R Notebook

```r
library("ISLR2")
```

```
## Warning: package 'ISLR2' was built under R version 4.3.3
```

```r
data("Hitters")
d<-Hitters
```

1.1.i

```r
fn.mse <- function(par,x,y) {
  if (!is.matrix(x)) {
    stop("x must be a matrix.")
  }
  if (!is.vector(y)) {
    stop("y must be a vector.")
  }
  if (length(y) != length(x[,1])) {
    stop("The size of y and x does not match.")
  }
  if (!is.vector(par)) {
    stop("par must be a vector.")
  }
  if (length(par) != length(x[1,])) {
    stop("The size of par and x does not match.")
  }

  eta <- x %*% matrix(par,length(par),1)
  mse <- mean((y-eta)^2)

  return(mse)
}
```

Answer 1.1.i

Row 2-4:
Checks if input ‘x’ (observation) is a matrix. If not a matrix, function stops and prints “x must be a matrix.”

Row 5-7:
Checks if input ‘y’ (target) is a vector. If not a matrix, function stops and prints “y must be a vector.”

Row 8-10:
Ensures that length of ‘y’ matches the number of rows of ‘x’. If not, stops and prints “The size of y and x does not match.”

Row 11-16:
Check if input ‘par’ (parameter) is a vector, if not prints “par must be a vector.”, and its length is same as number of rows of ‘x’, if not prints “The size of par and x does not match.”

Row 18:
Computes the predicted values eta by multiplying the matrix ‘x’ with the parameter vector ‘par’. This results in a vector of predicted values for the dependent variable.

1.1.ii

```
fn.lm <- function(formula,data) {
  d  <- na.omit(data)
  mf <- model.frame(formula,data=d)

  x <- model.matrix(formula,data=d)
  y <- model.extract(mf,"response")
  guess <- rep(0,length(x[1,]))

  aux <- optim(guess,fn=fn.mse,x=x,y=y,method="BFGS")
  if (aux$convergence != 0) {
    warning("Method did not converge.")
  }

  names(aux$par) <- colnames(x)
  return(list(mse=aux$value,beta=aux$par))
}
```

Answer 1.1.ii

Row 2-7: Data Preparation
Row 2 removes any rows with missing values
Row 3 create model frame ‘mf’
Row 5 create model matrix ‘x’, and then extracts the matrix
Row 6 extracts response variable ‘y’ from ‘mf’
Row 7: Initializes a vector ‘guess’ of zeroes for parameters, which are the starting values for optimisation algorithm to minimize the loss function.

Row 9: Uses the ‘optim’ function to minimize MSE, employing BFGS optimation method, and storing it in ‘aux’

1.1.iii

```
estimated_model <- fn.lm(Salary ~ ., d)

lm_model <- lm(Salary ~ ., data=d)

mse_estimated <- estimated_model$mse
coefficients_estimated <- estimated_model$beta
coefficients_lm <- coef(lm_model)
```

```
print(mse_estimated)
```

```
## [1] 92017.87
```

```
print("-------------------")
```

```
## [1] "-------------------"
```

```
print("-------------------")
```

```
## [1] "-------------------"
```

```
print(coefficients_estimated)
```

```
##   (Intercept)        AtBat         Hits        HmRun         Runs
RBI
##   163.0948541   -1.9798608    7.5007932    4.3309719   -2.3761654    -
1.0450216
##         Walks        Years       CAtBat        CHits       CHmRun
CRuns
##     6.2312205   -3.4888030   -0.1713398    0.1339814   -0.1728862
1.4543097
##          CRBI       CWalks      LeagueN    DivisionW      PutOuts
Assists
##     0.8077195   -0.8115658   62.6138746 -116.8473176    0.2818927
0.3710642
##        Errors   NewLeagueN
##    -3.3607099  -24.7758992
```

```
print("-------------------")
```

```
## [1] "-------------------"
```

```
print(coefficients_lm)
```

```
##   (Intercept)        AtBat         Hits        HmRun         Runs
RBI
##   163.1035878   -1.9798729    7.5007675    4.3308829   -2.3762100    -
```

```
1.0449620
##        Walks        Years       CAtBat        CHits       CHmRun
CRuns
##    6.2312863   -3.4890543   -0.1713405    0.1339910   -0.1728611
1.4543049
##         CRBI       CWalks      LeagueN    DivisionW      PutOuts
Assists
##    0.8077088   -0.8115709   62.5994230 -116.8492456    0.2818925
0.3710692
##       Errors   NewLeagueN
##   -3.3607605  -24.7623251
```

Answer 1.1.iii

MSE using fn.lm is 92017.87

Values of estimated parameters using fn.lm -

Variable	Coefficient
Intercept	163.0948541
AtBat	-1.9798608
Hits	7.5007932
HmRun	4.3309719
Runs	-2.3761654
RBI	-1.0450216
Walks	6.2312205
Years	-3.4888030
CAtBat	-0.1713398
CHits	0.1339814
CHmRun	-0.1728862
CRuns	1.4543097
CRBI	0.8077195
CWalks	-0.8115658
LeagueN	62.6138746
DivisionW	-116.8473176
PutOuts	0.2818927
Assists	0.3710642
Errors	-3.3607099
NewLeagueN	-24.7758992

Values of estimated parameters using R function lm -

Variable	Coefficient
Intercept	163.1035878
AtBat	-1.9798729
Hits	7.5007675
HmRun	4.3308829
Runs	-2.3762100
RBI	-1.0449620
Walks	6.2312863
Years	-3.4890543
CAtBat	-0.1713405
CHits	0.1339910
CHmRun	-0.1728611
CRuns	1.4543049
CRBI	0.8077088
CWalks	-0.8115709
LeagueN	62.5994230
DivisionW	-116.8492456
PutOuts	0.2818925
Assists	0.3710692
Errors	-3.3607605
NewLeagueN	-24.7623251

Comparing both tables, we can see that results from fn.lm, and R function lm, are very similar, indicating good performance of fn.lm.

1.2.i

```
fn.loss.lasso <- function(par,x,y,lambda=0) {
  res <- (fn.mse(par,x,y)
          +(lambda*sum(abs(par[-1]))))
  return(res)
}
```

Answer 1.2.i

The function **`fn.loss.lasso`** computes the loss for a LASSO regression model. It is the sum of the mean squared error (MSE) and regularization term. This regularization term is **`lambda * sum(abs(par[-1]))`**, where **`lambda`** is the regularization parameter, and **`par[-1]`** is used to exclude the intercept from the LASSO penalty.

1.2.ii

```
fn.lm.lasso <- function(formula,data,lambda=0,
                        guess=NULL) {
  d <- na.omit(data)
  y <- model.extract(model.frame(formula,data=d),"response")
  x <- model.matrix(formula,data=d)
  if (is.null(guess)) {
    guess <- rep(0,length(x[1,]))
  }

  aux <- optim(guess,fn=fn.loss.lasso,x=x,y=y,
               lambda=lambda,method="BFGS",
               control=list(maxit=2000))
  if (aux$convergence != 0) {
    warning("Method did not converge.")
  }

  names(aux$par) <- colnames(x)
  return(list(loss=aux$value,beta=aux$par))
}

#making a function for reusability in future questions
standardize_features <- function(df) {
    numeric_columns <- sapply(df, is.numeric)
    df[numeric_columns] <- scale(df[numeric_columns])
    return(as.data.frame(df))
}

d_standard <- standardize_features(d)
lambda_value <- 0
lasso_model <- fn.lm.lasso(Salary ~ ., d_standard, lambda=lambda_value) # y~x
as given
print(lasso_model)

## $loss
## [1] 0.4521584
##
## $beta
## (Intercept)       AtBat        Hits       HmRun        Runs         RBI
##  0.05169551 -0.67359664  0.77259102  0.08345829 -0.13700251 -0.06042384
##       Walks       Years      CAtBat       CHits      CHmRun       CRuns
##  0.29892403 -0.03820612 -0.88144390  0.19486492 -0.03211987  1.07626696
##        CRBI      CWalks     LeagueN   DivisionW     PutOuts     Assists
##  0.59490495 -0.48045922  0.13895257 -0.25902190  0.17542876  0.11257389
##      Errors  NewLeagueN
## -0.04745785 -0.05508263
```

Answer 1.2.ii

MSE of the lasso model:
0.4522

Values of estimated parameters using R function lm:

Variable	Coefficient
Intercept	0.05169551
AtBat	-0.67359664
Hits	0.77259102
HmRun	0.08345829
Runs	-0.13700251
RBI	-0.06042384
Walks	0.29892403
Years	-0.03820612
CAtBat	-0.88144390
CHits	0.19486492
CHmRun	-0.03211987
CRuns	1.07626696
CRBI	0.59490495
CWalks	-0.48045922
LeagueN	0.13895257
DivisionW	-0.25902190
PutOuts	0.17542876
Assists	0.11257389
Errors	-0.04745785
NewLeagueN	-0.05508263

These estimated parameters are in range of -1 and 1, as compared to parameters in 1.1.iii, which had broader range. This shows the effect of standardization, helps in comparing parameters.

1.3

```
fn.split.kfold <- function(df,k) {
  aux <- 1:length(df[,1])

  aux1   <- floor(length(aux) / k)
  aux2   <- length(aux) %% k
  n.fold <- rep(aux1,k)
  if (aux2 > 0) {
```

```r
    n.fold[1:aux2] <- aux1+1
  }

  fold <- vector("list",k)
  aux.id <- aux
  for (i in 1:(k-1)) {
    id <- sort(sample(aux.id,size=n.fold[i],replace=FALSE))
    fold[[i]]$n     <- n.fold[i]
    fold[[i]]$id    <- id
    fold[[i]]$train <- df[aux[-id],]
    fold[[i]]$test  <- df[id,]
    aux.id <- aux.id[!(aux.id %in% id)]
  }
  fold[[k]]$n     <- n.fold[k]
  fold[[k]]$id    <- aux.id
  fold[[k]]$train <- df[aux[-aux.id],]
  fold[[k]]$test  <- df[aux.id,]

  return(fold)
}

gs.lasso <- function(formula,
                     data,
                     lambda=seq(0,5,length.out=10),
                     kfold=5,
                     std=TRUE) {
  if (kfold < 2) {
    stop("kfold must be greater than or equal to 2.")
  }

  d <- na.omit(data)
  y <- model.extract(model.frame(formula,data=d),"response")
  if (std) {
    for (i in 1:length(d[1,])) {
      if (is.numeric(d[,i])) {
        d[,i] <- (d[,i]-mean(d[,i]))/sd(d[,i])
      }
    }
  }
  x <- model.matrix(formula,data=d)
  d <- data.frame(y,x[,-1])
  lambda <- sort(lambda,decreasing=FALSE)

  train.mse  <- rep(NA,length(lambda))
  train.loss <- rep(NA,length(lambda))
  beta       <- matrix(NA,length(lambda),length(x[1,]))
  colnames(beta) <- colnames(x)
  guess      <- lm(y ~ .,data=d)$coefficients
  for (i.lambda in 1:length(lambda)) {
```

```
  aux <- fn.lm.lasso(y ~ .,data=d,lambda=lambda[i.lambda],
                    guess=guess)
  guess <- aux$beta
  beta[i.lambda,]      <- aux$beta
  train.mse[i.lambda]  <- fn.mse(aux$beta,x,y)
  train.loss[i.lambda] <- aux$loss
}


d.fold <- fn.split.kfold(na.omit(data),kfold)
cv.error <- matrix(NA,length(d[,1]),length(lambda))
for (i.fold in 1:kfold) {
  n     <- d.fold[[i.fold]]$n
  id    <- d.fold[[i.fold]]$id
  train <- d.fold[[i.fold]]$train
  test  <- d.fold[[i.fold]]$test

  y.train <- model.extract(model.frame(formula,data=train),"response")
  y.test  <- model.extract(model.frame(formula,data=test),"response")
  if (std) {
    for (i in 1:length(train[1,])) {
      if (is.numeric(train[,i])) {
        test[,i]  <- ( test[,i]-mean(train[,i]))/sd(train[,i])
        train[,i] <- (train[,i]-mean(train[,i]))/sd(train[,i])
      }
    }
  }
  x.train <- model.matrix(formula,data=train)
  train   <- data.frame(y.train,x.train[,-1])
  x.test  <- model.matrix(formula,data=test)

  for (i.lambda in 1:length(lambda)) {
    aux <- fn.lm.lasso(y.train ~ .,data=train,
                       lambda=lambda[i.lambda],
                       guess=beta[i.lambda,])
    cv.error[id,i.lambda] <- fn.mse(aux$beta,x.test,y.test)
  }
}
cv.mse <- apply(cv.error,2,mean)

res <- cbind(lambda,train.loss,train.mse,cv.mse)
colnames(res) <- c("lambda","loss.train","mse.train","mse.cv")
res <- res[order(res[,1]),]

id <- which.min(res[,4])
return(list(best=list(lambda = res[id,1],
                      mse = res[id,3],
                      beta = beta[id,]),
```

```
                 all=list(mse = res, beta = beta)))
}
```

1.3.i

```
set.seed(34064064)

lambda_grid <- exp(seq(-2, 4, length.out=30))

lasso_results <- gs.lasso(Salary ~ ., data = d_standard, lambda =
lambda_grid, kfold = 10, std = TRUE)

id <- which.min(lasso_results$all$mse[,4])
best_lambda <- lasso_results$best$lambda
best_train_mse <- lasso_results$best$mse
best_cv_mse <- lasso_results$all$mse[id, "mse.cv"]
best_coefficients <- lasso_results$best$beta
print(paste("Best Lambda:", best_lambda))

## [1] "Best Lambda: 0.135335283236613"

print(paste("Best Train MSE:", best_train_mse))

## [1] "Best Train MSE: 0.536141207349947"

print(paste("Best Cross-validation MSE:", best_cv_mse))

## [1] "Best Cross-validation MSE: 0.569225257206612"

print("Estimated values of the parameters:")

## [1] "Estimated values of the parameters:"

print(best_coefficients)

##   (Intercept)         AtBat          Hits         HmRun          Runs
##  3.031505e-02  3.004736e-04  1.743113e-01  5.633848e-04  8.836302e-04
##           RBI         Walks         Years        CAtBat         CHits
##  6.780789e-04  1.019672e-01  1.376374e-04  7.162632e-04  2.754872e-02
##        CHmRun         CRuns          CRBI        CWalks       LeagueN
##  1.816204e-02  1.485996e-01  2.292528e-01  4.509272e-04  2.530706e-04
##     DivisionW       PutOuts       Assists        Errors    NewLeagueN
## -5.539532e-02  1.145977e-01 -5.813989e-05 -4.435345e-04  1.997111e-04
```

Answer 1.3.i

Best Lambda: 0.1353

MSE Values:

- **Best Train MSE:** 0.5361

- **Best Cross-validation MSE:** 0.5692

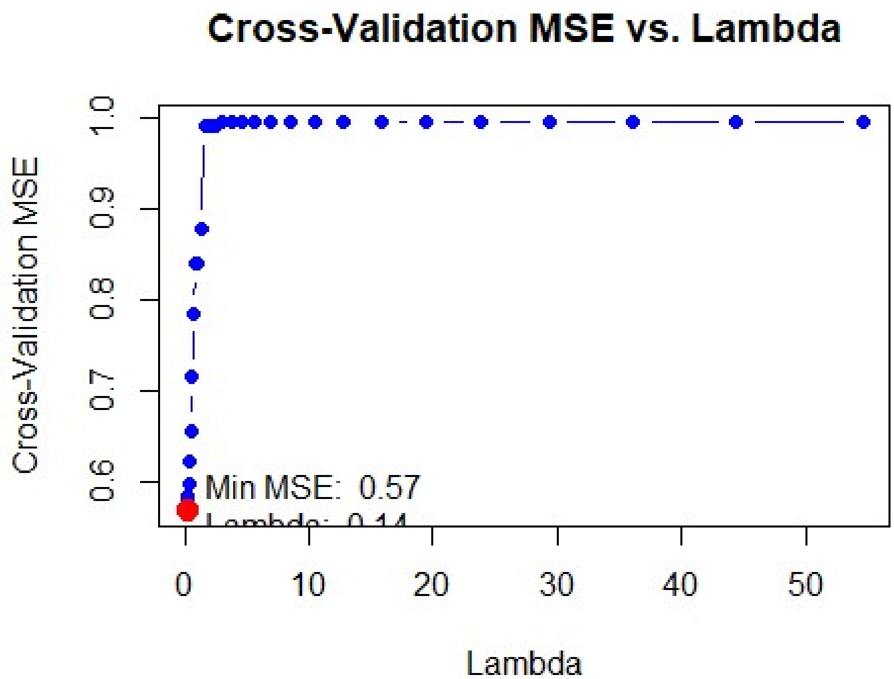
Estimated Parameter Values:

Parameter	Estimated Value
Intercept	0.03031505
AtBat	0.0003004736
Hits	0.1743113
HmRun	0.0005633848
Runs	0.0008836302
RBI	0.0006780789
Walks	0.1019672
Years	0.0001376374
CAtBat	0.0007162632
CHits	0.02754872
CHmRun	0.01816204
CRuns	0.1485996
CRBI	0.2292528
CWalks	0.0004509272
LeagueN	0.0002530706
DivisionW	-0.05539532
PutOuts	0.1145977
Assists	-0.00005813989
Errors	-0.0004435345
NewLeagueN	0.0001997111

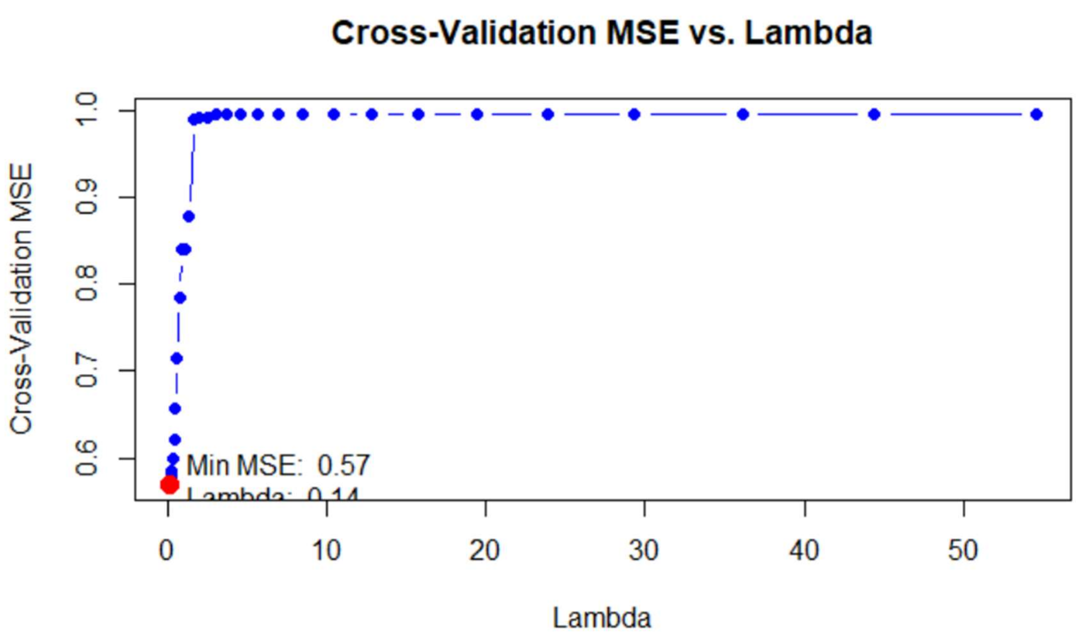
1.3.ii

```
lambda_values <- lasso_results$all$mse[,1]
cv_mse_values <- lasso_results$all$mse[,4]

plot(lambda_values, cv_mse_values, type = 'b', col = 'blue', pch = 19,
     xlab = "Lambda", ylab = "Cross-Validation MSE", main = "Cross-Validation
MSE vs. Lambda")

min_index <- which.min(cv_mse_values)
best_lambda <- lambda_values[min_index]
min_mse <- cv_mse_values[min_index]

points(best_lambda, min_mse, col = 'red', pch = 19, cex = 1.5)
text(best_lambda, min_mse, labels = paste("Min MSE: ", round(min_mse, 2),
```

```
"\nLambda: ", round(best_lambda, 2)),
     pos = 4)  # Adjust position as needed
```



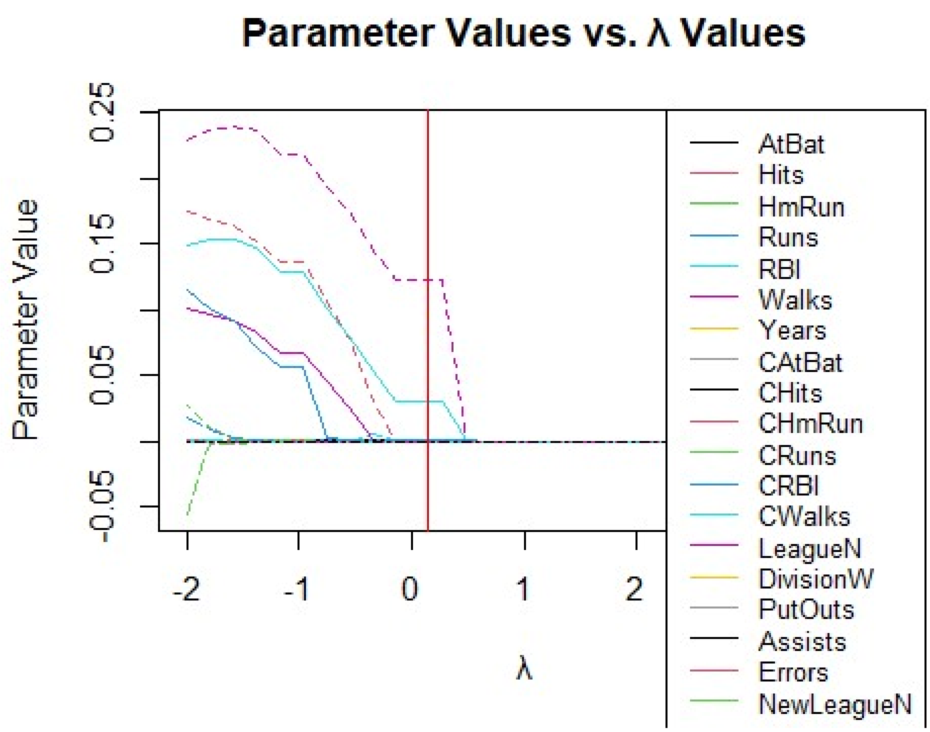
Answer 1.3.ii



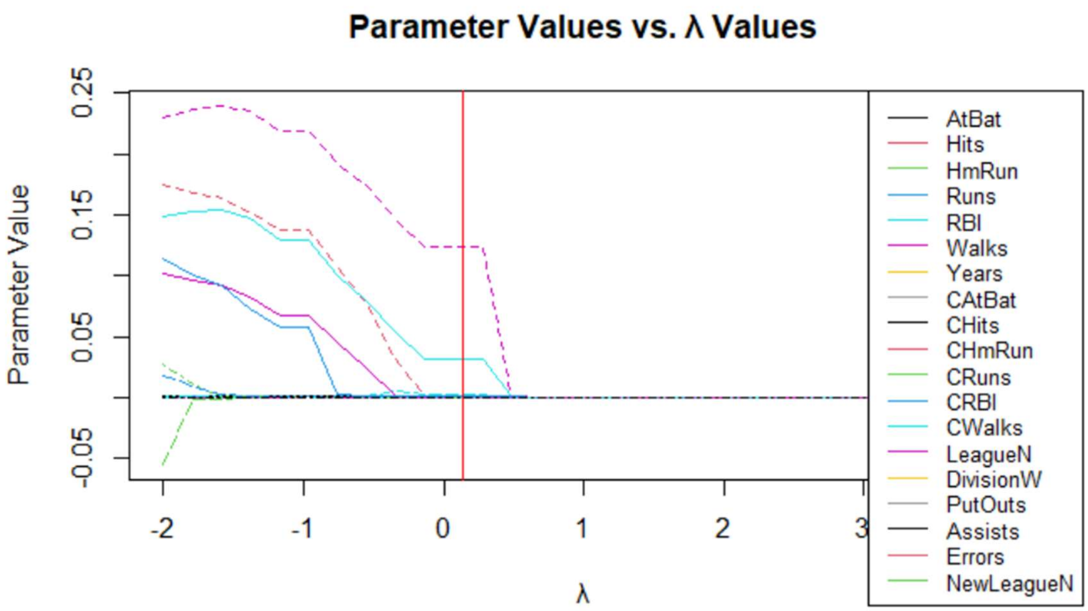
- **Lowest MSE:** The minimum MSE achieved is 0.57.
- **Lambda for Min MSE:** The lambda value corresponding to this lowest MSE is approximately 0.14.

1.3.iii

```
beta_vals <- lasso_results$all$beta[,-1]
matplot(log(lambda_values), (beta_vals), type = "l", xlab = "λ", ylab =
"Parameter Value", main = "Parameter Values vs. λ Values")
abline(v = best_lambda, col = "red")
legend("topright", legend = colnames(beta_vals), col = 1:ncol(beta_vals), lty
= 1, cex = 0.8, xpd = TRUE, inset = c(-0.05, 0))
```



Answer 1.3.iii



As we can see in the plot, the lambda is close to 0.14.

1.4

```r
library(glmnet)

## Warning: package 'glmnet' was built under R version 4.3.3

## Loading required package: Matrix

## Loaded glmnet 4.1-8

data(Hitters)
d <- na.omit(Hitters)
d_standardized_glm = standardize_features(d)

x <- model.matrix(Salary ~ ., data = d_standardized_glm)[, -1]
y <- d_standardized_glm$Salary

lambda_grid <- exp(seq(-2, 4, length.out = 30))

set.seed(34064064)

cv_fit <- cv.glmnet(x, y, alpha = 1, lambda = lambda_grid, nfolds = 10,
family = "gaussian")

best_lambda <- cv_fit$lambda.min
best_mse_cv <- min(cv_fit$cvm)
```

```
best_coefficients <- coef(cv_fit, s = "lambda.min", exact = TRUE)

fit_best_lambda <- glmnet(x, y, alpha = 1, lambda = c(cv_fit$lambda.min))

predictions <- predict(fit_best_lambda, newx = x, s = cv_fit$lambda.min)

train_mse_best <- mean((y - predictions)^2)



print(best_lambda)

## [1] 0.1353353

print(train_mse_best)

## [1] 0.5664791

print(best_mse_cv)

## [1] 0.6316845

print(list(as.numeric(best_coefficients)))

## [[1]]
##  [1]  0.02621250  0.00000000  0.14941863  0.00000000  0.00000000
0.00000000
##  [7]  0.08104180  0.00000000  0.00000000  0.00000000  0.00000000
0.12312093
## [13]  0.25000767  0.00000000  0.00000000 -0.05144693  0.06557447
0.00000000
## [19]  0.00000000  0.00000000
```

Answer 1.4

Metric/Parameter	Manual Grid Search (Q1.3.i)	cv.glmnet (Q1.4)
Best Lambda	0.1353	0.1353
Best Train MSE	0.536141207349947	0.5665
Best Cross-validation MSE	0.569225257206612	0.6317
Intercept	0.03031505	0.02621250
AtBat	0.0003004736	0 (shrunk)
Hits	0.1743113	0.14941863
HmRun	0.0005633848	0 (shrunk)
Runs	0.0008836302	0 (shrunk)
RBI	0.0006780789	0 (shrunk)

Metric/Parameter	Manual Grid Search (Q1.3.i)	cv.glmnet (Q1.4)
Walks	0.1019672	0.08104180
Years	0.0001376374	0 (shrunk)
CAtBat	0.0007162632	0 (shrunk)
CHits	0.02754872	0 (shrunk)
CHmRun	0.01816204	0 (shrunk)
CRuns	0.1485996	0.12312093
CRBI	0.2292528	0.25000767
CWalks	0.0004509272	0 (shrunk)
LeagueN	0.0002530706	0 (shrunk)
DivisionW	-0.05539532	-0.05144693
PutOuts	0.1145977	0.06557447
Assists	-0.00005813989	0 (shrunk)
Errors	-0.0004435345	0 (shrunk)
NewLeagueN	0.0001997111	0 (shrunk)

Lambda Consistency: Both methods determined the same best lambda value, showcasing consistent and similar performance of our model as compared to glmnet.

MSE Differences: The manual grid search achieved lower MSE values for both training and cv dataset, which suggests better fitting.

Coefficients: The **`cv.glmnet`** approach resulted in many coefficients being reduced to zero, which is typical for LASSO due to its working with respect to feature selection.

2.1

```
library("ISLR2")
library(caret)

## Warning: package 'caret' was built under R version 4.3.3

## Loading required package: ggplot2

## Loading required package: lattice

# Refreshing dataset
d<-Hitters
d <- na.omit(d)

set.seed(34064064)



fn.split <- function(d,p=0.2) {
```

```
  aux      <- 1:length(d[,1])
  id.test <- sort(sample(aux,size=floor(p*length(aux)),
                         replace=FALSE))
  d.test  <- d[id.test,]
  d.train <- d[-id.test,]
  return(list(train=d.train,test=d.test))
}

split_data <- fn.split(d, p=0.3)

train_data <- split_data$train
test_data <- split_data$test

first_row_test_data <- test_data[1, ]
print(first_row_test_data)

##              AtBat Hits HmRun Runs RBI Walks Years CAtBat CHits CHmRun
CRuns
## -Alan Ashby   315   81     7   24  38    39    14   3449   835     69
321
##              CRBI CWalks League Division PutOuts Assists Errors Salary
NewLeague
## -Alan Ashby  414    375      N        W     632      43     10    475
N
```

Answer 2.1

	AtBat	Hits	HmRun	Runs	RBI	Walks	Years	CAtBat	CHits	CHmRun	CRuns
Alan Ashby	315	81	7	24	38	39	14	3449	835	69	321

	CRBI	CWalks	League	Division	PutOuts	Assists	Errors	Salary	NewLeague
Alan Ashby	414	375	N	W	632	43	10	475	N

2.2

```
train_data_standardized <- standardize_features(train_data)
test_data_standardized <- standardize_features(test_data)


train_means <- colMeans(train_data_standardized[,
sapply(train_data_standardized, is.numeric), drop = FALSE])
print("Means of each numeric feature in the training data:")
```

```
## [1] "Means of each numeric feature in the training data:"

print(train_means)

##          AtBat          Hits         HmRun          Runs           RBI
## -1.541325e-16 -1.340144e-16 -6.126856e-17  4.395883e-17  1.911009e-17
##          Walks         Years        CAtBat         CHits        CHmRun
##  1.567627e-16  2.959344e-17 -3.664486e-17  6.164363e-17  1.181487e-17
##          CRuns          CRBI        CWalks       PutOuts       Assists
## -2.945748e-17 -2.751178e-17 -3.844522e-18 -1.426224e-17  2.458619e-17
##         Errors        Salary
## -3.600723e-17 -1.057947e-16

test_means <- colMeans(test_data_standardized[,
sapply(test_data_standardized, is.numeric), drop = FALSE])
print("Means of each numeric feature in the testing data:")

## [1] "Means of each numeric feature in the testing data:"

print(test_means)

##          AtBat          Hits         HmRun          Runs           RBI
##  1.186284e-16  3.217022e-17  3.380487e-17  1.152034e-17 -8.649153e-17
##          Walks         Years        CAtBat         CHits        CHmRun
## -1.599059e-16 -6.507437e-17 -6.983374e-18  7.712847e-17  2.671029e-17
##          CRuns          CRBI        CWalks       PutOuts       Assists
## -9.941300e-18 -5.030698e-17  2.086116e-17  3.053558e-17  2.820038e-17
##         Errors        Salary
##  5.693451e-18 -1.316611e-17
```

Answer 2.2

Feature	Mean in Training Data	Mean in Testing Data
AtBat	-1.541325e-16	1.186284e-16
Hits	-1.340144e-16	3.217022e-17
HmRun	-6.126856e-17	3.380487e-17
Runs	4.395883e-17	1.152034e-17
RBI	1.911009e-17	-8.649153e-17
Walks	1.567627e-16	-1.599059e-16
Years	2.959344e-17	-6.507437e-17
CAtBat	-3.664486e-17	-6.983374e-18
CHits	6.164363e-17	7.712847e-17
CHmRun	1.181487e-17	2.671029e-17
CRuns	-2.945748e-17	-9.941300e-18
CRBI	-2.751178e-17	-5.030698e-17
CWalks	-3.844522e-18	2.086116e-17

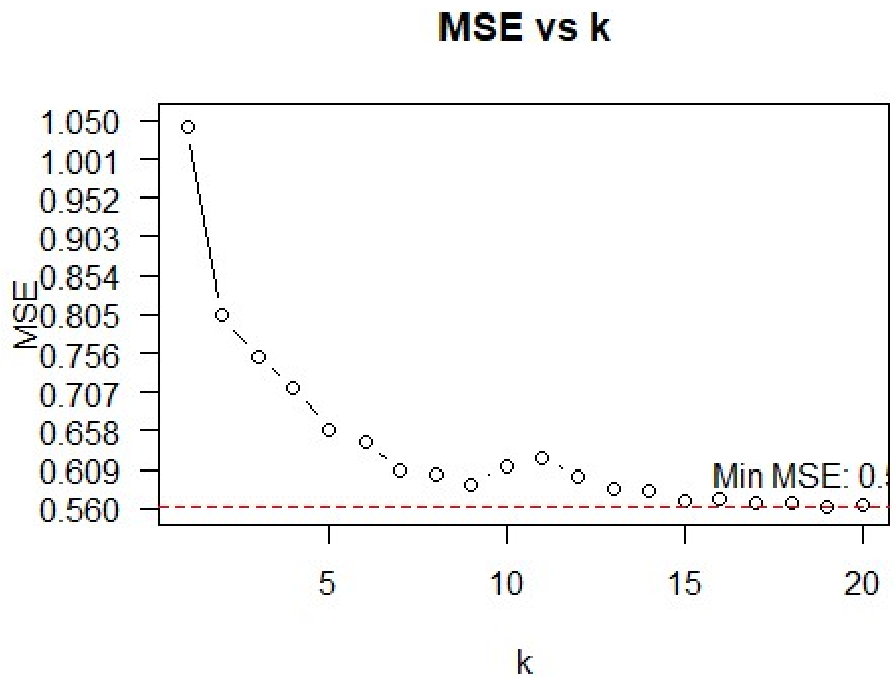
Feature	Mean in Training Data	Mean in Testing Data
PutOuts	-1.426224e-17	3.053558e-17
Assists	2.458619e-17	2.820038e-17
Errors	-3.600723e-17	5.693451e-18
Salary	-1.057947e-16	-1.316611e-17

Mean in training data is quite different than mean in test data.

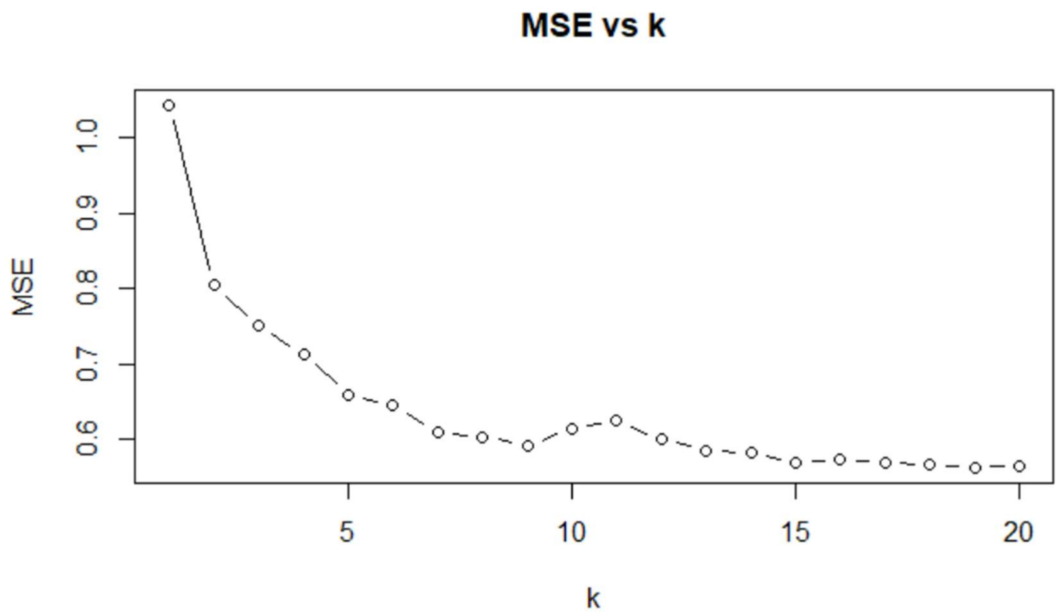
2.3

```
numeric_cols_train <- sapply(train_data_standardized, is.numeric)
numeric_cols_test <- sapply(test_data_standardized, is.numeric)


train_data_numeric <- train_data_standardized[, numeric_cols_train]
test_data_numeric <- test_data_standardized[, numeric_cols_test]

mse_values <- 0

for (k in 1:20) {

  knn_model <- knnreg(x = train_data_numeric[, -
which(names(train_data_numeric) == "Salary")],
              y = train_data_numeric$Salary, k = k)

  predictions <- predict(knn_model, test_data_numeric[, -
which(names(test_data_numeric) == "Salary")])

  mse <- mean((test_data_numeric[, 1] - predictions)^2, na.rm = TRUE)

  mse_values[k] <- mse
}


k_values <- 1:20

mse_min <- min(mse_values)
mse_max <- max(mse_values)

y_lower <- floor(mse_min * 100) / 100
y_upper <- ceiling(mse_max * 100) / 100

plot(k_values, mse_values, type = "b", xlab = "k", ylab = "MSE",
     main = "MSE vs k", ylim = c(y_lower, y_upper), yaxt = "n")

axis(2, at = seq(y_lower, y_upper, by = (y_upper - y_lower) / 10), las = 1)
```

```r
abline(h = mse_min, col = "red", lty = 2)

text(x = which.min(mse_values), y = mse_min, labels = paste("Min MSE:",
round(mse_min, 3)), pos = 3)
```



Answer 2.3

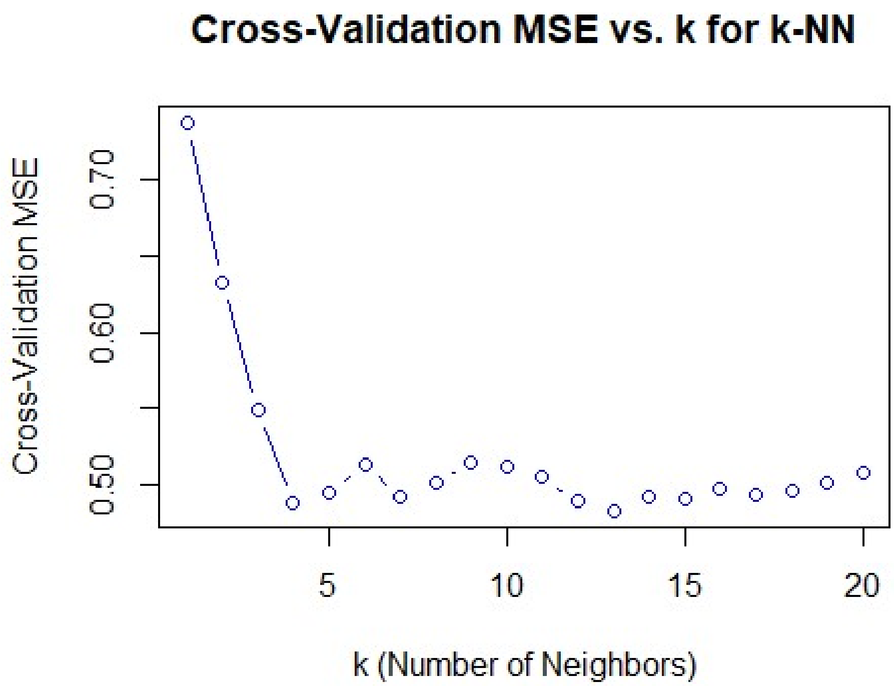


As we can see in the plot, minimum test MSE is 0.56, which is similar to train MSE in 1.4 (0.5665)

2.4

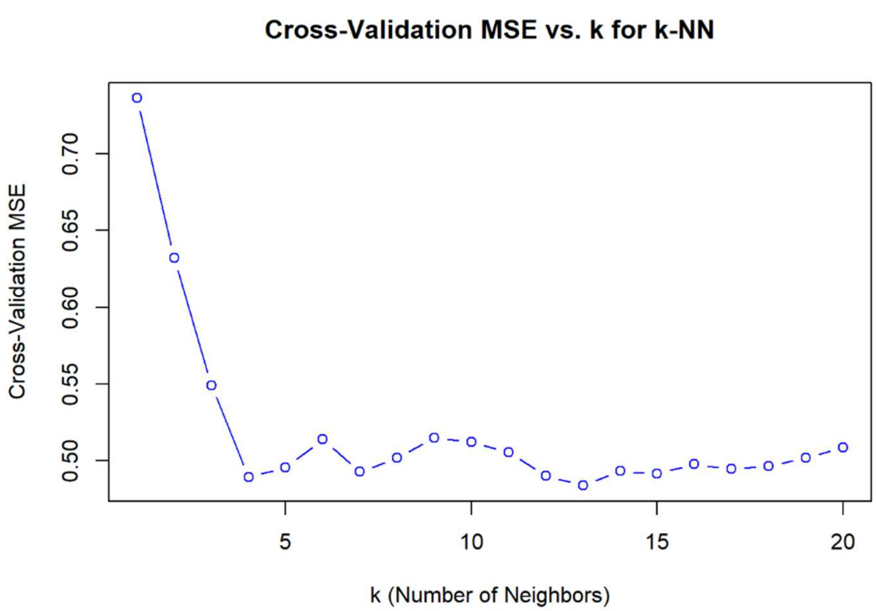
```
set.seed(34064064)
gs.knn <- function(formula,
                   data,
                   k=seq(1,5,1),
                   kfold=5) {
  if (kfold < 2) {
    stop("kfold must be greater than or equal to 2.")
  }

  d  <- na.omit(data)
  d.fold <- fn.split.kfold(data,kfold)

  cv.error <- matrix(NA,length(d[,1]),length(k))
  for (i.fold in 1:kfold) {
    test   <- d.fold[[i.fold]]$test
    train  <- d.fold[[i.fold]]$train
    id     <- d.fold[[i.fold]]$id
    n      <- d.fold[[i.fold]]$n
    mf     <- model.frame(formula,data=test)
    y.test <- model.extract(mf,"response")
```

```r
    for (i.k in k) {
      fit <- knnreg(formula,data=train,k=i.k)
      cv.error[id,i.k] <- (y.test - predict(fit, test))^2
    }
  }
  cv.mse <- apply(cv.error,2,mean)

  res <- cbind(k,cv.mse)
  colnames(res) <- c("k","mse.cv")
  res <- res[order(res[,1]),]

  id <- which.min(res[,2])
  return(list(best=list(k   = res[id,1],
                        mse = res[id,2]),
              cv.mse=res))
}

set.seed(34064064)

results <- gs.knn(Salary ~ ., data = train_data_numeric, k = 1:20, kfold =
10)

best_k <- results$best$k
knn_model_best <- knnreg(Salary ~ ., data = train_data_numeric, k = best_k)

test_predictions <- predict(knn_model_best, newdata = test_data_numeric)

test_mse_2.4 <- mean((test_predictions - test_data_numeric$Salary)^2)


cat("Best k:", best_k, "with Test MSE:", test_mse_2.4, "\n")

## Best k: 13 with Test MSE: 0.6303624

plot(results$cv.mse, type = 'b', col = 'blue', xlab = 'k (Number of
Neighbors)',
     ylab = 'Cross-Validation MSE', main = 'Cross-Validation MSE vs. k for k-
NN')

points(best_k, results$cv.mse[best_k], col = 'red', pch = 19, cex = 1.5)
text(best_k, results$cv.mse[best_k], labels = paste("Best k=", best_k,
"\nMSE=", round(results$cv.mse[best_k], 2)), pos = 4)
```



Answer 2.4

```
Best k is 13 with Test MSE of 0.6304
```



2.5

```
lambda_grid <- exp(seq(-2, 4, length.out=30))

lasso_results <- gs.lasso(Salary ~ ., data = train_data_standardized, lambda
= lambda_grid, kfold = 10, std = TRUE)

best_lambda <- lasso_results$best$lambda
best_train_mse <- lasso_results$best$mse
cv_mse <- min(lasso_results$all$mse[, "mse.cv"])
test_model <- glmnet(model.matrix(Salary ~ ., test_data_standardized)[,-1],
test_data_standardized$Salary, alpha = 1, lambda = best_lambda)
predictions <- predict(test_model, model.matrix(Salary ~ .,
test_data_standardized)[,-1])
test_mse_2.5 <- mean((test_data_standardized$Salary - predictions)^2)

results_table <- data.frame(
  Lambda = best_lambda,
  Train_MSE = best_train_mse,
  CV_MSE = cv_mse,
  Test_MSE = test_mse_2.5
)
print(results_table)

##           Lambda Train_MSE    CV_MSE  Test_MSE
## lambda 0.1664428 0.4530632 0.5063615 0.6399751
```

Answer 2.5

Lambda	Train MSE	CV MSE	Test MSE
0.1664	0.4531	0.5064	0.6400

2.6

```
lambda_grid <- exp(seq(-2, 4, length.out=30))
library(glmnet)

x_train <- model.matrix(Salary ~ ., train_data_standardized)[, -1]  # Remove
intercept
y_train <- train_data_standardized$Salary


cv_fit <- cv.glmnet(x_train, y_train, alpha = 1, lambda = lambda_grid, nfolds
= 10)

best_lambda <- cv_fit$lambda.min
best_cv_mse <- min(cv_fit$cvm)
model <- glmnet(x_train, y_train, alpha = 1, lambda = best_lambda)
```

```r
x_test <- model.matrix(Salary ~ ., test_data_standardized)[, -1]
predictions <- predict(model, s = best_lambda, newx = x_test)
test_mse_2.6 <- mean((test_data_standardized$Salary - predictions)^2)

train_predictions <- predict(model, s = best_lambda, newx = x_train)
train_mse_2.6 <- mean((y_train - train_predictions)^2)

results_table <- data.frame(
  Lambda = best_lambda,
  Train_MSE = train_mse_2.6,
  CV_MSE = best_cv_mse,
  Test_MSE = test_mse_2.6
)


print(results_table)

##      Lambda Train_MSE    CV_MSE  Test_MSE
## 1 0.1353353 0.4780857 0.5544935 0.7403519
```

Answer 2.6

Lambda	Train MSE	CV MSE	Test MSE
0.1353	0.4781	0.5584	0.7404

Answer 2.7

Model 2.4 - Built using k-NN, with best k being 13, gives a test MSE of 0.6303624, which is the **best** amongst all 3 models.

Model 2.5 - Built using our function, gs.lasso, gives a test MSE of 0.6399751, which is pretty close to 2.4. We can see that it overfits, as it has a much lower train MSE of 0.4531.

Model 2.6 - Built using glmnet, performs much worse than other 2 models, with a test MSE of 0.7404. Degree of overfitting is quite high as well, with train MSE at 0.4781.

To conclude, model 2.4 would be the best predict Salary.

2.8

```r
rmse_knn <- sqrt(test_mse_2.4)
rmse_lasso_gs <- sqrt(test_mse_2.5)
rmse_lasso_cv <- sqrt(test_mse_2.6)


mean_salary_test <- mean(test_data_standardized$Salary)
```

```
sd_salary_test <- sd(test_data_standardized$Salary)

cat("RMSE for k-NN Model:", rmse_knn, "\n")

## RMSE for k-NN Model: 0.7939536

cat("RMSE for LASSO (gs.lasso):", rmse_lasso_gs, "\n")

## RMSE for LASSO (gs.lasso): 0.7999844

cat("RMSE for LASSO (cv.glmnet):", rmse_lasso_cv, "\n")

## RMSE for LASSO (cv.glmnet): 0.860437

cat("Mean Salary on Test Data:", mean_salary_test, "\n")

## Mean Salary on Test Data: -1.317028e-17

cat("Standard Deviation of Salary on Test Data:", sd_salary_test, "\n")

## Standard Deviation of Salary on Test Data: 1
```

Answer 2.8

Metric	Value
RMSE for k-NN Model	0.7939536
RMSE for LASSO (gs.lasso)	0.7999844
RMSE for LASSO (cv.glmnet)	0.860437
Mean Salary on Test Data	-1.317028e-17
Standard Deviation of Salary on Test Data	1

In my opinion, k-NN and gs.lasso model are pretty good predict Salary.