HW7-Q10.3

2025-02-26

Question 10.3:

1. Using the GermanCredit data set germancredit.txt from http://archive.ics.uci.edu/ml/machine-learning-databases/statlog/german / (description at http://archive.ics.uci.edu/ml/datasets/Statlog+%28German+Credit+Data%29), use logistic regression to find a good predictive model for whether credit applicants are good credit risks or not. Show your model (factors used and their coefficients), the software output, and the quality of fit. You can use the glm function in R. To get a logistic regression (logit) model on data where the4 response is either zero or one, use family=binomial(link="logit") in your glm function call.
2. Because the model gives a result between 0 and 1, it requires setting a threshold probability to separate between "good" and "bad" answers. In this data set, they estimate that incorrectly identifying a bad customer as good, is 5 times worse than incorrectly classifying a good customer as bad. Determine a good threshold probability based on your model.

Code referenced from:

1. Office hour on Feb 24th.
2. ChatGPT, Google Gemini.
3. Site: https://www.statology.org/logistic-regression-in-r/, https://stackoverflow.com/questions/61568713/calculate-accuracy-of-model-created-with-logistic-regression.

Answer:

Load the data into R:

```
rm(list=ls())
set.seed(33)
german_credit <- read.table("germancredit.txt")
```

Make the response variable binary in terms of 0 and 1:

```
german_credit$V21[german_credit$V21==1] <- 0
german_credit$V21[german_credit$V21==2] <- 1
head(german_credit)
```

```
##    V1 V2  V3  V4   V5  V6  V7 V8  V9  V10 V11  V12 V13  V14  V15 V16  V17 V18
## 1 A11  6 A34 A43 1169 A65 A75  4 A93 A101   4 A121  67 A143 A152   2 A173   1
## 2 A12 48 A32 A43 5951 A61 A73  2 A92 A101   2 A121  22 A143 A152   1 A173   1
```

```
## 3 A14 12 A34 A46 2096 A61 A74  2 A93 A101   3 A121  49 A143 A152   1 A172   2
## 4 A11 42 A32 A42 7882 A61 A74  2 A93 A103   4 A122  45 A143 A153   1 A173   2
## 5 A11 24 A33 A40 4870 A61 A73  3 A93 A101   4 A124  53 A143 A153   2 A173   2
## 6 A14 36 A32 A46 9055 A65 A73  2 A93 A101   4 A124  35 A143 A153   1 A172   2
##    V19  V20 V21
## 1 A192 A201   0
## 2 A191 A201   1
## 3 A191 A201   0
## 4 A191 A201   0
## 5 A191 A201   1
## 6 A192 A201   0
```

Split the data into training set and test set:

```r
train_germancredit <- german_credit[1:800,]
test_germancredit <- german_credit[801:1000,]
```

Create the logistic regression model:

```r
model <- glm(V21~.,
             family = binomial(link="logit"),
             data = train_germancredit)
```

Run the stepAIC to find the best regression model with lowest AIC:

```r
library(MASS)
best_model <- stepAIC(model, direction="both", trace=FALSE)
```

Show the model's coefficients:

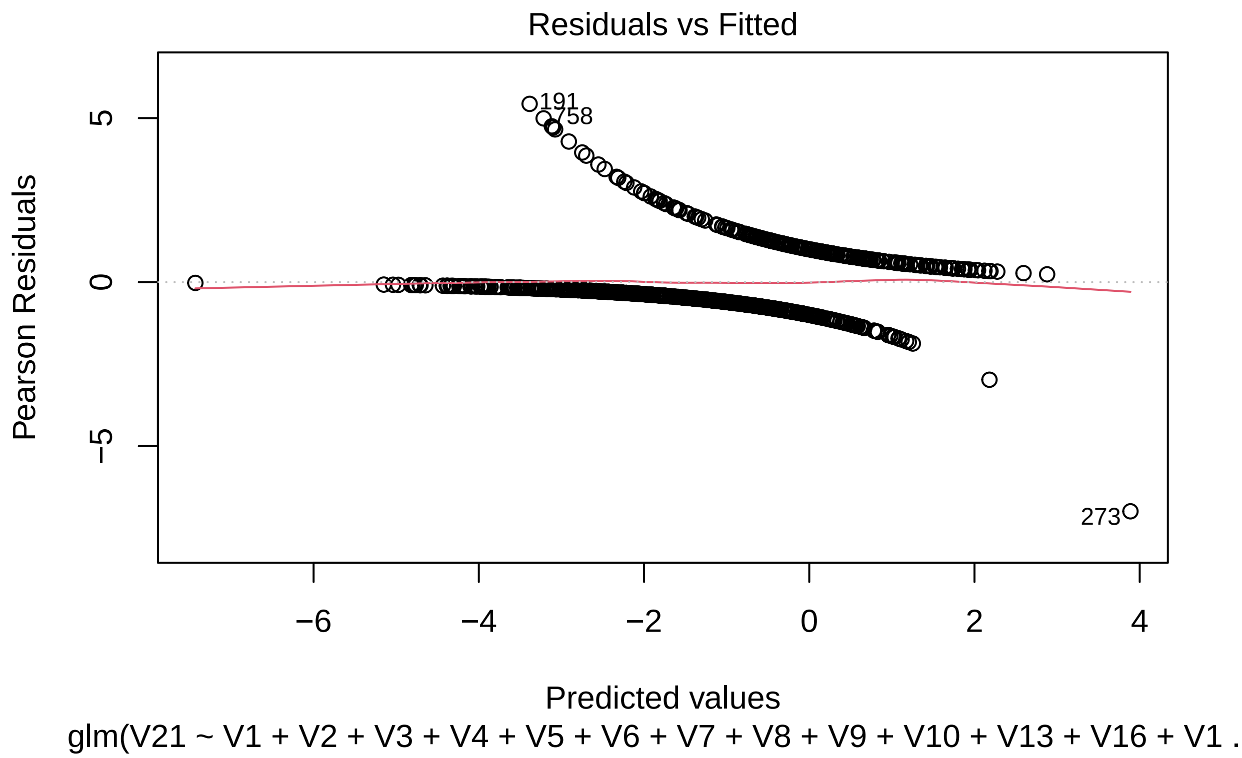
```r
summary(best_model)
```

```
##
## Call:
## glm(formula = V21 ~ V1 + V2 + V3 + V4 + V5 + V6 + V7 + V8 + V9 +
##     V10 + V13 + V16 + V18 + V20, family = binomial(link = "logit"),
##     data = train_germancredit)
##
## Coefficients:
##               Estimate Std. Error z value Pr(>|z|)
## (Intercept)  3.937e-01  9.924e-01   0.397 0.691568
## V1A12       -2.987e-01  2.378e-01  -1.256 0.209060
## V1A13       -9.614e-01  3.951e-01  -2.434 0.014951 *
## V1A14       -1.768e+00  2.639e-01  -6.700 2.08e-11 ***
## V2           2.997e-02  1.002e-02   2.992 0.002767 **
## V3A31        3.533e-01  6.028e-01   0.586 0.557839
## V3A32       -8.373e-01  4.679e-01  -1.790 0.073529 .
## V3A33       -9.210e-01  5.150e-01  -1.788 0.073710 .
## V3A34       -1.575e+00  4.831e-01  -3.260 0.001116 **
## V4A41       -1.676e+00  4.290e-01  -3.907 9.35e-05 ***
## V4A410      -1.368e+00  7.968e-01  -1.718 0.085888 .
## V4A42       -8.492e-01  2.913e-01  -2.916 0.003549 **
```

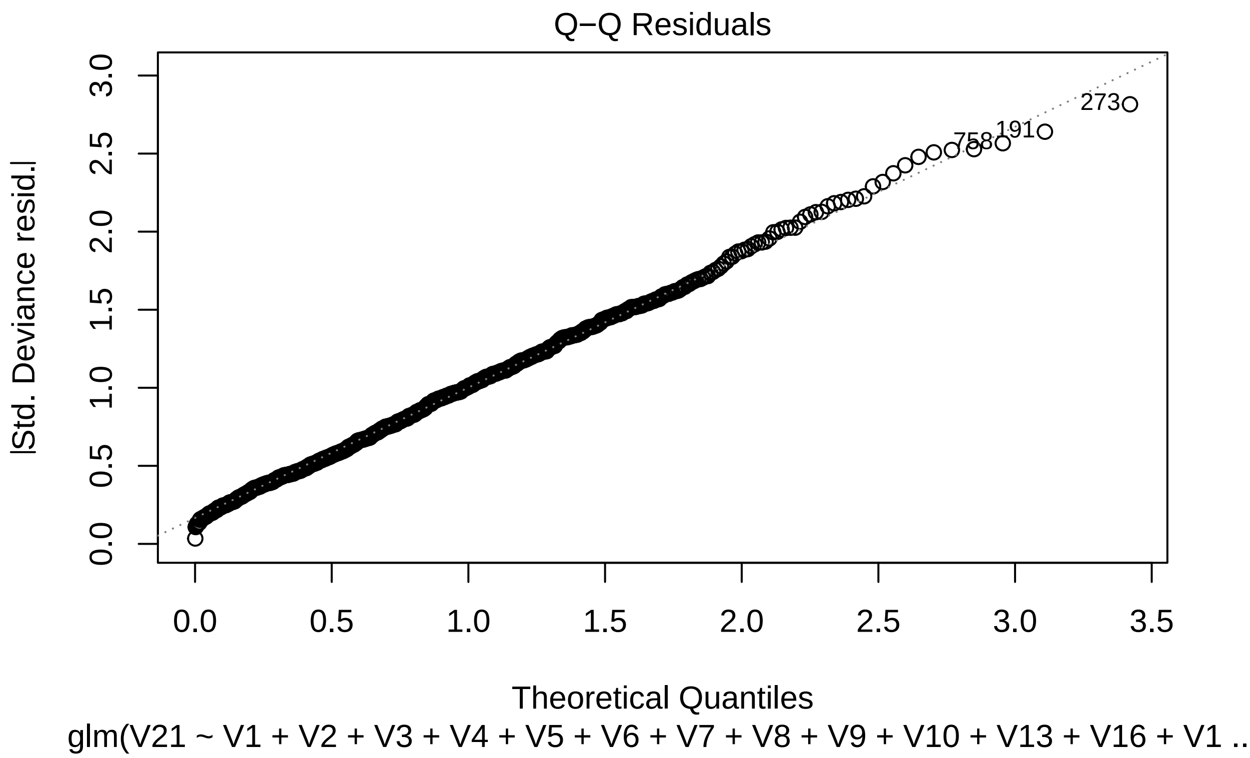
```
## V4A43       -9.546e-01  2.748e-01  -3.474 0.000513 ***
## V4A44       -9.276e-01  8.784e-01  -1.056 0.290983
## V4A45       -4.809e-01  5.949e-01  -0.808 0.418897
## V4A46        1.690e-01  4.151e-01   0.407 0.683923
## V4A48       -2.211e+00  1.192e+00  -1.855 0.063584 .
## V4A49       -8.435e-01  3.752e-01  -2.248 0.024558 *
## V5           1.157e-04  4.739e-05   2.442 0.014588 *
## V6A62       -3.345e-01  3.079e-01  -1.087 0.277254
## V6A63       -4.750e-01  4.765e-01  -0.997 0.318873
## V6A64       -1.207e+00  5.288e-01  -2.282 0.022501 *
## V6A65       -6.888e-01  2.813e-01  -2.449 0.014326 *
## V7A72       -1.436e-01  4.376e-01  -0.328 0.742696
## V7A73       -3.323e-01  4.068e-01  -0.817 0.414015
## V7A74       -1.085e+00  4.543e-01  -2.387 0.016979 *
## V7A75       -3.387e-01  4.154e-01  -0.815 0.414863
## V8           3.687e-01  9.631e-02   3.828 0.000129 ***
## V9A92       -3.871e-01  4.166e-01  -0.929 0.352768
## V9A93       -1.177e+00  4.105e-01  -2.868 0.004131 **
## V9A94       -4.581e-01  4.985e-01  -0.919 0.358127
## V10A102      8.665e-01  4.696e-01   1.845 0.065026 .
## V10A103     -9.132e-01  4.661e-01  -1.959 0.050076 .
## V13         -1.993e-02  9.679e-03  -2.059 0.039510 *
## V16          3.197e-01  2.044e-01   1.564 0.117744
## V18          5.152e-01  2.790e-01   1.847 0.064772 .
## V20A202     -1.443e+00  8.023e-01  -1.799 0.072009 .
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 975.68  on 799  degrees of freedom
## Residual deviance: 717.52  on 763  degrees of freedom
## AIC: 791.52
##
## Number of Fisher Scoring iterations: 5
```

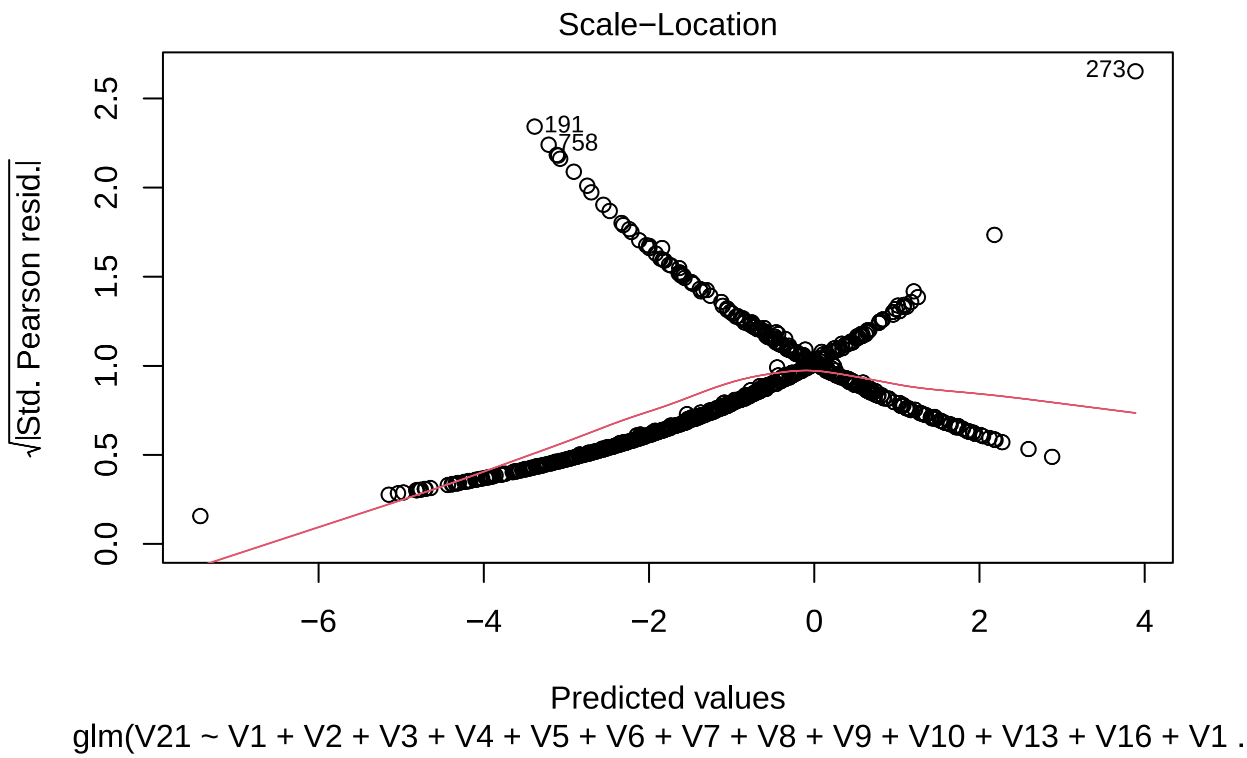
Plot the model:

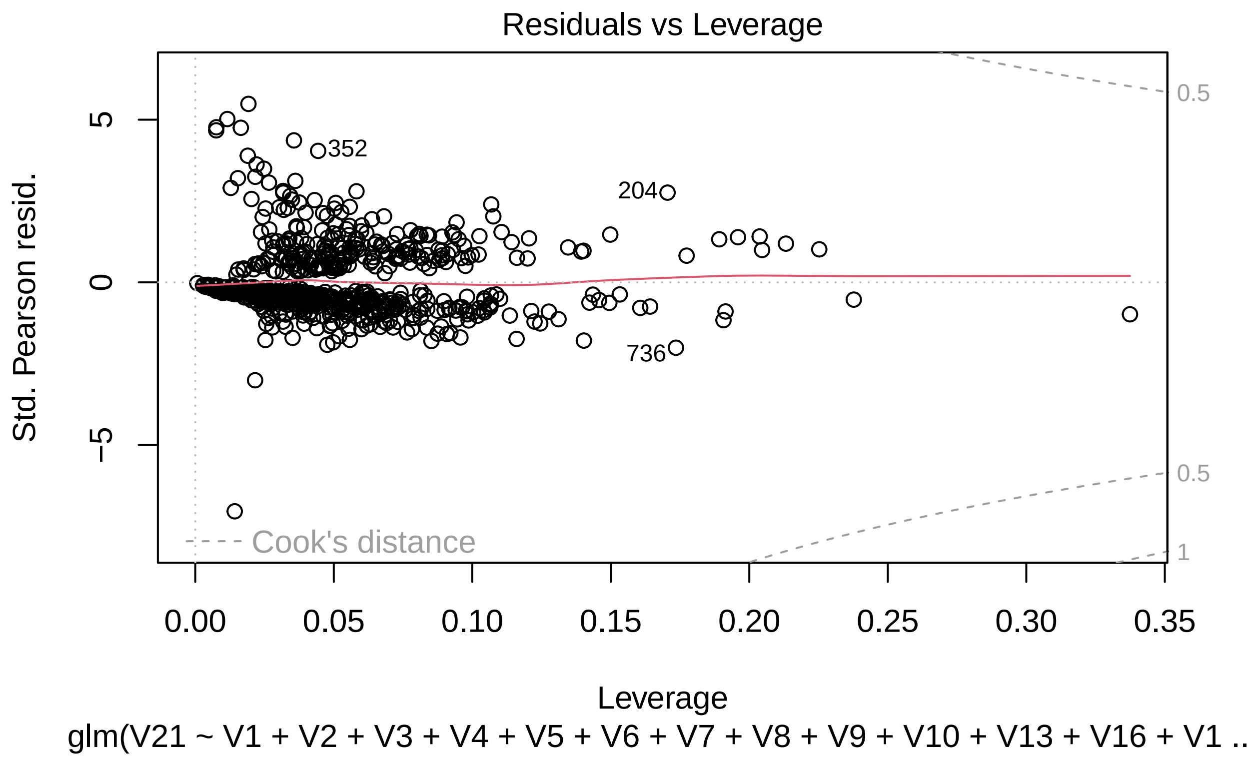
```r
plot(best_model)
```



Residuals vs Fitted
Pearson Residuals
5
0
−5
191
758
273
−6
−4
−2
0
2
4
Predicted values
glm(V21 ~ V1 + V2 + V3 + V4 + V5 + V6 + V7 + V8 + V9 + V10 + V13 + V16 + V1 ..


Q−Q Residuals
|Std. Deviance resid.|
0.0
0.5
1.0
1.5
2.0
2.5
3.0
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5
273
191
758
Theoretical Quantiles
glm(V21 ~ V1 + V2 + V3 + V4 + V5 + V6 + V7 + V8 + V9 + V10 + V13 + V16 + V1 ..


Scale−Location
√|Std. Pearson resid.|
0.0
0.5
1.0
1.5
2.0
2.5
273
191
758
−6
−4
−2
0
2
4
Predicted values
glm(V21 ~ V1 + V2 + V3 + V4 + V5 + V6 + V7 + V8 + V9 + V10 + V13 + V16 + V1 ..



Get prediction of the model on the test set:

```
yhat <- predict(best_model, test_germancredit, type="response")
yhat
```

```
##         801         802         803         804         805         806
## 0.210253786 0.116905737 0.350395573 0.024036697 0.532833324 0.631923091
##         807         808         809         810         811         812
## 0.111940011 0.024900662 0.508903100 0.831098345 0.396528835 0.073513010
##         813         814         815         816         817         818
## 0.315460172 0.444174618 0.749558149 0.708872545 0.025488153 0.018512108
##         819         820         821         822         823         824
## 0.936072863 0.573971292 0.311687827 0.273164449 0.463693452 0.336265181
##         825         826         827         828         829         830
## 0.197537281 0.704407568 0.568777197 0.265824107 0.218142343 0.601492487
##         831         832         833         834         835         836
## 0.134556016 0.825941973 0.887415585 0.255695473 0.250220858 0.739203343
##         837         838         839         840         841         842
## 0.097886820 0.105121549 0.125886965 0.249792249 0.463919576 0.057195952
##         843         844         845         846         847         848
## 0.352636760 0.128622693 0.247222395 0.070580769 0.072357652 0.450732346
##         849         850         851         852         853         854
## 0.078188162 0.214393362 0.182673873 0.006793464 0.054090310 0.845322860
##         855         856         857         858         859         860
## 0.509240374 0.353015381 0.009310942 0.061824262 0.751712225 0.010689446
##         861         862         863         864         865         866
```

```
## 0.020652695 0.199919404 0.664810105 0.076856644 0.050178153 0.085113999
##         867         868         869         870         871         872
## 0.658949683 0.021172014 0.141859872 0.561656329 0.116231228 0.025459692
##         873         874         875         876         877         878
## 0.163759340 0.121685449 0.586336271 0.251432026 0.790115359 0.192274684
##         879         880         881         882         883         884
## 0.518168916 0.017981489 0.034153612 0.075854536 0.349533781 0.091942499
##         885         886         887         888         889         890
## 0.400537522 0.710896784 0.146700910 0.780915456 0.338780258 0.130511954
##         891         892         893         894         895         896
## 0.617345064 0.045406730 0.160956899 0.208786682 0.033061839 0.048843848
##         897         898         899         900         901         902
## 0.679987890 0.002166311 0.018475159 0.455010879 0.107980117 0.146340193
##         903         904         905         906         907         908
## 0.029788460 0.074278489 0.097745251 0.324961312 0.140366314 0.620140318
##         909         910         911         912         913         914
## 0.046092062 0.275865482 0.395828355 0.373249268 0.311717193 0.033019439
##         915         916         917         918         919         920
## 0.763328679 0.668308332 0.040902721 0.448381948 0.477751351 0.457643135
##         921         922         923         924         925         926
## 0.154918530 0.182413608 0.500357565 0.364824449 0.851326505 0.829075635
##         927         928         929         930         931         932
## 0.497735262 0.718238730 0.043035406 0.682665895 0.392281857 0.526626834
##         933         934         935         936         937         938
## 0.084371011 0.040430531 0.566286134 0.429677271 0.207490914 0.458044865
##         939         940         941         942         943         944
## 0.871668255 0.021417564 0.094267430 0.054814766 0.029522700 0.045968774
##         945         946         947         948         949         950
## 0.336106793 0.918384912 0.807810053 0.137690728 0.472261349 0.068886461
##         951         952         953         954         955         956
## 0.615850057 0.290766636 0.300284113 0.733498560 0.558296348 0.152786950
##         957         958         959         960         961         962
## 0.088931939 0.111774430 0.597207803 0.341723220 0.046307415 0.581785853
##         963         964         965         966         967         968
## 0.101044225 0.069324292 0.477513335 0.446216314 0.241013458 0.142363105
##         969         970         971         972         973         974
## 0.089884607 0.339887088 0.184763102 0.216830484 0.953872675 0.908753968
##         975         976         977         978         979         980
## 0.148777406 0.131918129 0.059209445 0.220737147 0.356621011 0.729747262
##         981         982         983         984         985         986
## 0.106653005 0.278954419 0.300334035 0.492839722 0.021627024 0.499193118
##         987         988         989         990         991         992
## 0.841058785 0.045733650 0.478874716 0.204806636 0.087253469 0.259727173
##         993         994         995         996         997         998
## 0.192705956 0.685442963 0.078226379 0.066072509 0.600115497 0.064350559
##         999        1000
## 0.623815374 0.199958485
```

Determine a good threshold by running the threshold in a loop from 0.1 to 0.9, inscrease by 0.01:

```r
#Set the threshold in a range from 0.1 to 0.9, increase by 0.01:
thresholds <- seq(0.1, 0.9, by=0.01)

#Create a vector to contain the cost associated with the threshold:
```

```r
costs <- c()

#Run the threshold in a for loop:
for (thresh in thresholds) {
  yhat_thresh <- as.integer(yhat > thresh)
  #Load caret library to use confusionMatrix() function:
  library(caret)
  confusion_matrix <- confusionMatrix(as.factor(yhat_thresh),as.factor(test_germancredit$V21))

  #False Positives:
  FP <- confusion_matrix$table[2,1]

  #False Negatives:
  FN <- confusion_matrix$table[1,2]

  #Calculate cost that is associated with the threshold:
  cost <- (FP * 5) + (FN * 1)

  #Store the cost into the vector created above:
  costs <- c(costs, cost)
}
```

```
## Loading required package: ggplot2

## Loading required package: lattice
```

```r
#Find the threshold that gives the minimum cost:
optimal_threshold <- thresholds[which.min(costs)]
print(optimal_threshold)
```

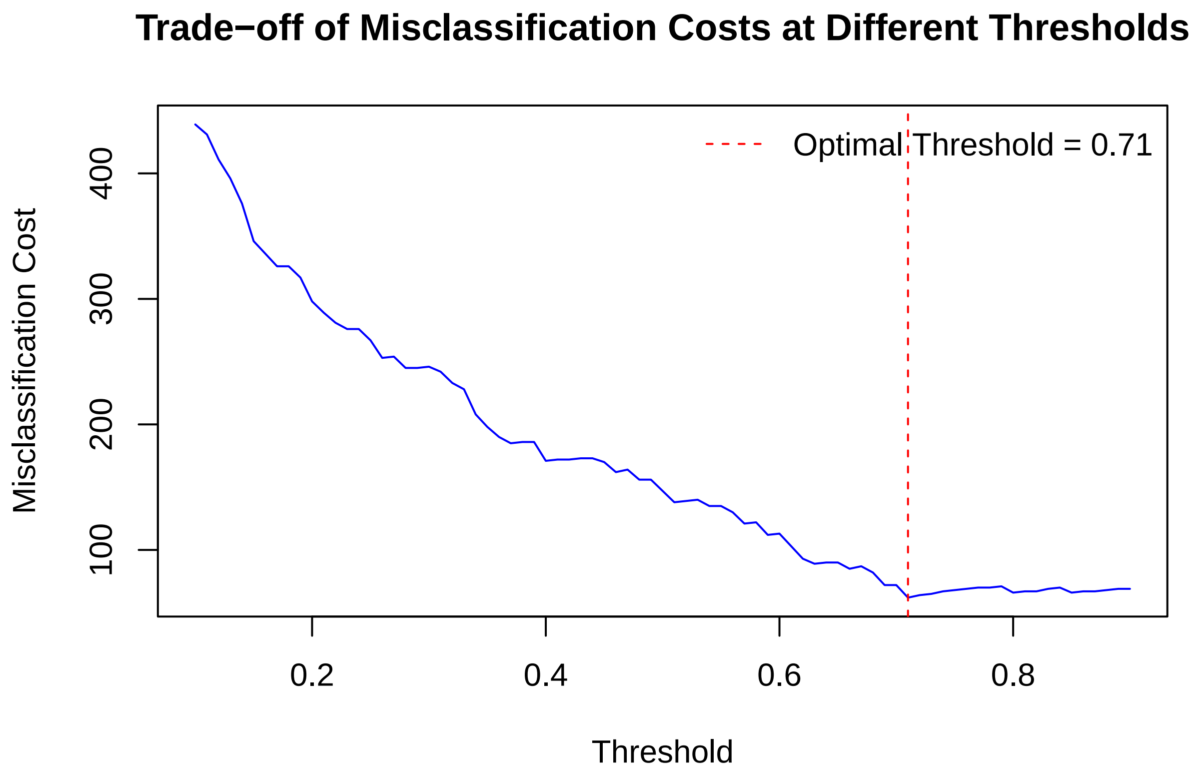
```
## [1] 0.71
```

So, with threshold = 0.71, the cost is minimized.

Before showing the final confusion matrix, I want to visualize the Trade-off of Costs at Different Thresholds:

```r
# Plot the costs at different thresholds
plot(thresholds, costs, type="l", pch=19, col="blue",
     xlab="Threshold", ylab="Misclassification Cost",
     main="Trade-off of Misclassification Costs at Different Thresholds")

# Add a line for the optimal threshold
abline(v=optimal_threshold, col="red", lty=2)
legend("topright", legend=paste("Optimal Threshold =", round(optimal_threshold, 2)),
       col="red", lty=2, bty="n")
```



We can see that at threshold =0.71, the cost drops lowest.

Show the final confusion matrix after getting the optimal threshold:

```
yhat_thresh <- as.integer(yhat > optimal_threshold)
confusion_matrix <- confusionMatrix(as.factor(yhat_thresh),as.factor(test_germancredit$V21))
confusion_matrix
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   0   1
##          0 135  42
##          1   4  19
##
##                Accuracy : 0.77
##                  95% CI : (0.7054, 0.8264)
##     No Information Rate : 0.695
##     P-Value [Acc > NIR] : 0.0115
##
##                   Kappa : 0.3426
##
##  Mcnemar's Test P-Value : 4.888e-08
##
##             Sensitivity : 0.9712
##             Specificity : 0.3115
##          Pos Pred Value : 0.7627
```

```
##          Neg Pred Value : 0.8261
##              Prevalence : 0.6950
##          Detection Rate : 0.6750
##    Detection Prevalence : 0.8850
##       Balanced Accuracy : 0.6413
##
##        'Positive' Class : 0
##
```

The matrix shows that 4 bad customers were wrongly classified as good (FP = 4), which keeps financial risk low. But 42 good customers were wrongly classified as bad (FN = 42), meaning many applicants were rejected unfairly.

> **The model correctly classifies 77% of all cases (both good and bad credit risks).**
>
> **The sensitivity of 0.9712 means that 97.12% of actual good credit customers were correctly classified as good.**
>
> **The specificity of 0.3115 means that only 31.15% of actual bad credit customers were correctly classified as bad.**

Calculate the ROC curve (evaluate the model's ability to separate good vs. bad credit risks at different thresholds) and the Area Under the Curve (AUC- tells you how well your model differentiates between the two classes):

```r
library(pROC)
```

```
## Type 'citation("pROC")' for a citation.
```

```
##
## Attaching package: 'pROC'
```

```
## The following objects are masked from 'package:stats':
##
##     cov, smooth, var
```

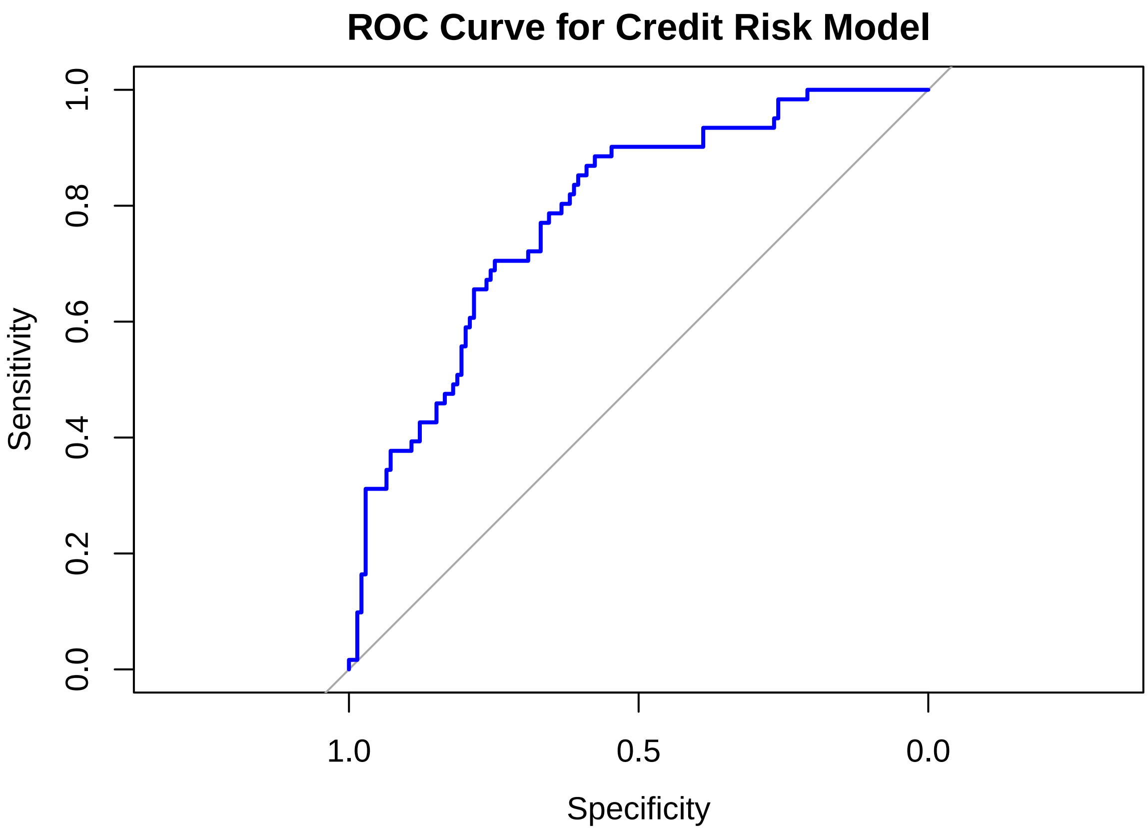
```r
roc_curve <- roc(test_germancredit$V21, yhat)
```

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

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

```r
#I did not round the yhat because ROC analysis requires probability scores, not hard classifications.
#the original continuous predictions (yhat) helps evaluate performance at different thresholds.

plot(roc_curve, col="blue", main="ROC Curve for Credit Risk Model")
```




```r
auc(roc_curve)
```

```
## Area under the curve: 0.7844
```

> The AUC score of 0.7844 suggests that the model is so close to being a good model!!! It has a fairly strong ability to distinguish between good and bad credit risks. This means that if we randomly pick one good and one bad applicant, the model correctly assigns a higher probability to the good applicant 78.44% of the time. An AUC of 1.0 would indicate a perfect model, while 0.5 would mean the model is no better than random guessing.

Calculate the misclassification cost:

```r
#False Positives (bad misclassified as good):
FP <- confusion_matrix$table[2,1]

#False Negatives (good misclassified as bad):
FN <- confusion_matrix$table[1,2]


total_cost <- (FP * 5) + (FN * 1)
print(total_cost)
```

```
## [1] 62
```

At a threshold of 0.71, the total misclassification cost is minimized to 62. This threshold reduces the risk of misclassifying bad customers as good, which is 5x more costly. If the bank wants to approve more good customers, a lower threshold (e.g., 0.7 or 0.6) could be tested. However, this would increase the number of false positives (bad customers approved) and could raise financial risks.

Conclusion:

In this logistic regression model, I used a threshold of 0.71 to classify credit applicants as good or bad risks. AUC = 0.7844, indicating good predictive ability. With this threshold, the confusion matrix results were:

- False Positives (FP) = 4 (Bad classified as good).
- False Negatives (FN) = 42 (Good classified as bad).

Total misclassification cost = 62, which was the lowest cost among tested thresholds.