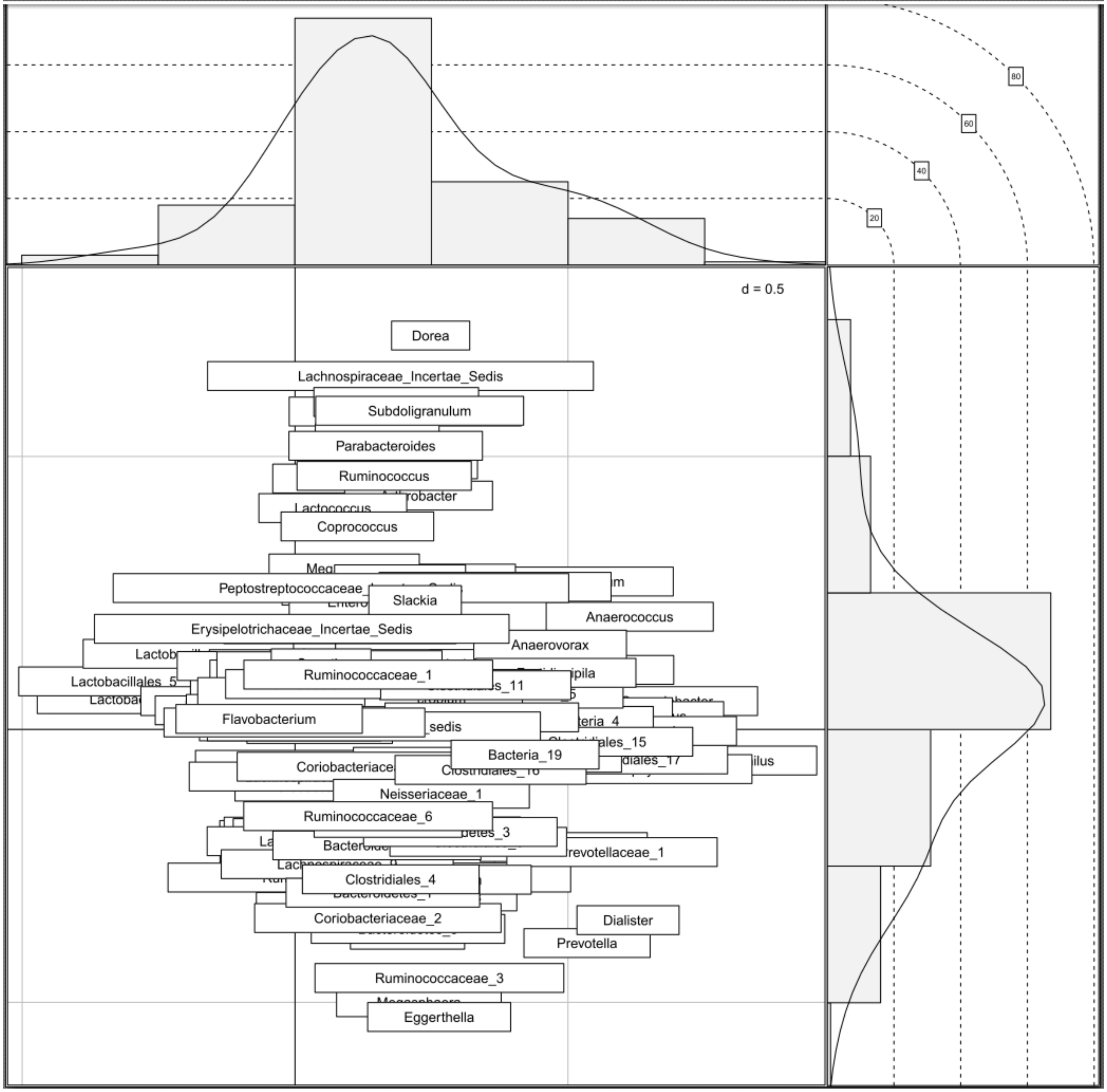
Supplemental material

Additional File 1: Figure S1
Additional File 2: Code S1
Additional File 3: Code S2
Additional File 4: Code S3
Additional File 5: Figure S2
Additional File 6: Figure S3
Additional File 7: Figure S4
Additional File 8: Code S4
Additional File 9: Table S1

Additional File 1: Figure S1



Additional File 2: Code S1

```
organizeCovariateColumns <- function(inputMatrix, covarColNums, covarColNames, doRenumber, doRename) {

  cnLen<-length(covarColNames)
  inputMatrix<-as.data.frame(inputMatrix)
  # does the column renaming and rearrangement
  if (doRenumber) {
    inputMatrix <- cbind(inputMatrix[, covarColNums], inputMatrix[, -covarColNums]) }
  if (doRename) {
    covDat<-as.data.frame(inputMatrix[, 1:cnLen])
    colnames(covDat) <- covarColNames
    inputMatrix<-data.frame(covDat, (inputMatrix[, -(1:cnLen)]))}
  return(inputMatrix)
}

# numCov is the number of covariates;
# dataSet is the ade4 output you are working with. It has k covariates and some number of taxa
# the program assumes that dataSet has column and row names
# out is the mapping file

mapFiler<-function(numCov, dataSet, fileName){

  # only printing covars
  dataSet = dataSet[,1:numCov]

  #unfactorize
  for (i in 1:numCov) {
    dataSet[,i] = as.character(unlist(dataSet[,i]))
  }

  rowCount = length(dataSet[,1])
  dataSet$SampID = seq(1, rowCount)
  numCov = numCov + 1

  SID<-as.vector(row.names(dataSet)) #get rownames

  # do column header
  headerRow = paste(c("#SampleID\tBarCodeSequence\tLinkerPrimerSequence", names(dataSet)), collapse="\t")

  fileConn<-file(fileName)
  writeLines(headerRow, fileConn)
  close(fileConn)
  out = rep("", rowCount)
  for (i in 1:rowCount) {
    name = paste(SID[i], "\t \t \t", sep="")
    covars = paste(dataSet[i,], collapse="\t")
    plus = paste(name, covars, sep="")
    out[i] = plus
  }

  write.table(out, file=fileName, append=T, row.names=F, col.names=F, quote=F)
}

ade4ToEmperor <- function(ade4Out, outputF, scaling=NULL,
                           type = c("samples", "species")){

  #Define default scalings
  if(is.null(scaling)){
    if (type == "samples") {scaling <- "l1"}
    else if (type == "species"){scaling <- "co"}
  }

  if (type == "samples"){
      if(scaling == "li") {tableF = ade4Out$li}
      else  {tableF = ade4Out$l1}
  }

  if (type == "species"){
      if(scaling == "c1") {tableF = ade4Out$c1}
      else  {tableF = ade4Out$co}
  }

  #define SVD outputs
  eigF = ade4Out$eig
  eigPF = eigF/sum(eigF)

  #Check number of eigenvectors agree in eigF, eigPF, tableF
  N_F <- length(eigF)
  N_PF <- length(eigPF)
  N_tableF <- dim(tableF)[2]

  #Keep the smallest number of eigenvalues in analysis
  Nkeep <- min(N_F, N_PF, N_tableF)

  #Format eigenvalues
  eigF <- eigF[1:Nkeep]
  E2 <- matrix(eigF,  dimnames = NULL) #remove headers
  nColE<-dim(E2)[1]  #ncols
  E2<-rbind(E2, "\n")
  E3<-toString(E2) #makes a string object
  E4<-gsub(",", "\t", E3) #removes all commas and replaces them with tabs
  E5<-gsub(" ", "", E4) #removes any space
  E6<-gsub("\t\n", "\n", E5)

  # 2) handles proportions file
  #EigProp <- read.csv(eigPF)
  #EP1 <- as.matrix(EigProp[-1])  #Test is a data frame-> matrix and remove index
  #EP2 <- matrix(EP1, dimnames = NULL) #remove headers
  eigPF <- eigPF[1:Nkeep]
  EP2 <- matrix(eigPF, dimnames = NULL) #remove headers
  nColEP<-dim(EP2)[1]  #ncols
  EP2<-rbind(EP2, "\n")
  EP3<-toString(EP2) #makes a string object
  EP4<-gsub(",", "\t", EP3) #removes all commas and replaces them with tabs
  EP5<-gsub(" ", "", EP4) #removes any space
  EP6<-gsub("\t\n", "\n", EP5)

  # 3) handles table file
  #Table <- read.csv(tableF)
  #P1 <- as.matrix(Table)  #Table is a data frame
  tableF <- tableF[,1:Nkeep]
  Table <- cbind(rownames(tableF), tableF)
  #P1 <- as.matrix(tableF)  #Table is a data frame
  #P2 <- matrix(P1, ncol = ncol(P1), dimnames = NULL) #remove headers
  P2 <- Table
  nRow<-dim(P2)[1]  #nrows need this later
  nCol<-dim(P2)[2]-1  #ncols need this later
  NL<-rep("\n", nRow) #each row must end in "new line"
  P3<-cbind(P2, NL) #data frame with newline at end or each row
  P3<-t(P3) #preparation for string
  P4<-toString(P3) #makes a string object
  P5<-gsub(",", "\t", P4) #removes all commas and replaces them with tabs
  P6<-gsub(" ", "", P5) #removes any space
  P7<-gsub("\t\n\t", "\n", P6)
  P8<-gsub("\t\n", "\n", P7)

  # 4) creates formatted output file
  sink(outputF)
  cat("Eigvals\t"); cat(nColE); cat("\n")
  cat(E6);cat("\n")
  cat("Proportion explained\t"); cat(nColEP); cat("\n")
  cat(EP6); cat("\n")
  cat("Species\t"); cat("0\t"); cat("0\n")
  cat("\n");
  cat("Site\t"); cat(nRow); cat("\t"); cat(nCol); cat("\n")
  cat(P8); cat("\n")
  cat("Biplot\t"); cat("0\t"); cat("0\n")
```

```
  cat("\n");
  cat("Site constraints\t"); cat("0\t"); cat("0\n")
  sink()
}

createSpeciesMappingFile <- function(taxaInfo, fileName) {
  #unfactorize
  for (i in 1:ncol(taxaInfo)) {
   taxaInfo[,i] = as.character(unlist(taxaInfo[,i]))
  }

  # do column header
  headerRow = paste(c("#SampleID\tBarCodeSequence\tLinkerPrimerSequence", names(taxaInfo)[-1]), collapse="\t")


  fileConn<-file(fileName)
  writeLines(headerRow, fileConn)
  close(fileConn)
  taxaInfoPlus = rep("", nrow(taxaInfo))
  for (i in 1:nrow(taxaInfo)) {
    sp_name = paste(taxaInfo[i,1], "\t \t \t", sep="")
    covars = paste(taxaInfo[i,-1], collapse="\t")
    plus = paste(sp_name, covars, sep="")
    taxaInfoPlus[i] = plus
  }

  write.table(taxaInfoPlus, file=fileName, append=T, row.names=F, col.names=F, quote=F)
}


# The function sortAndMapHandsFree takes the data frame (inputMatrix) and the dudi.pca object (ade4Out) and 
scaling ("li" or "l1").
# It requires file names for the mapping file and eigenvector file outputs.
# The output objects are for use in Emperor.
# It is not interactive.  In addition to the data frame and dudi.pca object, it also requires
#   the locations of the columns of the covariates : e.g.  covarColNums=c(1, 3, 8)
#   the names of the covariates: e.g. covarColNames=c("cov1", "cov2", "cov3") (default assumes they're already 
named)
# The default scaling for samples is "l1".  An alternative choice is "li".
# The default scaling for species is "co".  An alternative choice is "c1".

sortAndMapHandsFree <- function(inputMatrix, covarColNums = 0,
                      covarColNames = c(), ade4Out, eigenScaling=NULL,
                      type = c("species", "samples"),
                      mappingFile, eigenFile,
                      filteredTaxa=NULL,biplotFile=NULL) {

  if (type == "samples") {
   doRename = (length(covarColNames > 0))
   doRenumber = ! all(covarColNums == seq(1, max(covarColNums)))
   organized <- organizeCovariateColumns(inputMatrix, covarColNums, covarColNames, doRenumber, doRename)
   reorderedList = list(numberOfCovariates=length(covarColNums), orderedData=organized)
   mapFiler(reorderedList$numberOfCovariates, reorderedList$orderedData, mappingFile)
   ade4ToEmperor(ade4Out, eigenFile, eigenScaling, type)
   if (! is.null(filteredTaxa )) {
     makeBiPlotFile(filteredTaxa,biplotFile)
   }
  }
  else {
   createSpeciesMappingFile(inputMatrix, mappingFile)
   ade4ToEmperor(ade4Out, eigenFile, eigenScaling, type)
  }


  #Error check for sample names matching dudi output
  if (type == "samples"){
    if (! all(rownames(inputMatrix) == rownames(ade4Out$tab))) {
      print("Mismatched rownames in inputMatrix and ade4Out$tab.  You must fix this in order for the output to 
work in Emperor!")
```

```
    }
  }
}

makeBiPlotFile <- function(filteredTaxa,biplotFile) {
  filteredTaxa = t(filteredTaxa)
  topRow = c("# Constructed from OTU file")
  headerRow = c("#OTU ID", colnames(filteredTaxa))
  tworows = paste(topRow, paste(headerRow, collapse="\t"), sep="\n")

  fileConn<-file(biplotFile)
  writeLines(tworows, fileConn)
  close(fileConn)

  colnames(filteredTaxa) <- NULL

  batch1 = filteredTaxa[-nrow(filteredTaxa),]
  batch2 = tail(filteredTaxa, 1)

  write.table(batch1, biplotFile, append = TRUE, quote = FALSE, sep = "\t",
              eol = "\n", na = "NA", dec = ".", row.names = TRUE,
              col.names = FALSE)

  write.table(batch2, biplotFile, append = TRUE, quote = FALSE, sep = "\t",
              eol = "", na = "NA", dec = ".", row.names = TRUE,
              col.names = FALSE)

}
```

Additional File 3: Code S2

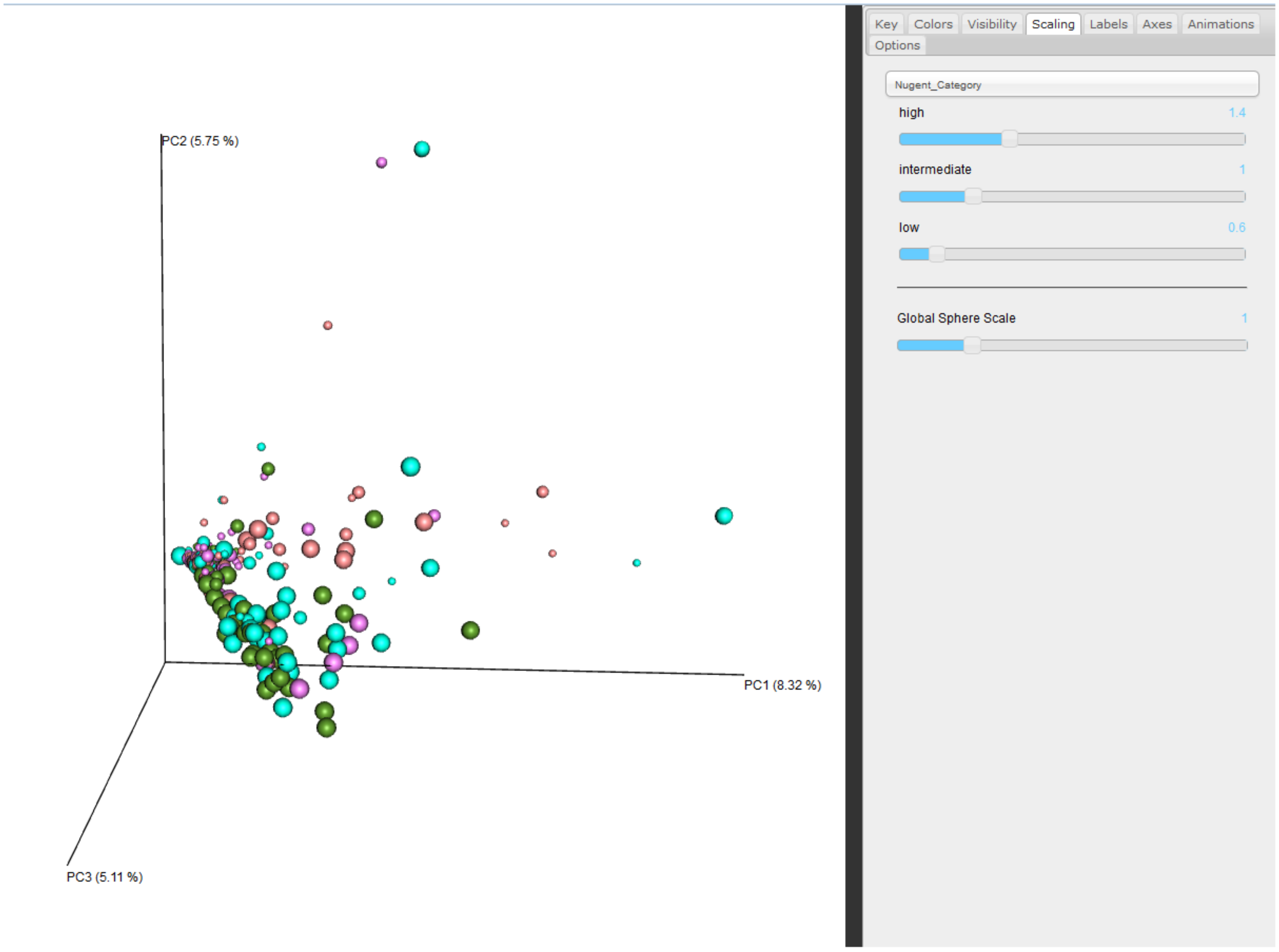
```
#setwd("~/Work/MicroBiome/Normalization/Code")

# screePlot makes plot of ordered eigenvalues to show percent of variance explained
# by the variables.
# Inputs:
#   First.pca is an ade4 ordination object
#   breaks is a vector of breakpoints (in percents (0, 100)) for the screeplot that
#   define intervals showing percent of variance explained.  The largest percent
#   entered in the vector will be considered the percent greater than which no
#   further cumulative variance will be calculated for the eigen values.  Example of
#   vector: c(25, 50, 75, 95)
#   mainTitle is optional text which will be appended to the title text which describes
#   the percent of variance explained.  Example: "Covariance PCA"

screePlot = function(First.pca, breaks, mainTitle=NULL) {
  library(Hmisc) # for all.is.numeric
  library(ggplot2) # for plotting

  isNum<-all.is.numeric(breaks) #tests that the elements are all numeric
  uBreaks<-unique(breaks) # removes duplicates
  uBLength<-length(uBreaks)  # vector length without duplicates
  orderedBreaks<-sort(uBreaks) # orders the regions

  outRange<-length(uBreaks[uBreaks > 100]) + length(uBreaks[uBreaks <= 0]) #checks that the values are between
zero and 100
  inrange<-(1-outRange>0)

  if (!isNum | !inrange ){
    stop("There's something wrong with the vector of values you entered.")
  }

  numBreaks = length(orderedBreaks)
  eigenVariance <- 100 * First.pca$eig/sum(First.pca$eig)
  cumulativeVariance <- cumsum(eigenVariance)
  threshold = max(orderedBreaks)
  cutoffIndex = which(cumulativeVariance == threshold)
  if (length(cutoffIndex) == 0) {
    cutoffIndex <- min(which(cumulativeVariance > threshold))
  }
  cumulativeVarianceKeep <- cumulativeVariance[1:cutoffIndex]

  Ind <- which(orderedBreaks < cumulativeVarianceKeep[1])
  if (length(Ind)>0 ){
    orderedBreaks <- orderedBreaks[-Ind]
  }

  VarExplained <- rep(0, cutoffIndex)
  VarExplained[which(cumulativeVarianceKeep < orderedBreaks[1])] <- paste("Less than ", orderedBreaks[1], "%",
sep="")

  nb<-numBreaks - 2
  for (i in 1:nb){
    a<-orderedBreaks[i]
    b<-orderedBreaks[i+1]
    VarExplained[which((cumulativeVarianceKeep >=a) & (cumulativeVarianceKeep < b))] <- paste(a, "%",  " to <",
b, "%", sep="")
  }

  VarExplained[which(cumulativeVarianceKeep >= orderedBreaks[nb+1])] <- paste(orderedBreaks[nb+1], "% to ",
threshold,"%", sep="")

  VarExplained <- factor(VarExplained, levels = unique(VarExplained))

  # number of eigenvalue per bin
  x_Breaks <- c(cumsum(table(VarExplained)))
  x_Labels <- c(as.character(cumsum(table(VarExplained))))
  plot_colors <- rainbow(numBreaks)
  df <- data.frame(1:cutoffIndex, First.pca$eig[1:cutoffIndex], cumulativeVarianceKeep, VarExplained)
  names(df) <- c("Rank",  "Value", "CumulativePercent", "VarExplained")
  if (is.null(mainTitle)) {
    mainTitle = ""
  }
  else {
    mainTitle = paste(mainTitle,"\n", sep="")
  }
  mainTitle = paste(mainTitle, "Largest", cutoffIndex, "of", length( First.pca$eig), "Eigenvalues\n Explain
",threshold,"Percent of Total Variance" )
  #Plot Eigenvalues
  p <- ggplot(df, aes(x = Rank, y = Value, fill = VarExplained)) +
    ggtitle(mainTitle) +
    ylab("Eigenvalue") +
    geom_bar(stat = "identity") + theme(panel.grid.minor = element_line(size=0.5))+
    scale_x_continuous("Rank of Eigenvalue",
                       limits = c(0,max(x_Breaks)+1),
                       breaks = x_Breaks, labels = x_Labels ) +
    theme(axis.text.x = element_text( face="italic", size = 12)) +
    scale_fill_manual(name = "Variance Explained",
                      values = plot_colors, breaks=levels(VarExplained)) + theme_bw() +
    theme( legend.position=c(.8,.7))

  return(p)
}
```

Additional File 4: Code S3

```
# functions for plotting the presence/absence and intensity plots
# for visualizing influence on variation by PC

# workflow:
# starting with a data frame of object projections on PCs,
# call getDistances() to turn projections into distances from origin
# call getDeciles or getTenths on distances to find rank indexes of objects by PC
#  getDeciles ranks by population density
#  getTenths ranks by "range", i.e. distance
# call plotPASD or plotIntensity to plot data frame of rank indexes
# or call makePCVisualizationPlot on projections to perform complete workflow in one step
#
# call makeCumulativeVariancePlot to create the associated cumulative variance explained plot

#rm(list=ls())
library(RColorBrewer)
library(colorspace)
getDistances <- function(df) {
  # takes a data frame of object projections
  # returns a new data frame with distances
  # by number of PCs used
  nr <- nrow(df)
  nc <- ncol(df)
  distances <- data.frame(abs(df[,1]))
  rownames(distances) <- rownames(df)
  for (i in 2:nc) {
    distances[,i] <- sqrt(rowSums((df[,1:i])^2))
  }
  colnames(distances) <- paste("distanceUsing", 1:nc, sep="")
  return(distances)
}

getDeciles <- function(df) {
  # takes a data frame of distances
  # by number of PCs used
  # returns a new data frame with decile indexes
  nr <- nrow(df)
  nc <- ncol(df)
  dec <- data.frame(rep(0,nr))
  rownames(dec) <- rownames(df)
  for (i in 1:nc) {
    ar<-rank(df[,i],ties.method = "first")
    dec[,i] <- cut(ar, quantile(ar,0:10/10), include.lowest=TRUE, labels=FALSE) # [),[), ...,[),[]
  }
  colnames(dec) <- paste("decileUsing", 1:nc, sep="")
  return(dec)
}

getTenths <- function(df) {
  # takes a data frame of distances
  # by number of PCs used
  # returns a new data frame with tenth indexes
  nr <- nrow(df)
  nc <- ncol(df)
  tenths <- data.frame(rep(0,nr))
  rownames(tenths) <- rownames(df)
  for (i in 1:nc) {
    ar<-rank(df[,i],ties.method = "first")
    tenths[,i] <- cut(ar, 10, include.lowest=TRUE, labels=FALSE) # [),[), ...,[),[]
  }
  colnames(tenths) <- paste("tenthUsing", 1:nc, sep="")
  return(tenths)
}

getRankLiers <- function(df, rank) {
 # takes a data frame of rank indexes and a rank of interest (1:10)
 # returns a new data frame with
 #  number of times in the rank
 #  max pcs used in rank

 nr <- nrow(df)
 nc <- ncol(df)

 liers <- data.frame(tally = rowSums(df[,] == rank))
 liers$lastOne <- 0

 for (i in 1:nrow(liers)) {
   ones <- which(df[i,] == rank)
   if (length(ones) > 0) {
    liers$lastOne[i] <- tail(ones, n=1)
   }
   else {
    liers$lastOne[i] <- 0
   }
 }

 return(liers)
}

plotPASD <- function(df, rank, covar=NULL, covarColors=NULL) {
  # takes a data frame of rank indexes and a rank and plots presence/absence

    # get dimensions
  nc <- ncol(df)
  liers <- getRankLiers(df,rank)
  nr <- nrow(liers)

  # Sorting:
  if (is.null(covar)) {
    # if no covariates are provided, sort distances by largest number of PCs used, then by sum of all times
    # used in a combination of PCs and assign one color
    liers$covar <- as.factor(rep("No covariates supplied", nr))
    numLevels <- 1
    covarColors <- c("blue")
    #order by max pcs in the decile/tenth, then sum of decile/tenth indicators
    liers <- liers[order(liers$lastOne, liers$tally),]
  }
  else {
    # if covariates are provided, assign up to 8 covariate colors.  Then order by covariates; then sort
distances by
    # largest number of PCs used, then by sum of all times used in a combination of PCs
    liers$covar <- as.factor(covar)
    numLevels <- nlevels(liers$covar)
    if (is.null(covarColors)) {
      if(numLevels > 8) {
        stop("Plot can only accomodate coloring up to 8 covar levels.")
      }
      covarColors <- brewer.pal(numLevels, "Dark2")
      #covarColors <- rainbow(numLevels)
    }
    #order by covar, then max pcs in the decile, then sum of decile indicators
    liers <- liers[order(liers$covar, liers$lastOne, liers$tally),]
  }

  liers <- liers[liers$tally > 0,]
  nr <- nrow(liers)

    # need to reverse order of groups, but not within groups
  # if (! is.null(covar)) {
  #  liers <- liers[order(nr:1),] #invert row order for printing
  # }

  # get the range for the x and y axis
  xrange <- c(1,nc)
  yrange <- c(1,nr)
  # set up the plot
  plot(xrange, yrange, type="n",  yaxt="n", ylim=yrange, xaxt="n", xlab="", ylab="")
  axis(1, at = seq(1, nc, by = 10))

  # Plotting
  for (j in 1:nr) {
    for (x in 1:nc) {
      if (df[row.names(liers)[j], x] == rank) {
        points(x, j, pch=15, col=covarColors[liers$covar[j]])
      }
      else {
        points(x, j, pch=15, col="gray96")
      }
    }
  }

  # for (i in 1:numLevels) {
  #   points(x=5, y=nr+(2*i), pch=15, col=covarColors[i])
  #   text(x=7, y=nr+(2*i), labels=rev(levels(liers$covar))[i], pos=4)
  # }
  legend("top", legend=levels(liers$covar), col=covarColors, pch=15, inset=c(0,-0.015), xpd=TRUE, horiz=TRUE,
bty="n")

  axis(2, at=1:nr, labels=row.names(liers), col.axis="blue", las=2, cex.axis=0.7)
}

plotIntensity <- function(df, covar=NULL, covarColors=NULL, plotOrder=NULL) {
  # takes a data frame of ranks indexes and plots them by intensity

  # get dimensions
  nc <- ncol(df)
  nr <- nrow(df)
  sums <- rowSums(df[,])

  if (is.null(covar)) {
    df$covar <- as.factor(rep("No covariates supplied", nr))
    numLevels <- 1
    #covarColors <- c("blue")

  }
  else {
    df$covar <- as.factor(covar)
    numLevels <- nlevels(df$covar)
    if (is.null(covarColors)) {
      if(numLevels > 8) {
        stop("Plot can only accomodate coloring up to 8 covar levels.")
      }
      covarColors <- brewer.pal(numLevels, "Dark2")
    }
  }

  if (is.null(plotOrder)) {
   # order by covar, then desc decile/tenth of max pcs, then decile/tenth sum
   plotOrder <- order(df$covar,-df[,nc], sums)
  }
  df <- df[plotOrder,]

  # need to reverse order of groups, but not within groups
  # if (! is.null(covar)) {
  #  df <- df[order(nr:1),] #invert row order for printing
  # }

  # get the range for the x and y axis
  xrange <- c(1,nc)
  #yrange <- c(1,nr+(2*numLevels))
  yrange <- c(1,nr)

  # set up color intensity levels
  # need unique color for each covar level
  # and 10 levels of intensity for each color
  colorIntensity <- data.frame(matrix(rep("", 10*numLevels), nrow=numLevels), stringsAsFactors = FALSE)
  if (is.null(covar)){
    colorIntensity[1,]=sequential_hcl(10, h = 264, c = c(90, 90), l = c(95, 20), power =0.8)
  }
  else {
    for (i in 1:numLevels) {
      covarRgb <- col2rgb(covarColors[i])/255
      colorIntensity[i,] <- rgb(covarRgb[1], covarRgb[2], covarRgb[3], alpha=(1:10 / 10))
    }
  }


  # set up the plot
  plot(xrange, yrange, type="n",  yaxt="n", ylim=yrange, xaxt="n", xlab="", ylab="")
  axis(1, at = seq(1, nc, by = 10))

  # plot data
  for (j in 1:nr) {
    for (x in 1:nc) {
      points(x, j, pch=15, col=colorIntensity[df$covar[j], df[j, x]])
    }
  }

  # add covar labels
  #for (i in 1:numLevels) {
  #  points(x=5, y=nr+(2*i), pch=15, col=covarColors[i])
  #  text(x=7, y=nr+(2*i), labels=rev(levels(df$covar))[i], pos=4)
  #}
  verticalOffset = 1/nr
  legend("top", legend=levels(df$covar), col=covarColors, pch=15, inset=c(0,-verticalOffset), xpd=TRUE,
horiz=TRUE, bty="n")

  axis(2, at=1:nr, labels=row.names(df), col.axis="blue", las=2, cex.axis=0.7)

  return(plotOrder)
}

makePCVisualizationPlot <- function(projections, type="presence", byRank="decile", atRank=1, plotTitle=NULL,
                                    xLabel="Number of PCs used", yLabel="Object", covar=NULL, covarColors=NULL,
                                    plotOrder=NULL, breaks=NULL) {
  # takes a data frame of projections and plots them according to parameters

  # check params
  if (!is.null(covar) & nrow(projections) != length(covar)) {
    stop("Covariate vector length must match number of objects.")
  }
  if (!is.null(covar) & !is.null(covarColors) & length(unique(covar)) != length(covarColors)) {
    stop("Covariate color vector length must match number of covariate levels.")
  }

  if (! type %in% c("presence","intensity")) {
    stop("Parameter 'type' must be 'presence' or 'intensity'")
  }

  if (! byRank %in% c("decile","tenth")) {
    stop("Parameter 'byRank' must be 'decile' or 'tenth'")
  }

  # get distances by number of PCs from projections
  mdistances <- getDistances(projections)

  # get rank indexes
  if (byRank=="decile") {
    rankIndexes <- getDeciles(mdistances)
    rankDescription <- "Decile"
  }
  else {
    rankIndexes <- getTenths(mdistances)
    rankDescription <- "Range"
  }

  # make plot
  newOrder <- NULL
  if (type=="presence") {
    plotPASD(rankIndexes, atRank, covar, covarColors)
    if (is.null(plotTitle)) {
      plotTitle <- paste(rankDescription, "-", atRank, " presence by number of PCs used", sep="")
    }
  }
  else {
    newOrder <- plotIntensity(rankIndexes, covar, covarColors, plotOrder)
    if (is.null(plotTitle)) {
      plotTitle <- paste(rankDescription, " intensity by number of PCs used", sep="")
    }
  }

  # add titles and axis labels
  if (is.null(xLabel)) {
    xLabel <- "Number of PCs used"
  }
  if (is.null(yLabel)) {
    yLabel <- "Object"
  }
  title(plotTitle, xlab=xLabel, ylab=yLabel)
  if (! is.null(breaks)) {
   abline(v=breaks)
  }
  return(newOrder)
}

makeCumulativeVariancePlot <- function(eigenValues, numPCs) {
  plot(c(1,numPCs), c(0,1), xaxt="n", type="n", main="", ylab="Cumulative Variance Explained", xlab="Eigen
Vector Rank")
  axis(1, at=c(1,seq(11, numPCs, 10)))
  points(cumsum(eigenValues[1:numPCs]/sum(eigenValues)))
}

# Counts to Props takes an OTU table and returns a table with proportions of each taxon for each sample
composition
CountstoProps<-function(OTU){
  cols<-ncol(OTU)
  den<-rowSums(OTU)
  den<-1/den
  Props = sweep(OTU,1,den,FUN="*")
  colnames(Props)<-colnames(OTU)
  rownames(Props)<-rownames(OTU)
  return(Props)
}

# evaluates if proportions of taxa in 2 tables grew (+1), shrank(-1), or did not change(0) from time1 to time2
TestDiffs<-function(Props1, Props2, CL=.8){
  cols<-ncol(Props1)
  shiftValue<-rep(0, cols)
  for(i in 1:cols){
    tDiff<-t.test(Props2[,i], Props1[,i], paired=TRUE,  conf.level = CL)
    if(tDiff$conf.int[2]<0){
      shiftValue[i] <- -1
    } else {
      if(tDiff$conf.int[1]>0){
        shiftValue[i] <- 1
      } else {
        shiftValue[i] <- 0
      }
    }
  }
  names(shiftValue)<-colnames(Props1)
  return(shiftValue)
```

```
}


plotIntensityCountsdiff <- function(df, df2, counts, counts2, title) {

  # get dimensions
  nc <- ncol(df)
  nr <- nrow(df)
  sums <- rowSums(df[,])

  # order by covar, then desc decile/tenth of max pcs, then decile/tenth sum
  plotOrder <- order(-df[,nc], sums)
  df <- df[plotOrder,]

  # get the range for the x and y axis
  xrange <- c(1,nc)
  yrange <- c(1,3*nr)

  # set up color intensity levels
  # need baseline plus 2 colors for above/below baseline
  # and 10 levels of intensity for each color
  colorIntensity <- data.frame(matrix(rep("", 30), nrow=3), stringsAsFactors = FALSE)

  colorIntensity[1,] <- sequential_hcl(10, h = 260, c = c(10, 0), l = c(90, 30), power = 1.5)  #gray

  # use this if "after" is more spread out
  colorIntensity[2,] <- sequential_hcl(10, h = 280, c = c(70, 40), l = c(90, 30), power = 1.5) #purple

  # use this if "after" is less spread out
  colorIntensity[3,] <- sequential_hcl(10, h = 100, c = c(80, 40), l = c(90, 30), power = 1.5) #green

  # set up the plot
  plot(xrange, yrange, type="n",  yaxt="n", ylim=yrange, xaxt="n", xlab="", ylab="", main=title)
  axis(1, at = seq(1, nc, by = 10))

  # plot data
  for (j in 1:nr) {
    rname = rownames(df)[j]
    range1 = max(counts[,rname]) - min(counts[,rname])
    range2 = max(counts2[,rname]) - min(counts2[,rname])
    ypos = 3*j - 2
    for (x in 1:nc) {
      if (range1 <= range2) {
        points(x, ypos, pch=15, col=colorIntensity[2, df2[rname, x]])
      }
      else {
        points(x, ypos, pch=15, col=colorIntensity[3, df2[rname, x]])
      }
      points(x, ypos+1, pch=15, col=colorIntensity[1, df[j, x]])
    }
  }

  axis(2, at=seq(2, 3*nr, by=3), labels=row.names(df), col.axis="blue", las=2, cex.axis=0.7)

}



plotIntensityPropsdiff <- function(df, df2, counts, counts2, title, plotOrder=NULL) {

  # get dimensions

  nc <- ncol(df)
  nr <- nrow(df)
  sums <- rowSums(df[,])

  props1<-CountstoProps(counts)
  props2<-CountstoProps(counts2)
  shiftValues<-TestDiffs(props1, props2)

  # order by covar, then desc decile/tenth of max pcs, then decile/tenth sum
  if (plotOrder == NULL) {
   plotOrder <- order(-df[,nc], sums)
  }
  df <- df[plotOrder,]

  # get the range for the x and y axis
  xrange <- c(1,nc)
  yrange <- c(1,3*nr)

  # set up color intensity levels
  # need baseline plus 2 colors for above/below baseline
  # and 10 levels of intensity for each color
  colorIntensity <- data.frame(matrix(rep("", 30), nrow=3), stringsAsFactors = FALSE)

  colorIntensity[1,] <- sequential_hcl(10, h = 0, c = c(0, 0), l = c(85, 15), power = 1)  #gray

  # use this if "after" is more spread out
  colorIntensity[2,] <- sequential_hcl(10, h = -3, c = c(20, 80), l = c(90, 30), power = .7) #red

  # use this if "after" is less spread out
  colorIntensity[3,] <- sequential_hcl(10, h = 203, c = c(10, 80), l = c(90, 20), power = .5) #blue

  # set up the plot
  plot(xrange, yrange, type="n",  yaxt="n", ylim=yrange, xaxt="n", xlab="", ylab="", main=title)
  axis(1, at = seq(1, nc, by = 10))

  # plot data
  for (j in 1:nr) {
    rname = rownames(df)[j]
    ypos = 3*j - 2
    for (x in 1:nc) {
      if (shiftValues[rname] == 1) {
        points(x, ypos, pch=15, col=colorIntensity[2, df2[rname, x]])
      }
      else if (shiftValues[rname] == -1){
        points(x, ypos, pch=15, col=colorIntensity[3, df2[rname, x]])
      }
      else {
        points(x, ypos, pch=15, col=colorIntensity[1, df2[rname, x]])
      }
      points(x, ypos+1, pch=15, col=colorIntensity[1, df[j, x]])
    }
  }

  axis(2, at=seq(2, 3*nr, by=3), labels=row.names(df), col.axis="blue", las=2, cex.axis=0.7)

}

#
# # test commands
# md <- getDistances(Prop.pca$co)
# mdec1 <- getDeciles(md[1:25,])
# mdec2 <- getDeciles(md[26:50,])
# rownames(mdec2) <- rownames(mdec1)
# counts1 <- OTU_Counts[1:25,]
# counts2 <- OTU_Counts[26:50,]
#
# #plotPASD(mrng[1:50,], 1)
#
#
# old.par <- par(no.readonly=TRUE)
# par(oma = c(8, 20, 1,20))
# png(filename="testStack.png",
#     type="cairo",
#     units="in",
#     width=18,
#     height=max(ceiling(0.25 * nrow(Prop.pca$co)), 7),
#     pointsize=16,
#     res=96)
# par(mar=c(5,12,4,8)+0.1,mgp=c(8,1,0))
# plotIntensity2(mdec1, mdec2, counts1, counts2, "test stacks")
# #makePCVisualizationPlot(Prop.pca$l1[1:50,], type="intensity", atRank=1, byRank="decile",
covar=data$Ethnic_Group[1:50], breaks=c(2,25,35,50))
# #makePCVisualizationPlot(Prop.pca$l1[1:50,], type="presence", atRank=1, byRank="decile", covar=NULL,
breaks=c(2,25,35,50))
# dev.off()
# par(old.par)


# mrng <- getTenths(md)
#
# dForRangeTest <- data.frame(d1=c(2,22,32,42,52), d2=c(53,43,33,13,3), d3=c(4,4,4,44,44),
d4=c(11,111,11,111,11))
# getTenths(dForRangeTest)
#
# plotPASD(mdec[1:50,], 1, covar=data$Ethnic_Group[1:50])
# plotPASD(mrng[1:50,], 1, covar=data$Ethnic_Group[1:50])
# plotPASD(mrng[1:50,], 1, covar=NULL)
# plotIntensity(mdec[200:250,], covar=data$Nugent_Category[200:250])
# plotIntensity(mrng[1:50,])


#makePCVisualizationPlot(Prop.pca$l1[1:50,], type="intensity", atRank=1, byRank="decile",
covar=data$Ethnic_Group[1:50], breaks=c(2,25,35,50))
# makePCVisualizationPlot(Prop.pca$l1[1:50,], type="intensity", byRank="decile", covar=data$Ethnic_Group[1:50])
# makePCVisualizationPlot(Prop.pca$l1[1:50,], type="presence", atRank=1, byRank="tenth",
covar=data$Ethnic_Group[1:50])
# makePCVisualizationPlot(Prop.pca$l1[1:50,], type="intensity", byRank="tenth", covar=data$Ethnic_Group[1:50])
```

Additional File 5: Figure S2



Nugent_Category

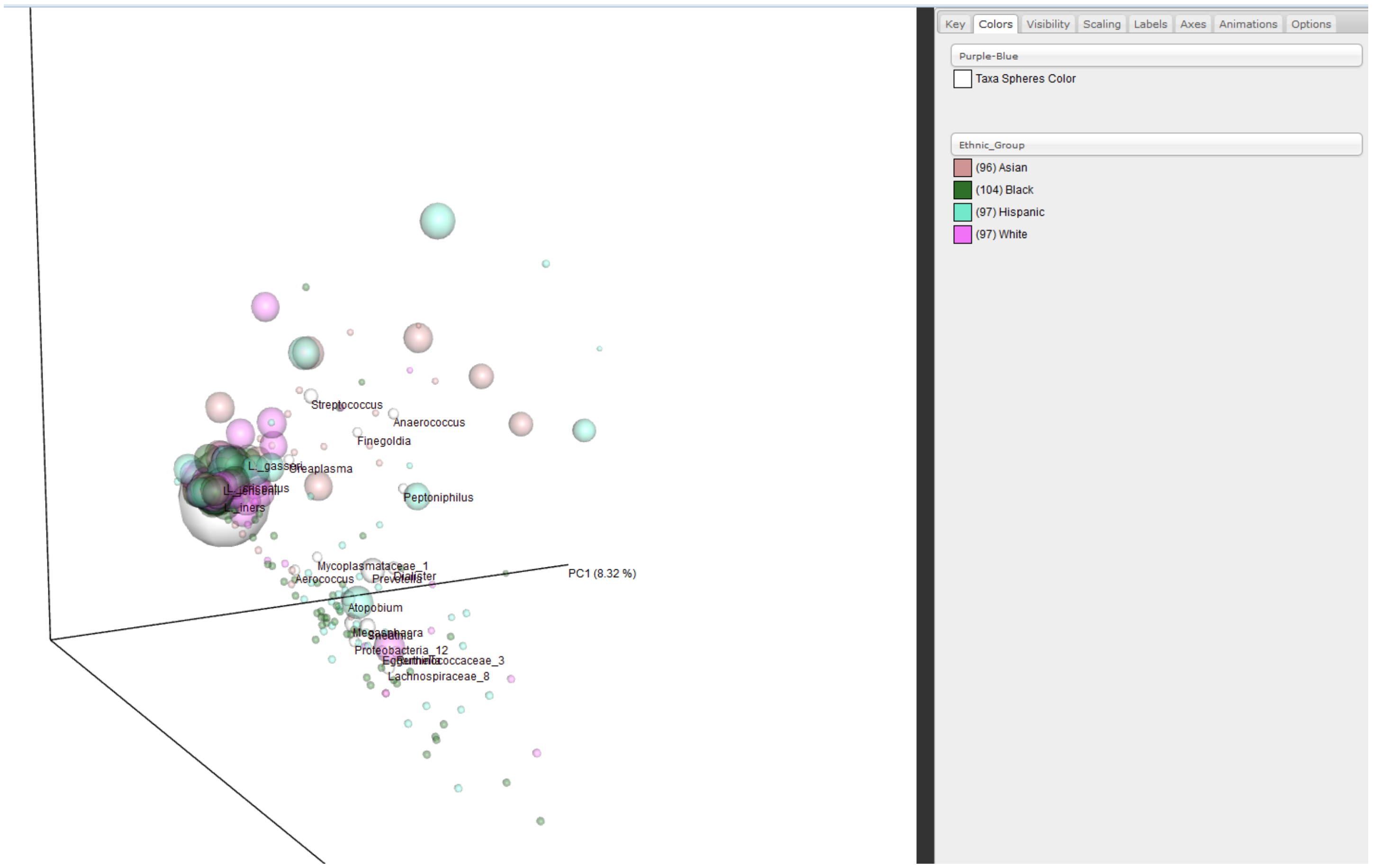
high 1.4

intermediate 1

low 0.6

Global Sphere Scale 1

Additional File 6: Figure S3



Purple-Blue

☐ Taxa Spheres Color

Ethnic_Group

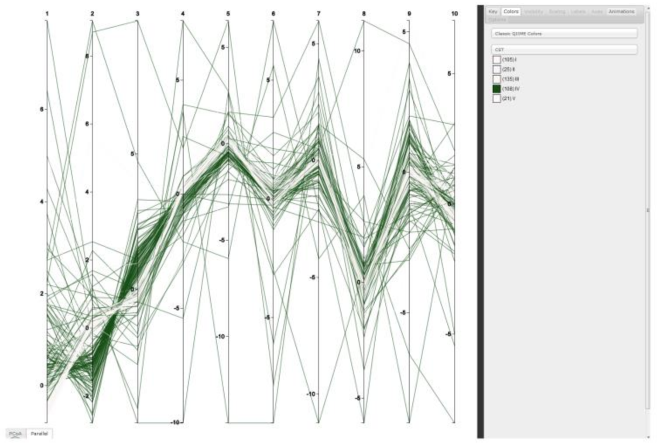
🟫 (96) Asian

🟩 (104) Black

🟦 (97) Hispanic

🟪 (97) White

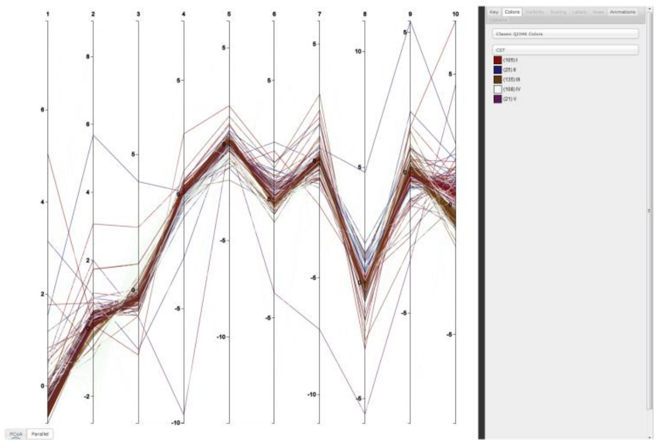
Additional File 7: Figure S4



Classic QIIME Colors

CST

- ☐ (105) I
- ☐ (25) II
- ☐ (135) III
- 🟩 (108) IV
- ☐ (21) V



Classic QIIME Colors

CST

- 🟥 (105) I
- 🟦 (25) II
- 🟫 (135) III
- ☐ (108) IV
- 🟪 (21) V

Additional File 8: Code S4

```
CoinertiaPlot <- function(coin,
                  Quant = 0.9, Prop.Var = 0.9,
                  Env_Var=NULL, Env_Var2 = NULL, color=NULL, shape=NULL,
                  PtColor= "magenta",PtShape=4, PtSize=2,
                  linetype=2, LblSize=2, LabelsOpt = NULL,
                  ArrLen=0.15, ArrAngle=20, ArrColor = "purple", axes = c(1,2)){


#Plot scores from each data set
XLim <- c(min(coin$mX$NorS1, coin$mY$NorS1),
          max(coin$mX$NorS1, coin$mY$NorS1))

YLim <- c(min(coin$mX$NorS2, coin$mY$NorS2),
          max(coin$mX$NorS2, coin$mY$NorS2))
#Get summary of distances between the two co-inertia sets
  Inert <- round(cumsum(coin$eig/sum(coin$eig)),6)
  percvar <- round((100 * coin$eig/sum(coin$eig))[axes],2)
#Find distance based on the first k axes that explain more that Prop.Var of Inertia
  k <- min(which(Inert > Prop.Var))
  k <- min(k, dim(coin$mY)[2])#min of var explained and number of axes kept
  Dist <- coin$mX[,1:k] - coin$mY[,1:k]
  Length <- sqrt(Dist$NorS1^2 + Dist$NorS2^2)
     if(all(substr(rownames(coin$mX), 1,1) == "X")){Char.rm <- TRUE}
     else{Char.rm <- FALSE}
  names(Length) <- rownames(coin$mX)
     if(Char.rm == TRUE){names(Length) <- substring(names(Length), first = 2)}
  MidPoint <- (coin$mX + coin$mY)/2
  Quantiles <- unique(sort(c(seq(0,1,0.25), seq(0,1, 0.1))))
  Summary <- as.data.frame(matrix(quantile(Length, probs = Quantiles), nrow = 1))
  names(Summary) <- paste("Qu_", Quantiles, sep = "")
  length(Length[Length > Summary$Qu_0.9])
#label only the points that are larger that 0.9th quantile
  QPlot <- quantile(Length, probs = Quant)
  Ind  <- which(Length > QPlot)

  Labels <- rownames(coin$mX)[Ind]
      if(Char.rm == TRUE) {Labels <- substring(Labels, first = 2)}
      if(!is.null(LabelsOpt)){Labels <- LabelsOpt[Ind]}
  df3 = data.frame(cbind(MidPoint$NorS1[Ind],  MidPoint$NorS2[Ind]), Labels)
  names(df3)[1:2] <- names(MidPoint)[axes]
#From here we change to a ggplot
  x = colnames(coin$mY)[1]
  y = colnames(coin$mY)[2]
#Create a table identifier
  Table <- c(rep(1, nrow(coin$mX)), rep(2, nrow(coin$mX)))
  df <- rbind(coin$mX, coin$mY)
  df <- cbind(df, Table)


  x = colnames(df)[1]
  y = colnames(df)[2]


  ord_map = aes_string(x = x, y = y)

  p <- ggplot(df, ord_map) + geom_point(na.rm = TRUE, colour = PtColor, shape = PtShape,size = PtSize) +
theme_bw()


#Check for additional plotting variables to color arrows
    #Save df names
  df2 <- cbind(coin$mX[,axes], coin$mY[,axes])
  names(df2) <- c("xbeg", "ybeg", "xend", "yend")
  df_Names <- names(df2)

  if(!is.null(Env_Var)){
    if(!is.null(color)){
      Var_Col <- which(names(Env_Var) == color)
      df2 <- data.frame(df2, Env_Var[,Var_Col])
      names(df2) <- c(df_Names,color)}

    if (!is.null(shape)) {
      Var_Shape <- which(names(Env_Var) == shape)
      df2 <- data.frame(df2, Env_Var[,Var_Shape])
      names(df2)[ncol(df2) ]<- shape
    }

  }#end if not null Env_Var


  ord_map_Arr = aes_string(x = "xbeg", y = "ybeg", xend = "xend", yend="yend",
                     color = color, shape = shape,
                     na.rm = TRUE)
  p = p+ geom_segment(ord_map_Arr, data = df2,
                    arrow = arrow(angle = ArrAngle, length = unit(ArrLen, "inches")),
                    linetype=linetype, col = ArrColor)

#Add percent var explained by axes on plot
  Ax1 <- paste("Axis", axes[1])
  Ax2 <- paste("Axis", axes[2])
  strivar = c(Ax1, Ax2)
  strivar = paste0(strivar, "   [", percvar, "%]")
  p = p + xlab(strivar[1]) + ylab(strivar[2])

#Finally add labels with
  p= p+annotate("text", x=df3$NorS1, y=df3$NorS2, label= df3$Labels, size = LblSize)

#Produce dissimilarity graph for samples data based on collection days
Dissimilarity <- data.frame(names(Length), Length)
names(Dissimilarity) <-c("Name", "Dissimilarity")
       if(!is.null(LabelsOpt)){Dissimilarity <- cbind(Dissimilarity,LabelsOpt) }
DistQuant <- rep(NA, length(Length))
              for (i in 1:(length(Summary)-1)){
                  Ind <- Length >=Summary[,i] & Length < Summary[,i+1]
                  DistQuant[Ind] <- Quantiles[i]
                          if(i == (length(Summary)-1)) {DistQuant[Length >= Summary[,i+1]] <- Quantiles[i+1]}
              }#end for
Dissimilarity <- cbind(Dissimilarity, DistQuant)
names(Dissimilarity)[dim(Dissimilarity)[2]] <- "Quantile"

     if(!is.null(Env_Var$CollectionDays)){
         TimeDiff <- Env_Var$CollectionDays - Env_Var2$CollectionDays
         BV_Status <- Env_Var$bv
         #DistQuant <- rep(NA, length(Length))
              #for (i in 1:(length(Summary)-1)){
                  #Ind <- Length >=Summary[,i] & Length < Summary[,i+1]
                  #DistQuant[Ind] <- Quantiles[i]
                          #if(i == (length(Summary)-1)) {DistQuant[Length >= Summary[,i+1]] <- Quantiles[i+1]}
              #}#end for

        #Dissimilarity <- data.frame(names(Length), Length, TimeDiff,BV_Status, DistQuant)
         Dissimilarity <- cbind(Dissimilarity, TimeDiff,BV_Status)
         last <- dim(Dissimilarity)[2]
         names(Dissimilarity)[(last-1):last] <- c( "TimeBtwVisits","BV")
         names(Dissimilarity)[1] <- "repeat_code"
     }#end if(!is.null(Env_Var$CollectionDays)
Dissimilarity <- Dissimilarity[order(Dissimilarity$Dissimilarity), ]

return(list(Summary = Summary,  Dissimilarity = Dissimilarity, p=p))
}#End of Function CoinertiaPlot

###############################################################################
PlotDissimilarity <- function(Dissimilarity,Title="", label = "point",
                          plot_colors =c("red",  "blue", "green")){
  #browser()
```

```r
A <- abs(Dissimilarity$TimeBtwVisits[Dissimilarity$Quantile < 0.5])
Ind <- Dissimilarity$Quantile >= 0.5 & Dissimilarity$Quantile <0.9
B <- abs(Dissimilarity$TimeBtwVisits[Ind])
C <- abs(Dissimilarity$TimeBtwVisits[Dissimilarity$Quantile >= 0.9])

summary(A)
summary(B)
summary(C)

DissimilTime <- c(A,B,C)

DissimilQuantile <- c(rep("Less_Than_Median", length(A)),
          rep("Greater_Than_Median", length(B)),
          rep("Greater_Than_Q90", length(C)))


A <- as.character(Dissimilarity$BV[Dissimilarity$Quantile < 0.5])
B <- as.character(Dissimilarity$BV[Ind])
C <- as.character(Dissimilarity$BV[Dissimilarity$Quantile >= 0.9])

BV_Status <- c(A,B,C)

A <- as.character(Dissimilarity$repeat_code[Dissimilarity$Quantile < 0.5])
B <- as.character(Dissimilarity$repeat_code[Ind])
C <- as.character(Dissimilarity$repeat_code[Dissimilarity$Quantile >= 0.9])

repeat_code <- c(A,B,C)

df <- data.frame(repeat_code, DissimilTime, DissimilQuantile, BV_Status)
df$BV_Status <- as.factor(df$BV_Status)
df$DissimilQuantile <- factor(df$DissimilQuantile)
#diamonds$cut <- factor(diamonds$cut, levels = rev(levels(diamonds$cut)))
ds <- ddply(df, .(DissimilQuantile), summarise, mean = mean(DissimilTime), sd = sd(DissimilTime))
myColors <- brewer.pal(length(levels(df$BV_Status)),"Set1")
names(myColors) <- levels(df$BV_Status)
colScale <- scale_colour_manual(name = "BV_History",values = myColors)

Groups <- as.character(df$DissimilQuantile)
Levels <- unique(df$DissimilQuantile)
Lab1 <- which(df$DissimilQuantile == Levels[1])
Lab2 <- which(DissimilQuantile == Levels[2])
Lab3 <- which(df$DissimilQuantile == Levels[3])
Groups[Lab1] <- "G1"
Groups[Lab2] <- "G2"
Groups[Lab3] <- "G3"

df <- data.frame(df, Groups)
df$Groups <- as.factor(df$Groups)

Groups <- c("G2", "G3", "G1")
ds <- data.frame(ds, Groups)
df$BV_Status <- factor(df$BV_Status, levels = c("Yes", "No", "NS"))
if(label == "text"){
Plot <- ggplot(df,aes(x =Groups,  y=DissimilTime, color = BV_Status)) + geom_text(aes(x = DissimilQuantile, y =
DissimilTime, label=repeat_code, color = BV_Status), size = 2, data=df, parse = T)}

else{Plot <- ggplot(df,aes(x =Groups,  y=DissimilTime, color = BV_Status)) + geom_point(stat="identity") +
geom_point(data = ds, aes(y = mean), colour = 'black', size = 3)}

Levels <- unique(DissimilQuantile)
Lab1 <- which(DissimilQuantile == Levels[1])
Lab2 <- which(DissimilQuantile == Levels[2])
Lab3 <- which(DissimilQuantile == Levels[3])
Labels <- rep(0, length(DissimilQuantile))
Labels[Lab1] <- "Q<50"
Labels[Lab2] <- "50 < Q < 90"
Labels[Lab3] <- "Q>90"
#browser()
#browser()
```

```
Cols <- c("Yes" = plot_colors[1], "No" = plot_colors[2],"NS" = plot_colors[3])
Brks <- c("Yes", "No", "NS")
Labs <- c("Yes", "No", "NS")

Plot <- Plot + ggtitle(Title) + scale_x_discrete(name = "Quantile") + scale_y_continuous(name = "Time Between
Visits") +scale_colour_manual(name = "BV History", values = Cols, breaks = Brks, labels = Labs)
#ggsave(file = paste(path, name, ".pdf", sep = ""), Plot)

return(Plot)
}
###################################################################
PlotCW <- function(coin, name, path, color = "red",
            Title1 = "Canonical Weights for the First Visit",
            Title2 = "Canonical Weights for the Second Visit",
            Labels1 = NULL, Labels2 = NULL, scale = FALSE){
    #browser()
  library(grid)
  library(gridExtra)

  if(is.null(Labels1)){Labels1 <- 1:nrow(coin$co)}
  if(is.null(Labels2)){Labels2 <- 1:nrow(coin$li)}

  x = colnames(coin$co)[1]
  y = colnames(coin$co)[2]

  ord_map = aes_string(x = x, y = y)
  df <- data.frame(coin$co$Comp1, coin$co$Comp2,
          rep(0, length(coin$co$Comp1)), rep(0, length(coin$co$Comp2)),
          Labels1)
  rownames(df) <- rownames(coin$co)
  names(df) <- c(x,y, "x_0", "y_0", "Labels1")
  arrows_map =  aes_string(x = "x_0", y = "y_0",
                xend = x, yend = y)
    #browser()
CW_X <- ggplot(coin$co, ord_map) + geom_point(colour = color)
CW_X <- CW_X + scale_x_continuous(name="") + scale_y_continuous(name="")
CW_X <- CW_X + geom_segment(data = df, mapping = arrows_map ,size = 0.4,
            linetype = 1, colour = color,
            arrow = arrow(angle = 30, length = unit(0.25, "cm")))

#Last thing: Fix order of labels
lbl_map = aes_string(x = x, y = y, label = "Labels1")
CW_X <- CW_X + geom_text(data = df, mapping = lbl_map, size = 5)
CW_X <- CW_X + ggtitle(Title1) +theme(plot.title = element_text(size = 7, colour = "black"),
                 panel.background = element_rect(fill = "white"),
                 panel.grid.major = element_line(colour = "grey90"))


#Canonical Weight for the second data set
x = colnames(coin$li)[1]
y = colnames(coin$li)[2]
ord_map = aes_string(x = x, y = y)
df <- data.frame(coin$li$Axis1, coin$li$Axis2,
                 rep(0, length(coin$li$Axis1)), rep(0, length(coin$li$Axis2)),
                 Labels2)
rownames(df) <- rownames(coin$li)
names(df) <- c(x,y, "x_0", "y_0", "Labels2")
arrows_map =  aes_string(x = "x_0", y = "y_0",
                         xend = x, yend = y)

CW_Y <- ggplot(coin$li, ord_map) + geom_point(colour = color)
CW_Y <- CW_Y + scale_x_continuous(name="") + scale_y_continuous(name="")
CW_Y <- CW_Y + geom_segment(data = df, mapping = arrows_map,
                            size = 0.4, linetype = 1, colour = color,
                            arrow = arrow(angle = 30, length = unit(0.25, "cm")))
lbl_map = aes_string(x = x, y = y, label = "Labels2")
CW_Y <- CW_Y + geom_text(data = df, mapping = lbl_map, size = 5)
CW_Y <- CW_Y + ggtitle(Title2) +theme(plot.title = element_text(size = 6, colour = "black"),
                                      panel.background = element_rect(fill = "white"),
                                      panel.grid.major = element_line(colour = "grey90"))

#browser()

if(scale == TRUE){

  xmin <- min(coin$co$Comp1, coin$li$Axis1)
  xmax <- max(coin$co$Comp1, coin$li$Axis1)

  ymin<- min(coin$co$Comp2, coin$li$Axis2)
  ymax<- max(coin$co$Comp2, coin$li$Axis2)

  CW_X <- CW_X + scale_y_continuous(name="", limits=c(ymin,ymax)) +
         scale_x_continuous(name="", limits=c(xmin,xmax))

  CW_Y <- CW_Y + scale_y_continuous(name="", limits=c(ymin,ymax)) +
    scale_x_continuous(name="", limits=c(xmin,xmax))
}

pushViewport(viewport(layout = grid.layout(1, 2)))
print(CW_X, vp = viewport(layout.pos.row = 1, layout.pos.col = 1))
print(CW_Y, vp = viewport(layout.pos.row = 1, layout.pos.col = 2))

name2 <- paste(name, "_CW", sep = "")
pdf(paste(path, name2, ".pdf", sep = ""))
grid.arrange(CW_X, CW_Y, ncol=2,nrow=1)
dev.off()
 return(list(CW_1 = CW_X, CW_2 = CW_Y))
}
```

Additional File 9: Table S1

"Average" taxa

	Ethnicity			
Taxon	Black (4)	White (8)	Asian (12)	Hispanic (16)
Lactobacillales 5	4	8	12	15
Lactobacillales 2	4	8	12	13
L. iners	4	8	12	14
Ureaplasma	4	3	7	6
L. crispatus	3	5	6	9
L. vaginalis	2	4	5	8
L. jensenii	3	4	5	8
Streptococcus	1	3	2	5
L. gasseri	3	3	1	5
Corynebacterium	0	1	4	6
Finegoldia	0	3	3	5
Anaerococcus	0	4	2	4
Prevotella	0	2	2	7
Lactobacillales 6	2	3	0	4
Veillonella	1	1	0	2
Staphylococcus	1	0	5	4
Campylobacter	0	1	1	0
Ruminococcaceae 4	0	1	1	0
Dialister	0	2	0	5
Peptostreptococcus	0	1	0	4
Peptoniphilus	0	2	0	4
Atopobium	0	1	0	4
Porphyromonas	0	1	0	3
Bacteroides	0	2	0	2
Actinomyces	0	0	1	1
Sneathia	0	0	1	2
Lactobacillales 1	0	0	1	0
Facklamia	0	0	1	0
Exiguobacterium	0	0	1	0
Lachnospiraceae Incertae Sedis	0	0	1	0
Parvimonas	0	0	0	3
Megasphaera	0	0	0	3
Lachnospiraceae 7	0	0	0	2
Lactobacillus 4	0	0	0	2
Fusobacterium	0	0	0	2
Gemella	0	0	0	2
Clostridiales 17	0	0	0	2
Flavobacteriaceae 4	0	0	0	1
Peptococcus	0	0	0	1
Segniliparus	0	0	0	1
Bacteroidales 1	0	0	0	1
Lactobacillus 2	0	0	0	1
Gardnerella	0	0	0	1
Eggerthella	0	0	0	1
Lactobacillus 3	0	2	0	0
Aerococcus	0	2	0	0
Propionibacterium	0	1	0	0

Table 1: Taxa that appear within microbiomes of the 40 women closest to the origin based on 61 principal components and sorted by ethnicity.