1 b) [0.5 points]

We define a function that takes μ_1, μ_2 and $\mathbf{\Sigma}$ as arguments and computes $p(1|2)$.

```r
miscl_theo <- function(mean1, mean2, sigma){
  mah_dist <- sqrt( t(mean1 - mean2) %*% solve(sigma) %*% (mean1-mean2))
  pnorm( -0.5* mah_dist)
}
```

We can now compute the theoretical probability of misclassification for the given values of μ_1, μ_2 and $\mathbf{\Sigma}$.

```r
mu1 <- c(1, -2)
mu2 <- c(2,0)
covSigma <- diag(c(1,1))


miscl_theo(mean1 = mu1, mean2 = mu2, sigma = covSigma)
```

```
##           [,1]
## [1,] 0.1317762
```

Thus, the probability of making a wrong classificaion is 13.1776239%.

We draw a sample from the corresponding Gaussian mixture model.

```r
samp1 <- mvtnorm::rmvnorm(100, mean = mu1, sigma = diag(c(1,1)))
samp2 <-  mvtnorm::rmvnorm(100, mean = mu2, sigma = diag(c(1,1)))

gmm_sample <- data.frame(rbind(samp1, samp2) , Class = rep(1:2, each = 100))
```

Since the components are independent here (covariance is 0, for normal variables that coincides with independence) one doesn't need the `mvtnorm` package. One can also use base R's `rnorm()`, which samples from the univariate normal distribution, as follows. First, for each component, sample 200 observations from $\mathcal{N}(0,1)$:

```r
gmm_sample <- data.frame( X1 = rnorm(200), X2 = rnorm(200), Class = rep(1:2, each = 100))
```

The joint distribution is then $\mathcal{N}(0, I_2)$ with I_2 the identity matrix on $\mathbb{R}^2$.

Now we adjust the means (note that for $X \sim \mathcal{N}(0,1)$ one has $\mu + X \sim \mathcal{N}(\mu, 1)$).

```r
gmm_sample[1:100, 1] <- gmm_sample[1:100, 1] + 1
gmm_sample[1:100, 2] <- gmm_sample[1:100, 2] - 2
gmm_sample[101:200, 1] <- gmm_sample[101:200, 1] + 2
```

Now we write a function that estimates the out-of-sample error based on leaving-one-out cross-validation (LOOCV). The function's argument `sample_data` needs to have a column named `Class` containing the class labels.

```r
compute_loocv <- function(j, sample_data = gmm_sample){
  lda_loo <- MASS::lda(Class ~ ., data = sample_data[-j, ])  # fit LDA to all but j-th observations
  # return 1 (FALSE) if wrong class is predicted, else return 0 (TRUE)
  predict(lda_loo, newdata = sample_data[j, ])$class != sample_data$Class[j]
}
```

Then we apply the function for each `j` in `1:200` (i.e. each row is left out once), and then we average

```r
mean(sapply( 1:200 , compute_loocv))
```

```
## [1] 0.12
```

The LOOCV misclassification rate is 0.12.

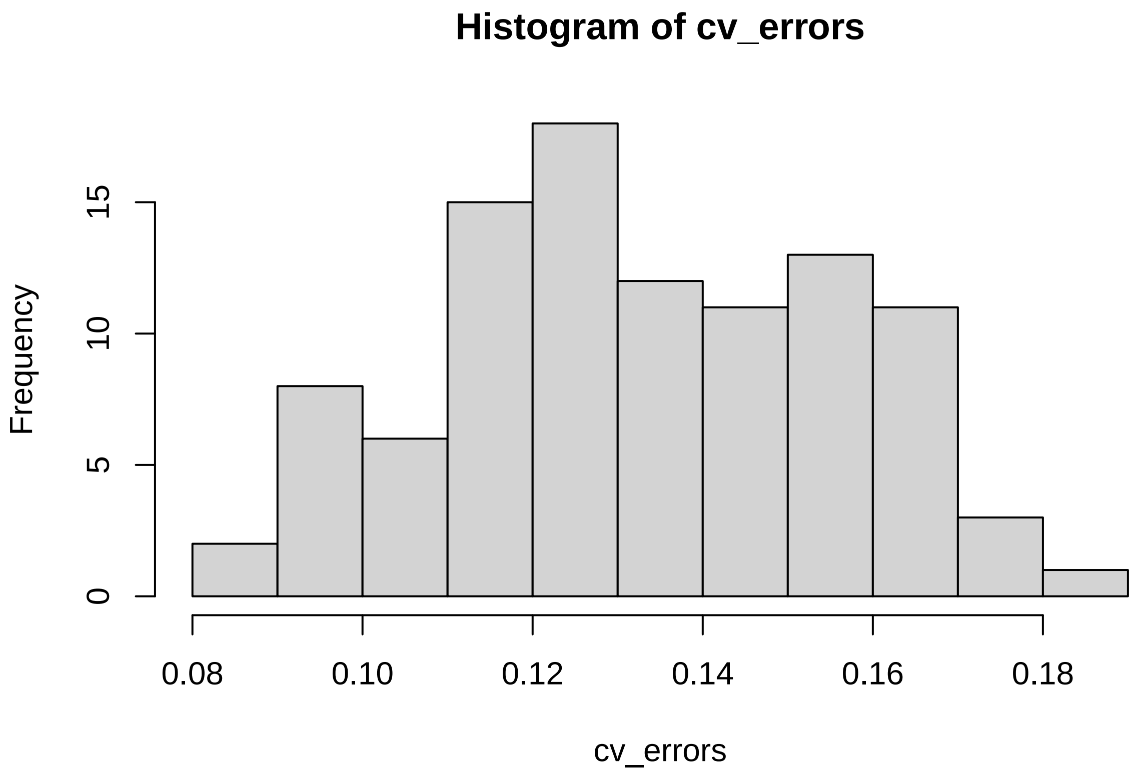
c) [0.5 points]

We write a function that performs one iteration, i.e. sampling and computing LOOCV error. The arguments of the function are `seed`, which is set at the beginning of each iteration to make things reproducible, the mean vectors `mean1` and `mean2` as well as the covariance matrix. The default value of the latter is the identity matrix on $\mathbb{R}^2$.

```r
iteration <- function(seed , mean1 , mean2, sigma = diag(c(1,1))){
  set.seed(seed)
  samp1 <- mvtnorm::rmvnorm(100, mean = mu1, sigma = sigma )
  samp2 <-  mvtnorm::rmvnorm(100, mean = mu2, sigma = sigma)

  gmm_samp <- data.frame(rbind(samp1, samp2) , Class = rep(1:2, each = 100))

  mean(sapply( 1:200 , compute_loocv, sample_data = gmm_samp)) # this is what the function returns

}
```

Then we make 100 iterations, i.e. we apply the above functions with seed from 1 to 100 and make a histogram, compute mean and standard deviatin.

```r
cv_errors <- sapply(1:100,  function(x){iteration(seed = x, mean1 = mu1,
                                                 mean2 = mu2, sigma = covSigma)})
hist(cv_errors)
```



```r
mean(cv_errors)
```

```
## [1] 0.13545
```

```
sd(cv_errors)
```

```
## [1] 0.02371011
```

```
miscl_theo(mean1 = mu1, mean2 = mu2, sigma = covSigma)
```

```
##           [,1]
## [1,] 0.1317762
```

We see that the mean is pretty close (deviates in third decimal place) and that the standard deviation is small.

Exercise 2

First of all, we load the required packages and the `penguins` data.

In the text it says to only keep columns containing measurements of quantitative variables, as well as the column containing the penguin's species. Later we will need the column `sex`, too (I forgot to mention it here, sorry). Further, we want to keep only the rows with complete observations on these variables. We generate a new dataframe following these directions.

```
pengus <- dplyr::select(penguins, -c("year", "island"))
# only complete cases
pengus <- pengus[complete.cases(pengus), ]
```

a) [0.5 points]

Now we randomly sample 80% of the data as training data. Then we look at the proportions of the different species within the training data.

```
set.seed(4017)
# sample indices of rows that make up the training data
ind <- sample(nrow(pengus), nrow(pengus)*.8 )

# training data
trainsamp <- pengus[ind , ]
# distribution of species within training data
table(trainsamp$species)/nrow(trainsamp)
```

```
##
##    Adelie Chinstrap    Gentoo
## 0.4774436 0.1729323 0.3496241
```

The test data consists of the observations that are not part of the training data. We compute the distribution of species within the test data.

```
testsamp <- pengus[-ind, ]

table(testsamp$species)/nrow(testsamp)
```

```
##
##    Adelie Chinstrap    Gentoo
## 0.2835821 0.3283582 0.3880597
```

We see that the distributions of the species differ quite drastically for the two datasets. This can affect the model's performance, when the true distribution of classes is not represented well in the training data. Since random splitting into training and test data doesn't guarantee the latter, one should be careful when only using one train-test-split.

We compare to the distribution of classes for the whole dataset.

```
table(pengus$species)/nrow(pengus)
```

```
##
##    Adelie Chinstrap    Gentoo
## 0.4384384 0.2042042 0.3573574
```

b) [0.5 points]

There are many possible ways, and of course there are also packages that provide functions for this kind of sampling. Here, we present one solution that uses some self-written functions.

First, we split the dataframe into a list of three dataframes, each containing the measurements belonging to one species.

```
pengus_split <- split(pengus, pengus$species)
pengus_split
```

```
## $Adelie
## # A tibble: 146 x 6
##    species bill_length_mm bill_depth_mm flipper_length_mm body_mass_g sex
##    <fct>            <dbl>         <dbl>             <int>       <int> <fct>
##  1 Adelie            39.1          18.7               181        3750 male
##  2 Adelie            39.5          17.4               186        3800 female
##  3 Adelie            40.3          18                 195        3250 female
##  4 Adelie            36.7          19.3               193        3450 female
##  5 Adelie            39.3          20.6               190        3650 male
##  6 Adelie            38.9          17.8               181        3625 female
##  7 Adelie            39.2          19.6               195        4675 male
##  8 Adelie            41.1          17.6               182        3200 female
##  9 Adelie            38.6          21.2               191        3800 male
## 10 Adelie            34.6          21.1               198        4400 male
## # ... with 136 more rows
##
## $Chinstrap
## # A tibble: 68 x 6
##    species   bill_length_mm bill_depth_mm flipper_length_mm body_mass_g sex
##    <fct>              <dbl>         <dbl>             <int>       <int> <fct>
##  1 Chinstrap           46.5          17.9               192        3500 female
##  2 Chinstrap           50            19.5               196        3900 male
##  3 Chinstrap           51.3          19.2               193        3650 male
##  4 Chinstrap           45.4          18.7               188        3525 female
##  5 Chinstrap           52.7          19.8               197        3725 male
##  6 Chinstrap           45.2          17.8               198        3950 female
##  7 Chinstrap           46.1          18.2               178        3250 female
##  8 Chinstrap           51.3          18.2               197        3750 male
##  9 Chinstrap           46            18.9               195        4150 female
## 10 Chinstrap           51.3          19.9               198        3700 male
## # ... with 58 more rows
##
## $Gentoo
## # A tibble: 119 x 6
##    species bill_length_mm bill_depth_mm flipper_length_mm body_mass_g sex
##    <fct>            <dbl>         <dbl>             <int>       <int> <fct>
##  1 Gentoo            46.1          13.2               211        4500 female
##   2 Gentoo             50             16.3                230         5700 male
##   3 Gentoo             48.7           14.1                210         4450 female
##   4 Gentoo             50             15.2                218         5700 male
##   5 Gentoo             47.6           14.5                215         5400 male
##   6 Gentoo             46.5           13.5                210         4550 female
##   7 Gentoo             45.4           14.6                211         4800 female
##   8 Gentoo             46.7           15.3                219         5200 male
##   9 Gentoo             43.3           13.4                209         4400 female
##  10 Gentoo             46.8           15.4                215         5150 male
## # ... with 109 more rows
```

Now we write a function that samples a proportion of `train_prop` from the `1:nrow(data)`:

```
get_train_index <- function(data, train_prop = 0.8, seed = 1){
  set.seed(seed)  # set seed to make reproducible
  ndata <- nrow(data)
  sample(ndata, floor(ndata*train_prop))
}
```

The function that returns training data based on rows sampled with `get_train_index()`:

```
get_train_data <- function(data, train_prop = 0.8, seed = 1){
  samp_ind <- get_train_index(data = data, train_prop = train_prop, seed = seed)
  train_data <- data[samp_ind, ]
  return(train_data)
}
```

And function that returns test data based on rows sampled in `get_train_index`. Note that, when applying these functions, the same seed must be used for the test data as for the training data.

```
get_test_data <- function(data, train_prop = 0.8, seed = 1){
  samp_ind <- get_train_index(data = data, train_prop = train_prop, seed = seed)
  test_data <- data[-samp_ind, ]
  return(test_data)
}
```

We use the function `purrr::map_dfr` which maps the function `get_train_data()` to the list elements in `pengus_split` and binds the resulting dataframes by rows (dfr means dataframe rowbind).

```
pengus_train <- purrr::map_dfr(pengus_split, get_train_data)
pengus_test <- purrr::map_dfr(pengus_split, get_test_data)
pengus_train
```

```
## # A tibble: 265 x 6
##    species bill_length_mm bill_depth_mm flipper_length_mm body_mass_g sex
##    <fct>            <dbl>         <dbl>             <int>       <int> <fct>
##   1 Adelie             45.8           18.9                197         4150 male
##   2 Adelie             38.1           17.6                187         3425 female
##   3 Adelie             36             17.9                190         3450 female
##   4 Adelie             34.4           18.4                184         3325 female
##   5 Adelie             39             17.5                186         3550 female
##   6 Adelie             35.7           18                  202         3550 female
##   7 Adelie             35.3           18.9                187         3800 female
##   8 Adelie             45.6           20.3                191         4600 male
##   9 Adelie             42.1           19.1                195         4000 male
##  10 Adelie             39.2           19.6                195         4675 male
## # ... with 255 more rows
```

`map_dfr()` is kind of advanced, you can also use `lapply()` and assemble by hand.

```
table(pengus_train$species)/nrow(pengus_train)
```

```
##
##    Adelie Chinstrap    Gentoo
## 0.4377358 0.2037736 0.3584906
```

```
table(pengus_test$species)/nrow(pengus_test)
```

```
##
##    Adelie Chinstrap    Gentoo
## 0.4411765 0.2058824 0.3529412
```

c) [0.5 points]

```
lda_peng <- lda(species ~ . - sex, data = pengus_train)  # . - sex uses all columns but `sex`
pred_peng_test <- predict(lda_peng, newdata = pengus_test)
conf_test <- table(pred_peng_test$class, pengus_test$species)
1- sum(diag(conf_test))/nrow(pengus_test)
```

```
## [1] 0.01470588
```

The misclassification error on the test set is 0.0147059.

d) [1 point]

```
cv_kfold <- function(K, data){

  n_data <- nrow(data)

  # K must be an integer between 1 and nrow(data)
  if( !( K == round(K)) |  K == 0 | K > n_data){stop( "Choose different value for K.")}

  n_subsamp <- floor(n_data/K)     # length of each of the K subsamples

  data <- data[sample(n_data), ]  # shuffle data

  # assign each observation to one of the K subsamples, column 'K' specifies the subsample
  # if n_data is not a multiple of K, assign the last n_data- K*n_subsamp observations randomly

  if( n_subsamp == n_data/K) {
      data$K <- rep(1:K, each = n_subsamp)
    }
  else{
    data$K <- c(rep(1:K, each = n_subsamp), sample(1:K, (n_data - K*n_subsamp)))
  }

  # initialise vector that stores the misclassification rates
  cv_errors <- numeric(K)

  for( k in 1:K){
    # training data in iteration k
    data_train <- subset(data, !( K == k))   # leave out k-th subset
    data_train <- dplyr::select(data_train, - "K")  # get rid of column specifying the subset
```

```r
    # test data in iteration k
    data_test <-  subset(data, K == k)        # k-th subset
    data_test <- dplyr::select(data_test, - "K") # get rid of column specifying the subset

    lda_fit <- lda(species ~ .,
                   data = data_train)    # fit lda to training data
    test_pred <- predict(lda_fit, newdata = data_test) # predict class for test data

    conf_mat <- table(test_pred$class, data_test$species)  # compute confusion matrix

    cv_errors[k] <- 1- sum(diag(conf_mat))/nrow(data_test) # misclassification rate in iteration k
  }
  # return the average misclassification rate and also the vector of misclassification rares,
  # in case one is interested in variance
  return(list(avg_misclass = mean(cv_errors),  errors = cv_errors))
}
```

We apply the function:

```r
cv_kfold(K =  10, data = pengus)
```

```
## $avg_misclass
## [1] 0.006060606
##
## $errors
##  [1] 0.00000000 0.00000000 0.00000000 0.03030303 0.00000000 0.00000000
##  [7] 0.03030303 0.00000000 0.00000000 0.00000000
```

e) [1.5 points]

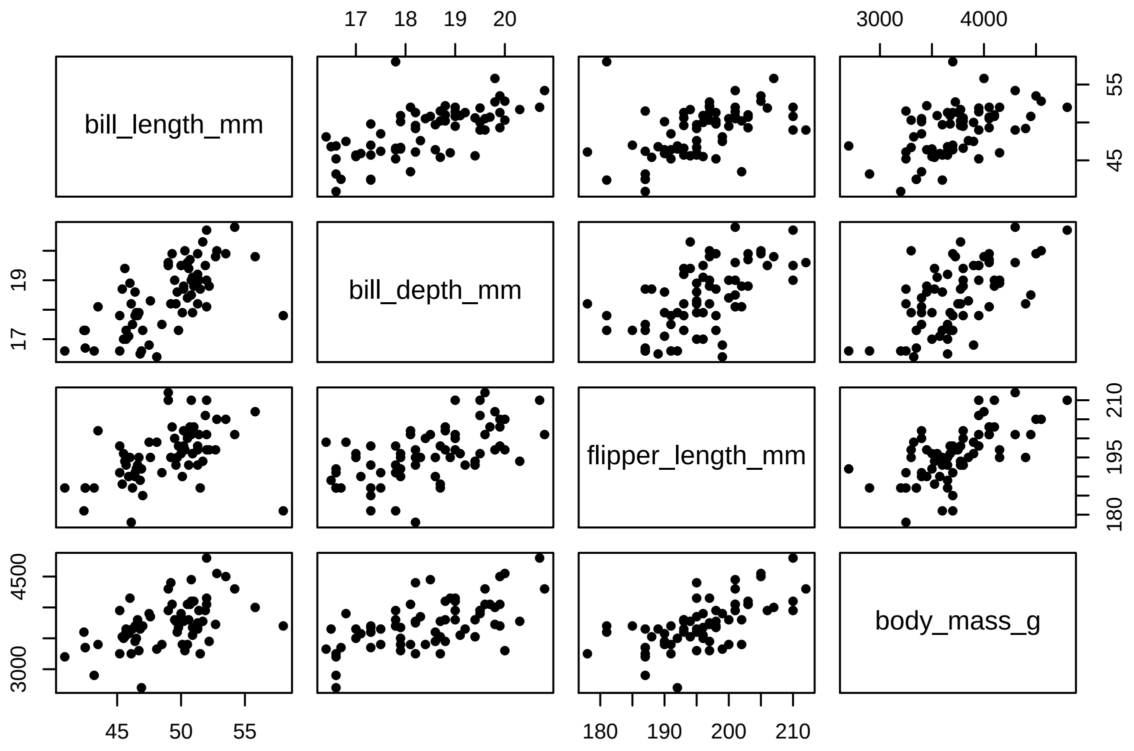
First, we subset the measurements of Chinstrap species.

```r
pengus_chin <- subset(pengus, species == "Chinstrap")
pengus_chin
```

```
## # A tibble: 68 x 6
##    species   bill_length_mm bill_depth_mm flipper_length_mm body_mass_g sex
##    <fct>              <dbl>         <dbl>             <int>       <int> <fct>
##  1 Chinstrap           46.5          17.9               192        3500 female
##  2 Chinstrap           50            19.5               196        3900 male
##  3 Chinstrap           51.3          19.2               193        3650 male
##  4 Chinstrap           45.4          18.7               188        3525 female
##  5 Chinstrap           52.7          19.8               197        3725 male
##  6 Chinstrap           45.2          17.8               198        3950 female
##  7 Chinstrap           46.1          18.2               178        3250 female
##  8 Chinstrap           51.3          18.2               197        3750 male
##  9 Chinstrap           46            18.9               195        4150 female
## 10 Chinstrap           51.3          19.9               198        3700 male
## # ... with 58 more rows
```

We can make a pairwise scatter plot to check linear relation of predictors and target.

```r
plot(pengus_chin[ , -c(1,6)], pch = 16)
```



```r
# also check correlation matrix
cor(pengus_chin[ , -c(1, 6)])
```

```
##                   bill_length_mm bill_depth_mm flipper_length_mm body_mass_g
## bill_length_mm         1.0000000     0.6535362         0.4716073   0.5136383
## bill_depth_mm          0.6535362     1.0000000         0.5801429   0.6044983
## flipper_length_mm      0.4716073     0.5801429         1.0000000   0.6415594
## body_mass_g            0.5136383     0.6044983         0.6415594   1.0000000
```

Assumption of linear relation seems alright.

Now we fit the linear models and estimate the out-of-sample error based on leaving-one-out cv. The Mean squared error is the squared error here, because in each iteration we only have one predicted value.

First model: only bill depth as explanatory variable

```r
compute_loocv1 <- function(j){
  lm_loo <- lm( bill_length_mm ~ bill_depth_mm,   data = pengus_chin[-j ,  ]) # leave out j-th observat
  yhat <- predict(lm_loo, newdata = pengus_chin[j, ]) # predict bill_length based on model fit
  (pengus_chin$bill_length_mm[j] - yhat)^2   # return squared error
}
```

Apply function to all indices in `1:nrow(pengus)`, i.e. leave out each row once, then compute the average.

```r
cv_mse1 <- mean(sapply(1:nrow(pengus_chin), compute_loocv1))
cv_mse1
```

```
## [1] 6.638773
```

Second model: body mass and bill depth as explanatory variables.

```
compute_loocv2 <- function(j){
  lm_loo <- lm( bill_length_mm ~  body_mass_g + bill_depth_mm,  data = pengus_chin[-j ,  ]) # leave out
  yhat <- predict(lm_loo, newdata = pengus_chin[j, ]) # predict bill_length based on model fit
  (pengus_chin$bill_length_mm[j] - yhat)^2   # return squared error
}

cv_mse2 <- mean(sapply(1:nrow(pengus_chin), compute_loocv2))
cv_mse2
```

```
## [1] 6.524676
```

Third model: all three variables as predictors

```
compute_loocv3 <- function(j){
  lm_loo <- lm( bill_length_mm ~ flipper_length_mm + body_mass_g +  bill_depth_mm ,
                data = pengus_chin[-j ,  ]) # leave out j-th observation
  yhat <- predict(lm_loo, newdata = pengus_chin[j, ]) # predict bill_length based on model fit
  (pengus_chin$bill_length_mm[j] - yhat)^2   # return squared error
}

cv_mse3 <- mean(sapply(1:nrow(pengus_chin), compute_loocv3))
cv_mse3
```

```
## [1] 7.096585
```

Compare the MSEs:

```
cv_mse2 < cv_mse1
```

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

```
cv_mse3 < cv_mse2
```

```
## [1] FALSE
```

```
cv_mse3 < cv_mse1
```

```
## [1] FALSE
```

Model 2 has smallest MSE. However, reduction of MSE from model 2 to model 1 is only marginal. Therefore, one would probably go for model 1 (also ok if you chose model 2).