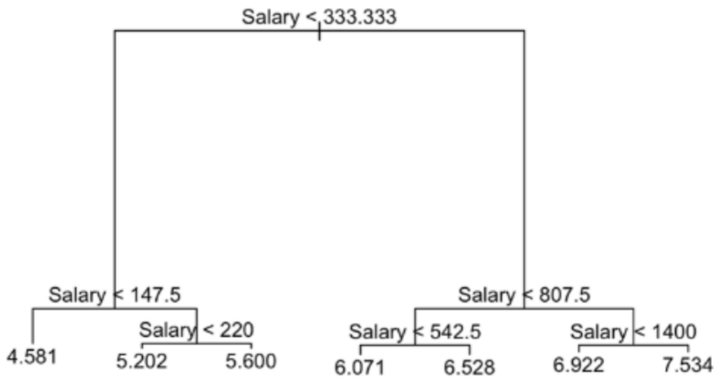
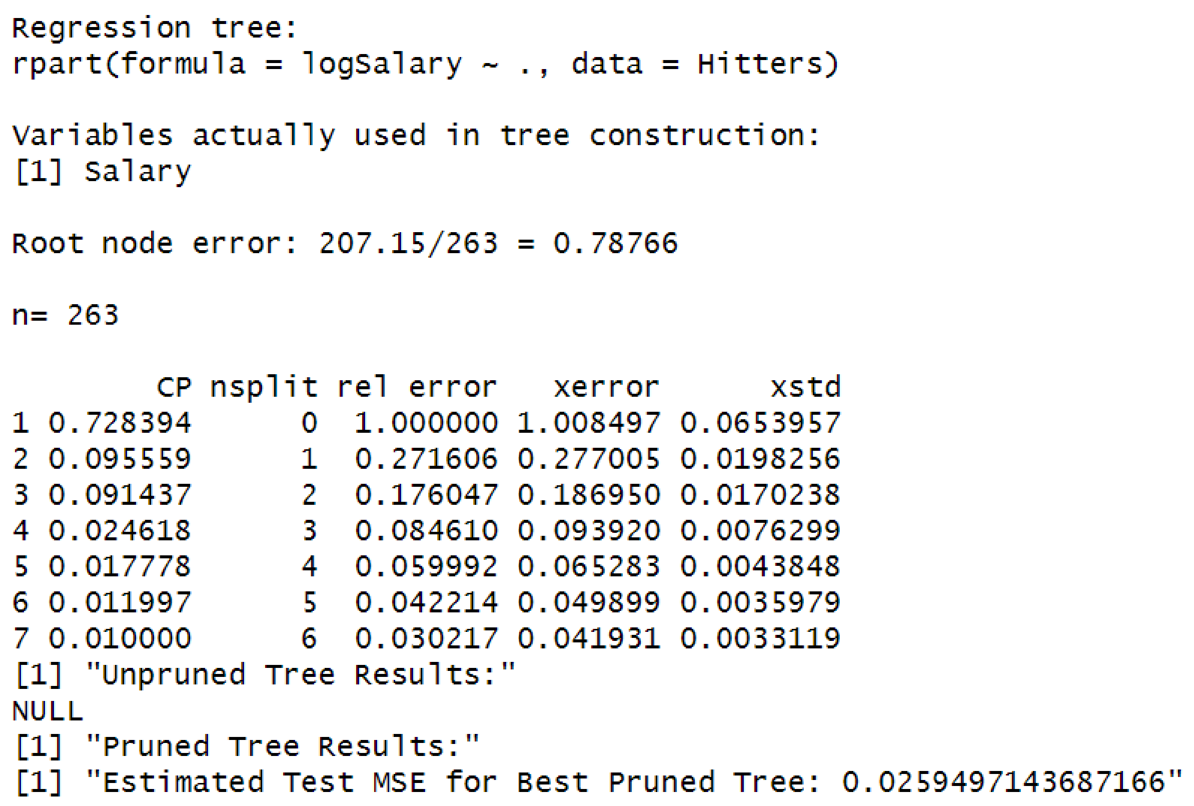
STAT 4360 (Introduction to Statistical Learning, Spring 2023)
Mini Project 6
Name: Sayema Rahman

1. (a) The regression tree shows the 7 nodes in the tree. The first and main split is 333.333. The second two splits are 147.5 and 807.5. The next split that occurs at 147.5 is between 4.581 and 220. The next split is at 220 and the two values are 5.202 and 5.600. Going back to the first node on the tree, the right side split, and 807.5 splits again to get 542.5 and 1400. The 542.5 node splits into two nodes labeled 6.071 and 6.528. The 1400 node splits into two two nodes labeled 6.922 and 7.534. The estimated test MSE is 0.03770513. Due to the value of the MSE being close to 0, the squared difference between the predicted logarithm of salaries and the actual log of the salaries is quite small. This all leads to the there being a good prediction accuracy.

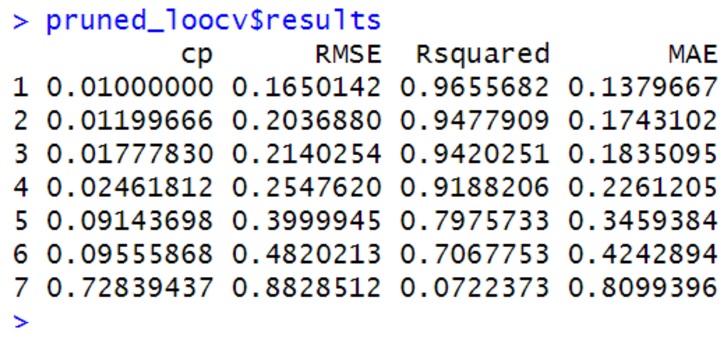


```
1) root 212 167.7000 5.927
  2) Salary < 333.333 90  19.7400 5.046
    4) Salary < 147.5 40   1.9490 4.586 *
    5) Salary > 147.5 50   2.5930 5.413
     10) Salary < 227.5 27   0.3430 5.229 *
     11) Salary > 227.5 23   0.2588 5.629 *
  3) Salary > 333.333 122  26.4200 6.578
    6) Salary < 685 54   2.1210 6.163 *
    7) Salary > 685 68   7.6820 6.906
     14) Salary < 1137.5 48   0.7414 6.718 *
     15) Salary > 1137.5 20   1.1390 7.359 *
[1] 0.03770513
```

(b) The unpruned tree has a greater RMSE value than the pruned tree. The pruned tree is significantly better than the unpruned tree. The most important predictors seemed to be salary.



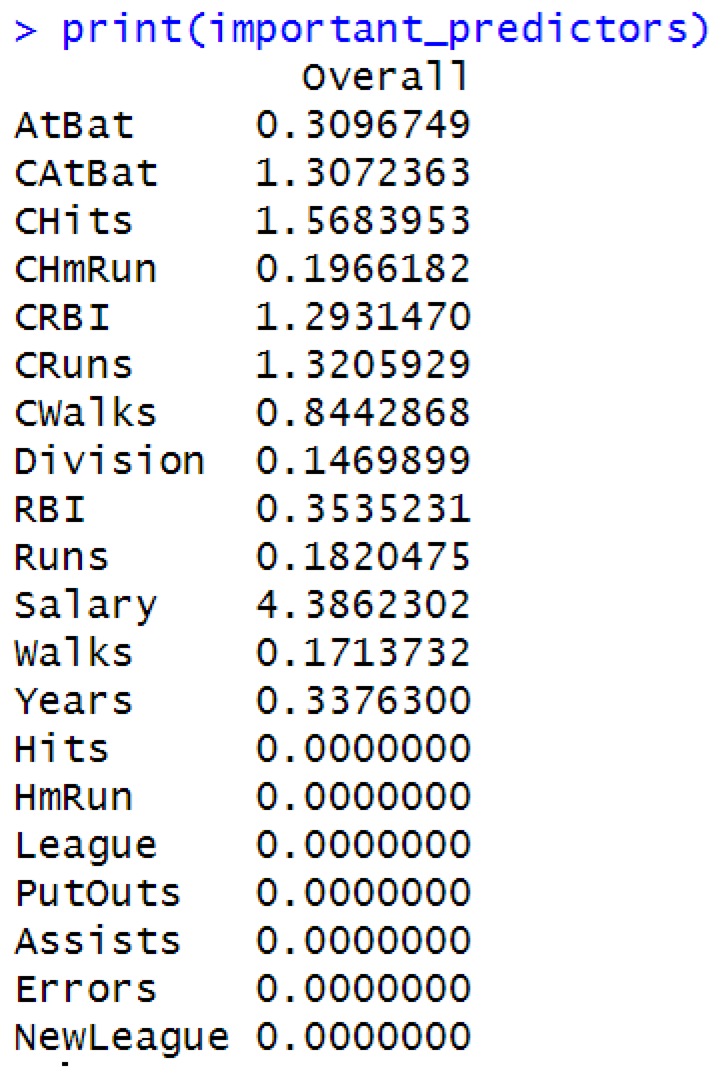
```
Regression tree:
rpart(formula = logSalary ~ ., data = Hitters)

Variables actually used in tree construction:
[1] Salary

Root node error: 207.15/263 = 0.78766

n= 263

        CP nsplit rel error   xerror      xstd
1 0.728394      0  1.000000 1.008497 0.0653957
2 0.095559      1  0.271606 0.277005 0.0198256
3 0.091437      2  0.176047 0.186950 0.0170238
4 0.024618      3  0.084610 0.093920 0.0076299
5 0.017778      4  0.059992 0.065283 0.0043848
6 0.011997      5  0.042214 0.049899 0.0035979
7 0.010000      6  0.030217 0.041931 0.0033119
[1] "Unpruned Tree Results:"
NULL
[1] "Pruned Tree Results:"
[1] "Estimated Test MSE for Best Pruned Tree: 0.0259497143687166"
```



```
> pruned_loocv$results
          cp      RMSE  Rsquared       MAE
1 0.01000000 0.1650142 0.9655682 0.1379667
2 0.01199666 0.2036880 0.9477909 0.1743102
3 0.01777830 0.2140254 0.9420251 0.1835095
4 0.02461812 0.2547620 0.9188206 0.2261205
5 0.09143698 0.3999945 0.7975733 0.3459384
6 0.09555868 0.4820213 0.7067753 0.4242894
7 0.72839437 0.8828512 0.0722373 0.8099396
>
```

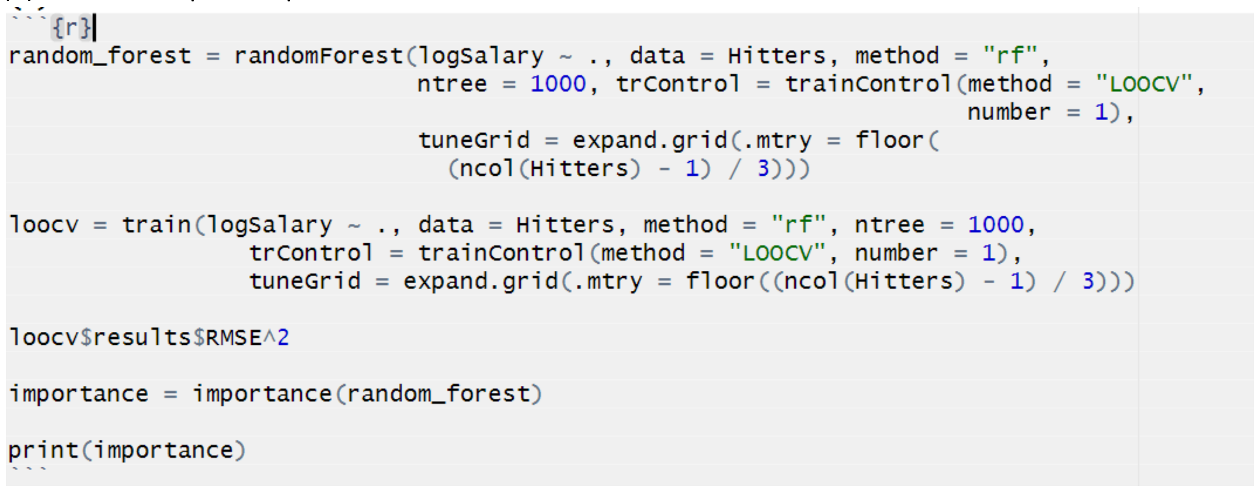
(c) The test MSE is 9.50350999076997e-05 for Random Forest which is fairly low. This indicates that the model is good for predicting the salary based on these predictors. Cbat, Walks, and Chits are all important predictors.

```
model <- randomForest(logSalary ~ ., data = Hitters, mtry = ncol(Hitters)-1, ntree = 1000)

predictions = predict(model, newdata = Hitters)
bagging = mean((Hitters$logSalary - predictions)^2)
print(paste("Test MSE for Model:", bagging))

importance = importance(model)
print(importance)

```
```

```
[1] "Test MSE for Model: 9.50350999076997e-05"
          IncNodePurity
AtBat      1.172444e-02
Hits       8.662642e-03
HmRun      6.422724e-03
Runs       1.761487e-02
RBI        9.344264e-03
Walks      1.997124e-02
Years      6.519051e-03
CAtBat     1.679037e-02
CHits      1.714956e-02
CHmRun     1.301456e-02
CRuns      1.514062e-02
CRBI       1.065231e-02
CWalks     1.334762e-02
League     2.081534e-03
Division   4.698867e-03
PutOuts    1.351578e-02
Assists    1.145240e-02
Errors     1.778280e-02
Salary     2.055031e+02
NewLeague  1.776642e-03
```
```



```
> print(important_predictors)
              Overall
AtBat       0.3096749
CAtBat      1.3072363
CHits       1.5683953
CHmRun      0.1966182
CRBI        1.2931470
CRuns       1.3205929
CWalks      0.8442868
Division    0.1469899
RBI         0.3535231
Runs        0.1820475
Salary      4.3862302
Walks       0.1713732
Years       0.3376300
Hits        0.0000000
HmRun       0.0000000
League      0.0000000
PutOuts     0.0000000
Assists     0.0000000
Errors      0.0000000
NewLeague   0.0000000
```

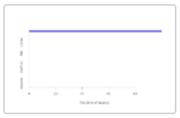
(d) The most important predictors are CAtBat and hits.

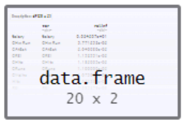


~~~
```{r}
random_forest = randomForest(logSalary ~ ., data = Hitters, method = "rf",
                             ntree = 1000, trControl = trainControl(method = "LOOCV",
                                                                    number = 1),
                             tuneGrid = expand.grid(.mtry = floor(
                               (ncol(Hitters) - 1) / 3)))

loocv = train(logSalary ~ ., data = Hitters, method = "rf", ntree = 1000,
              trControl = trainControl(method = "LOOCV", number = 1),
              tuneGrid = expand.grid(.mtry = floor((ncol(Hitters) - 1) / 3)))

loocv$results$RMSE^2

importance = importance(random_forest)

print(importance)
```
~~~

(e) The most important predictors are Salary and CHmRun.

```
oosting_model = gbm(logSalary ~., data = Hitters, distribution = "gaussian",
                    n.trees = 1000, interaction.depth = 1, shrinkage = 0.01,
                    cv.folds = nrow(Hitters), n.cores = NULL)
SE = min(boosting_model$cv.error)
rint(MSE)

ummary = summary(boosting_model)
rint(summary)
``
```

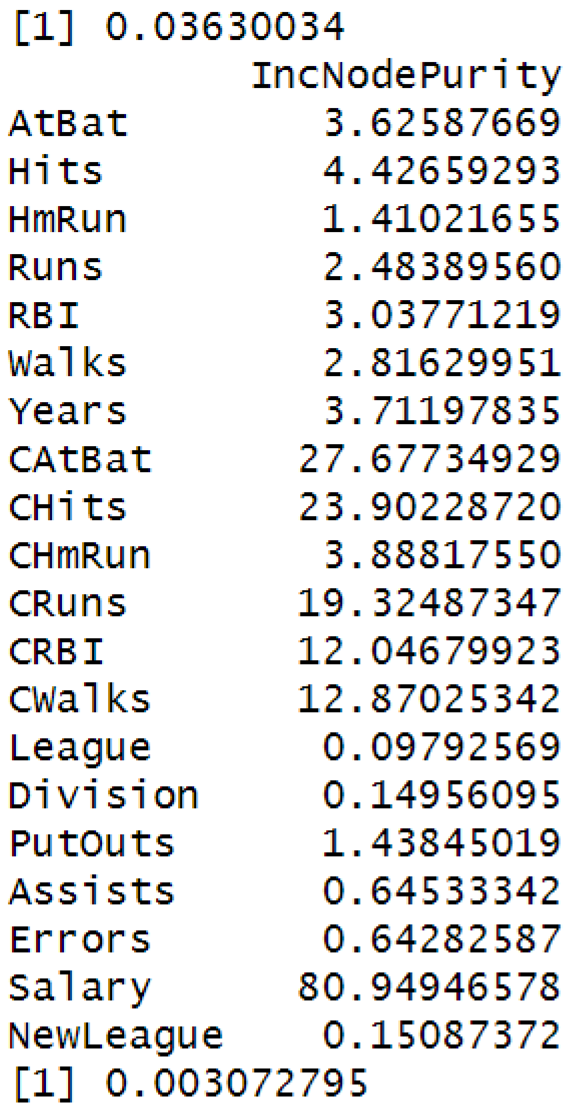




Description: df [20 x 2]

	var <chr>	rel.inf <dbl>
Salary	Salary	9.984267e+01
CHmRun	CHmRun	5.771258e-02
CAtBat	CAtBat	2.040630e-02
CRBI	CRBI	1.158551e-02
CHits	CHits	1.132665e-02
CRuns	CRuns	1.109609e-02
CWalks	CWalks	1.075639e-02
Hits	Hits	8.078839e-03
RBI	RBI	7.635873e-03
AtBat	AtBat	5.156068e-03

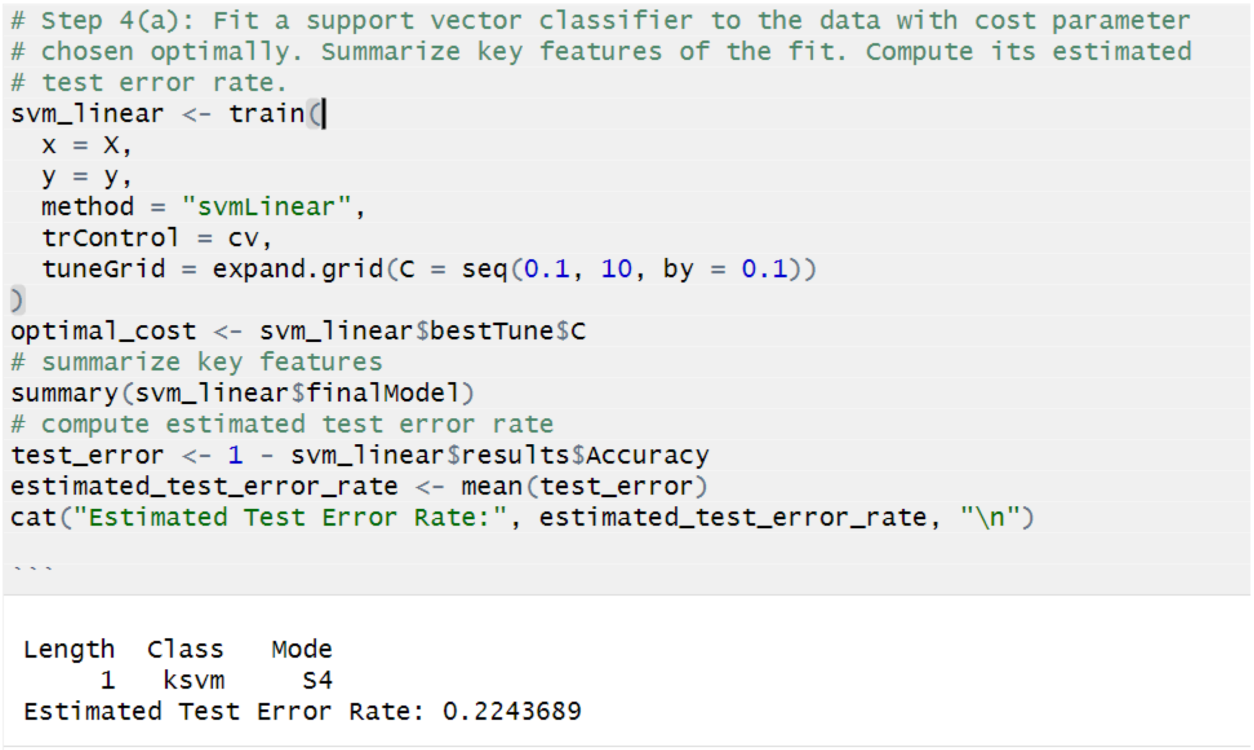
1-10 of 20 rows

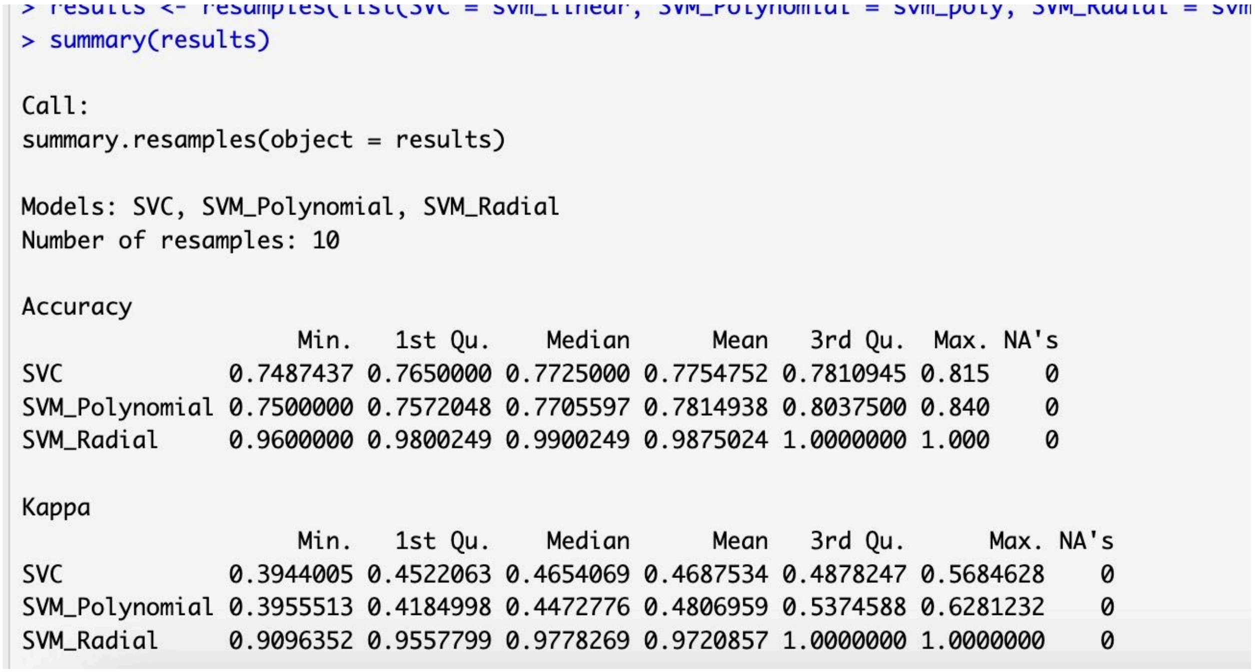


```
[1] 0.03630034
          IncNodePurity
AtBat        3.62587669
Hits         4.42659293
HmRun        1.41021655
Runs         2.48389560
RBI          3.03771219
Walks        2.81629951
Years        3.71197835
CAtBat      27.67734929
CHits       23.90228720
CHmRun       3.88817550
CRuns       19.32487347
CRBI        12.04679923
CWalks      12.87025342
League       0.09792569
Division     0.14956095
PutOuts      1.43845019
Assists      0.64533342
Errors       0.64282587
Salary      80.94946578
NewLeague    0.15087372
[1] 0.003072795
```

(f) I would recommend the boosting approach as the best approach, as it is better when it comes to giving us an analysis of the data.

2. (a) The estimated test error rate is approximately 0.22. The key features of the fit are Length, Class, and Mode.



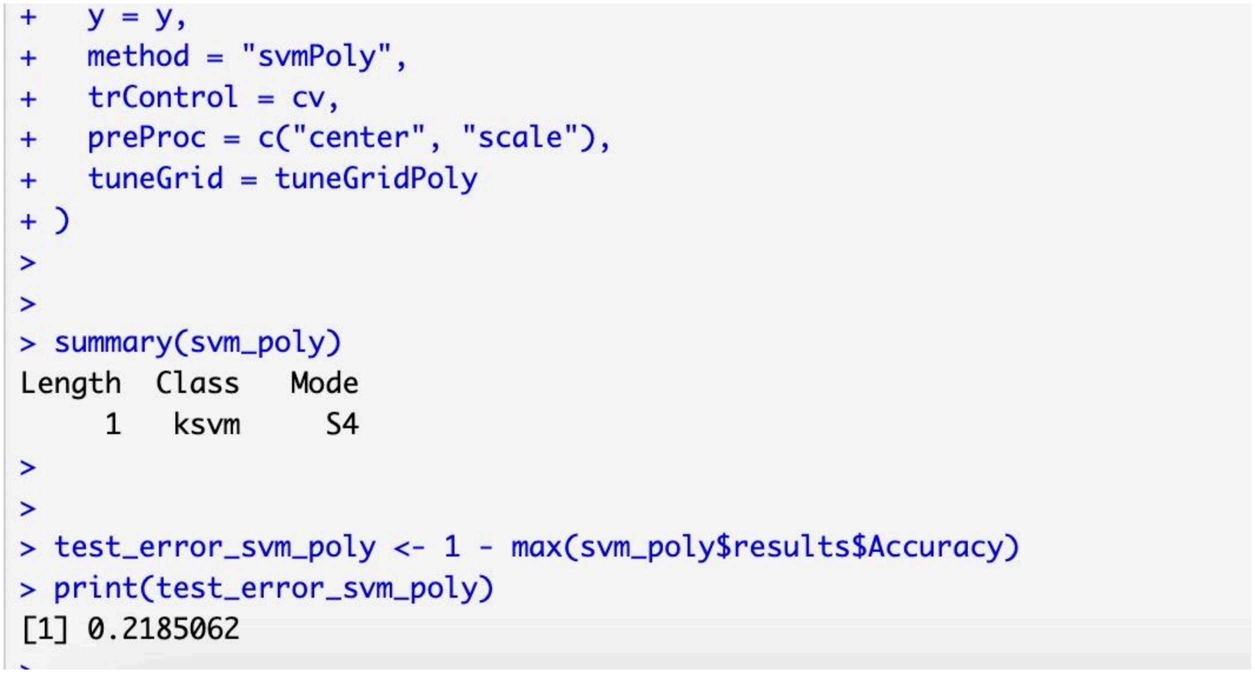
```
# Step 4(a): Fit a support vector classifier to the data with cost parameter
# chosen optimally. Summarize key features of the fit. Compute its estimated
# test error rate.
svm_linear <- train(
  x = X,
  y = y,
  method = "svmLinear",
  trControl = cv,
  tuneGrid = expand.grid(C = seq(0.1, 10, by = 0.1))
)
optimal_cost <- svm_linear$bestTune$C
# summarize key features
summary(svm_linear$finalModel)
# compute estimated test error rate
test_error <- 1 - svm_linear$results$Accuracy
estimated_test_error_rate <- mean(test_error)
cat("Estimated Test Error Rate:", estimated_test_error_rate, "\n")

```
```

```
Length  Class   Mode
     1   ksvm     S4
Estimated Test Error Rate: 0.2243689
```

(b) The key features of the fit are Length, Class, and Mode. The estimated test error rate is approximately 0.21.



```
> results <- resamples(list(SVC = svm_linear, SVM_Polynomial = svm_poly, SVM_Radial = svm
> summary(results)

Call:
summary.resamples(object = results)

Models: SVC, SVM_Polynomial, SVM_Radial
Number of resamples: 10

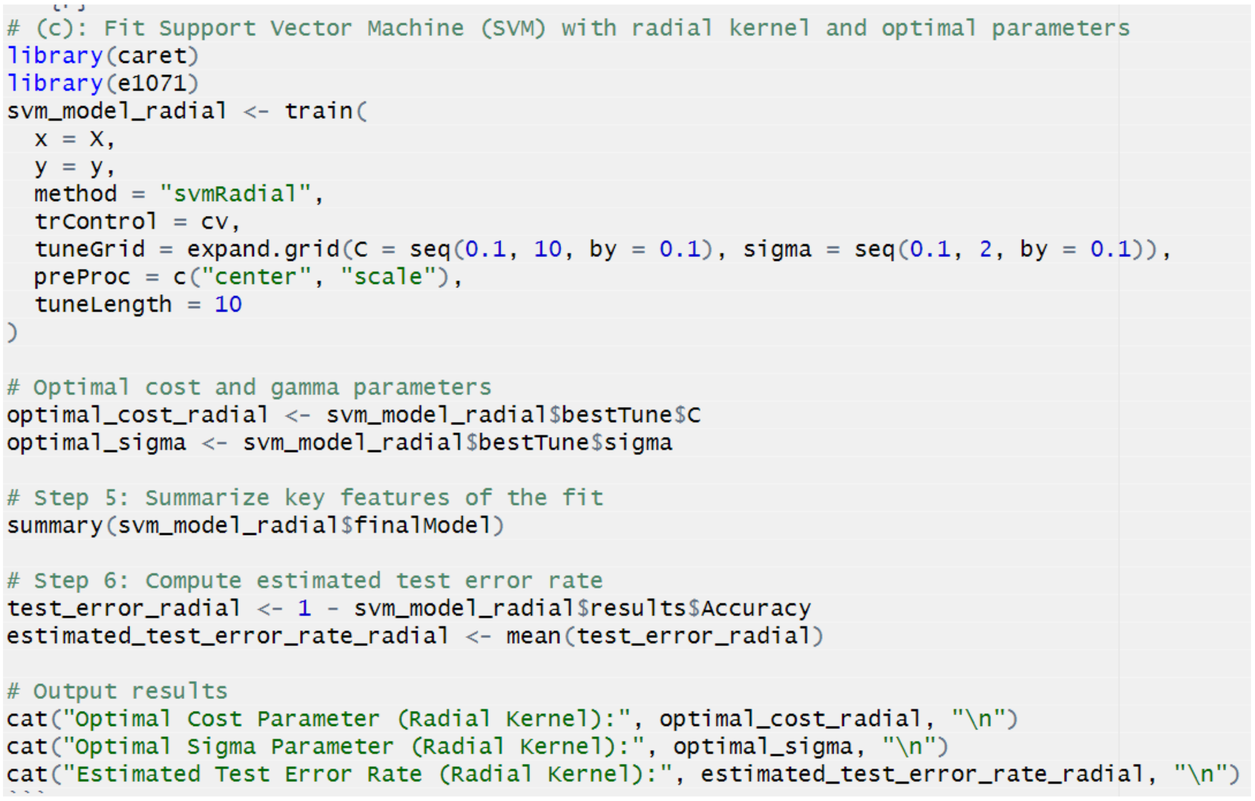
Accuracy
                    Min.   1st Qu.    Median      Mean   3rd Qu.  Max. NA's
SVC            0.7487437 0.7650000 0.7725000 0.7754752 0.7810945 0.815    0
SVM_Polynomial 0.7500000 0.7572048 0.7705597 0.7814938 0.8037500 0.840    0
SVM_Radial     0.9600000 0.9800249 0.9900249 0.9875024 1.0000000 1.000    0

Kappa
                    Min.   1st Qu.    Median      Mean   3rd Qu.      Max. NA's
SVC            0.3944005 0.4522063 0.4654069 0.4687534 0.4878247 0.5684628    0
SVM_Polynomial 0.3955513 0.4184998 0.4472776 0.4806959 0.5374588 0.6281232    0
SVM_Radial     0.9096352 0.9557799 0.9778269 0.9720857 1.0000000 1.0000000    0
```
```

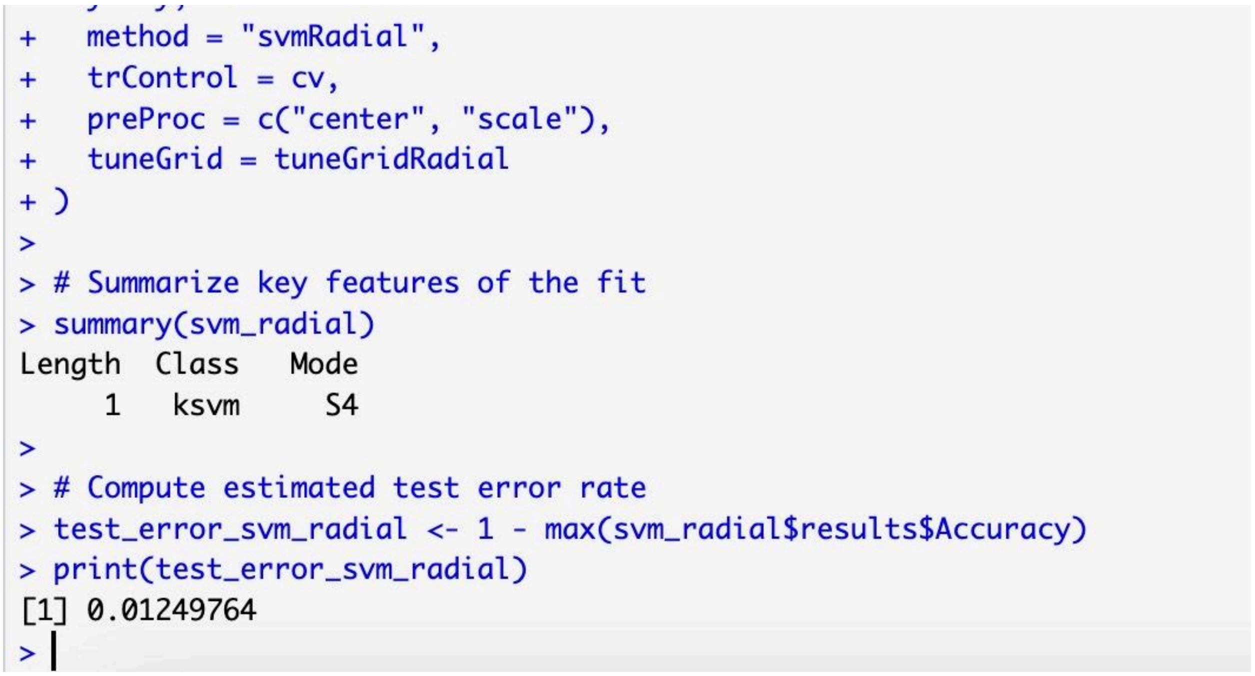


```
+   y = y,
+   method = "svmPoly",
+   trControl = cv,
+   preProc = c("center", "scale"),
+   tuneGrid = tuneGridPoly
+ )
>
>
> summary(svm_poly)
Length  Class   Mode
     1   ksvm     S4
>
>
> test_error_svm_poly <- 1 - max(svm_poly$results$Accuracy)
> print(test_error_svm_poly)
[1] 0.2185062
```

(c) The estimated test error rate is approximately 0.01249764.



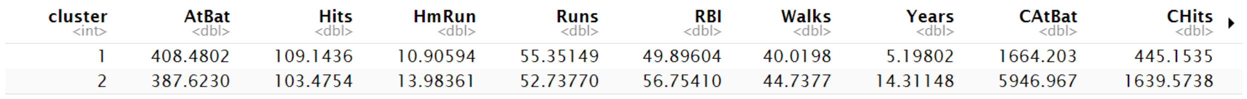
```r
# (c): Fit Support Vector Machine (SVM) with radial kernel and optimal parameters
library(caret)
library(e1071)
svm_model_radial <- train(
  x = X,
  y = y,
  method = "svmRadial",
  trControl = cv,
  tuneGrid = expand.grid(C = seq(0.1, 10, by = 0.1), sigma = seq(0.1, 2, by = 0.1)),
  preProc = c("center", "scale"),
  tuneLength = 10
)

# Optimal cost and gamma parameters
optimal_cost_radial <- svm_model_radial$bestTune$C
optimal_sigma <- svm_model_radial$bestTune$sigma

# Step 5: Summarize key features of the fit
summary(svm_model_radial$finalModel)

# Step 6: Compute estimated test error rate
test_error_radial <- 1 - svm_model_radial$results$Accuracy
estimated_test_error_rate_radial <- mean(test_error_radial)

# Output results
cat("Optimal Cost Parameter (Radial Kernel):", optimal_cost_radial, "\n")
cat("Optimal Sigma Parameter (Radial Kernel):", optimal_sigma, "\n")
cat("Estimated Test Error Rate (Radial Kernel):", estimated_test_error_rate_radial, "\n")
```



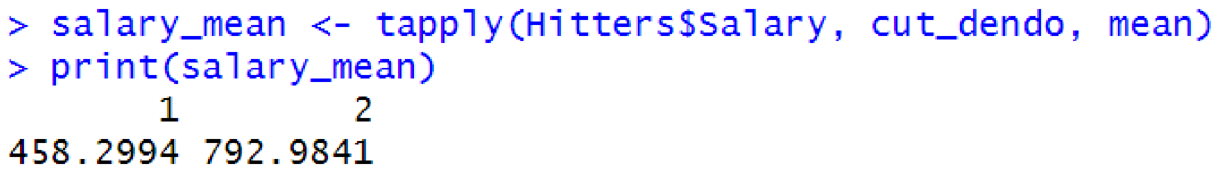
```
+   method = "svmRadial",
+   trControl = cv,
+   preProc = c("center", "scale"),
+   tuneGrid = tuneGridRadial
+ )
>
> # Summarize key features of the fit
> summary(svm_radial)
Length  Class   Mode 
     1   ksvm     S4 
>
> # Compute estimated test error rate
> test_error_svm_radial <- 1 - max(svm_radial$results$Accuracy)
> print(test_error_svm_radial)
[1] 0.01249764
> 
```

(d) I would recommend Radial Kernel as the recommended method as it gives the best test error value.

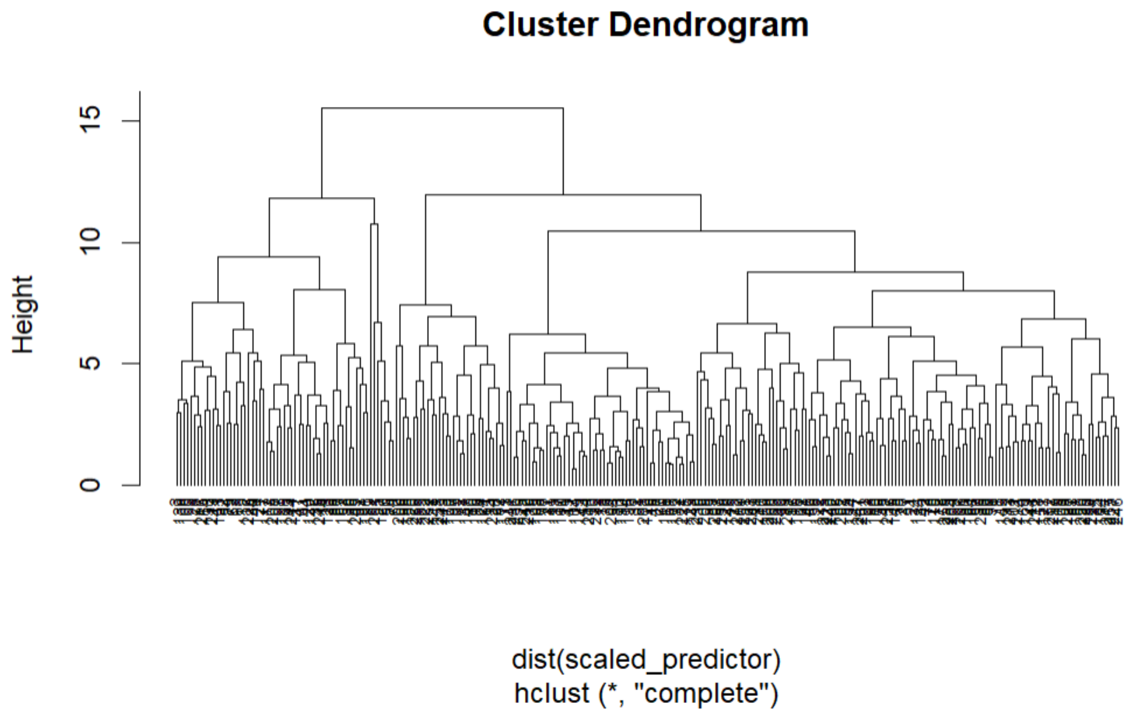
3. (a) Standardizing the variables before clustering is a good idea.
   (b) I would use a metric-based system to cluster the players.
   (C) The mean salary of the two players was 458.3 and 792.98. This suggests that cluster 2 has more high-performance players than cluster 1.



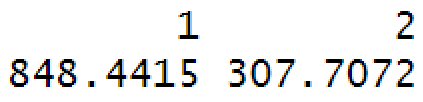
cluster	AtBat	Hits	HmRun	Runs	RBI	Walks	Years	CAtBat	CHits
<int>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>
1	408.4802	109.1436	10.90594	55.35149	49.89604	40.0198	5.19802	1664.203	445.1535
2	387.6230	103.4754	13.98361	52.73770	56.75410	44.7377	14.31148	5946.967	1639.5738



```
> salary_mean <- tapply(Hitters$Salary, cut_dendo, mean)
> print(salary_mean)
       1        2 
458.2994 792.9841 
```



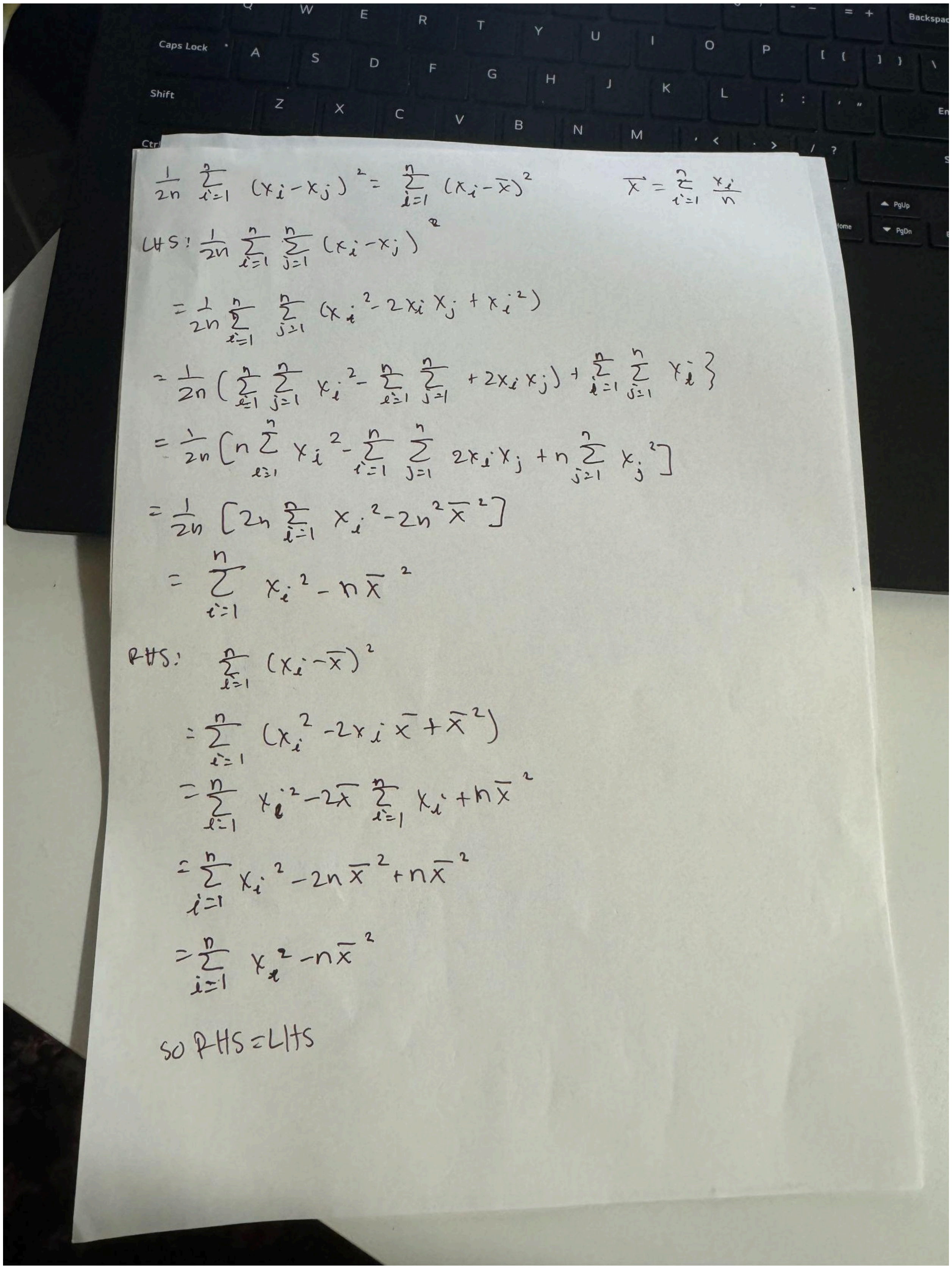
(d) The two clusters show the mean salaries of the players. In Cluster 1 we have more high-performance players while Cluster 2 would suggest that the players are more low-performing players.



```
       1        2 
848.4415 307.7072 
```

(e) The k-means algorithm gives the more sensible result.

4)



$$\frac{1}{2n}\sum_{i=1}^{n}(x_i-x_j)^2 = \sum_{i=1}^{n}(x_i-\bar{x})^2 \qquad \bar{x}=\sum_{i=1}^{n}\frac{x_i}{n}$$

LHS: $\frac{1}{2n}\sum_{i=1}^{n}\sum_{j=1}^{n}(x_i-x_j)^2$

$$=\frac{1}{2n}\sum_{i=1}^{n}\sum_{j=1}^{n}(x_i^2-2x_ix_j+x_i^2)$$

$$=\frac{1}{2n}\left(\sum_{i=1}^{n}\sum_{j=1}^{n}x_i^2-\sum_{i=1}^{n}\sum_{j=1}^{n}+2x_ix_j\right)+\sum_{i=1}^{n}\sum_{j=1}^{n}x_i\}$$

$$=\frac{1}{2n}\left[n\sum_{i=1}^{n}x_i^2-\sum_{i=1}^{n}\sum_{j=1}^{n}2x_ix_j+n\sum_{j=1}^{n}x_j^2\right]$$

$$=\frac{1}{2n}\left[2n\sum_{i=1}^{n}x_i^2-2n^2\bar{x}^2\right]$$

$$=\sum_{i=1}^{n}x_i^2-n\bar{x}^2$$

RHS: $\sum_{i=1}^{n}(x_i-\bar{x})^2$

$$=\sum_{i=1}^{n}(x_i^2-2x_i\bar{x}+\bar{x}^2)$$

$$=\sum_{i=1}^{n}x_i^2-2\bar{x}\sum_{i=1}^{n}x_i+n\bar{x}^2$$

$$=\sum_{i=1}^{n}x_i^2-2n\bar{x}^2+n\bar{x}^2$$

$$=\sum_{i=1}^{n}x_i^2-n\bar{x}^2$$

so RHS = LHS

Python Code (or R Code)

~~~
1a)
```{r}
library(ISLR)
library(tree)
library(MASS)
library(rpart)
library(caret)
library(randomForest)
library(gbm)
library(e1071)
library(ISLR2)
# load the data
data("Hitters")
# clean the data set
Hitters <- na.omit(Hitters)
Hitters$logSalary <- log(Hitters$Salary)
tree = rpart(logSalary ~ ., data=Hitters)
plot(tree)
text(tree, pretty = 0)

set.seed(123) # For reproducibility
index <- createDataPartition(Hitters$logSalary, p = 0.8, list =
FALSE)
training_data <- Hitters[index, ]
test_data <- Hitters[-index, ]

# Fit tree
tree_fit <- tree(logSalary ~., data = training_data)
print(tree_fit)

# Predict on testing data
predictions <- predict(tree_fit, test_data)

# Calculating the MSE
MSE_Test <- mean((test_data$logSalary - predictions)^2)
print(MSE_Test)

```
1b)
```{r}
printcp(tree)

prunedTree = prune(tree, cp =
tree$cptable[which.min(tree$cptable[,"xerror"]), "CP"])
prunedLoocv <- train(logSalary ~ ., data = Hitters, method = "rpart",
                      trControl = trainControl(method = "LOOCV"),
~~~

~~~
           tuneGrid = expand.grid(.cp =
tree$cptable[,"CP"]))
prunedLoocv$results


# Compare results
print("Unpruned Tree Results:")
print(tree$results)
print("Pruned Tree Results:")
print(prunedLoocv$results)

# Report estimated test MSE for the best pruned tree
best_cp <-
prunedLoocv$results$cp[which.min(prunedLoocv$results$RMSE)]
best_pruned_tree <- prune(tree, cp = best_cp)
test_predictions <- predict(best_pruned_tree, newdata = test_data)
test_mse <- mean((test_data$logSalary - test_predictions)^2)
print(paste("Estimated Test MSE for Best Pruned Tree:", test_mse))

# Identify important predictors
important_predictors <- varImp(best_pruned_tree)
print(important_predictors)
```
(c)
```{r}
model <- randomForest(logSalary ~ ., data = Hitters, mtry =
ncol(Hitters)-1, ntree = 1000)

predictions = predict(model, newdata = Hitters)
bagging = mean((Hitters$logSalary - predictions)^2)
print(paste("Test MSE for Model:", bagging))

importance = importance(model)
print(importance)

```
(d)
```{r}
random_forest = randomForest(logSalary ~ ., data = Hitters, method =
"rf",
              ntree = 1000, trControl =
trainControl(method = "LOOCV",

number = 1),
              tuneGrid = expand.grid(.mtry = floor(
                (ncol(Hitters) - 1) / 3)))

loocv = train(logSalary ~ ., data = Hitters, method = "rf", ntree =
1000,
        trControl = trainControl(method = "LOOCV", number =
1),
~~~

~~~
                 tuneGrid = expand.grid(.mtry = floor((ncol(Hitters)
- 1) / 3)))

loocv$results$RMSE^2

importance = importance(random_forest)

print(importance)
```
(e)
```{r}
boosting_model = gbm(logSalary ~., data = Hitters, distribution =
"gaussian",
                     n.trees = 1000, interaction.depth = 1, shrinkage
= 0.01,
                     cv.folds = nrow(Hitters), n.cores = NULL)
MSE = min(boosting_model$cv.error)
print(MSE)

summary = summary(boosting_model)
print(summary)
```



Question 2
```{r}
library(e1071)
library(caret)
library(ggplot2)
library(MASS)
diabetes <- read.table("C:/Users/sayem/Downloads/diabetes.csv",
sep=",", header = T)
# renaming columns to get rid of .. at the end of each variable
names(diabetes) <- c("Pregnancies", "Glucose", "BloodPressure",
"SkinThickness",
                     "Insulin", "BMI", "DiabetesPedigreeFunction",
"Age", "Outcome")
head(diabetes)
X <- diabetes[, -9] # Features
y <- diabetes$Outcome
y <- factor(y, levels = c(0, 1))

# 10-fold cross validation
set.seed(123) # For reproducibility
cv <- trainControl(method = "cv", number = 10)
```
```{r}
#DONE
# Step 4(a): Fit a support vector classifier to the data with cost
parameter
# chosen optimally. Summarize key features of the fit. Compute its
estimated
~~~

~~~
# test error rate.
svm_linear <- train(
  x = X,
  y = y,
  method = "svmLinear",
  trControl = cv,
  tuneGrid = expand.grid(C = seq(0.1, 10, by = 0.1))
)
optimal_cost <- svm_linear$bestTune$C
# summarize key features
summary(svm_linear$finalModel)
# compute estimated test error rate
test_error <- 1 - svm_linear$results$Accuracy
estimated_test_error_rate <- mean(test_error)
cat("Estimated Test Error Rate:", estimated_test_error_rate, "\n")

```
```{r}
library(caret)
library(e1071)

# (b): Fit Support Vector Machine (SVM) with polynomial kernel and
optimal cost parameter

svm_model_poly <- train(
  x = X,
  y = y,
  method = "svmPoly",
  trControl = cv,
  tuneGrid = expand.grid(degree = 1:3, scale = c(0.1, 1, 10), C =
seq(0.1, 10, by = 0.1))
)

# Optimal cost parameter
optimal_cost_poly <- svm_model_poly$bestTune$C

# Summarize key features of the fit
print(svm_model_poly)

# Compute estimated test error rate
test_error_poly <- 1 - svm_model_poly$results$Accuracy
estimated_test_error_rate_poly <- mean(test_error_poly)

# Output results
cat("Optimal Cost Parameter (Polynomial Kernel): ",
optimal_cost_poly, "\n")
cat("Estimated Test Error Rate (Polynomial Kernel): ",
estimated_test_error_rate_poly, "\n")



```
~~~

~~~
```{r}
# (c): Fit Support Vector Machine (SVM) with radial kernel and
optimal parameters
library(caret)
library(e1071)
svm_model_radial <- train(
  x = X,
  y = y,
  method = "svmRadial",
  trControl = cv,
  tuneGrid = expand.grid(C = seq(0.1, 10, by = 0.1), sigma = seq(0.1,
2, by = 0.1)),
  preProc = c("center", "scale"),
  tuneLength = 10
)

# Optimal cost and gamma parameters
optimal_cost_radial <- svm_model_radial$bestTune$C
optimal_sigma <- svm_model_radial$bestTune$sigma

# Step 5: Summarize key features of the fit
summary(svm_model_radial$finalModel)

# Step 6: Compute estimated test error rate
test_error_radial <- 1 - svm_model_radial$results$Accuracy
estimated_test_error_rate_radial <- mean(test_error_radial)

# Output results
cat("Optimal Cost Parameter (Radial Kernel):", optimal_cost_radial,
"\n")
cat("Optimal Sigma Parameter (Radial Kernel):", optimal_sigma, "\n")
cat("Estimated Test Error Rate (Radial Kernel):",
estimated_test_error_rate_radial, "\n")
~~~
```{r}
# (d) Compare results from the above three methods and also from the
method you
# recommended for these data in Mini Projects 3 and 4. Which method
would you
# recommend now?
results <- resamples(list(SVC = svm_linear, SVM_Polynomial =
svm_poly, SVM_Radial = svm_radial))
summary(results)
```

Question 3
```{r}

Hitters <- as.data.frame(sapply(Hitters, as.numeric))

# Extract predictors
```

```
predictors <- Hitters[, -which(names(Hitters) == "Salary")]

#(a) Standardizing variables before clustering
scaled_predictor <- scale(predictors)

#(b) Selecting a distance measure
# Since we standardized the variables, Euclidean distance is good for
clustering
#(c) Hierarchical clustering with complete linkage and Euclidean
distance
hierarchical_cluster <- hclust(dist(scaled_predictor), method =
"complete")

# Cut dendrogram at a height resulting in two distinct clusters
cut_dendo <- cutree(hierarchical_cluster, k = 2)

# Summarize cluster-specific means of variables
cluster_mean <- aggregate(predictors, by = list(cluster = cut_dendo),
FUN = mean)
print(cluster_mean)

# Summarize mean salaries of players in the two clusters
salary_mean <- tapply(Hitters$Salary, cut_dendo, mean)
print(salary_mean)

# dendrogram
plot(hierarchical_cluster, hang = -1, cex = 0.6)

# (d) K-means clustering with K = 2
kmeans_cluster <- kmeans(scaled_predictor, centers = 2)

kmeans_clustermean <- aggregate(predictors, by = list(cluster =
kmeans_cluster$cluster), FUN = mean)
print(kmeans_clustermean)

kmeans_salarymean <- tapply(Hitters$Salary, kmeans_cluster$cluster,
mean)
print(kmeans_salarymean)
` ` `
```