Exercise 2

a) (0.5 points)

Linear and quadratic discriminant analysis are based on the assumption that the feature vector $\mathbf{X} \in \mathbb{R}^p$ is multivariate normal distributed within the classes, i.e. we have $P_{\mathbf{X}|Y=\ell} \sim \mathcal{N}_p(\mu_\ell, \mathbf{\Sigma}_\ell)$, where $Y \in \{0, \ldots, k\}$ denotes the random variable that describes the class label. For linear discriminant analysis, it is further assumed that $\mathbf{\Sigma}_1 = \ldots = \mathbf{\Sigma}_k$, i.e. the covariance matrix of the feature vector is the same for all classes.

b) (0.5 points)

First, we load the `ggplot2` package for plotting and the `MASS` package for performing LDA/QDA. Then we read the training and test datasets into our environment and take a look at the first ten rows and 6 columns of the training dataset.

```
library(ggplot2)
library(MASS)

digits_train <-  read.csv(file= paste0("C:/Users/leaz9/OneDrive/Dokumente/StatLearn WS22",
                                       "/data/train_digits.csv"))
digits_train[ 1:10, 1:6]
```

```
##    V1 V2 V3 V4 V5 V6
## 1   0  0  0  0  0  0
## 2   4  0  0  0  0  0
## 3   1  0  0  0  0  0
## 4   1  0  0  0  0  0
## 5   1  0  0  0  0  0
## 6   4  0  0  0  0  0
## 7   6  0  0  0  0  0
## 8   1  0  0  0  0  0
## 9   8  0  0  0  0  0
## 10  6  0  0  0  0  0
```

```
digits_test <-  read.csv(file= paste0("C:/Users/leaz9/OneDrive/Dokumente/StatLearn WS22",
                                      "/data/test_digits.csv"))
```

Next, we try to use LDA for classifying the observations according to their class label (the true digit).

```
lda_dig <- lda(V1 ~ ., data = digits_train)
```

```
## Error in lda.default(x, grouping, ...): variables   1   2   3   4   5   6   7   8   9  10  11  12  13
```

The error message states that there are some variables that are constant. This is because all digits are centered, i.e. some of the pixels on the edges are always white. Running

```
summary(digits_train$V2)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##       0       0       0       0       0       0
```

confirms that (the top left pixel has value zero for all observations). This violates the assumption of normality, but more importantly, the (pooled) covariance matrix is singular and cannot be inverted (constant variables have zero variance and covariance). This is why we get an error message.

c) 1.5 points

We perform a principal component analysis on the data. We only center but don't scale the observations, because a constant variable cannot be normalized to have variance 1. Since all variables are measured on the same scale (values in $\{0, \ldots, 255\}$), scaling is not necessary anyways.

```
pca_dig <- prcomp(digits_train[ , -1], center  = TRUE, scale = FALSE)
```

To find out how many principal components we need to explain at least 85% of total variance, we use the importance matrix.

```
smry_pc <- summary(pca_dig)
which(smry_pc$importance[3, ] > .85)[1]
```

```
## PC49
##   49
```

The first 49 PCs explain 85% of total variance. We save the scores of the first 49 PCs and assemble a new dataframe containing these scores and the class label. As before, we name the column giving the class labels `V1`.

```
pc_scores <- pca_dig$x[ , 1:49]
dfpc <- data.frame(V1 = digits_train$V1, pc_scores )
dfpc[1:10, 1:6]
```

```
##    V1        PC1        PC2        PC3         PC4        PC5
## 1   0 1041.96812 -880.93526  415.72087 -284.602637  441.600480
## 2   4  -82.70198  460.88589 -579.16562   -6.872195  603.987536
## 3   1 -975.22103 -577.79951 -283.34872  437.767602  224.081738
## 4   1 -730.68625   91.60530  -39.00045 -534.956588 -378.467566
## 5   1 -831.47849  -53.62307 -134.83740 -608.756673   -4.235556
## 6   4 -257.84857  512.66492  213.24514  264.704010  758.469623
## 7   6  239.78384  691.29895  538.78315 -293.120231    1.308429
## 8   1 -859.63186 -104.08067 -130.70021 -634.502852  -74.338125
## 9   8 -456.80321 -425.00536  448.87708  701.538191  124.811682
## 10  6 -239.50543  209.83315  106.72439 -452.045813 -120.110100
```

Now we perform LDA on the PC scores.

```
lda_pc <- lda( V1 ~ . , data = dfpc)
```

We predict the classes for the training data and compute the confusion matrix.

```
pr_lda_pc <- predict(lda_pc, newdata = dfpc)
conf_train <- table(pr_lda_pc$class, dfpc$V1)
conf_train
```

```
##
##         0    1    4    6    8
##    0  968    0    0    8    9
##    1    0 1085   13    9   52
##    4    5    2  930    7   21
##    6    8    3   17  967   20
##    8    5   23    9   11  828
```

```
print(paste("The percentage of misclassified observations is",
(1- sum(diag(conf_train))/nrow(dfpc)  ) *100 , "%."))
```

```
## [1] "The percentage of misclassified observations is 4.44 %."
```

To classify the test dataset, the observations need to be transformed in the same way that the trainingsdata was transformed. Therefore we subtract the column means of the trainingsdata (this is the centering step of the PCA) and then rotate this 'centered' data according to the PC rotation matrix. In other words, we compute the principal component scores of the test dataset in the same way the principal component scores of the training data were computed.

```
dim(digits_test[ , -1])    # dimension of test dataset (without column giving the class label)
```

```
## [1] 3000  784
```

```
rotation_mat <- pca_dig$rotation[ , 1:49]    # first 49
dim(rotation_mat)
```

```
## [1] 784  49
```

```
# column means of trainingsdata can be found in pca_dig$center
test_cent <- digits_test[, -1] - matrix(rep( pca_dig$center, each = nrow(digits_test)  ), ncol = 784)

test_pc_scores <- as.matrix(test_cent) %*% rotation_mat

# easier: use predict()-function
test_pc_scores2 <- predict(pca_dig, newdata = digits_test[, -1] )[, 1:49]   # only need the first 49
identical(test_pc_scores, test_pc_scores2)
```

```
## [1] TRUE
```

```
dfpc_test <- data.frame( V1 = digits_test$V1, test_pc_scores )

dfpc_test[1:10 , 1:6]
```

```
##    V1         PC1       PC2        PC3       PC4         PC5
## 1   0  1420.43371 -711.5472  307.15633 -373.6414   597.58845
## 2   4   339.26084  436.8294 -914.48582 -126.8062   746.34049
## 3   0  1152.46955 -620.8421   27.12602 -571.1380   544.64091
## 4   4  -145.00246  341.0784 -528.51064  470.7555   273.69970
## 5   1  -964.68975 -245.6697   30.87218 -639.8138  -107.68208
## 6   4   -66.88842  305.2650 -491.60034  294.7320   367.84960
## 7   0  1013.03181 -517.0169 -881.45683 -484.5765  -451.70361
## 8   6   415.51698  703.4514 -599.51189 -113.5146   145.64693
## 9   1 -1094.14777 -644.9836  -27.55960 -166.1981    80.34767
## 10  8    50.35839   43.4387 -333.18104  642.8252 -1135.06763
```

Now we can predict the classes of the test data and compute the confusion matrix.

```
dfpc_test <- data.frame(cbind(digits_test$V1, test_pc_scores) )

pr_lda_test <- predict(lda_pc, newdata = dfpc_test)
# works also without the column giving the true class labels
pr_lda_test1 <- predict(lda_pc, newdata = dfpc_test[ , -1])
identical(pr_lda_test, pr_lda_test1)
```

```
## [1] TRUE
```

```
conf_test <- table(pr_lda_test$class, dfpc_test$V1)
conf_test
```

```
##
```

```
##       0   1   4   6   8
##   0 575   0   0   6   6
##   1   0 650   7   7  44
##   4   7   2 549   4  17
##   6  12   1  12 544   2
##   8   7  22   4  13 509
```

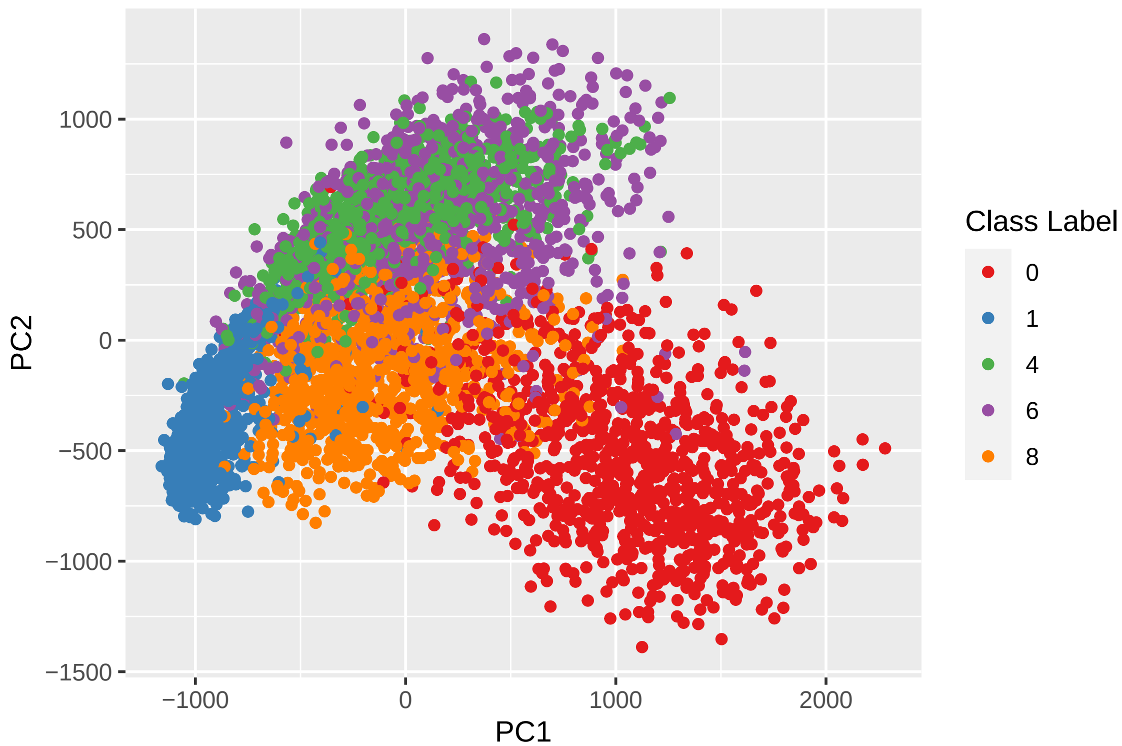
```
print( paste("The misclassification rate is",
round((1- sum(diag(conf_test))/nrow(dfpc_test))*100, 4),  "%"))
```

```
## [1] "The misclassification rate is 5.7667 %"
```

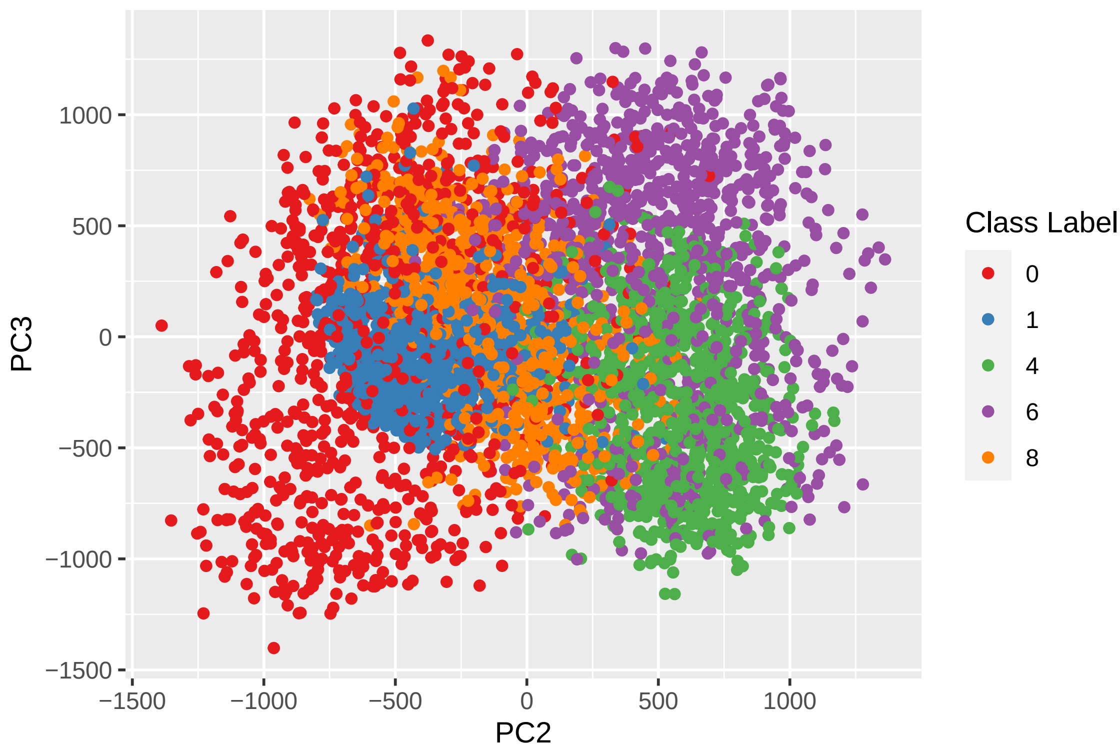
d) (1.5 points)

We use `ggplot()` because the base R plot uses white colour for some points when specifying `col = dfpc$V1`.

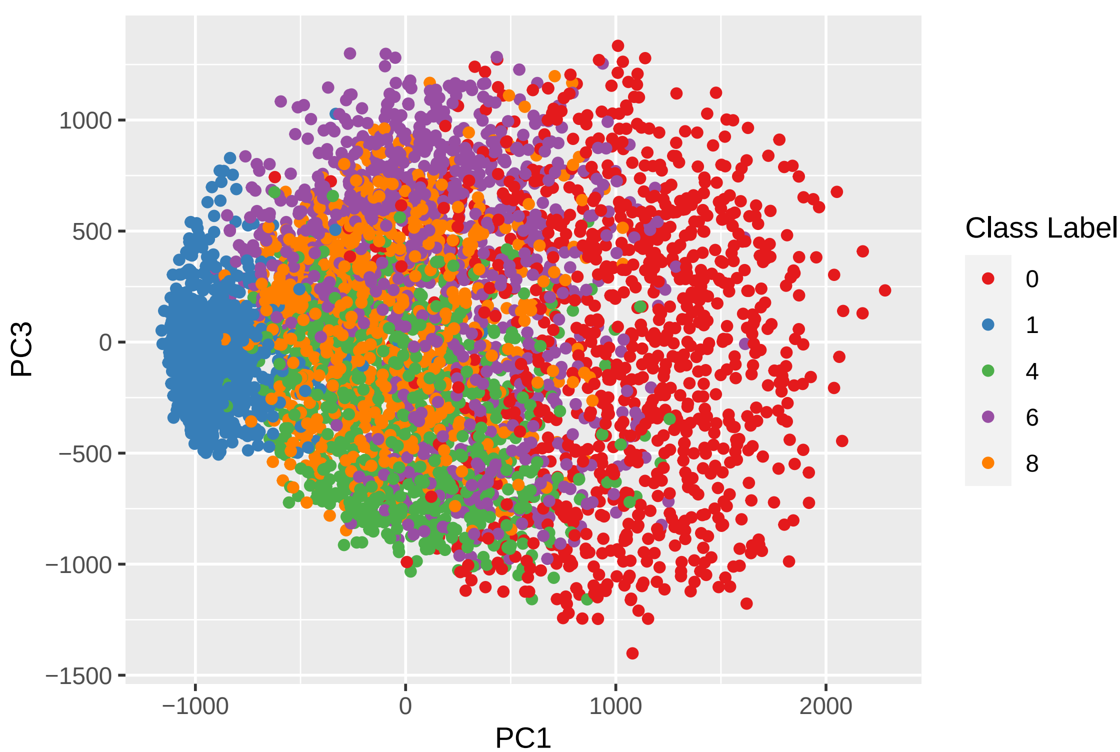
```
ggplot(dfpc, aes( x = PC1, y = PC2, colour = as.factor(V1))) +
  geom_point(size = 1.5) +
  scale_color_brewer(palette = "Set1")+
  labs(colour = "Class Label")
```




```
ggplot(dfpc, aes( x = PC2, y = PC3, colour = as.factor(V1))) +
  geom_point(size = 1.5) +
  scale_color_brewer(palette = "Set1")+
  labs(colour = "Class Label")
```




```
ggplot(dfpc, aes( x = PC1, y = PC3, colour = as.factor(V1))) +
  geom_point() +
  scale_color_brewer(palette = "Set1")+
  labs(colour = "Class Label")
```



Actually, some digits seem to have less variance, e.g. the digit 1.Therefore QDA might be a good idea, because it can account for class-specific covariance matrices.

```
qda_pc <- qda( V1 ~ . , data = dfpc)

pr_qda_pc <- predict(qda_pc, dfpc)

conf_train_qda <- table(pr_qda_pc$class, dfpc$V1)
conf_train_qda
```

```
##
##        0    1    4    6    8
##   0  983    0    0    6    2
##   1    0 1064    0    0    0
##   4    1    8  963    0    2
##   6    0    3    4  988    2
##   8    2   38    2    8  924
```

```
print( paste( "The misclassification rate for the training data based on QDA is",
round((1- sum(diag(conf_train_qda))/nrow(dfpc))*100, 4), "%"))
```

```
## [1] "The misclassification rate for the training data based on QDA is 1.56 %"
```

Indeed, that is less than for LDA. Let's check on the test data.

```
pr_qda_pc_test <- predict(qda_pc, dfpc_test)

conf_test_qda <- table(pr_qda_pc_test$class, dfpc_test$V1)
conf_test_qda
```

```
##
##       0   1   4   6   8
##   0 594   0   1   3   3
##   1   0 641   0   1   3
##   4   1   2 563   1   5
##   6   1   3   2 560   0
##   8   5  29   6   9 567
```

```
1- sum(diag(conf_test_qda))/nrow(dfpc_test)
```

```
## [1] 0.025
```

That is 2.5% of misclassified observations. Therefore QDA does indeed perform better than LDA, also on test data.