Making graphs with ggplot2

There are two major graphics approaches in R.
Base: what we've used until now; fast, but limited. "pen on paper" model
ggplot2: grid based, with strongly structured grammar, with easy control of complex graphics.

IMPORTANT! Please install and load the ggplot2 and grid packages

#Sample data sets

```
library(ggplot2); library(grid)
rd<-read.csv("http://people.ucsc.edu/~ggilbert/Rclass_docs/ggplotdata.csv")
t<-read.csv("http://people.ucsc.edu/~ggilbert/Documents/FERPTopoStreamTrailData.csv")  #get
ferp elevation data
ard<-aggregate(rd$precip, by=list(landuse=rd$use), FUN=mean) #find means
```

#end sample data sets

ggplot2 package

ggplot2 provides a very rich structure in which to quickly make complex plots. It is particularly useful when you want to display different groups of data on the same plot, or in a panel of plots. It is by far the favorite graphics package out there in R.

Main limitation: qplot, the workhorse function of ggplot2, is not generic in the way plot is. Recall that the base function plot will figure out the appropriate kind of plot to make from just about any R object (output of lm, glm, trees, whatever). qplot won't do that for you.

Detailed documentation. *Ggplot2: Elegant Graphics for Data Analysis* by Hadley Wickham. Available through UC library Springer Link to download as pdfs.

ggplot-particular terminology

data: what you want to visualize
aesthetic attributes: color, shape, size, alpha (transparency).
mapping: how variables are mapped to aesthetic attributes you can perceive. An attribute can be fixed (take one value – all black dots) or be mapped to a variable (different colors depending on the value of a grouping variable)
geoms: geometric objects you see on the plot (point, smooth, line, path, bar, boxplot, freqpoly, density)
stats: statistical transformations and summarizations
scales: what map data values into a visual aesthetic space, including legends and axes
coord: coordinate system (usually Cartesian, can be log, semilog, polar, etc.)
facet: break data into display subsets (like latticing) row_var ~ col_var
layers: puts different kinds of graphs on top of each other (like points() or lines() in base)
viewport: rectangular subregion of a display
themes: the appearance of non-data element (like background, axes, strip, gridlines, ticks fonts, labels
alpha: transparency, from 0 (transparent) to 1 (opaque)

# qplot

qplot is the simple plotting workshorse. (ggplot gives more control over everything)
qplot has the following overall basic structure
qplot(*x, y, data, geom*)
color, shape, size, and *fill* control those elements of symbols like col, pch, cex, bg in plot
alpha controls transparency of symbols, from 0 to 1 (transparent to opaque)
xlim, ylim, xlab, ylab, main, sub, and *log* do the same as in plot
method and *formula* control model fits. e.g. method="lm" formula=y~x
facets controls a trellis of rowvar ~ colvar
stat = list() controls what statistics to use

Look at the data in rd; temperature, precipitation, number of species for 13 sites in each of three land-use types (easement, park, and private).

```
  temp precip num_spp      use
1   33     13      52 easement
2   16     39     138 easement
3   12     71     296 easement
```

Scatterplots using qplot

#first make a simple scatterplot of all 39 points

```
qplot(precip,num_spp,data=rd, geom="point") #make a basic scatterplot
```

#note that if you supply x and y, the default is to geom="point"

#and you can use all the basic variations we are comfortable with

```
qplot(log(precip),num_spp,data=rd)  #plots the natural log of the precip
qplot(precip,num_spp,data=rd, log="x")  #plots precip on a log scale
qplot(precip,num_spp,data=rd,xlab="Precipitation (mm)",ylab="Species richness",
main="my test plot")  #add axis labels and title
```

```
#Build the graphs in layers (like adding points or lines to plot)
ggplot2 allows you to modify or add to plot by adding new components
p<-qplot(precip,num_spp,data=rd, col=use, geom="point")
p  #shows the basic plot with different colors for each use group
p + stat_smooth(method="lm",formula=y~x,se=FALSE)
#basic plot plus the line fit for each group
```

You can add as many layers as you like to previous plots,
separated by a +

stat_smooth(), theme(), and scale() (see below) are common components to add to plots.
These largely replace par() from base plot

#Set the aesthetic attributes of symbols

```
#You can set aesthetics manually. Must use I() function
qplot(precip,num_spp,data=rd, size=I(4)) #set symbol size larger
qplot(precip,num_spp,data=rd, shape=I(17)) #set particular symbols
qplot(precip,num_spp,data=rd, color=I("purple")) #set symbol color
qplot(precip,num_spp,data=rd, color=I("purple"), shape=I(17), size=I(3))
```

#map aesthetic attributes to subgroup (factors) or other variables
#qplot is smart about automating this to default colors

```
qplot(precip,num_spp,data=rd,color=use) #automatic colors and legend
qplot(precip,num_spp,data=rd,shape=use) #automatic symbols and legend
qplot(precip,num_spp,data=rd,size=temp) #automatically size and legend
qplot(precip,num_spp,data=rd,size=temp,color=use,shape=use) #combined

#Create your own discrete scales for color, shape, size, lines, alpha, etc.
#To specify the aesthetic attributes for each group in a plot, first indicate
inside the qplot call that the aesthetic attribute should vary according to some
variable or factor, then add the descriptor that specifies which values to map to
the factor using scale_x_manual options
qplot(precip,num_spp,data=rd,size=I(3),color=use, shape=use) +
scale_color_manual(values=c("purple","yellow","black")) +
scale_shape_manual(values=c(2,19,8))


# Create an object with the colors to use associated with each group
gregcolors<-c("orange","red","black") #vector with the colors to use
names(gregcolors)<-c("private","park","easement") #give name to match group
#then call directly to the object of color names
qplot(precip,num_spp,data=rd,size=I(3),color=use, shape=use) +
scale_color_manual(values=gregcolors) + scale_shape_manual(values=c(2,19,8))

#But what happens if the names don't match the groups?
gregcolors<-c("orange","red","black") #vector with the colors to use
names(gregcolors)<-c("fred","bill","martha") #give name to match group
qplot(precip,num_spp,data=rd,size=I(3),color=use, shape=use) +
scale_color_manual(values=gregcolors) + scale_shape_manual(values=c(2,19,8))
#Since color names do not match the use groups, it does not work

#But if the colors in the object have no names, they are used in order
#following alphabetical order of groups
gregcolors<-c("orange","red","black") #vector with the colors to use
qplot(precip,num_spp,data=rd,size=I(3),color=use, shape=use) +
scale_color_manual(values=gregcolors) + scale_shape_manual(values=c(2,19,8))

#Create your own continuous scales for color, shape, size, lines, alpha, etc.
#To specify the aesthetic attributes for each group in a plot, first indicate
inside the qplot call that the aesthetic attribute should vary according to some
variable or factor, then add the descriptor that specifies which values to map to
the factor using scale_x_manual options
qplot(precip,num_spp,data=rd,size=I(3),color=temp, shape=use) +
scale_color_continuous(low="purple",high="yellow")+
scale_shape_manual(values=c(2,19,8))


#making a color palette
#Use scale_x_manual functions to create a formal attribute palette and then call it later
#Create a palette, indicate colors should differ among groups, then apply palette at the end

myFillPalette<-scale_fill_manual(values=c("black","red","white"))
myColorPalette<-scale_color_manual(values=c("black","red","white"))

qplot(precip,num_spp,data=rd,size=I(3),color=use) #sets colors automatically
qplot(precip,num_spp,data=rd,size=I(3),color=use) + myColorPalette
 #uses colors as defined in the palette, in that order

#using RColorBrewer pallettes
library(RColorBrewer) #need to load it and works well with ggplot
#There are a number of pallettes with 3-12 values
```

```
#to see them all:
display.brewer.all(colorblindFriendly=TRUE) #FALSE gives additional ones to avoid!
display.brewer.pal(n=12, "Paired") #shows a particular palette called Paired

qp<-qplot(precip,num_spp,data=rd,size=I(3),color=use)
qp+scale_colour_brewer(palette="Paired")  #uses the first three colors from Paired

myColors<-brewer.pal(8,"Paired")[c(2,6,8)] #palette of col 2,6,8 from Paired
qp+scale_color_manual(values=myColors)

#a note on alpha – transparency (useful for crowded graphs)
qplot(precip,num_spp,data=rd, size=temp, alpha=I(1/3))
#adjust transparency with alpha(I(1/3) or alpha(I(0.33))
#means 3 points must overlap to be completely opaque)
#Add fitted lines to the data using stat_smooth
#First go back to the simple graph of all data
qplot(precip,num_spp,data=rd, geom="point") #make a basic scatterplot
qplot(precip,num_spp,data=rd, geom="point")+ stat_smooth(method="lm")
# this gives a fitted linear regression with default 95% CI

#call the same kind of graph with more specificity
qplot(precip,num_spp,data=rd, geom="point")+
stat_smooth(method="lm",formula=y~x,level=0.90)
#note the formula uses x and y, not precip and num_spp
#specifies a 90% CI

#turn off the confidence intervals using se
qplot(precip,num_spp,data=rd, geom="point")+
stat_smooth(method="lm",formula=y~x,se=FALSE)
#note the formula uses x and y, not precip and num_spp

#log transforming x so fits model y~log(x)
qplot(precip,num_spp,data=rd, geom="point")+
stat_smooth(method="lm",formula=y~log(x))
#note the formula uses x and y, not precip and num_spp
#or to use a different model, like glm for poisson regression
#can take lm, glm, gam, loess, rlm
qplot(precip,num_spp,data=rd, geom="point")+
stat_smooth(method="glm",formula=y~x,family="poisson")

#but remember that there are really three subsets of data
#if you want to fit each with a separate line
qplot(precip,num_spp,data=rd,col=use, geom="point",size=I(5)) +
stat_smooth(method="lm",se=FALSE,lwd=I(1),lty=I(2))
```

```
#add annotation text to a plot
p + annotate(geom="text", x=0,y=800,label="your logo \n here",hjust=0)
#add some text to the plot at coordinates 0,800;  hjust ranges from 0 to 1 for horizontal justification
```

Using theme() and scale() to control plot appearance

Theme() controls the appearance of non-data elements of the plot, such as labels, the background, ticks and grids, text size, color, and orientation. Scale() controls length and tick placement along axes.

There are five major **theme elements: `axis, legend, panel, plot, and strip`**.
Each major theme element has specific elements within it:
`axis:` `axis.line, axis.text, axis.ticks, axis.title`
`legend:` `legend.background, legend.key, legend.text, legend.title`
`panel:` `panel.background, panel. border, panel.grid.major, panel.grid.minor`
`plot:` `plot.background, plot.title`
`strip:` `strip.background, strip.text`

Note that for axis.text, axis.ticks, axis.title, and strip.text the theme elements can be further specified as x or y, as
`axis.text.x   axis.text.y   axis.title.x   strip.text.x   etc.`

There are four major **element functions** to control the theme elements
`element_text``():family, face, color, size, hjust, vjust, angle, lineheight`
`element_line``(): color, size, linetype`
`element_rect``(): fill, color (the border), linetype`
`element_blank``(): draws nothing.  Use to suppress element you don't want`

Control themes by adding them in a theme() call after the qplot call. e.g., qplot() + theme()
For example, start with a basic plot, clear out the background, move the legend, and adjust the x axis

```
p1<-qplot(precip,num_spp,data=rd,xlab="Precipitation (mm)",ylab="Species
richness",color=use,main="Effect of precipitation on species richness")
p1 #shows basic plot

#now use theme() to adjust features you want
p1 + theme(
panel.background=element_blank(),  #clear background
panel.border=element_rect(color="black", fill=NA), #draw box around plot with no
fill
panel.grid=element_blank(),  #remove grid lines
axis.text=element_text(color="black",size=14), # make axis text black and 14
points
legend.key=element_blank(), #clear background on the legend
legend.background=element_rect(color="black",fill=NA,size=0.05),  #draw a box
around the legend
legend.position=c(.21,.77) #reposition the legend
) +
scale_x_continuous(limits=c(0, 300),breaks=seq(0,300,50)) + #adjust x axis limits
and ticks
scale_y_continuous(limits=c(0,1600), breaks=seq(0,1600,400)) #adjust y axis limits
and ticks
```

#Give your favorite theme combinations a name, and then add that to a model
#here is a set of theme that I apply to most graphs as a starting point. It gets rid of the background and grids, and generally cleans up the graph

```
myFaveThemes<-theme(panel.grid.minor=element_blank(),
panel.grid.major=element_blank(),
panel.background=element_rect(fill=0,color=1),legend.key=element_blank(),
legend.background=element_rect(color="black",fill=NA,size=0.2))
#to use it, just add it after the base plot
p1+myFaveThemes

#And make a stat_smooth theme to put regression lines on your graphs
simpleregline<- stat_smooth(method="lm",formula=y~x,se=FALSE)
p1+myFaveThemes + simpleregline
```

Some other kinds of graphs you can make using qplot

Bubble charts (scatterplot, with symbol size mapped to a variable)

```
qplot(precip,num_spp,data=rd,size=temp) #automatically size and legend
```

Smoothed Line plots

```
qplot(precip,num_spp,data=rd,geom="smooth") #smoothed line and standard error
qplot(precip,num_spp,data=rd,col=use,geom="smooth") #by category
```

Connect the dot line plots (time series plots) without points

```
qplot(precip,num_spp,data=rd,geom="line") # connect the dots
```

Points connected by lines (path plots)

```
qplot(precip,num_spp,data=rd,geom=c("point","path"),col=use) # connect the dots
within each use category, note use of both point and path geoms
```

Boxplot

```
qplot(use, precip, data=rd,geom="boxplot",fill=use) #boxplot with x group and y
depvar
```

Histogram

```
qplot(num_spp, data=rd,geom="bar",binwidth=100,fill=I("blue"),color=I("white"))
#histogram of depvar
```

Stacked Histogram (showing categories within bars)

```
qplot(num_spp, data=rd,geom="bar",binwidth=100,fill=use) #color code by use
```

Frequency polygon

```
qplot(num_spp, data=rd,geom="freqpoly",binwidth=100,col=use) #frequency polygon
with separate lines for each use
```

Density plot

```
qplot(num_spp, data=rd,geom="density",binwidth=100) #frequency polygon
qplot(num_spp, data=rd,geom="density",binwidth=100,col=use) #frequency polygon
```

Bar plot

```
ard<-aggregate(rd$precip, by=list(landuse=rd$use), FUN=mean) #find means
qplot(landuse,x, data=ard,geom="bar",stat="identity") #frequency polygon
```

Scatterplot with fitted line

```
qplot(precip,num_spp,data=rd,geom=c("point","smooth"),se=FALSE) #smoothed line, no
standard error

qplot(precip,num_spp,data=rd,geom=c("point","smooth"),method="lm") #linear model
with standard error, defaulting to straight line model

qplot(precip,num_spp,data=rd,geom=c("point","smooth"),method="lm",formula =
y~log(x)) #specify linear model

qplot(precip,num_spp,data=rd,geom=c("point","smooth"),method="lm",col=use) #linear
model fit separately to each use group
```

Note: you can load other libraries like MASS, splines, mgcv, etc. to fit other models in the same way. See ggplot2 documentation.

Faceting (=Plotting on a grid, Lattice plots, Trellis plot)

Facets allow you to divide your data into groups and plot them on adjacent graphs to facilitate comparison (instead of plotting them on the same graph).
Note: see Viewports and Rectangular Grid, below, if you want to put different kinds of plots on a single page)

The faceting formula takes the form row_var ~ col_var.
This means that
facets = row_var ~. will create vertically stacked graphs (in rows) according to row_var
facets = .~col_var will create horizontally arranged graphs (columns) following col_var
facets = row_var ~ col_var will create a matrix of plots

#stacked graphs
```
qplot(num_spp,
data=rd,geom="bar",binwidth=100,fill=I("blue"),col=I("white"),facets=use~.)
```

#horizontal graphs
```
qplot(num_spp,
data=rd,geom="bar",binwidth=100,fill=I("red"),col=I("white"),facets=.~use)
+theme(strip.background=element_rect("yellow"),
strip.text=element_text(face=c("bold.italic")))
```

Coordinate systems

There are several ways to transform the coordinate systems.
Start with a simple plot in Cartesian coordinates:
```
qplot(precip,num_spp,data=rd,color=use)

#three ways to have log10(x) axis
qplot(precip,num_spp,data=rd, log="x") #transforms data
qplot(precip,num_spp,data=rd) + scale_x_log10() #transforms data
qplot(precip,num_spp,data=rd) + coord_trans(x="log10") #raw labels

#three ways to have log-log scales
qplot(precip,num_spp,data=rd, log="xy") #x and y axes on log
qplot(precip,num_spp,data=rd) + scale_x_log10() + scale_y_log10()
qplot(precip,num_spp,data=rd) + coord_trans(x="log10",y="log10")

#polar coordinates
qplot(precip,num_spp,data=rd) + coord_polar("x") #x axis on spokes
qplot(precip,num_spp,data=rd) + coord_polar("y") #y axis on spokes
qplot(precip,num_spp,data=rd) + coord_polar("x",start=2,direction=-1)
#x axis on spokes starting 2 radians from 12 oclock going counterclockwise

#create a base plot by putting the plot object into "b", and then appending
b<- qplot(precip,num_spp,data=rd,color=use)
b   #sends the plot to a quartz window
b + coord_flip()  # exchanges the x and y axes
b + coord_trans(x="log10")  #show with x axis on log10 scale
b + coord_trans(y="sqrt")  #show with x axis on square-root transformed scale
b + coord_polar("x") #show in polar coordinates with x axis around the wheel
```

Saving simple plots from ggplot2
(On macs you must first install XQuartz from http://xquartz.macosforge.org/landing/)

ggplot has a really cool and convenient function called ggsave() to save graphs to disk.
By default, if you give ggsave a filename to save to, it will take the last graph displayed to the quarts window, the dimensions of the quartz window, and will guess the device type (e.g., pdf, png) from the filename.
Thus all you need, after displaying a graph to your satisfaction, is:
ggsave(filename="myfirstgraph.png")

If you want more control, you can to that too.
It has the general form of:

```
ggsave(filename,plot,scale,width,height,units,dpi)
```

devices are determined by tag on filename: raster formats (jpeg, tiff, png, bmp), vector formats (svg, eps, pdf)
scale: scaling factor (relative to onscreen display)
width and height: defaults to inches, but can set units to "in", "cm", or "mm"
dpi: for raster graphics only

ggsave("myfirstgraph.jpg",b, dpi=600,height=4, width=6)
ggsave("myfirstgraph.svg",height=4, width=6)

Saving more plots from ggplot2 using a device

You can also use the standard device approach, then print graphs to that device

```
png("myplot.png", width=600, height=400)
print(p1)
dev.off()
```

# other devices include pdf, postscript, xfig, cairo_pdf, svg, png, jpeg, bmp, tiff, quartz.

Viewports to put multiple graphs on the same canvas
Need to load the grid package for this to work

```
library(grid)
```

Viewports are rectangular subregions on a display that allow you to put multiple graphs on a single page.
First create three graphs, assigning each to an object.

```
p1<-qplot(precip,num_spp,data=rd,xlab="Precipitation (mm)",ylab="Species
richness",color=use)
p2<-qplot(num_spp, data=rd,geom="bar",binwidth=100,fill=use)
p3<-qplot(use, precip, data=rd,geom="boxplot",fill=use)
```

Viewports are created with the viewport() function from the grid package

```
viewport(x, y, width, height)
```

x and y denote the coordinates of the <u>*center*</u> of the viewport on the page.
Coordinates are usually given in npc units.
npc units range from 0 to 1 (0,0) is bottom left, (1,1) is top right, (0.5,0.5 is the center)
Alternatively, can use absolute units like x=unit(2,"cm") or y=unit(3,"inch")

```
vp0<-viewport(width=1, height=1, x=0.5, y=0.5)  #viewport takes up full page
```

Create three viewports (viewports can overlap, if you want):

```
vp1<-viewport(width=0.5, height=1,x=0.25,y=0.5) #left half of page
vp2<-viewport(width=0.5, height=0.5, x=0.75, y=0.75) #top right of page
vp3<-viewport(width=0.5, height=0.5, x=0.75, y=0.25) #bottom right of page
```

print(p1,vp=vp1) #use the print call to display the graphs you saved in the appropriate viewport
print(p2,vp=vp2)
print(p3,vp=vp3)

Placing graphs on a rectangular grid

You can set up a series of viewports, or you can take the easy way and use grid.layout()
This has four steps:

(1) create a layout with a #x# grid (e.g., 2x2) (with layout=grid.layout())
(2) assign that grid to a viewport
(3) push the viewport onto a plotting device (with pushViewport())
(4) draw each graph onto its own grid on the viewport

Let's use the same three graphs as above (p1, p2, p3) and put them into a 3x1 grid.

```
grid.newpage()  #erases current device, and gets ready for a new grid layout
pushViewport(viewport(layout=grid.layout(3,1)))  #creates a 3x1 layout in a
viewport and pushes to device
print(p1,vp=viewport(layout.pos.col=1, layout.pos.row=1))
print(p2,vp=viewport(layout.pos.col=1, layout.pos.row=2))
print(p3,vp=viewport(layout.pos.col=1, layout.pos.row=3))
```

You can push this to a file by embedding it in call to a device (e.g., pdf();   dev.off())
You can use a range of rows or columns to spread a graph across multiple cells in a grid

```
png("agridofgraphs.png",width=600,height=600)
grid.newpage()  #erases current device, and gets ready for a new grid layout
pushViewport(viewport(layout=grid.layout(2,2)))  #creates a 2x2 layout in a
viewport and push to device
print(p1,vp=viewport(layout.pos.row=1, layout.pos.col=1:2)) #puts across both
columns in top row
print(p2,vp=viewport(layout.pos.row=2, layout.pos.col=1)) #bottom left
print(p3,vp=viewport(layout.pos.row=2, layout.pos.col=2)) #bottom right
dev.off()
```

#use annotate to add error bars to a bar plot

```
#first summarize your data
ard<-aggregate(rd$precip, by=list(landuse=rd$use), FUN=mean) #find means
srd<-aggregate(rd$precip, by=list(landuse=rd$use), FUN=sd) #find sd
ard$sd<-srd[,2]; colnames(ard)<-c("use","mean","sd")

bp<-qplot(use,mean,data=ard,geom="bar",stat="identity") #make barplot
bp+ annotate("pointrange", x=ard$use, y=ard$mean, ymin=ard$mean-
ard$sd,ymax=ard$mean+ard$sd) #annotate with error bars
```

#brief intro to ggplot

#ggplot provides even more control than qplot but with very similar syntax
#it requires specifying the aesthetics in an aes statement in the function

gp<-ggplot(data=rd, aes(x=precip, y=num_spp)) + geom_point()
#note ggplot call sets up the aesthetics of the plot, but must then add geom_point() layer to display

```
gp+ geom_smooth(method = "lm", se=FALSE, color="black", formula = y ~ x)
```

#adds in the linear regression line as another layer on top