***Note: Code that I created or substantially modified from Alvaro I have highlighted in yellow.

HW2 Library.R

***Note: I skip the libraries and loading messages.

```
# CVEN 6833
# HW 2 Library
# Adopted from Alvaro Ossandon
# Edited by Nathan Bonham

## import data

Summer_Precip=read.table('http://civil.colorado.edu/~balajir/CVEN6833/HWs/HW-2/data/Precipitaton_data.txt',
header=T)
Pm=Summer_Precip$Prec
X=Summer_Precip[-which(colnames(Summer_Precip)=='Prec')]

x0 =  read.delim("G:/My Drive/CU Boulder/Classes/20-21/CVEN 6833 Advanced
Stats/HW/HW2/data/Elevation_grid_1deg.txt")
x1 = x0[,1:3]

###################### SKIP libraries and loading messages ########################
# Quilt plotting function
Quilt_plotting=function(lon, lat, ypred, lon1, lat1, yob,type=""){
  if (type=="binary"){
    num.xgrids=(range(lon)[2]-range(lon)[1])/2
    num.ygrids=(range(lat)[2]-range(lat)[1])/2
    par(mfrow=c(3,1), mar=c(4,4,2,.5), mgp=c(1.5,.5,0)) #mfrow: rows and columns of plots, mar: margins on edges
of plots, mgp: axis label locations
    quilt.plot(nx=num.xgrids,ny=num.ygrids,lon, lat,ypred$fit,xlab="Longitude (deg)",ylab="Latitude
(deg)",main='Posterior Binary Precipitation on DEM Grid',zlim=range(ypred$fit,yob))
    grid(col="gray70",lty=2)
    US(add=TRUE, col="gray50", lwd=2,xlim=range(-125,-100))
    box()

    quilt.plot(lon1, lat1,yob,xlab="Longitude (deg)",ylab="Latitude (deg)",main='Observed Binary
Precipitation',zlim=range(ypred$fit,yob))
    grid(col="gray70",lty=2)
    US(add=TRUE, col="gray50", lwd=2,xlim=range(-125,-100))
    box()

    quilt.plot(lon, lat,ypred$se,xlab="Longitude (deg)",ylab="Latitude (deg)",main='Standard Error on DEM Grid')
    grid(col="gray70",lty=2)
    US(add=TRUE, col="gray50", lwd=2,xlim=range(-125,-100))
    box()
  } else{
    num.xgrids=(range(lon)[2]-range(lon)[1])/2
    num.ygrids=(range(lat)[2]-range(lat)[1])/2
    par(mfrow=c(3,1), mar=c(4,4,2,.5), mgp=c(1.5,.5,0)) #mfrow: rows and columns of plots, mar: margins on edges
of plots, mgp: axis label locations
```

```
   quilt.plot(nx=num.xgrids,ny=num.ygrids,lon, lat,ypred$fit,xlab="Longitude (deg)",ylab="Latitude
(deg)",main='Posterior Binary Precipitation on DEM Grid',zlim=range(ypred$fit,yob))
   grid(col="gray70",lty=2)
   US(add=TRUE, col="gray50", lwd=2,xlim=range(-125,-100))
   box()

   quilt.plot(lon1, lat1,yob,xlab="Longitude (deg)",ylab="Latitude (deg)",main='Observed Precipitation
(mm)',zlim=range(ypred$fit,yob))
   grid(col="gray70",lty=2)
   US(add=TRUE, col="gray50", lwd=2,xlim=range(-125,-100))
   box()

   quilt.plot(lon, lat,ypred$se,xlab="Longitude (deg)",ylab="Latitude (deg)",main='Standard Error on DEM Grid
(mm)')
   grid(col="gray70",lty=2)
   US(add=TRUE, col="gray50", lwd=2,xlim=range(-125,-100))
   box()
  }
}

prediction_ggplot=function(lon, lat, ypred, lon1, lat1, yob,type=""){
 xlim1=range(lon,lon1) # longitude range
 ylim1=range(lat,lat1) # latitude range
 zlim=range(ypred$fit,yob)
 df <- data.frame(ypred$fit, lon, lat) # dataframe for predictions
 names(df) <- c("Pmhat","Lon", "Lat")
 df.obs<-data.frame(yob, lon1,lat1)
 zlime=range(df$Pmhat)
 zlab=seq(10,60, by=10) # establish labels for z axis (Pmhat)
 siz1=14
 siz2=10
 MainStates <- map_data("state") # take data from the maps package, load into a df ready for plotting with
ggplot2
 states_full=c("arizona","colorado","new mexico","utah")

 par(mfrow=c(2,1))

 if(type=='binary'){
   predict_plot=ggplot() +
    geom_tile(data = df, aes(x = Lon, y = Lat, fill=Pmhat), show.legend = F)+ #draws precip prediction raster
    geom_polygon( data=MainStates, aes(x=long, y=lat, group=group), color="gray20", fill="transparent" )+ #draws
the states
    geom_point(data=df.obs, aes(x=lon1, y=lat1, color=yob),size=4) +
    geom_point(data=df.obs, aes(x=lon1, y=lat1, size=4), shape=1, colour='black') +
    ggtitle("Predicted vs Observed 3-Day Max Summer Precip") +
    xlab("Longitude") +
    ylab("Latitude")+
    scale_color_gradient(name='Prob > Q75', low='red', high='blue', space='Lab', limits=zlim )+ # Precip color
legend
    scale_fill_gradient(name="Pred Prec (mm)", low='red', high='blue', space='Lab', limits=zlim )+
    scale_size(name='Obs Prec', labels=NULL)+ # Observed Precip Legend
    theme_bw()+
    theme(legend.position="right")+
```

```
    theme(panel.grid.major = element_line(colour = "grey50", linetype = "dashed"),
          legend.text = element_text(size = 10),plot.margin=unit(c(0.0,0.01,0.01,0.01),"in"))+
    coord_sf(xlim = xlim1, ylim = ylim1,label_axes = list(bottom = "E", left = "N"), expand =FALSE)

  # plot standard error grid
  df.se=data.frame(ypred$se, lon,lat)
  colnames(df.se)=c('se','Lon','Lat')
  se_plot=ggplot() +
    geom_tile(data = df.se, aes(x = Lon, y = Lat, fill=se))+ #draws precip prediction raster
    geom_polygon( data=MainStates, aes(x=long, y=lat, group=group), color="gray20", fill="transparent" )+ #draws
the states
    ggtitle("Standard Error") +
    xlab("Longitude") +
    ylab("Latitude")+
    scale_fill_gradient(name="se", low='red', high='blue', space='Lab')+
    theme_bw()+
    theme(legend.position="right")+
    theme(panel.grid.major = element_line(colour = "grey50", linetype = "dashed"),
          legend.text = element_text(size = 10),plot.margin=unit(c(0.0,0.01,0.01,0.01),"in"))+
    coord_sf(xlim = xlim1, ylim = ylim1,label_axes = list(bottom = "E", left = "N"), expand =FALSE)

  return(list(predict_plot,se_plot))
 } else {
  # plot predictions and observations

  predict_plot=ggplot() +
    geom_tile(data = df, aes(x = Lon, y = Lat, fill=Pmhat), show.legend = F)+ #draws precip prediction raster
    geom_polygon( data=MainStates, aes(x=long, y=lat, group=group), color="gray20", fill="transparent" )+ #draws
the states
    geom_point(data=df.obs, aes(x=lon1, y=lat1, color=yob, size=yob)) +
    geom_point(data=df.obs, aes(x=lon1, y=lat1, size=yob), shape=1, colour='black') +
    ggtitle("Predicted vs Observed 3-Day Max Summer Precip") +
    xlab("Longitude") +
    ylab("Latitude")+
    scale_color_gradient(name='Prec (mm)', low='red', high='blue', space='Lab', limits=zlim )+ # Precip color legend
    scale_fill_gradient(name="Pred Prec (mm)", low='red', high='blue', space='Lab', limits=zlim )+
    scale_size(name='Obs Prec (mm)')+ # Precip size legend
    theme_bw()+
    theme(legend.position="right")+
    theme(panel.grid.major = element_line(colour = "grey50", linetype = "dashed"),
          legend.text = element_text(size = 10),plot.margin=unit(c(0.0,0.01,0.01,0.01),"in"))+
    coord_sf(xlim = xlim1, ylim = ylim1,label_axes = list(bottom = "E", left = "N"), expand =FALSE)

  # plot standard error grid
  df.se=data.frame(ypred$se, lon,lat)
  colnames(df.se)=c('se','Lon','Lat')
  se_plot=ggplot() +
    geom_tile(data = df.se, aes(x = Lon, y = Lat, fill=se))+ #draws precip prediction raster
    geom_polygon( data=MainStates, aes(x=long, y=lat, group=group), color="gray20", fill="transparent" )+ #draws
the states
    ggtitle("Standard Error") +
    xlab("Longitude") +
    ylab("Latitude")+
```

```
    scale_fill_gradient(name="se (mm)", low='red', high='blue', space='Lab')+
    theme_bw()+
    theme(legend.position="right")+
    theme(panel.grid.major = element_line(colour = "grey50", linetype = "dashed"),
          legend.text = element_text(size = 10),plot.margin=unit(c(0.0,0.01,0.01,0.01),"in"))+
    coord_sf(xlim = xlim1, ylim = ylim1,label_axes = list(bottom = "E", left = "N"), expand =FALSE)

  return(list(predict_plot,se_plot))

 }

}



# Drop-10% CV function
Drop_10_pred = function(X,bestTree,type) {
 mod_data = X
 N = length(X$prec)
 drop = 10
 nsample = 500
 i_full = 1:N
 # initialize skill score vectors
 skill_rmse = vector(mode="numeric", length=nsample)
 skill_cor = vector(mode="numeric", length=nsample)
 for (i in 1:nsample){
  if (type=="tree"){
   i_drop = sample(i_full,N*drop/100)          # can add argument replace=TRUE
   drop_dt = mod_data[-i_drop,] # drop 10% precip value
   myTree =tree(prec ~ PC1+PC2+PC3, data = drop_dt, model = T)
   drop_mod= prune.tree(myTree, best = bestTree)
   drop_pred=predict(drop_mod,newdata=mod_data[i_drop,])
   drop_actual = mod_data[i_drop,1]
   skill_rmse[i] = sqrt(mean((drop_actual - drop_pred)^2))
   skill_cor[i] = cor(drop_actual,drop_pred)
  }else{
   i_drop = sample(i_full,N*drop/100)          # can add argument replace=TRUE
   drop_dt = mod_data[-i_drop,] # drop 10% precip value
   drop_mod=randomForest(prec ~ PC1+PC2+PC3, data = drop_dt)
   drop_pred=predict(drop_mod,newdata=mod_data[i_drop,])
   drop_actual = mod_data[i_drop,1]
   skill_rmse[i] = sqrt(mean((drop_actual - drop_pred)^2))
   skill_cor[i] = cor(drop_actual,drop_pred)
  }

 }
 CV=as.data.frame(cbind(skill_rmse,skill_cor))
 names(CV)=c("rmse","cor")
 return(CV)
}
logit2prob = function(logit){
 odds <- exp(logit)
 prob <- odds / (1 + odds)
```

```
  return(prob)
}

cauchit2prob= function(cauchit){
  prob=(atan(cauchit)+pi/2)/pi
  return(prob)
}


#Function to prune the tree
treeFun = function(myTree, toPlot = F, title = ""){

  #Perform CV on tree object
  cvTree <- cv.tree(myTree)
  optTree <- which.min(cvTree$dev)
  bestTree <- cvTree$size[optTree]

  #prune Tree based on CV results
  pruneTree <- prune.tree(myTree, best = bestTree)

  #If plotting is selected
  if(toPlot){

    #Plot unpruned Tree
    plot(myTree)
    text(myTree, cex = .75)
    title(main = paste("Unpruned Tree for", title))

    #Plot CV
    plot(cvTree$size, cvTree$dev, type = "b",
         main = paste("Cross Validation for", title))

    #Plot Prunned Tree
    plot(pruneTree)
    text(pruneTree, cex = .75)
    title(main = paste("Pruned Tree for", title))
  }
  pruneTree$besTree=bestTree
  return(pruneTree)
}
#Extreme clustering function
pam_fmado_ll = function (x, k, ll) {

  # x - Design matrix, typically colums are stations, rows are time
  # k - Number of clusters
  # ll - two column matrix of lat long points (or preferably projected) with N rows

  N = ncol(x) # number of stations
  T = nrow(x) # number of time points

  # compute the F-madogram distance
  V = array(NaN, dim = c(T, N))
  for (p in 1:N) {
```

```
   x.vec = as.vector(x[, p]) # time series for one location
   Femp = ecdf(x.vec)(x.vec) # empirical probability of each observation
   V[, p] = Femp # place probability in matrix V
  }
  DD = dist(t(V), method = "manhattan", diag = TRUE, upper = TRUE)/(2 * T)

  # weight by physical distance
  DDll = dist(ll,method='manhattan')
  DDw = as.matrix(DD) + t(t(as.matrix(DDll))/apply(as.matrix(DDll),2,max))*max(as.matrix(DD))

  # do the clustering
  output = pam(DDw, k, diss = TRUE, medoids = NULL)
  return(output)
}
#multiplot function
multiplot <- function(..., plotlist=NULL, file, cols=1, layout=NULL) {
  library(grid)

  # Make a list from the ... arguments and plotlist
  plots <- c(list(...), plotlist)

  numPlots = length(plots)

  # If layout is NULL, then use 'cols' to determine layout
  if (is.null(layout)) {
    # Make the panel
    # ncol: Number of columns of plots
    # nrow: Number of rows needed, calculated from # of cols
    layout <- matrix(seq(1, cols * ceiling(numPlots/cols)),
                     ncol = cols, nrow = ceiling(numPlots/cols))
  }

  if (numPlots==1) {
    print(plots[[1]])

  } else {
    # Set up the page
    grid.newpage()
    pushViewport(viewport(layout = grid.layout(nrow(layout), ncol(layout))))

    # Make each plot, in the correct location
    for (i in 1:numPlots) {
      # Get the i,j matrix positions of the regions that contain this subplot
      matchidx <- as.data.frame(which(layout == i, arr.ind = TRUE))

      print(plots[[i]], vp = viewport(layout.pos.row = matchidx$row,
                                      layout.pos.col = matchidx$col))
    }
  }
}

################################### GLM and Local Poly ##########################
```

```
### Find best GLM
GLM_fit = function(Pm, X, family, month) {

 if (family == "gamma") {
  links = c("log", "inverse","identity")

  # clean data and remove zeros
  Pm = ifelse(Pm <=0, runif(1, 0.0001, 0.001), Pm)

 } else if (family == "binomial"){
  links = c("logit", "probit", "cauchit")
 } else if (family == "gaussian"){
  links = c("identity")
 }

 N = length(Pm)

 combs = leaps(X,Pm, nbest=25)    #  GEt upto 25 combinations for each
 # number of predictors
 combos = combs$which
 ncombos = length(combos[,1])
 glm_press=vector(length = length(links))
 best_comb=vector(length = length(links))
 xpress=1:ncombos
 xmse = 1:ncombos

 for(j in 1:length(links)) {
  aux_var=1 # allow to fit glm
  ##remove small values for gamma distribution
  if (family == "gamma"){
   if (links[j]=="identity"){
    if( month==1){
     Pm[Pm<=0.005]=0.005
    }else{
     Pm[Pm<=2.5]=2.5
    }

   }else if (links[j]=="inverse" & month==1){
    Pm[Pm<=25]=25 # it is necessary to modific significantly the small values,
    #so, "inverse" is not fitted for January
    aux_var=0
   }
  }
  if (aux_var==1)
  {for(i in 1:ncombos) {
   xx = X[,combos[i,]]
   xx=as.data.frame(xx)
   if (family == "gamma"){
    zz=glm(Pm ~ ., data=xx, family = Gamma(link=links[j]), maxit=500)
   }else if (family == "binomial"){
    zz=glm(Pm ~ ., data=xx, family = binomial(link=links[j]), maxit=500)
   }else if (family == "gaussian"){
    zz=glm(Pm ~ ., data=xx, family = gaussian(link=links[j]), maxit=500)
```

```
    }
    xpress[i]=PRESS(zz)
    xmse[i] = sum((zz$res)^2) / (N - length(zz$coef))
    #print(xpress[i])
   }}
   if (aux_var==1){
    # Test using PRESS objective function
    glm_press[j]=min(xpress)
    best_comb[j]=which.min(xpress)
   }else{
    # Test using PRESS objective function
    glm_press[j]=200000
    best_comb[j]=which.min(xpress)
   }
  }

  press_df = data.frame(glm_press)
  rownames(press_df) = links[1:length(links)]

  print("Results of PRESS for bestfit GLM")
  print(press_df)
  print(best_comb[which.min(glm_press)])

  sprintf("Choosing the GLM which minimizes PRESS: %s family and %s link function.", family,
links[which.min(glm_press)])
  xx = X[,combos[best_comb[which.min(glm_press)],]]
  xx=as.data.frame(xx)
  if (family == "gamma") {
    bestmod = glm(Pm ~ ., data = xx, family = Gamma(link=links[which.min(glm_press)]))
  } else if (family == "binomial") {
    bestmod = glm(Pm ~ ., data = xx, family = binomial(link=links[which.min(glm_press)]))
  }  else if (family == "gaussian") {
    bestmod = glm(Pm ~ ., data = xx, family = gaussian(link=links[which.min(glm_press)]))
  } else {
    print("Error!")
  }
  bestmod$Call$LINK=links[which.min(glm_press)]
  bestmod$Call$PRESS=PRESS(bestmod)
  bestmod$Call$combo=combos[best_comb[which.min(glm_press)],]
  return(bestmod)
}
##### Local Polynomail Fitting ######

#search for best alpha over a range of alpha values between 0 and 1

locpoly_fit = function(Pm, X, family="", link="", glm=FALSE,plot=FALSE) {


  nvar=length(X[1,])    #number of variables
  N=length(Pm) #number ofdata points
```

```
 if(glm==TRUE) {
  if (family == "gamma") {
   Pm = ifelse(Pm <=0, runif(1, 0.0001, 0.001), Pm)
  }
  # print("checking inputs")
  # print(family)
  # print(link)
  porder=1
  minalpha=0.6
  alpha_grid=seq(minalpha,1.0,by=0.05)
  n=length(alpha_grid)

  porder=2
  minalpha=0.6
  alpha1_grid=seq(minalpha,1.0,by=0.05)
  alpha=alpha2_grid=c(alpha_grid,alpha1_grid)

  #get the GCV values for all the alpha values in alpha for order of
  # polynomial = 1 and 2. kern="bisq" argument is to use the bisquare kernel
  # in computing the weights of the neighbors, which are then used in
  # the weighted least squares..
  gcv_deg1=gcvplot(Pm ~ ., data=X, maxk = 100000, alpha=alpha_grid,deg=1,kern="bisq",
ev=dat(),scale=TRUE,family=family,link=link)
  gcv_deg2=gcvplot(Pm ~ ., data=X, maxk = 100000,
alpha=alpha1_grid,deg=2,kern="bisq",ev=dat(),scale=TRUE,family=family,link=link)
  # pick the best alpha and the degree of the polynomial that
  # gives the least GCV
  z2=order(c(gcv_deg1$values,gcv_deg2$values))
  bestdeg=1
  if(z2[1] > n)bestdeg=2
  best_alpha = alpha2_grid[z2[1]]
  best_gcv = c(gcv_deg1$values,gcv_deg2$values)[z2[1]]
  output=c(bestdeg, best_alpha, best_gcv)    #the best parameter set

  # Now fit the LOCFIT model using the best alpha and degree obtained from above..
  if (plot==FALSE){
   bestmod=locfit(Pm ~., data=X, alpha=best_alpha, maxk = 10000, deg=bestdeg,kern="bisq"
            , scale = T, family=family, link=link,ev=dat())
  }else {
   bestmod=locfit(Pm ~., data=X, alpha=best_alpha, maxk = 10000, deg=bestdeg,kern="bisq"
            , scale = T, family=family, link=link)
  }

  bestmod$call$alpha = best_alpha
  bestmod$call$deg = bestdeg
  bestmod$call$family = family
  bestmod$call$link = link
  bestmod$call$gcv = best_gcv
 }
 else{
  porder=1
  minalpha=2*(nvar*porder+1)/N
  alpha_grid=seq(minalpha,1.0,by=0.05)
```

```
   n=length(alpha_grid)

   porder=2
   minalpha=2*(nvar*porder+1)/N
   alpha1_grid=seq(minalpha,1.0,by=0.05)
   alpha=alpha2_grid=c(alpha_grid,alpha1_grid)

   #get the GCV values for all the alpha values in alpha for order of
   # polynomial = 1 and 2. kern="bisq" argument is to use the bisquare kernel
   # in computing the weights of the neighbors, which are then used in
   # the weighted least squares..
   gcv_deg1=gcvplot(Pm ~ ., data=X, maxk = 100000, alpha=alpha_grid,deg=1,kern="bisq", ev=dat(),scale=TRUE)
   gcv_deg2=gcvplot(Pm ~ ., data=X, maxk = 100000, alpha=alpha1_grid,deg=2,kern="bisq",ev=dat(),scale=TRUE)
   # pick the best alpha and the degree of the polynomial that
   # gives the least GCV
   z2=order(c(gcv_deg1$values,gcv_deg2$values))
   bestdeg=1
   if(z2[1] > n)bestdeg=2
   best_alpha = alpha2_grid[z2[1]]
   best_gcv = c(gcv_deg1$values,gcv_deg2$values)[z2[1]]
   output=c(bestdeg, best_alpha, best_gcv)    #the best parameter set

   # Now fit the LOCFIT model using the best alpha and degree obtained from above..
   if (plot==FALSE){
     bestmod=locfit(Pm ~., data=X, alpha=best_alpha, maxk = 10000, deg=bestdeg,kern="bisq",ev=dat())
   }else{
     bestmod=locfit(Pm ~., data=X, alpha=best_alpha, maxk = 10000, deg=bestdeg,kern="bisq")
   }

   bestmod$call$alpha = best_alpha
   bestmod$call$deg = bestdeg
   bestmod$call$gcv = best_gcv

  }

  return(bestmod)

}
### F-test for Local Polynomial Fitting ###

loc_Ftest = function(Pm,Pmhat,X, bestmod) {

  N=length(Pm) #number ofdata points
  RSS1 = sum((Pm-Pmhat$fit)^2)

  nu1 = bestmod$dp[6] # trace(L) [lk]
  nu2 = bestmod$dp[7] # trace(L^T L) [df1]

  nu11 = N-2*nu1 + nu2

  #linear regression ###
  # #linear regression
  X=as.matrix(X)
```

```
  zzLin=lm(Pm~X)

  XX = cbind(rep(1,N), X)
  # Compute the Hat matrix
  hatm = XX %*% solve(t(XX) %*% XX) %*% t(XX)

  II = diag(N)
  delta0 = t(II-hatm)%*%(II-hatm)    #Equation 9.2
  nu00 = sum(diag(delta0))

  RSS0 = sum(residuals(zzLin)^2)


  Fdata = (RSS0 - RSS1)/(nu00 - nu11)
  Fdata = (Fdata / (RSS1 / nu11))
  Ftheor = qf(0.95,(nu00-nu11), nu11)    #95% confidence level..

  ## Fdata > Ftheor   - reject null - i.e., data is otherwise (local polynomial)
  if (Fdata > Ftheor) {
    print("F-test:")
    sprintf("Reject the Null because F(local poly) = %0.2f > %0.2f = F(linear model).", Fdata, Ftheor)

    #### The null hypothesis is that the linear model best fits the data. Rejecting the null means
    # the local poly is significantly better than linear.

  }
}



############################ PCA function with scree plot ############################

# all code from Alvaro, Nathan just turned it into a function

pca=function(data, scale=T){

  if(scale==T){
    scaled_data=scale(data)
  } else {
    scaled_data=data
  }

  covariance_mat=var(scaled_data) # covariance matrix
  zsvd=svd(covariance_mat) # SVD results in three matrices. u, eigen value diagonal (lamdas), V. u is the matrix of
loading scores, AKA phi, AKA eigen vector matrix
  pcs=scaled_data %*% zsvd$u # Z = X times u matrix
  lambdas = zsvd$d/sum(zsvd$d) # d is variance in each PC. Lambda is proportion of variance explained

  scree=ggplot() +
    geom_line(mapping = aes(c(1:25), lambdas[1:25])) +
    geom_point(mapping = aes(c(1:25), lambdas[1:25]), shape = 21, size = 3, color = "gray30", fill ="cadetblue1") +
    labs(title = "Eigen Spectrum",x = "Modes",y = "Frac. Var. explained")+
    theme(plot.title = element_text(size = 13, face = "bold", hjust = 0.5))
```

```
  pca_results=list()
  pca_results[['pcs']]=pcs
  pca_results[['u']]=zsvd$u
  pca_results[['PVE']]=lambdas
  pca_results[['scree_plot']]=scree


  return(pca_results)
}


################################### Read SST Data ##############################


readSST=function(){
  ### Lat - Long grid..
  # information obtained from: http://iridl.ldeo.columbia.edu/SOURCES/.KAPLAN/.EXTENDED/.v2/

  ygrid=seq(-87.5,87.5,by=5)
  ny=length(ygrid) # number of latitudinal locations

  xgrid=seq(27.5,382.5,by=5)
  nx=length(xgrid) # number of longitudinal locations

  nglobe = nx*ny
  # creation of spatial grid
  xygrid=matrix(0,nrow=nx*ny,ncol=2)
  i=0
  for(iy in 1:ny){
    for(ix in 1:nx){
      i=i+1
      xygrid[i,1]=ygrid[iy]
      xygrid[i,2]=xgrid[ix]
    }
  }


  ################## Read Gridded monthly global sea surface temp (SST) anomalies (AKA Kaplan SST)
  #

  nyrs = 118    #Nov-Mar 1901 - Nov-Mar 2018
  data=readBin(con='http://civil.colorado.edu/~balajir/CVEN6833/HWs/HW-2/Kaplan-SST-DJFM1900-
DJFM2018.r4', what='numeric', n=(nx*ny*nyrs),
               size=4, endian = 'swap')

  data = array(data = data, dim=c( nx, ny, nyrs ) )
  data1=data[,,1]


  # the lat -long data grid..

  index=1:(nx*ny)
```

```
 index1=index[data1 < 20 & data1 != "NaN"]   # only non-missing data.
 # what is less than 20 for?
 xygrid1=xygrid[index1,] # these are the locations of remaining data, corresponding to the cols sstannavg

 nsites=length(index1)
 data2=data1[index1]

 ### SSTdata matrix - rows are seasonal (i.e. one value per year)
 ## and columns are locations
 sstannavg=matrix(NA,nrow=nyrs, ncol=nsites)

 for(i in 1:nyrs){
  data1=data[,,i]
  index1=index[data1 < 20 & data1 != "NaN"]
  data2=data1[index1]
  sstannavg[i,]=data2
 }

 indexgrid = index1
 rm("data")  #remove the object data to clear up space

 lon=sort(unique(xygrid[,2]))
 lat = sort(unique(xygrid[,1]))

 return(list(sst=sstannavg, nglobe=nglobe, indexgrid=indexgrid, lon=lon,lat=lat))

}

############ leave one out cross validation for canonical correlation analysis model ######

cca_loocv=function(y, x, y_pc, x_pc, y_loading){

 nyear=nrow(y)

 loocv=matrix(NA,nrow=nrow(y), ncol=length(y))
 loocv_corr=rep(NA, nrow(y))
 loocv_RMSE=rep(NA, nrow(y))
 for (i in 1:nyear){

  drop_X=x_pc[-i,]
  drop_Y=y_pc[-i,]

  keep_X=x_pc[i,]
  keep_Y=y_pc[i,]

  M=dim(drop_X)[2]
  J=dim(drop_Y)[2]
  J=min(M,J)
  N = length(drop_Y[,1])
  Qx1 = qr.Q(qr(drop_X))
  Qy1 = qr.Q(qr(drop_X))
  T11 = qr.R(qr(drop_X))
```

```
  T22 = qr.R(qr(drop_Y))
  VV=t(Qx1) %*% Qy1
  BB = solve(T22) %*% svd(VV)$v * sqrt(N-1)
  wm1 = drop_Y %*% BB
  AA = solve(T11) %*% svd(VV)$u * sqrt(N-1)
  vm1 = drop_X %*% AA

  ### Predict the winter precipitation PCS
  betahat = solve(t(AA) %*% t(drop_X)%*% drop_X %*% AA) %*% t(AA) %*% t(drop_X) %*% drop_Y
  ypred=keep_X %*% AA %*% betahat
  ### first npc PCs from the PC forecast above and the remaining PCs are set to
  ## their means - i.e., 0

  N1 = dim(y)[2]-(npc)

  PCpred = c(ypred,rep(0,N1)) # Z matrix, which is X*E

  ## back transform to get the winter precipitation field

  ###  Keep only the first npc Eigen Vectors and set rest to zero
  E = matrix(0,nrow=dim(y)[2],ncol=dim(y)[2])
  E[,1:npc]=y_loading[,1:npc] # E matrix, ie loading or eigenvectors

  ## back transform to get the winter precipitation field
  ypred =  PCpred %*% t(E)

  ######## rescale winter precip

  s=scale(y)
  scale_vals=attributes(s)
  center=scale_vals$`scaled:center`
  stdev=scale_vals$`scaled:scale`

  loocv[i,]=t(t(ypred)*stdev +center)
  loocv_corr[i]=cor(loocv[i,], as.numeric((y[i,])))
  errors=loocv[i,]-y[i,]
  loocv_RMSE[i]=sqrt(sum(errors^2)/length(errors))
 }
 results=list(corr=loocv_corr, rmse=loocv_RMSE)
 return(results)
}


############### Fitting CART to 1:npc PCs using other PCs as predictors ############

CART_PCs=function(y_pc, x_pc, npc){
 tree.list=list() # preallocate a list

 for (i in 1:npc){ # model first npc PCs
  df=data.frame(y_pc=y_pc[,i], x_pc[,1:npc])
  tree.unpruned=tree(y_pc ~ ., data = df, model =T) # need model = T for cv.tree to have dataframe

  #Perform CV on tree object
```

```
  cvTree <- cv.tree(tree.unpruned)
  optTree <- which.min(cvTree$dev)
  bestTree <- cvTree$size[optTree]

  if (bestTree ==1){
    bestTree = 2
  }

  #prune Tree based on CV results
  pruneTree <- prune.tree(tree.unpruned, best = bestTree)

  tree.list[[paste('PC',i, sep='')]]=pruneTree

 }
 return(tree.list)

}

############### drop 10 analysis for CART ##################

CART_drop10=function(y, x, y_pc, x_pc, y_loading, npc,iter=500){

 nyear=nrow(y)

 loocv_corr=rep(NA, nrow(y))
 loocv_RMSE=rep(NA, nrow(y))

 cor_vec=1
 rmse_vec=1

 index=1:nyear

 for (i in 1:iter){

  drop_index=sample(index, size = round(0.1*nyear))

  drop_X=x_pc[drop_index,]
  drop_Y=y_pc[drop_index,]

  keep_X=x_pc[-drop_index,]
  keep_Y=y_pc[-drop_index,]

 ##### Tree fitting #################
  tree.list=CART_PCs(y_pc=keep_Y, x_pc=keep_X, npc=npc)

  Zmat=matrix(0, nrow=round(0.1*nyear), ncol=ncol(y))

  for(p in 1:npc){
    Zmat[,p]=predict(tree.list[[p]],newdata=data.frame(drop_X))
  }

  ###  Keep only the first npc Eigen Vectors and set rest to zero
  E = matrix(0,nrow=dim(y)[2],ncol=dim(y)[2])
```

```
  E[,1:npc]=y_loading[,1:npc] # E matrix, ie loading or eigenvectors


  scaled_pred=Zmat %*% t(E)

  s=scale(y[drop_index,])
  scale_vals=attributes(s)
  center=scale_vals$`scaled:center`
  stdev=scale_vals$`scaled:scale`

  ypred=scaled_pred*stdev+center

  yobs=y[drop_index,]
  errors=ypred-yobs

  for (c in 1:length(drop_index)){
    cor_add=cor(ypred[c,],as.numeric(yobs[c,]))
    cor_vec=c(cor_vec, cor_add)

    rmse_add=sqrt(sum(errors[c,]^2)/ncol(errors))
    rmse_vec=c(rmse_vec, rmse_add)
  }

 }

 cor_vec=cor_vec[-1]
 rmse_vec=rmse_vec[-1]

 results=list(corr=cor_vec, rmse=rmse_vec)
 return(results)
}

############################ Random Forest PC Fitting ################

RF_PCs=function(y_pc, x_pc, npc){

 predPC=matrix(0, nrow=nrow(y_pc), ncol=ncol(y_pc))

 for (i in 1:npc){ # model first npc PCs
  df=data.frame(y_pc=y_pc[,i], x_pc[,1:npc])
  rf=randomForest(y_pc ~., data = df)
  predPC[,i]=predict(rf)

 }
 return(predPC)

}

############################## Random Forest drop 10 ###################


RF_drop10=function(y, x, y_pc, x_pc, y_loading, npc,iter=500){
```

```
nyear=nrow(y)

loocv_corr=rep(NA, nrow(y))
loocv_RMSE=rep(NA, nrow(y))

cor_vec=1
rmse_vec=1

index=1:nyear

for (i in 1:iter){

  drop_index=sample(index, size = round(0.1*nyear))

  drop_X=x_pc[drop_index,]
  drop_Y=y_pc[drop_index,]

  keep_X=x_pc[-drop_index,]
  keep_Y=y_pc[-drop_index,]

  ##### Tree fitting ################

  Zmat=matrix(0, nrow=round(0.1*nyear), ncol=ncol(y))

  for (i in 1:npc){ # model first npc PCs
    df=data.frame(y_pc=y_pc[,i], x_pc[,1:npc])
    rf=randomForest(y_pc ~., data = df)
    Zmat[,i]=predict(rf, newdata=data.frame(drop_X))

  }


  ###  Keep only the first npc Eigen Vectors and set rest to zero
  E = matrix(0,nrow=dim(y)[2],ncol=dim(y)[2])
  E[,1:npc]=y_loading[,1:npc] # E matrix, ie loading or eigenvectors


  scaled_pred=Zmat %*% t(E)

  s=scale(y[drop_index,])
  scale_vals=attributes(s)
  center=scale_vals$`scaled:center`
  stdev=scale_vals$`scaled:scale`

  ypred=scaled_pred*stdev+center

  yobs=y[drop_index,]
  errors=ypred-yobs

  for (c in 1:length(drop_index)){
    cor_add=cor(ypred[c,],as.numeric(yobs[c,]))
    cor_vec=c(cor_vec, cor_add)
```

```
    rmse_add=sqrt(sum(errors[c,]^2)/ncol(errors))
    rmse_vec=c(rmse_vec, rmse_add)
  }

 }

 cor_vec=cor_vec[-1]
 rmse_vec=rmse_vec[-1]

 results=list(corr=cor_vec, rmse=rmse_vec)
 return(results)
}
```

P1: Bayesian spatial hierarchical model

```
################# P1: Bayesian Heirarchical ###################

rm(list=ls())


## Source libraries and suppress loading messages

# Note: source HW2 library also imports datasets

setwd("G:/My Drive/CU Boulder/Classes/20-21/CVEN 6833 Advanced Stats/HW/HW2")
suppressPackageStartupMessages(source('R Scripts/HW2 Library.R'))
source('R Scripts/HW2 Library.R')

############### Fit GLM with Lat, Lon, Elev ##########################

glmfit=glm(Pm ~ ., data=X, family= Gamma(link='log'))
summary(glmfit)

# Note that Elevation is statistically insignificant

############### Fit variogram and perform MCMC ########################

# obtain initial estimate of effective range (phi)

geod= as.geodata(cbind(glmfit$residuals, X$Lon, X$Lat), data.col = 1) # as.geodata() forces a dataframe or matrix
into a geodata object
 # data.col states which column is the data (here it is the residuals)
  # this step is needed to use variog() command

vg=variog(geod) # computes variogram using bins defined by a sequence of breaks
plot(vg)

################## IMPORTANT ####################
######## semivariance variables are defined differently in spBayes and fields packages!

# in spBayes, tau.sq is the nugget
tau.sq=min(vg$v) #vg$v is a vector of semivariogram values at distances given in vg$u
tau.sq.lo=0.6*tau.sq
tau.sq.hi=tau.sq

# in spBayes, sigma.sq is effective sill (sill minue nugget)
sigma.sq=quantile(vg$v,.9) - tau.sq # subtract nugget for effective sill
sigma.sq.lo=.8*sigma.sq
sigma.sq.hi=1.2*sigma.sq

# phi is effective range
phi.val=median(vg$u) # phi is effective range (distance at which semivariance flattens out)
phi.lo=quantile(vg$u, 0.25)
phi.hi=quantile(vg$u, 0.75)
```

```
abline(h=sigma.sq+tau.sq, col='red')
abline(h=tau.sq, col='blue')
abline(v=c(phi.lo,phi.hi), col='green')
legend('topleft', legend=c('Sill estimate', 'Nugget estimate', 'Effective range range' ), lty=c(1,1,1),
col=c('red','blue','green'), cex=.7)

# Input number of samples
n.samples=1500
# coordinates of observations
coords=cbind(X$Lon, X$Lat)

# prior distributions defined (limited distributions available: see help (spLM))
help(spLM)
  #The function spLM fits Gaussian univariate Bayesian spatial regression models.
  # Given a set of knots, spLM will also fit a predictive process model (see references below).

priors=list('beta.flat', 'phi.unif'=c(phi.lo, phi.hi),
          'sigma.sq.ig'=c(2,1), 'tau.sq.ig'=c(2,1))
  #beta.flat: a flat prior is also known as an uninformative prior. Essentially uniform from negative to positive
infinity (pdf at any point near 0)
  #sigma.sq.ig: Inverse Gamma
  #tau.sq.ig: Inverse Gamma

# priors is an input into spLM(): a list with each tag corresponding to a parameter name. Valid tags are sigma.sq.ig,
# tau.sq.ig, phi.unif, nu.unif, beta.norm, and beta.flat.

# starting values for Markov Chain Monte Carlo (MCMC)

starting = list("phi"=phi.val, "sigma.sq"=sigma.sq, "tau.sq"=tau.sq)
# adjustment factor for MCMC sampling routine
tuning = list("phi"=1, "sigma.sq"=1,"tau.sq"=1)

# Perform Monte Carlo Markov Chain Analysis
bat.fm = spLM(Pm~., data=X, coords=as.matrix(X[,1:2]), priors=priors, tuning=tuning, starting=starting,
           cov.model = "exponential", n.samples = n.samples, verbose = FALSE, n.report = 50)

# add small amount to duplicated coordinates (there are no duplicates in this data)
dup = duplicated(bat.fm$coords); bat.fm$coords[dup] <- bat.fm$coords[dup] + 1e-3

# burn-in samples
bat.fm = spRecover(bat.fm, start=((1/3)*n.samples)+1, thin=1, verbose=T)
  #spRecover: function for recovering regression coefficients and spatial
  # random effects from spLM using composition sampling
   # what is composition sampling?

# plot posterior pdfs
beta = bat.fm$p.beta.recover.samples
theta = bat.fm$p.theta.recover.samples
plot(beta)

plot(theta)

summary(window(bat.fm$p.beta.recover.samples))
```

```
#########################################################################
## II. Spatially map the model estimates and the standard error ####
#########################################################################

####### obtain model predictions

y = matrix(nrow=length(x1[,1]), ncol=nrow(beta))
yse = matrix(nrow=length(x1[,1]), ncol=nrow(beta))

################ important!!!! #############
# semivariogram params denoted differently in fields package than spBayes
# spBayes = fields
# tau.sq = sigma2
# sigma.sq = rho
# phi = 1/theta


# loop through every vector of Beta and Theta in the posterior distribution
for (i in 1:nrow(beta)){ # this takes 2-4 minutes
  zz = Krig(X[,1:2],glmfit$residuals,rho=theta[i,1],theta=1/theta[i,3],sigma2=theta[i,2], m=1) # Krig using ith theta
and Beta on GLM residuals
  # recall that rho is sill (sigma.sq) , theta is range (1/phi), tau.sq is nugget
  y2 = predict.Krig(zz,x=x1[,1:2],drop.Z=TRUE) # predict residuals
  yse[,i] = predictSE(zz, x=x1[,1:2], drop.Z=TRUE) # predict standard error
  y1 = beta[i,1] + beta[i,2]*x1[,1]+beta[i,3]*x1[,2]+beta[i,4]*x1[,3] # mean function
  y[,i] = y1+y2 # mean function plus Krig
}
mean.y = apply(y, 1, FUN = mean) # take the mean and convert from percent to fraction
mean.yse = apply(yse, 1, FUN = mean) # take the mean and convert from percent to fraction
ypred=data.frame(fit=mean.y, se=mean.yse)
ggplot() +
  geom_histogram(aes(as.vector(y)),fill="gray50",col='black') +
  labs(title = "Histogram of Precipitation Posterior",
       x = "Predictions")+
  #theme_light()+
  theme(plot.title = element_text(size = 13, face = "bold", hjust = 0.5))

################ Plotting

a=prediction_ggplot(lon=x1$Lon,lat=x1$Lat,ypred,lon1=X$Lon,lat1=X$Lat,Pm) #Quilt_plotting=function(lon, lat,
ypred, lon1, lat1, yob,type="")
a
```

P2: Logistic regression

```
########################## Logistic Regression ##############################

rm(list=ls())

## Source libraries and suppress loading messages
# Note: source HW2 library also imports datasets
setwd("G:/My Drive/CU Boulder/Classes/20-21/CVEN 6833 Advanced Stats/HW/HW2")
suppressPackageStartupMessages(source('R Scripts/HW2 Library.R'))
source('R Scripts/HW2 Library.R')

############### compute 75th percentile and classify as binary ############################

q75=quantile(Pm, 0.75) # calculate 75th percentile

Pm_bin=1:length(Pm) # preallocate
Pm_bin=ifelse(Pm>q75, 1,0) # compute binary. If Pm > q75, set = 1, else 0
Pm_bin[which(Pm_bin <= 0)]=runif(n=1, min=.0001, max=0.001)
q75
check=data.frame(Pm, Pm_bin)

############## fit GLM (logistic regression) ################

log_mod=GLM_fit(Pm_bin,X, family = 'binomial')
# note that link functions for bimomial are logit, probit, and cauchit
  # for a nice figure of how these link functions differ, see https://arxiv.org/pdf/1502.04742.pdf fig 1

summary(log_mod)
anova(log_mod)

ypred=predict.glm(log_mod, newdata = x1, type='response', se.fit = T)

a=prediction_ggplot(lon=x1$Lon,lat=x1$Lat,ypred,lon1=X$Lon,lat1=X$Lat,Pm_bin, type='binary')
#Quilt_plotting=function(lon, lat, ypred, lon1, lat1, yob,type="")
a

################### Local Polynomial ###########


local_mod=locpoly_fit(Pm_bin,X,family='binomial', link='cauchit', glm=T, plot=T)
summary(local_mod)
Pmhat=predict(local_mod, newdata = X, type='response', se.fit=T)
loc_Ftest(Pm_bin, Pmhat, X, local_mod)

ypred_local=predict(local_mod, newdata = x1, type='response', se.fit=T)

b=prediction_ggplot(lon=x1$Lon,lat=x1$Lat,ypred_local,lon1=X$Lon,lat1=X$Lat,Pm_bin, type='binary')
#Quilt_plotting=function(lon, lat, ypred, lon1, lat1, yob,type="")
b
```

P3: Bayesian spatial logistic regression

```
rm(list=ls())

## Source libraries and suppress loading messages

# Note: source HW2 library also imports datasets

setwd("G:/My Drive/CU Boulder/Classes/20-21/CVEN 6833 Advanced Stats/HW/HW2")
suppressPackageStartupMessages(source('R Scripts/HW2 Library.R'))
source('R Scripts/HW2 Library.R')

# Run next two lines if you want to analyze winter max precip, not summer. Summer is read by library file

# Winter_Precip=read.table('http://civil.colorado.edu/~balajir/CVEN6833/HWs/HW-
2/data/Max_Winter_Seas_Prec.txt', header=T)
#
# Pm=apply(Winter_Precip,2,FUN=median)[-1]



######################## Find Q75 and create binary data ##################

q75=quantile(Pm, 0.75) # calculate 75th percentile

Pm_bin=1:length(Pm) # preallocate
Pm_bin=ifelse(Pm>q75, 1,0) # compute binary. If Pm > q75, set = 1, else 0
Pm_bin[which(Pm_bin <= 0)]=runif(n=1, min=.0001, max=0.001)
q75
check=data.frame(Pm, Pm_bin)

################### Binomial regression (first level of hierarchy) ################

log_mod=glm(Pm_bin ~ ., X, family=binomial(link='logit'))

ypred_glm=predict.glm(log_mod, newdata=x0, type='response', se=T)

summary(log_mod)
anova(log_mod)
  # the best model uses cauchit link, alpha=.6, and lat,lon and elev as predictors

################

# obtain initial estimate of effective range (phi)

geod= as.geodata(cbind(log_mod$residuals, X$Lon, X$Lat), data.col = 1) # as.geodata() forces a dataframe or
matrix into a geodata object
# data.col states which column is the data (here it is the residuals)
# this step is needed to use variog() command

vg=variog(geod) # computes variogram using bins defined by a sequence of breaks
plot(vg)
```

```
################## IMPORTANT ####################
######## semivariance variables are defined differently in spBayes and fields packages!

# in spBayes, tau.sq is the nugget
tau.sq=min(vg$v) #vg$v is a vector of semivariogram values at distances given in vg$u
tau.sq.lo=0.6*tau.sq
tau.sq.hi=tau.sq

# in spBayes, sigma.sq is effective sill (sill minue nugget)
sigma.sq=quantile(vg$v,.9) - tau.sq # subtract nugget for effective sill
sigma.sq.lo=.8*sigma.sq
sigma.sq.hi=1.2*sigma.sq

# phi is effective range
phi.val=median(vg$u) # phi is effective range (distance at which semivariance flattens out)
phi.lo=quantile(vg$u, 0.25)
phi.hi=quantile(vg$u, 0.75)


abline(h=sigma.sq+tau.sq, col='red')
abline(h=tau.sq, col='blue')
abline(v=c(phi.lo,phi.hi), col='green')
legend('topleft', legend=c('Sill estimate', 'Nugget estimate', 'Effective range range' ), lty=c(1,1,1),
col=c('red','blue','green'), cex=.7)


# Input number of samples
n.samples=1500
# coordinates of observations
coords=cbind(X$Lon, X$Lat)

# prior distributions defined (limited distributions available: see help (spLM))
help(spLM)
#The function spLM fits Gaussian univariate Bayesian spatial regression models.
# Given a set of knots, spLM will also fit a predictive process model (see references below).

priors=list('beta.norm', 'phi.unif'=c(phi.lo, phi.hi), # try beta norm!
        'sigma.sq.ig'=c(2,1), 'tau.sq.ig'=c(2,1))
#beta.flat: a flat prior is also known as an uninformative prior. Essentially uniform from negative to positive infinity
(pdf at any point near 0)
#sigma.sq.ig: Inverse Gamma
#tau.sq.ig: Inverse Gamma

# priors is an input into spLM(): a list with each tag corresponding to a parameter name. Valid tags are sigma.sq.ig,
# tau.sq.ig, phi.unif, nu.unif, beta.norm, and beta.flat.

# starting values for Markov Chain Monte Carlo (MCMC)

starting = list("phi"=phi.val, "sigma.sq"=sigma.sq, "tau.sq"=tau.sq, "beta"=log_mod$coefficients)
# adjustment factor for MCMC sampling routine
tuning = list("phi"=1, "sigma.sq"=1,"tau.sq"=1)
```

```
# Perform Monte Carlo Markov Chain Analysis
bat.fm = spLM(Pm_bin~., data=X, coords=as.matrix(X[,1:2]), priors=priors, tuning=tuning, starting=starting,
          cov.model = "exponential", n.samples = n.samples, verbose = FALSE, n.report = 50)

# add small amount to duplicated coordinates (there are no duplicates in this data)
dup = duplicated(bat.fm$coords); bat.fm$coords[dup] <- bat.fm$coords[dup] + 1e-3

# burn-in samples
bat.fm = spRecover(bat.fm, start=((1/3)*n.samples)+1, thin=1, verbose=T)
#spRecover: function for recovering regression coefficients and spatial
# random effects from spLM using composition sampling
# what is composition sampling?

# plot posterior pdfs
beta = bat.fm$p.beta.recover.samples
theta = bat.fm$p.theta.recover.samples
plot(beta)

plot(theta)

summary(window(bat.fm$p.beta.recover.samples))

######################################################################
## II. Spatially map the model estimates and the standard error ####
######################################################################


####### obtain model predictions

n.samples2 = floor(n.samples*(2/3))
ylog = matrix(nrow=length(x1[,1]), ncol=n.samples2)
yselog = matrix(nrow=length(x1[,1]), ncol=n.samples2)

################ important!!!! #############
# semivariogram params denoted differently in fields package than spBayes
# spBayes = fields
# tau.sq = sigma2
# sigma.sq = rho
# phi = 1/theta


# loop through every vector of Beta and Theta in the posterior distribution
for (i in 1:n.samples2){
  zz = Krig(X[,1:2],log_mod$residuals,rho=theta[i,1],theta=1/theta[i,3],m=1,sigma2=theta[i,2])
  y2 = predict.Krig(zz,x=x1[,1:2],drop.Z=TRUE)
  yselog[,i] = predictSE(zz, x=x1[,1:2], drop.Z=TRUE)
  y1 = beta[i,1] + beta[i,2]*x1[,1]+beta[i,3]*x1[,2]+beta[i,4]*x1[,3]
  ylog[,i] = y1+y2
}
y=logit2prob(ylog) #to obtain the real value
yse=logit2prob(yselog) #to obtain the real value
mean.y = apply(y, 1, FUN = mean) # take the mean and convert from percent to fraction
mean.yse = apply(yse, 1, FUN = mean) # take the mean and convert from percent to fraction
```

```
ypred=data.frame(fit=mean.y, se=mean.yse)

ggplot() +
  geom_histogram(aes(as.vector(y)),fill="gray50",col='black') +
  labs(title = "Histogram of Precipitation Posterior",
       x = "Predictions")+
  #theme_light()+
  theme(plot.title = element_text(size = 13, face = "bold", hjust = 0.5))

# mean of posterior
a=prediction_ggplot(lon=x1$Lon,lat=x1$Lat,ypred,lon1=X$Lon,lat1=X$Lat,Pm_bin, type='binary')
#Quilt_plotting=function(lon, lat, ypred, lon1, lat1, yob,type="")
a

# median of posterior

median.y=apply(y,1,FUN=median)
median.yse=apply(yse,1,FUN=median)
ypred_median=data.frame(fit=median.y, se=median.yse)


####### model coefficient estimates ##############

mean.beta=apply(beta, 2, FUN=mean)
median.beta=apply(beta, 2, FUN=median)

n.samples2 = floor(n.samples*(2/3))
ylog = 1:n.samples2
yselog =1:n.samples2

mean.theta=apply(theta, 2, FUN=mean)
mean.beta=apply(beta, 2, FUN=mean)

zz = Krig(X[,1:2],log_mod$residuals,rho=mean.theta[1],theta=1/mean.theta[3],m=1,sigma2=mean.theta[2])
y2 = predict.Krig(zz,x=x1[,1:2],drop.Z=TRUE)
yselog= predictSE(zz, x=x1[,1:2], drop.Z=TRUE)
y1 = mean.beta[1] + mean.beta[2]*x1[,1]+mean.beta[3]*x1[,2]+mean.beta[4]*x1[,3]
ylog = y1+y2

y=logit2prob(ylog) #to obtain the real value
yse=logit2prob(yselog) #to obtain the real value

ypred_mean_params=data.frame(fit=y, se=yse)
```

P4: Space-time PCA and correlation analysis

```
rm(list=ls())

## Source libraries and suppress loading messages

# Note: source HW2 library also imports datasets

setwd("G:/My Drive/CU Boulder/Classes/20-21/CVEN 6833 Advanced Stats/HW/HW2")
suppressPackageStartupMessages(source('R Scripts/HW2 Library.R'))
source('R Scripts/HW2 Library.R')

# unload packages that create a conflict with map("world2",add=T)
detach(package:mclust)
detach(package:sm)
detach(package:kohonen)

### Lat - Long grid..
# information obtained from: http://iridl.ldeo.columbia.edu/SOURCES/.KAPLAN/.EXTENDED/.v2/

ygrid=seq(-87.5,87.5,by=5)
ny=length(ygrid) # number of latitudinal locations

xgrid=seq(27.5,382.5,by=5)
nx=length(xgrid) # number of longitudinal locations

nglobe = nx*ny
# creation of spatial grid
xygrid=matrix(0,nrow=nx*ny,ncol=2)
i=0
for(iy in 1:ny){
  for(ix in 1:nx){
    i=i+1
    xygrid[i,1]=ygrid[iy]
    xygrid[i,2]=xgrid[ix]
  }
}


################### Read Gridded monthly global sea surface temp (SST) anomalies (AKA Kaplan SST)
#

nyrs = 118    #Nov-Mar 1901 - Nov-Mar 2018
data=readBin(con='http://civil.colorado.edu/~balajir/CVEN6833/HWs/HW-2/Kaplan-SST-DJFM1900-DJFM2018.r4',
what='numeric', n=(nx*ny*nyrs),
          size=4, endian = 'swap')

data = array(data = data, dim=c( nx, ny, nyrs ) )
data1=data[,,1]


# the lat -long data grid..
```

```
index=1:(nx*ny)

index1=index[data1 < 20 & data1 != "NaN"]   # only non-missing data.
  # what is less than 20 for?
xygrid1=xygrid[index1,] # these are the locations of remaining data, corresponding to the cols sstannavg

nsites=length(index1)
data2=data1[index1]

### SSTdata matrix - rows are seasonal (i.e. one value per year)
## and columns are locations
sstannavg=matrix(NA,nrow=nyrs, ncol=nsites)

for(i in 1:nyrs){
  data1=data[,,i]
  index1=index[data1 < 20 & data1 != "NaN"]
  data2=data1[index1]
  sstannavg[i,]=data2
}

indexgrid = index1
rm("data")  #remove the object data to clear up space

## write out the grid locations..
write(t(xygrid1),file="kaplan-sst-locs.txt",ncol=2) # this text file is also available from class website, HW2 folder


############# i: PCA on summer global SST anomalies ##############

# see HW2 library for pca() function
pca_results=pca(data=sstannavg, scale=T)
# plot eigen variance spectrum for first 25 modes
pca_results$scree_plot # return scree plot, AKA eigen spectrum

knee=4 # from visual inspection of scree plot
(PVE=sum(pca_results$PVE[1:knee]))

# plot leading 4 spatial (eigen vectors) and temporal (PCs) modes of variability

xlong = sort(unique(xygrid[,2]))
ylat = sort(unique(xygrid[,1]))

par(mfrow = c(4, 2))
par(mar = c(3, 4, 2, 1))
for(i in 1:4){
  zfull = rep(NaN,nglobe)   #also equal 72*36
  zfull[indexgrid]=pca_results$u[,i] # loading scores AKA phi AKA eigen vector
  zmat = matrix(zfull,nrow=nx,ncol=ny)
  image.plot(xlong,ylat,zmat,ylim=range(-40,70),ann=FALSE)
  mtext(paste("EOF no.",i,sep=""), side = 3, line = 0.2, cex = 0.8)
  contour(xlong,ylat,(zmat),ylim=range(-40,70),add=TRUE,nlev=6,lwd=2)
  map("world2",add=T)
```

```
  box()

  plot(1901:2018, scale(pca_results$pcs[,i]),type="l",xlab="Year",axes=FALSE,ann=FALSE)
  axis(2)
  axis(1)
  mtext(paste("PC no.",i,sep=""), side = 3, line = 0.2, cex = 0.8)
}

# the left panel of plots shows the loading score (u) for every lat-long location, for PC 1-4
# the right panel shows the principal component (Z) value (pca_results$pcs[,i], where i the PC number) as a
function of time
 # rows in the pc are years, columns are Z1, Z2, ... Zp


############## ii:  Perform a rotated PCA (rotate the first 6 PCs) and plot the leading 4 spatial and
# temporal modes of variability. Compare the results with (i) above.

zrot = varimax(pca_results$u[,1:6],normalize=FALSE) #varimax rotates loading matrices given loadings (u)
rotpcs=sstannavg %*% zrot$loadings
xlong = sort(unique(xygrid[,2]))
ylat = sort(unique(xygrid[,1]))
par(mfrow = c(4, 2))
par(mar = c(3, 4, 2, 1))

for(i in 1:4){
  zfull = rep(NaN,nglobe)   #also equal 72*36
  zfull[indexgrid]=zrot$loadings[,i]
  zmat = matrix(zfull,nrow=nx,ncol=ny)
  image.plot(xlong,ylat,zmat,ylim=range(-50,70),ann=FALSE)
  mtext(paste("Rotated EOF no.",i,sep=""), side = 3, line = 0.2, cex = 0.8)
  contour(xlong,ylat,(zmat),ylim=range(-50,70),add=TRUE,nlev=6,lwd=2)
  map("world2",add=T)
  box()

  plot(1901:2018, scale(rotpcs[,i]),type="l",xlab="Year",axes=FALSE,ann=FALSE)
  axis(2)
  axis(1)
  mtext(paste("Rotated PC no.",i,sep=""), side = 3, line = 0.2, cex = 0.8)
}


############### iii:  Correlate the first four PCs of winter 3-day max precip with the SSTs and vice-versa and, show
#correlation maps.


# read data

winter_precip=read.table('http://civil.colorado.edu/~balajir/CVEN6833/HWs/HW-
2/data/Max_Winter_Seas_Prec.txt', header=T)
xlats=X$Lat
xlongs=X$Lon

years=winter_precip$YEAR
```

```
winter_precip=winter_precip[,-1] # remove year column. rows are years, columns are locations defined by xlats and xlongs

# perform PCA on winter precip data

precip_pca=pca(winter_precip, scale=T)
# plot scree plot
precip_pca$scree_plot

(precip_pve=sum(precip_pca$PVE[1:4]))

######## correlation of precip PCAs with SST anomalies

# make precip and sstannavg years match

# SST data:1901-2018
# precip: 1964 - 2018
match_index=1+nrow(sstannavg)-nrow(precip_pca$pcs)
sstannavg1964=sstannavg[match_index:nrow(sstannavg),]


pPCA_sst_corr=list() # preallocate a list

for (i in 1:4){
  pPCA_sst_corr[[paste('PCA',i,sep = '')]]=cor(scale(precip_pca$pcs[,i]),sstannavg1964)
}
  # each column is the correlation of SST at a given location to PC1, PC2, PC3, and PC4 of precip data, respectively
  # the lat long pairs for each column are given in xygrid1

##### Plot correlation of SST to precip PC1 - PC4 ##################
par(mfrow = c(4,1))
par(mar = c(3, 4, 2, 1))
for(i in 1:4){
  quilt.plot(x=xygrid1[,2],y= xygrid1[,1],z=pPCA_sst_corr[[i]],ann=FALSE, ylim=range(-40,70))
  mtext(paste("PC",i,sep=""), side = 3, line = 0.2, cex = 0.8)
  grid(col="gray70",lty=2)
  map("world2",add=T)
  box()

}

########## correlation of SST PC1-PC4 winter precip observations


precip_sstPCA_corr=list() # preallocate a list

for (i in 1:4){
  precip_sstPCA_corr[[paste('PCA',i,sep =
'')]]=cor(scale(pca_results$pcs[match_index:nrow(pca_results$pcs),i]),winter_precip)
}
# each column is the correlation of precip at a given location to PC1, PC2, PC3, and PC4 of SST data, respectively
# the lat long pairs for each column are given in X
```

```
##### Plot correlation of winter precip to SST PC1 - PC4 ##################
par(mfrow = c(4,1))
par(mar = c(3, 4, 2, 1))
for(i in 1:4){
  quilt.plot(x=X$Lon,y=X$Lat,z=precip_sstPCA_corr[[i]],ann=FALSE, nx=20, ny=20)
  mtext(paste("PC",i,sep=""), side = 3, line = 0.2, cex = 0.8)
  grid(col="gray70",lty=2)
  US(add=T)
  box()
}
```

P5: PCA and multinomial regression

```
rm(list=ls())

## Source libraries and suppress loading messages

# Note: source HW2 library also imports datasets

setwd("G:/My Drive/CU Boulder/Classes/20-21/CVEN 6833 Advanced Stats/HW/HW2")
suppressPackageStartupMessages(source('R Scripts/HW2 Library.R'))
source('R Scripts/HW2 Library.R')
rm('X') # remove X variable

################ Load construction data

path='http://civil.colorado.edu/~balajir/CVEN6833/const-data/'

# retain top 20 variables
to.retain=20
imp.vars.energy=as.character(read.csv(paste(path,"gbm.importance.50.energy.csv",sep
=''),header=TRUE)[1:to.retain,"var"])
imp.vars.inj.type=as.character(read.csv(paste(path,"gbm.importance.50.code.csv",sep=''),header=TRUE)[1:to.retai
n,"var"])
imp.vars.body=as.character(read.csv(paste(path,"gbm.importance.50.body.csv",
sep=''),header=TRUE)[1:to.retain,"var"])

imp.vars.severity=c("crane","heavy.material.tool","stairs","unpowered.tool","exiting.transitioning","heavy.vehicle"
,"uneven.surface","drill","ladder","object.at.height","improper.procedure.inattention","unpowered.transporter","
machinery","improper.security.of.materials","working.at.height","no.improper.PPE","small.particle","steel.steel.se
ctions","stripping","scaffold")

# find column numbers corresponding to the important variables (where binary is some attribute and outcome
data set)

## Read Body part data and select the important variables
data = read.csv(paste(path,"data.body.part.csv", sep=''))
index=which(colnames(data)%in%imp.vars.body)

Xpredictors = data[,index]

bpart=data[,1]
bpart=as.character(bpart)
bpart[bpart == "head"]=1
bpart[bpart == "neck"]=2
bpart[bpart == "trunk"]=3
bpart[bpart == "upper extremities"]=4
bpart[bpart == "lower extremities"]=5

### Now fit Multinomial logistic regression using Xpredictors and ## bpartbin


################################################## PCA ############################
```

```
#pca function in Library
X_pca=pca(Xpredictors, scale=T)

# scree plot:
X_pca$scree_plot

# For first four eigen vectors, plot u (loading score) against variable (ie, what is loading score of the variable)
# record first 4 eigen vectors

par(mfrow=c(2,2))
par(mar=c(4,4,3,1))
p=length(X_pca$u[,1])
for (i in 1:4){

 df=data.frame(phi=abs(X_pca$u[,i]), var=1:p)
 plot(x=df$var, y=df$phi, xlab='var', ylab='loading magnitude', main=paste('PC',i,sep=''), xaxt='none')
 axis(1,seq(2,p, by=2))
 abline(v=seq(2,p, by=2),col='lightgray', lty='dotted')

}

var_name_map=data.frame(index=1:p, name=colnames(Xpredictors))


## part(b)
############################################ fit Multinomial regression ############################
 # use principal components as predictors to model categorical probability of injuries to the five categories of body
parts.
 # compute RPSS (ranked probability skill score)

### create binary vector
bpartbin=as.numeric(bpart) # supervisor variable

pcs=cbind(bpartbin,X_pca$pcs)
pcs = data.frame(pcs)
## model with first 5 PCS
zz = multinom(bpartbin ~ V2+V3+V4+V5+V6, data=pcs)

## or model with all the PCS
#zz = multinom(bpartbin ~ ., data=pcs)

N = log(length(bpart)) # model complexity penalty coefficient for BIC

## BIC
 #k is coefficient in the AIC error term. If k=2, then AIC. If k = log(n), then BIC
zbest=stepAIC(zz,k=N)
summary(zbest) # best model, according to BIC, uses all 5 PC
#################### RPSS
ypred = predict(zz, type = "probs")
# RPSS calculations
acc2 = as.numeric(bpart) # just a vector of observations, for use in rps() below
N = length(acc2)
```

```
### using verification package you get the same results..
# Empirical probabilities
p1 <- length(acc2[acc2 == 1])/N
p2 <- length(acc2[acc2 == 2])/N
p3 <- length(acc2[acc2 == 3])/N
p4 <- length(acc2[acc2 == 4])/N
p5 <- length(acc2[acc2 == 5])/N

climo <- c(p1, p2, p3,p4,p5)

# ranked probability skill score
rps(obs=acc2, pred=ypred, baseline=NULL)$rpss # why is this so different?

# part (c)
##################################### Multinomial regression to predict injury severity
############################

## Read Body part data and select the important variables
data = read.csv(paste(path,"data.worst.case.severity.csv", sep=''))
index=which(colnames(data)%in%imp.vars.body)

Xpredictors = data[,index]

sev=as.character(data[,1])

# Divide body parts into five classes
sev[sev == "1st Aid"]=1
sev[sev == "Medical Case"]=2
sev[sev == "Lost Work Time"]=3
sev[sev == "Permanent Disablement"]=4
sev[sev == "Fatality"]=5

### create binary vector
sevbin=as.numeric(sev)

############################## PCA ############################

#pca function in Library
sev_pca=pca(Xpredictors, scale=T)

# scree plot:
sev_pca$scree_plot

# For first four eigen vectors, plot u (loading score) against variable (ie, what is loading score of the variable)
# record first 4 eigen vectors

par(mfrow=c(2,2))
par(mar=c(4,4,3,1))
p=length(sev_pca$u[,1])
for (i in 1:4){

  df=data.frame(phi=abs(sev_pca$u[,i]), var=1:p)
```

```
 plot(x=df$var, y=df$phi, xlab='var', ylab='loading magnitude', main=paste('PC',i,sep=''), xaxt='none')
 axis(1,seq(2,p, by=2))
 abline(v=seq(2,p, by=2),col='lightgray', lty='dotted')

}

var_name_map=data.frame(index=1:p, name=colnames(Xpredictors))


## model with first 5 PCS
pcs=cbind(sevbin, sev_pca$pcs[,1:5])
pcs=data.frame(pcs)
zz = multinom(sevbin ~ V2+V3+V4+V5+V6, data=pcs)

## or model with all the PCS
#zz = multinom(bpartbin ~ ., data=pcs)

N = log(length(sevbin)) # model complexity penalty coefficient for BIC

## BIC
#k is coefficient in the AIC error term. If k=2, then AIC. If k = log(n), then BIC
zbest=stepAIC(zz,k=N)
summary(zbest) # best model, according to BIC, uses only PC1 (V2)
##################### RPSS
ypred = predict(zz, type = "probs")
# RPSS calculations
acc2 = as.numeric(sevbin) # just a vector of observations, for use in rps() below
N = length(acc2)

### using verification package you get the same results..
# Empirical probabilities
p1 <- length(acc2[acc2 == 1])/N
p2 <- length(acc2[acc2 == 2])/N
p3 <- length(acc2[acc2 == 3])/N
p4 <- length(acc2[acc2 == 4])/N
p5 <- length(acc2[acc2 == 5])/N

climo <- c(p1, p2, p3,p4,p5)

# ranked probability skill score
rps(obs=acc2, pred=ypred, baseline=NULL)$rpss



######################################### Fit a CART and compare results ############################


par(mfrow=c(1,1))
par(mar=c(2,2,2,2))
# to force tree() to do classification, Y must be a factor
pcs$sev=as.factor(sev) # add column of y variable as categories. Must be factor.
# I used rpart instead of tree() to use cp parameter. Controls the amount of improvement in purity in each split
must be attained
```

```
  # i reduced the default such that more variables would be included.
sev_tree=rpart(sev ~ V2+V3+V4+V5+V6, data=pcs, model=T, control = rpart.control(minbucket = 1,cp=0.003))

# With tree function, results are only one split, and everthing predicts 2 as outcome.
# sev_tree=tree(sev ~ V2+V3+V4+V5+V6, data=pcs, model=T, control = tree.control(nobs=nrow(pcs),mincut = 0,
minsize = 0))

summary(sev_tree)
plot(sev_tree)
text(sev_tree, cex=.7)
mtext('Severity Tree', line=1.1 ,cex=.9)

ypredTree=predict(sev_tree)
rps(obs=acc2, pred=ypredTree, baseline = NULL)$rpss
# treeFit=treeFun(sev_tree, toPlot=T, title='Injury Severity')


sum(pcs$sevbin=='1')
sum(pcs$sevbin=='2')
sum(pcs$sevbin=='3')
sum(pcs$sevbin=='4')
sum(pcs$sevbin=='5')
```

P6: Multivariate forecasting with PCA and CART

```
rm(list=ls())

## Source libraries and suppress loading messages

# Note: source HW2 library also imports datasets

setwd("G:/My Drive/CU Boulder/Classes/20-21/CVEN 6833 Advanced Stats/HW/HW2")
suppressPackageStartupMessages(source('R Scripts/HW2 Library.R'))
source('R Scripts/HW2 Library.R')

##################################### Read SST data ########################################

sst_data=readSST() # see HW2 Library.R
sstannavg=sst_data$sst

########################################## Read winter precip data
################################################

winter_precip=read.table('http://civil.colorado.edu/~balajir/CVEN6833/HWs/HW-
2/data/Max_Winter_Seas_Prec.txt', header=T)
xlats=X$Lat
xlongs=X$Lon

years=winter_precip$YEAR
winter_precip=winter_precip[,-1] # remove year column. rows are years, columns are locations defined by xlats
and xlongs

# make precip and sstannavg years match

# SST data:1901-2018
# precip: 1964 - 2018
match_index=1+nrow(sstannavg)-nrow(winter_precip)
sstannavg1964=sstannavg[match_index:nrow(sstannavg),]

############ PCA on summer global SST anomalies and Precip Data #############

# see HW2 library for pca() function
pca_SST=pca(data=sstannavg1964, scale=T)
# plot eigen variance spectrum for first 25 modes
pca_SST$scree_plot # return scree plot, AKA eigen spectrum

# perform PCA on winter precip data
pca_precip=pca(winter_precip, scale=T)
# plot scree plot
pca_precip$scree_plot

############################################## Fit CART ###################################

tree.list=list() # preallocate a list
```

```
set.seed(5)

npc=10

par(mfrow=c(npc/2,2))
par(mar=c(2,3,2.5,1))



for (i in 1:npc){ # model first npc PCs
 df=data.frame(precip_pc=pca_precip$pcs[,i], pca_SST$pcs[,1:npc])
 tree.unpruned=tree(precip_pc ~ ., data = df, model =T) # need model = T for cv.tree to have dataframe

 plot(tree.unpruned)
 text(tree.unpruned, cex=.7)
 mtext(paste('PC', i, 'untrimmed'), line=1.1 ,cex=.9)

 #Perform CV on tree object
 cvTree <- cv.tree(tree.unpruned)
 optTree <- which.min(cvTree$dev)
 bestTree <- cvTree$size[optTree]

 if (bestTree ==1){
   bestTree = 2
   print(paste ( 'Min deviance for PC', i, ' is 1 node.'))
 }

 #prune Tree based on CV results
 pruneTree <- prune.tree(tree.unpruned, best = bestTree)

 tree.list[[paste('PC',i, sep='')]]=pruneTree

 plot(tree.list[[paste('PC',i, sep='')]])
 text(tree.list[[paste('PC',i, sep='')]], cex=.7)
 mtext(paste('PC', i, 'trimmed'), line=1.1, cex=.9)
}

####### predict precip at locations

Zmat=matrix(NA, nrow=nrow(winter_precip), ncol=npc)

for(i in 1:npc){
  Zmat[,i]=predict(tree.list[[i]])
}

phi_npc=pca_precip$u[,1:npc]

ypred=Zmat %*%t(phi_npc) # need to adjust for centering and scaling from scale() function in pca

s=scale(winter_precip)
scale_vals=attributes(s)
center=scale_vals$`scaled:center`
stdev=scale_vals$`scaled:scale`
```

```
precpred=ypred*stdev+center


############## Part (iii): Evaluate the performance by computing R2 between the observed and predicted
# winter precipitation at each grid point.

precPC=pca_precip$pcs[,1:npc]

### correlate the forecasted PCs with the historical PCs
pccor = diag(cor(ypred,precPC))

print("the correlation between observed and forecasted PCs is:")
print(round(pccor*1000)/1000)

####### correlate forecasted rainfall with historical winter precip
preccor = diag(cor(precpred,winter_precip))
print('the range of correlation between observed and forecasted winter precip is:')

print(range(preccor))

# plot correlation spatially

#plot of R2 between the observed and predicted winter precipitation at each grid point
nlevel=10
coltab=rev(brewer.pal(nlevel-1,'RdBu')) # rev just reverses order. brewer.pal created color pallettes
xlon=X$Lon
xlat=X$Lat

par(mar=c(5,5,3,3))
par(mfrow=c(1,1))
zr=range(preccor)
break_a=round(seq(zr[1],zr[2],by=((zr[2]-zr[1])/(nlevel-1)))*100)/100
quilt.plot(xlon, xlat,z=(preccor) ,xlim=range(-115,-103),ylim=c(29,42),nx=30, ny=30,col=coltab,
        main = bquote(" Corr. between obs.& pred. winter precip"),zlim=zr,lab.breaks=break_a, xlab='Longitude',
ylab='Latitude')
grid(col="gray70",lty=2)
US(add=TRUE, col="gray50", lwd=2,xlim=range(-125,-100))
box()

############################## Drop 10 analysis ###################################

sstPC=pca_SST$pcs
precPC=pca_precip$pcs
prec_loading=pca_precip$u

drop10_results=CART_drop10(y=winter_precip, x=sstannavg1964, y_pc=precPC, x_pc=sstPC,
y_loading=prec_loading, npc=npc)

par(mfrow=c(1,2))
par(mar=c(3,3,3,1))
boxplot(drop10_results$corr, main='Corr')
boxplot(drop10_results$rmse, main='RMSE (mm)')
```

```
############################################## Repeat with Random Forest
#################################################################

# see HW2 Library R for RF_PCs function. It fits random forest to each y PC using x PCs

predPC=RF_PCs(y_pc=precPC, x_pc=sstPC, npc=10)

prec_loading=pca_precip$u

ypred_scaled=predPC %*% t(prec_loading)
ypred=ypred_scaled*stdev+center

####### correlate forecasted rainfall with historical winter precip
preccor = diag(cor(ypred,winter_precip))
print('the range of correlation between observed and forecasted winter precip is:')

print(range(preccor))

# plot correlation spatially

#plot of R2 between the observed and predicted winter precipitation at each grid point

par(mar=c(5,5,3,3))
par(mfrow=c(1,1))
zr=range(preccor)
break_a=round(seq(zr[1],zr[2],by=((zr[2]-zr[1])/(nlevel-1)))*100)/100
quilt.plot(xlon, xlat,z=(preccor) ,xlim=range(-115,-103),ylim=c(29,42),nx=30, ny=30,col=coltab,
        main = bquote(" Corr. between obs.& pred. winter precip"),zlim=zr,lab.breaks=break_a, xlab='Longitude',
ylab='Latitude')
grid(col="gray70",lty=2)
US(add=TRUE, col="gray50", lwd=2,xlim=range(-125,-100))
box()

################## Drop 10 with random forest ########


RF_drop10results=RF_drop10(y=winter_precip, x=sstannavg1964, y_pc=precPC, x_pc=sstPC,
y_loading=prec_loading,npc=npc)

par(mfrow=c(1,2))
par(mar=c(3,3,3,1))
boxplot(RF_drop10results$corr, main='Corr')
boxplot(RF_drop10results$rmse, main='RMSE (mm)')
```

P7: K-means clustering

```
rm(list=ls())

## Source libraries and suppress loading messages

# Note: source HW2 library also imports datasets

setwd("G:/My Drive/CU Boulder/Classes/20-21/CVEN 6833 Advanced Stats/HW/HW2")
suppressPackageStartupMessages(source('R Scripts/HW2 Library.R'))
source('R Scripts/HW2 Library.R')
library(ggConvexHull)

######### Read winter data. Summer read by HW2 Library.R

winter_precip_ts=read.table('http://civil.colorado.edu/~balajir/CVEN6833/HWs/HW-
2/data/Max_Winter_Seas_Prec.txt', header=T)

winter_precip=apply(winter_precip_ts,2,FUN=median)[-1]
winter_precip=data.frame(Pm=winter_precip, Lat=X$Lat, Lon=X$Lon, Elev=X$Elev)

################################### Cluster Summer & Winter Precip ################################

summer_scale=scale(Summer_Precip)
winter_scale=scale(winter_precip)

Niter=50
WSS = matrix(nrow=Niter, ncol=15)
for (j in 1:Niter) {
  WSS[j,1]= (nrow(summer_scale)-1)*sum(apply(summer_scale,2,var))

  for (i in 2:15) {
    WSS[j,i] = sum(kmeans(summer_scale, centers=i)$withinss)
  }}
wss=apply(WSS, 2, FUN = mean)

P1=ggplot() +
  geom_line(mapping = aes(1:15, wss)) +
  geom_point(mapping = aes(1:15, wss), shape = 21, size = 3, color = "gray30", fill ="cadetblue1")+
  labs(title = "Summer Max Precip",x = "Number of Clusters",y = "Within groups sum of squares")+
  theme(plot.title = element_text(size = 13, face = "bold", hjust = 0.5))



WSS1 = matrix(nrow=Niter, ncol=15)
for (j in 1:Niter) {
  WSS1[j,1]= (nrow(winter_scale)-1)*sum(apply(winter_scale,2,var))

  for (i in 2:15) {
    WSS1[j,i] = sum(kmeans(winter_scale, centers=i)$withinss)
  }}
wss1=apply(WSS1, 2, FUN = mean)
```

```
P2=ggplot() +
 geom_line(mapping = aes(1:15, wss1)) +
 geom_point(mapping = aes(1:15, wss1), shape = 21, size = 3, color = "gray30", fill ="cadetblue1")+
 labs(title = "Winter Max Precip",x = "Number of Clusters",y = "Within groups sum of squares")+
 theme(plot.title = element_text(size = 13, face = "bold", hjust = 0.5))



###################### Summer ###################

## find the number of clusters
# hand select from elbow in P1
P1
num_clusters=5

summer_fit=kmeans(summer_scale, centers=num_clusters, iter.max = 100, nstart=30)

means=aggregate(Summer_Precip,by=list(summer_fit$cluster),FUN=mean) # split data into subsets based on
cluster assignment, compute mean of each cluster
# append cluster assignment
Cluster=factor(summer_fit$cluster)
data_1 = data.frame(Summer_Precip,Cluster)

mainStates=map_data('state')
states_full=c("arizona","colorado","new mexico","utah")

xlim1=range(X$Lon)
ylim1=range(X$Lat)


ggplot()+
 geom_polygon(data=mainStates, aes(x=long, y=lat, group=group), color='gray20', fill='transparent')+
 geom_point(data=data_1,mapping=aes(x=Lon, y=Lat, color=Cluster, size=Prec))+
 geom_convexhull(data=data_1, alpha=.3, mapping=aes(fill=Cluster, x=Lon, y=Lat))+
 ggtitle("Summer max precip. clusters") +
 xlab("Longitude") +
 ylab("Latitude")+
 scale_size(name='Obs Prec (mm)')+ # Observed Precip Legend
 theme_bw()+
 theme(legend.position="right")+
 theme(panel.grid.major = element_line(colour = "grey50", linetype = "dashed"),
       legend.text = element_text(size = 10),plot.margin=unit(c(0.0,0.01,0.01,0.01),"in"))+
 coord_sf(xlim = xlim1, ylim = ylim1,label_axes = list(bottom = "E", left = "N"), expand =FALSE)


###################### Winter ###################

## find the number of clusters
#Considering elevation
P2

num_clusters=5
```

```
winter_fit=kmeans(winter_scale, centers=num_clusters, iter.max = 100, nstart=30)


means=aggregate(winter_precip,by=list(winter_fit$cluster),FUN=mean)
# append cluster assignment
Cluster=factor(winter_fit$cluster)
data_2 = data.frame(winter_precip,Cluster)

ggplot()+
  geom_polygon(data=mainStates, aes(x=long, y=lat, group=group), color='gray20', fill='transparent')+
  geom_point(data=data_2,mapping=aes(x=Lon, y=Lat, color=Cluster, size=Pm))+
  geom_convexhull(data=data_2, alpha=.3, mapping=aes(fill=Cluster, x=Lon, y=Lat))+
  ggtitle("Winter max precip. clusters") +
  xlab("Longitude") +
  ylab("Latitude")+
  scale_size(name='Obs Prec (mm)')+ # Observed Precip Legend
  theme_bw()+
  theme(legend.position="right")+
  theme(panel.grid.major = element_line(colour = "grey50", linetype = "dashed"),
      legend.text = element_text(size = 10),plot.margin=unit(c(0.0,0.01,0.01,0.01),"in"))+
  coord_sf(xlim = xlim1, ylim = ylim1,label_axes = list(bottom = "E", left = "N"), expand =FALSE)


################################ Hierarchical Clustering ############################


######## summer

summer_hier=hclust(d=dist(summer_scale), method='complete')
plot(summer_hier)
rect.hclust(summer_hier, k=5, border=2:6)

# append cluster assignment

summer_cut=as.character(cutree(summer_hier, k=5))
data_3 = data.frame(Summer_Precip, Cluster=summer_cut)

ggplot()+
  geom_polygon(data=mainStates, aes(x=long, y=lat, group=group), color='gray20', fill='transparent')+
  geom_point(data=data_3,mapping=aes(x=Lon, y=Lat, color=Cluster, size=Prec))+
  geom_convexhull(data=data_3, alpha=.3, mapping=aes(fill=Cluster, x=Lon, y=Lat))+
  ggtitle("Summer max precip. clusters: hierarchical") +
  xlab("Longitude") +
  ylab("Latitude")+
  scale_size(name='Obs Prec (mm)')+ # Observed Precip Legend
  theme_bw()+
  theme(legend.position="right")+
  theme(panel.grid.major = element_line(colour = "grey50", linetype = "dashed"),
      legend.text = element_text(size = 10),plot.margin=unit(c(0.0,0.01,0.01,0.01),"in"))+
  coord_sf(xlim = xlim1, ylim = ylim1,label_axes = list(bottom = "E", left = "N"), expand =FALSE)

######## winter
```

```
winter_hier=hclust(d=dist(winter_scale), method='complete')
plot(winter_hier)
rect.hclust(winter_hier, k=5, border=2:6)
#

winter_cut=as.character(cutree(winter_hier, k=5))

data_4 = data.frame(winter_precip, Cluster=winter_cut)

ggplot()+
 geom_polygon(data=mainStates, aes(x=long, y=lat, group=group), color='gray20', fill='transparent')+
 geom_point(data=data_4,mapping=aes(x=Lon, y=Lat, color=Cluster, size=Pm))+
 geom_convexhull(data=data_4, alpha=.3, mapping=aes(fill=Cluster, x=Lon, y=Lat))+
 ggtitle("winter max precip. clusters: hierarchical") +
 xlab("Longitude") +
 ylab("Latitude")+
 scale_size(name='Obs Prec (mm)')+ # Observed Precip Legend
 theme_bw()+
 theme(legend.position="right")+
 theme(panel.grid.major = element_line(colour = "grey50", linetype = "dashed"),
     legend.text = element_text(size = 10),plot.margin=unit(c(0.0,0.01,0.01,0.01),"in"))+
 coord_sf(xlim = xlim1, ylim = ylim1,label_axes = list(bottom = "E", left = "N"), expand =FALSE)
```

P8: Extremes clustering

```
rm(list=ls())

setwd("G:/My Drive/CU Boulder/Classes/20-21/CVEN 6833 Advanced Stats/HW/HW2")
suppressPackageStartupMessages(source('R Scripts/HW2 Library.R'))
source('R Scripts/HW2 Library.R')
library(ggConvexHull)

winter_precip_ts=read.table('http://civil.colorado.edu/~balajir/CVEN6833/HWs/HW-
2/data/Max_Winter_Seas_Prec.txt', header=T)

# winter_precip=apply(winter_precip_ts,2,FUN=median)[-1]
# winter_precip=data.frame(Pm=winter_precip, Lat=X$Lat, Lon=X$Lon, Elev=X$Elev)
grid=matrix(c(X$Lat, X$Lon), ncol=2)

######################################## Clustering Extremes using method in Bracket et al. 2015
########################

ks = 2:20

# Do the clustering for all k values
cluster_list = list()

# this function is looking for a data frame where rows are time, columns are locations

for(k in ks){
  #message('Trying ',k,' clusters')
  cluster_list[[which(ks == k)]] = pam_fmado_ll(x=winter_precip_ts[,-1], k=k, ll=grid)
}

# find the average silhouette value (smaller is better)
sil = sapply(cluster_list,function(x)x$silinfo$avg.width)

# plot the ave silhouette value versus k, look for a maximum
ggplot() +
  geom_line(mapping = aes(ks,sil)) +
  geom_point(mapping = aes(ks,sil), shape = 21, size = 3, color = "gray30", fill ="cadetblue1") +
  labs(title = "Silhouette width",x = 'Number of clusters',y = 'Silhouette width')+
  theme(plot.title = element_text(size = 13, face = "bold", hjust = 0.5))

# find the optimal k
optimal_k = ks[which.max(sil)]
message('The optimal number of clusters is: ',optimal_k)


############################### Spatial Plotting of Clusters ################################
clusters=data.frame(Lon=X$Lon, Lat=X$Lat, Cluster=as.character(cluster_list[[which(ks==5)]]$clustering))


mainStates=map_data('state')
states_full=c("arizona","colorado","new mexico","utah")
```

```
xlim1=range(X$Lon)
ylim1=range(X$Lat)


ggplot()+
 geom_polygon(data=mainStates, aes(x=long, y=lat, group=group), color='gray20', fill='transparent')+
 geom_point(data=clusters,mapping=aes(x=Lon, y=Lat, color=Cluster))+
 geom_convexhull(data=clusters, alpha=.3, mapping=aes(fill=Cluster, x=Lon, y=Lat))+
 ggtitle("Winter Extremes Clustering: k=5") +
 xlab("Longitude") +
 ylab("Latitude")+
 theme_bw()+
 theme(legend.position="right")+
 theme(panel.grid.major = element_line(colour = "grey50", linetype = "dashed"),
     legend.text = element_text(size = 10),plot.margin=unit(c(0.0,0.01,0.01,0.01),"in"))+
 coord_sf(xlim = xlim1, ylim = ylim1,label_axes = list(bottom = "E", left = "N"), expand =FALSE)

########################## Use K = 5 to directly compare to Kmeans in P7 ##########

clusters=data.frame(Lon=X$Lon, Lat=X$Lat, Cluster=as.character(cluster_list[[which(ks==optimal_k)]]$clustering))


mainStates=map_data('state')
states_full=c("arizona","colorado","new mexico","utah")

xlim1=range(X$Lon)
ylim1=range(X$Lat)


ggplot()+
 geom_polygon(data=mainStates, aes(x=long, y=lat, group=group), color='gray20', fill='transparent')+
 geom_point(data=clusters,mapping=aes(x=Lon, y=Lat, color=Cluster))+
 geom_convexhull(data=clusters, alpha=.3, mapping=aes(fill=Cluster, x=Lon, y=Lat))+
 ggtitle("Winter Extremes Clustering") +
 xlab("Longitude") +
 ylab("Latitude")+
 theme_bw()+
 theme(legend.position="right")+
 theme(panel.grid.major = element_line(colour = "grey50", linetype = "dashed"),
     legend.text = element_text(size = 10),plot.margin=unit(c(0.0,0.01,0.01,0.01),"in"))+
 coord_sf(xlim = xlim1, ylim = ylim1,label_axes = list(bottom = "E", left = "N"), expand =FALSE)
```

P9: Self-Organizing Maps

```
rm(list=ls())

setwd("G:/My Drive/CU Boulder/Classes/20-21/CVEN 6833 Advanced Stats/HW/HW2")
suppressPackageStartupMessages(source('R Scripts/HW2 Library.R'))
source('R Scripts/HW2 Library.R')
library(ggConvexHull)


############## Winter Precip Data #######################################
winter_precip_ts=read.table('http://civil.colorado.edu/~balajir/CVEN6833/HWs/HW-
2/data/Max_Winter_Seas_Prec.txt', header=T)

# winter_precip=apply(winter_precip_ts,2,FUN=median)[-1]
# winter_precip=data.frame(Pm=winter_precip, Lat=X$Lat, Lon=X$Lon, Elev=X$Elev)
grid=data.frame(Lat=X$Lat, Lon=X$Lon)

prec=winter_precip_ts[,-1]
############ SST Anomaly ##################################

# I took Alvaro's code to read the SST data and turned it into a function in the HW Library.R file
SSTdata=readSST()
sstannavg=SSTdata$sst

######## Make precip and sst years the same
# SST data:1901-2018
# precip: 1964 - 2018
match_index=1+nrow(sstannavg)-nrow(winter_precip_ts)
sstannavg1964=sstannavg[match_index:nrow(sstannavg),]

############################################### SOM Analysis
########################################
##########################################################################################################
########

#################### 3x3 SOM

Ng=3
N_node=Ng^2
set.seed(5)

SOM=som(scale(prec), grid=somgrid(3,3, 'rectangular'), rlen=10000)

plot(SOM)
plot(SOM,type='changes')

# plot # of observations in each node

colors <- function(n, alpha = 1) {
  rev(heat.colors(n, alpha))
}
```

```
plot(SOM, type = "counts",
    palette.name = colors,
    heatkey = TRUE)


prec_nodes=as.data.frame(array(NaN, dim = c(dim(prec)[2],N_node)))
for(i in 1:N_node){
 index=which(SOM$unit.classif==i) # find years that belong to cluster i
 tmp_df=prec[index,] # place those years, for every location, in a temp. df
 if(length(index)>1){
  prec_nodes[,i]=colMeans(tmp_df,na.rm=T) #if length(index) > 1, then you have multiple years, and you need a
summary statistic (mean) to plot
 }else{
  prec_nodes[,i]=unlist(tmp_df)
 }
}


zr=quantile(as.vector(t(prec_nodes)), probs=c(0,0.95)) # there is a HUGE outlier, so use 95th percentile
#plot composited WSMP data for each node
Nx=20
par( oma=c(0,2,3,6)) # save some room for the legend
par(mfrow = c(Ng, Ng))
#set.panel(4,4)
par(mar = c(3, 3, 2, 1))
for(i in 1:N_node){
 quilt.plot(grid$Lon, grid$Lat,prec_nodes[,i],main=paste("Node ",i,sep = ""),zlim=zr,add.legend=FALSE,nx=Nx,
ny=Nx)

 US( add=TRUE, col="gray30", lwd=1)
}
mtext("Winter Seasonal Maximun Precipitation SOM Maps 3x3", outer = TRUE, side = 3, cex = 1.2, line = 1)
par( oma=c(1,0,0,1))
par(mar = c(0, 0, 0, 0))
image.plot(zlim=zr,legend.only=TRUE,legend.shrink=1,legend.width=3)


######################################## Display composited SST for each node case: 3x3
#################

# unload package kohonen that creates a conflict with map("world2",add=T)
detach(package:kohonen)
# Display composited SST days for each node case: 3x3
SST_nodes=as.data.frame(array(NaN, dim = c(dim(sstannavg1964)[2],N_node)))
for(i in 1:N_node){
 index=which(SOM$unit.classif==i)
 tmp_df=sstannavg1964[index,]
 if(length(index)>1){
  SST_nodes[,i]=colMeans(tmp_df,na.rm=T)
 }else{
  SST_nodes[,i]=tmp_df
 }
```

```
}


nglobe=SSTdata$nglobe
indexgrid=SSTdata$indexgrid
lon=SSTdata$lon
lat=SSTdata$lat
nrows=72
ncols=36

zr=range(SST_nodes)
Nx=50
par( oma=c(0,0,2.5,1)) # save some room for the legend
par(mfrow = c(Ng, Ng))
#set.panel(4,4)
par(mar = c(2, 3, 2, 1))
for(i in 1:N_node){
  zfull = rep(NA,(nglobe))   #also equal 72*36
  zfull[indexgrid]=SST_nodes[,i]
  zmat = matrix(zfull,nrow=nrows,ncol=ncols)
  # zmat[which(is.na(zmat))]=0
  # quilt.plot(x=lon, y=lat, z=zfull, ylim=range(c(-40,70)), zlim=zr)
  image.plot(lon,lat,zmat,ylim=range(-40,70), zlim=zr,main = paste("Node ",i,sep=""))
  contour(lon,lat,zmat,ylim=range(-40,70),add=TRUE,nlev=6,lwd=1)
  # map("world2",add=T)
}
mtext("Composite SST SOM Maps 3x3", outer = TRUE, side = 3, cex = 1.2, line = 1)
```

P10: Canonical Correlation Analysis

```
rm(list=ls())

## Source libraries and suppress loading messages

# Note: source HW2 library also imports datasets

setwd("G:/My Drive/CU Boulder/Classes/20-21/CVEN 6833 Advanced Stats/HW/HW2")
suppressPackageStartupMessages(source('R Scripts/HW2 Library.R'))
source('R Scripts/HW2 Library.R')



################ read SST ann avg data and winter precip data ##############################

SST_data=readSST()
sstannavg=SST_data$sst

# read winter precip data

winter_precip=read.table('http://civil.colorado.edu/~balajir/CVEN6833/HWs/HW-
2/data/Max_Winter_Seas_Prec.txt', header=T)
xlats=X$Lat
xlongs=X$Lon

years=winter_precip$YEAR
winter_precip=winter_precip[,-1] # remove year column. rows are years, columns are locations defined by xlats
and xlongs

# make precip and sstannavg years match

# SST data:1901-2018
# precip: 1964 - 2018
match_index=1+nrow(sstannavg)-length(years)
sstannavg1964=sstannavg[match_index:nrow(sstannavg),]


############################################## Part (i): PCA and CCA with leading 4 pcs
#############################

############## Perform PCA ####################

pca_precip=pca(winter_precip, scale = T)

pca_sst=pca(sstannavg1964, scale = T)

npc=4 # number of pcs chosen for analysis

sstPC=pca_sst$pcs[,1:npc]
precPC=pca_precip$pcs[,1:npc]
```

```
############## CCA on the PCs ####################

M=dim(sstPC)[2]
J=dim(precPC)[2]
J=min(M,J) # number of PCs
N = length(precPC[,1]) #number of years
Qx1 = qr.Q(qr(sstPC)) # qr is QR decomposition of matrix to solve Ax=b. qr.Q returns orthogonal transformation, Q
Qy1 = qr.Q(qr(precPC))
T11 = qr.R(qr(sstPC)) # returns component R
T22 = qr.R(qr(precPC))
VV=t(Qx1) %*% Qy1
BB = solve(T22) %*% svd(VV)$v * sqrt(N-1)
wm1 = precPC %*% BB
AA = solve(T11) %*% svd(VV)$u * sqrt(N-1)
vm1 = sstPC %*% AA

############ Part (ii): Fit a regression for each canonical variate - the canonical variate of precip related to
canonical variate
# of SSTs

### Predict the winter precipitation PCS
betahat = solve(t(AA) %*% t(sstPC)%*% sstPC %*% AA) %*% t(AA) %*% t(sstPC) %*% precPC
ypred=sstPC %*% AA %*% betahat
### first npc PCs from the PC forecast above and the remaining PCs are set to
## their means - i.e., 0

N1 = dim(winter_precip)[2]-(npc)

precPCpred = cbind(ypred,matrix(rep(0,N),ncol=N1,nrow=N)) # Z matrix, which is X*E

## back transform to get the winter precipitation field

###  Keep only the first npc Eigen Vectors and set rest to zero
E = matrix(0,nrow=dim(winter_precip)[2],ncol=dim(winter_precip)[2])
E[,1:npc]=pca_precip$u[,1:npc] # E matrix, ie loading or eigenvectors


## back transform to get the winter precipitation field
precpred =  precPCpred %*% t(E)


######## rescale winter precip

s=scale(winter_precip)
scale_vals=attributes(s)
center=scale_vals$`scaled:center`
stdev=scale_vals$`scaled:scale`

precpred=t(t(precpred)*stdev +center)

############## Part (iii): Evaluate the performance by computing R2 between the observed and predicted
# winter precipitation at each grid point.
```

```
### correlate the forecasted PCs with the historical PCs
pccor = diag(cor(ypred,precPC))

print("the correlation between observed and forecasted PCs is:")
print(round(pccor*1000)/1000)

####### correlate forecasted rainfall with historical winter precip
preccor = diag(cor(precpred,winter_precip))
print('the range of correlation between observed and forecasted winter precip is:')

print(range(preccor))


# plot correlation spatially

#plot of R2 between the observed and predicted winter precipitation at each grid point
nlevel=10
coltab=rev(brewer.pal(nlevel-1,'RdBu')) # rev just reverses order. brewer.pal created color pallettes
xlon=X$Lon
xlat=X$Lat


zr=range(preccor)
break_a=round(seq(zr[1],zr[2],by=((zr[2]-zr[1])/(nlevel-1)))*100)/100
quilt.plot(xlon, xlat,z=(preccor) ,xlim=range(-115,-103),ylim=c(29,42),nx=30, ny=30,col=coltab,
        main = bquote(" Corr. between obs.& pred. winter precip"),zlim=zr,lab.breaks=break_a, xlab='Longitude',
ylab='Latitude')
grid(col="gray70",lty=2)
US(add=TRUE, col="gray50", lwd=2,xlim=range(-125,-100))
box()

zr=range(preccor^2)
break_a=round(seq(zr[1],zr[2],by=((zr[2]-zr[1])/(nlevel-1)))*100)/100
quilt.plot(xlon, xlat,z=(preccor^2) ,xlim=range(-115,-103),ylim=c(29,42),nx=30, ny=30,col=coltab,
        main = bquote(~ R^2 ~ "between obs.& pred. winter precip"),zlim=zr,lab.breaks=break_a, xlab='Longitude',
ylab='Latitude')
grid(col="gray70",lty=2)
US(add=TRUE, col="gray50", lwd=2,xlim=range(-125,-100))
box()


################################ Part (iv): Do a leave-one-out cross validation
#########################################################
# Compare with results from problem 6

# see cca_loocv function in HW2 Library.R

loocv_results=cca_loocv(y=winter_precip,x=sstannavg1964, y_pc=precPC, x_pc=sstPC, y_loading=pca_precip$u )
loocv_corr=loocv_results$corr
loocv_RMSE=loocv_results$rmse

# get correlation/RMSE between fitted model and observations for each year. Earlier, you got correlation for each
location
```

```
fit_year_corr=rep(NA, nrow(winter_precip))
fit_year_RMSE=rep(NA, nrow(winter_precip))
for (i in 1:nrow(winter_precip)){

 fit_year_corr[i]=cor(precpred[i,], as.numeric(winter_precip[i,]))
 errors=precpred[i,]-winter_precip[i,]
 fit_year_RMSE[i]=sqrt(sum(errors^2)/length(errors))
}


par(mar=c(5,5,3,3))
plot(x=1964:2018, y=loocv_corr, type='l', xlab='Year', ylab='Correlation',
    main='Cor. between Observations and Model', col='red', lwd=3)
lines(x=1964:2018, y = fit_year_corr, col='blue', lty=2)
abline(h=seq(0,1, by=.2), col='lightgrey', lty=3)
abline(v=seq(1965,2020, by=5), col='lightgrey', lty=3)
legend('bottomright', legend=c('LOOCV', 'all years fit'), lty=c(1,2), col=c('red', 'blue'), lwd=c(3,1), cex=.7)

boxplot(fit_year_corr, main='LOOCV Correlation')


range(loocv_RMSE, fit_year_RMSE)
plot(x=1964:2018, y=loocv_RMSE, type='l', xlab='Year', ylab='RMSE',
    main='RMSE between Observations and Model', col='red', lwd=3)
lines(x=1964:2018, y = fit_year_RMSE, col='blue', lty=2)
abline(h=seq(10,180, by=10), col='lightgrey', lty=3)
abline(v=seq(1965,2020, by=5), col='lightgrey', lty=3)
legend('topright', legend=c('LOOCV', 'all years fit'), lty=c(1,2), col=c('red', 'blue'), lwd=c(3,1), cex=.7)

boxplot(fit_year_RMSE, main='LOOCV RMSE [mm]')
median(fit_year_RMSE)
```