Exercise 5

Advanced Methods for Regression and Classification

Rita Selimi

19/11/2024

Predict Status using Leastsquares Regression

Load the necessary libraries

Load Data

```r
# Load the Loan dataset
data("Loan", package = "ROCit")

# View the dataset's structure
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 : Factor w/ 2 levels "CO","FP": 1 1 2 2 1 2 2 2 2 2 ...
##  $ 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     CO 200.9581
## 2  22.97   36  0.1198 0.032      D     RENT  30900     CO 179.6058
## 3  54.05   36  0.1166 0.032      D MORTGAGE 111900     FP 161.7622
## 4  24.32   36  0.1733 0.034      A     RENT  66000     FP 196.6619
## 5  43.24   36  0.1723 0.034      A MORTGAGE  71900     CO 203.4912
## 6  16.22   36  0.1355 0.033      B     RENT  27614     FP 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   CO:131   Min.   : -5.449
##  OWN     : 95   1st Qu.: 43900   FP:769   1st Qu.:169.358
##  RENT    :376   Median : 63000            Median :189.120
##                 Mean   : 72903            Mean   :187.440
##                 3rd Qu.: 86900            3rd Qu.:205.064
##                 Max.   :502000            Max.   :269.171
```

1. **`Amount`**: The loan amount in thousands
2. **`Term`**: The term of the loan in months
3. **`IntRate`**: The interest rate as a decimal
4. **`ILR`**: An index value representing the individual's loan rate
5. **`EmpLen`**: Employment length categorized into 5 levels
6. **`Home`**: Home ownership status categorized into 3 levels
7. **`Income`**: The individual's income in dollars
8. **`Status`**: Loan repayment status with two levels
9. **`Score`**: A score associated with the individual's loan profile

1. Data Splitting and lm()

Data Preprocessing **Converting `Status` to Numerical** The `Status` variable is a factor, which we need to convert to a binary numerical format (0 for "CO" and 1 for "FP").

```
# Check levels of the Status variable
levels(Loan$Status)
```

```
## [1] "CO" "FP"
```

```
# Recode factor levels explicitly (if 'CO' should be 0 and 'FP' should be 1)
Loan$Status <- as.numeric(factor(Loan$Status, levels = c("CO", "FP"))) - 1

# Check the resulting values
table(Loan$Status)
```

```
##
##   0   1
## 131 769
```

Handling Categorical Variables For this assignment, we do not need to manually convert categorical variables into dummy variables for the `lm()` function, because lm() automatically handles factor variables. Instead, you can directly use the dataset with factors, and `lm()` will create the necessary dummy variables behind the scenes.

Check for Missing Values

```r
# Check for missing values
sum(is.na(Loan))
```

```
## [1] 0
```

```r
# Remove rows with missing values
Loan <- na.omit(Loan)

# Check for zero or negative values in Income
sum(Loan$Income <= 0)  # Should return 0 for valid data
```

```
## [1] 0
```

Scale Numerical Variables Scale numerical predictors to standardize their ranges for interpretability and stability.

```r
# Identify numerical columns
num_cols <- c("Amount", "Income", "IntRate", "ILR", "Score")

# Standardize the numerical columns
Loan[, num_cols] <- scale(Loan[, num_cols])

# Confirm scaling
summary(Loan[, num_cols])
```

```
##      Amount            Income           IntRate              ILR
##  Min.   :-1.4501   Min.   :-1.3595   Min.   :-1.74047   Min.   :-1.8857
##  1st Qu.:-0.7411   1st Qu.:-0.6463   1st Qu.:-0.77234   1st Qu.:-0.8167
##  Median :-0.3257   Median :-0.2207   Median :-0.04065   Median :-0.2821
##  Mean   : 0.0000   Mean   : 0.0000   Mean   : 0.00000   Mean   : 0.0000
##  3rd Qu.: 0.4282   3rd Qu.: 0.3119   3rd Qu.: 0.61140   3rd Qu.: 0.7870
##  Max.   : 2.7966   Max.   : 9.5626   Max.   : 3.22460   Max.   : 3.4597
##      Score
##  Min.   :-6.58864
##  1st Qu.:-0.61764
##  Median : 0.05737
##  Mean   : 0.00000
##  3rd Qu.: 0.60199
##  Max.   : 2.79173
```

```r
max_outlier <- Loan[which(scale(Loan$Income) > 9), "Income"]
max_outlier  # Check the exact value
```

```
## [1] 9.562607 9.562607
```

Winsorize the Outlier Winsorizing (capping) limits the impact of the outlier while retaining the observation, making it a good middle-ground option.

```r
# Winsorize Income at 3 standard deviations
Loan$Income <- ifelse(Loan$Income > 3, 3, Loan$Income)

summary(Loan[, c("Amount", "IntRate", "ILR", "Score")])
```

```
##      Amount           IntRate              ILR              Score
##  Min.   :-1.4501   Min.   :-1.74047   Min.   :-1.8857   Min.   :-6.58864
##  1st Qu.:-0.7411   1st Qu.:-0.77234   1st Qu.:-0.8167   1st Qu.:-0.61764
##  Median :-0.3257   Median :-0.04065   Median :-0.2821   Median : 0.05737
##  Mean   : 0.0000   Mean   : 0.00000   Mean   : 0.0000   Mean   : 0.00000
##  3rd Qu.: 0.4282   3rd Qu.: 0.61140   3rd Qu.: 0.7870   3rd Qu.: 0.60199
##  Max.   : 2.7966   Max.   : 3.22460   Max.   : 3.4597   Max.   : 2.79173
```

Splitting the Data We randomly split the dataset into training (2/3) and test (1/3) sets for model development and evaluation.

```
# Set a seed for reproducibility
set.seed(12332281)

# Split data into training and test sets
train_indices <- sample(1:nrow(Loan), size = 2/3 * nrow(Loan))
train_data <- Loan[train_indices, ]
test_data <- Loan[-train_indices, ]
```

Building the Linear Model Using the training dataset, we fit a linear regression model with `lm()`.

```
# Build the linear regression model
lm_model <- lm(Status ~ ., data = train_data)
```

2. Outcome

```
summary(lm_model)
```

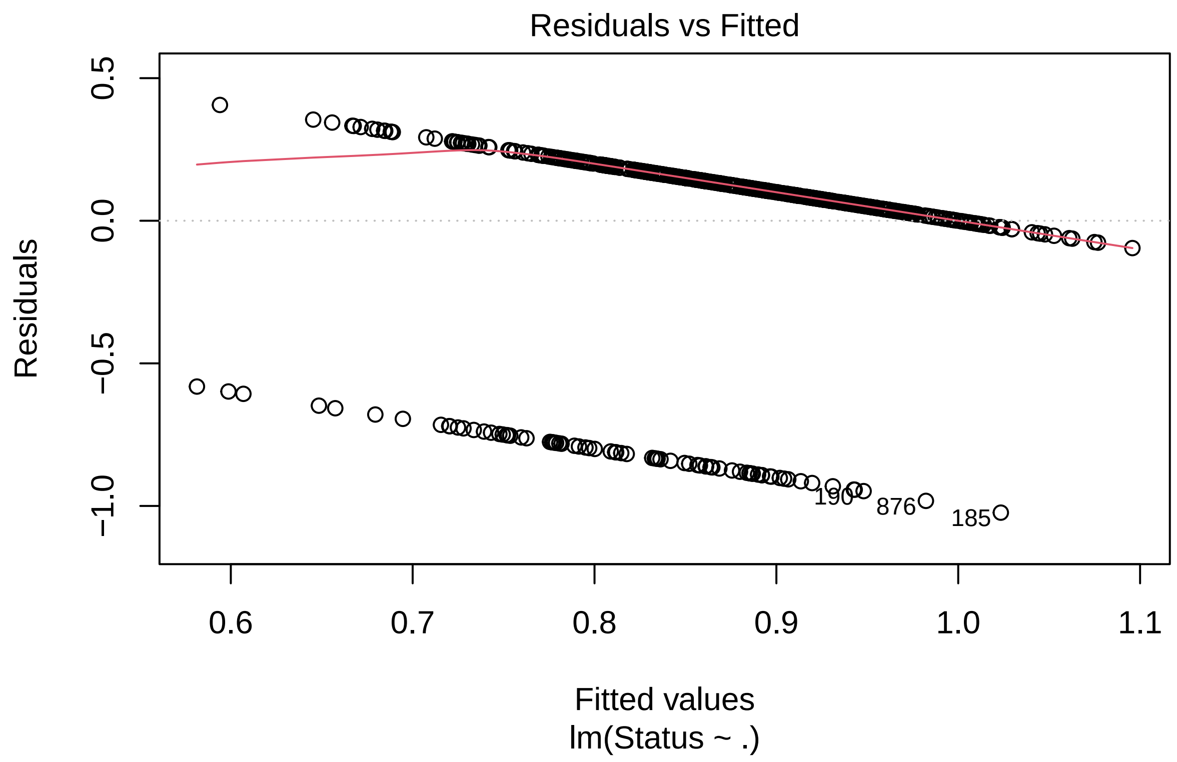
```
##
## Call:
## lm(formula = Status ~ ., data = train_data)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -1.02343  0.03263  0.10494  0.16497  0.40598
##
## Coefficients: (1 not defined because of singularities)
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept)  0.870929   0.034359  25.348   <2e-16 ***
## Amount       0.013020   0.067539   0.193    0.847
## Term               NA         NA      NA       NA
## IntRate     -0.124499   0.161562  -0.771    0.441
## ILR          0.075141   0.087228   0.861    0.389
## EmpLenB     -0.029673   0.040487  -0.733    0.464
## EmpLenC     -0.038431   0.043774  -0.878    0.380
## EmpLenD      0.032962   0.037611   0.876    0.381
## EmpLenU     -0.083565   0.060040  -1.392    0.165
## HomeOWN      0.026259   0.046535   0.564    0.573
## HomeRENT     0.015271   0.030530   0.500    0.617
## Income       0.006091   0.112197   0.054    0.957
## Score       -0.025783   0.170776  -0.151    0.880
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.3267 on 588 degrees of freedom
## Multiple R-squared:  0.06479,    Adjusted R-squared:  0.0473
## F-statistic: 3.703 on 11 and 588 DF,  p-value: 4.044e-05
```

- The low R-squared value (~6.5%) indicates that the predictors explain only a small portion of the variability in `Status`. This suggests that the model does not adequately capture the factors influencing `Status`.
- None of the individual predictors are statistically significant (`p > 0.05`), meaning there is little evidence that any variable has a strong linear relationship with `Status`.
- The variable `Term` was excluded due to singularities, likely caused by multicollinearity or redundancy with other predictors.
- The residuals show a reasonable distribution, but the high residual standard error (0.3267) and lack of significant predictors suggest that the model struggles to accurately predict `Status`.
- The significant F-statistic (p = 4.044e-05) indicates the model as a whole explains some variance, but its practical utility is limited due to the low explanatory power and weak predictor significance.

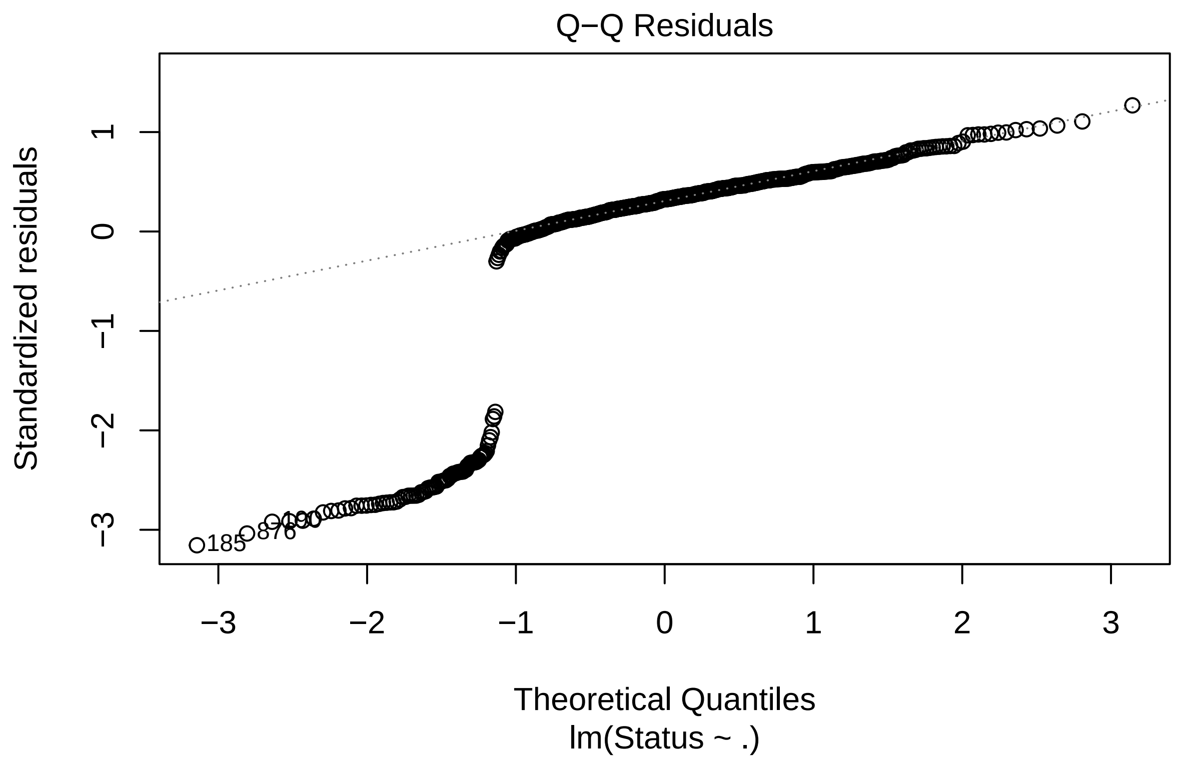
Overall Conclusion The linear regression model is not well-suited for predicting `Status`, as it assumes a linear relationship between predictors and the response, which may not hold in this case. Additionally, imbalanced group sizes further challenge the model's performance, as simple metrics like accuracy can be misleading. The predictors show weak relationships with the response, and issues like multicollinearity or non-linearity may also hinder performance.

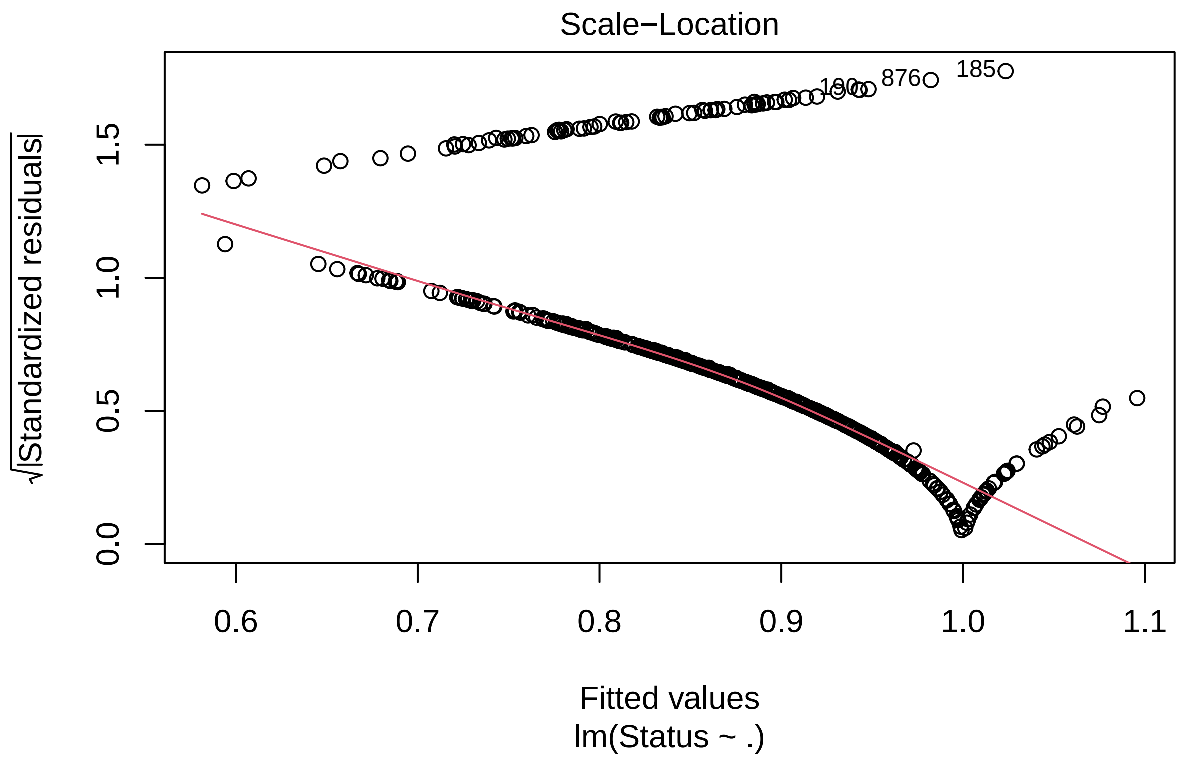
3. Plot()

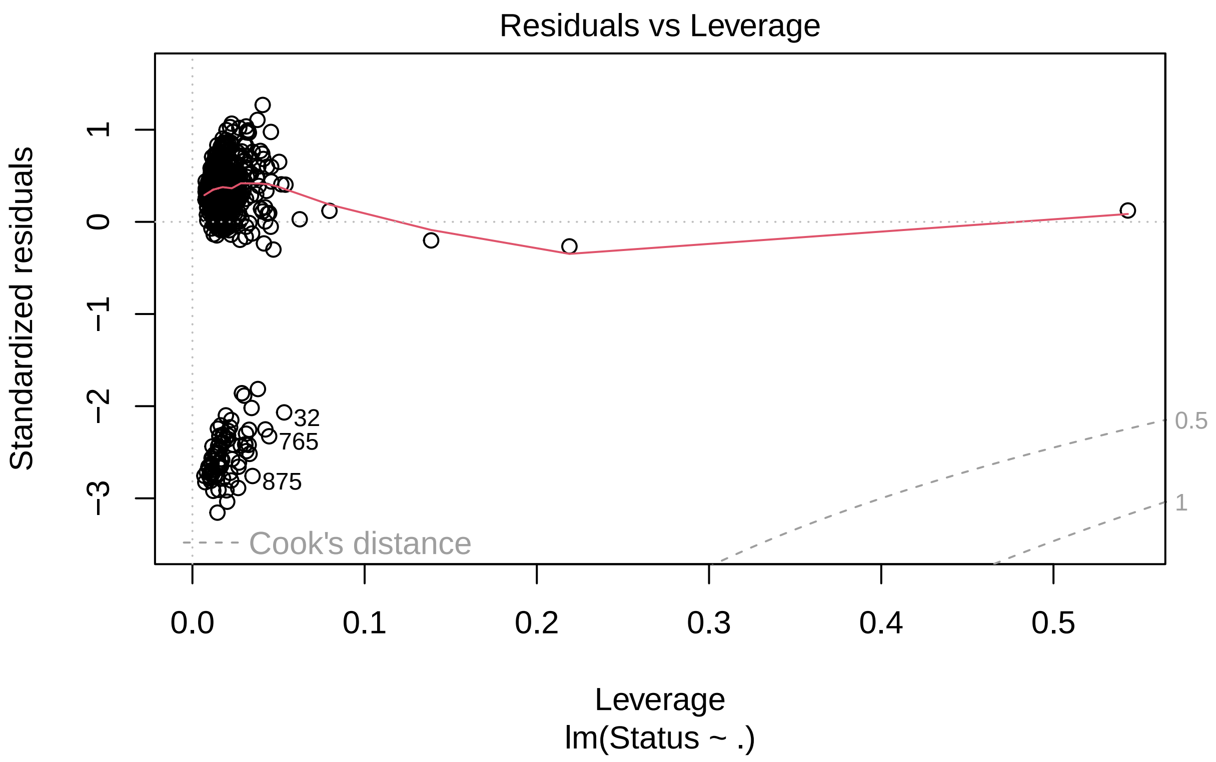
```
plot(lm_model)
```



Residuals vs Fitted
Residuals
0.5
0.0
−0.5
−1.0
0.6
0.7
0.8
0.9
1.0
1.1
190
876
185
Fitted values
lm(Status ~ .)


Q−Q Residuals
Standardized residuals
Theoretical Quantiles
lm(Status ~ .)
185
876
190
−3
−2
−1
0
1
2
3


Scale−Location
√|Standardized residuals|
0.0
0.5
1.0
1.5
0.6
0.7
0.8
0.9
1.0
1.1
190
876
185
Fitted values
lm(Status ~ .)

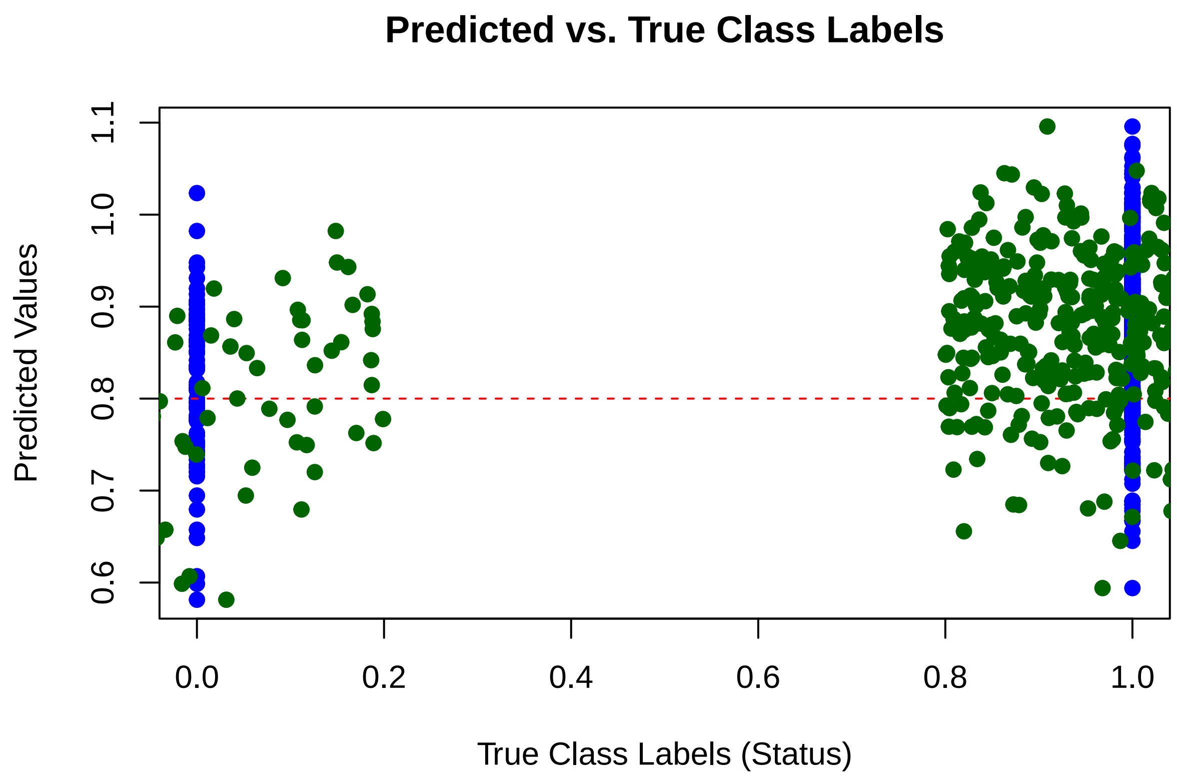


1. Residuals vs. Fitted Plot: The residuals are not randomly scattered around `0`. There is a visible pattern, especially a downward curve, which indicates a **non-linear relationship** between predictors and the response variable. A few observations, such as `190`, `876`, and `185`, stand out as potential **outliers** or leverage points. **2. Normal Q-Q Plot**: The points deviate significantly from the diagonal line, especially in the lower tail. This indicates that the **residuals are not normally distributed**. **3. Scale-Location Plot**: The spread of residuals is not constant. The plot shows a **clear curvature** and an increasing trend, which suggests **heteroscedasticity** (non-constant variance of residuals). Heteroscedasticity can lead to inefficient estimates of coefficients. **4. Residuals vs. Leverage Plot**: Points `32`, `765`, and `875` are potential **high-leverage points**. The points near the Cook's distance lines may have a disproportionate impact.

Overall Concerns The diagnostic plots reveal several concerns with the linear regression model. The pattern in the residuals indicates that the model is not capturing the relationship well, suggesting a violation of the linearity assumption. Additionally, the residuals deviate significantly from normality, which affects the reliability of statistical inference, such as p-values and confidence intervals. The presence of heteroscedasticity, where the variance of residuals is not constant, reduces the efficiency of the coefficient estimates. Furthermore, a few high-leverage points appear to have a disproportionate influence on the model, potentially skewing the results.

4. predict()

```r
# Predict the response for the training set
train_predictions <- predict(lm_model, newdata = train_data)

# Visualize predictions vs. true class labels
plot(
  train_data$Status,         # True class labels on the x-axis
  train_predictions,         # Predicted values on the y-axis
  xlab = "True Class Labels (Status)",
  ylab = "Predicted Values",
  main = "Predicted vs. True Class Labels",
  col = "blue",
  pch = 19
)

# Add a horizontal line at cutoff 0.9
abline(h = 0.8, col = "red", lty = 2)

# Add jitter to true class labels to make the scatter clearer
points(jitter(train_data$Status), train_predictions, col = "darkgreen", pch = 19)
```



The scatterplot reveals the predicted values against the true class labels (`Status`), highlighting the model's performance. Predictions for `Status = 0` cluster around the range of 0.6–0.9, rather than closer to 0, indicating the model struggles to confidently predict this class. Predictions for `Status = 1` are more concentrated near 0.8–1.1, suggesting slightly better alignment with the desired value of 1. This pattern reflects the linear model's limitations in handling binary classification, especially with potential class imbalance and insufficient separation in the data.

Evaluate Performance Across Different Cutoff Values

```r
# Define a sequence of cutoff values
cutoffs <- seq(0, 1, by = 0.01)
```

```r
# Initialize vectors to store TPR, FPR, and Balanced Accuracy
tpr <- fpr <- balanced_accuracy <- numeric(length(cutoffs))

# Loop through each cutoff and calculate metrics
for (i in seq_along(cutoffs)) {
    predicted_class <- ifelse(train_predictions > cutoffs[i], 1, 0)

    # Confusion matrix components
    tp <- sum(predicted_class == 1 & train_data$Status == 1)
    tn <- sum(predicted_class == 0 & train_data$Status == 0)
    fp <- sum(predicted_class == 1 & train_data$Status == 0)
    fn <- sum(predicted_class == 0 & train_data$Status == 1)

    # Calculate TPR, FPR, and Balanced Accuracy
    tpr[i] <- tp/(tp + fn)
    fpr[i] <- fp/(fp + tn)
    balanced_accuracy[i] <- (tpr[i] + (1 - fpr[i]))/2
}

# Identify the best cutoff value
best_cutoff <- cutoffs[which.max(balanced_accuracy)]
best_cutoff
```

```
## [1] 0.82
```

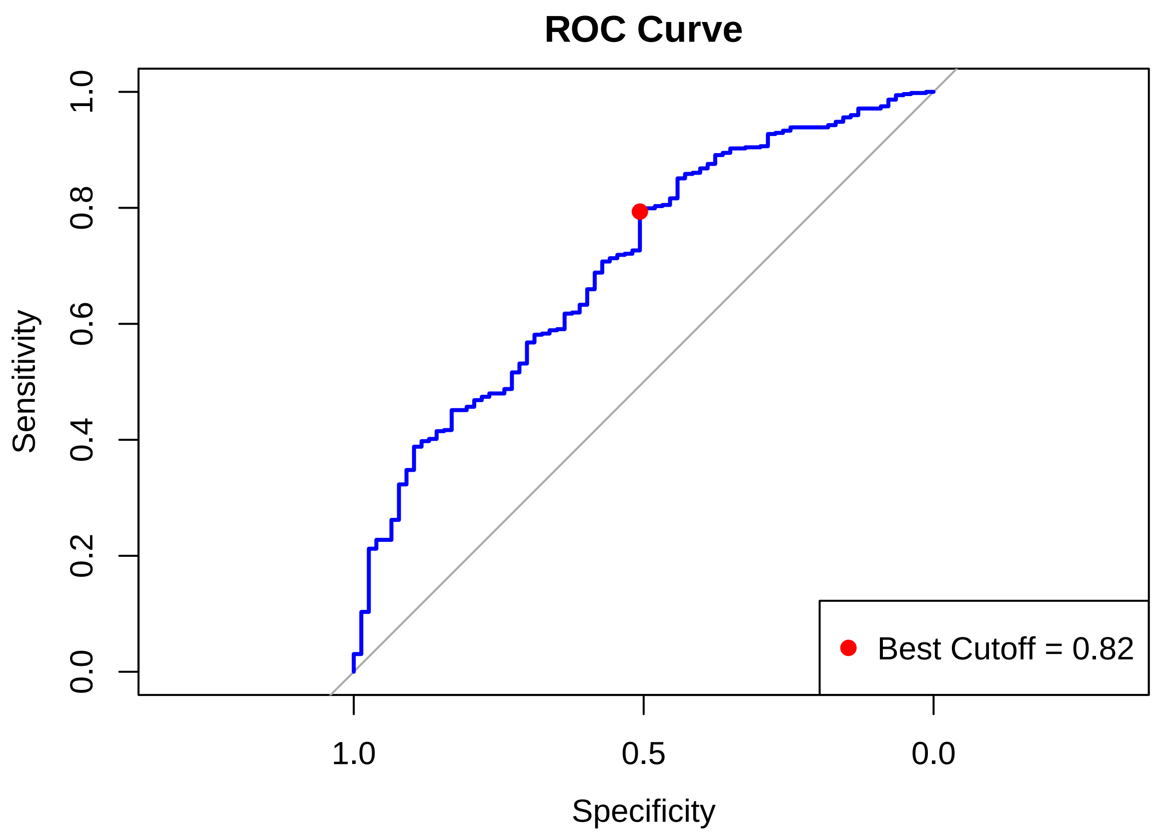
Visualize the ROC Curve

```r
# Plot the ROC Curve
roc_curve <- roc(train_data$Status, train_predictions)
```

```
## Setting levels: control = 0, case = 1
```

```
## Setting direction: controls < cases
```

```r
plot(roc_curve, main = "ROC Curve", col = "blue", lwd = 2)

# Highlight the best cutoff on the ROC curve
points(1 - fpr[which.max(balanced_accuracy)], tpr[which.max(balanced_accuracy)],
    col = "red", pch = 19)
legend("bottomright", legend = paste("Best Cutoff =", round(best_cutoff, 2)), col = "red",
    pch = 19)
```



The ROC curve illustrates the trade-off between sensitivity (true positive rate) and specificity (true negative rate) for the binary classification model at various cutoff values. The point marked in red corresponds to the **best cutoff value of 0.82**, which optimally balances sensitivity and specificity based on the given dataset. This value is chosen as it represents the point closest to the top-left corner of the ROC plot, achieving a desirable trade-off where false positives are minimized while true positives are maximized. This suggests that the model performs best at this threshold for distinguishing between the two classes.

5. Confusion Matrix

```r
# Use the best cutoff value (e.g., 0.82)
best_cutoff <- 0.82

# Predict class labels based on the best cutoff
train_predicted <- ifelse(train_predictions > best_cutoff, 1, 0)

# Create the confusion matrix
confusion_matrix <- table(Actual = train_data$Status, Predicted = train_predicted)
print(confusion_matrix)
```

```
##       Predicted
## Actual   0   1
##      0  39  38
##      1 108 415
```

```r
# Calculate metrics
tp <- confusion_matrix[2, 2]  # True Positives
tn <- confusion_matrix[1, 1]  # True Negatives
fp <- confusion_matrix[1, 2]  # False Positives
fn <- confusion_matrix[2, 1]  # False Negatives

tpr <- tp/(tp + fn)  # Sensitivity / Recall
tnr <- tn/(tn + fp)  # Specificity
balanced_accuracy <- (tpr + tnr)/2

cat("Sensitivity (TPR):", tpr, "\n")
```

```
## Sensitivity (TPR): 0.793499
```

```r
cat("Specificity (TNR):", tnr, "\n")
```

```
## Specificity (TNR): 0.5064935
```

```r
cat("Balanced Accuracy:", balanced_accuracy, "\n")
```

```
## Balanced Accuracy: 0.6499963
```

The confusion matrix and metrics reveal that the model performs better at identifying positive cases (`1`) than negative cases (`0`). Specifically, it achieves a **sensitivity of 79.3%**, meaning it correctly identifies 79.3% of actual positives. However, the **specificity is much lower at 50.6%**, indicating the model struggles to correctly identify negative cases, leading to a higher number of false positives.

The **balanced accuracy of 65.0%** shows that the model has moderate overall performance when accounting for both sensitivity and specificity. While it is effective at detecting positive cases, the high number of false negatives (108) and false positives (38) suggests that the predictions for the negative class are less reliable.

The imbalance in the dataset, with significantly more positive cases (`1`) than negatives (`0`), likely contributes to the model's bias toward predicting positives. To improve performance, techniques such as addressing class imbalance or using more advanced models like logistic regression or ensemble methods could be explored. Adjusting the cutoff value could also help achieve a better balance between sensitivity and specificity, depending on the specific requirements of the problem.

6. ROC Curve Evaluation

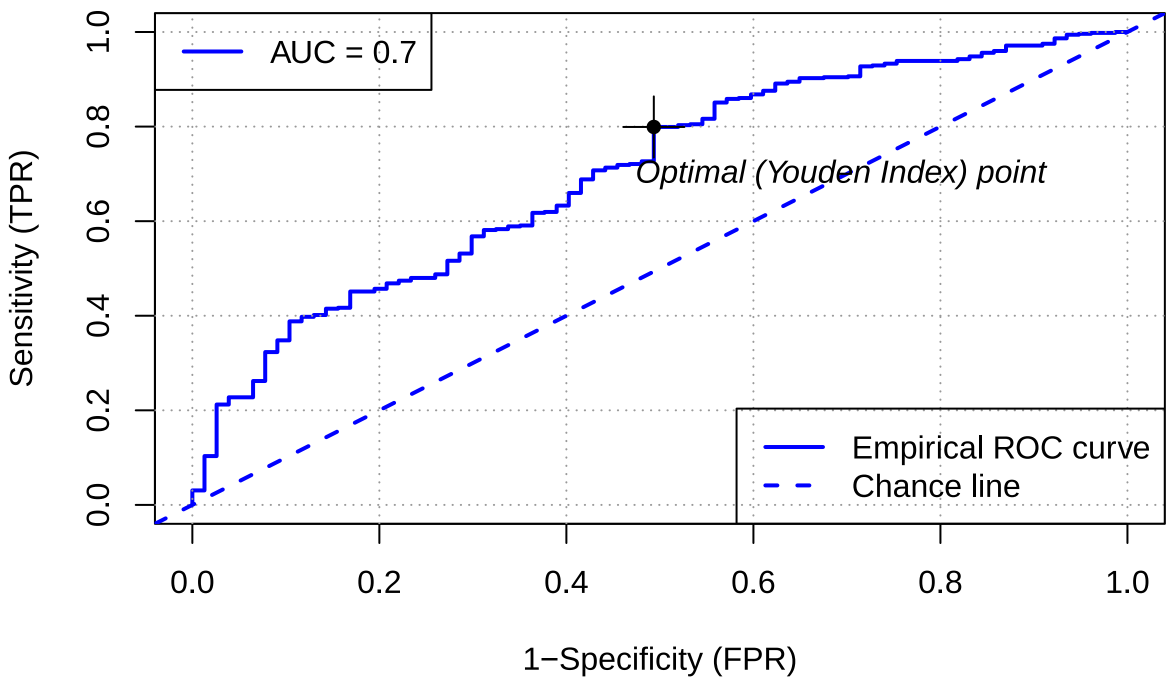
Generate ROC Curve and Evaluate

```r
# Apply rocit() function using training predictions and actual labels
roc_result <- rocit(score = train_predictions, class = train_data$Status)

# Display summary of the ROC result
summary(roc_result)
```

```
##
##  Method used: empirical
##  Number of positive(s): 523
##  Number of negative(s): 77
##  Area under curve: 0.7021
```

Visualize ROC Curve

```r
# Plot the ROC curve
plot(roc_result, main = "ROC Curve for Model", col = "blue", lwd = 2)

# Add AUC legend to the plot
legend("topleft", legend = paste("AUC =", round(roc_result$AUC, 2)), col = "blue",
    lwd = 2)
```



Which value would indicate the quality of your classifier?

The **Area Under the Curve (AUC)** shows how well the classifier works. Here, the AUC is **0.7021**, meaning the model is better than random guessing (**AUC = 0.5**) but far from perfect (**AUC = 1.0**). An AUC of **0.7** suggests the classifier can distinguish between positive and negative cases most of the time, but it's not very strong.

Is the classifier doing a good job?

The classifier is doing a **fair job** but not a great one. With an AUC of **0.7021**, it shows moderate effectiveness in predicting the correct class. However, it struggles with some misclassifications, likely due to imbalanced data (523 positives vs. 77 negatives). Improving the model with techniques like balancing the dataset or using better algorithms could make it perform better.

7. Compute Balanced Accuracy and Find Optimal Cutoff

Calculate Balanced Accuracy

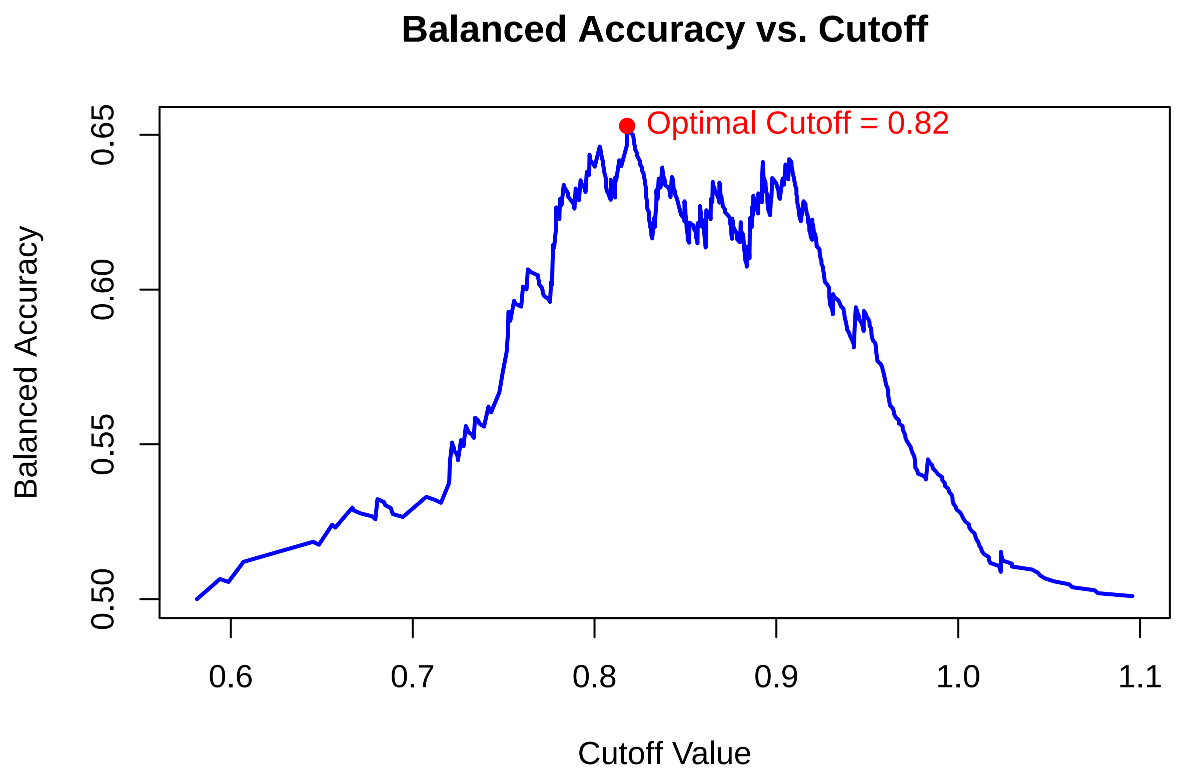
```r
# Generate measureit object
measure_result <- measureit(
  score = train_predictions,  # Predicted probabilities from the model
  class = train_data$Status,  # Actual class labels
  measure = c("TPR", "TNR"),  # Sensitivity (TPR) and Specificity (TNR)
  cutoff = seq(0, 1, by = 0.01)  # Sequence of cutoff values
)

# Extract Balanced Accuracy (TPR + TNR) / 2
balanced_accuracy <- (measure_result$TPR + measure_result$TNR) / 2

# Find the optimal cutoff value
optimal_index <- which.max(balanced_accuracy)
optimal_cutoff <- measure_result$Cutoff[optimal_index]
optimal_cutoff
```

```
## [1] 0.8179281
```

Balanced accuracy is particularly useful when dealing with imbalanced data because it considers both sensitivity (true positive rate) and specificity (true negative rate). This ensures that both classes are evaluated fairly, avoiding bias toward the majority class.

Plot Balanced Accuracy vs. Cutoff

```r
# Plot Balanced Accuracy vs. Cutoff
plot(measure_result$Cutoff, balanced_accuracy, type = "l", col = "blue", lwd = 2,
    xlab = "Cutoff Value", ylab = "Balanced Accuracy", main = "Balanced Accuracy vs. Cutoff")

# Highlight the optimal cutoff
points(optimal_cutoff, balanced_accuracy[optimal_index], col = "red", pch = 19)
text(optimal_cutoff, balanced_accuracy[optimal_index], labels = paste("Optimal Cutoff =",
    round(optimal_cutoff, 2)), pos = 4, col = "red")
```



The balanced accuracy curve highlights the trade-off between sensitivity and specificity across cutoff values. As expected, the balanced accuracy is lowest at extreme cutoffs (0 and 1), where predictions favor only one class. The curve peaks at the optimal cutoff of 0.82, which balances sensitivity and specificity effectively.

A cutoff of 0.82 indicates that predicted probabilities greater than this value should be classified as '1' (positive), while those below are classified as '0' (negative). This threshold ensures that both classes are treated fairly in predictions, making it a reasonable choice for this classification problem. However, the suitability of this cutoff may depend on the broader context and limitations of the model.

8. Confusion Matrix on Test Data

```r
# Predict on the test set using the optimal cutoff
optimal_cutoff <- 0.82
test_predictions <- predict(lm_model, newdata = test_data)
test_predicted_classes <- ifelse(test_predictions > optimal_cutoff, 1, 0)

# Create the confusion matrix
confusion_matrix <- table(Actual = test_data$Status, Predicted = test_predicted_classes)

# Display the confusion matrix
print(confusion_matrix)
```

```
##       Predicted
## Actual  0  1
##      0 17 37
##      1  56 190
```

```r
# Calculate performance metrics
tp <- confusion_matrix[2, 2]
tn <- confusion_matrix[1, 1]
fp <- confusion_matrix[1, 2]
fn <- confusion_matrix[2, 1]

# Sensitivity (TPR)
sensitivity <- tp/(tp + fn)

# Specificity (TNR)
specificity <- tn/(tn + fp)

# Balanced Accuracy
balanced_accuracy <- (sensitivity + specificity)/2

# Print the metrics
cat("Sensitivity (TPR):", sensitivity, "\n")
```

```
## Sensitivity (TPR): 0.7723577
```

```r
cat("Specificity (TNR):", specificity, "\n")
```

```
## Specificity (TNR): 0.3148148
```

```r
cat("Balanced Accuracy:", balanced_accuracy, "\n")
```

```
## Balanced Accuracy: 0.5435863
```

The final confusion matrix and performance metrics show that the linear regression model, with the chosen optimal cutoff (0.82), performs moderately well in terms of sensitivity (77.2%), indicating that it correctly identifies most positive cases (Status = 1). However, the specificity (31.5%) is low, meaning the model struggles to correctly identify negative cases (Status = 0). The balanced accuracy of 54.4% reflects that the model's overall performance is only slightly better than random classification.

This result highlights the limitations of using linear regression for binary classification, especially in the presence of imbalanced data and weak predictor relationships. For better results, alternative classification techniques, such as logistic regression or ensemble methods (e.g., random forests), should be explored. Additionally, addressing class imbalance through techniques like oversampling or weighting may improve the model's ability to generalize across both classes.