Practice

2025-05-20

#install.packages("tinytex")

50/50 Split

```
# Creates a sequence of integers from 1 to 20
dataseq <- c(seq(20))

# Reshapes the sequence into a 5x4 matrix
mydata <- matrix(dataseq, nrow = 5, ncol = 4)

# Stores the number of rows (observations) in the matrix
n <- nrow(mydata)

# Sets the random seed to ensure reproducibility of the sampling
set.seed(123)

# Randomly selects floor(n/2) row indices for training data
train_indicies <- sample(1:n, floor(n / 2))

# Extracts the training subset based on sampled indices
train_data <- mydata[train_indicies, ]

# Extracts the remaining rows as the test subset
test_data <- mydata[-train_indicies, ]

# Prints the training data
print(train_data)
```

```
##      [,1] [,2] [,3] [,4]
## [1,]    3    8   13   18
## [2,]    2    7   12   17
```

```
# Prints the test data
print(test_data)
```

```
##      [,1] [,2] [,3] [,4]
## [1,]    1    6   11   16
## [2,]    4    9   14   19
## [3,]    5   10   15   20
```

Simple Linear Regression in R

```
library(ISLR)
```

```
## Warning: package 'ISLR' was built under R version 4.4.2
```

```
library(tidyverse)
```

```
## Warning: package 'tidyverse' was built under R version 4.4.3
```

```
## Warning: package 'ggplot2' was built under R version 4.4.2
```

```
## Warning: package 'tidyr' was built under R version 4.4.2
```

```
## Warning: package 'readr' was built under R version 4.4.2
```

```
## Warning: package 'purrr' was built under R version 4.4.2
```

```
## Warning: package 'dplyr' was built under R version 4.4.2
```

```
## Warning: package 'forcats' was built under R version 4.4.2
```

```
## Warning: package 'lubridate' was built under R version 4.4.2
```

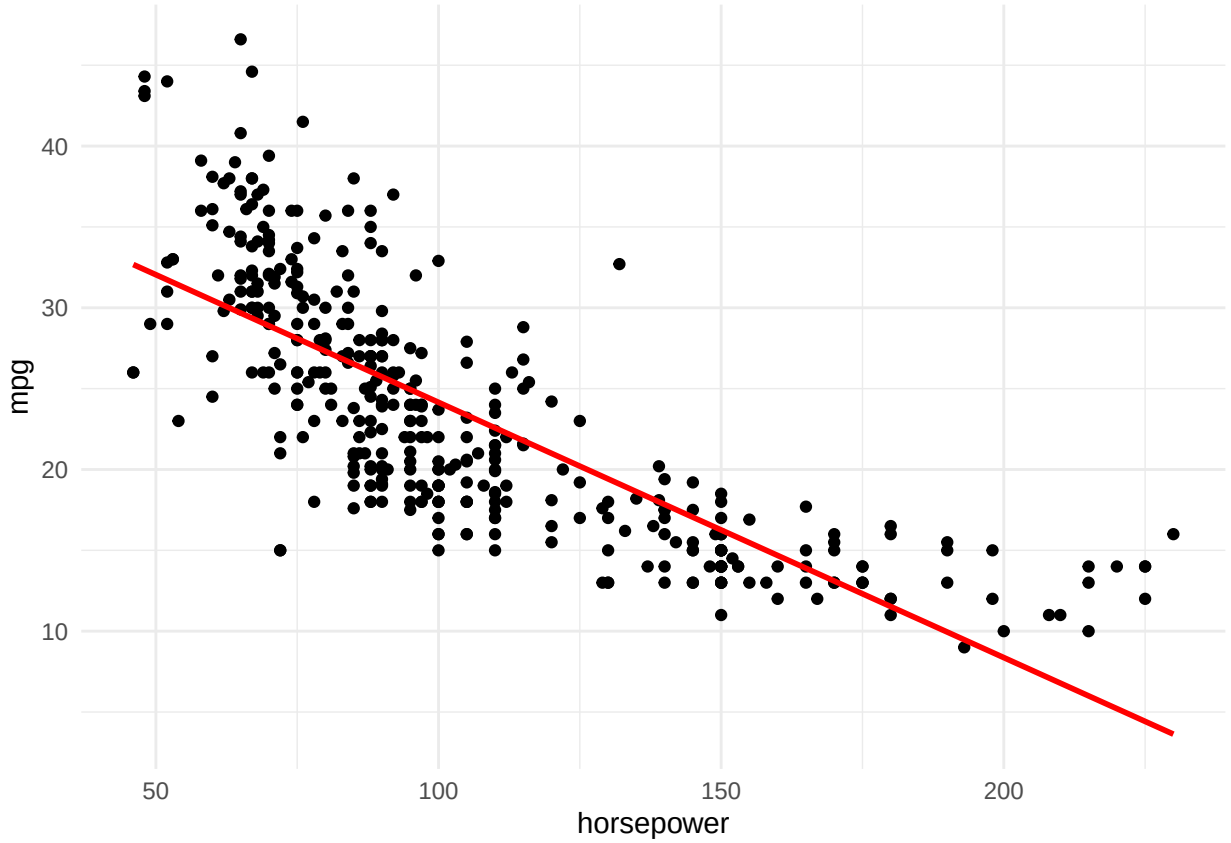
```
## -- Attaching core tidyverse packages ------------------------ tidyverse 2.0.0 --
## v dplyr     1.1.4     v readr     2.1.5
## v forcats   1.0.0     v stringr   1.5.1
## v ggplot2   3.5.1     v tibble    3.2.1
## v lubridate 1.9.3     v tidyr     1.3.1
## v purrr     1.0.2
## -- Conflicts ------------------------------------------ tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
## i Use the conflicted package (<http://conflicted.r-lib.org/>) to force all conflicts to become errors
```

```
library(ggplot2)
library(ggthemes)
```

```
## Warning: package 'ggthemes' was built under R version 4.4.2
```

```
ggplot(data = Auto, aes(x = horsepower, y = mpg)) +
  geom_point()+
  theme_minimal()+
  geom_smooth(method= "lm", se= F, col= "red")
```

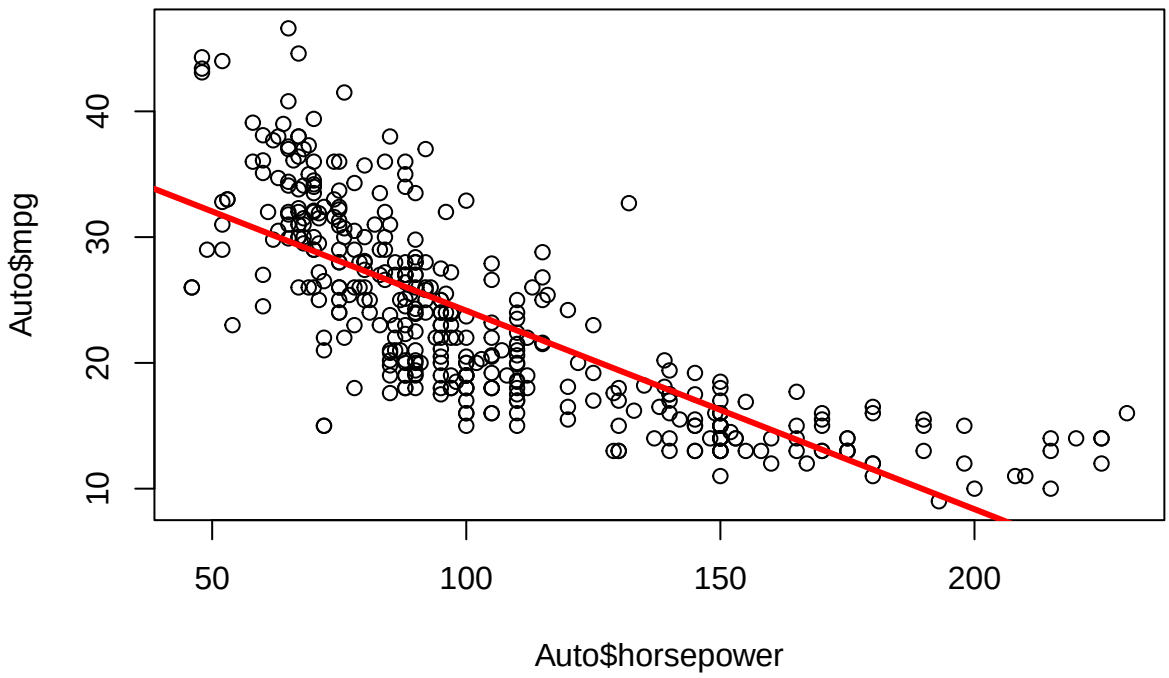
```
## `geom_smooth()` using formula = 'y ~ x'
```




```
lm.fit<- lm(mpg ~ horsepower, data = Auto)

summary(lm.fit)
```

```
##
## Call:
## lm(formula = mpg ~ horsepower, data = Auto)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -13.5710  -3.2592  -0.3435   2.7630  16.9240
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept) 39.935861   0.717499   55.66   <2e-16 ***
## horsepower  -0.157845   0.006446  -24.49   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 4.906 on 390 degrees of freedom
## Multiple R-squared:  0.6059, Adjusted R-squared:  0.6049
## F-statistic: 599.7 on 1 and 390 DF,  p-value: < 2.2e-16
```

```
plot(Auto$horsepower, Auto$mpg)
abline(lm.fit, col ="red", lwd=3)
```



Manual estimation of linear regression coefficients using the ordinary least squares (OLS) formula

```r
# Constructs the design matrix X with an intercept (column of 1s) and 'horsepower' as the explanatory v
X_design <- cbind(1, Auto$horsepower)

# Defines the response variable y as miles per gallon (mpg)
y_response <- Auto$mpg

# Manually computes the OLS coefficients using the formula: (XX)^(-1) X^T*y
beta_hat_manual <- solve(t(X_design) %*% X_design) %*% t(X_design) %*% y_response

# Prints the estimated regression coefficients
print(beta_hat_manual)
```

```
##             [,1]
## [1,] 39.9358610
## [2,] -0.1578447
```

Multiple Linear Regression for Boston Housing Data

```r
library(MASS)
```

```
##
## Attaching package: 'MASS'
```

```
## The following object is masked from 'package:dplyr':
##
##     select
```

```r
lm.fit.multi <- lm(medv ~ lstat + age, data = Boston)
summary(lm.fit.multi)
```

```
##
## Call:
## lm(formula = medv ~ lstat + age, data = Boston)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -15.981  -3.978  -1.283   1.968  23.158
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) 33.22276    0.73085  45.458  < 2e-16 ***
## lstat       -1.03207    0.04819 -21.416  < 2e-16 ***
## age          0.03454    0.01223   2.826  0.00491 **
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 6.173 on 503 degrees of freedom
## Multiple R-squared:  0.5513, Adjusted R-squared:  0.5495
## F-statistic:   309 on 2 and 503 DF,  p-value: < 2.2e-16
```

```r
lm.fit.all <- lm(medv ~ ., data = Boston)
summary(lm.fit.all)
```

```
##
## Call:
## lm(formula = medv ~ ., data = Boston)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -15.595  -2.730  -0.518   1.777  26.199
##
## Coefficients:
##               Estimate Std. Error t value Pr(>|t|)
## (Intercept)  3.646e+01  5.103e+00   7.144 3.28e-12 ***
## crim        -1.080e-01  3.286e-02  -3.287 0.001087 **
## zn           4.642e-02  1.373e-02   3.382 0.000778 ***
## indus        2.056e-02  6.150e-02   0.334 0.738288
## chas         2.687e+00  8.616e-01   3.118 0.001925 **
## nox         -1.777e+01  3.820e+00  -4.651 4.25e-06 ***
```

```
## rm           3.810e+00  4.179e-01   9.116  < 2e-16 ***
## age          6.922e-04  1.321e-02   0.052 0.958229
## dis         -1.476e+00  1.995e-01  -7.398 6.01e-13 ***
## rad          3.060e-01  6.635e-02   4.613 5.07e-06 ***
## tax         -1.233e-02  3.760e-03  -3.280 0.001112 **
## ptratio     -9.527e-01  1.308e-01  -7.283 1.31e-12 ***
## black        9.312e-03  2.686e-03   3.467 0.000573 ***
## lstat       -5.248e-01  5.072e-02 -10.347  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 4.745 on 492 degrees of freedom
## Multiple R-squared:  0.7406, Adjusted R-squared:  0.7338
## F-statistic: 108.1 on 13 and 492 DF,  p-value: < 2.2e-16
```

Confidence and Prediction Intervals in R

```r
# Create a new data frame with specific horsepower values
new_hp_values <- data.frame(horsepower = c(98, 150, 200))

# Predict the expected MPG for these horsepower values with 95% confidence intervals
predict(lm.fit, newdata = new_hp_values, interval = "confidence")
```

```
##         fit       lwr       upr
## 1 24.467077 23.973079 24.961075
## 2 16.259151 15.504025 17.014277
## 3  8.366914  7.061985  9.671844
```

```r
# Predict the MPG for individual cars with these horsepower values, including 95% prediction intervals
predict(lm.fit, newdata = new_hp_values, interval = "prediction")
```

```
##         fit       lwr      upr
## 1 24.467077 14.809396 34.12476
## 2 16.259151  6.584598 25.93370
## 3  8.366914 -1.365999 18.09983
```

Checking VIF in R <- Variance Inflation Factor

```r
library(car)
```

```
## Warning: package 'car' was built under R version 4.4.3
```

```
## Loading required package: carData
```

```
## Warning: package 'carData' was built under R version 4.4.3
```

```
##
## Attaching package: 'car'
```

```
## The following object is masked from 'package:dplyr':
##
##     recode

## The following object is masked from 'package:purrr':
##
##     some
```

```
vif_values <- vif(lm.fit.all)
print(vif_values)
```

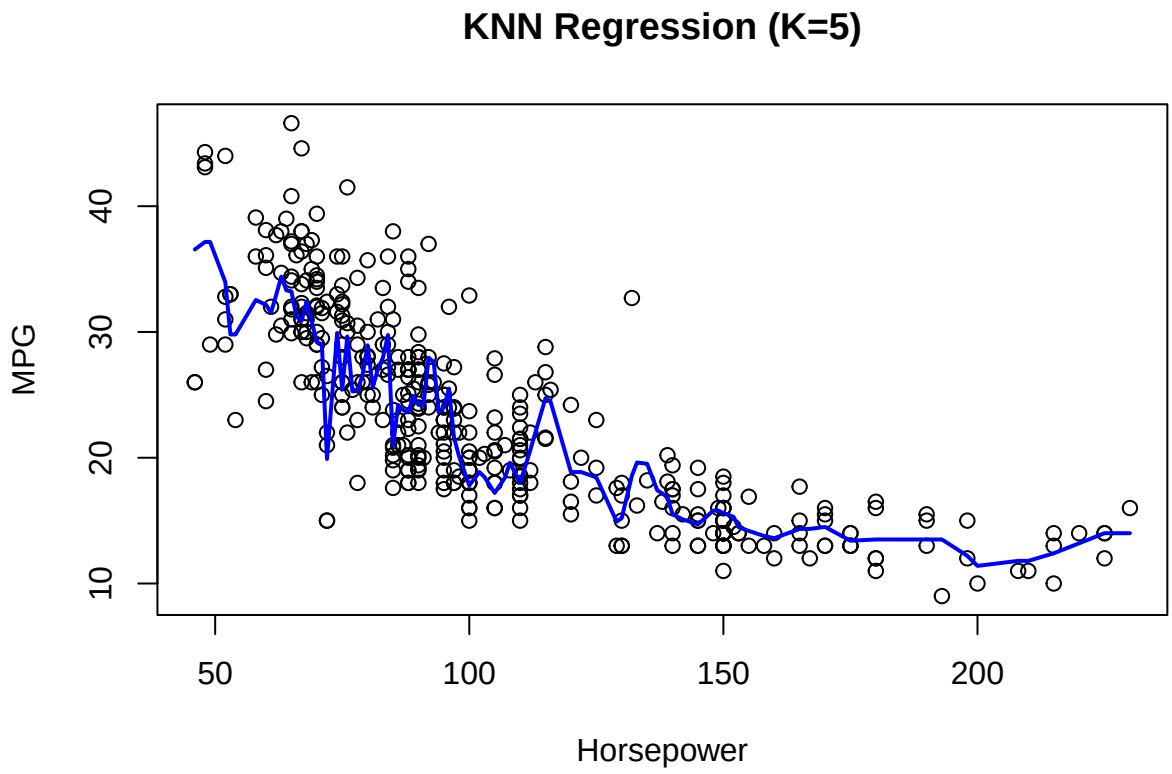
```
##     crim       zn    indus     chas      nox       rm      age      dis
## 1.792192 2.298758 3.991596 1.073995 4.393720 1.933744 3.100826 3.955945
##      rad      tax  ptratio    black    lstat
## 7.484496 9.008554 1.799084 1.348521 2.941491
```

```
# VIF values indicate low to moderate multicollinearity for most predictors,
# except 'rad' and 'tax', which show high multicollinearity and should be examined further.
```

Conseptual KNN

```
# Define a function to perform a single KNN prediction
knn_function_single_pred <- function(x0, x_train, y_train, K_val = 20) {
  # Calculate the absolute distances between the target point (x0) and all training points (x_train)
  distances <- abs(x_train - x0)
  # Get the indices that would sort the distances in ascending order
  ordered_indices <- order(distances)
  # Select the indices of the K_val nearest neighbors
  neighbor_indices <- ordered_indices[1:K_val]
  # Predict the y value by taking the mean of the y_train values for the K_val nearest neighbors
  predicted_y <- mean(y_train[neighbor_indices])
  # Return the predicted y value
  return(predicted_y)
}

# Load the ISLR package, which contains the Auto dataset
library(ISLR)
# Load the Auto dataset into the current R session
data(Auto)

# Create a sorted vector of horsepower values from the Auto dataset
x_values_sorted <- sort(Auto$horsepower)
# Apply the knn_function_single_pred to each value in x_values_sorted to get KNN predictions
y_predictions_knn <- sapply(x_values_sorted, function(x0) {
  # For each x0 in x_values_sorted, call the KNN prediction function

  knn_function_single_pred(x0, x_train = Auto$horsepower, y_train = Auto$mpg, K_val = 5)
  # Use Auto$horsepower as the training x values
  # Use Auto$mpg as the training y values
  # Set the number of neighbors (K) to 5
})
```

```r
# Create a scatter plot of horsepower vs. mpg from the Auto dataset
plot(Auto$horsepower, Auto$mpg, xlab="Horsepower", ylab="MPG", main="KNN Regression (K=5)")
# Add a line to the plot representing the KNN predictions
lines(x_values_sorted, y_predictions_knn, col = "blue", lwd = 2)
```



Linear Probability Model

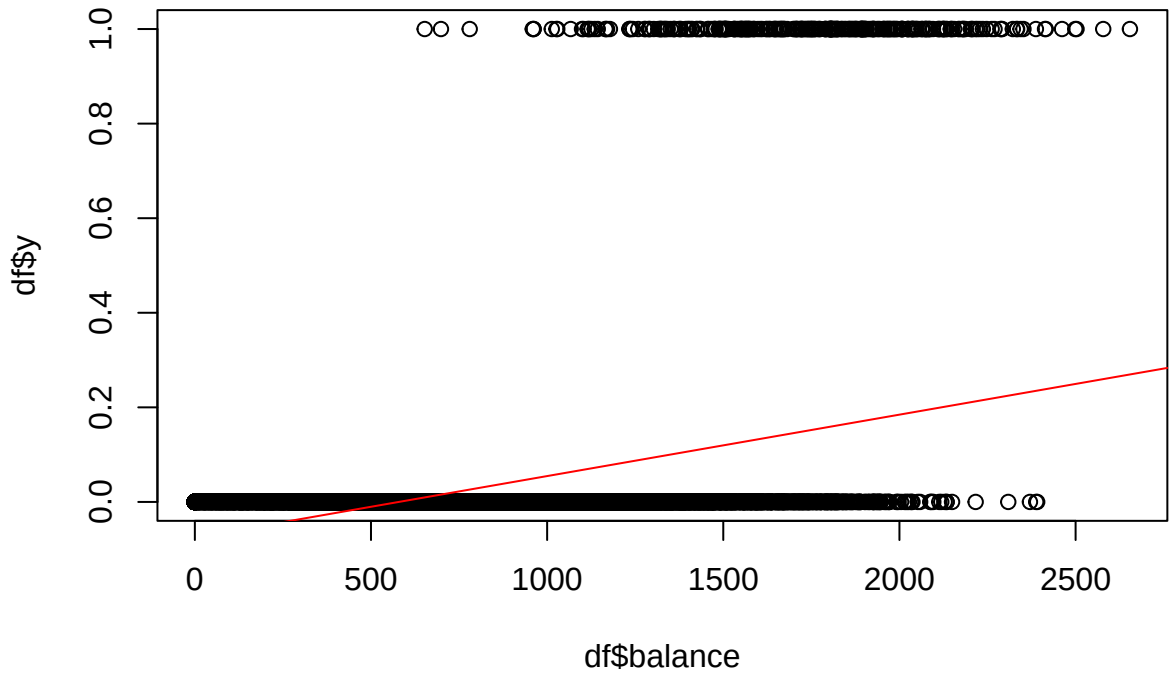
```r
library(ISLR)

df <- Default
df$y <- ifelse(df$default == "Yes", 1, 0)

head(df)
```

```
##   default student   balance    income y
## 1      No      No  729.5265 44361.625 0
## 2      No     Yes  817.1804 12106.135 0
## 3      No      No 1073.5492 31767.139 0
## 4      No      No  529.2506 35704.494 0
## 5      No      No  785.6559 38463.496 0
## 6      No     Yes  919.5885  7491.559 0
```

```r
linprob <- lm(y ~balance, data = df)
summary(linprob)
```

```
##
## Call:
## lm(formula = y ~ balance, data = df)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -0.23533 -0.06939 -0.02628  0.02004  0.99046
##
## Coefficients:
##               Estimate Std. Error t value Pr(>|t|)
## (Intercept) -7.519e-02  3.354e-03  -22.42   <2e-16 ***
## balance      1.299e-04  3.475e-06   37.37   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.1681 on 9998 degrees of freedom
## Multiple R-squared:  0.1226, Adjusted R-squared:  0.1225
## F-statistic:  1397 on 1 and 9998 DF,  p-value: < 2.2e-16
```

```r
plot(df$balance, df$y)
abline(linprob, col = "red")
```



##LogIT

```
StudentDefault <- glm(default ~ student, family = "binomial", data = Default)
StudentDefault
```

```
##
## Call:  glm(formula = default ~ student, family = "binomial", data = Default)
##
## Coefficients:
## (Intercept)   studentYes
##     -3.5041       0.4049
##
## Degrees of Freedom: 9999 Total (i.e. Null);  9998 Residual
## Null Deviance:       2921
## Residual Deviance: 2909  AIC: 2913
```

```
#The odds are
print("____________")
```

```
## [1] "____________"
```

```
exp(0.4)
```

```
## [1] 1.491825
```

```
print(

## Propbability
  (exp(-3.5041+0.4049*1))/
    (1+ exp(-3.5041+0.4049*1))
      )
```

```
## [1] 0.04314027
```

Logistic Regression and Confussion Matrix

```
# Fit a logistic regression model predicting default based on student status, balance, and income
logprob <- glm( default ~ student + balance + income, data = Default, family = "binomial")

# Predict the probability of default using the fitted model
pred_probs <- predict(logprob, type = "response")

# Convert predicted probabilities to class labels: "Yes" if > 0.5, else "No"
pred_class <- ifelse(pred_probs > 0.5, "Yes", "No")

# Create a confusion matrix comparing actual vs. predicted default values
conf_matrix_full <- table(Default$default, pred_class)

# Print the confusion matrix
print(conf_matrix_full)
```

```
##      pred_class
##         No  Yes
##   No  9627   40
##   Yes  228  105
```

Example with test/train data – Logistic Regression and Confussion Matrix

```r
# Set a seed for reproducibility of the random sampling
set.seed(123)

# Get the number of observations in the dataset
n <- nrow(Default)

# Randomly select half of the indices for the training set
train_indicies <- sample(1:n, n/2)

# Create the training dataset using the sampled indices
train_data <- Default[train_indicies,]

# Create the test dataset using the remaining indices
test_data <- Default[-train_indicies,]

# Fit a logistic regression model on the training data
logprob_train <- glm(default ~ student + balance + income, data = train_data, family="binomial")

# Predict default probabilities on the test data
pred_probs_test <- predict(logprob_train, newdata = test_data, type="response")

# Convert probabilities to class labels: "Yes" if > 0.5, otherwise "No"
pred_class_test <- ifelse(pred_probs_test > 0.5, "Yes", "No")

# Create a confusion matrix comparing actual vs. predicted defaults in test data
conf_matrix_test <- table(test_data$default, pred_class_test)

# Print the confusion matrix
print(conf_matrix_test)
```

```
##      pred_class_test
##         No  Yes
##   No  4809   20
##   Yes  116   55
```

```r
# Calculate accuracy by dividing correct predictions by total predictions
accuracy_test <- sum(diag(conf_matrix_test)) / sum(conf_matrix_test)

# Print the test set accuracy
print(accuracy_test)
```

```
## [1] 0.9728
```

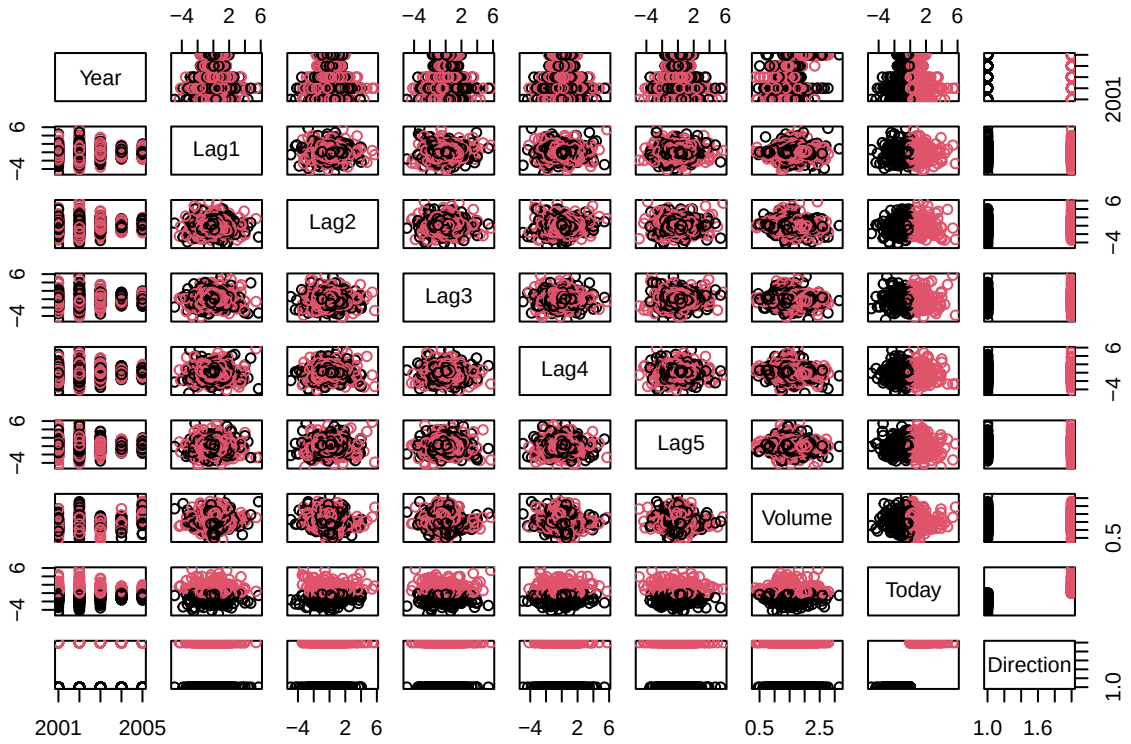
Multiple Log Regression on Weekly Data

```
# Load the ISLR package, which contains the 'Weekly' dataset and other resources
library(ISLR)

# Fit a logistic regression model to predict 'Direction' using lagged returns and trading volume
# The response variable is binary ("Up" or "Down"), so we use the binomial family
glm.fit <- glm(Direction ~ Lag1 + Lag2 + Lag3 + Lag4 + Lag5 + Volume,
               data = Weekly,
               family = "binomial")

# Display a detailed summary of the fitted logistic regression model
# Includes coefficient estimates, standard errors, z-values, and p-values
summary(glm.fit)
```

```
##
## Call:
## glm(formula = Direction ~ Lag1 + Lag2 + Lag3 + Lag4 + Lag5 +
##     Volume, family = "binomial", data = Weekly)
##
## Coefficients:
##             Estimate Std. Error z value Pr(>|z|)
## (Intercept)  0.26686    0.08593   3.106   0.0019 **
## Lag1        -0.04127    0.02641  -1.563   0.1181
## Lag2         0.05844    0.02686   2.175   0.0296 *
## Lag3        -0.01606    0.02666  -0.602   0.5469
## Lag4        -0.02779    0.02646  -1.050   0.2937
## Lag5        -0.01447    0.02638  -0.549   0.5833
## Volume      -0.02274    0.03690  -0.616   0.5377
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 1496.2  on 1088  degrees of freedom
## Residual deviance: 1486.4  on 1082  degrees of freedom
## AIC: 1500.4
##
## Number of Fisher Scoring iterations: 4
```

LogReg Video

```
#names(Smarket)
#summary(Smarket)
#?Smarket

pairs(Smarket, col = Smarket$Direction)
```



Conf Matrix on Weekly Data

```
# Predict probabilities of the "Up" direction using the logistic regression model (glm.fit)
glm.probs <- predict(glm.fit, type = "response")

# Initialize a vector of predictions with the default class "Down"
glm.pred <- rep("Down", length(glm.probs))

# Assign the class "Up" to all predictions where the predicted probability is greater than 0.5
glm.pred[glm.probs > 0.5] <- "Up"

# Create a confusion matrix comparing actual market direction to predicted direction
conf_matrix <- table(Weekly$Direction, glm.pred)

# Print the confusion matrix to evaluate classification performance
print(conf_matrix)
```

```
##       glm.pred
##        Down  Up
##   Down   54 430
##   Up     48 557
```

```r
# Calculate the classification accuracy by comparing predicted directions to actual directions
accuracy <- mean(glm.pred == Weekly$Direction)

# Print the accuracy to evaluate the overall prediction performance of the model
print(accuracy)
```

```
## [1] 0.5610652
```

LDA on Default Data

```r
x <- Default$balance
# Extract the 'balance' variable from the dataset and store it in x.

cl <- Default$default
# Extract the class labels ("No" or "Yes") and store in cl.

nk <- table(cl)
# Count how many observations belong to each class.

n_num_classes <- length(nk)
# Calculate the number of classes (e.g., 2).

pi_hat <- nk / sum(nk)

mu_hat <- as.matrix(by(x, cl, mean))
# Compute the class means for 'balance' and store them as a matrix.

s2_no <- sum((x[cl == "No"] - mu_hat[1])^2)
# Compute the sum of squared deviations from the mean for class "No".

s2_yes <- sum((x[cl == "Yes"] - mu_hat[2])^2)
# Compute the sum of squared deviations from the mean for class "Yes".

s2_pooled <- (s2_no + s2_yes) / (sum(nk) - n_num_classes)
# Compute the pooled variance estimate across both classes.

delta_k <- function(x_val, k_idx, mu_vec, s2_val, pi_vec) {
  mu_k <- mu_vec[k_idx]
  # Select the mean for class k.
  pi_k <- pi_vec[k_idx]
  # Select the prior probability for class k.
  return(x_val * mu_k / s2_val - mu_k^2 / (2 * s2_val) + log(pi_k))
  # Return the discriminant function value for class k.
}

delta1_vals <- delta_k(x, 1, mu_hat, s2_pooled, pi_hat)
# Calculate discriminant values for class 1 ("No").

delta2_vals <- delta_k(x, 2, mu_hat, s2_pooled, pi_hat)
# Calculate discriminant values for class 2 ("Yes").
```

```r
pred_lda_manual <- ifelse(delta2_vals > delta1_vals, "Yes", "No")
# Predict class: assign "Yes" if class 2 has a higher discriminant value, else "No".

table(cl, pred_lda_manual)
```

```
##      pred_lda_manual
## cl      No  Yes
##   No  9643   24
##   Yes  257   76
```

```r
# Create a confusion matrix comparing actual vs. predicted labels.
```

Using MASS::lda()

```r
# Load the MASS package which contains the lda() function for Linear Discriminant Analysis
library(MASS)

# Create a logical vector to identify training observations (years before 2009)
train_indicies <- (Weekly$Year < 2009)

# Subset the data for training using the logical vector
train_data <- Weekly[train_indicies,]

# Subset the data for testing (i.e., observations from 2009 and later)
test_data <- Weekly[!train_indicies,]

# Fit an LDA model using Lag2 as the predictor for the binary outcome Direction
lda.fit <- lda(Direction ~ Lag2, data = train_data)

# Use the fitted model to predict on the test dataset
lda.pred_obj <- predict(lda.fit, newdata = test_data)

# Extract the predicted class labels from the prediction object
lda.class <- lda.pred_obj$class

# Create a confusion matrix comparing actual test labels with predicted labels
conf_matrix_lda <- table(test_data$Direction, lda.class)

# Print the confusion matrix to evaluate model performance
print(conf_matrix_lda)
```

```
##       lda.class
##        Down Up
##   Down    9 34
##   Up      5 56
```

```r
# Display the first few rows of the posterior probabilities from the prediction
head(lda.pred_obj$posterior)
```

```
##          Down        Up
## 986 0.4736555 0.5263445
## 987 0.3558617 0.6441383
## 988 0.5132860 0.4867140
## 989 0.5142948 0.4857052
## 990 0.4799727 0.5200273
## 991 0.4597586 0.5402414
```

##QDA on Weekly Data

```
library(MASS)
qda.fit <- qda(Direction ~ Lag2, data = train_data)
qda.pred_obj <- predict(qda.fit, newdata = test_data)
qda.class <- qda.pred_obj$class
conf_matrix_qda <- table(test_data$Direction, qda.class)
print(conf_matrix_qda)
```

```
##       qda.class
##        Down Up
##   Down    0 43
##   Up      0 61
```

##Evaluation Classification Models -> Generating ROC Curve Data

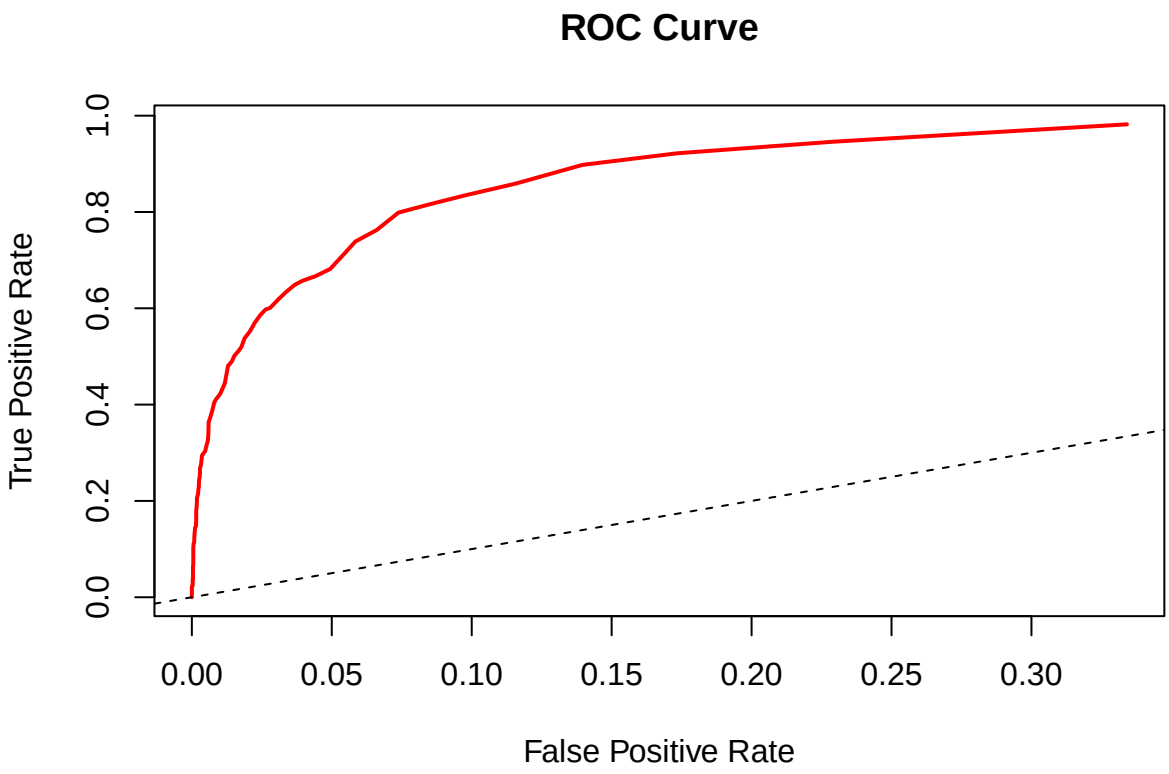
```
lda1 <- lda(cl ~ x, data = Default)
lda1
```

```
## Call:
## lda(cl ~ x, data = Default)
##
## Prior probabilities of groups:
##     No    Yes
## 0.9667 0.0333
##
## Group means:
##             x
## No   803.9438
## Yes 1747.8217
##
## Coefficients of linear discriminants:
##           LD1
## x 0.002206916
```

##ROC_Manual

```
# Extract the posterior probabilities from the LDA prediction
pr <- predict(lda1)$posterior

# Convert the true class labels ("Yes"/"No") to numeric: Yes = 1, No = 0
cl_numeric <- as.numeric(Default$default) - 1

# Define a sequence of threshold values from 0.01 to 0.99 (for ROC curve)
```

```r
thrange <- seq(0.01, 0.99, by = 0.01)

# Create an empty data frame to store False Positive Rate (FPrate) and True Positive Rate (TPrate)
roc_data <- data.frame(FPrate = numeric(length(thrange)),
                       TPrate = numeric(length(thrange)))

# Loop over each threshold to compute TPR and FPR
for (i in 1:length(thrange)) {

  # Set the current threshold
  th <- thrange[i]

  # Predict class: if posterior probability for "Yes" > threshold, classify as 1 (Yes), else 0 (No)
  pred_class <- ifelse(pr[, "Yes"] > th, 1, 0)

  # Calculate confusion matrix components:
  TP <- sum(pred_class == 1 & cl_numeric == 1)  # True Positives
  FN <- sum(pred_class == 0 & cl_numeric == 1)  # False Negatives
  FP <- sum(pred_class == 1 & cl_numeric == 0)  # False Positives
  TN <- sum(pred_class == 0 & cl_numeric == 0)  # True Negatives

  # Calculate and store True Positive Rate (Sensitivity)
  roc_data$TPrate[i] <- TP / (TP + FN)

  # Calculate and store False Positive Rate (1 - Specificity)
  roc_data$FPrate[i] <- FP / (FP + TN)
}

# Plot the ROC curve: FPR on x-axis, TPR on y-axis
plot(roc_data$FPrate, roc_data$TPrate, type = "l", col = "red", lwd = 2,
     xlab = "False Positive Rate", ylab = "True Positive Rate", main = "ROC Curve")

# Add a reference diagonal line (random classifier line)
abline(a = 0, b = 1, lty = 2)
```



##pROC Package

```
# Load the package
library(pROC)
```

```
## Warning: package 'pROC' was built under R version 4.4.2
```

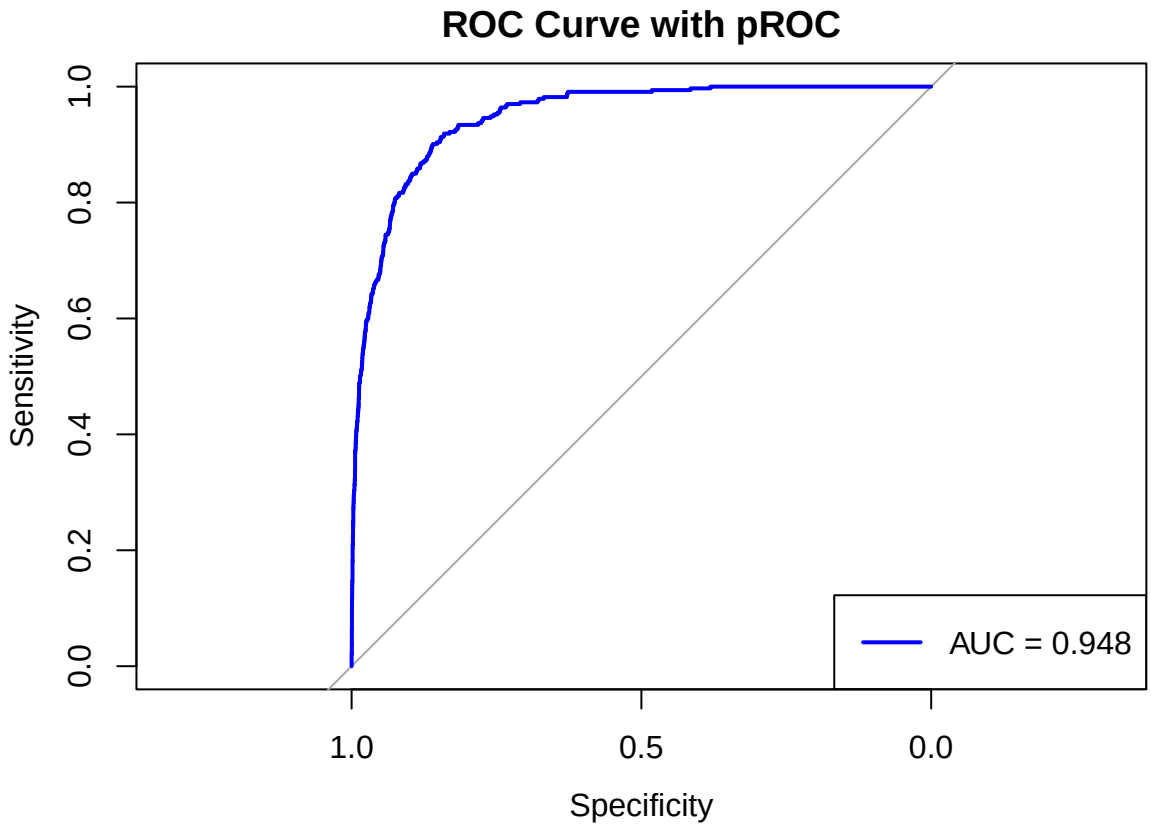
```
## Type 'citation("pROC")' for a citation.
```

```
##
## Attaching package: 'pROC'
```

```
## The following objects are masked from 'package:stats':
##
##     cov, smooth, var
```

```
# Posterior probability for "Yes" class
prob_yes <- predict(lda1)$posterior[, "Yes"]

# True binary class labels
cl_numeric <- as.numeric(Default$default) - 1

# Generate ROC object
roc_obj <- roc(response = cl_numeric,
               predictor = prob_yes,
```

```
               levels = c(0, 1),  # 0 = control, 1 = case
               direction = "<")

# Plot ROC curve
plot(roc_obj, col = "blue", lwd = 2, main = "ROC Curve with pROC")

# Add AUC to plot
auc_val <- auc(roc_obj)
legend("bottomright", legend = paste("AUC =", round(auc_val, 3)), col = "blue", lwd = 2)
```



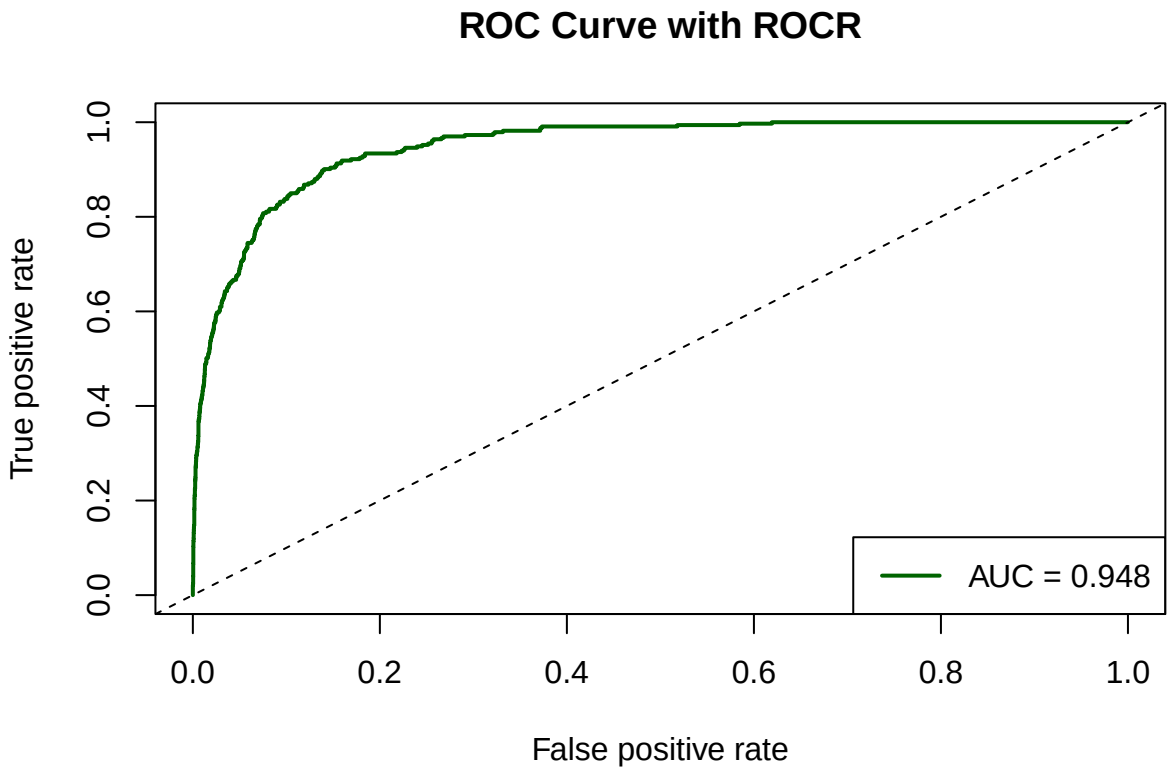
##ROCR Package

```
# Load the package
library(ROCR)
```

```
## Warning: package 'ROCR' was built under R version 4.4.3
```

```
# Create prediction and performance objects
pred <- prediction(prob_yes, cl_numeric)
perf <- performance(pred, "tpr", "fpr")

# Plot ROC curve
plot(perf, col = "darkgreen", lwd = 2, main = "ROC Curve with ROCR")
abline(a = 0, b = 1, lty = 2)
```

```r
# Calculate AUC
auc_perf <- performance(pred, measure = "auc")
auc_val <- auc_perf@y.values[[1]]
legend("bottomright", legend = paste("AUC =", round(auc_val, 3)), col = "darkgreen", lwd = 2)
```



KNN Classifier

```r
library(class)  # Load the 'class' package, which provides the knn() function

# If there is an object named 'knn' in the environment that masks the true function, remove it
if ("knn" %in% ls() && !is.function(knn)) rm(knn)

# Convert the Lag2 variable from the training dataset to a matrix format (required by knn)
train_X_knn <- as.matrix(train_data$Lag2)

# Convert the Lag2 variable from the test dataset to a matrix format
test_X_knn <- as.matrix(test_data$Lag2)

# Extract the true class labels (Direction) from the training data
train_Direction_knn <- train_data$Direction

# Set a random seed for reproducibility of the KNN prediction
set.seed(123)
```

```r
# Apply the KNN algorithm with k = 1, using the 'class' package's knn function
knn.pred_k1 <- class::knn(train = train_X_knn,
                          test = test_X_knn,
                          cl = train_Direction_knn,
                          k = 1)

# Create a confusion matrix comparing true labels (from test data) with predicted labels
conf_matrix_knn_k1 <- table(test_data$Direction, knn.pred_k1)

# Print the confusion matrix to evaluate classification performance
print(conf_matrix_knn_k1)
```

```
##       knn.pred_k1
##        Down Up
##   Down   21 22
##   Up     29 32
```

```r
knn.pred_k10 <- knn(train = train_X_knn, test = test_X_knn, cl=train_Direction_knn, k =10)
```

```r
knn.pred_k10 <- knn(train = train_X_knn, test = test_X_knn, cl=train_Direction_knn, k =10)
accuracy_k10 <- mean(knn.pred_k10 == test_data$Direction)
accuracy_k10
```

```
## [1] 0.5865385
```

Resampling Methods - Cross-Validation

##Validation Set Approach in R

```r
library(ISLR)

# Number of rows in Auto dataset
n <- nrow(Auto)

# Draw a random 50/50 split
set.seed(123)
draw <- sample(1:n, size = floor(n / 2))

# Split into training and test sets
train_data <- Auto[draw, ]
test_data <- Auto[-draw, ]

# Fit linear and quadratic models
mod1 <- lm(mpg ~ horsepower, data = train_data)
mod2 <- lm(mpg ~ horsepower + I(horsepower^2), data = train_data)

# Predict on test set
pred1_test <- predict(mod1, newdata = test_data)
mse1_test <- mean((test_data$mpg - pred1_test)^2)
```

```r
pred2_test <- predict(mod2, newdata = test_data)
mse2_test <- mean((test_data$mpg - pred2_test)^2)

# Output the mean squared errors
print(mse1_test)
```

```
## [1] 21.24991
```

```r
print(mse2_test)
```

```
## [1] 16.48112
```

##Leave One Out Cross Validation (LOOCV)

```r
MSE1_loo <- numeric(n)  # Initialize a numeric vector to store LOO MSEs for the linear model
MSE2_loo <- numeric(n)  # Initialize a numeric vector to store LOO MSEs for the quadratic model

for( i in 1:n){  # Loop over each observation in the dataset

  train_loo <- Auto[-i,]  # Exclude the i-th observation to form the training set
  test_loo <- Auto[i,]  # Use the i-th observation as the test set

  mod1_loo <- lm(mpg ~ horsepower, data = train_loo)  # Fit linear model on LOO training data
  pred1_loo <- predict(mod1_loo, newdata = test_loo)  # Predict mpg for the left-out observation using
  MSE1_loo[i] <- (test_loo$mpg - pred1_loo)^2  # Store squared prediction error for the linear model

  mod2_loo <- lm(mpg ~ horsepower + I(horsepower^2), data = train_loo)  # Fit quadratic model on LOO tr
  pred2_loo <- predict(mod2_loo, newdata = test_loo)  # Predict mpg for the left-out observation using
  MSE2_loo[i] <- (test_loo$mpg - pred2_loo)^2  # Store squared prediction error for the quadratic model

  mean_MSE1_loo <- mean(MSE1_loo)  # Calculate current mean LOO MSE for linear model
  mean_MSE2_loo <- mean(MSE2_loo)  # Calculate current mean LOO MSE for quadratic model
}

print(mean_MSE1_loo)  # Print final mean LOO MSE for the linear model
```

```
## [1] 24.23151
```

```r
print(mean_MSE2_loo)  # Print final mean LOO MSE for the quadratic model
```

```
## [1] 19.24821
```

##Using the boot for LOOCV with linear models

```r
library(boot)
```

```
##
## Attaching package: 'boot'
```

```
## The following object is masked from 'package:car':
##
##     logit
```

```
# Load the 'boot' package, which provides the cv.glm() function for cross-validation

glm.fit.linear <- glm(mpg ~ horsepower, data = Auto)
# Fit a simple linear regression model predicting mpg from horsepower

cv.err.linear <- cv.glm(Auto, glm.fit.linear)
# Perform leave-one-out cross-validation (LOOCV) on the linear model using the entire Auto dataset

print(cv.err.linear$delta[1])
```

```
## [1] 24.23151
```

```
# Print the estimated LOOCV error (delta[1]) for the linear model



glm.fit.quad <- glm(mpg ~ poly(horsepower, 2), data = Auto)
# Fit a quadratic regression model using a second-degree polynomial transformation of horsepower

cv.err.quad <- cv.glm(Auto, glm.fit.quad)
# Perform LOOCV on the quadratic model

print(cv.err.quad$delta[1])
```

```
## [1] 19.24821
```

```
# Print the estimated LOOCV error for the quadratic model
```

##k-Fold CV using boot

```
# Load the boot library for cv.glm()
library(boot)

# Fit linear regression model on Auto dataset
glm.fit.auto <- glm(mpg ~ horsepower, data = Auto)

# Perform 10-fold CV for the linear model
cv.err.10fold <- cv.glm(Auto, glm.fit.auto, K = 10)

# Print raw 10-fold CV mean squared error
print(cv.err.10fold$delta[1])
```

```
## [1] 24.24665
```

```
# Fit logistic regression model on Default dataset
glm.fit.default <- glm(default ~ income + balance,
                       data = Default,
```

```
                        family = binomial)

# Perform 10-fold CV for the logistic model
cv.err.default.10fold <- cv.glm(Default, glm.fit.default, K = 10)

# Print raw 10-fold CV error
cv.err.default.10fold$delta[1]
```

```
## [1] 0.02151599
```

```
# Define custom cost function for classification (error rate)
cost.function <- function(r, pi = 0) mean(abs(r != (pi > 0.5)))

# Apply 10-fold CV using custom cost function for classification
cv.err.classification <- cv.glm(Default, glm.fit.default,
                                K = 10, cost = cost.function)

# Print 10-fold classification error rate
cv.err.classification$delta[1]
```

```
## [1] 0.0262
```

```
# Set number of folds for manual implementation
k <- 10

# Randomly assign each observation to a fold
folds <- sample(cut(seq(1, nrow(Default)), breaks = k, labels = FALSE))

# Preallocate vector to store error rates
cv_errors <- numeric(k)

# Loop through each fold
for (j in 1:k) {
  # Identify test indices for current fold
  test_indices <- which(folds == j, arr.ind = TRUE)

  # Subset test data for the fold
  test_data_fold <- Default[test_indices, ]

  # Subset training data (all except test indices)
  train_data_fold <- Default[-test_indices, ]

  # Fit logistic model on training fold
  fit_fold <- glm(default ~ income + balance,
                  data = train_data_fold,
                  family = binomial)

  # Predict probabilities on the test fold
  probs_fold <- predict(fit_fold, newdata = test_data_fold, type = "response")

  # Convert probabilities to class predictions
  preds_fold <- ifelse(probs_fold > 0.5, "Yes", "No")
```

```r
  # Compute classification error for current fold
  cv_errors[j] <- mean(preds_fold != test_data_fold$default)
}

# Calculate mean classification error over all folds
mean_cv_error_10fold <- mean(cv_errors)

# Print final averaged 10-fold CV classification error
print(mean_cv_error_10fold)
```

```
## [1] 0.0262
```

#Resampling Methods - The Bootstrap Coefficients

```r
library(ISLR)  # Loads the ISLR package, which contains the 'Default' dataset used for this analysis.
library(boot)  # Loads the boot package, which provides functions for bootstrapping.

glm.fit.full <- glm(default ~ income + balance, data = Default, family = "binomial")
# Fits a logistic regression model predicting 'default' using 'income' and 'balance' as predictors.

#summary(glm.fit.full)
# (Commented out) Would display a summary of the fitted logistic regression model.

boot.fn.coeffs <- function(data, index){
  # Defines a function for bootstrapping. It takes a dataset and a vector of row indices.

  fit <- glm(default ~ income + balance, data = data[index,], family = "binomial")
  # Fits a logistic regression model on the bootstrap sample defined by the given indices.

  return(coef(fit))
  # Returns the estimated coefficients from the fitted model.
}

set.seed(123)
# Sets the random seed to ensure reproducibility of the bootstrap results.

boot_results <- boot(data = Default, statistic = boot.fn.coeffs, R = 1000)
# Performs bootstrap resampling 1000 times using 'boot.fn.coeffs' on the 'Default' dataset.

print(boot_results)
```

```
##
## ORDINARY NONPARAMETRIC BOOTSTRAP
##
##
## Call:
## boot(data = Default, statistic = boot.fn.coeffs, R = 1000)
##
##
## Bootstrap Statistics :
##          original        bias     std. error
## t1* -1.154047e+01 -2.754771e-02 4.204817e-01
```

```
## t2*  2.080898e-05  1.582518e-07 4.729534e-06
## t3*  5.647103e-03  1.296980e-05 2.217214e-04
```

```
# Displays the results of the bootstrap procedure, including estimated standard errors.
```

Bootstrap SE for the sample median

```
library(MASS)  # Loads the MASS package, which includes the 'Boston' housing dataset.
library(boot)  # Loads the boot package for performing bootstrap resampling.

medv.median.original <- median(Boston$medv)
# Calculates the original sample median of the 'medv' variable (median home value) from the Boston data

boot.fn.median <- function(data_vector, index){
  # Defines a function for the bootstrap procedure that computes the median of a sample.

  return(median(data_vector[index]))
  # Returns the median of the resampled data points.
}

set.seed(123)
# Sets a seed to ensure reproducibility of the bootstrap results.

boot_median_results <- boot(data = Boston$medv, statistic = boot.fn.median, R = 1000)
# Performs the bootstrap with 1000 resamplings of the 'medv' variable using the defined function.

print(boot_median_results)
```

```
##
## ORDINARY NONPARAMETRIC BOOTSTRAP
##
##
## Call:
## boot(data = Boston$medv, statistic = boot.fn.median, R = 1000)
##
##
## Bootstrap Statistics :
##     original  bias    std. error
## t1*     21.2 -0.0203   0.3676453
```

```
# Prints the bootstrap results, including the original statistic and estimated standard error.
```

Bootstrap Confidence Intervals using boot.ci

```
conf_intervals <- boot.ci(boot_results, type = c("norm", "perc"), index = 2)
# Computes confidence intervals for the second parameter (index = 2) in the bootstrap results object 'b
# using the normal and percentile methods. The BCa method is excluded due to potential estimation issue

print(conf_intervals)
```

```
## BOOTSTRAP CONFIDENCE INTERVAL CALCULATIONS
## Based on 1000 bootstrap replicates
##
## CALL :
## boot.ci(boot.out = boot_results, type = c("norm", "perc"), index = 2)
##
## Intervals :
## Level      Normal             Percentile
## 95%   ( 0,  0 )   ( 0,  0 )
## Calculations and Intervals on Original Scale
```

```
# Displays the computed confidence intervals for the selected coefficient.

boot.ci(boot_median_results, type = c("norm", "perc", "bca"))
```

```
## BOOTSTRAP CONFIDENCE INTERVAL CALCULATIONS
## Based on 1000 bootstrap replicates
##
## CALL :
## boot.ci(boot.out = boot_median_results, type = c("norm", "perc",
##     "bca"))
##
## Intervals :
## Level      Normal             Percentile            BCa
## 95%   (20.50, 21.94 )   (20.50, 21.80 )   (20.40, 21.70 )
## Calculations and Intervals on Original Scale
```

```
# Computes and displays confidence intervals for the bootstrap median estimates,
# using normal, percentile, and BCa methods. This usually works well for scalar statistics like the med
```

Probability an Observation is in a Bootstrap Sample

```
pr_in_bootstrap <- function(n){
  return(1-(1-1/n)^n)

}

n_values <- c(5,100,10000)

sapply(n_values, pr_in_bootstrap)
```

```
## [1] 0.6723200 0.6339677 0.6321390
```

Mean and SE of Mean for Bostonmedv

```
library(MASS)
data(Boston)
```

```r
mu_hat_medv <- mean(Boston$medv)
se_mu_hat_formula <- sd(Boston$medv)/ sqrt(nrow(Boston))

mu_hat_medv
```

```
## [1] 22.53281
```

```r
se_mu_hat_formula
```

```
## [1] 0.4088611
```

Bootstrap for SE of Mean for Bostonmedv

```r
boot.fn_mean <- function(data_vector, index){
  # Defines a function that computes the mean of a bootstrap sample.
  # It takes a data vector and a set of indices, then returns the sample mean.

  return(mean(data_vector[index]))
  # Calculates and returns the mean of the data points selected by the bootstrap indices.
}

set.seed(1)
# Sets the random seed to ensure reproducibility of the bootstrap results.

boot_mean_results <- boot(Boston$medv, boot.fn_mean, R = 1000)
# Performs bootstrap resampling 1000 times on the 'medv' variable from the Boston dataset,
# using the custom function 'boot.fn_mean' to compute the sample mean in each resample.

print(boot_mean_results)
```

```
##
## ORDINARY NONPARAMETRIC BOOTSTRAP
##
##
## Call:
## boot(data = Boston$medv, statistic = boot.fn_mean, R = 1000)
##
##
## Bootstrap Statistics :
##     original      bias    std. error
## t1* 22.53281 0.007650791   0.4106622
```

```r
# Displays the results of the bootstrap procedure,
# including the original sample mean, estimated bias, and standard error of the mean.
```

Bootstrap SE for 10th Percentile of Bostonmedv

```r
boot.fn_q10 <- function(data_vector, index){
  # Defines a function that computes the 10th percentile (quantile at 0.1) from a bootstrap sample.
  # It takes a numeric data vector and a vector of bootstrap indices as input.

  return(quantile(data_vector[index], probs = 0.1))
  # Calculates and returns the 10th percentile of the resampled data points.
}

set.seed(1)
# Sets the random seed to ensure reproducibility of the bootstrap results.

boot_q10_results <- boot(Boston$medv, boot.fn_q10, R = 1000)
# Performs bootstrap resampling 1000 times on the 'medv' variable from the Boston dataset,
# using the custom function 'boot.fn_q10' to compute the 10th percentile for each resample.

print(boot_q10_results)
```

```
##
## ORDINARY NONPARAMETRIC BOOTSTRAP
##
##
## Call:
## boot(data = Boston$medv, statistic = boot.fn_q10, R = 1000)
##
##
## Bootstrap Statistics :
##     original  bias    std. error
## t1*    12.75  0.0339   0.4767526
```

```r
# Displays the results of the bootstrap procedure,
# including the original 10th percentile, estimated bias, and standard error.
```

Best Subset Selection with leaps::regsubsets

```r
Auto$age <- 83 - Auto$year
# Oppretter en ny variabel 'age' i datasettet Auto. Den representerer alderen til bilen, antatt basert

Auto_subset <- Auto[, !(names(Auto) %in% c("name", "origin", "year"))]
# Fjerner variablene 'name', 'origin' og 'year' fra datasettet, ettersom disse ikke skal inngå i regres

library(leaps)
```

```
## Warning: package 'leaps' was built under R version 4.4.3
```

```r
# Laster pakken 'leaps', som inneholder funksjoner for subset selection i regresjonsanalyse.

regfit.full <- regsubsets(mpg ~ ., data = Auto_subset, nvmax = 6)
# Kjører 'best subset selection' på mpg (miles per gallon) med de gjenværende variablene i datasettet.
# nvmax = 6 angir at vi ønsker å vurdere modeller med opptil 6 prediktorer.
```

```r
reg.summary <- summary(regfit.full)
# Lager en oppsummering av subset-seleksjonen med statistikker som R², justert R², Mallows' Cp og BIC f

print(reg.summary)
```

```
## Subset selection object
## Call: regsubsets.formula(mpg ~ ., data = Auto_subset, nvmax = 6)
## 6 Variables  (and intercept)
##              Forced in Forced out
## cylinders        FALSE      FALSE
## displacement     FALSE      FALSE
## horsepower       FALSE      FALSE
## weight           FALSE      FALSE
## acceleration     FALSE      FALSE
## age              FALSE      FALSE
## 1 subsets of each size up to 6
## Selection Algorithm: exhaustive
##          cylinders displacement horsepower weight acceleration age
## 1  ( 1 ) " "       " "          " "        "*"    " "          " "
## 2  ( 1 ) " "       " "          " "        "*"    " "          "*"
## 3  ( 1 ) " "       " "          " "        "*"    "*"          "*"
## 4  ( 1 ) " "       "*"          " "        "*"    "*"          "*"
## 5  ( 1 ) "*"       "*"          " "        "*"    "*"          "*"
## 6  ( 1 ) "*"       "*"          "*"        "*"    "*"          "*"
```

```r
# Skriver ut hele oppsummeringen av modellutvalget.

names(reg.summary)
```

```
## [1] "which"  "rsq"    "rss"    "adjr2"  "cp"     "bic"    "outmat" "obj"
```

```r
# Viser hvilke elementer som er tilgjengelige i reg.summary, f.eks. 'which', 'rsq', 'adjr2', 'bic', osv
```

##Shows which variables are in the best model of each size

```r
cbind(reg.summary$which[,-1], adjR2 = round(reg.summary$adjr2,4))
```

```
##   cylinders displacement horsepower weight acceleration age  adjR2
## 1         0            0          0      1            0   0 0.6918
## 2         0            0          0      1            0   1 0.8072
## 3         0            0          0      1            1   1 0.8071
## 4         0            1          0      1            1   1 0.8068
## 5         1            1          0      1            1   1 0.8068
## 6         1            1          1      1            1   1 0.8063
```

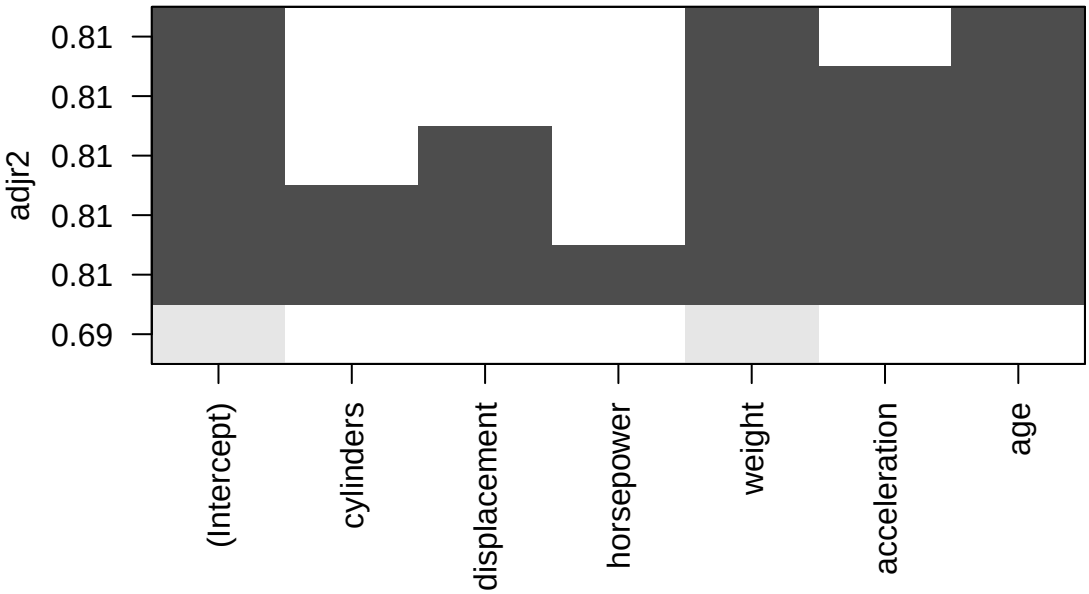
Forward Stepside Selection with leaps::regsubsets

```r
regfit.fwd <- regsubsets(mpg ~ ., data = Auto_subset, nvmax = 6, method = "forward")
# Performs forward stepwise subset selection to predict 'mpg' using all predictors in 'Auto_subset'.
# It considers models with up to 6 variables (nvmax = 6).

summary.fwd <- summary(regfit.fwd)
# Stores the summary of the forward stepwise regression object, including R², adjusted R², Cp, and BIC.

cbind(summary.fwd$which[,-1], adjR2 = round(summary.fwd$adjr2, 4))
```

```
##   cylinders displacement horsepower weight acceleration age  adjR2
## 1         0            0          0      1            0   0 0.6918
## 2         0            0          0      1            0   1 0.8072
## 3         0            0          0      1            1   1 0.8071
## 4         0            1          0      1            1   1 0.8068
## 5         1            1          0      1            1   1 0.8068
## 6         1            1          1      1            1   1 0.8063
```

```r
# Combines a matrix of which variables are included in each model (excluding the intercept column)
# with the corresponding adjusted R² values, rounded to 4 decimals.
# This makes it easy to compare variable combinations with their adjusted R² scores.
```

```r
plot(regfit.fwd, scale = "adjr2", col = gray.colors(10))
```



```
# Plots the models selected by forward stepwise selection, with adjusted R² as the evaluation metric.
# Each row represents a model size (1 to nvmax), and filled boxes indicate included variables.
# The gray scale is used for visual clarity.
```

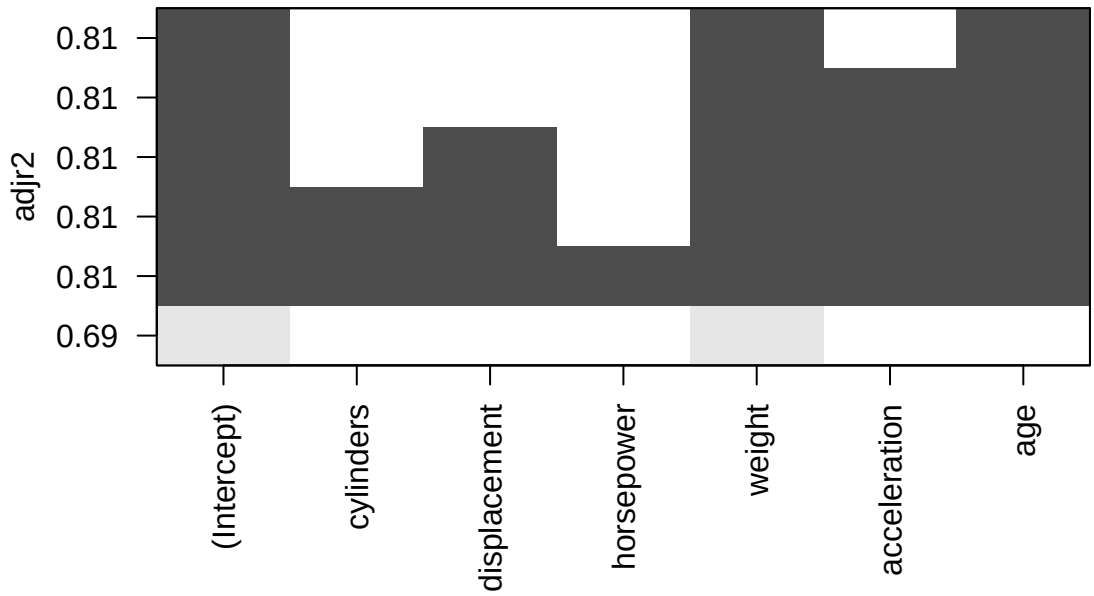
Backward Stepside Selection with leaps::regsubsets

```
regfit.bwd <- regsubsets(mpg ~ ., data = Auto_subset, nvmax = 6, method = "backward")
# Performs backward stepwise subset selection to predict 'mpg' using all predictors in 'Auto_subset'.
# It considers models with up to 6 variables (nvmax = 6).

summary.bwd <- summary(regfit.bwd)
# Stores the summary of the forward stepwise regression object, including R², adjusted R², Cp, and BIC.

cbind(summary.bwd$which[,-1], adjR2 = round(summary.bwd$adjr2, 4))
```

```
##   cylinders displacement horsepower weight acceleration age  adjR2
## 1         0            0          0      1            0   0 0.6918
## 2         0            0          0      1            0   1 0.8072
## 3         0            0          0      1            1   1 0.8071
## 4         0            1          0      1            1   1 0.8068
## 5         1            1          0      1            1   1 0.8068
## 6         1            1          1      1            1   1 0.8063
```

```
# Combines a matrix of which variables are included in each model (excluding the intercept column)
# with the corresponding adjusted R² values, rounded to 4 decimals.
# This makes it easy to compare variable combinations with their adjusted R² scores.
```

```
plot(regfit.bwd, scale = "adjr2", col = gray.colors(10))
```




```
# Plots the models selected by forward stepwise selection, with adjusted R² as the evaluation metric.
# Each row represents a model size (1 to nvmax), and filled boxes indicate included variables.
# The gray scale is used for visual clarity.
```

OLS Regression Example

```
set.seed(123)

n <- 100
p_vars <- 10
X_sim <- matrix(0, n, p_vars)

for(j in 1:p_vars) X_sim[,j] <- rnorm(n, sd= j)

beta_sim <- 2:(p_vars+1)
e_sim <- rnorm(n, sd = 200)

y_sim <- 1 + X_sim %*% beta_sim + e_sim  # <-- bruk matriseprodukt

# Sørg for at y_sim er en vektor, ikke en matrise
y_sim <- as.vector(y_sim)

# Klassisk OLS
ols.fit <- lm(y_sim ~ X_sim)
```

Ridge Example

```
# Ridge-regresjon
library(glmnet)
```

```
## Warning: package 'glmnet' was built under R version 4.4.3
```

```
## Loading required package: Matrix
```

```
##
## Attaching package: 'Matrix'
```

```
## The following objects are masked from 'package:tidyr':
##
##     expand, pack, unpack
```

```
## Loaded glmnet 4.1-8
```

```
ridge.fit_small_lambda <- glmnet(X_sim, y_sim, alpha = 0, lambda = 0.01)

# Sammenligning av koeffisienter
ols_coefs <- coef(ols.fit)
ridge_coefs <- coef(ridge.fit_small_lambda)

cbind(OLS = ols_coefs, Ridge = ridge_coefs)
```

```
## 11 x 2 sparse Matrix of class "dgCMatrix"
##                    OLS        s0
## (Intercept) 28.384879 28.385361
## X_sim1      14.920853 14.918499
## X_sim2       2.572595  2.571485
## X_sim3      -5.809090 -5.809277
## X_sim4      11.906594 11.906435
## X_sim5       8.807303  8.806624
## X_sim6       5.756564  5.756556
## X_sim7      10.655767 10.655310
## X_sim8      11.741306 11.740911
## X_sim9       8.900012  8.899606
## X_sim10     13.740091 13.739608
```

Choosing a lambda by LOOCV

```
cv.ridge <- cv.glmnet(X_sim, y_sim, alpha = 0, nfolds = n)
```

```
## Warning: Option grouped=FALSE enforced in cv.glmnet, since < 3 observations per
## fold
```

```r
lambda_min_ridge <- cv.ridge$lambda.min

ridgemin.fit <- glmnet(X_sim, y_sim, alpha = 0 , lambda =lambda_min_ridge)

cbind(coef(ols.fit), coef(ridgemin.fit))
```

```
## 11 x 2 sparse Matrix of class "dgCMatrix"
##                               s0
## (Intercept) 28.384879 29.129747
## X_sim1      14.920853 13.168075
## X_sim2       2.572595  1.440477
## X_sim3      -5.809090 -5.907230
## X_sim4      11.906594 11.261095
## X_sim5       8.807303  7.538552
## X_sim6       5.756564  5.506364
## X_sim7      10.655767  9.407275
## X_sim8      11.741306 10.632123
## X_sim9       8.900012  7.831225
## X_sim10     13.740091 12.427516
```