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
the response is either zero or one, use family=binomial(link="logit") in your glm function
call.

Summary :

- German Credit Data set has various data points including binary, numerical and categorical data. The data cleaning was to use ifelse method on all the data including the response to binary data points for ease of analysis downstream

- Perform a total of 2 iterations to get to the most significant predictors; eliminating insignificant variables

- Build Logistic Model

- Because of FP cost 5x more than FN, need to fine tune the threshold and the optimal threshold for classification was at 13%

- At last, the AUC at 69% dictates that a random person from the yes group would probably get a higher estimate relative to the person from the no group

Observation

The test set sensitivity for positive class '1' is higher in the test set than from cross validation on the build set since 0.**88** is greater than **0.4018385**

Also test set specificity for negative class '0' is lower than from cross validation since 0.4018 is much lower than **0.7988784**

This is a good result still since there is a greater cost for misclassifying the positive class as the negative class.

Test set area under the curve is less than Builddata cross validated AUC of 0.**69**.

```
# -------------------- Code for Question 10.1.1 ----------------------------

# Clear environment

rm(list = ls())

# Setting the random number generator seed so that our results are reproducible

# (Your solution doesn't need this, but it's usually good practice to do)

set.seed(1)

#-------- Load Libraries ------- #

# --------------------------- Data manipulation ------------------------------------

#First, Read in the data

#

data_gc = read.table("germancredit.txt",sep = "")

#

# optional check to make sure the data is read correctly
```

```
#

head(data_gc,2)

  V1 V2  V3  V4   V5  V6  V7 V8  V9  V10 V11  V12 V13  V14  V15 V16  V17 V18  V19  V20 V21
1 A11  6 A34 A43 1169 A65 A75  4 A93 A101   4 A121  67 A143 A152   2 A173   1 A192 A201   1
2 A12 48 A32 A43 5951 A61 A73  2 A92 A101   2 A121  22 A143 A152   1 A173   1 A191 A201   2


#Relabel the response variable as 0 and 1.  Set 1 to 0 for 'Good' and  2 to 1 for 'Bad'
#bad is the positive class
data_gc$V21<-ifelse(data_gc$V21==1,0,ifelse(data_gc$V21==2,1,data_gc$V21))
summary(data_gc$V21)
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
    0.0     0.0     0.0     0.3     1.0     1.0
prop.table(table(data_gc$V21))
  0   1
0.7 0.3
#Based on above probability 70% Good Credit risk and 30% bad
 #split the data into 70% building,30% test
#Randomize the data first
data_gc1 <- data_gc[sample(1:nrow(data_gc)),]

random_row<- sample(1:nrow(data_gc1),as.integer(0.7*nrow(data_gc1 ),replace=F))
builddata = data_gc1[random_row,]
#Assign the test data set to the remaining 30% of the original set
testdata = data_gc1 [-random_row,]

table(data_gc1$V21)
```

#Do glm on all future set in Build Data

```
set.seed(713)

reg1 <- glm(V21 ~.,family=binomial(link = "logit"),data=builddata)

summary(reg1 )
```

```
 Call:
glm(formula = V21 ~ ., family = binomial(link = "logit"), data = builddata)

Deviance Residuals:
    Min       1Q   Median       3Q      Max
-2.0550  -0.6914  -0.3713   0.6255   2.7030

Coefficients:
             Estimate Std. Error z value Pr(>|z|)
(Intercept)  1.281e+00  1.405e+00   0.911 0.362054
V1A12       -2.793e-01  2.673e-01  -1.045 0.296080
V1A13       -1.580e+00  5.365e-01  -2.945 0.003225 **
V1A14       -1.563e+00  2.790e-01  -5.601 2.14e-08 ***
V2           3.432e-02  1.139e-02   3.014 0.002575 **
V3A31        3.189e-02  6.695e-01   0.048 0.962006
V3A32       -6.360e-01  5.097e-01  -1.248 0.212157
V3A33       -5.165e-01  5.570e-01  -0.927 0.353753
V3A34       -1.239e+00  5.128e-01  -2.417 0.015640 *
V4A41       -1.500e+00  4.529e-01  -3.312 0.000925 ***
V4A410      -1.667e+00  9.565e-01  -1.743 0.081297 .
V4A42       -6.392e-01  3.202e-01  -1.996 0.045894 *
V4A43       -7.132e-01  3.007e-01  -2.372 0.017680 *
V4A44        4.628e-01  1.009e+00   0.459 0.646586
V4A45        6.217e-01  6.758e-01   0.920 0.357584
V4A46        1.646e-01  4.795e-01   0.343 0.731397
```

V4A48	-1.501e+01	5.061e+02	-0.030	0.976335	
V4A49	-8.667e-01	4.136e-01	-2.096	0.036115	*
V5	6.817e-05	5.545e-05	1.230	0.218876	
V6A62	-1.963e-01	3.660e-01	-0.536	0.591674	
V6A63	-1.036e+00	5.281e-01	-1.961	0.049875	*
V6A64	-1.468e+00	6.067e-01	-2.420	0.015540	*
V6A65	-8.774e-01	3.113e-01	-2.818	0.004827	**
V7A72	1.205e-01	5.240e-01	0.230	0.818069	
V7A73	-2.039e-01	4.980e-01	-0.409	0.682242	
V7A74	-1.006e+00	5.471e-01	-1.838	0.066061	.
V7A75	-1.709e-01	5.073e-01	-0.337	0.736167	
V8	2.737e-01	1.081e-01	2.532	0.011338	*
V9A92	-8.649e-01	4.723e-01	-1.831	0.067040	.
V9A93	-1.136e+00	4.626e-01	-2.457	0.014017	*
V9A94	-9.721e-01	5.649e-01	-1.721	0.085277	.
V10A102	3.078e-01	5.033e-01	0.612	0.540792	
V10A103	-6.958e-01	4.863e-01	-1.431	0.152462	
V11	-8.948e-03	1.055e-01	-0.085	0.932411	
V12A122	5.914e-01	3.098e-01	1.909	0.056232	.
V12A123	3.720e-01	2.982e-01	1.247	0.212255	
V12A124	9.804e-01	5.343e-01	1.835	0.066518	.
V13	-1.435e-02	1.109e-02	-1.294	0.195741	
V14A142	-1.444e-01	4.823e-01	-0.299	0.764668	
V14A143	-6.105e-01	2.995e-01	-2.038	0.041524	*
V15A152	-6.422e-01	2.876e-01	-2.233	0.025529	*
V15A153	-1.035e+00	5.906e-01	-1.753	0.079533	.
V16	1.728e-01	2.369e-01	0.729	0.465829	
V17A172	2.852e-01	1.004e+00	0.284	0.776407	
V17A173	5.242e-01	9.678e-01	0.542	0.588097	

```
V17A174      3.168e-01  9.668e-01   0.328 0.743133
V18         -1.682e-02  3.082e-01  -0.055 0.956461
V19A192     -1.709e-01  2.427e-01  -0.704 0.481255
V20A202     -1.418e+00  7.042e-01  -2.014 0.044048 *
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

(Dispersion parameter for binomial family taken to be 1)

    Null deviance: 841.21  on 699  degrees of freedom
Residual deviance: 611.85  on 651  degrees of freedom
AIC: 709.85

Number of Fisher Scoring iterations: 14
```

#Based on above Test, we identify insignificant variables based on the level or variables p-value >0.05.

#Drop V20, V19, V18, V17, V15, V13, V12, V11, V10, V7, V5 since those terms had p-values greater than 0.05.

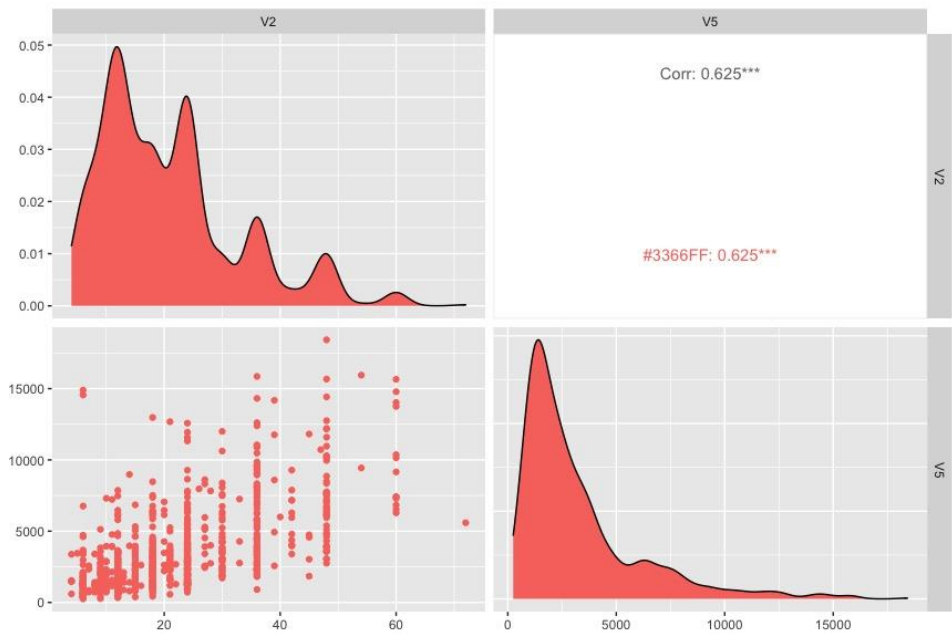
```
#Check for multicolinearity
#
library(car)
vif(reg1)
alias(reg1)
#
#Variables V2 and V5 have slightly elevated levels of variable inflation factors.
#There could be a positive relationship between duration of the loan and credit amount since bigger loans have larger loan durations.
```

#Use scatter plot to explore the extent

#

library(GGally)

ggpairs(data_gc, columns = c('V2', 'V5'), mapping=ggplot2::aes(color= '#3366FF'))



#

#confirms correlation is positive. The anova test revealed we could drop V5 so we'll just retain V2.

#Rerun the model without the insignificant predictors V20, V19, V18, V17, V15, V13, V12, V11, V10, V7, V5 from the model and rerun

#

reg2<-update(reg1,.~.-V20-V19-V18-V17-V15-V13-V12-V11-V10-V7-V5)

summary(reg2)

```
Call:
glm(formula = V21 ~ V1 + V2 + V3 + V4 + V6 + V8 + V9 + V14 +
    V16, family = binomial(link = "logit"), data = builddata)

Deviance Residuals:
    Min       1Q   Median       3Q      Max
-2.0997  -0.7219  -0.4122   0.7047   2.7850
```

```
Coefficients:
            Estimate Std. Error z value Pr(>|z|)
(Intercept)   0.899443   0.833958   1.079  0.28080
V1A12       -0.229943   0.248998  -0.923  0.35576
V1A13       -1.479960   0.506436  -2.922  0.00347 **
V1A14       -1.538294   0.265202  -5.800 6.61e-09 ***
V2          0.044739   0.008459   5.289 1.23e-07 ***
V3A31       -0.367203   0.623662  -0.589  0.55601
V3A32       -0.881794   0.486126  -1.814  0.06969 .
V3A33       -0.754188   0.539185  -1.399  0.16189
V3A34       -1.568869   0.492545  -3.185  0.00145 **
V4A41       -1.278108   0.428456  -2.983  0.00285 **
V4A410       -1.248226   0.885577  -1.410  0.15869
V4A42       -0.311196   0.296121  -1.051  0.29330
V4A43       -0.740112   0.280919  -2.635  0.00842 **
V4A44        0.739196   0.952095   0.776  0.43752
V4A45        0.568370   0.643672   0.883  0.37723
V4A46        0.362919   0.451306   0.804  0.42131
V4A48      -15.078590 505.775652  -0.030  0.97622
V4A49       -0.799780   0.389145  -2.055  0.03986 *
V6A62       -0.074669   0.344678  -0.217  0.82849
V6A63       -1.006593   0.511247  -1.969  0.04897 *
V6A64       -1.439070   0.579647  -2.483  0.01304 *
V6A65       -0.897226   0.295765  -3.034  0.00242 **
V8          0.201132   0.091310   2.203  0.02761 *
V9A92       -0.632101   0.438133  -1.443  0.14910
V9A93       -1.099886   0.428547  -2.567  0.01027 *
V9A94       -0.877724   0.537281  -1.634  0.10233
```

```
V14A142     -0.084506   0.458866  -0.184  0.85389
V14A143     -0.547113   0.284601  -1.922  0.05456 .
V16          0.151009   0.219033   0.689  0.49055
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

(Dispersion parameter for binomial family taken to be 1)

    Null deviance: 841.21  on 699  degrees of freedom
Residual deviance: 646.94  on 671  degrees of freedom
AIC: 704.94

Number of Fisher Scoring iterations: 14
```

```
#
#We now have a new model with selected terms.  AIC has improved for the model since it decreased from 709 to 704.94
#The 4th variable has a really strange coefficient estimate for A48 and very high p-values for both A44 and A48.
#Need to explore this more and decide whether to combine these levels with another level.
#
View(builddata$V4)
#
#After viewing variable 4 which is a categorical variable for the Purpose of the loan I found there is only 4 observations for A48 factor level which stands for the loan purpose of retraining.
#This purpose has such a low number of observations in the build set and is causing unstability in its estimation so I'll combine A48 with the base level A40 (car new) and rerun the model.
#Factor A44 has twice as many observation in the build set but at 8 that number is still low, so I will also combine this level with the base to zero out the coefficient.
#See if AIC improves.
#
```

```
levels(builddata$V4)[levels(builddata$V4)==c("A48","A44")]<-"A40"

reg2.5<-update(reg2,.~.)

summary(reg2.5)

#

#AIC value is not changed much... so improved the stability of the model by relabelling the variable.

#Show the cross validated performance of the model Using 7 folds so we have 100 obs in each fold

#

library(caret)

builddata$risk<-as.factor(ifelse(builddata$V21==1,'bad','good'))

buildfolds<-caret::createFolds(builddata$risk,k=7)

set.seed(478)

cvrm<-train(risk ~ V1 + V2 + V3 + V4 + V6 + V8 + V9 + V14 + V16,

         data=builddata,

         method='glm',

         trControl=trainControl(method = 'cv', number = 7, index = buildfolds, classProbs = TRUE,
summaryFunction = twoClassSummary),

   metric='ROC')

cvrm
```

Generalized Linear Model

700 samples

9 predictor

2 classes: 'bad', 'good'

No pre-processing

Resampling: Cross-Validated (7 fold)

Summary of sample sizes: 100, 100, 100, 100, 100, 100, ...

Resampling results:

ROC Sens Spec

0.6530515 0.4018385 0.7988784

```
cvrm$finalModel

#
#Here is the 7-fold cross validated area under the curve=0.6788661, sensitivity=0.4526674 and specificity= 0.8099621
#assuming a 50% threshold and '1' is the positive class.
#Caret found some predictions were equal to 0 or 1 when the output is probability which happens with logit regression.
#performance changes on the test set for the 50% threshold.
#Make prediction on the test set similar to Buildata
#
levels(testdata$V4)[levels(testdata$V4)==c("A48","A44")]<-"A40"
reg2test<-predict(reg2.5, newdata=testdata,type='response')
#set threshold at 50%
reg2testfact<-as.factor(ifelse(reg2test>0.5,1,0))#positive class is bad risks
confusionMatrix(reg2testfact,as.factor(testdata$V21))#using 50% threshold
```

Confusion Matrix and Statistics

```
          Reference
Prediction   0   1
         0 178  51
         1  24  47
```

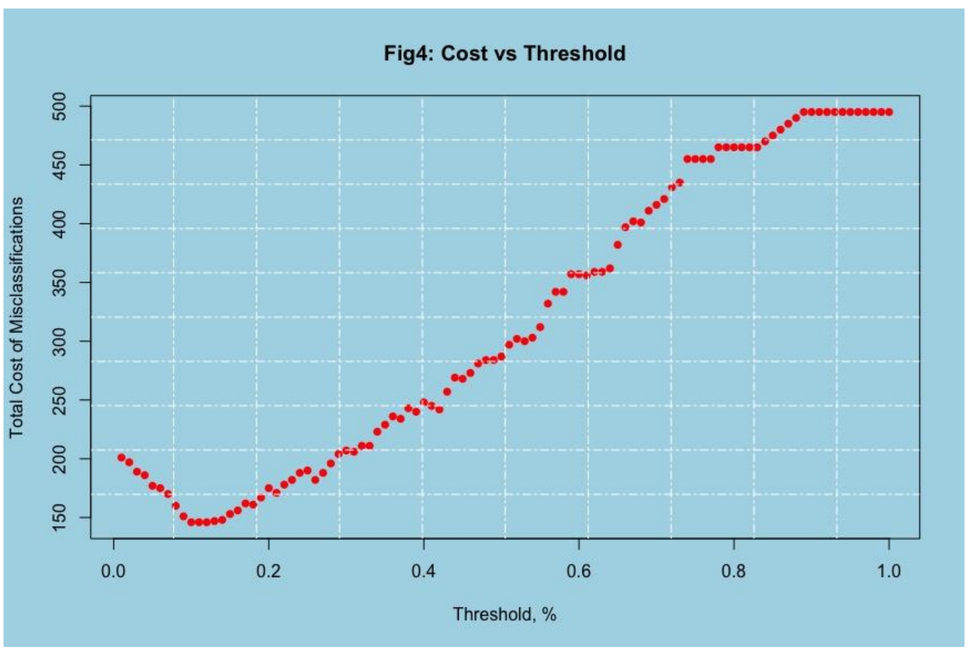
```
Accuracy : 0.75
95% CI : (0.697, 0.798)
No Information Rate : 0.6733
P-Value [Acc > NIR] : 0.002357

Kappa : 0.3883

Mcnemar's Test P-Value : 0.002680

Sensitivity : 0.8812
Specificity : 0.4796
Pos Pred Value : 0.7773
Neg Pred Value : 0.6620
Prevalence : 0.6733
Detection Rate : 0.5933
Detection Prevalence : 0.7633
Balanced Accuracy : 0.6804

'Positive' Class : 0
```

```
library(pROC)
roc(testdata$V21,ifelse(reg2test>0.5,1,0))#inputs must be numeric
plot.roc(testdata$V21,ifelse(reg2test>0.5,1,0)

# Total Accuracy 75%
#Sensitvity 88%
# Specificity 47%
```

10.3.2. 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.

```
costs = matrix(c(0, 5, 1, 0), nrow = 2)

dimnames(costs) = list(Actual = c("good", "bad"), Predicted= c("good", "bad"))

print(costs) #Cost of misclassifications 5x for FP vs. FN


    Predicted

Actual good bad

 good    0   1

 bad     5   0


#initialize list

cost <- vector(mode = "list")

set.seed(713)

predicted <- predict(reg2  , testdata, type="response")




for (i in 1:100){

  predicted_roundup <- as.integer(predicted > i/100 )

  cm_matrix <- as.matrix(table(testdata$V21 ,predicted_roundup))

  #Ensuring NO out of bounds issues while looping
```

```
  if(nrow(cm_matrix)==2) {fp<-cm_matrix[2,1]} else {fp=0}

  if(ncol(cm_matrix)==2){fn<-cm_matrix[1,2]} else {fn=0}


  cost<-c(cost, fn*1+fp*5)

}

#Plots ov Total cost vs % thresholds

par(bg = 'lightblue')

plot(x=seq(0.01,1,by=0.01),y=cost,xlab = "Threshold, %",ylab = "Total Cost of Misclassifications",main = "Fig4: Cost vs Threshold",col='red',pch=16)

grid (10,10, lty = 6, col = "white")


numerator<-which.min(cost)

min.threshold <-numerator/100

print(sprintf("Minimum Threshold due to Uneven Error Cost= %0.3f", min.threshold))
```




```
# "Minimum Threshold due to Uneven Error Cost= 0.100"
```