179.6058
## 3  54.05   36  0.1166 0.032      D MORTGAGE 111900      1 161.7622
## 4  24.32   36  0.1733 0.034      A     RENT  66000      1 196.6619
## 5  43.24   36  0.1723 0.034      A MORTGAGE  71900      0 203.4912
## 6  16.22   36  0.1355 0.033      B     RENT  27614      1 186.3070
```

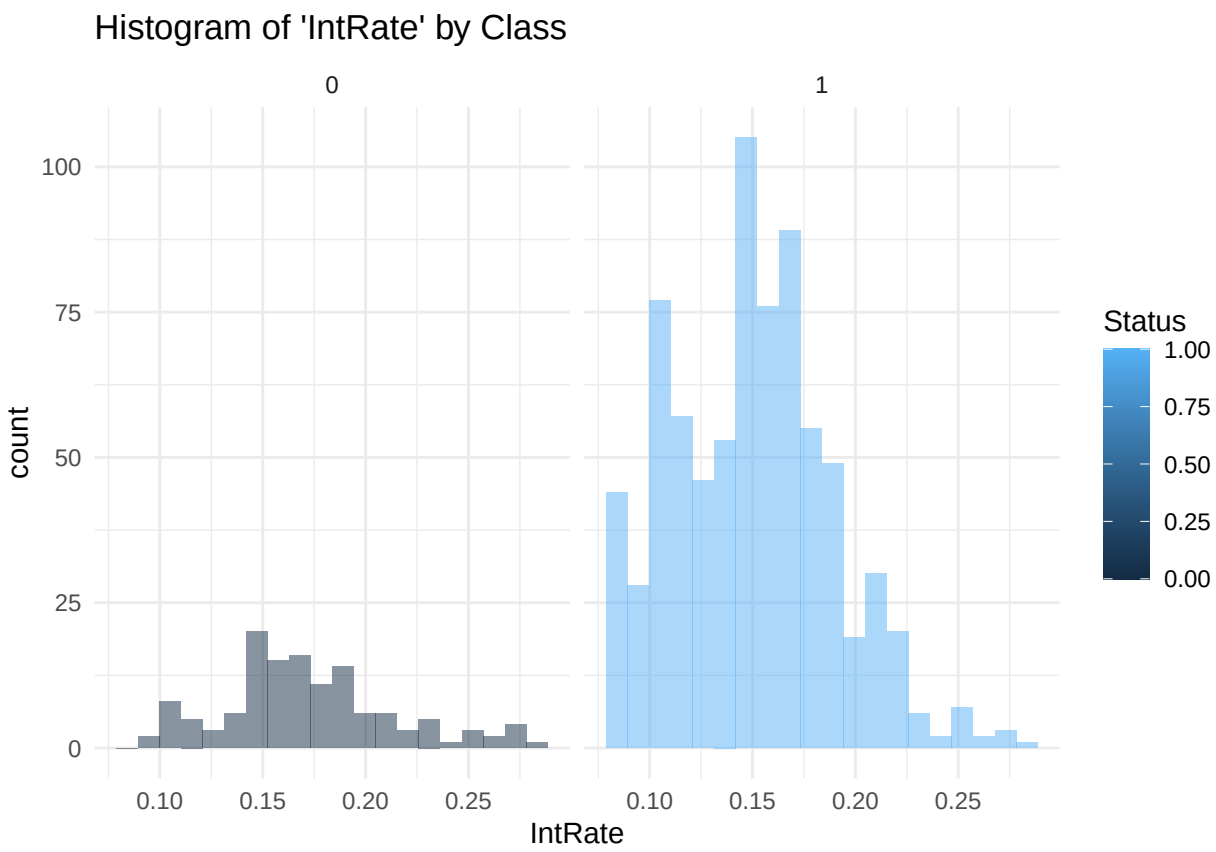
```r
summary(Loan)
```

```
##      Amount           Term       IntRate            ILR          EmpLen
```

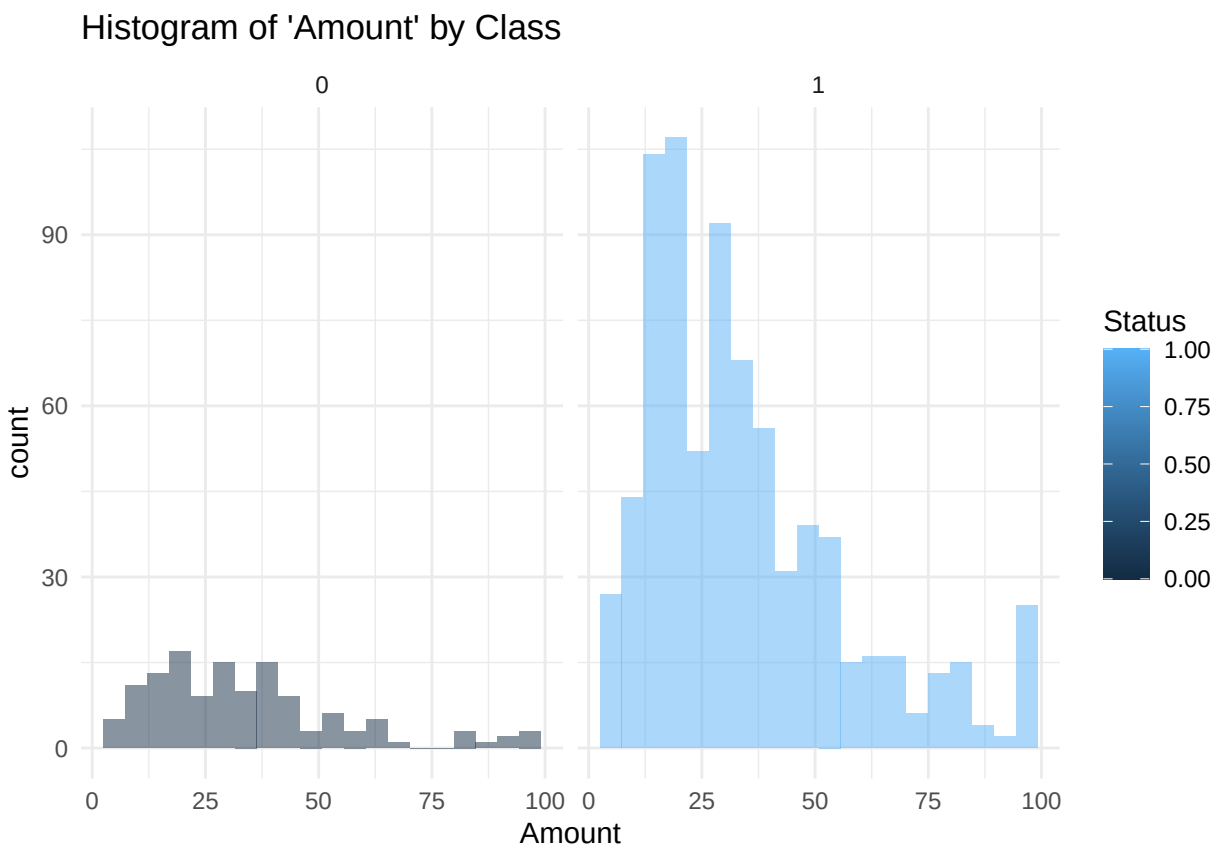
```
##  Min.   : 2.70   Min.   :36   Min.   :0.0830   Min.   :0.03000   A:198
##  1st Qu.:18.04   1st Qu.:36   1st Qu.:0.1219   1st Qu.:0.03200   B:198
##  Median :27.03   Median :36   Median :0.1513   Median :0.03300   C:141
##  Mean   :34.08   Mean   :36   Mean   :0.1529   Mean   :0.03353   D:305
##  3rd Qu.:43.34   3rd Qu.:36   3rd Qu.:0.1775   3rd Qu.:0.03500   U: 58
##  Max.   :94.59   Max.   :36   Max.   :0.2825   Max.   :0.04000
##        Home         Income           Status           Score
##  MORTGAGE:429   Min.   : 11900   Min.   :0.0000   Min.   : -5.449
##  OWN     : 95   1st Qu.: 43900   1st Qu.:1.0000   1st Qu.:169.358
##  RENT    :376   Median : 63000   Median :1.0000   Median :189.120
##                 Mean   : 72903   Mean   :0.8544   Mean   :187.440
##                 3rd Qu.: 86900   3rd Qu.:1.0000   3rd Qu.:205.064
##                 Max.   :502000   Max.   :1.0000   Max.   :269.171
```

```r
set.seed(123)

sample <- sample(c(TRUE, FALSE), nrow(Loan), replace=TRUE, prob=c(0.7,0.3))
train_data <- Loan[sample, ]
test_data <- Loan[!sample, ]

train_data <- train_data[, !names(train_data) %in% "Term"]
test_data <- test_data[, !names(test_data) %in% "Term"]

train_data <- train_data[, !names(train_data) %in% "ILR"]
test_data <- test_data[, !names(test_data) %in% "ILR"]

train_data <- train_data[, !names(train_data) %in% "Score"]
test_data <- test_data[, !names(test_data) %in% "Score"]
```

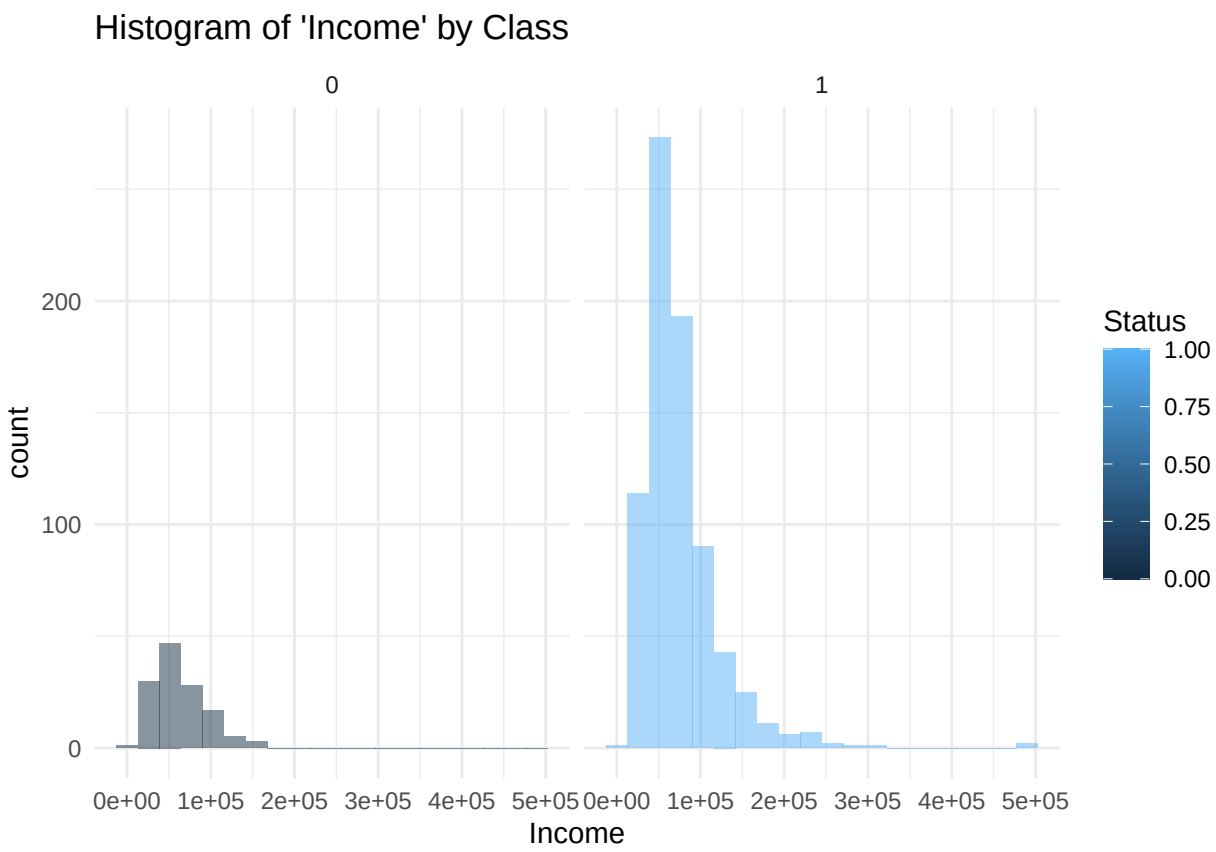
```r
# Assuming you have a dataset named 'Loan' and a target variable named 'Outcome'
library(ggplot2)

# Plot histograms of 'Amount' feature by class
ggplot(Loan, aes(x = IntRate, fill = Status)) +
  geom_histogram(alpha = 0.5, position = "identity", bins = 20) +
  facet_wrap(~ Loan$Status) +
  theme_minimal() +
  labs(title = "Histogram of 'IntRate' by Class")
```



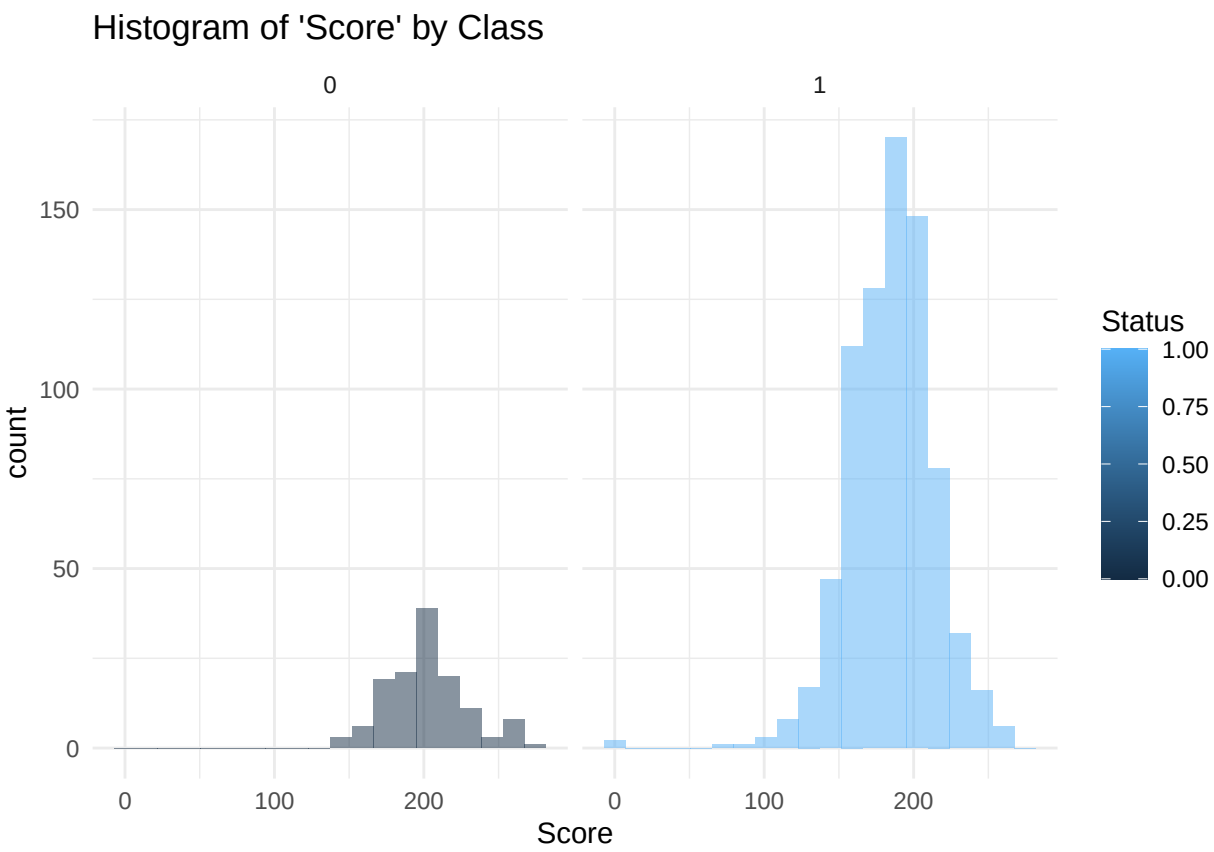

```
ggplot(Loan, aes(x = Amount, fill = Status)) +
  geom_histogram(alpha = 0.5, position = "identity", bins = 20) +
  facet_wrap(~ Loan$Status) +
  theme_minimal() +
  labs(title = "Histogram of 'Amount' by Class")
```




```
ggplot(Loan, aes(x = Income, fill = Status)) +
  geom_histogram(alpha = 0.5, position = "identity", bins = 20) +
  facet_wrap(~ Loan$Status) +
  theme_minimal() +
  labs(title = "Histogram of 'Income' by Class")
```




```
ggplot(Loan, aes(x = Score, fill = Status)) +
  geom_histogram(alpha = 0.5, position = "identity", bins = 20) +
  facet_wrap(~ Loan$Status) +
  theme_minimal() +
  labs(title = "Histogram of 'Score' by Class")
```



We can see some predictors are not normal distributed, we could tranform it, but we will leave it for now.

1) Linear Discrimant Analysis (LDA)

a)

```
library(MASS)
```

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

```
lda_model <- lda(formula = train_data$Status ~ ., data = train_data)
print(lda_model)
```

```
## Call:
## lda(train_data$Status ~ ., data = train_data)
##
## Prior probabilities of groups:
##         0         1
## 0.1419558 0.8580442
##
## Group means:
##     Amount   IntRate   EmpLenB   EmpLenC   EmpLenD    EmpLenU   HomeOWN
## 0 34.63211 0.1728789 0.1888889 0.1888889 0.3555556 0.07777778 0.1111111
```

```
## 1 34.35667 0.1483965 0.2150735 0.1525735 0.3529412 0.06250000 0.1084559
##    HomeRENT   Income
## 0 0.4444444 64689.69
## 1 0.3952206 74125.78
##
## Coefficients of linear discriminants:
##                     LD1
## Amount   -1.462031e-02
## IntRate  -2.284724e+01
## EmpLenB  -2.432756e-01
## EmpLenC  -5.567128e-01
## EmpLenD  -2.431733e-01
## EmpLenU  -3.793516e-01
## HomeOWN  -4.271189e-02
## HomeRENT -1.578914e-01
## Income    7.266007e-06
```

The group means in LDA represent the average values of each feature for each class. Greater separation between means implies that the classes are more distinct in the feature space.

The goal of LDA is to maximize the ratio of between-class variance to within-class variance for each feature, so for a feature with high variance between classes and a low variance within a class leads to a higher coefficient.

Is any data preprocessing necessary or advisable?:

LDA assumes that each feature is normally distributed within each class. Therefore, it's important to check for any major deviations from normality. From the distribution plots, we can see that some features are not normally distributed, so applying a transformation could be helpful.

It also assumes that features are not highly correlated with one another. In a previous exercise, we noticed that Score is correlated with other variables, so we removed "Score" as well as "ILR", because there both highly correlated to "IntRate". Notice: "ILR" was removed after we run in some Rank issue building the qda model.

Since LDA relies on covariance matrices, it is sensitive to the scale of the features. Features with different scales can dominate the discriminant calculations, which leads to suboptimal results. Therefore, it's advisable to standardize all features before applying LDA.

b)

```r
lda_predictions_train <- predict(lda_model, newdata = train_data)

predicted_classes_train <- lda_predictions_train$class

conf_matrix_train <- table(Predicted = predicted_classes_train, Actual = train_data$Status)

print(conf_matrix_train)
```

```
##          Actual
## Predicted   0   1
##         0   2   1
##         1  88 543
```

```r
TP <- conf_matrix_train[2, 2] # True Positives
TN <- conf_matrix_train[1, 1] # True Negatives
FP <- conf_matrix_train[2, 1] # False Positives
FN <- conf_matrix_train[1, 2] # False Negatives

misclassification_rate <- (FP + FN) / (TP + TN + FP + FN)

print(paste("Misclassification Rate:", round(misclassification_rate, 4)))
```

```
## [1] "Misclassification Rate: 0.1404"
```

```r
TPR <- TP / (TP + FN)
TNR <- TN / (TN + FP)

balanced_accuracy <- (TPR + TNR) / 2

print(paste("Balanced Accuracy:", round(balanced_accuracy, 4)))
```

```
## [1] "Balanced Accuracy: 0.5102"
```

Conclussion:

The misclassification rate of 0.1404 indicates that 14.04% of the predictions made by the model on the training data were incorrect, meaning 85.96% of the training samples were correctly classified.

However, the balanced accuracy of 0.5102 suggests that the model is only slightly better than random guessing. This is likely due to class imbalance, as indicated by the prior probabilities of 0.14 for group 0 and 0.8580442 for group 1.

To address this, we could adjust the prior probabilities in the lda() function to give more weight to the minority class. Additionally, it would be beneficial to apply the data preprocessing techniques we discussed earlier to improve the model's performance.

c)

```r
lda_predictions_test <- predict(lda_model, newdata = test_data)

predicted_classes_test <- lda_predictions_test$class

conf_matrix_test <- table(Predicted = predicted_classes_test, Actual = test_data$Status)

print(conf_matrix_test)
```

```
##          Actual
## Predicted   0   1
##         0   2   1
##         1  39 224
```

```r
TP <- conf_matrix_test[2, 2] # True Positives
TN <- conf_matrix_test[1, 1] # True Negatives
FP <- conf_matrix_test[2, 1] # False Positives
FN <- conf_matrix_test[1, 2] # False Negatives

misclassification_rate_test <- (FP + FN) / (TP + TN + FP + FN)

print(paste("Misclassification Rate on Test Data:", round(misclassification_rate_test, 4)))
```

```
## [1] "Misclassification Rate on Test Data: 0.1504"
```

```r
TPR <- TP / (TP + FN)
TNR <- TN / (TN + FP)


balanced_accuracy_test <- (TPR + TNR) / 2

print(paste("Balanced Accuracy on Test Data:", round(balanced_accuracy_test, 4)))
```

```
## [1] "Balanced Accuracy on Test Data: 0.5222"
```

Conclusion:

The misclassification rate of 0.8459 indicates that 84.59% of the predictions made by the model on the test data were incorrect, meaning that only 15.4% of the test samples were correctly classified.

Furthermore, the balanced accuracy of 0.52 suggests that the model is essentially performing at the level of random guessing. This poor performance could be a result of class imbalance, as seen previously in the training data with prior probabilities of 0.14 for group 0 and 0.85 for group 1.

2)

a) Undersampling

```r
library(ROSE)
```

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

```
## Loaded ROSE 0.0-4
```

```r
undersampling_train_data <- ovun.sample(Status ~ ., data = train_data, method = "under", N = min(table(

# Confirm the number of samples in the undersampled training set
table(undersampling_train_data$Status)
```

```
##
##  0  1
## 90 90
```

```r
lda_undersampling_model <- lda(formula = Status ~ ., data = undersampling_train_data)
lda_predictions_undersampling <- predict(lda_undersampling_model, newdata = test_data)
predicted_classes_undersampling <- lda_predictions_undersampling$class

conf_matrix <- table(Predicted = predicted_classes_undersampling, Actual = test_data$Status)

print(conf_matrix)
```

```
##           Actual
## Predicted   0   1
##         0  25  99
##         1  16 126
```

```r
TP <- conf_matrix[2, 2] # True Positives
TN <- conf_matrix[1, 1] # True Negatives
FP <- conf_matrix[2, 1] # False Positives
FN <- conf_matrix[1, 2] # False Negatives

misclassification_rate_balanced_test <- (FP + FN) / (TP + TN + FP + FN)
print(paste("Misclassification Rate on Test Data (Balanced):", round(misclassification_rate_balanced_tes
```

```
## [1] "Misclassification Rate on Test Data (Balanced): 0.4323"
```

```r
TPR <- TP / (TP + FN)
TNR <- TN / (TN + FP)

balanced_accuracy_balanced_test <- (TPR + TNR) / 2
print(paste("Balanced Accuracy on Test Data (Balanced):", round(balanced_accuracy_balanced_test, 4)))
```

```
## [1] "Balanced Accuracy on Test Data (Balanced): 0.5849"
```

Conclusion:

After creating a balanced training set using undersampling, the misclassification rate on the test data is 0.4323, meaning 43.23% of the predictions made by the model were incorrect. This is an improvement compared to the previous misclassification rate of 85% with the original unbalanced model.

The balanced accuracy has also improved from 0.5 to 0.58, indicating that the model now performs better than without samling. This improvement, although modest, shows that balancing the training set helped the model to distinguish between classes more effectively.

a) Oversampling

```r
library(ROSE)
oversampling_train_data <- ovun.sample(Status ~ ., data = train_data, method = "over", N = max(table(tra

table(oversampling_train_data$Status)
```

```
##
##   0   1
## 544 544
```

```r
lda_balanced_model <- lda(formula = Status ~ ., data = oversampling_train_data)


lda_predictions_balanced_test <- predict(lda_balanced_model, newdata = test_data)

predicted_classes_balanced_test <- lda_predictions_balanced_test$class

conf_matrix_balanced_test <- table(Predicted = predicted_classes_balanced_test, Actual = test_data$Statu
print(conf_matrix_balanced_test)
```

```
##          Actual
## Predicted   0   1
##         0  26  85
##         1  15 140
```

```r
TP <- conf_matrix_balanced_test[2, 2] # True Positives
TN <- conf_matrix_balanced_test[1, 1] # True Negatives
FP <- conf_matrix_balanced_test[2, 1] # False Positives
FN <- conf_matrix_balanced_test[1, 2] # False Negatives

misclassification_rate_balanced_test <- (FP + FN) / (TP + TN + FP + FN)
print(paste("Misclassification Rate on Test Data (Balanced with ROSE Oversampling):", round(misclassifi
```

```
## [1] "Misclassification Rate on Test Data (Balanced with ROSE Oversampling): 0.3759"
```

```r
TPR <- TP / (TP + FN)
TNR <- TN / (TN + FP)
balanced_accuracy_balanced_test <- (TPR + TNR) / 2
print(paste("Balanced Accuracy on Test Data (Balanced with ROSE Oversampling):", round(balanced_accuracy
```

```
## [1] "Balanced Accuracy on Test Data (Balanced with ROSE Oversampling): 0.6282"
```

Conclusion:

After applying oversampling using the ROSE package to balance the training set, the misclassification rate on the test data was 0.38% of the predictions were incorrect. This shows an improvement compared to the previous misclassification rate of 85% with the unbalanced model, and also compared to the 43.23% error rate after using undersampling.

The balanced accuracy has also improved from 0.5 to 0.63, indicating that the model now performs much better at distinguishing between the two classes, compared to the results obtained with undersampling. This improvement highlights that the oversampling technique helped reduce the model's bias towards the majority class, resulting in better classification of both classes.

In conclusion, oversampling with ROSE was more effective than undersampling in improving both the misclassification rate and balanced accuracy, and it also allowed the model to learn a better representation of the minority class.

Which strategy is more successful, and why?:

Oversampling with ROSE proved to be more effective than undersampling for this dataset, as shown by the higher balanced accuracy and lower misclassification rate.

Undersampling involves removing data points from the majority class, which can result in a loss of valuable information. Reducing the sample size risks eliminating important patterns within the majority class, ultimately making the model less effective.

In conclusion, oversampling is the better approach in this scenario because it maintains the balance between classes without reducing any patterns from the majority class. Given that the dataset is relatively small, oversampling is the better fit as it allows us to retain all available information. If we had a much larger dataset, undersampling could be a more viable option, as the risk of losing significant information would be lower.

3) Quadratic Discrimant Analysis

Undersampling

```
library(MASS)
qda_model_undersampled <- qda(formula = Status ~ ., data = undersampling_train_data)

qda_predictions_undersampled <- predict(qda_model_undersampled, newdata = test_data)

predicted_classes_undersampled <- qda_predictions_undersampled$class

conf_matrix_undersampled <- table(Predicted = predicted_classes_undersampled, Actual = test_data$Status)
print(conf_matrix_undersampled)
```

```
##          Actual
## Predicted   0   1
##         0  30 129
##         1  11  96
```

```
TP <- conf_matrix_undersampled[2, 2]  # True Positives
TN <- conf_matrix_undersampled[1, 1]  # True Negatives
FP <- conf_matrix_undersampled[2, 1]  # False Positives
FN <- conf_matrix_undersampled[1, 2]  # False Negatives

misclassification_rate_undersampled <- (FP + FN) / (TP + TN + FP + FN)
print(paste("Misclassification Rate on Test Data (QDA with Undersampling):", round(misclassification_ra
```

```
## [1] "Misclassification Rate on Test Data (QDA with Undersampling): 0.5263"
```

```
TPR <- TP / (TP + FN)
TNR <- TN / (TN + FP)
balanced_accuracy_undersampled <- (TPR + TNR) / 2
print(paste("Balanced Accuracy on Test Data (QDA with Undersampling):", round(balanced_accuracy_undersa
```

```
## [1] "Balanced Accuracy on Test Data (QDA with Undersampling): 0.5792"
```

Initially, the QDA model ran into a rank deficiency error because the covariance matrices were not invertible. This happened due to high correlations between features and limited data from undersampling, so we removed ILR and Score.

If features are highly correlated or there are too few samples, the covariance matrix may be singular, leading to rank deficiency errors.The presence of highly correlated features meant that the covariance matrix was not of full rank. Removing these features helped reduce collinearity, thereby mitigating the rank deficiency.

QDA with undersampling showed limited improvement.

Oversampling

```
qda_model_oversampled <- qda(formula = Status ~ ., data = oversampling_train_data)

qda_predictions_oversampled <- predict(qda_model_oversampled, newdata = test_data)


predicted_classes_oversampled <- qda_predictions_oversampled$class

conf_matrix_oversampled <- table(Predicted = predicted_classes_oversampled, Actual = test_data$Status)
print(conf_matrix_oversampled)
```

```
##          Actual
## Predicted   0   1
##         0  23  89
##         1  18 136
```

```
TP <- conf_matrix_oversampled[2, 2]  # True Positives
TN <- conf_matrix_oversampled[1, 1]  # True Negatives
FP <- conf_matrix_oversampled[2, 1]  # False Positives
FN <- conf_matrix_oversampled[1, 2]  # False Negatives

misclassification_rate_oversampled <- (FP + FN) / (TP + TN + FP + FN)
print(paste("Misclassification Rate on Test Data (QDA with ROSE Oversampling):", round(misclassificatio
```

```
## [1] "Misclassification Rate on Test Data (QDA with ROSE Oversampling): 0.4023"
```

```
TPR <- TP / (TP + FN)
TNR <- TN / (TN + FP)
balanced_accuracy_oversampled <- (TPR + TNR) / 2
print(paste("Balanced Accuracy on Test Data (QDA with ROSE Oversampling):", round(balanced_accuracy_ove
```

```
## [1] "Balanced Accuracy on Test Data (QDA with ROSE Oversampling): 0.5827"
```

We had same issue here with the rank deficiency error. QDA with oversampling provided a tiny improvement over other approaches, but the model still struggled to effectively classify the minority class.

4) Regularized Discrimant Analysis

Undersampling

```r
library(klaR)
```

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

```r
rda_model_undersampled <- rda(Status ~ ., data = undersampling_train_data, gamma = 0.1, lambda = 0.1)

rda_predictions_undersampled <- predict(rda_model_undersampled, newdata = test_data)

predicted_classes_undersampled <- rda_predictions_undersampled$class

conf_matrix_undersampled <- table(Predicted = predicted_classes_undersampled, Actual = test_data$Status)
print(conf_matrix_undersampled)
```

```
##          Actual
## Predicted   0   1
##         0  41 217
##         1   0   8
```

```r
TP <- conf_matrix_undersampled[2, 2]
TN <- conf_matrix_undersampled[1, 1]
FP <- conf_matrix_undersampled[2, 1]
FN <- conf_matrix_undersampled[1, 2]

misclassification_rate_undersampled <- (FP + FN) / (TP + TN + FP + FN)
print(paste("Misclassification Rate on Test Data (RDA with Undersampling):", round(misclassification_ra
```

```
## [1] "Misclassification Rate on Test Data (RDA with Undersampling): 0.8158"
```

```r
TPR <- TP / (TP + FN)
TNR <- TN / (TN + FP)
balanced_accuracy_undersampled <- (TPR + TNR) / 2
print(paste("Balanced Accuracy on Test Data (RDA with Undersampling):", round(balanced_accuracy_undersa
```

```
## [1] "Balanced Accuracy on Test Data (RDA with Undersampling): 0.5178"
```

RDA with undersampling showed minimal improvement, with the model still struggling to classify the minority class effectively due to the high misclassification rate and limited sample size.

Oversampling

```r
rda_model_oversampled <- rda(Status ~ ., data = oversampling_train_data, gamma = 0.1, lambda = 0.1)

rda_predictions_oversampled <- predict(rda_model_oversampled, newdata = test_data)

predicted_classes_oversampled <- rda_predictions_oversampled$class

conf_matrix_oversampled <- table(Predicted = predicted_classes_oversampled, Actual = test_data$Status)
print(conf_matrix_oversampled)
```

```
##          Actual
## Predicted   0   1
##         0  40 216
##         1   1   9
```

```r
TP <- conf_matrix_oversampled[2, 2]  # True Positives
TN <- conf_matrix_oversampled[1, 1]  # True Negatives
FP <- conf_matrix_oversampled[2, 1]  # False Positives
FN <- conf_matrix_oversampled[1, 2]  # False Negatives

misclassification_rate_oversampled <- (FP + FN) / (TP + TN + FP + FN)
print(paste("Misclassification Rate on Test Data (RDA with Oversampling):", round(misclassification_rat
```

```
## [1] "Misclassification Rate on Test Data (RDA with Oversampling): 0.8158"
```

```r
TPR <- TP / (TP + FN)
TNR <- TN / (TN + FP)
balanced_accuracy_oversampled <- (TPR + TNR) / 2
print(paste("Balanced Accuracy on Test Data (RDA with Oversampling):", round(balanced_accuracy_oversampl
```

```
## [1] "Balanced Accuracy on Test Data (RDA with Oversampling): 0.5078"
```

RDA with oversampling did not provide much improvement, resulting in a high misclassification rate of 81.58% and a balanced accuracy of 0.5078. The model struggled significantly, particularly in correctly identifying the minority class, indicating that oversampling alone was not sufficient to improve performance.

Gamma and Lambda in RDA:

In Regularized Discriminant Analysis (RDA), the parameters gamma and lambda help to control how the model calculates the covariance matrices, which are used to classify the data.

- Gamma adjusts how much the model shares the covariance information between different classes.
- When gamma is 0, the model behaves like QDA, using separate covariance matrices for each class, which gives the model more flexibility
- When gamma is 1, the model behaves like LDA, using a single pooled covariance matrix for all classes, which makes it more stable and less likely to overfit.
- Lambda adjusts how much we shrink the covariance matrix towards a simpler form.
- When lambda is 0, there is no shrinkage, and the regular covariance matrix is used.
- When lambda is 1, the covariance matrix is shrunk to an identity matrix, meaning it ignores relationships between features, treating them as independent.

We can think of lambda in RDA similarly to how it's used in Ridge Regression—it acts like a penalty value that controls the extent of shrinkage. RDA in classification is like a balance between LDA and QDA, much like how adaptive lasso represents a middle ground between ridge and lasso regression for regression tasks.