pca_score.R

sandi

Wed Aug 29 12:33:23 2018

Functions for plotting pca results in score plot @param pca_data the PCA model output of prcomp() @param PCs numeric; vector of Principal Components to consider @param colour string; name of column to be used as the colour aesthetic @param shape string; name of column to be used as the shape aesthetic @param size string; name of column to be used as the size aesthetic @param label string; name of column to be used as the point labels @param pt.size numeric; default 3; size of points @param CIgroup' draws an ellipse based on covariance matrix at the 95% confidence level around the specified group (one of the metadata columns) specifies the groupings around which ellipses should be drawn

@return list of: 'p' ggplot object with generated plot 'collected_data' all the data used to plot pca data using ggplot2. can use collected_data for further customization using ggplot

```
pca_score = function(pca_data, PCs=1:2,
                     colour=NULL, shape=NULL, size=NULL, label=NULL, title=NULL,
                     legend.ori='vertical', pt.size=3, outline=FALSE,
                     CIgroup = NULL)
{
  #Input testing-------------------------------------------------------------------

  #Testing if the pca_data is indeed the result of a prcomp function
  if (class(pca_data) != 'prcomp')
  {
    stop('Input "pca_data" to pca_score must be the output of prcomp() function')
  }

  #Testing if PCs are numeric
  if (!is.numeric(PCs))
  {
    stop('Input "PCs" to pca_score must be a vector of numeric values corresponding to target principal
  }

  #Testing to make sure that PCs are in the input data
  if (any(PCs > ncol(pca_data$x)))
  {
    stop('Input "PCs" to pca_score must not be outside of the pca_data principal component range')
  }

  #Extract scores------------------------------------------------------------------
  score.data = as.data.frame(pca_data$x)

  #Copying out the standard deviation info from pca_data as a dataframe
  variance = as.data.frame(pca_data$sdev)

  #Converting standard deviations to variance
  variance = variance**2
  variance = variance/sum(variance)*100
  variance = round(variance, digits=1)
  #Initializing a new vector to store collected.data
  collected.data = c()
```

```
#Iterating through the PCs, two at a time.
for (i in seq(PCs[1], PCs[length(PCs)], by=2))
{
  view.data = score.data[score.data$x.PC==i,]
  view.data$view.var = paste('PC', i, '(', variance[i,], '%) ','vs.',
                             'PC', i+1,'(', variance[i+1,], '%)')

  #Adding to collected.data
  collected.data = rbind(collected.data, view.data)
}

#Converting View to factor, preserving ordering
collected.data$view.var = factor(collected.data$view.var,
                                 levels=unique(collected.data$view.var))

#Iterating through the PCs, two at a time, to draw ellipse for 95% confidence

ellipse_custom <- function(d, x='x.value', y='y.value', axes='view')
{

  df <- d[d$view == unique(d[,axes]),]

  xvar <- var(df[,x])
  yvar <- var(df[,y])
  cvar <- cov(df[,x], df[,y])

  cen <- c(mean(df[,x]), mean(df[,y]))

  cov.m <- matrix(c(xvar, cvar, cvar, yvar), ncol=2, byrow=TRUE)
  e <- ellipse(cov.m, centre=cen)
  return(e)
}

elp.data <- c()
if (!is.null(CIgroup))
{
  for(i in seq(PCs[1], PCs[length(PCs)], by=2))
  {
    elp <- plyr::ddply(collected.data[collected.data$x.PC==i,],
                CIgroup, ellipse_custom)

    #putting each elp interation into one dataframe
    elp.data <- rbind(elp.data, elp)
  }
}

#Calculating plot limits
xlimit = max(abs(collected.data$x.value))*1.15
ylimit = max(abs(collected.data$y.value))*1.1

#Generating plot-----------------------------------------------------------------

p = ggplot(collected.data)
```

```
#If size descriptor has been specified, use it as an aesthetic,
 if(!is.null(size)) {
  p = p + geom_point(aes_string(x='x.value', y='y.value',
                          colour=colour, shape=shape, size=size), alpha=0.8)
}
else
{ #otherwise, default to point size 3 and make points with black outline
  if(outline==TRUE){
    p = p + geom_point(aes_string(x='x.value', y='y.value', shape=shape),
                       colour='black', size=(pt.size+0.5), alpha=0.8)

    #     #Alternative script for making points with black outline
    #     #though there is internal bug with legend
    #     p = p + geom_point(aes_string(x='x.value', y='y.value',
    #                                   fill=colour, shape=shape),
    #                        colour='black', size=pt.size, alpha=0.8)
    #     p = p + scale_shape_manual(values=21:25)
  }

  p = p + geom_point(aes_string(x='x.value', y='y.value',
                                colour=colour, shape=shape),
                     size=pt.size, alpha=0.8)

}

#Cusomizing colours of points
#if colour is discrete give customized colours
if(!is.numeric(collected.data[,colour]) &
        length(unique(collected.data[,colour])) >= 8 &
        length(unique(collected.data[,colour])) < 12 ) {
  p = p + scale_colour_brewer(palette='Dark2')

}

#if colour is continuous give colour gradient
else if(is.numeric(collected.data[,colour])) {
  p = p + scale_colour_gradient(low='red')
}


#Adding text only if the "label" descriptor has been provided
if (!is.null(label))
{
 p = p + geom_text_repel(aes_string(x='x.value', y='y.value', label=label),
                  hjust=-0.5, vjust=0.3,
                  size=2.5, fontface='plain', lineheight=0.8)
}

#Drawing an ellipse for 95% confidence interval
if (!is.null(CIgroup)) {
  p = p + geom_path(data=elp.data, aes_string(x='x', y='y',
                                              group=colnames(elp.data[1])))
```

```
  }

  #Adding title only if "title" descriptor has been provided
  if (!is.null(title)) p = p + ggtitle(label=title)

  #Drawing origin
  p = p + geom_vline(xintercept=0, alpha=0.3)
  p = p + geom_hline(yintercept=0, alpha=0.3)

  p = p + xlab('PC score') +
    ylab('PC score') +
    theme_bw(12) +
    theme(axis.title.x = element_text(size=14, face='bold'),
          axis.title.y = element_text(size=14, face='bold'),
          axis.text = element_text(size=12, colour='black'),
          legend.title = element_text(size=14),
          legend.text = element_text(size=14),
          legend.key = element_rect(colour='white'),
          legend.spacing = unit(0, "cm"),
          legend.position = 'bottom',
          panel.grid=element_blank(),
          panel.border=element_rect(size=2, colour='black'),
          panel.spacing=unit(.05, 'npc'))

  #Legend orientation
  if (legend.ori == 'horizontal') {
    p = p + theme(legend.direction='horizontal',
                  legend.position='bottom')
  }


  #Setting limits
  p = p + xlim(-xlimit,xlimit)
  p = p + ylim(-ylimit,ylimit)

  # facet by pairs of PC
  p = p + facet_wrap(~ view.var, ncol=min(floor(length(PCs)/2), 3)) +
  theme(strip.text.x = element_text(size=12, face='bold'))

  return(list('collected.data'=collected.data, 'p'=p, 'elp.data'=elp.data))
}
```