ggplot2 seminar: phylopic, ggpairs & NMDS

Olivia Burge

11 October 2014

This tutorial is an extension to the introductory ggplot handouts found at https://github.com/orb16/seminaRs/tree/master/session_6_ggplot.. If you are new to ggplot it would be worth running through them first. This tutorial covers using phylopic images in plots, using ggpairs and changing the default colourscheme, and extracting information from an NMDS object to recreate it in ggplot2.

Animals in your plots

But of course. Here we source from phylopic.com. You should of course attribute (as per the website, see end of document for mine). Failing that, make you own (and submit to phylopic too :)): http://blogevolved.blogspot.co.nz/2011/02/making-silhouettes-for-phylopic.html

Packages required:

```
# Either run the code chunk above (hidden in output) or install the packages below using
# install.packages("") with package name inbetween the "".
require(ggthemes)
require(ggplot2)
require(vegan)
require(GGally)
require(grid)
require(gridExtra)
require(dplyr)
require(tidyr)

#install_github("devtools")
library(devtools)

#install_github("sckott/rphylopic")
library(rphylopic) #install Rphylopic package
```

```
data(mite)
data(mite.env)
mite_all <- cbind(mite, mite.env)
# names(mite_all) #check which ones we want
mite_long <- gather(mite_all[c(1:3, 36:40) ], key = "species",
                    value = "count", Brachy:HPAV)

mite_summarised_2 <- mite_long %>%
  group_by(species) %>%
  summarise(
    mean_count = mean(count),
    sd = sd(count),
    se = sd(count)/sqrt(length(count))
    )
```

```
base_mite <- ggplot(mite_summarised_2, aes(x = species, y = mean_count,
                                           fill = species))+
  geom_bar(stat = "identity", colour = "grey2") +
   scale_x_discrete(limits = c("PHTH", "HPAV", "Brachy"))+
   scale_fill_brewer(guide = FALSE, palette = "BrBG")+
   coord_cartesian(y = c(0, 15))+
  theme_minimal()
```

Now to get some 'mite' images:

```
mite_PHTH <- get_image("949c0ec2-97ae-4037-b2c5-991fb62c26e4", size = 512)[[1]]
mite_HPAV <- get_image("fed0b312-56bb-49be-869a-51951fbdea50", size = 512)[[1]]
mite_Brachy <- get_image("8465f7ef-8cc2-4249-9823-749f25c33543", size = 512)[[1]]

(phylo_mite <- base_mite +
  add_phylopic(mite_HPAV,
               x = 2, y = mite_summarised_2[[3,2]] + (mite_summarised_2[[3,2]]/7),
               ysize = 2)+
  add_phylopic(mite_PHTH,
               x = 1, y = mite_summarised_2[[2,2]] + (mite_summarised_2[[2,2]]/1.5),
               ysize = 4) +
  add_phylopic(mite_Brachy, x = 3, y = mite_summarised_2[[1,2]] +
                 (mite_summarised_2[[1,2]]/5),
               ysize = 3) +
   labs(x = "Mite species", y = "Mean abundance"))
```

```
ggsave(phylo_mite, file = "phylo_mite_yeah.png")
```

```
## Saving 6.5 x 4.5 in image
```

Images as backgrounds

See my friend Monica's blog post on this for another example: http://descienceblog.tumblr.com/post/91071648745/lions-and-tigers-and-bears-if-you-have-data-from

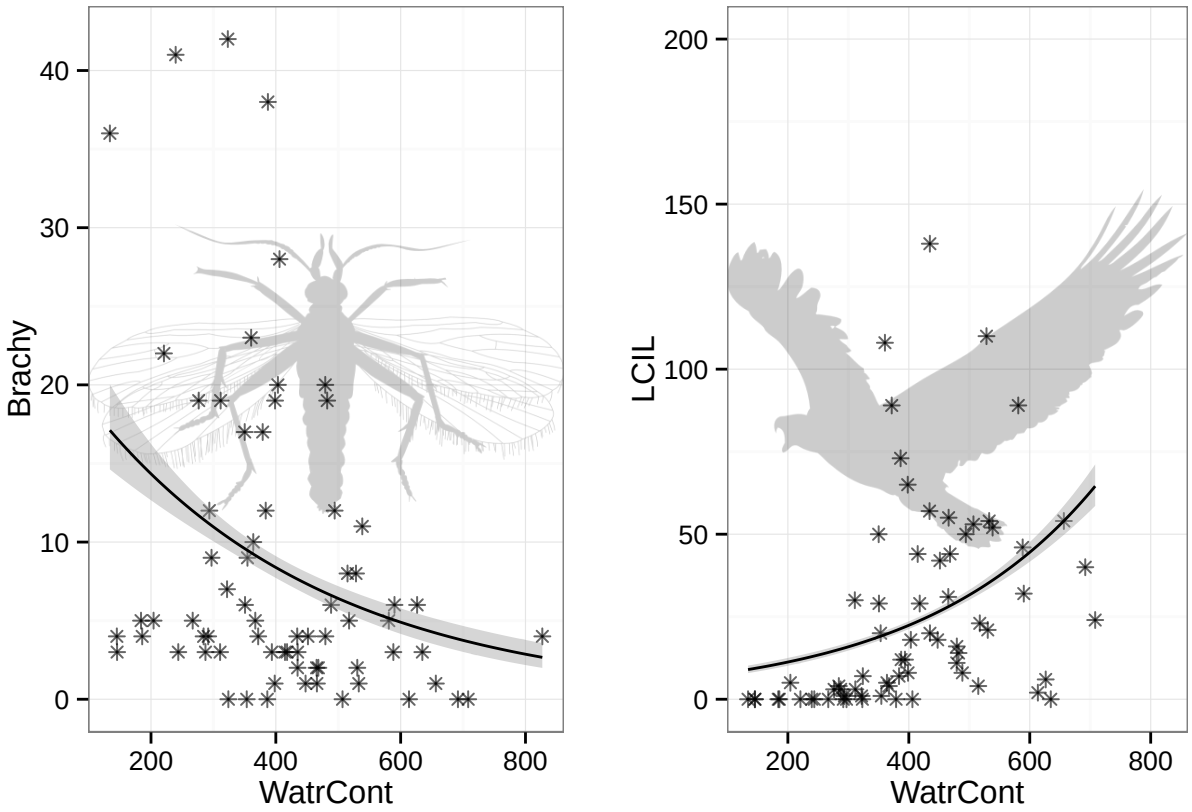
```
# Or as backgrounds
ggplot(mite_all, aes(x = WatrCont, y = Brachy)) +
  geom_point()

mite_funny <- get_image("8b09a9bd-3a2a-4979-8921-2505dad27c43", size = 512)[[1]]
mite_diff <- get_image("804de116-5dbe-4e53-bedb-ddbfb4e895eb", size = 512)[[1]]

brachy <- ggplot(mite_all, aes(x = WatrCont, y = Brachy)) +
  geom_point(alpha = 0.6, shape = 8) +
  add_phylopic(mite_funny) +
  geom_smooth(method = "glm", family = "poisson", colour = "black")+
  theme_bw()
brachy

lcil <- ggplot(mite_all, aes(x = WatrCont, y = LCIL)) +
  geom_point(alpha = 0.6, shape = 8) +
  ylim(0, 200)+ #excludes the outlier from our analysis
  add_phylopic(mite_diff) +
  geom_smooth(method = "glm", family = "poisson", colour = "black")+
  theme_bw()
lcil

require(gridExtra) # required for arrangeGrob.
# You could also use grid.arrange and do almost the same thing, but
# then you wouldn't be able to use ggsave()
brachylcil_plot <- arrangeGrob(brachy, lcil, ncol = 2)
brachylcil_plot
```




```
#ggsave(brachylcil_plot, file = "brachylcil_plot.pdf")
```

Visualising *much* data at once: ggpairs

This is quite useful for checking out your environmental variables before modelling them. There is also the `pairs` function in base graphics, which is muuch faster (also using ggpairs in R - not Rstudio - is also randomly faster sometimes). *But* `pairs` won't auto plot in different `geoms` as appropriate.

`ggpairs` *is* highly customisable, but I've never enjoyed trying to get my head around it. I'd rather use `ggplot` to build a number of plots of then either `grid.arrange` or `arrangeGrob` them (you'll need to check the interweb for how those work).

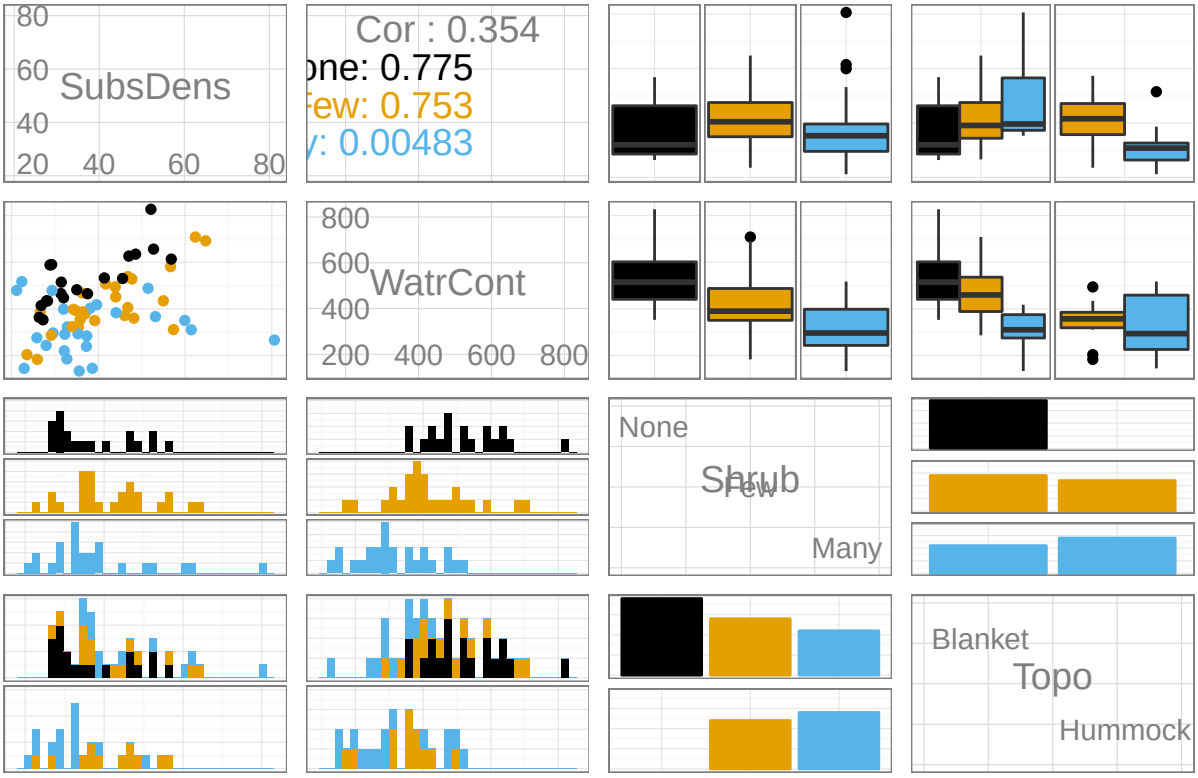
```
# the data
data(mite.env)
```

This part below is necessary to change the defaults of `ggplot`, which is referenced by `ggpairs`. It will throw errors if you haven't loaded `ggthemes` before plotting, because that's where we get the `scale_colour_colorblind` from. You could also use other scales (e.g. `scale_brewer` or `scale_manual`). Note that if you change to continuous colour data for ggpairs things will go wrong (specify a continuous colour scale in the `ggplotB` and rerun).

```
ggplotB <- function(...) ggplot2::ggplot(...) + scale_color_colorblind() +
  scale_fill_colorblind()
theme_set(theme_bw())
unlockBinding("ggplot",parent.env(asNamespace("GGally")))
assign("ggplot",ggplotB,parent.env(asNamespace("GGally")))
```

After all that, the `ggpairs` part is easy as. We exclude the `Substrate` variable to speed things up.

```
ggpair_mite <- ggpairs(mite.env[ , !names(mite.env) == "Substrate"], color = "Shrub")
ggpair_mite
```




```
#can't ggsave this. use pdf() or png()
```

Multivariate data

This is more about getting the components out. The autoplot command for base graphics is, to be fair, a lot easier. You can see how to do an ordisurf on my webpage, I'm not including it just here - http://oliviarata.wordpress.com/2014/07/17/ordinations-in-ggplot2-v2-ordisurf/.

This code is pretty highly annotated, and uses a dataset from `vegan`.

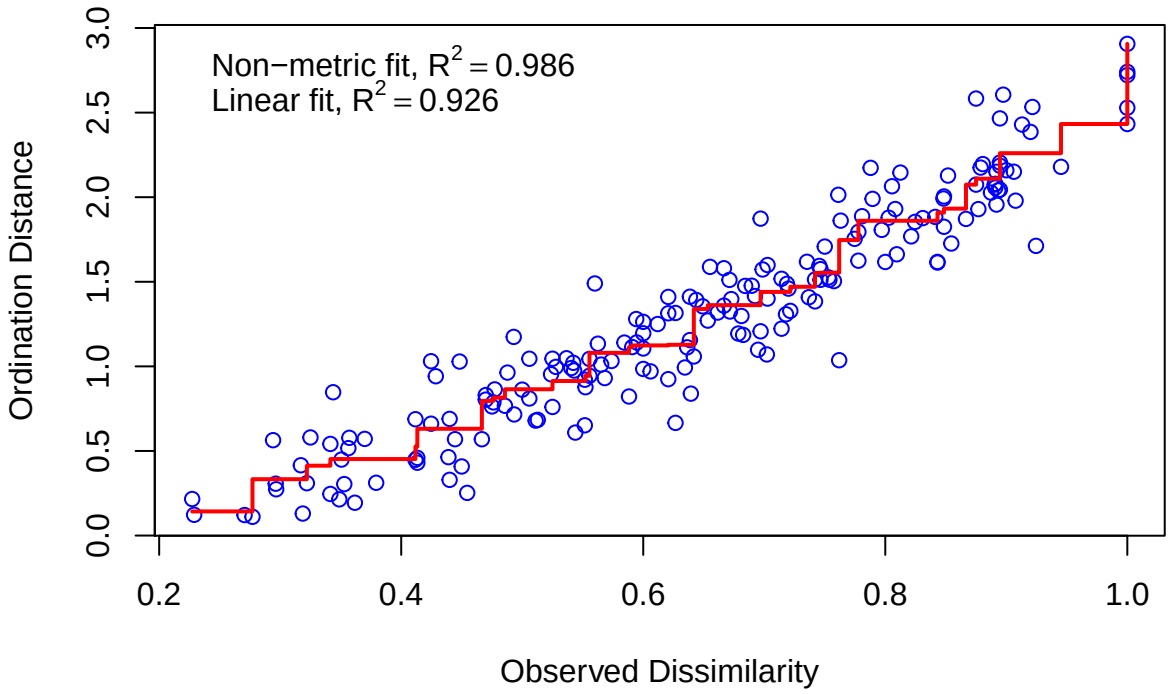
```
#required packages: install if not already
require(vegan)
require(ggplot2)
require(grid) #this is only required for the envfit arrows.
# I've spent many a happy few minutes questioning
# what on earth the arrowhead error was coming up for.

#data
data(dune)
data(dune.env)

#set the seed:
set.seed(201) # this allows us to reproduce the same result in the future
meta.nmds.dune <- metaMDS(dune) #no transformation of species data
```

```
## Run 0 stress 0.1193
## Run 1 stress 0.1193
## ... procrustes: rmse 0.0001045  max resid 0.000306
## *** Solution reached
```

```
stressplot(meta.nmds.dune)
```



```r
# envfit
dune.envfit <- envfit(meta.nmds.dune, env = dune.env, perm = 999)
dune.envfit
```

```
##
## ***VECTORS
##
##    NMDS1 NMDS2   r2 Pr(>r)
## A1  0.99  0.14 0.38  0.017 *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
## P values based on 999 permutations.
##
## ***FACTORS:
##
## Centroids:
##              NMDS1 NMDS2
## Moisture1    -0.51 -0.04
## Moisture2    -0.39  0.01
## Moisture4     0.28 -0.39
## Moisture5     0.66  0.15
## ManagementBF -0.45 -0.02
## ManagementHF -0.27 -0.13
## ManagementNM  0.30  0.58
## ManagementSF  0.15 -0.47
## UseHayfield  -0.17  0.36
## UseHaypastu  -0.04 -0.34
## UsePasture    0.30  0.04
## Manure0       0.30  0.58
## Manure1      -0.24 -0.03
## Manure2      -0.31 -0.18
## Manure3       0.30 -0.25
## Manure4      -0.34 -0.56
##
## Goodness of fit:
##              r2 Pr(>r)
## Moisture   0.50  0.001 ***
## Management 0.41  0.006 **
## Use        0.20  0.093 .
## Manure     0.42  0.025 *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
## P values based on 999 permutations.
```

```r
# data for plotting
## NMDS points
dune.NMDS.data<-dune.env #there are other ways of doing this.
# But this is the way I do it for ease of plotting

dune.NMDS.data$NMDS1<-meta.nmds.dune$points[ , 1] #this puts the NMDS scores
# for the plots  into a new dataframe.
# you could put them into an existing one if you preferred.
dune.NMDS.data$NMDS2<-meta.nmds.dune$points[ , 2]
```

```
## species data
stems<-colSums(dune) #total abundances for each species
spps <- data.frame(scores(meta.nmds.dune, display = "species"))
# dataframe of species scores for plotting

spps$species <- row.names(spps) # making a column with species names
spps$colsums <- stems #adding the colSums from above

spps <- spps[!is.na(spps$NMDS1) & !is.na(spps$NMDS2),] #removes NAs

spps.colmedian <- median(spps$colsums) #create an object that is the
# median of the abundance of the measured species
spps.colmean <- mean(spps$colsums) #creates a mean instead if you wish to use

spps2 <- subset(spps,spps$colsums > spps.colmean)
# select the most abundant species.
# Could discard fewer e.g. spps$colsums>(spps.colmedian/2) instead

spps2$species <- factor(spps2$species)
#otherwise factor doesn't drop unused levels
# and it might throw an error

# data for the envfit arrows
env.scores.dune <- as.data.frame(scores(dune.envfit, display = "vectors"))
#extracts relevant scores from envifit

env.scores.dune <- cbind(env.scores.dune,
                         env.variables = rownames(env.scores.dune))
#and then gives them their names

# function for ellipses - just run this, is used later
# taken from the excellent stackoverflow Q+A:
```

See also: http://stackoverflow.com/questions/13794419/plotting-ordiellipse-function-from-vegan-package-onto-nmds-plot-creat

```
# run this function, no changing it:
veganCovEllipse <- function (cov, center = c(0, 0), scale = 1, npoints = 100) {
  theta <- (0:npoints) * 2 * pi/npoints
  Circle <- cbind(cos(theta), sin(theta))
  t(center + scale * t(Circle %*% chol(cov)))
  }

#data for ellipse, in this case using the management factor
df_ell.dune.management <- data.frame()
#sets up a data frame before running the function.

for(g in levels(dune.NMDS.data$Management)){
  df_ell.dune.management <- rbind(
    df_ell.dune.management,
    cbind(as.data.frame(with(
      dune.NMDS.data[
        dune.NMDS.data$Management==g,],
      veganCovEllipse(cov.wt(cbind(NMDS1,NMDS2),
```

```
                              wt = rep(1/length(NMDS1),
                                       length(NMDS1)))$cov,
                        center=c(mean(NMDS1),mean(NMDS2)))))),
      Management=g))}
```

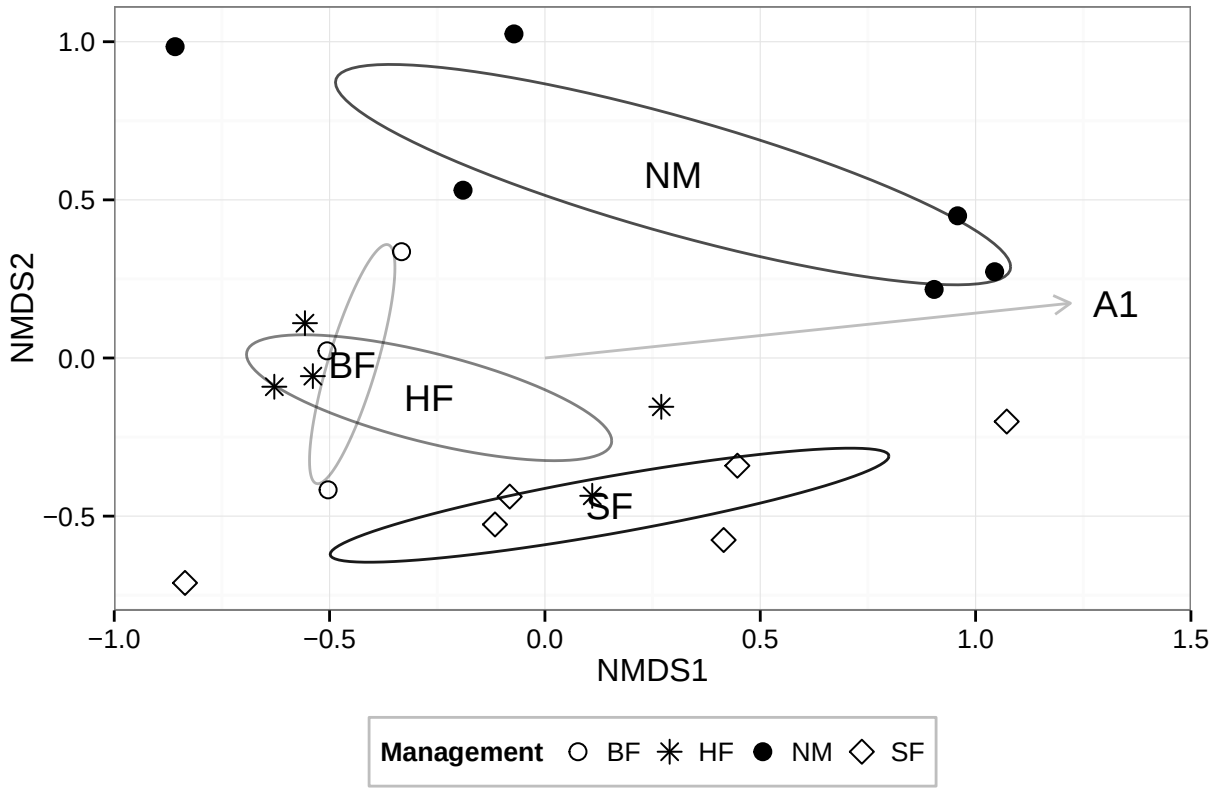
```
# data for labelling the ellipse
NMDS.mean.dune = aggregate(dune.NMDS.data[ ,c("NMDS1", "NMDS2")],
                           list(group = dune.NMDS.data$Management),
                           mean)

# data for labelling the ellipse
NMDS.mean = aggregate(dune.NMDS.data[,c("NMDS1", "NMDS2")],
                      list(group = dune.NMDS.data$Management),
                      mean)

## finally plotting.
mult <- 2 #multiplier for the arrows and text for envfit below.
#You can change this and then rerun the plot command.

(dune.nmds.gg1 <- ggplot(data = dune.NMDS.data, aes(y = NMDS2, x = NMDS1))+
   ## sets up the plot.
   ## brackets around the entire thing to make it draw automatically
   geom_path(data = df_ell.dune.management,
             aes(x = NMDS1, y = NMDS2, group = Management, alpha=Management))+
   ## this is the ellipse, seperate ones by Site.
   ## If you didn't change the "alpha" (the shade) then you need to keep the "group""
   scale_alpha_manual(guide = FALSE,values=c(0.3, 0.5, 0.7, 0.9))+
   ## sets the shade for the ellipse
   geom_point(aes(shape = Management), size = 3) +
   ## puts the site points in from the ordination,
   ## shape determined by site, size refers to size of point
   # geom_text(data=spps2, aes(x=spps2$NMDS1, y=spps2$NMDS2, label=species),
   # size = 3.3, hjust=1.1)+
   ## labelling the species. hjust used to shift them slightly from their points
   annotate("text",x = NMDS.mean$NMDS1,y = NMDS.mean$NMDS2,label=NMDS.mean$group) +
   ## labels for the centroids - I haven't used this since we have a legend.
   ## but you could also ditch the legend, but plot will get v messy
   geom_segment(data = env.scores.dune,
                aes(x = 0, xend = mult*NMDS1, y = 0, yend = mult*NMDS2),
                arrow = arrow(length = unit(0.25, "cm")), colour = "grey") +
   ## arrows for envfit.  doubled the length for similarity to the plot() function.
   ## NB check ?envfit regarding arrow length if not familiar with lengths
   geom_text(data = env.scores.dune, # labels the environmental variable
             aes(x = mult*NMDS1, y = mult*NMDS2, label=env.variables),
             size = 5,
             hjust = -0.5)+
   # geom_point(data=spps2, alpha = .6, shape = 4)+
   ## these are the species points, made lighter and a specific shape
   scale_shape_manual(values = c(1,8,19,5))+
   ## sets the shape of the plot points
   ## instead of using whatever ggplot2 automatically provides
   coord_cartesian(xlim = c(-1,1.5))+
   ## NB this changes the visible area of the plot only
```

```
  theme_bw()+
  theme(legend.position = "bottom",
        legend.key = element_blank(),
        legend.background = element_rect(colour = "grey")))
```




```
# NB I usually use the "pdf" function in rstudio to export the plot.
#But you can also:

# ggsave(plot = dune.nmds.gg1, filename = "awesome_dune.pdf") #if you don't specify
# the plot it will save the last one
## many other options - see ?ggsave.  Can set dimensions, units, dpi
```

Credits for phylopic images:

Frank Förster, for mite_Brachy (http://phylopic.org/image/8465f7ef-8cc2-4249-9823-749f25c33543/), license: http://creativecommons.org/licenses/by-sa/3.0/

Ghedoghedo and T. Michael Keesey, for mite_HPAV (http://phylopic.org/image/fed0b312-56bb-49be-869a-51951fbdea50/), license: http://creativecommons.org/licenses/by-sa/3.0/

Ghedoghedo (vectorized by T. Michael Keesey), mite_PHTH (http://phylopic.org/image/949c0ec2-97ae-4037-b2c5-991fb62c26e license: http://creativecommons.org/licenses/by-sa/3.0/.

`mite_diff` and `mite_funny` have no copyright attached.

Happy plotting!