Amel Bejtic: 202305234
Jens Christian Andersen: 202305285

Machine Learning - Compulsory assignment

Problem 1

1.A

Code

```
# Packages

library(forecast)  #  accuracy()

library(glmnet)                    #Ridge / LASSO

# 0. Load BS pre-process data

# Reading College-data. We make first column the name for the different rows, as it's the name of the universities.

data <- read.csv("College.csv", row.names = 1)

# The column "Private" is a binary variable, so we turn it into a dummy-variable which R can read

data$Private <- ifelse(data$Private == "Yes", 1, 0)


# 1. Train/test split (used in both A and B)

set.seed(1) # setting the seed so we can re-run the code.

n <- nrow(data)

train_ind <- sample(1:n, n * 0.6)  # splitting the data into train/test data. Train 60%


train_data <- data[train_ind, ]

test_data  <- data[-train_ind, ]


n_p <- nrow(test_data)   # number of observations in test


# 2.  Predicting APPS

# Using OLS-model on traindata

reg_ols <- lm(Apps ~ ., data = train_data)

summary(reg_ols)
```

Amel Bejtic: 202305234
Jens Christian Andersen: 202305285

```
# Prediction on testdata

pred_ols <- predict(reg_ols, newdata = test_data)

# 3. Performance measure

accuracy_ols <- forecast::accuracy(pred_ols, test_data$Apps)

accuracy_ols

# Coomparing our test to the naive benchmark

nb_pred <- rep(mean(train_data$Apps), n_p)

accuracy_nb <- forecast::accuracy(nb_pred, test_data$Apps)

accuracy_nb
```

Choice of model:

The predictor, Apps, is a continuous variable which makes a regression model useable. With n(777) > p(17), we have a low-dimensional dataset, where OLS performs better when the number of predictors is small relative to the sample size. As the goal is also just prediction, OLS is a suited model for this problem.

Answer:

The model was trained on 60% of the data, and we see it working significantly better than the naïve benchmark where we have a smaller average bias(ME), it's errors are smaller on average(RMSE) and a much smaller percentage of bias (MPE). Most critical measure would be considered to be RMSE, as we want to figure where we find smallest errors on average. We also compared to the naïve benchmark, just to showcase it works much better than "your first guess"

In the model, several predictors are highly correlated, which isn't a problem as we are only focusing on prediction. When p is small relative to n, multicollinearity doesn't really affect predictive performance as much as it affects interpretability.

OLS:

```
                ME     RMSE      MAE       MPE     MAPE
Test set -115.1038 1060.416 650.6231 0.1534152 38.28258
```

Naïve benchmark:

```
                ME    RMSE      MAE       MPE     MAPE
Test set -198.4695 3576.14 2541.052 -207.1778 232.1215
```

Amel Bejtic: 202305234
Jens Christian Andersen: 202305285

1.B

Code

 # 1. Designmatrix with all interactions being (.^2)

#model_matrix removes the intercept column. The function ensures our data is mathematically readable by handling categorical variables and leaving continuous variables as they are. Any non-binary factor variable is converted into a dummy variable. The (.^2) gives us a new column with the interactions of all variables, so A x B, A x C etc.

```
X_train <- model.matrix(Apps ~ .^2, data = train_data)[ , -1]

X_test  <- model.matrix(Apps ~ .^2, data = test_data)[ , -1]

Y_train <- train_data$Apps

Y_test  <- test_data$Apps
```

 # 2. Cross-validation for optimal lambda (LASSO: alpha = 1)

```
cv_out <- cv.glmnet(X_train, Y_train, alpha = 1, nfolds = 5) #cv.glmnet looks through a sequence of possible values for lambda and tells us which one results in the lowest prediction error on new data. We divide it here into 5 randomly equalled folds - considered enough. Eveytime one fold will be set aside as validation set, the remaining 4 will be used to train the model with different values of lambda. The prediction error is calculated by the single validation set.

plot(cv_out)

lambda_opt <- cv_out$lambda.min

lambda_opt
```

 # 3. Final LASSO-model with optimal lambda

```
reg_lasso <- glmnet(X_train, Y_train, alpha = 1, lambda = lambda_opt)

coef(reg_lasso)  # looking at all the coefficients, where we see many will be 0, which is a result of the shrinkage. The model forces coefficients to be within a constraint, where our optimal solution is found on this edge where our model touches it. This is the essence of the model as it does feature selection to stay within the constraint.
```

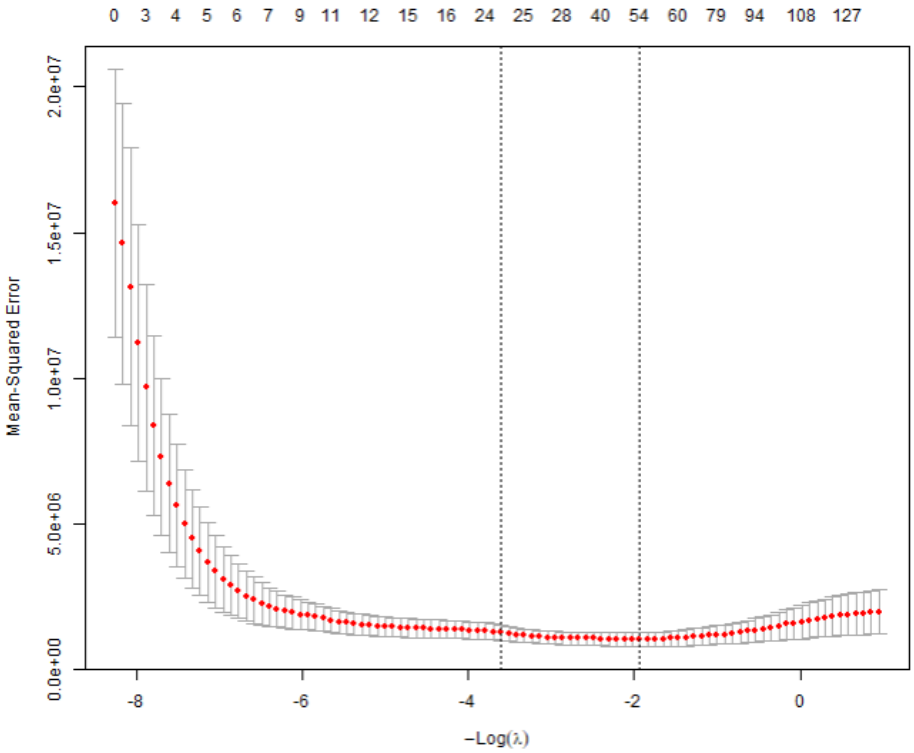
 # 4. Prediction og performance for LASSO

```
pred_lasso <- as.vector(predict(reg_lasso, newx = X_test)) # converted to vector, as it the standard format in the glmnet package.

accuracy_lasso <- forecast::accuracy(pred_lasso, Y_test)

accuracy_lasso
```

Amel Bejtic: 202305234
Jens Christian Andersen: 202305285

Choice of model:

We chose LASSO because of the shrinkage effect of the model. The original data itself has 17 predictors, and after the interactions, we created (17 x 16) / 2 = 136 terms more. With over 150 predictors we suddenly work in a high dimensional setting, where OLS doesn't work as well, causing high variance and overfitting. LASSO adds a penalty (alpha = 1) that shrinks some coefficients to exactly zero through feature selection.



LASSO:

	ME	RMSE	MAE	MPE	MAPE
Test set	-49.1563	867.3451	467.6234	-3.417096	18.88178

Answer

When we expand the feature space, we see that OLS cannot handle this and overfits the training data. This is reflected in the test performance, where RMSE is 1060, where we have 867 for lasso. For regression RMSE is a good measure because it penalizes larger errors more. But we also see a better performance in the other metrics. Lasso clearly outperformances OLS.

Amel Bejtic: 202305234
Jens Christian Andersen: 202305285

Problem 2

Code for A and B

```
# 0. libraries

library(caret)   # stratified split, preProcess, train(), confusionMatrix


# 0.1. Load and preprocess data

bank <- read.csv("UniversalBank.csv")

# dropping ID-variables (not predictive variables). Zip.code would just make noise in the data unless we chose to distribute in some way, but since it's not part of the exercise, we simply just remove it.

bank <- subset(bank, select = -c(ID, ZIP.Code))


# We make sure our "goal-variable" is saved as an integer as expects numbers, not text. Afterwards we also create a new column 'loanfactor' which takes the numbers 0/1 from personal.loan and makes them categorical with the names no/yes. This is because caret -> confusionmatrix requires input to be factors. So we now have two versions.

bank$Personal.Loan <- as.integer(bank$Personal.Loan)

bank$LoanFactor    <- factor(bank$Personal.Loan, levels = c(0, 1),

                             labels = c("No", "Yes"))

# Looking at the Y-distribution

cat("Outcome distribution (counts):\n") # writing the title and printing numbers of 0 and 1's

print(table(bank$Personal.Loan))

cat("\nOutcome distribution (proportions):\n") # same as before, just calculated as percentages

print(prop.table(table(bank$Personal.Loan)))


# 2. Doing a stratified test/train split

set.seed(1) # making sure the split is the same everytime we run the code

train_ind  <- createDataPartition(bank$Personal.Loan, p = 0.6, list = FALSE) # this makes sure that the distribution is the same. So it basically makes sure that if there are 10% "Yes" in the whole data-set, there is also exactly  10% "yes" in the 60%

train_data <- bank[train_ind, ]

test_data  <- bank[-train_ind, ]
```

```
# saving the variables in separate vectors

y_train_num <- train_data$Personal.Loan

y_test_num  <- test_data$Personal.Loan

y_train_fac <- train_data$LoanFactor

y_test_fac  <- test_data$LoanFactor
```

 # 3. Logistic Regression (baseline model)

```
# We use all the predictors, except for the factor-version of the outcome. Predicting personal.loan based on training data

log_mod <- glm(Personal.Loan ~ . - LoanFactor,

               data   = train_data,

               family = "binomial")


summary(log_mod) # visualizing


# Predicted probabilities & classes (cut-off 0.5) on test-data. Type = response makes sure we probabilities as outcome

prob_log <- predict(log_mod, newdata = test_data, type = "response")

pred_log_num <- ifelse(prob_log > 0.5, 1, 0) # if probability > 50%, we guess 1(loan) else 0.

pred_log_fac <- factor(pred_log_num, levels = c(0, 1), labels = c("No", "Yes")) # translating to 0/1 again so that we can compare.


# Confusion matrix with guess and true + keynumbers. Positive = Yes tells R that the interesting outcome is Yes - meaning they want the loan

cm_log <- confusionMatrix(pred_log_fac, y_test_fac, positive = "Yes")

cm_log # printing


log_err  <- mean(pred_log_num != y_test_num) # calculating errorrate. The mean is the percentage of errors

log_sens <- cm_log$byClass["Sensitivity"]

log_spec <- cm_log$byClass["Specificity"]
```

 # 4. k-NN med tuning via k-fold cross-validation (caret::train)

```
# Setting predictors - all except outcome and factorversion as we only want input-variables

X_cols <- setdiff(names(train_data), c("Personal.Loan", "LoanFactor"))

# making 2 matrices only containing X-variables

train_X <- train_data[ , X_cols]

test_X  <- test_data[  , X_cols]


set.seed(1) # resetting the randomness as CV involves a random division of folds

# We here use the caret::train with the method KNN

knn_fit <- train(                  # naming the function train

                  x = train_X,     # input

                  y = y_train_fac,                  # output-variable

                  method = "knn",

                  preProcess = c("center", "scale"),                # standardizing all X variables based on train. So we minus all columns with the mean and divide by standard deviation. This makes higher and smaller numbers weight the same, when KNN measures the distance. Family is an example of small number and income high.

                  tuneGrid = data.frame(k = 1:15), # meaning we test k = 1, …, 15

                  trControl = trainControl(method = "cv", number = 5)

) # Caret trains the model 5 times for each k (4 train and test on 1), and in the end choses the k with lowest mean CV-error, which also means we don’t need to split into an extra separate validation set


knn_fit        # visualization

k_opt <- knn_fit$bestTune$k

cat("\nOptimal k chosen by 5-fold CV:", k_opt, "\n\n")        # getting the optimal k and printing it.


# Prediction on test-data with optimal/chosen k.

pred_knn_fac <- predict(knn_fit, newdata = test_X)

pred_knn_num <- ifelse(pred_knn_fac == "Yes", 1, 0)          # again converting. So that we can calculate with the column again.


# Confusion matrix + keynumbers

cm_knn <- confusionMatrix(pred_knn_fac, y_test_fac, positive = "Yes")

cm_knn
```

```
# saving errorrate, sensitivity and specificity manually

knn_err  <- mean(pred_knn_num != y_test_num)

knn_sens <- cm_knn$byClass["Sensitivity"]

knn_spec <- cm_knn$byClass["Specificity"]
```

 # 5. Doing a naive rule

```
# We first make a list only 0's as long as y_test_num (tested persons). Which means we here we guess 0=no on all.

naive_pred_num <- rep(0, length(y_test_num))

naive_pred_fac <- factor(naive_pred_num, levels = c(0, 1),

                         labels = c("No", "Yes"))          # chancing no -> 0

cm_naive <- confusionMatrix(naive_pred_fac, y_test_fac, positive = "Yes")
                          # testing how good "always no" strategy is.

# saving the keynumbers

naive_err  <- mean(naive_pred_num != y_test_num)

naive_sens <- cm_naive$byClass["Sensitivity"]

naive_spec <- cm_naive$byClass["Specificity"]
```

 # 6. Comparison

 # 6.1 printing the numbers for naïve rule

```
cat("Naive rule (always 'No'):\n")

cat("  Error rate :", round(naive_err, 4),

    " | Sensitivity :", round(naive_sens, 4),

    " | Specificity :", round(naive_spec, 4), "\n\n")
```

 # 6.2 Printing for logistic regression

```
cat("Logistic regression:\n")

cat("  Error rate :", round(log_err, 4),

    " | Sensitivity :", round(log_sens, 4),

    " | Specificity :", round(log_spec, 4), "\n\n")
```

 # 6.3 Printing for KNN

```
cat("k-NN (tuned with 5-fold CV, k_opt =", k_opt, "):\n")

cat("  Error rate :", round(knn_err, 4),
```

```
  " | Sensitivity :", round(knn_sens, 4),
  " | Specificity :", round(knn_spec, 4), "\n")
```

2.A

Choice of model

The goal is to predict if a customer will accept a personal loan, which is a binary classification problem. So we have to use a supervised learning algorithm. We have multiple options for such a problem in this course, which consists of logistic regression, k-NN and the bayes classifier. We decided to run the logistic model. It models the probability of acception the loan directly, can handle continuous and categorical predictors. It also offers interpretable results, which could provide valuable insights into what drives the classifications of loan takers.

We also run the k-NN, which is a non parametric alternative. It uses a distance based method, so we standardise the predictors and select the optimal value of k using cross validation.

Amel Bejtic: 202305234
Jens Christian Andersen: 202305285

2.B

```
> cat("Naive rule (always 'No'):\n")
Naive rule (always 'No'):
> cat("  Error rate :", round(naive_err, 4),
+     " | Sensitivity :", round(naive_sens, 4),
+     " | Specificity :", round(naive_spec, 4), "\n\n")
  Error rate : 0.1025  | Sensitivity : 0  | Specificity : 1

> cat("Logistic regression:\n")
Logistic regression:
> cat("  Error rate :", round(log_err, 4),
+     " | Sensitivity :", round(log_sens, 4),
+     " | Specificity :", round(log_spec, 4), "\n\n")
  Error rate : 0.0505  | Sensitivity : 0.6439  | Specificity : 0.9844

> cat("k-NN (tuned with 5-fold CV, k_opt =", k_opt, "):\n")
k-NN (tuned with 5-fold CV, k_opt = 1 ):
> cat("  Error rate :", round(knn_err, 4),
+     " | Sensitivity :", round(knn_sens, 4),
+     " | Specificity :", round(knn_spec, 4), "\n")
  Error rate : 0.0435  | Sensitivity : 0.6976  | Specificity : 0.9861
```

We look at the balance of the Y, and see that it's heavely weighted towards more 0's, in this case saying no to the loan. This imbalance makes the error-rate misleading. Considering the naive rule, it has a fairly low error rate (0,1025), but with a sensitivity at 0, meaning it can't find any customers. So even though it has an error rate at 10,25%, the model gives no value for the bank.

The problem doesnt state anything about interpretability of what drives predictions, so a solution could be to run multiple models and choose the best one purely based on predictive performance. Given the banks goal to find customers who will accept the loan, we consider sensitivity as the most critical metric. It measures the ability to identify "yes" customers correctly. The cost of false negative is considered higher than false positive as false positive means no possible customer.

Comparing k-NN and logistic regression, k-NN outperforms. Both are significantly better than the naive rule, but k-NN achieves a higher sensitivity and has lowest error-rate. Therefore we also choose k-NN as a model.

Amel Bejtic: 202305234
Jens Christian Andersen: 202305285

Problem 3

Code

```
install.packages("devtools", dependencies = T)

library(devtools)

install_version("space", version = "0.1-1.1", repos = "http://cran.us.r-project.org")


library(space)    # SPACE estimator

library(igraph)   # network + spectral clustering

library(Matrix)   # sparse matrices etc.


# (0) Load data and patch missing series with emergency data. Header = false makes sure the first row isnt the name of the columns

price <- read.csv("companies-price.csv", header = FALSE)

emerg <- read.csv("emergency-data.csv", header = FALSE)

dates <- read.csv("companies-dates.csv", header = FALSE)[,1]  # only the first column, which is dates

info <- read.csv("companies-info.csv",

                 header = FALSE,

                 stringsAsFactors = FALSE)  # no factors

# Naming the columns right

colnames(info) <- c("Ticker", "Name", "Sector")

# We then replace the first columns in price data as it's all NA - replacing with emergency data

price[, 1:10] <- as.matrix(emerg)

# Giving column names from tickers(firms) - easier to interpret

colnames(price) <- info$Ticker
```

Amel Bejtic: 202305234
Jens Christian Andersen: 202305285

3.A

```
# Converting as matrix and taking the log of all prices

P <- as.matrix(price)

logP <- log(P)

# log returns: r_t = log P_t - log P_{t-1} - which is the formula for log prices

R_mat <- apply(logP, 2, diff)


# multiplying the log by 100

R_mat <- 100 * R_mat

# Dropping first date - there is no day before the first day, so it's purely NA. Removing it.

ret_dates <- dates[-1]
```

3.B

```
# Calculating the meaan for all stocks everyday. This is thé "market" price - removing NA also

common_factor <- rowMeans(R_mat, na.rm = TRUE)

# Subtracting the common factor from each stock

R_demeaned <- R_mat - common_factor

# This is the data matrix X we will feed into SPACE:

X <- R_demeaned

# Cleaning X: dropping data with any NA or zero variance

na_cols   <- apply(X, 2, function(z) any(is.na(z))) # stocks that have missing data

zero_var  <- apply(X, 2, function(z) var(z, na.rm = TRUE) < 1e-8) # stocks not moving - variance zero

drop_cols <- na_cols | zero_var #dropping stocks that are either of the two above


# removing the columns from drop_cols and save the clean data in X_clean

cat("Dropping", sum(drop_cols), "columns due to NAs/zero variance\n")


X_clean   <- X[, !drop_cols]

tickers   <- colnames(X)[!drop_cols]

n_obs <- nrow(X_clean)

p_var <- ncol(X_clean)

cat("Final data matrix: ", n_obs, "obs x", p_var, "stocks\n") # 5030 obs x 83 stocks
```

Amel Bejtic: 202305234
Jens Christian Andersen: 202305285

3.C

Code:

```
# Selecting lambda by BIC and estimate network via SPACE

S_hat <- cov(X_clean)   # calculating sample covariance-matrix


# reconstructing precision matrix Omega from a space.joint fit

get_Omega_from_space <- function(fit) {

 Par <- fit$ParCor

 d   <- fit$sig.fit

 p   <- length(d)


 Omega <- matrix(0, nrow = p, ncol = p)

 diag(Omega) <- d


 # Ω_ij = -Par_ij * sqrt(Ω_ii * Ω_jj)

 for (i in 1:p) {

  for (j in 1:p) {

   if (i != j && Par[i, j] != 0) {

    Omega[i, j] <- - Par[i, j] * sqrt(d[i] * d[j])

   } } }

 return(Omega) }


# calculating BIC to choose lambda

bic_space <- function(fit, S, n) {

 Omega <- get_Omega_from_space(fit)

 logdet_Om <- as.numeric(determinant(Omega, logarithm = TRUE)$modulus)

 trace_SO  <- sum(S * Omega)

 loglik    <- n * (logdet_Om - trace_SO)

 k <- sum(Omega[upper.tri(Omega)] != 0)

 bic <- -2 * loglik + k * log(n)

 return(bic) }
```

```
lambda_grid <- seq(1.00, 1.45, by = 0.05) # the values we wanna test

bic_vals   <- numeric(length(lambda_grid))
fits       <- vector("list", length(lambda_grid))
parcors    <- vector("list", length(lambda_grid))   # NEW: store ParCor for each lambda

set.seed(1)
# making a loop that runs through all lambda values. Lambda is multiplied by n_obs which is required for the space function
for (i in seq_along(lambda_grid)) {
  lam <- lambda_grid[i]
  # We use lam1 = lambda * n_obs (T) as in the PS10 example
  fit <- space.joint(
    X_clean,
    lam1  = lam * n_obs,
    lam2  = 0,
    weight = TRUE)
  fits[[i]]    <- fit
  parcors[[i]] <- fit$ParCor          # storing partial correlations
  bic_vals[i]  <- bic_space(fit, S_hat, n_obs) # storing the BIC score for each model
  cat("lambda =", lam, ", BIC =", bic_vals[i], "\n") }

best_idx   <- which.min(bic_vals) # finding the lambda with the lowest BIC-score
lambda_opt <- lambda_grid[best_idx]
cat("Optimal lambda by BIC:", lambda_opt, "\n")

space_fit <- fits[[best_idx]] # getting the winner-model
Omega_hat <- get_Omega_from_space(space_fit)

P_hat <- parcors[[best_idx]]
```

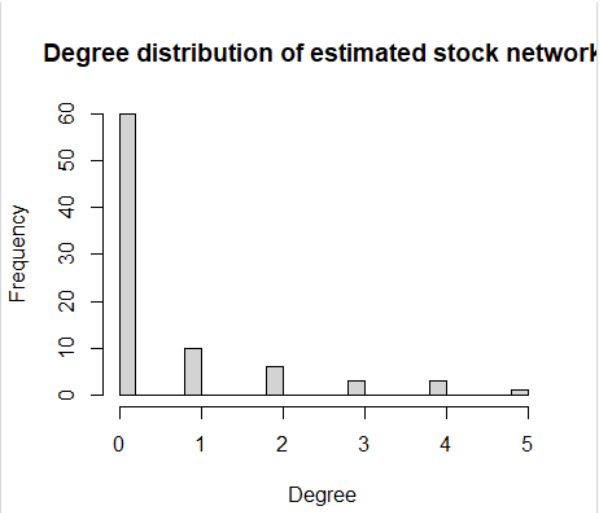
```
# Making the adjacency matrix. returning true/false - true = 1, false = 0
A <- 1 * (P_hat != 0)
diag(A) <- 0
```

3.D

Code:

```
# Adjacency: edge if Omega_ij != 0, no self-loops
A <- (Omega_hat != 0) * 1
diag(A) <- 0 # A is a networkmap with 0 and 1's. But setting the diagonal to 0
deg <- rowSums(A) #counting connections for each stock - the degree
# drawing a histogram over degrees
hist(deg,
    breaks = 20,
    main   = "Degree distribution of estimated stock network",
    xlab   = "Degree",
    ylab   = "Frequency")
```

Result/plot:



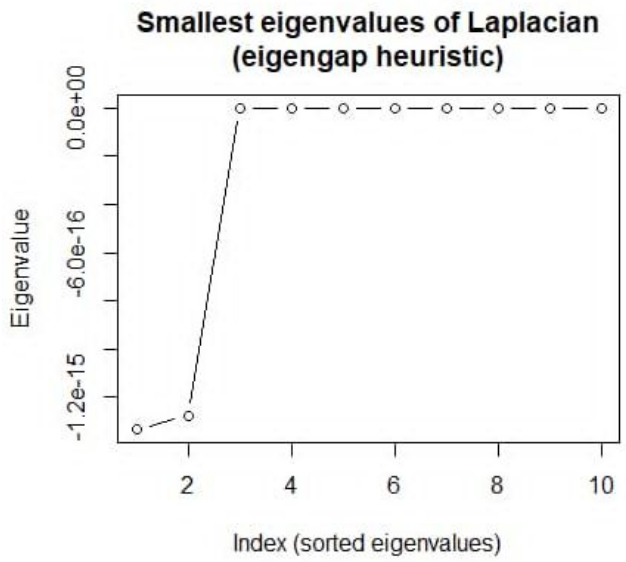
Amel Bejtic: 202305234
Jens Christian Andersen: 202305285

3.E

```
g <- graph_from_adjacency_matrix(A, mode = "undirected", diag = FALSE) # converting so that we can use networkfunctions

deg_vec <- deg

L <- diag(deg_vec) - A # calculating laplacian matrice

eig_L <- eigen(L, symmetric = TRUE) # finding smalles eigenvalues


# Plotting smallest eigenvalues to inspect eigengap

vals_sorted <- sort(eig_L$values)

plot(1:10, vals_sorted[1:10],

    type = "b",

    xlab = "Index (sorted eigenvalues)",

    ylab = "Eigenvalue",

    main = "Smallest eigenvalues of Laplacian\n(eigengap heuristic)")


# Choosing k =1 (talked about in our answer

k <- 1

cat("Using k =", k, "clusters\n")

# Take eigenvectors for the k smallest non-zero eigenvalues

ord <- order(eig_L$values)

U   <- eig_L$vectors[, ord[1:k]]

set.seed(1)

cl_kmeans <- kmeans(U, centers = k, nstart = 50)$cluster # using eigenvectors to part stocks in k-groups

comm <- cl_kmeans
```

Answer/plot

The laplacian eigenvalues show that several eigenvalues are extremely close to 0, indicating many disconnected components and no clear enginegap. So no meaningful k>1. We treat the network as having no nontrivial communities.



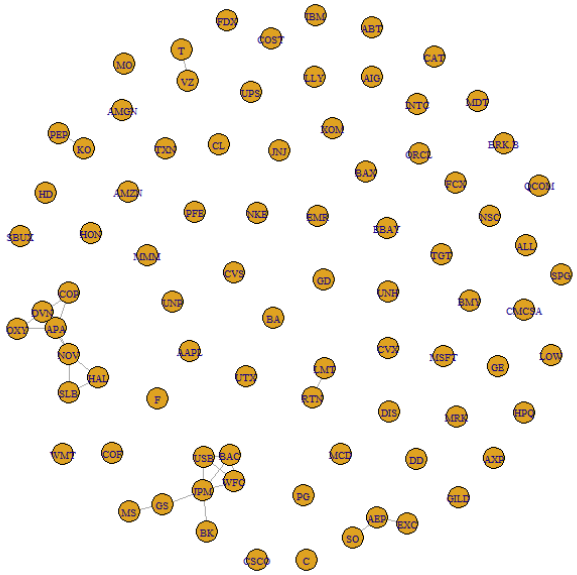
3.F

Code:

```
set.seed(1)

lay <- layout_with_fr(g) # Fruchterman-Reingold layout. Placing neighbours close to each other and not neighbours further away. Makes the plot look better


plot(g,
    layout      = lay,
    vertex.size  = 4,
    vertex.label = NA,
    vertex.color = comm,
    main = "Estimated stock network\n(colours = spectral clustering communities)")
```

Amel Bejtic: 202305234
Jens Christian Andersen: 202305285

Answer/plot:



All vertices are the same color because we picked k=1. The plot shows that most stocks are isolated nodes, and there are a few small connected components. This matches the structure suggested by the eigen values.

3.G

Code:

```
cat("Average degree:", mean(deg), "\n")

cat("Max degree    :", max(deg), " (ticker:",

    tickers[which.max(deg)], ")\n")


# Global clustering coefficient (triangles / triplets)

clust_coeff <- transitivity(g, type = "global") # tells us: if A knows B, B knows C, does A then also know C?

cat("Global clustering coefficient:", clust_coeff, "\n")


# Average shortest path length (on largest connected component)

if (!is.connected(g)) {

  gcc <- components(g)

  big_comp <- which.max(gcc$csize)

  g_big <- induced_subgraph(g, which(gcc$membership == big_comp))
```

```
  avg_path <- average.path.length(g_big)
  cat("Average path length (giant component):", avg_path, "\n")
} else {
  avg_path <- average.path.length(g)
  cat("Average path length:", avg_path, "\n")}
```

Answer:

The network is extremely sparse, having removed the common factor. We don’t see any strong hubs or typical regularities. We spot small clusters and short paths within the small connected components. It overall looks fragmented and community structure is very limited. Overall the empirical patterns appear only in a weak form.

Amel Bejtic: 202305234
Jens Christian Andersen: 202305285

Problem 4

Code for A and B

 # Setting the pseudo-random number generator seed to 1.

```
set.seed(1)
```

 # Simulation parameters:

```
n    <- 50        # training size

n_p  <- 100       # test size

R    <- 1000      # number of replications

p_list <- seq(5, 45, 5) # making a list of numbers from 5-45 with in between. This is the number of predictors we want to test
```

 # Functions:

```
# manually calculating beta-hat.

basic_ols <- function(Y, X){

  beta_hat <- solve(t(X) %*% X) %*% (t(X) %*% Y) # the solution to OLS

  return(beta_hat)}

# simple function where we calculate MSE manually

mse <- function(Y_hat, Y_true){

  mean((Y_hat - Y_true)^2) }


# Storage for results - so that we can easily plot it after. We create three empty matrices with 1000 rows(R) and the amount of columns as values in p_list

MSE_storage   <- matrix(0, nrow = R, ncol = length(p_list))

VAR_storage   <- matrix(0, nrow = R, ncol = length(p_list))

BIAS2_storage <- matrix(0, nrow = R, ncol = length(p_list))


colnames(MSE_storage) <- colnames(VAR_storage) <- colnames(BIAS2_storage) <- p_list
                # giving the values from p_list as names for the columns in the other 3 matrices (MSE, VAR, BIAS2). Gives easier interpretation
```

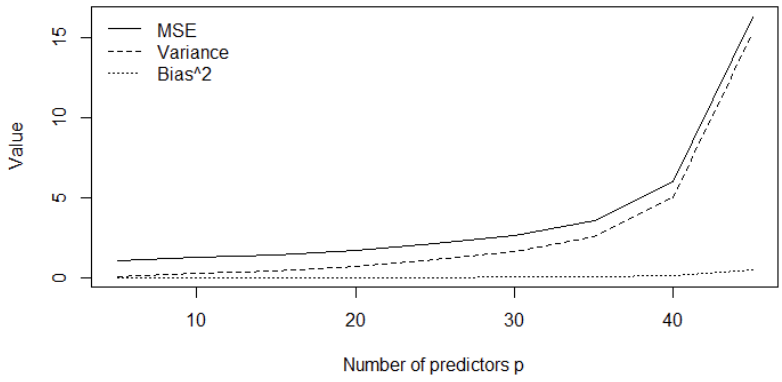
 # Simulation loops

```
for (j in seq_along(p_list)) { # first we run a loop through p_list, where P=5 first, after p=10 and so on until 45

                p <- p_list[j] # getting the amount of predictors

                beta_true <- runif(p + 1, min = 0, max = 5) # Draw true beta for this p: length p+1 (incl. intercept), entries ~ U(0,5) which is asked for in the exercise

# Fixing test design X_test for all replications. Cbind(1) adds a column with 1's for the intercept.

                 X_test <- cbind(1, matrix(rnorm(n_p * p), nrow = n_p, ncol = p))

                 f_true_test <- as.vector(X_test %*% beta_true)   # true regression function. This is line we want to hit.

# Storage of predictions for this p across replications (til bias/var)

                Y_hat_mat <- matrix(0, nrow = n_p, ncol = R)

  # Loop over replications:

  for (r in 1:R) {

# --- Simulating training data. X_train makes a new training matrix with 50 rows (n). eps_tr creates noise - from rnorm. Y_train creates our Y-values in a vector with the formula Y=Xβ+ϵ

                X_train <- cbind(1, matrix(rnorm(n * p), nrow = n, ncol = p))

                eps_tr  <- rnorm(n, mean = 0, sd = 1)

                Y_train <- as.vector(X_train %*% beta_true + eps_tr)

                # --- Fit OLS. With the model we try finding β based on training data

                beta_hat <- basic_ols(Y_train, X_train)

    # --- Predicting on test-values. Using the beta_hat values with our X_test to guess Y-values

                 Y_hat_test <- as.vector(X_test %*% beta_hat)

                Y_hat_mat[, r] <- Y_hat_test # saving our estimate

# Simulating noisy test responses to compute MSE: eps_te makes noise for our test-set. Y_test is our true Y in the test set = the true line + the noise, we just created.

                eps_te  <- rnorm(n_p, mean = 0, sd = 1)

                Y_test  <- f_true_test + eps_te

                 # Calculating MSE between our guess and the observed test set

                MSE_storage[r, j] <- mse(Y_hat_test, Y_test)

# calculating the approximate mean of the distances between our guess and the right line.

                 VAR_storage[r, j] <- mean((f_true_test - Y_hat_test)^2)

# Squared bias (OLS almost unbiased here, result therefore very low):
```

```
                    BIAS2_storage[r, j] <- (mean(f_true_test - Y_hat_test))^2 } }

# Averaging over replications. We have 1000 results for each p. Here we take the mean of them all,
so  we get one value for MSE, VAR and bias^2.

MSE_mean   <- colMeans(MSE_storage)

VAR_mean   <- colMeans(VAR_storage)

BIAS2_mean <- colMeans(BIAS2_storage)


# simply making a data.frame with the values

results <- data.frame(

                    p      = p_list,

                    MSE    = MSE_mean,

                    Variance = VAR_mean,

                    Bias2    = BIAS2_mean)

# Plotting the three measures for easier interpretation

plot(results$p, results$MSE, type = "l", ylim = range(c(MSE_mean, VAR_mean)),

    xlab = "Number of predictors p", ylab = "Value")

lines(results$p, results$Variance, lty = 2)

lines(results$p, results$Bias2, lty = 3)

legend("topleft",

      legend = c("MSE", "Variance", "Bias^2"),

      lty = c(1,2,3), bty = "n")
```

Amel Bejtic: 202305234
Jens Christian Andersen: 202305285

4.A



Variance

When looking at the variance, it measures the expected spread of $\hat{f}(x)$ around its mean. The quantity here rises with the number of predictors, meaning it would perform bad on a different set of training data due to overfitting. As (p) is increasing -> n the setting comes closer to a high dimensional setting (when n is fixed), where overfitting is one of the main problems, causing a very high variance

Squarred bias

The squared bias measures the difference between the expected value of the estimator and the true value. A higher bias would indicate potential underfitting, but in our model the squared bias remains close to zero for all values of p. This happens because the data we generated are from a true linear model and OLS is an unbiased estimator under the classical assumptions. The expected value of our true estimator would just be the true function under this assumption, $E\left[\hat{f}(x)\right] = f(x)$ also meaning $bias^2 = 0$

MSE

Looking at the test MSE, Mean squared error, we see it increasing as p, the number of predictors, increases. The test MSE is calculated as:

$$MSE(x) = E\left[\left(Y - \hat{f}(x)\right)^2\right]$$

$$MSE(x) = Bias(x)^2 + Var(x) + \sigma^2$$

As we concluded the bias is near 0, and the variance is increasing with p -> ∞, we only haven't spoken about the irreducible error (var(ϵ)) which represent the present noise in our data. But this has already been considered, as we set delta squared = 1 in the simulation, representing a constant value.

This gives us a MSE following the variance, but with a higher intercept.

Amel Bejtic: 202305234
Jens Christian Andersen: 202305285

4.B

Some of this has already been considered in (a), but it's mostly due to the irreducible error being fixed at 1. This makes MSE 1 higher than the sum of the squared bias and variance, as MSE follows the variance otherwise because of our squared bias being near 0. In our simulation we define the error term as $\epsilon \sim N(0,1)$, which is setting the variance of the noise term to be 1, and as our squared bias is near 0 due to OLS being an unbiased estimator of our true linear model, the MSE will approximately be equal to the variance plus the error term, which we also see being true in the plot.

Amel Bejtic: 202305234
Jens Christian Andersen: 202305285

Problem 5

Code

```
set.seed(1)


# (i) Drawing a 200 × 2 matrix of standard normal values

X <- matrix(rnorm(200 * 2), ncol = 2)


# (ii) Computing Euclidean lengths for each row

row_len <- sqrt(rowSums(X^2))


# (iii) Multiplying rows with length > 1 by 5

X[row_len > 1, ] <- 5 * X[row_len > 1, ]


# (iv) Standardise the matrix X (each column)

X <- scale(X)


# (iv) Plotting the data

plot(X, pch = 20, col = "black",
     main = "Simulated data (two clusters)",
     xlab = "X1", ylab = "X2")


# (iv() Applying k-means

set.seed(1)

km <- kmeans(X, centers = 2, nstart = 50)


# (iv) Plot coloured by cluster

plot(X, col = km$cluster, pch = 20,
     main = "k-means clustering result",
     xlab = "X1", ylab = "X2")
```

```
# Add cluster centres
points(km$centers, col = 1:2, pch = 8, cex = 2)
```

Comments on results:

We see the data form two intuitive clusters, a middle and a cloud surrounding it, which is further away. This is what happens because observations with euclidean length > 1, were multiplied by 5.

We apply the k-means with k=2, and the algorithm does not catch the clusters(ball and cloud). Instead we see that the outside cloud is divided into 2 and the middle takes the cluster/color of which of the centroids is closer. I have plotted the centroid as stars. The result happens because it assigns each point to the closest centroid, measured by euclidian distance. It tries to minimize the sum of squared distances within each cluster, and so we see a different result than what makes intuitive sense to the naked eye. This illustrates a limitation to the method.