ggplot2 (inspired by R Graphics Cookbook by Winston Chang)

Melanie Frazier (NCEAS) & Lee McCoy Cheatsheet

Step 1: Setting everything up

The ggplot portion of the code doesn't actually plot anything, but it defines the data that is used to create the plot.

The arguments within *aes()* link a variable in the data to an aspect of plot appearance: x and y describe the axes, and another argument can be added to describe a z variable (e.g. size of points based on variable z)

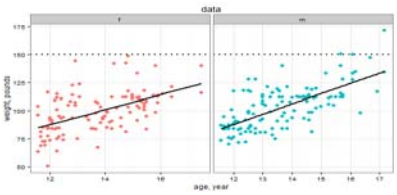
```
ggplot(data, aes(x= , y= , size=      ))
                 group=   #categorical var.
                 color=  #color of lines/points
                 fill=  #color within polygons
                 label= #if points are labels
                 linetype= #type of line
                 shape=  #style of point
                 alpha= #transparency (0-1)
```

Example

```
library(gcookbook)
hw  <- heightweight
head(hw)
```

```
sex ageYear ageMonth heightIn weightLb
  f   11.92      143     56.3     85.0
  f   12.92      155     62.3    105.0
  f   12.75      153     63.3    108.0
  f   13.42      161     59.0     92.0
  f   15.92      191     62.5    112.5
  f   14.25      171     62.5    112.0
```

```
ggplot(hw, aes(x=ageYear, y=weightLb, color=sex)) +
 geom_point(size=3) +
 facet_grid(. ~ sex, scales="free") +
 theme_bw() +
 stat_smooth(method=lm, se=FALSE, color="black",
                        size=1) +
 geom_hline(yintercept=150, linetype=3, size=1) +
 labs(x='age, year', y='weight, pounds', title='data')
```



Extra stuff

Themes

Black and white theme:

```
theme_bw()
```

My scatterplot theme:

```
source('https://raw.githubusercontent.com/OHI-Science/ohiprep/master/src/R/scatterTheme.txt')
plot + scatterTheme
```

Saving

```
ggsave('filename.png/pdf/jpg/wmf/svg',
        height = , width = , units = "cm", "in",
        dpi = )
```

Automating

```
p %+% new_data
```

Replaces the default dataframe in the ggplot object

Parse turns character strings into expressions.
To check: returns error if not valid:

```
expression(r^2 == 0.42)
?plotmath  #info for math expressions
demo(plotmath)
```

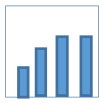
Step 2: Selecting the plot geom

Geoms are added to the ggplot function to create the plot. Arguments for *aes()* can be included and will override any *aes()* arguments from Step 1.

```
geom_point()
  size=2 #default value
```

NOTE: shape 16 can appear jagged in bitmap files. 19 is better.

```
geom_line()
```

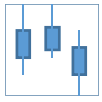


```
geom_bar()
  stat="identity" #(y values vs. counts)
  fill="blue"  #(color of bars)
  colour = "black" #(color of outline)
  width = 0 to 1 #default is 0.9
```

```
ggplot(data, aes(x=, y=, fill=)) +
  geom_bar(position="dodge",
  stat="identity") +
  geom_errorbar(aes(ymin = , ymax=))
```

```
ggplot(data, aes(x=, y=, fill=)) +
  geom_bar(stat="identity")
```

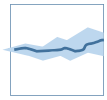
```
geom_histogram()
```



```
geom_boxplot()
  can add interactions:
  aes(x=interaction(var1, var2))
```

```
geom_area()
```

```
ggplot(data, aes(x=, y=, fill=)) +
  geom_area()
```

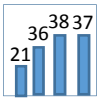


```
geom_ribbon()
  aes(ymin= , ymax=)
```

```
geom_abline(slope = 0 , intercept = 1)
geom_vline(xintercept= )
geom_hline(yintercept= )
```

Adding text

When the text is a variable in the data (e.g., labeling points in a scatter plot):




```
geom_text()
  ggplot(data, aes(x=, y=)) +
  geom_text(aes(label=y), vjust=-1.5)
  #vjust value is added to each y value
  size=5 #default
```

When adding annotation* (e.g., model equation):

```
annotate("text", label="r^2 == 0.42",
  parse = TRUE,
  x= , y= , colour= ,
  size= rel(), #relative to default
  family = 'Times', 'Helvetica', 'Courier',
  fontface= 'bold', 'italic', 'bold.italic',
  hjust = 0=left, 0.5=center, 1=right,
  vjust = 0=bottom, 0.5=middle, 1=top,
  angle = angle in degrees,
  lineheight = line spacing multiplier)
```

* can also use annotate to draw segments, rectangles, etc.

Step 3: Fine-tuning, axes and legends

To fully control the details of the plot (axes labels, legends, etc.), you need to use themes and elements. These are described in Step 5. But, here are some handy shortcuts.

Range of data

```
ylim(), xlim()
```

Labels

```
labs(x='label1', y='label2',
     title= 'plot title',
      fill = 'legend label')
```

Controlling tick-marks on plot

For a continuous variable

```
scale_y_continuous(limits=c(0, 10),  #range of data
                   breaks =c(), #location of breaks
                   labels = c()) #labels at breaks
```

For discrete variables, limits can be used to change the order of variables on the axis:

```
scale_x_discrete(limits=c('trt1', 'ctrl', 'trt2'))
```

Controlling the legend

Controlling discrete variable

```
scale_fill_discrete(
    limits=c('trt1', 'trt2', 'ctrl') #order of legend items
    names= "condition"  #title of legend
    guide_legend(title =NULL) #get rid of title
```

Reversing continuous variable

```
guides(fill=guide_legend(reverse=TRUE))
```

Step 4: Fine-tuning, color

color = lines and outlines of polygons
fill = area of polygons
For most points shapes, the color of the entire point is controlled by color, not fill. The exceptions are point shapes 21-25 with both a fill and outline.

RColorBrewer provides nice color palettes:

```
library(RColorBrewer)
display.brewer.all()
```

ColorBrewer provides discrete colors, and the maximum number of colors for any palette is limited. If more colors are needed:

```
myPalette <- colorRampPalette(brewer.pal(11,
"Spectral"), space = "Lab")
myPalette(100)
```

Website for Hex colors:
http://www.colorpicker.com/

Discrete variables	scale_fill_brewer scale_colour_brewer	scale_fill_brewer(palette="Oranges")
	scale_fill_manual scale_colour_manual	scale_color_manual(values=c("blue", "red"), limits=c("cold", "hot"), breaks=c(-Inf, 10, Inf))
Continuous variables	scale_fill_gradient scale_color_gradient	gradient between 2 colors: scale_color_gradient(low="black", high="white")
	scale_fill_gradient2 scale_color_gradient2	gradient with a midpoint: scale_color_gradient2(low=muted("red"), mid="white", high=muted("blue"), midpoint=100)
	scale_fill_gradientn scale_color_gradientn	gradient of n colors: scale_color_gradientn(colours=c("darkred", "orange", "yellow", "white") scale_fill_gradientn(brewer.pal(5, "Spectral"))

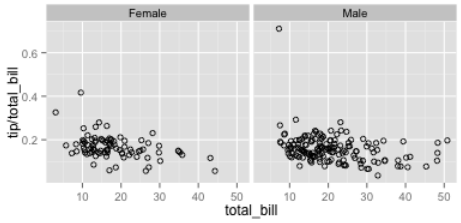
Step 4: Fine-tuning (themes and elements)

	Name	Description	Element type/use
General appearance	text	all text elements	element_text()
	rect	all rectangular elements	element_rect(fill = "white, colour=NA)
	line	all line elements	element_line()
	panel.background	background of plotting area	element_rect()
	panel.border	border around plotting area	element_line()
	panel.grid.major panel.grid.major.x panel.grid.major.y	major grid lines	element_line() cuts the x grid lines and alters appearance of y grid: theme(panel.grid.major.x = element_blank(), panel.grid.minor.x = element_blank(), panel.grid.major.y = element_line(color="grey60", linetype="dashed"))
	panel.grid.minor panel.grid.minor.x panel.grid.minor.y	minor grid lines	element_line()
	plot.background	background of entire plot	element_rect()
	plot.title	title text appearance	element_text()
Axis appearance	axis.line	lines along axes	element_line()
	axis.title axis.title.x axis.title.y	appearance of axis labels	element_text()
	axis.text axis.text.x axis.text.y	appearance of tick labels on axis	element_text() Change x labels to 30 degrees, align, and format: theme(axis.text.x = element_text(angle = 30, hjust = 1, vjust = 1, family = "Times", face ="italic", colour = "red", size = rel(0.9))) #relative to default size
Legend appearance	legend.background	background of legend	element_rect
	legend.text	legend text appearance	element_text
	legend.title	legend title	element_text
	legend.position	position of legend	Change position of legend theme(legend.position = c(x.loc, y.loc)) other options: "left", "right", "bottom", "top", "none"
Facet appearance	strip.background	background of facet labels	element_rect()
	strip.text strip.text.x strip.text.y	text appearance for facet labels	element_text()

element_text options	family	Helvetica, Times, Courier
	face	plain, bold, italic, bold.italic
	colour	color(name or '#RRGGBB')
	size	font size (use rel(0.7) to make it relative to default)
	hjust	horizontal alignment: 0=left, 0.5=center, 1=right
	vjust	vertical alignment: 0=bottom, 0.5=middle, 1=top
	angle	angle in degrees
	lineheight	line spacing multiplier
element_rect options	fill	fill color
	colour	border color
	size	border line thickness
	linetype	border linetype: 'solid', 'dashed', 'dotted', 'dotdash', 'longdash', 'twodash'
element_line options	colour	border color
	size	border line thickness
	linetype	type of line

Additional functionality

Faceting



facet_wrap(~variable, nrow =)
ncol =
scales="free" # scales each plot
scales="free_x" independently
scales="free_y"
facet_grid(y ~ x)
. ~ x
y ~ .

Fitting data

stat_smooth()
method = lm #default is loess spline
level=0.99 #99% CI
se = FALSE #no CI
method=glm, family=binomial #logistic

fitting data from an existing model:

```
model <- lm(x~y, data)
predicted <- data.frame(x=seq(1:100))
predicted$y <- predict(model, predicted)
sp <- ggplot(data, aes(x=x, y=y)) +
      geom_point()
sp + geom_line(data=predicted)
```

NOTE: expand.grid for multiple variables

Some commonly used data manipulations

```
subset(data, test %in% c('dog', 'cat'))
#categorizing a continuous variable:
cut(data, breaks=c(-Inf, 100, Inf),
        labels=c("<100", ">100"))
#remove unused levels from a factor:
droplevels(variable)
#change order of levels:
factor(sizes, levels=c("small", "medium", "large"))
#change names of factor levels:
plyr::revalue(sizes, c(small="s", medium="m", large="l"))
#make long/wide conversions: library(tidyr)
spread(data, variable_2_spread_on, variable_value)
gather(data, key=name_of_key_var,
value=name_of_value_var, columns_to_gather)
```