Exercise 1: The idea of credit scorning is that you try to classify people into a group that will pay the money back and a group that won't. We'll use a version of a well-known dataset called Kredit, which contained data from 1000 germans. This version we'll use is from kaggle.

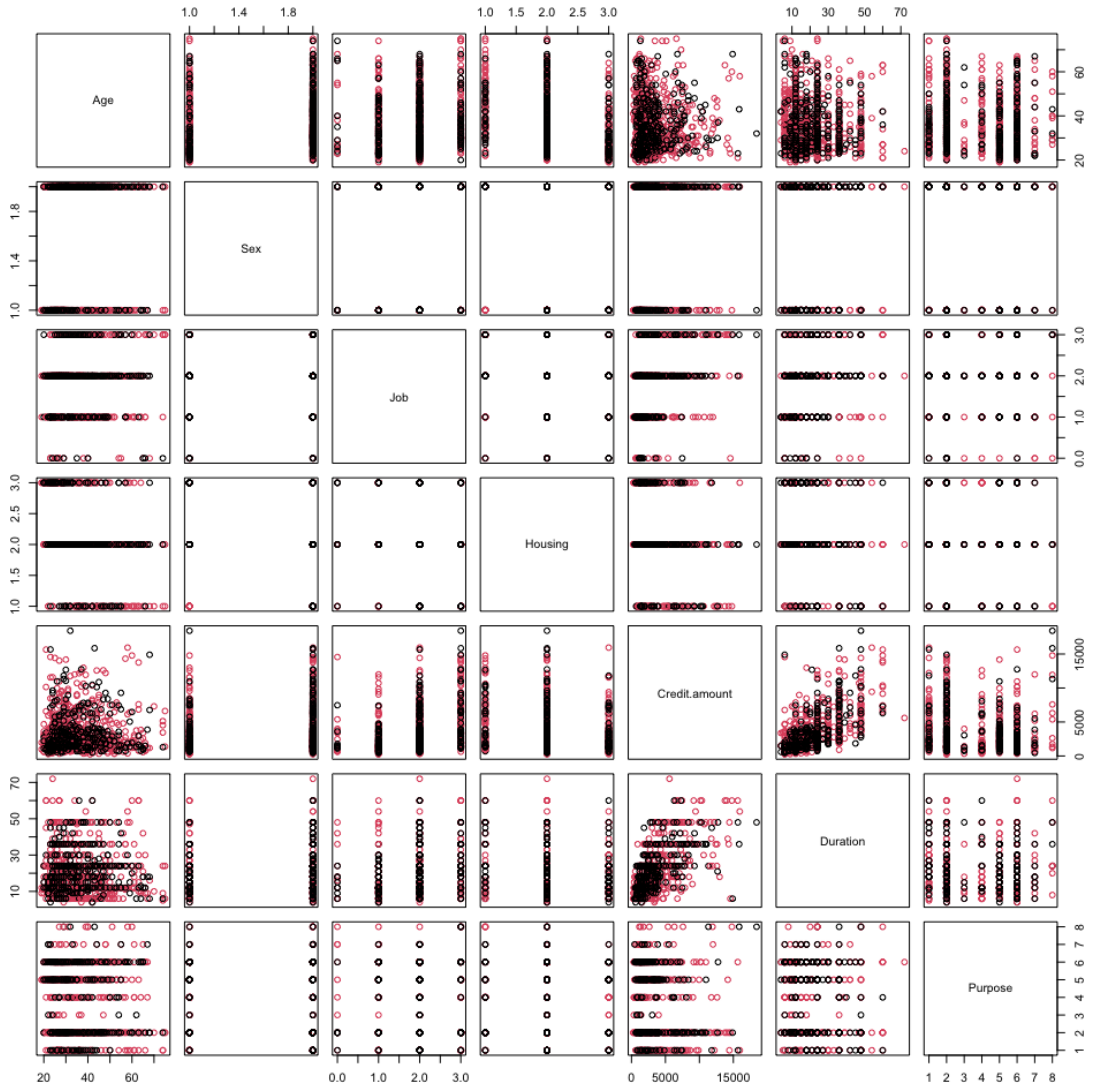
a. Load the csv german_credit_data.

Solution:
Reading in the data we find that the checking and saving account variables contain all the missing values. Simply removing all the rows with missing values reduces our number of observations to 522, which is about half. I am not necessary sure we should be throwing out so much of our data, for two variables which are not entirely indicative of an individuals credit responsibility. There are a few actions we can take to try and save this data. We could impute the medium value for these missing values, use a discriminant method that is robust to missing values like decision trees, or we could simply drop these categorical predictors. I ended up deciding to remove the variables mainly because KNN performs better on lower dimensional data. Plotting the pair plots shows us that this data is not in the slightest separable so I don't think any method will perform very well at all; this is also corroborated by the manova analysis below.
Code:

```
## Reading in the data.
data <- read.csv('german_credit_data.csv', header = TRUE)

## Note that checking and saving account variable contains a lot of NAs.
length(na.omit(data$Saving.accounts))
# [1] 817
length(na.omit(data$Checking.account))
# [1] 606
length(na.omit(data)[,1])

## Dropping these two columns removes all NAs. Perhaps we
## can explore a method more robust to NA, like decision trees
## and bring that data in.
data <- data[, -c(1,7,8)]

## Turning categorical variables into factors (Works for LDA).
data$Sex <- as.factor(data$Sex)
data$Housing <- as.factor(data$Housing)
data$Purpose <- as.factor(data$Purpose)
data$Kredit <- as.factor(data$Kredit)

## Dummy Variables were used for KNN()
## library(fastDummies)
## dataDummies <- dummy_cols(data, select_columns=c('Sex', 'Housing', 'Purpose'),
##                 remove_first_dummy=TRUE)
## dataDummies <- dataDummies[,-c(3,5,8)]
```

```
## data <- dataDummies


## Plotting Pairs
pairs(data[,-1], col = data[,1])



-------------------------------------------------------------------------------
## MANOVA

> summary.aov(
    manova(cbind(Age, Sex, Job, Housing, Credit.amount, Duration, Purpose) ~ data[
    data = data[,-1]))
## None of the Predictors are significantly different between classes.
 Response Age :
             Df Sum Sq Mean Sq F value Pr(>F)
data[, 1]     1     23  23.467  0.1812 0.6704
Residuals   998 129248 129.507

 Response Sex :
             Df  Sum Sq  Mean Sq F value Pr(>F)
data[, 1]     1   0.019 0.019048  0.0889 0.7657
Residuals   998 213.881 0.214310

 Response Job :
             Df Sum Sq Mean Sq F value Pr(>F)
data[, 1]     1   0.18 0.18305  0.4282  0.513
Residuals   998 426.60 0.42746

 Response Housing :
             Df  Sum Sq Mean Sq F value Pr(>F)
data[, 1]     1   0.282 0.28233  1.0003 0.3175
Residuals   998 281.677 0.28224

 Response Credit.amount :
             Df     Sum Sq Mean Sq F value Pr(>F)
data[, 1]     1    5325750 5325750  0.6682 0.4139
Residuals   998 7954549877 7970491

 Response Duration :
             Df Sum Sq Mean Sq F value Pr(>F)
data[, 1]     1      9   9.261  0.0636 0.8009
Residuals   998 145260 145.551

 Response Purpose :
             Df Sum Sq Mean Sq F value Pr(>F)
data[, 1]     1    1.1  1.1293  0.2884 0.5914
Residuals   998 3908.0  3.9158
```

Figure 1: Pair Plot of Variables Colored by Kredit.



b. Try a k-nearest neighbors classification. Select a value of k that you think is reasonable, then split the data into a training set and a testing set, then look at how well it classified the test data.

Solution:
There are several ways to decide a value of k, I have seen that as a rule of thumb some people simply decide on the square root of the number of observations in the data. Since our data has 1000 observations we will try with $k = \sqrt{(1000)} \approx 32$. Ideally we would run the knn classification on multiple k and see what performs best. Performing an 80-20 split on the data an conducting a knn classification with

32 neighbors, we found that the model got correctly classified 148 of the 200 test observations getting an accuracy of .75 and a misclassification rate of .25. However I suspect that we chose a k that was too large and the model is simply classifying all of the test data into one category. This strategy seems like it will yield an accuracy that is the same as the prior distribution of the observations, we should be able to do better.

Code:

```
## Split the data and cluster using knn
set.seed(123)
split <- sample(c(rep(0, 0.8 * nrow(data)), rep(1, 0.2 * nrow(data))))
test_data <- data[split == 1,]
train_data <- data[split == 0,]
library(class)
init_KNN <- knn(train = train_data[,-1],
                test = test_data[,-1],
                cl = train_data[,1], k = 32)

table(test_data[,1], init_KNN)

## init_KNN
##      0   1
## 0    0  58
## 1    0 142
```

c. Now repeat (b), but this time perform a crossvalidation (either leave one out, or k-fold). Which of these confusion matrices do you think gives you a better idea as to how well KNN will classify new observations?

Solution:

Performing one-out cross validation, as one could guess generated a similar confusion matrix. Here we found that the $k = 32$ knn model correctly classified 697 of the 1000 one-out cross validation samples achieving an accuracy of .679 and a misclassification rate of .321. Consider that when knn includes all neighbors the model will classify every observation to the class with the largest prior distribution, clearly this example with $k = 32$ is large enough to get a similar behavior. We know that 700 of the 100 observations are classified as 1, and the cross validation technique is closer to this value than the splitting method. Ideally we would train the model on all the data then collect more to test, but for this case I trust the cross validated heuristic better.

Code:

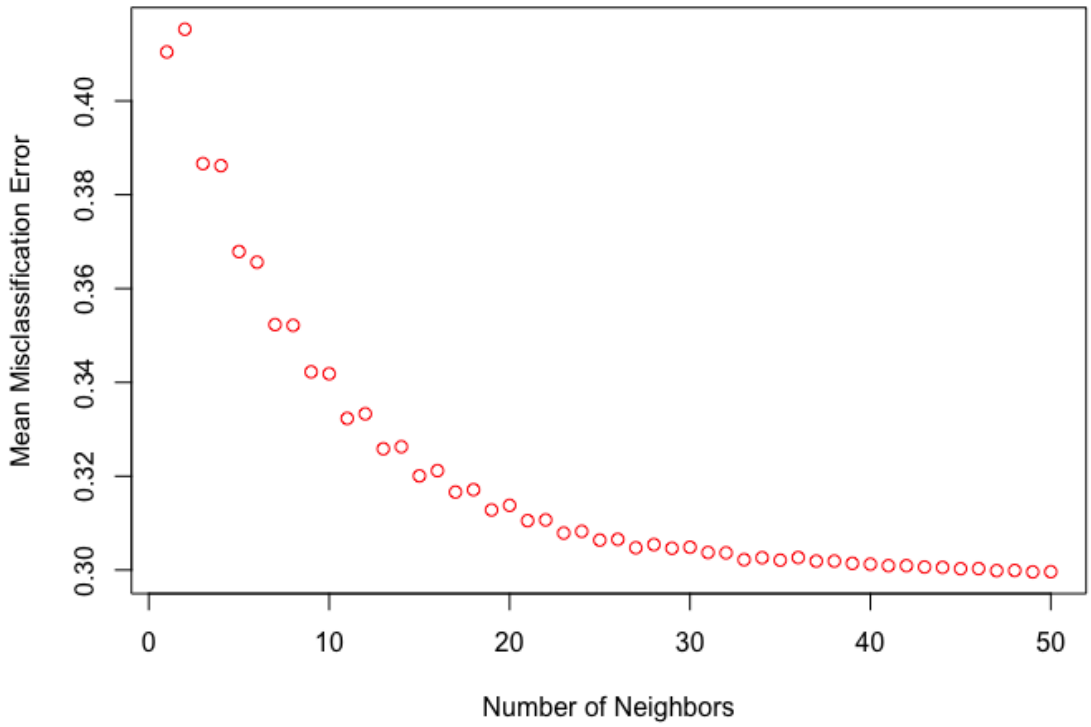
```
## One-out Cross Validation.
CV_classifications <- rep(NA,1000)
for(i in 1:1000){
  ## Build classifier on one-out data
```

```
  tmp_rule <- knn(train=data[-i,-1],
                  test=data[i,-1],
                  cl=data[-i,1], k = 32)
  ## Predict one-out data
  CV_classifications[i] <- tmp_rule[1]
}

## Generating confusion matrix
table(data[,1], CV_classifications - 1)
##      0   1
## 0    0 300
## 1    3 697

## Checking prior dist of data
length(data$Kredit[data$Kredit == 1])
[1] 700
```

d. Try several values of k and look at the test/train confusion matrix from (b) for each value of k. What is your preferred value of k?

Solution:
Finding the optimal k we train the model several times and consider the accuracy/-classification rate. Plotting values of k against the misclassification rate we would expect it to dip down and then increase and level off, kind of matching the bias and variance tradeoff. Interestingly it seems as though the plot of the misclassification does not show a minimum and the smallest values correspond to the most underfit classifications.

Figure 2: Results from Repeated Splits



Code:

```
## Split the data and cluster using knn
RandomSplits <- data.frame(matrix(ncol = 100, nrow = 50))
for (j in 1:100){
  split <- sample(c(rep(0, 0.5 * nrow(data)), rep(1, 0.5 * nrow(data))))
  test_data <- data[split == 1,]
  train_data <- data[split == 0,]
  MissClassErrorTest <- rep(NA,50)
  for(i in 1:50){
    Pred_test <- knn(train = train_data[,-1],
                     test = test_data[,-1],
                     cl = train_data[,1], k = i)
    MissClassErrorTest[i] <- (1 - (sum(diag(table(test_data[,1], Pred_test)))/500))
  }
  RandomSplits[,j] <- MissClassErrorTest
}

plot(rowMeans(RandomSplits), col='red',
     xlab = 'Number of Neighbors',
     ylab = 'Mean Misclassification Error')
```

I think we might need to use a different evaluation method or more likely the data are too homogenous and finding a model to predict Kredit with these parameters is not very possible.

Exercise 2.: Now take the same data as before, but run a linear discriminant analysis using the function lda().

a. Run the analysis and classify all of the data using a rule built by all of the data (i.e. the naive confusion matrix). Does the classification seem to work?

Solution:
Running the linear discriminant analysis using lda(), we get another issue similar to the knn model. It seems as though both models result in applying the a single classification to almost all of the data. I imagine what is causing this is that both of the data groups are lying right on top of each other inside the ellipsoid which spans the data.(I ran the analysis including the variables that we removed in the beginning and still had the same problem).
Code:

```
library(MASS)
classifier <- lda(data[,1] ~.  ,data = data[,-1])

library(caret)
confusionMatrix(factor(data[,1]),
                predict(classifier)$class)
```

```
------------------------------------------
Confusion Matrix and Statistics

          Reference
Prediction   0   1
         0   0 300
         1   0 700

               Accuracy : 0.7
                 95% CI : (0.6705, 0.7283)
    No Information Rate : 1
    P-Value [Acc > NIR] : 1

                  Kappa : 0

 Mcnemar's Test P-Value : <2e-16

            Sensitivity :  NA
            Specificity : 0.7
         Pos Pred Value :  NA
```

```
          Neg Pred Value :  NA
              Prevalence : 0.0
          Detection Rate : 0.0
    Detection Prevalence : 0.3
       Balanced Accuracy :  NA

        'Positive' Class : 0
```

b. lda() has an argument that performs cross-validation. Use it to get a cross validated confusion matrix. How well do you think the classification is working?

Solution:
Running the cross validated lda we get the the same confusion matrix, because the model in both cases is just assigning the same class to every observation. The data are most definitely not linearly separable so it seems its assigning all the observations to the class with the greater proportions.
Code:

```
CVclassifier <- lda(data[,1] ~.  ,data = data[,-1], CV = TRUE)
confusionMatrix(factor(data[,1]),CVclassifier$class)
```

```
-----------------------------------------------
Confusion Matrix and Statistics

          Reference
Prediction   0   1
         0   0 300
         1   0 700

               Accuracy : 0.7
                 95% CI : (0.6705, 0.7283)
    No Information Rate : 1
    P-Value [Acc > NIR] : 1

                  Kappa : 0

 Mcnemar's Test P-Value : <2e-16

            Sensitivity :  NA
            Specificity : 0.7
         Pos Pred Value :  NA
         Neg Pred Value :  NA
             Prevalence : 0.0
         Detection Rate : 0.0
   Detection Prevalence : 0.3
      Balanced Accuracy :  NA

       'Positive' Class : 0
```

c. When do you expect linear discriminant analysis to work especially well?

Solution:
We expect linear discriminant analysis to work best when the data are linearly separable . To some extent we can allow the data to overlap and LDA will provide a suitable mean boundary that can be tuned by priors, as we saw in the examples during class. With the Kredit data the two classes are almost entirely overlapping.

Exercise 3: In problem two, we have the option of setting prior probabilities for each group. The default is to use the actual size of the groups in the data (70 percent creditworthy), but you can modify the priors. Run an lda analysis with the prior belief that 40 percent of the population should be credit- worthy and 60 percent not. You can just use a testing/training approach for this problem (not cross validation).

Solution:
Running the LDA with theses priors we get a training accuracy of 0.3238 and a testing accuracy 0.345. When the data are completly indeparble our classifyer before was able to do relatively well by taking advantage of the proportional priors in our data(assigning all observations to largest proportional class size). We cannot assume that credit worthy people make up 70 percent of the population and similarly with non-credit worthy people. In reality our prior models were likely at best randomly assigning credit worthiness. It feels like the only reason we have an accuracy lower than 50% is because our priors and the proportion of classes in our data are at odds.
Code:

```
split <- sample(c(rep(0, 0.8 * nrow(data)), rep(1, 0.2 * nrow(data))))
test_data <- data[split == 1,]
train_data <- data[split == 0,]


classifier <- lda(train_data[,1] ~.  ,data = train_data[,-1], prior = c(.6, .4))
confusionMatrix(factor(train_data[,1]), predict(classifier)$class)
------------------------------------------------------
Train Confusion Matrix

          Reference
Prediction   0   1
         0 229   8
         1 533  30

Accuracy : 0.3238

-------------------------------------------------------ss
Pred_test <- predict(classifier , newdata = test_data[,-1])
confusionMatrix(factor(test_data[,1]), Pred_test$class)
------------------------------------------------------
Test Confusion Matrix
```

```
          Reference
Prediction   0   1
         0  59   4
         1 127  10

 Accuracy : 0.345
```

Exercise 4: Now repeat problem two, but using quadratic discriminant analysis:

a. Run the analysis and classify all of the data using a rule built by all of the data (i.e. the naive confusion matrix). Does the classification seem to work?

Solution:
Running the analysis we get better spread of classification. The confusion matrix shows that the model predicted 0 for 145 of the observations which in general feels more productive than classifying every observation as a single class even though we achieved slightly lower accuracy at 0.693. I'm surprised that QDA was actually able to achieve this as it seems that, looking at the pair plot the data does not seem to be quadratically separable.
Code:

```
QDAclassifier <- qda(data[,1] ~.  ,data = data[,-1])
confusionMatrix(factor(data[,1]),
                predict(QDAclassifier)$class)
```

```
--------------------------------------------------
Confusion Matrix and Statistics

          Reference
Prediction   0   1
         0  69 231
         1  76 624

Accuracy : 0.693
```

b. qda() has an argument that performs crossvalidation. Use it to get a crossvalidated confusion matrix. How well do you think the classification is working?

Solution:
Generating the cross validated confusion matrix we get an accuracy of 0.639, which seems more representative of a method that is actually predicting some of the observations to be 0. **Code:**

```
QDACVclassifier <- qda(data[,1] ~.  ,data = data[,-1], CV = TRUE)
confusionMatrix(factor(data[,1]),QDACVclassifier$class)

-----------------------------------------------------
Confusion Matrix and Statistics

          Reference
Prediction   0   1
         0  35 265
         1  96 604

Accuracy : 0.639
```

c. When do you expect quadratic discriminant analysis to work especially well?

Solution:
We expect quadratic discriminant analysis to work best when the data are quadratically separable (what does one of these decision boundaries look like in 3d and hyperspace? Are they degenerate like a cylinder or non-degenerate like an ellipsoid?).

d. Of the models (Mahalanobis, k-nearest neighbor, linear discriminant, and quadratic discriminant), which are most likely to overfit the data? Which are most likely to underfit the data?

Solution:
What you are describing is the bias-variance tradeoff that each model has. KNN has the property that it can underfit with large values of k and overfit with small values of K (like a tree, it has options). Generally LDA and QDA are methods that underfit the data. I can imagine the Mahalanobis classifier making curved decision boundaries similar to QDA so it also likely underfits data.

Exercise 5: Which of the analyses you did in this assignment did you prefer? Did one seem to work better with the variables you selected?

Solution:
I preferred KNN. I like how we can tune it like decision trees to over and underfit the data. It seems like boosting them leads to a form of weighted KNN. I don't think bagging them will lead to substantially better results since bootstrapping the data will likely give similar decision boundaries. I'm not sure if I did a bad job preparing the data, from the pair plot and MANOVA it seems like they were right on top of each other which makes discriminant analysis difficult. The QDA model seemed to actually acknowledge the other class in the

data so I wonder if there is some sort of transformation in the predictors that really separates the data.