hw03

Samuel Swain

2023-03-05

Problem 1

1(a)

```
## KNN N-Fold CV K Performances (R Squared):
##
##  Best K: 8
##
##  - K = 6 : 0.5674
##  - K = 8 : 0.5868
##  - K = 10 : 0.5843
##  - K = 12 : 0.5769
```

N-Fold CV with Nearest Neighbors Pros:

- Lower bias since the model is trained on the entire data set
- No randomness since we are taking one row at a time instead of a random selection of rows
- Will not overestimate test error rate
- Fast model. Computational expense isn't as much of an issue here

Cons:

- Possibly high computational expense

1(b)

```
## Nearest Neighbor Prediction Error Standard Deviation: 0.5363876
```

1(c)

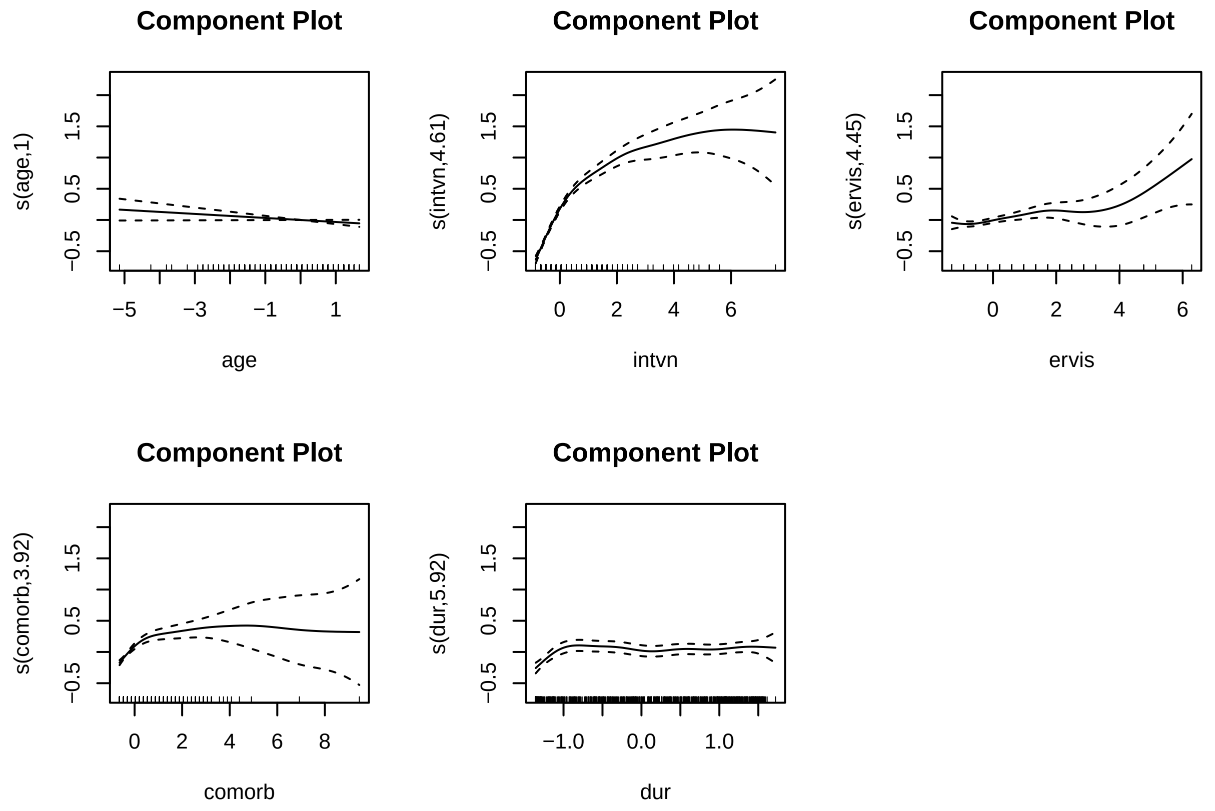
```
## log(cost) prediction: 3.379489
```

Problem 2

2(a)

```
##
```

```
## Family: gaussian
## Link function: identity
##
## Formula:
## cost ~ s(age) + gend + s(intvn) + drugs + s(ervis) + comp + s(comorb) +
##     s(dur)
##
## Parametric coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  2.73172    0.01656 164.919  < 2e-16 ***
## gend        -0.02975    0.01685  -1.766   0.0777 .
## drugs       -0.02904    0.02074  -1.400   0.1619
## comp         0.07184    0.01729   4.154 3.63e-05 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Approximate significance of smooth terms:
##             edf Ref.df       F  p-value
## s(age)    1.000  1.000   3.617  0.05757 .
## s(intvn)  4.610  5.563 136.168  < 2e-16 ***
## s(ervis)  4.450  5.435   3.101  0.00609 **
## s(comorb) 3.924  4.809  17.023  < 2e-16 ***
## s(dur)    5.917  7.043   5.514 3.52e-06 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## R-sq.(adj) =  0.685   Deviance explained = 69.4%
## GCV = 0.22296  Scale est. = 0.2162    n = 788
```



As we can see in the component plots above, intvn by far has the greatest effect on cost. We can also see that comorb, dur, and comp have a significant effect on cost. This is consistent with what we've seen in homework 2. Looking at the plots, we can see they are consistent with the most significant variables mentioned above.

2(b)

```
## GAM Prediction Error Standard Deviation: 0.4754392
```

N-Fold CV with Generalized Additive Models Pros:

- Lower bias since the model is trained on the entire data set
- No randomness since we are taking one row at a time instead of a random selection of rows
- Will not overestimate test error rate

Cons:

- Heavy computational expense. This model is harder to fit than a nearest neighbor model which will increase the run-time by a lot.

2(c)

```
## log(cost) prediction: 3.556478
```

Problem 3

3(a)

```
## Best Loess terms (CV):
##  - R Squared 0.6792923
##  - Degree: 1
##  - Span: 0.26
```

3(b)

```
## Best Loess terms (Cp):
##  - Cp: 0.1863456
##  - Degree: 1
##  - Span: 0.04
```

The hyperparameters selected through cross-validation are reasonably consistent with those chosen for Cp, except for a slightly lower value of span selected by Cp.

3(c)

```
## PPR Prediction Error Standard Deviation: 0.4694975
```

3(d)

```
## log(cost) prediction: 3.533041
```

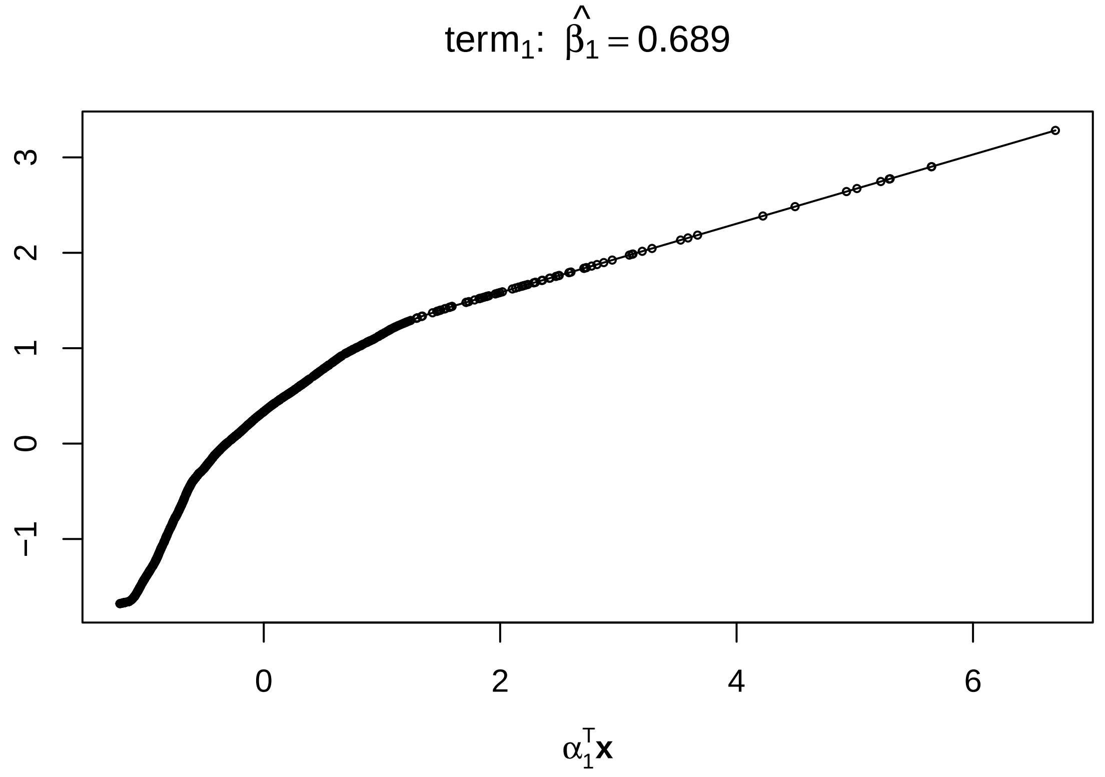
Question 4

4(a)

```
## Best PPR terms:
##  - n_terms: 1
```

4(b)

```
## PPR Prediction Error Standard Deviation: 0.4675063
```



```
## Call:
## ppr(formula = cost ~ ., data = df_1, nterms = best_nterms)
##
## Goodness of fit:
##  1 terms
## 165.5263
##
## Projection direction vectors ('alpha'):
##         age        gend       intvn       drugs       ervis        comp
## -0.02759078 -0.02686806  0.93275393 -0.07100287  0.10038411  0.15520531
##      comorb         dur
##  0.28667863  0.08423339
##
## Coefficients of ridge terms ('beta'):
##    term 1
## 0.6888639
```

The projection direction vectors, also known as the "alphas," show the relative importance of each variable in the model. In this case, the "intvn" variable had the highest importance with a value of 0.9328. The "age" and "gend" variables had negative importance with values of -0.0276 and -0.0269, respectively. The other variables, "drugs," "ervis," "comp," "comorb," and "dur" had positive importance values, but they were much smaller than the importance of "intvn." This conclusion aligns with the previous results.

4(c)

```
## log(cost) prediction: 3.510637
```

Problem 5

5(a)

```
## Best KNN Results:
##  - K: 8
##  - Misclass: 0.1527103
```

5(b)

```
## Best GAM Misclass: 0.1588785
```

5(c)

```
## Best NNet Results:
##  - Misclass: 0.1401869
##  - Size: 20
##  - Decay: 0.3
```

Based on the given results, the best neural network model (NNet) outperforms the KNN and GAM models in terms of misclassification rate. The NNet model has a lower misclassification rate (0.1401869) compared to KNN (0.1527103) and GAM (0.1588785).

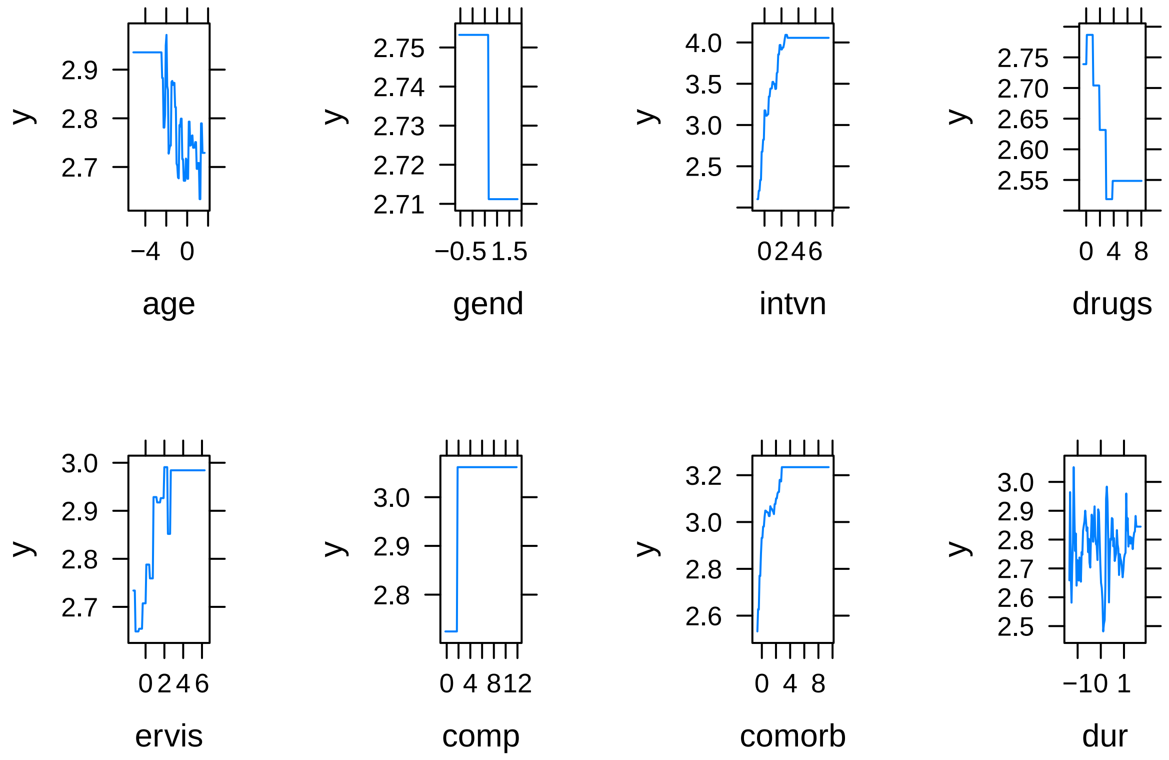
Problem 6

6(a)

```
## Best GBM Results:
##  - Best Error: 0.2113447
##  - Shrinkage: 0.05
##  - Best Depth: 3
```

6(b)

```
##           var    rel.inf
## intvn   intvn 71.9146940
## comorb comorb 11.1525469
## dur       dur  8.9565669
## ervis   ervis  3.6369610
## age       age  1.6729372
## comp     comp  1.4508561
## drugs   drugs  0.6877848
## gend     gend  0.5276531
```



As we can see, the top four important variables agree with question 2. They almost completely agree with the variables displayed in the ALE plot from question 2c, intvn, comorb, and dur vary the cost most. Comparing to the tree model fit in hw2 question 3c, we can see again intvn is the most important variable along with comorb. All of these models tell a very similar story. The most important feature by far is intervention (intvn).

6(c)

```
## log(cost) prediction: 3.493922
```

Problem 7

7(a)

```
## Random Forest Results:
##   - Avg. R Squared: 0.6699879
##   - Avg. MSE: 0.2259237
```

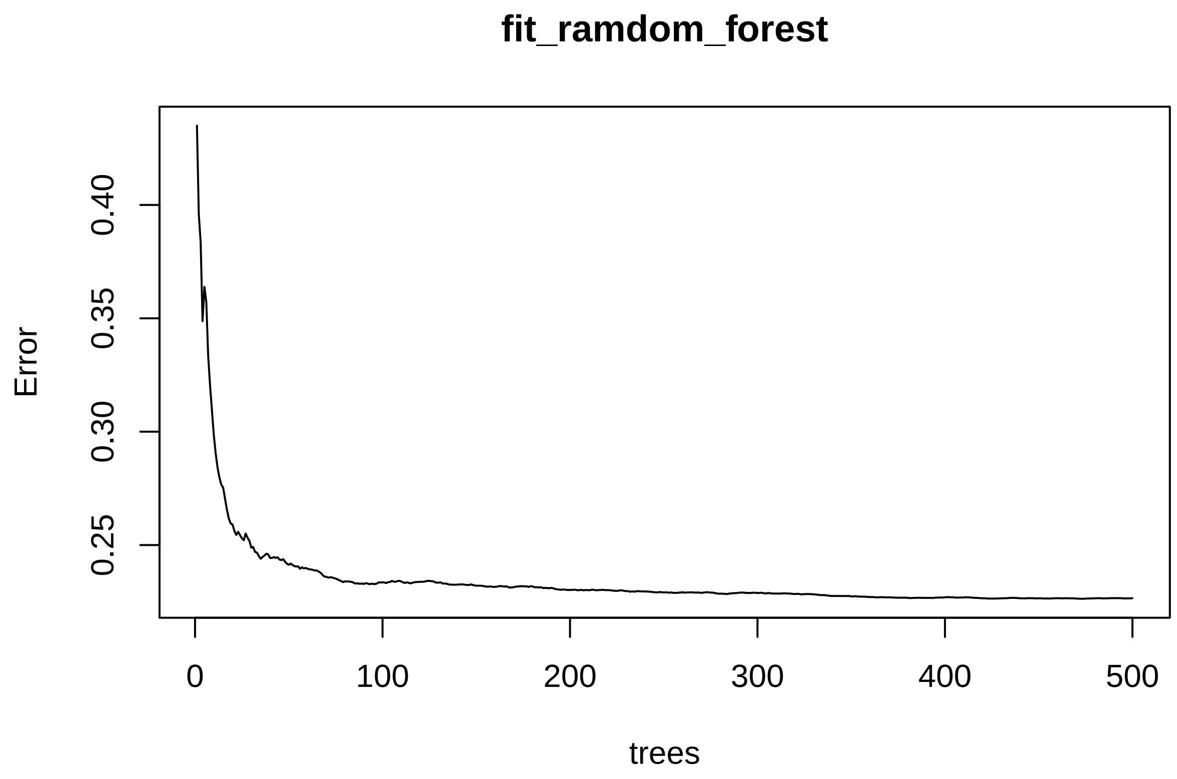
I ran the function 10 times and calculated the averages displayed above. In addition, I generated two lists of results, one for R Squared and one for MSE, which you can see below:

- R Squared: 0.6662825, 0.670427, 0.670467, 0.6679033, 0.6711121, 0.6731218, 0.6696226, 0.6712308, 0.670696, 0.6690163

- MSE: 0.2284604, 0.2256231, 0.2255957, 0.2273508, 0.2251541, 0.2237783, 0.2261738, 0.2250728, 0.225439, 0.2265889

The results do not change very much from trial to trial.

7(b)



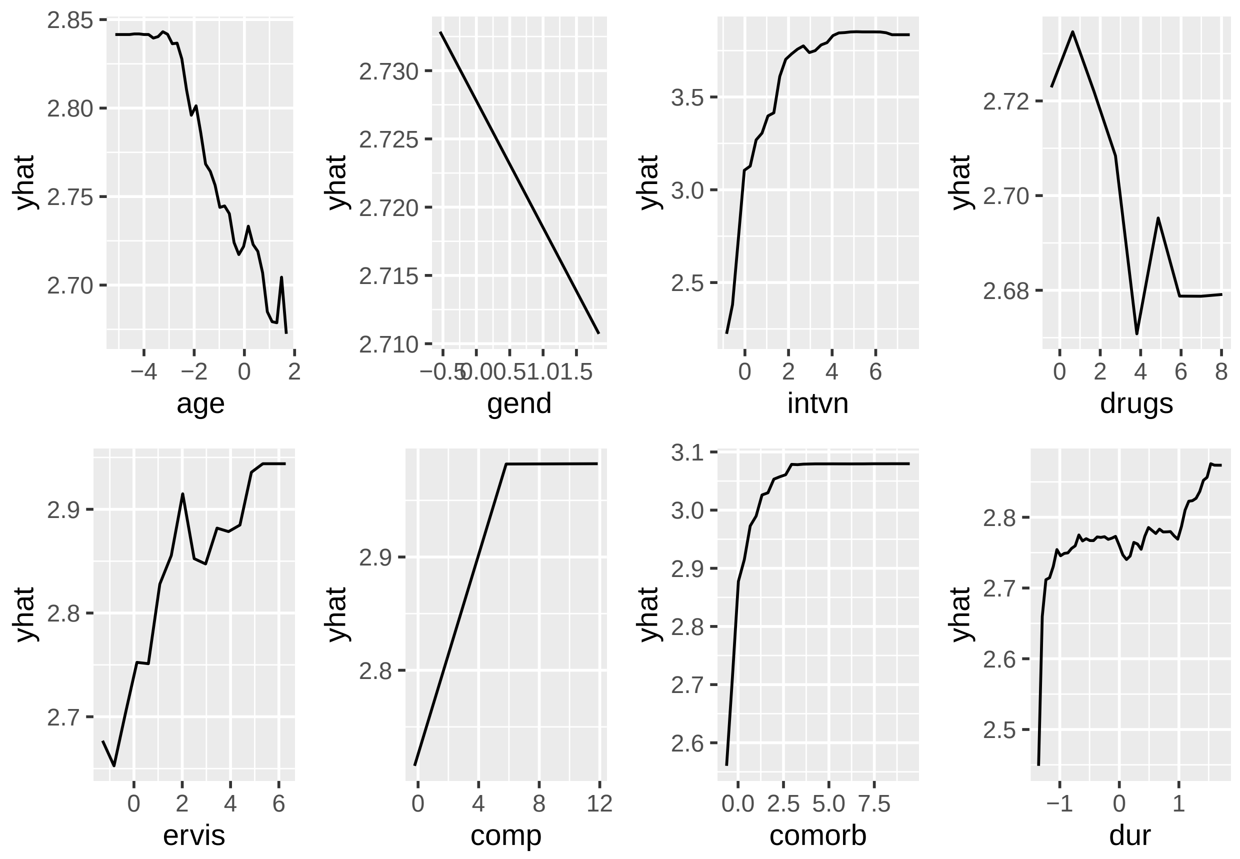
As depicted in the plot above, the OOB MSE reaches a steady state well before the number of trees hits 500. So yes, we have chosen a large enough number of trees.

7(c)

```
##           %IncMSE IncNodePurity
## age     -1.647639     32.613120
## gend    -2.746736      5.661579
## intvn  109.242328    246.407439
## drugs    2.122749     12.266366
## ervis   15.247719     44.656888
## comp    16.866591      9.646828
## comorb  39.891403     58.645770
## dur     21.888359     97.206801
```

Intvn, dur, and comorb are listed above as the most important variables in terms of the effect on the dependent variable. This follows the trend we have been seeing so far, that intvn has been the most influential.

7(d)



The top three predictors from the importance function above here are all increasing for every x. After increasing sharply, they still increase just more slowly. This is in agreement with the other tree model's dependence plot as well.

7(e)

```
## log(cost) prediction: 3.591617
```

7(f)

```
## Different mtrys (Random Forest):
##  - mtry=1: 0.5651916
##  - mtry=2: 0.6685748
##  - mtry=3: 0.6690163
```

It seems that the random forest model with mtry=3 yields the best performance. Mtry refers to the "Number of variables randomly sampled as candidates at each split."

7(g)

```
## Best Tree Model:
##  - Bucket Size: 16
##  - Cp: 0
## Best NNet Model:
##   - Size: 15
##   - Decay: 0.9
```

```
##    Model CV_R_squared
## 5    GBM    0.6870233
## 8   NNet    0.6847959
## 4    PPR    0.6827583
## 3 Loess    0.6744751
## 2    GAM    0.6703365
## 6     RF    0.6662337
## 7   Tree    0.6658147
## 1    KNN    0.5825930
```

Based on the table above, it appears that the GBM Model has the highest cross-validated R-squared value of 0.6870233. This suggests that the GBM model may be the most effective in predicting cost, as it explains a larger proportion of the variance in the outcome variable compared to the other models. The knn algorithm did significantly worse than all of the other algorithms. In general, NNet, PPR, and Loess performed pretty similarly to the GBM model.

Appendix

Commented out code is the output used to attain the results above. Commended to shorted final submission pdf.

```
# Libraries
library(readxl)
library(yaImpute)
library(combinat)
library(gbm)
library(mgcv)
library(nnet)
library(randomForest)
library(gridExtra)
library(pdp)
library(boot)
library(glmnet)
library(ALEPlot)
library(rpart)

# Cross Validation
CVInd <- function(n,K) {
  # n is sample size; K is number of parts;
  # returns K-length list of indices for each part
  m<-floor(n/K) #approximate size of each part
  r<-n-m*K
  I<-sample(n,n) #random reordering of the indices
  Ind<-list() #will be list of indices for all K parts
  length(Ind)<-K
  for (k in 1:K) {
    if (k <= r) kpart <- ((m+1)*(k-1)+1):((m+1)*k)
    else kpart<-((m+1)*r+m*(k-r-1)+1):((m+1)*r+m*(k-r))
    Ind[[k]] <- I[kpart] #indices for kth part of data
  }
  Ind
}

# Standardize Function
standardize_predictors <- function(data) {
  predictors <- data
  standardized_predictors <- scale(predictors)
  data <- standardized_predictors
  return(data)
}
```

Problem 1

```
# read in the data
df_1 <- read_excel("HW3_data.xls")

# remove the first column
df_1 <- df_1[, -1]

# Get the mean and sd for columns
means <- colMeans(df_1[, -c(1)])
sds <- apply(df_1[, -c(1)], 2, sd)

# take the base-10 logarithm of the "cost" column
df_1$cost <- log10(df_1$cost)

# standardize predictors
X_1 = df_1[, -c(1)]
df_1[, c(2, 3, 4, 5, 6, 7, 8, 9)] = standardize_predictors(X_1)
```

1(a)

```
#### N-fold cross validation
K<-nrow(df_1)  #N-fold CV on each replicate
n.models = 4 #number of different models to fit
n=nrow(df_1)
y<-df_1$cost
yhat=matrix(0,n,n.models)
MSE<-matrix(0,1,n.models)

for (k in 1:K) {
  train<-as.matrix(df_1[-k,-c(1)])
  test<-as.matrix(df_1[k,-c(1)])
  ytrain<-df_1[-k,1]$cost
  K1=6;K2=8;K3=10;K4=12;
  out<-ann(train,test,K1,verbose=F)
```

```r
  ind<-t(as.matrix(out$knnIndexDist[,1:K1]))
  yhat[k,1]<-apply(ind,1,function(x) mean(ytrain[x]))
  out<-ann(train,test,K2,verbose=F)
  ind<-t(as.matrix(out$knnIndexDist[,1:K2]))
  yhat[k,2]<-apply(ind,1,function(x) mean(ytrain[x]))
  out<-ann(train,test,K3,verbose=F)
  ind<-t(as.matrix(out$knnIndexDist[,1:K3]))
  yhat[k,3]<-apply(ind,1,function(x) mean(ytrain[x]))
  out<-ann(train,test,K4,verbose=F)
  ind<-t(as.matrix(out$knnIndexDist[,1:K4]))
  yhat[k,4]<-apply(ind,1,function(x) mean(ytrain[x]))
} #end of k loop
MSE[1,]=apply(yhat,2,function(x) sum((y-x)^2))/n

MSEAve <- apply(MSE,2,mean) # averaged mean square CV error
MSEsd <- apply(MSE,2,sd) # SD of mean square CV error
r2<-1-MSEAve/var(y) # CV r^2

k_list = c(K1, K2, K3, K4)
best_K = k_list[which.max(r2)]

cat(
  "KNN N-Fold CV K Performances (R Squared):", "\n", "\n", "Best K:", best_K, "\n", "\n",
  "- K =", K1, ":", round(r2[1], 4), "\n",
  "- K =", K2, ":", round(r2[2], 4), "\n",
  "- K =", K3, ":", round(r2[3], 4), "\n",
  "- K =", K4, ":", round(r2[4], 4), "\n"
)
```

1(b)

```r
Nrep<-50 #number of replicates of CV
K<-10  #K-fold CV on each replicate
n.models = 1 #number of different models to fit
n=nrow(df_1)
y<-df_1$cost
yhat=matrix(0,n,n.models)
MSE<-matrix(0,Nrep,n.models)
for (j in 1:Nrep) {
  Ind<-CVInd(n,K)
  for (k in 1:K) {
    train<-as.matrix(df_1[-Ind[[k]],-c(1)])
    test<-as.matrix(df_1[Ind[[k]],-c(1)])
    ytrain<-df_1[-Ind[[k]],1]$cost
    K1=best_K;
    best_knn<-ann(train,test,K1,verbose=F)
    ind<-as.matrix(best_knn$knnIndexDist[,1:K1])
    yhat[Ind[[k]],1]<-apply(ind,1,function(x) mean(ytrain[x]))
  } #end of k loop
  MSE[j,]=apply(yhat,2,function(x) sum((y-x)^2))/n
} #end of j loop
sd_prediction_error_ave <- sqrt(apply(MSE,2,mean)) # sd_prediction_error
```

```r
cat("Nearest Neighbor Prediction Error Standard Deviation:", sd_prediction_error_ave[1])
```

1(c)

```r
# create a dataframe for the new observation
new_obs <- data.frame(
  age = 59,
  gend = 0,
  intvn = 10,
  drugs = 0,
  ervis = 3,
  comp = 0,
  comorb = 4,
  dur = 300
)

# standardize the new observation using the mean and SD from the training data
new_obs_std <- scale(new_obs, center = means, scale = sds)[1,]

# Predict on unseen data
train<-as.matrix(df_1[,-c(1)])
test<-t(as.matrix(new_obs_std))
ytrain<-df_1$cost
best_knn<-ann(train,test,best_K,verbose=F)
ind<-t(as.matrix(best_knn$knnIndexDist[,1:best_K]))
D<-as.matrix(best_knn$knnIndexDist[,(1+best_K):(2*best_K)])
prediction<-apply(ind,1,function(x) mean(ytrain[x]))

cat("log(cost) prediction:", prediction)
```

Problem 2

2(a)

```r
# Fit the GAM model
gam_fit <- gam(cost ~ s(age) + gend + s(intvn) + drugs + s(ervis) + comp + s(comorb) + s(dur),
               data = df_1, family = gaussian(), sp = c(-1,-1,-1,-1,-1,-1,-1,-1))

# Create a 2x3 grid of plots
par(mfrow = c(2, 3))

# Out summary
summary(gam_fit)

# Plot the GAM model
plot(gam_fit, main = "Component Plot")
```

2(b)

```r
##Now use multiple reps of CV to compare Neural Nets and linear reg models###
Nrep<-50 # number of replicates of CV
K<-10 # K-fold CV on each replicate
n.models = 1 # number of different models to fit
n=nrow(df_1)
y<-df_1$cost
yhat=matrix(0,n,n.models)
MSE<-matrix(0,Nrep,n.models)
for (j in 1:Nrep) {
  Ind<-CVInd(n,K)
  for (k in 1:K) {
    out <- gam(cost ~ s(age) + gend + s(intvn) + drugs + s(ervis) + comp + s(comorb) + s(dur),
               data = df_1[-Ind[[k]],], family = gaussian(), sp = c(-1,-1,-1,-1,-1,-1,-1,-1))
    yhat[Ind[[k]],1] <- as.numeric(predict(out,df_1[Ind[[k]],]))
  } #end of k loop
  MSE[j,]=apply(yhat,2,function(x) sum((y-x)^2))/n
} #end of j loop

sd_prediction_error_ave <- sqrt(apply(MSE,2,mean)) # sd_prediction_error

cat("GAM Prediction Error Standard Deviation:", sd_prediction_error_ave[1])
```

2(c)

```r
# Predicting using GAM on new data
prediction = predict(gam_fit, newdata = data.frame(t(as.matrix(new_obs_std))))

# Print prediction
cat("log(cost) prediction:", prediction)
```

Problem 3

3(a)

```r
####CV to choose the best K
Nrep<-5 #number of replicates of CV
K<-10  #K-fold CV on each replicate
n.models = 1 #number of different models to fit
n=nrow(df_1)
y<-df_1$cost
yhat=matrix(0,n,n.models)
MSE<-matrix(0,Nrep,n.models)

# Hyperparameters to try
span = seq(.02,.3,.02)
degree = c(0, 1, 2)
```

```r
# Store best params
best_r2 <- -Inf
best_loess_coefs_cv = c(0, 0)

for (dg in degree) {
  for (sp in span) {
    for (j in 1:Nrep) {
      Ind<-CVInd(n,K)
      for (k in 1:K) {

        out<-loess(cost ~., df_1[-Ind[[k]], c(1, 4, 8, 9, 6)], degree=dg, span=sp,
                   control=loess.control(surface="direct"))
        yhat[Ind[[k]],1]<-as.numeric(predict(out,df_1[Ind[[k]],]))

      } #end of k loop
      MSE[j,]=apply(yhat,2,function(x) sum((y-x)^2))/n
    } #end of j loop
    MSEAve <- apply(MSE,2,mean) # averaged mean square CV error
    MSEsd <- apply(MSE,2,sd) # SD of mean square CV error
    r2<-1-MSEAve/var(y) # CV r^2

    if (r2 > best_r2) {
      best_r2 <- r2
      best_loess_coefs_cv <- c(dg, sp)
    }

  }
}

cat("Best Loess terms (CV):", "\n",
    "- R Squared", best_r2, "\n",
    "- Degree:", best_loess_coefs_cv[1], "\n",
    "- Span:", best_loess_coefs_cv[2], "\n")
```

3(b)

```r
# Define initial best sigma-hat value
sig_hat_selection <- c(degree = 0, span = 0, s = Inf)

# Loop over degree and span values
for (dg in degree) {
  for (sp in span) {
    # Fit loess model and check s statistic
    out <- loess(cost ~ ., df_1[, c(1, 4, 8, 9, 6)], degree = dg, span = sp)

    # Update sigma-hat selection if current s is lower
    if (out$s < sig_hat_selection[3]) {
      sig_hat_selection <- c(degree = dg, span = sp, s = out$s)
    }
  }
}
```

```
# Select sigma-hat value
sig_hat <- 0.1

# Print selected sigma-hat values
# print(sig_hat_selection) # Uncomment to see selected s value

# Define initial best parameters
best_loess_coefs_cp <- c(degree = 0, span = 0, Cp = Inf)

# Loop over degree and span values
for (dg in degree) {
  for (sp in span) {

    # Fit loess model and compute Cp statistic
    out <- loess(cost ~ ., df_1[, c(1, 4, 8, 9, 6)], degree = dg, span = sp)
    SSE <- sum((df_1$cost - out$fitted)^2)
    Cp <- (SSE + 2 * out$trace.hat * sig_hat^2) / nrow(df_1)

    # Update best parameters if current Cp is lower
    if (Cp < best_loess_coefs_cp[3]) {
      best_loess_coefs_cp <- c(degree = dg, span = sp, Cp = Cp)
    }
  }
}

# Print best parameters
cat("Best Loess terms (Cp):\n",
    "- Cp:", best_loess_coefs_cp[3], "\n",
    "- Degree:", best_loess_coefs_cp[1], "\n",
    "- Span:", best_loess_coefs_cp[2])
```

3(c)

```
####CV to Find Best nterms in PPR on Concrete Data
Nrep<-50 #number of replicates of CV
K<-10  #K-fold CV on each replicate
n.models = 1 #number of different models to fit
n=nrow(df_1)
y<-df_1$cost
yhat=matrix(0,n,n.models)
MSE<-matrix(0,Nrep,n.models)

for (j in 1:Nrep) {
  Ind<-CVInd(n,K)
  for (k in 1:K) {
    out <- loess(cost ~ ., df_1[-Ind[[k]], c(1, 4, 8, 9, 6)],
                 degree = best_loess_coefs_cv[1],
                 span = best_loess_coefs_cv[2],
                 control=loess.control(surface="direct"))
    yhat[Ind[[k]],1]<-as.numeric(predict(out,df_1[Ind[[k]],]))
  } #end of k loop
```

```r
  MSE[j,]=apply(yhat,2,function(x) sum((y-x)^2))/n
} #end of j loop
sd_prediction_error_ave <- sqrt(apply(MSE,2,mean)) # sd_prediction_error

cat("PPR Prediction Error Standard Deviation:", sd_prediction_error_ave[1], "\n")
```

3(d)

```r
# Fit best loess model
best_loess <- loess(cost ~ ., df_1[, c(1, 4, 8, 9, 6)], degree = best_loess_coefs_cv[1],
                    span = best_loess_coefs_cv[2])

# Predicting using PPR on new data
prediction = predict(best_loess, newdata = data.frame(t(as.matrix(new_obs_std))))

# Print prediction
cat("log(cost) prediction:", prediction)
```

Question 4

4(a)

```r
# Define constants
Nrep <- 5 # Number of replicates of CV
K <- 10 # K-fold CV on each replicate
n.models <- 1 # Number of different models to fit
n <- nrow(df_1)
y <- df_1$cost
yhat <- matrix(0, n, n.models)
MSE <- matrix(0, Nrep, n.models)

# Hyperparameters to try
n_terms <- seq(1, 10, 1)

# Store best params
best_r2 <- -Inf
best_nterms <- 0

# Iterate over hyperparameters
for (term in n_terms) {

  # Run cross-validation
  for (j in 1:Nrep) {
    Ind <- CVInd(n, K)
    for (k in 1:K) {
      out <- ppr(cost ~ ., data = df_1[-Ind[[k]], ], nterms = term)
      yhat[Ind[[k]], 1] <- as.numeric(predict(out, df_1[Ind[[k]], ]))
    } # End of k loop
    MSE[j, ] <- apply(yhat, 2, function(x) sum((y - x)^2)) / n
  } # End of j loop

  # Calculate CV r^2 and update best params
  MSEAve <- apply(MSE, 2, mean) # Averaged mean square CV error
  r2 <- 1 - MSEAve / var(y) # CV r^2

  if (r2 > best_r2) {
    best_r2 <- r2[1]
    best_nterms <- term
  }

} # End of term loop

cat("Best PPR terms:", "\n", "- n_terms:", best_nterms)
```

4(b)

```r
# CV to Find Best nterms in PPR on Concrete Data
Nrep <- 50  # number of replicates of CV
K <- 10  # K-fold CV on each replicate
n.models <- 1  # number of different models to fit
n <- nrow(df_1)
y <- df_1$cost
yhat <- matrix(0, n, n.models)
MSE <- matrix(0, Nrep, n.models)

for (j in 1:Nrep) {
  Ind <- CVInd(n, K)
  for (k in 1:K) {
    out <- ppr(cost ~ ., data = df_1[-Ind[[k]], ], nterms = best_nterms)
    yhat[Ind[[k]], 1] <- as.numeric(predict(out, df_1[Ind[[k]], ]))
  }
  MSE[j, ] <- apply(yhat, 2, function(x) sum((y - x)^2)) / n
}

# Calculate prediction error standard deviation
sd_prediction_error_ave <- sqrt(apply(MSE, 2, mean))

cat("PPR Prediction Error Standard Deviation:", sd_prediction_error_ave[1], "\n\n")

# Final model
best_ppr <- ppr(cost ~ ., data = df_1, nterms = best_nterms)

plot(best_ppr)
summary(best_ppr)
```

4(c)

```r
# Predicting using PPR on new data
prediction = predict(best_ppr, newdata = data.frame(t(as.matrix(new_obs_std))))
```

```r
# Print prediction
cat("log(cost) prediction:", prediction)
```

Problem 5

```r
# Read data
df_5 <- read_excel("HW3_data.xls", sheet = "FGL data")

# remove the first column
df_5 <- df_5[, -1]

# Convert data to binary classification
df_5$type <- ifelse(df_5$type %in% c("WinF", "WinNF"), 0, 1)

# standardize predictors
X_5 = df_5[, -c(10)]
df_5[, -c(10)] = standardize_predictors(X_5)
```

5(a)

```r
# Set up variables
Nrep <- 50 # Number of replicates of CV
K <- 10   # K-fold CV on each replicate
n.models <- 1 # Number of different models to fit
n <- nrow(df_5)
y <- df_5$type
phat <- matrix(0, n, n.models)
misclass <- matrix(0, Nrep, n.models)

# Hyperparameters to try
nearest_neighbors <- seq(1, 50, 1)

# Set up variables to keep track of the best model so far
best_misclass <- 1
best_k <- 0

# Loop through different values of K and number of nearest neighbors
for (nearest_n in nearest_neighbors) {
  for (j in 1:Nrep) {
    Ind <- CVInd(n, K)
    y_hat <- y
    for (k in 1:K) {

      train <- as.matrix(df_5[-Ind[[k]], -c(10)])
      test <- as.matrix(df_5[Ind[[k]], -c(10)])
      ytrain <- df_5[-Ind[[k]], ]$type

      best_knn <- ann(train, test, nearest_n, verbose = FALSE)
      ind <- as.matrix(best_knn$knnIndexDist[, 1:nearest_n])
      p_hat <- apply(ind, 1, function(x) mean(ytrain[x]))
      y_hat[Ind[[k]]] <- ifelse(p_hat >= 0.5, 1, 0)

    } # End of k loop
    misclass[j, ] <- mean(y != y_hat)
  } # End of j loop

  misclass_ave <- mean(misclass)

  # Check if this model is better than the previous best model
  if (misclass_ave < best_misclass) {
    best_misclass <- misclass_ave
    best_k <- nearest_n
  }
}

# Print the best model found
cat("Best KNN Results:", "\n", "- K:", best_k, "\n", "- Misclass:", best_misclass)

# Save best knn results
best_misclass_knn_5a <- best_misclass
```

5(b)

```r
# Set up variables
Nrep <- 3 # Number of replicates of CV
K <- 10  # K-fold CV on each replicate
n <- nrow(df_5)
y <- df_5$type
misclass <- matrix(0, Nrep, 1)

predictors <- c("s(RI)", "s(Na)", "s(Mg)",
                "s(Al)", "s(Si)", "s(K)",
                "s(Ca)", "s(Ba)", "s(Fe)")

for (j in 1:Nrep) {
  Ind <- CVInd(n, K)
  y_hat <- y
  for (k in 1:K) {

    formula_str <- paste("type ~", predictors[1], "+", predictors[2], "+", predictors[3],
                         "+", predictors[4], "+", predictors[5], "+", predictors[6],
                         "+", predictors[7], "+", predictors[8], "+", predictors[9])

    out <- gam(as.formula(formula_str), data = df_5[-Ind[[k]],],
               family = binomial(), method = 'ML')

    p_hat <- predict(out, newdata = df_5[Ind[[k]],], type = "response")
    y_hat[Ind[[k]]] <- ifelse(p_hat >= 0.5, 1, 0)

  } # End of k loop
  misclass[j, ] <- mean(y != y_hat)
} # End of j loop

misclass_ave <- mean(misclass)

# Print the best model found
cat("Best GAM Misclass:", misclass_ave)

# Save best gam results
best_misclass_gam_5b <- misclass_ave
```

5(c)

```
# Define constants
Nrep <- 3 # Number of replicates of CV
K <- 10 # K-fold CV on each replicate
n.models <- 1 # Number of different models to fit
n <- nrow(df_5)
y <- df_5$type
phat <- matrix(0, n, n.models)
misclass <- matrix(0, Nrep, n.models)

# Hyperparameters to try
num_nodes <- seq(5, 20, 5)
decay <- c(seq(0, 1, .1), 2, 5, 10)

# Set up variables to keep track of the best model so far
best_misclass <- 1
best_num_nodes <- 0
best_decay <- 0

# Iterate over hyperparameters
for (node in num_nodes) {
  for (dec in decay) {
    # Run cross-validation
    for (j in 1:Nrep) {
      Ind <- CVInd(n, K)
      y_hat <- y
      for (k in 1:K) {

        # Fit nnet model for each fold and rep
        out<-nnet(type~.,df_5[-Ind[[k]],], linout=T, skip=F, size=node,
                  decay=dec, maxit=1000, trace=F)
        p_hat <- predict(out, newdata = df_5[Ind[[k]],], type = "raw")
        y_hat[Ind[[k]]] <- ifelse(p_hat >= 0.5, 1, 0)

      } # End of k loop
      misclass[j, ] <- mean(y != y_hat)
    } # End of j loop

    # Calculate CV misclass
    misclass_ave <- mean(misclass)
```

```r
    # Check if this model is better than the previous best model
    if (misclass_ave < best_misclass) {
      best_misclass <- misclass_ave
      best_num_nodes <- node
      best_decay <- dec
    }
  }
}

# Print NNet results
cat("Best NNet Results:", "\n",
    "- Misclass:", best_misclass, "\n",
    "- Size:", best_num_nodes, "\n",
    "- Decay:", best_decay)

best_misclass_nnet_5c <- best_misclass
```

Problem 6

6(a)

```r
# create grid of hyperparameters - 4 values for shrinkage, 4 values for interaction depth
# the optimal number of trees will be chosen in the loop
confs <- expand.grid(
  shrinkage = c(0.005,0.01,0.05,0.1),
  interaction.depth = c(1,3,5,7)
)

# NOTE: loop takes some time to run
for (i in 1:nrow(confs)){

  # set seed to use the same K-fold partition across different hyperparameter values
  set.seed(123)

  gbm_tune <- gbm(
    cost~., data=as.data.frame(df_1),  distribution="gaussian", n.trees=2000,
    n.minobsinnode = 10,
    # hyperparameters
    shrinkage=confs$shrinkage[i], interaction.depth=confs$interaction.depth[i],
    # 5-fold cross-validation
    cv.folds = 5,
    verbose=FALSE
  )

  # update optimal number of trees and best cross-validation loss
  confs$optimal.trees[i] <- which.min(gbm_tune$cv.error)
  confs$best.error[i] <- gbm_tune$cv.error[confs$optimal.trees[i]] # mean-squared error
}

# Fit best gbm
```

```r
best_GBM <- gbm(
  cost~., data=as.data.frame(df_1),  distribution="gaussian", n.trees=2000,
  n.minobsinnode = 10,
  # hyperparameters
  shrinkage=confs[order(confs$best.error),][1, 1],
  interaction.depth=confs[order(confs$best.error),][1, 2],
  # 5-fold cross-validation
  cv.folds = 5,
  verbose=FALSE
)

# Best hyperparameters
best_shrinkage <- confs[order(confs$best.error),][1, 1]
best_depth <- confs[order(confs$best.error),][1, 2]
best_n_trees <- 2000
best_iteration <- confs[order(confs$best.error),][1, 3]
best_sse_cv <- confs[order(confs$best.error),][1, 4]

# Print NNet results
cat("Best GBM Results:", "\n",
    "- Best Error:", best_sse_cv, "\n",
    "- Shrinkage:", best_shrinkage, "\n",
    "- Best Depth:", best_depth)
```

6(b)

```r
best.iter <- confs[order(confs$best.error),][1, 3]
summary_output <- summary(best_GBM, n.trees = best.iter, plotit = FALSE)
summary_output
par(mfrow=c(2, 3))
pd_plots <- lapply(1:8,function(i) plot(best_GBM,i.var=i))
do.call(gridExtra::grid.arrange,list(grobs=pd_plots,ncol=4))
```

6(c)

```r
# assume your new data is stored in a data frame called "new_data"
prediction <- predict(best_GBM, newdata = data.frame(t(as.matrix(new_obs_std))),
                      n.trees = best.iter)

# Print prediction
cat("log(cost) prediction:", prediction)
```

Problem 7

7(a)

```r
r_sq_list = c()
mse_list = c()

for (i in 1:10) {
  fit_ramdom_forest <- randomForest(cost~., data=df_1, mtry=3, ntree = 500,
                                    importance = TRUE)

  r_sq_list = c(r_sq_list, tail(fit_ramdom_forest$rsq, 1))
  mse_list = c(mse_list, tail(fit_ramdom_forest$mse, 1))

}

cat("Random Forest Results:", "\n",
      "- Avg. R Squared:", mean(r_sq_list), "\n",
      "- Avg. MSE:", mean(mse_list))
```

7(b)

```r
plot(fit_ramdom_forest)
```

7(c)

```r
importance(fit_ramdom_forest)
```

7(d)

```r
plots <- lapply(1:8, function(i) {
  pdp::partial(fit_ramdom_forest, pred.var = names(df_1)[i+1], pred.data = df_1, plot = TRUE,
               plot.engine = "ggplot2")
})

# arrange the plots in a 2 x 4 grid
grid.arrange(grobs = plots, ncol = 4)
```

7(e)

```r
# assume your new data is stored in a data frame called "new_data"
prediction <- predict(fit_ramdom_forest, newdata = data.frame(t(as.matrix(new_obs_std))))

# Print prediction
cat("log(cost) prediction:", prediction)
```

7(f)

```r
# mtry=3 rsq
rf_mtry_rsq <- c(0, 0, tail(fit_ramdom_forest$rsq, 1))

# mtry=1
fit_ramdom_forest <- randomForest(cost~., data=df_1, mtry=1, ntree = 500,
                                  importance = TRUE)
rf_mtry_rsq[1] <- tail(fit_ramdom_forest$rsq, 1)

# mtry=2
fit_ramdom_forest <- randomForest(cost~., data=df_1, mtry=2, ntree = 500,
                                  importance = TRUE)
rf_mtry_rsq[2] <- tail(fit_ramdom_forest$rsq, 1)

cat("Different mtrys (Random Forest):", "\n",
    "- mtry=1:", rf_mtry_rsq[1], "\n",
    "- mtry=2:", rf_mtry_rsq[2], "\n",
    "- mtry=3:", rf_mtry_rsq[3])
```

7(g)

```r
# Find best tree and nnet model

# Best tree
Nrep<-10 # number of replicates of CV
K<-10  # K-fold CV on each replicate
n.models = 1 # number of different models to fit
n=nrow(df_1)
y<-df_1$cost
yhat=matrix(0,n,n.models)
MSE<-matrix(0,Nrep,n.models)

# Hyperparameters to try
bucket_sizes <- c(seq(2, 20, 2))
cps <- c(0, 0.0000001, 0.000001, 0.00001, 0.0001, 0.001, 0.01, 0.1, 1, 10)

# Best hyperparameters
best_bucket_size <- 0
best_cp <- 0
best_rsq <- 0

for (size in bucket_sizes) {
  for (cp in cps) {
    for (j in 1:Nrep) {
      Ind<-CVInd(n,K)
      for (k in 1:K) {
        control <- rpart.control(minbucket = size, cp = cp, maxsurrogate = 0,
                                 usesurrogate = 0, xval = 0)
        out <- rpart(cost~.,df_1[-Ind[[k]],], control = control)
        yhat[Ind[[k]],1] <- predict(out,df_1[Ind[[k]],])
      } #end of k loop
      MSE[j,]=apply(yhat,2,function(x) sum((y-x)^2))/n
    } #end of j loop
    MSEAve <- apply(MSE,2,mean)
    MSEsd <- apply(MSE,2,sd)
    r2<-1-MSEAve/var(y)

    # Check if this model is better than the previous best model
    if (best_rsq < r2) {
      best_rsq <- r2
      best_bucket_size <- size
      best_cp <- cp
    }

  }
}

cat("Best Tree Model:", "\n",
    "- Bucket Size:", best_bucket_size, "\n",
    "- Cp:", best_cp)
```

```r
# Best NNet
Nrep<-1 # number of replicates of CV
K<-10  # K-fold CV on each replicate
n.models = 1 # number of different models to fit
n=nrow(df_1)
y<-df_1$cost
yhat=matrix(0,n,n.models)
MSE<-matrix(0,Nrep,n.models)

# Hyperparameters to try
num_nodes <- seq(5, 20, 5)
decay <- c(seq(0, 1, .1), 2)

# Best hyperparameters
best_decay_7g <- 0
best_size_7g <- 0
best_rsq <- 0

for (size in num_nodes) {
  for (dec in decay) {
    for (j in 1:Nrep) {
      Ind<-CVInd(n,K)
      for (k in 1:K) {
        out<-nnet(cost~.,df_1[-Ind[[k]],], linout=T, skip=F, size=size,
                  decay=dec, maxit=1000, trace=F)
        yhat[Ind[[k]],1]<-predict(out,df_1[Ind[[k]],])
      } #end of k loop
      MSE[j,]=apply(yhat,2,function(x) sum((y-x)^2))/n
    } #end of j loop
    MSEAve <- apply(MSE,2,mean)
    MSEsd <- apply(MSE,2,sd)
    r2<-1-MSEAve/var(y)
    # Check if this model is better than the previous best model
    if (best_rsq < r2) {
      best_rsq <- r2
      best_size_7g <- size
      best_decay_7g <- dec
    }

  }
}

cat("Best NNet Model:", "\n",
    "- Size:", best_size_7g, "\n",
    "- Decay:", best_decay_7g)
```

```
# Best NNet
Nrep<-1 # number of replicates of CV
K<-10  # K-fold CV on each replicate
n.models = 8 # number of different models to fit
n=nrow(df_1)
y<-df_1$cost
yhat=matrix(0,n,n.models)
MSE<-matrix(0,Nrep,n.models)

for (j in 1:Nrep) {
  Ind<-CVInd(n,K)
  for (k in 1:K) {

    set.seed(123)

    # KNN
    train<-as.matrix(df_1[-Ind[[k]],-c(1)])
    test<-as.matrix(df_1[Ind[[k]],-c(1)])
    ytrain<-df_1[-Ind[[k]],1]$cost
    out<-ann(train,test,best_K,verbose=F)
    ind<-as.matrix(out$knnIndexDist[,1:best_K])
    yhat[Ind[[k]], 1]<-apply(ind,1,function(x) mean(ytrain[x]))

    # GAM
    out <- gam(cost ~ s(age) + gend + s(intvn) + drugs + s(ervis) + comp + s(comorb) + s(dur),
               data = df_1[-Ind[[k]],], family = gaussian(), sp = c(-1,-1,-1,-1,-1,-1,-1,-1))
    yhat[Ind[[k]], 2] <- predict(out,df_1[Ind[[k]], ])

    # Loess
    out <- loess(cost ~ ., df_1[-Ind[[k]], c(1, 4, 8, 9, 6)], degree = best_loess_coefs_cv[1],
                 span = best_loess_coefs_cv[2], control=loess.control(surface="direct"))
    yhat[Ind[[k]], 3] <- predict(out,df_1[Ind[[k]], ])

    # PPR
    out <- ppr(cost ~ ., data = df_1[-Ind[[k]], ], nterms = best_nterms)
    yhat[Ind[[k]], 4] <- predict(out, df_1[Ind[[k]], ])

    # GBM
    out <- gbm(cost ~ ., data=df_1[-Ind[[k]], ], distribution="gaussian", n.trees=best_n_trees,
               shrinkage=best_shrinkage, interaction.depth=best_depth, bag.fraction = .5,
               train.fraction = 1, n.minobsinnode = 10, cv.folds = 10,
               keep.data=TRUE, verbose=FALSE)
    yhat[Ind[[k]], 5] <- predict(out, df_1[Ind[[k]], ])

    # RF
    out <- randomForest(cost~., data=df_1[-Ind[[k]], ], mtry=c(1, 2, 3)[which.max(rf_mtry_rsq)],
                        ntree = 500, importance = TRUE)
    yhat[Ind[[k]], 6] <- predict(out, df_1[Ind[[k]], ])

    # Tree
    control <- rpart.control(minbucket = best_bucket_size, cp = best_cp, maxsurrogate = 0,
                             usesurrogate = 0, xval = 0)
    out <- rpart(cost~.,df_1[-Ind[[k]],], control = control)
    yhat[Ind[[k]], 7] <- predict(out,df_1[Ind[[k]],])

    # NNet
    out<-nnet(cost~.,df_1[-Ind[[k]],], linout=T, skip=F, size=best_size_7g, decay=best_decay_7g,
              maxit=1000, trace=F)
    yhat[Ind[[k]], 8] <- predict(out,df_1[Ind[[k]],])

  } #end of k loop
  MSE[j,]=apply(yhat,2,function(x) sum((y-x)^2))/n
} #end of j loop
MSEAve <- apply(MSE,2,mean)
MSEsd <- apply(MSE,2,sd)
r2<-1-MSEAve/var(y)

# Create a vector of model names
model_names <- c("KNN", "GAM", "Loess", "PPR",
                 "GBM", "RF", "Tree", "NNet")

# Combine the vectors into a data frame
cv_table <- data.frame(Model = model_names, CV_R_squared = r2)
cv_table <- cv_table[order(cv_table$CV_R_squared, decreasing = TRUE),]

# Print the table
print(cv_table)
```