

Data Visualization in Base R

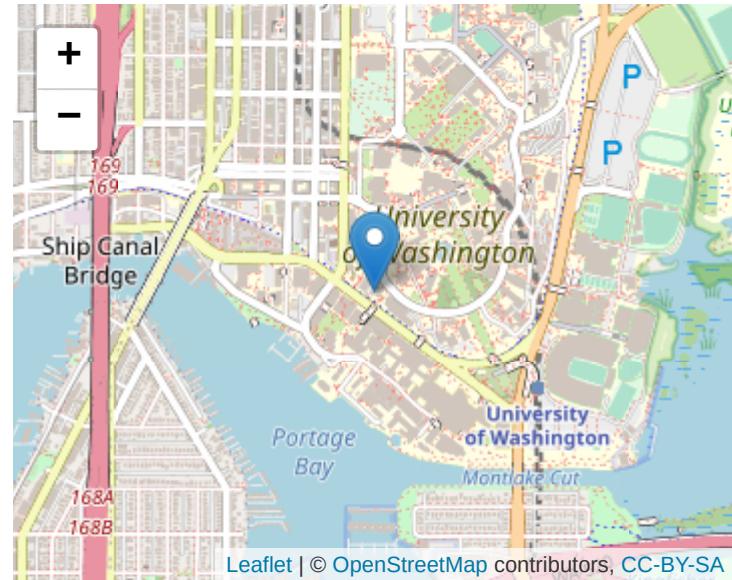
Adam Kuczynski



Why use R for data visualization?

- R data visualization is extremely flexible! Almost any data visualization you can think of is possible to create in R
- Creating visualizations in R allows you to create dynamic plots that change with new data. This is useful when you want to create plots on a recurring basis (e.g., monthly revenue reports) or even realize that you missed some data initially. Repeat the same code every time!
- There are dozens of packages that make it easier to create complex figures, including `ggplot2`, `patchwork`, `lattice`, `diagrammeR`, and more!
- Create interactive visualizations with `plotly`, `ggvis`, `htmlwidgets`, `leaflet`, shiny apps, and other R tools

🧠 Psychology's new home



The Generic `plot()` Function

Many data visualizations created in R start with the same function: `plot()`

`plot()` knows how to handle several different types of objects because it is a **generic function** with lots of methods:

```
methods(plot)
```

```
## [1] plot.acf*          plot.data.frame*    plot.decomposed.ts*  
## [4] plot.default        plot.dendrogram*   plot.density*  
## [7] plot.ecdf          plot.factor*       plot.formula*  
## [10] plot.function       plot.hclust*      plot.histogram*  
## [13] plot.HoltWinters*  plot.isoreg*      plot.lm*  
## [16] plot.medpolish*    plot.mlm*        plot.ppr*  
## [19] plot.prcomp*       plot.princomp*   plot.profile.nls*  
## [22] plot.R6*          plot.raster*     plot.spec*  
## [25] plot.stepfun       plot.stl*        plot.table*  
## [28] plot.ts           plot.tskernel*   plot.TukeyHSD*  
## see '?methods' for accessing help and source code
```

plot() Arguments

```
plot(x, y = NULL, type = "p", xlim = NULL, ylim = NULL,  
      log = "", main = NULL, sub = NULL, xlab = NULL, ylab = NULL,  
      ann = par("ann"), axes = TRUE, frame.plot = axes,  
      panel.first = NULL, panel.last = NULL, asp = NA,  
      xgap.axis = NA, ygap.axis = NA,  
      ...)
```

There are **a lot** of arguments to `plot()`!

Several of these arguments will be discussed in these slides, but not all of them. That means that making plots often involves teaching yourself something new each time with the help pages, Stack Overflow, and other various websites and blogs.

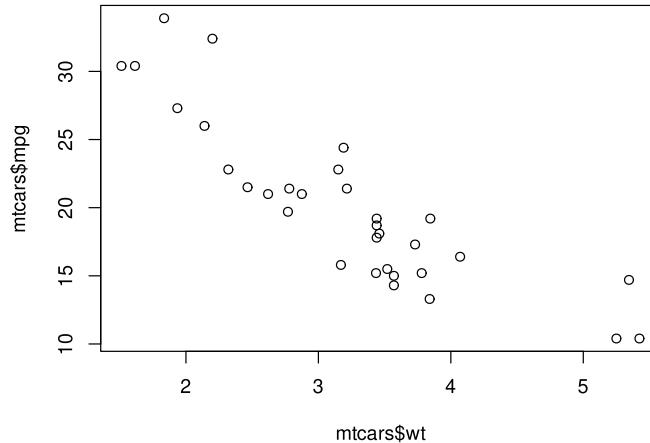
See `help(par)` for a full list of graphical parameters, many of which can be used within the `...` argument

Scatterplot

```
# Vectors of coordinates  
plot(x = mtcars$wt,  
     y = mtcars$mpg)
```

```
# Formula (`y~x`)  
plot(mtcars$mpg ~ mtcars$wt)
```

```
# Two-column dataframe of x, y coordinates  
plot(mtcars[, c("wt", "mpg")])
```

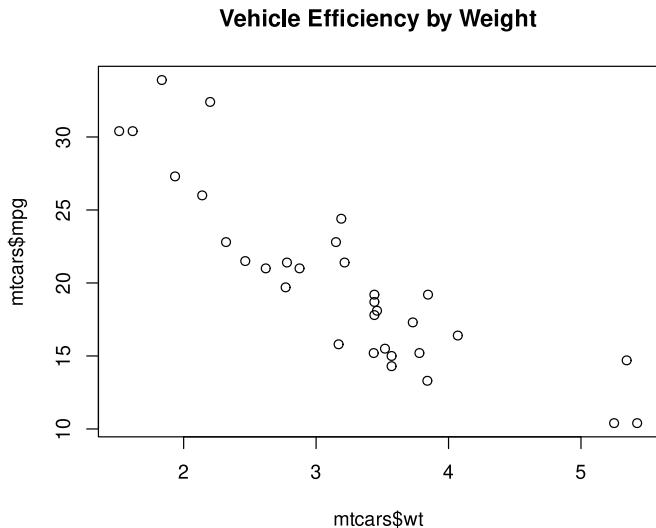


- What defaults do you notice?
 - Plots points (`type = "p"`) of a specific shape (`pch = 1`)
 - Axis labels (R code supplied to the arguments)
 - No header (`main = NULL`)
 - Chooses axis ticks for you
 - ...and hundreds more!

Plot Titles

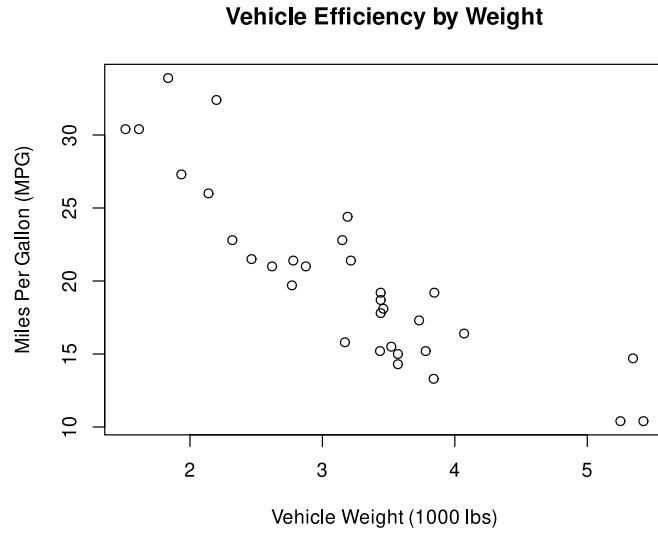
Main Title

```
plot(x = mtcars$wt, y = mtcars$mpg,  
     main = "Vehicle Efficiency by Weight")
```



Axis Titles

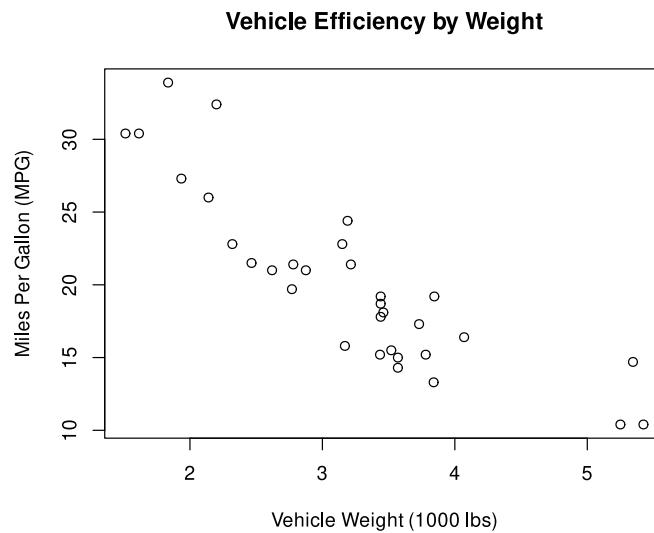
```
plot(x = mtcars$wt, y = mtcars$mpg,  
     main = "Vehicle Efficiency by Weight",  
     xlab = "Vehicle Weight (1000 lbs)",  
     ylab = "Miles Per Gallon (MPG)")
```



Fixing Ugly Axes

R default axes are not publication ready!

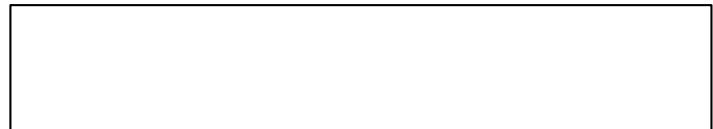
- The axes overlap with the box around the plot
- The y-axis tickmarks are vertical
- The axes are thin and hard to see
- The default tick marks may not be desirable
- The default tick labels may not be desirable



box()

The `box()` function is responsible for placing a box around your points. There are several different box types (specified with the `bty` **box type** argument):

```
# full box (default)
plot.new()
box(bty = "o")
```



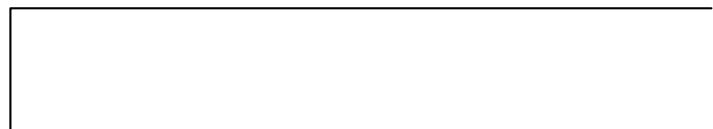
```
# bottom and left
plot.new()
box(bty = "L")
```



```
# top and right
plot.new()
box(bty = "7")
```



```
# top, left, and bottom
plot.new()
box(bty = "C")
```

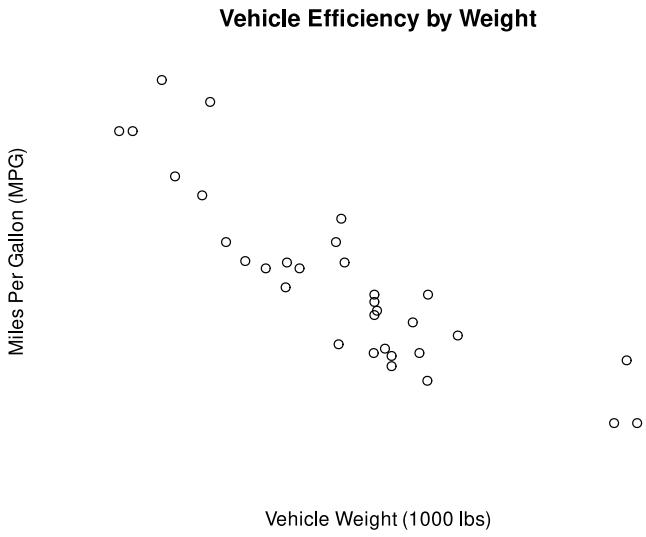


```
# left, bottom, and right
plot.new()
box(bty = "U")
```

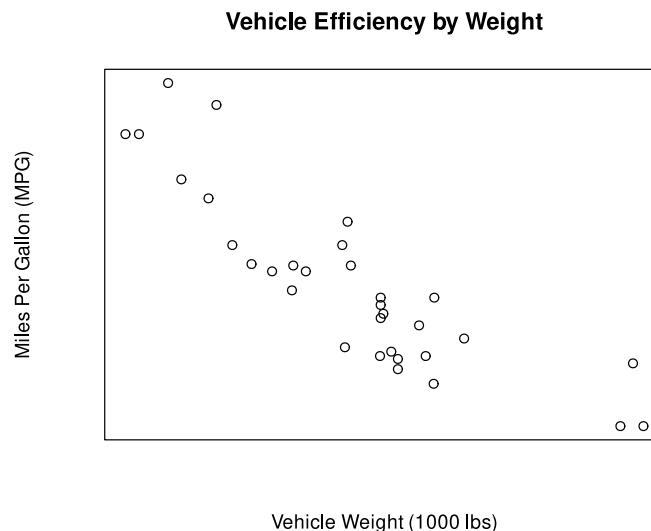


Fixing the `box()`

```
plot(x = mtcars$wt, y = mtcars$mpg,  
      main = "Vehicle Efficiency by Weight",  
      xlab = "Vehicle Weight (1000 lbs)",  
      ylab = "Miles Per Gallon (MPG)",  
      axes = F) # do not plot axes
```



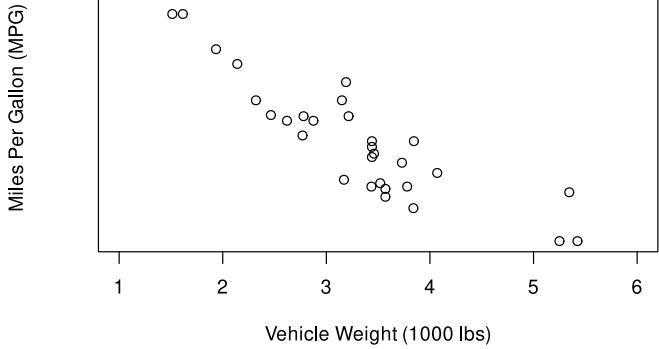
```
plot(x = mtcars$wt, y = mtcars$mpg,  
      main = "Vehicle Efficiency by Weight",  
      xlab = "Vehicle Weight (1000 lbs)",  
      ylab = "Miles Per Gallon (MPG)",  
      axes = F)  
box() # plot box
```



Add the axes back in

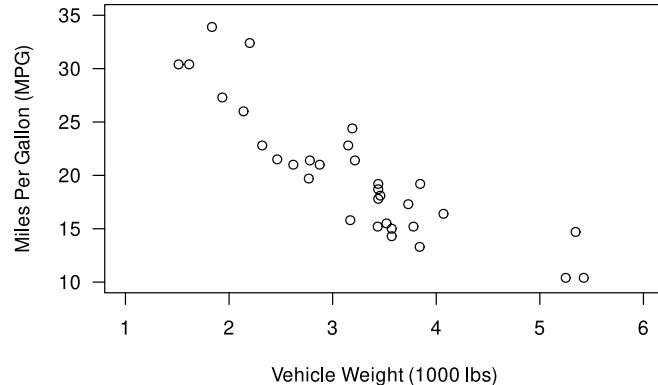
x-axis

```
plot(...,  
      axes = F,  
      xlim = c(1, 6))  
  
box()  
  
axis(side = 1, # x-axis  
     at = 1:6, # ticks at 1 through 6  
     labels = 1:6, # labels numbers 1 through 6  
     lwd = 0, # do not plot axis  
     lwd.ticks = 1) # plot tick marks
```



y-axis

```
plot(...,  
      axes = F,  
      xlim = c(1, 6),  
      ylim = c(10, 35))  
box()  
axis(side = 1, at = 1:6, labels = 1:6, lwd = 1)  
axis(side = 2, # y-axis  
     at = seq(10, 35, 5), # ticks every 5 points  
     labels = seq(10, 35, 5), # labels 10:35  
     lwd = 0, lwd.ticks = 1,  
     las = 1) # horizontal tick labels
```



Change box() width and line type

```
box(bty = "o",  
    lwd = 3,  
    lty = 1)
```



```
box(bty = "o",  
    lwd = 3,  
    lty = 2)
```



```
box(bty = "o",  
    lwd = 3,  
    lty = 3)
```



```
box(bty = "o",  
    lwd = 3,  
    lty = 4)
```



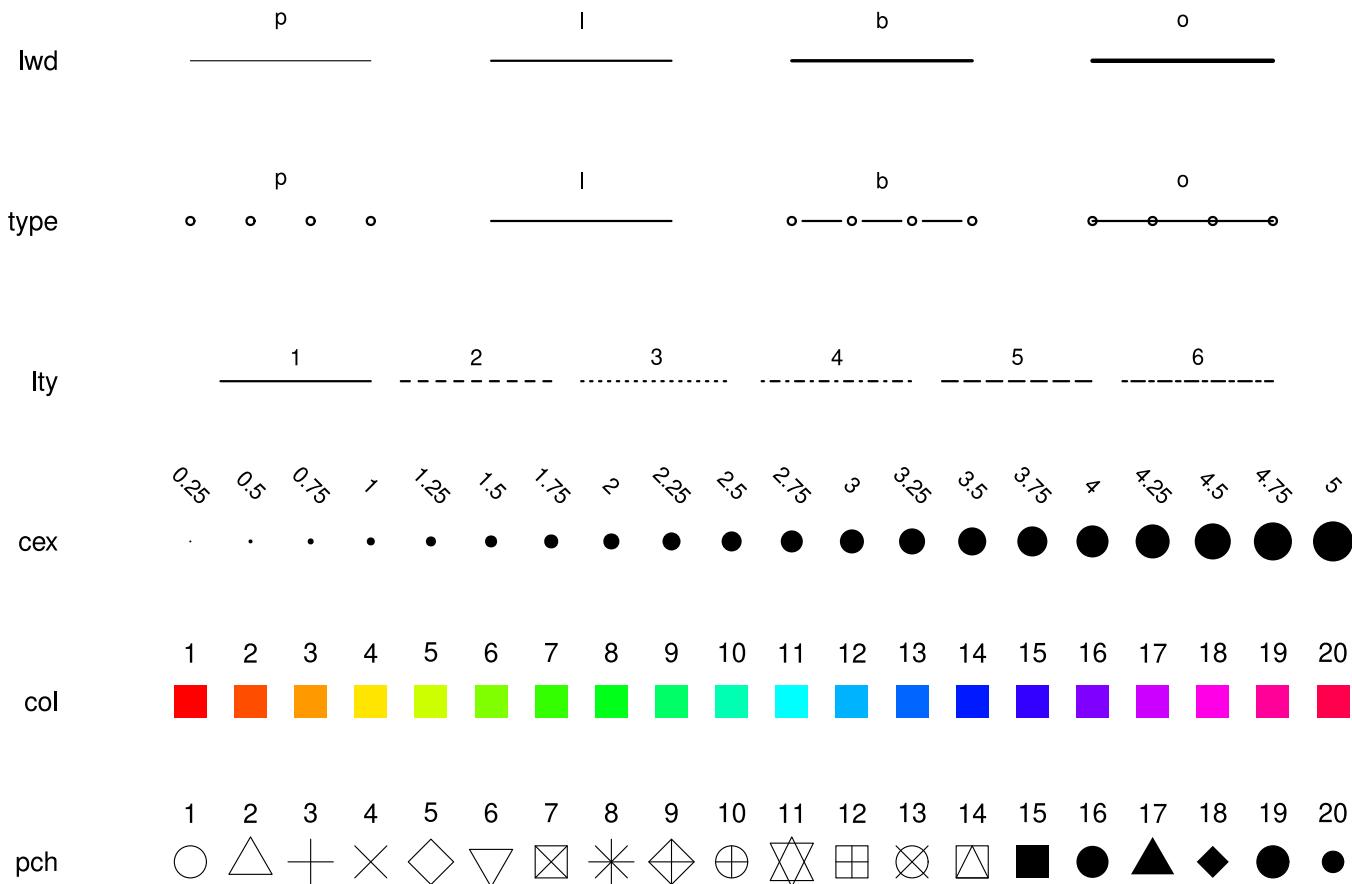
```
box(bty = "o",  
    lwd = 3,  
    lty = 5)
```



```
box(bty = "o",  
    lwd = 3,  
    lty = 6)
```

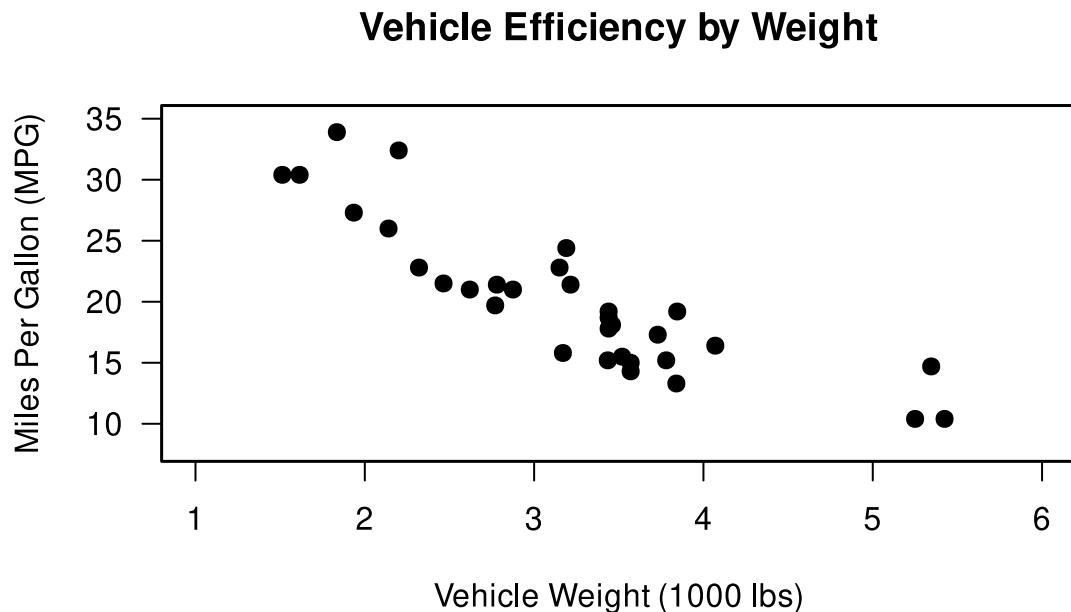


Point and Line Types



Adjusting Point Shape with `pch`

```
plot(x = mtcars$wt, y = mtcars$mpg,  
      main = "Vehicle Efficiency by Weight",  
      xlab = "Vehicle Weight (1000 lbs)",  
      ylab = "Miles Per Gallon (MPG)",  
      cex = 1.25, # size  
      pch = 16) # shape
```

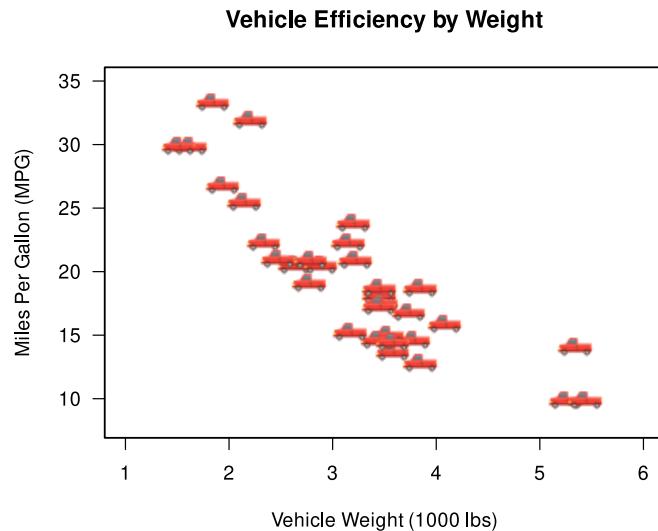


pch is customizable

```
plot(x = mtcars$wt, y = mtcars$mpg,  
      main = "Vehicle Efficiency by Weight",  
      xlab = "Vehicle Weight (1000 lbs)",  
      ylab = "Miles Per Gallon (MPG)",  
      cex = 1.5,  
      pch = "+")
```



```
plot(x = mtcars$wt, y = mtcars$mpg,  
      main = "Vehicle Efficiency by Weight",  
      xlab = "Vehicle Weight (1000 lbs)",  
      ylab = "Miles Per Gallon (MPG)",  
      cex = 1.5,  
      pch = "🚚") # truck emoji
```

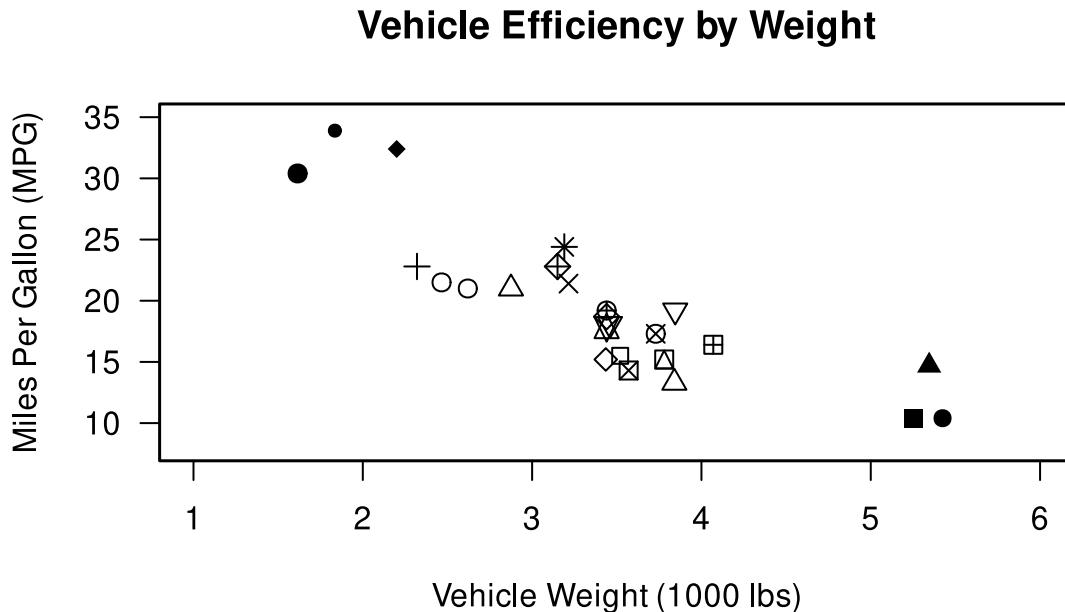


pch is vectorized

```
plot(x = mtcars$wt, y = mtcars$mpg,  
      main = "Vehicle Efficiency by Weight",  
      xlab = "Vehicle Weight (1000 lbs)",  
      ylab = "Miles Per Gallon (MPG)",  
      cex = 1.25,  
      pch = 1:nrow(mtcars))
```

👉 Because there are only 1:20 valid values of pch, R will give you a warning and recycle the 1:20 vector

👉 Each value 1:20 maps onto the element passed to x and y



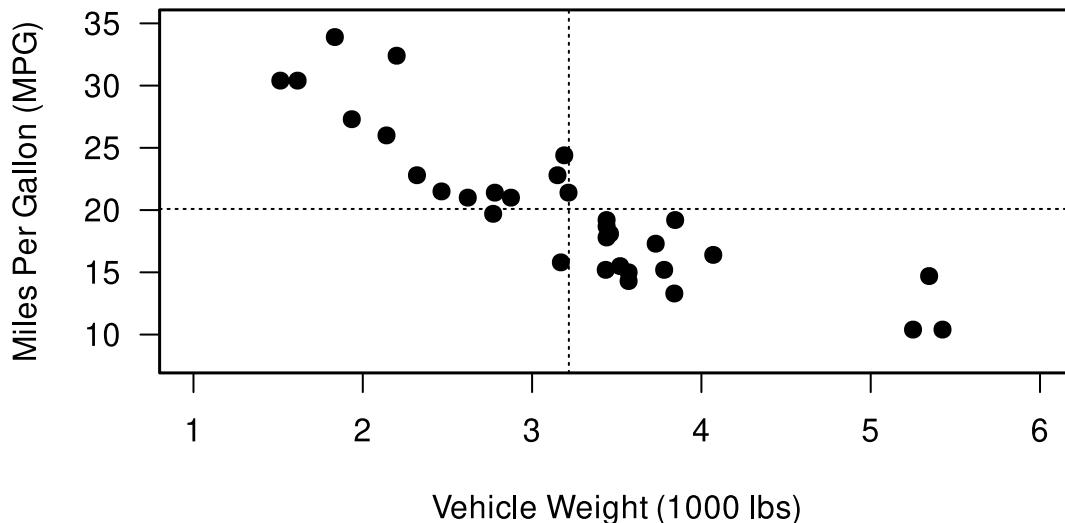
Adding lines to your plot

There are *many* ways to add lines to a plot in R. Some of the most common lines are vertical or horizontal lines, regression lines, and local regression (LOESS) lines.

The `abline()` function can take either **(a)** a fitted regression object, **(b)** the intercept (`a`) and slope (`b`) values, **(c)** a y-axis value for horizontal lines (`h`), or **(d)** an x-axis value for vertical lines (`v`).

For example, to add lines at the mean of x and y:

```
plot(...)  
abline(v = mean(mtcars$wt, na.rm = T),  
       h = mean(mtcars$mpg, na.rm = T),  
       lty = 3)
```



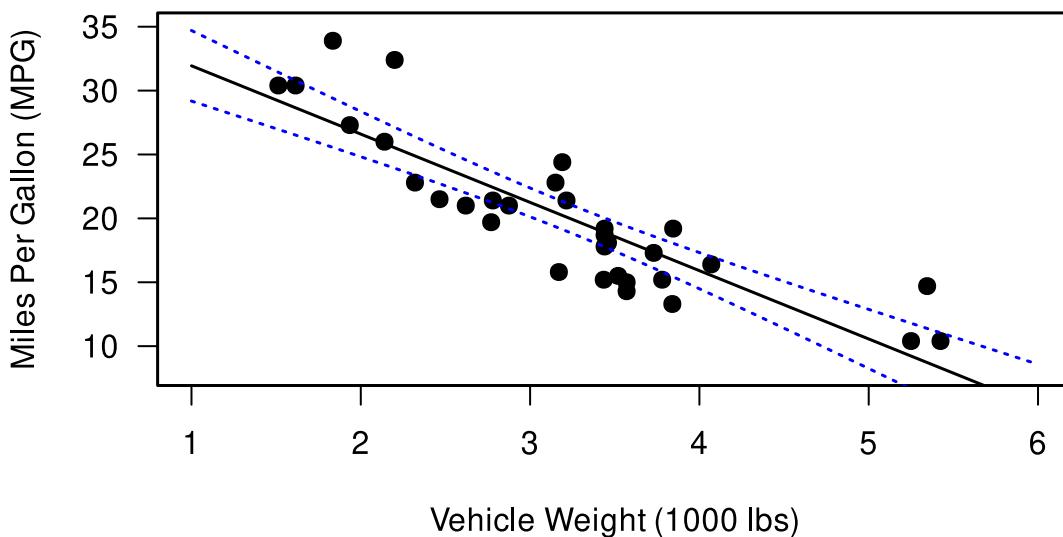
matlines()

```
plot(...)

fit <- lm(mpg ~ wt,
          data = mtcars)

new_wt <- seq(1, 6, .05)
pred <- predict(fit,
                newdata = data.frame(wt = new_wt),
                interval = "confidence",
                level = 0.95)

# Plots multiple lines based on a matrix of cols (with x and y line coords)
matlines(new_wt, pred, lty = c(1, 3, 3), lwd = 1.5, col = c("black", "blue", "blue"))
```



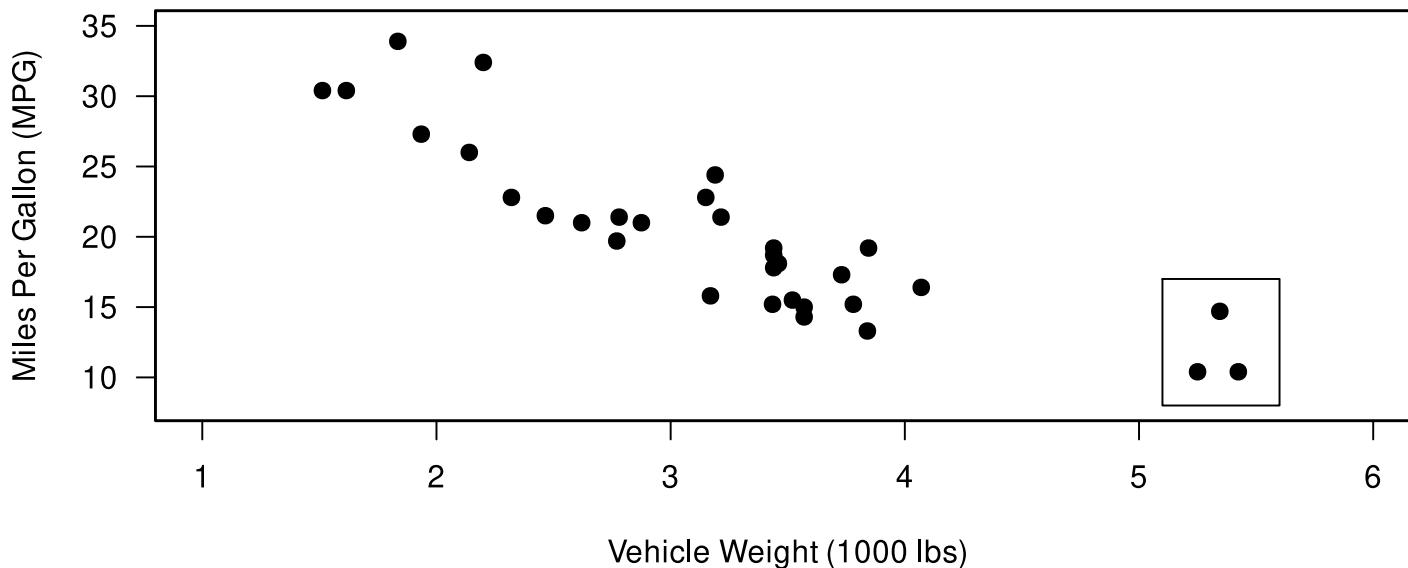
lines()

The `lines()` function is a generic function that takes either `x` and `y` coordinates to plot a line (similar to `matlines()`) or a formula to compute these coordinates

For example, let's draw a box around the three heaviest cars:

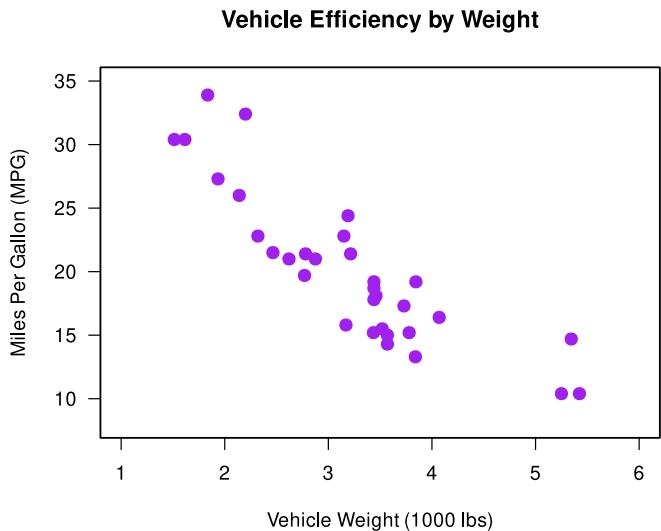
```
plot(...)

lines(x = c(5.1, 5.6, 5.6, 5.1, 5.1),
      y = c(17, 17, 8, 8, 17))
```

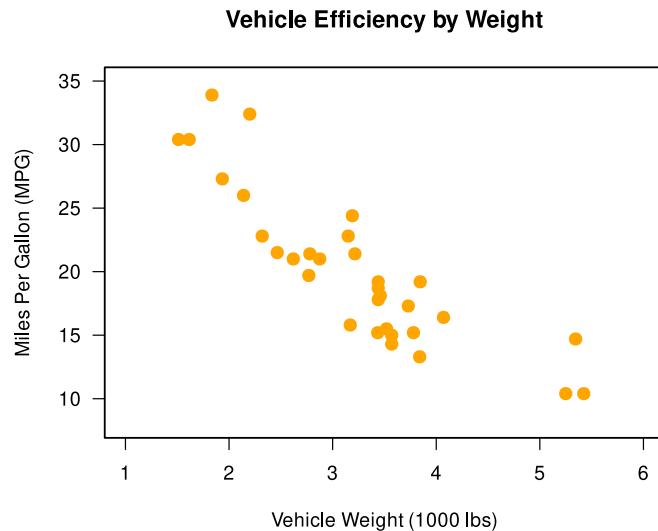


Adjusting Point Colors with `col`

```
plot(x = mtcars$wt, y = mtcars$mpg,  
      main = "Vehicle Efficiency by Weight",  
      xlab = "Vehicle Weight (1000 lbs)",  
      ylab = "Miles Per Gallon (MPG)",  
      cex = 1.5,  
      pch = 16,  
      col = "purple")
```



```
plot(x = mtcars$wt, y = mtcars$mpg,  
      main = "Vehicle Efficiency by Weight",  
      xlab = "Vehicle Weight (1000 lbs)",  
      ylab = "Miles Per Gallon (MPG)",  
      cex = 1.5,  
      pch = 16,  
      col = "orange")
```



R's Colors

R has built-in colors that can be accessed by name or by index with the `col` argument (we did this in the previous slide). To see the list of all 657 colors, use the `colors()` function, or [see this PDF](#).

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25
26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48	49	50
51	52	53	54	55	56	57	58	59	60	61	62	63	64	65	66	67	68	69	70	71	72	73	74	75
76	77	78	79	80	81	82	83	84	85	86	87	88	89	90	91	92	93	94	95	96	97	98	99	100
101	102	103	104	105	106	107	108	109	110	111	112	113	114	115	116	117	118	119	120	121	122	123	124	125
126	127	128	129	130	131	132	133	134	135	136	137	138	139	140	141	142	143	144	145	146	147	148	149	150
151	152	153	154	155	156	157	158	159	160	161	162	163	164	165	166	167	168	169	170	171	172	173	174	175
176	177	178	179	180	181	182	183	184	185	186	187	188	189	190	191	192	193	194	195	196	197	198	199	200
201	202	203	204	205	206	207	208	209	210	211	212	213	214	215	216	217	218	219	220	221	222	223	224	225
226	227	228	229	230	231	232	233	234	235	236	237	238	239	240	241	242	243	244	245	246	247	248	249	250
251	252	253	254	255	256	257	258	259	260	261	262	263	264	265	266	267	268	269	270	271	272	273	274	275
276	277	278	279	280	281	282	283	284	285	286	287	288	289	290	291	292	293	294	295	296	297	298	299	300
301	302	303	304	305	306	307	308	309	310	311	312	313	314	315	316	317	318	319	320	321	322	323	324	325
326	327	328	329	330	331	332	333	334	335	336	337	338	339	340	341	342	343	344	345	346	347	348	349	350
351	352	353	354	355	356	357	358	359	360	361	362	363	364	365	366	367	368	369	370	371	372	373	374	375
376	377	378	379	380	381	382	383	384	385	386	387	388	389	390	391	392	393	394	395	396	397	398	399	400
401	402	403	404	405	406	407	408	409	410	411	412	413	414	415	416	417	418	419	420	421	422	423	424	425
426	427	428	429	430	431	432	433	434	435	436	437	438	439	440	441	442	443	444	445	446	447	448	449	450
461	462	463	464	465	466	467	468	469	470	471	472	473	474	475										
476	477	478	479	480	481	482	483	484	485	486	487	488	489	490	491	492	493	494	495	496	497	498	499	500
501	502	503	504	505	506	507	508	509	510	511	512	513	514	515	516	517	518	519	520	521	522	523	524	525
526	527	528	529	530	531	532	533	534	535	536	537	538	539	540	541	542	543	544	545	546	547	548	549	550
551	552	553	554	555	556	557	558	559	560	561	562	563	564	565	566	567	568	569	570	571	572	573	574	575
576	577	578	579	580	581	582	583	584	585	586	587	588	589	590	591	592	593	594	595	596	597	598	599	600
601	602	603	604	605	606	607	608	609	610	611	612	613	614	615	616	617	618	619	620	621	622	623	624	625
626	627	628	629	630	631	632	633	634	635	636	637	638	639	640	641	642	643	644	645	646	647	648	649	650
651	652	653	654	655	656	657	658	659	660	661	662	663	664	665	666	667								

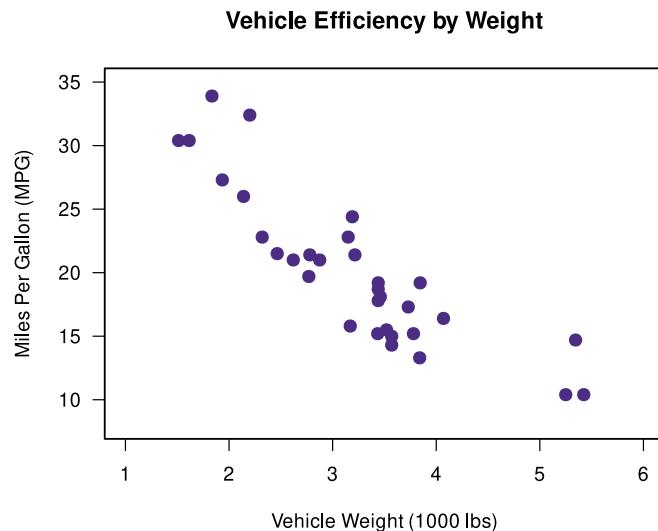
HTML Color Codes

R can also take hex codes or RGB (red, blue, green) color codes, which gives you access to infinite colors. Use [this tool](#) to help you find exactly the color you want.

When you use RGB color codes you can also specify the **alpha channel**, which gives the colors transparency (this is also possible with HEX codes, just harder)

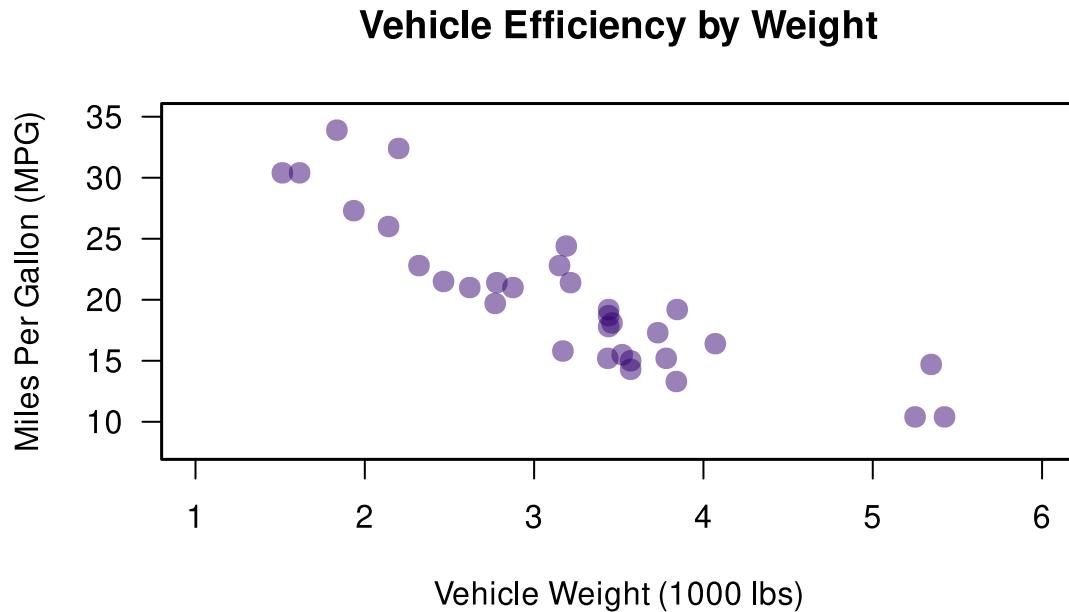
```
plot(x = mtcars$wt, y = mtcars$mpg,  
      main = "Vehicle Efficiency by Weight",  
      xlab = "Vehicle Weight (1000 lbs)",  
      ylab = "Miles Per Gallon (MPG)",  
      cex = 1.5, pch = 16,  
      col = "#4b2e83") # UW purple
```

```
plot(x = mtcars$wt, y = mtcars$mpg,  
      main = "Vehicle Efficiency by Weight",  
      xlab = "Vehicle Weight (1000 lbs)",  
      ylab = "Miles Per Gallon (MPG)",  
      cex = 1.5, pch = 16,  
      col = rgb(51, 0, 111, # UW purple  
                maxColorValue = 255))
```



Transparency with Alpha

```
plot(x = mtcars$wt, y = mtcars$mpg,  
      main = "Vehicle Efficiency by Weight",  
      xlab = "Vehicle Weight (1000 lbs)",  
      ylab = "Miles Per Gallon (MPG)",  
      cex = 1.5, pch = 16,  
      col = rgb(51, 0, 111, 255*.5, # 0 = transparent, 1 = completely opaque  
                maxColorValue = 255))
```



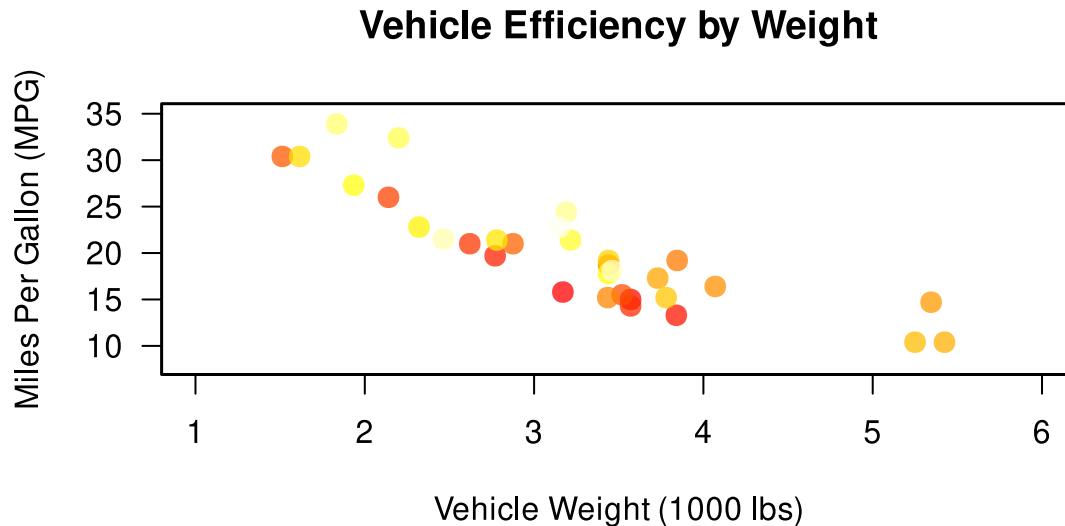
R's Color Functions

- `rainbow()`: n colors (with `alpha` transparency) corresponding with the rainbow color spectrum
- `heat.colors()`: n colors (with `alpha` transparency) ranging from red to light yellow
- `terrain.colors()`: n colors (with `alpha` transparency) corresponding with terrain map colors
- `topo.colors()`: n colors (with `alpha` transparency) corresponding with topography map colors
- `cm.colors()`: n colors (with `alpha` transparency) ranging from cyan to magenta
- `hcl()`: create vector of colors from vectors specifying hue (`h`), chroma (`c`), and luminance (`l`)
- RColorBrewer: An R package with convenient color scheme and functions. See `RColorBrewer::display.brewer.all()` to plot the available color palettes.

col is vectorized

The `col` argument is vectorized, which means you can do things like create another dimension (color) in your data to represent more information. For example, using `heat.colors()`, we will color the fastest 1/4 mile time red and the slowest light yellow:

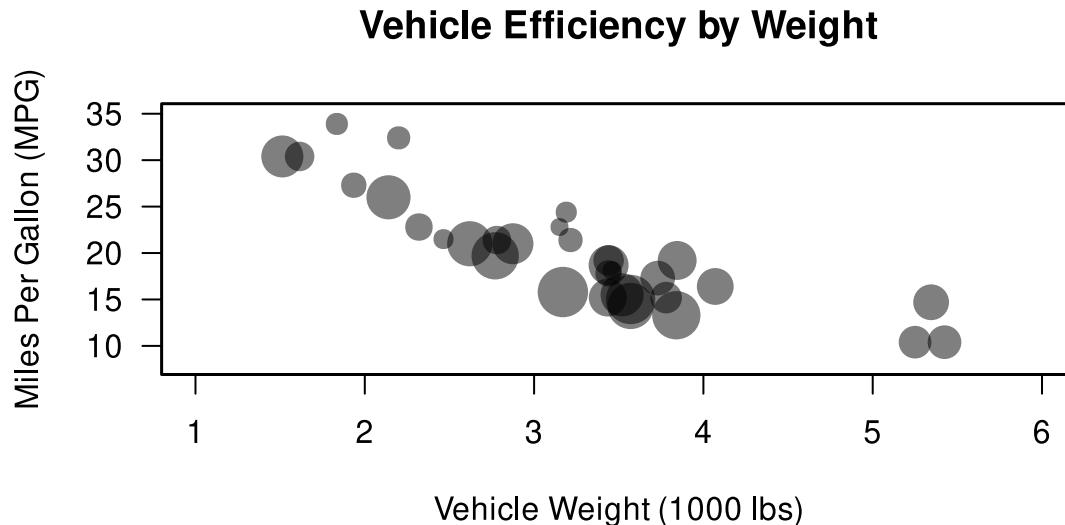
```
plot(x = mtcars[order(mtcars$qsec), c("wt", "mpg")], # order rows by qsec  
      main = "Vehicle Efficiency by Weight",  
      xlab = "Vehicle Weight (1000 lbs)",  
      ylab = "Miles Per Gallon (MPG)",  
      pch = 16, cex = 1.50,  
      col = heat.colors(length(mtcars$wt), alpha = .75))
```



cex is vectorized

The `cex` argument is also vectorized and can be used to create new dimensions in your figures. In this plot, larger points represent faster 1/4 mile times.

```
plot(x = mtcars[order(mtcars$qsec), c("wt", "mpg")], # order rows by qsec  
      main = "Vehicle Efficiency by Weight",  
      xlab = "Vehicle Weight (1000 lbs)",  
      ylab = "Miles Per Gallon (MPG)",  
      pch = 16,  
      cex = seq(3.5, 1.25, length.out = nrow(mtcars)),  
      col = rgb(0, 0, 0, .5))
```

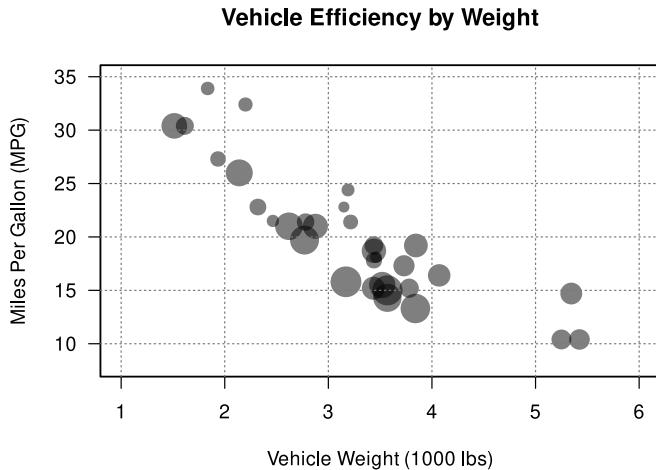


Creating Grids

```
grid(nx = NULL, ny = nx, col = "lightgray", lty = "dotted",
      lwd = par("lwd"), equilogs = TRUE)
```

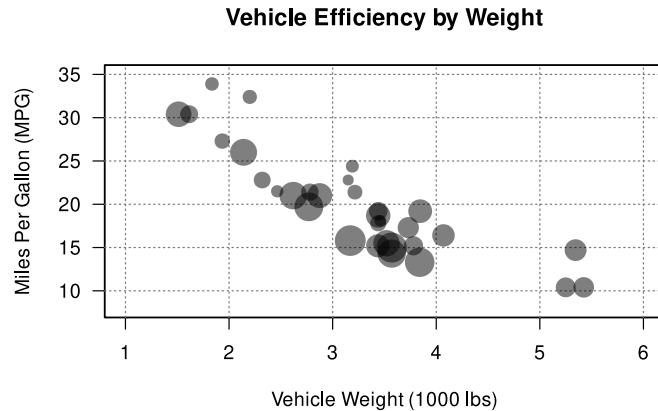
By default the number of lines in the x and y directions will match the number of tick marks

```
plot(...)  
grid(col = "gray48")
```



👉 Problem: gridlines placed on top of the points

```
plot(..., type = "n") # do not plot points  
grid(col = "gray48")  
points(x = mtcars[order(mtcars$qsec), c("wt"  
  pch = 16,  
  cex = seq(3.5, 1.25, length.out = nr  
  col = rgb(0, 0, 0, .5))
```

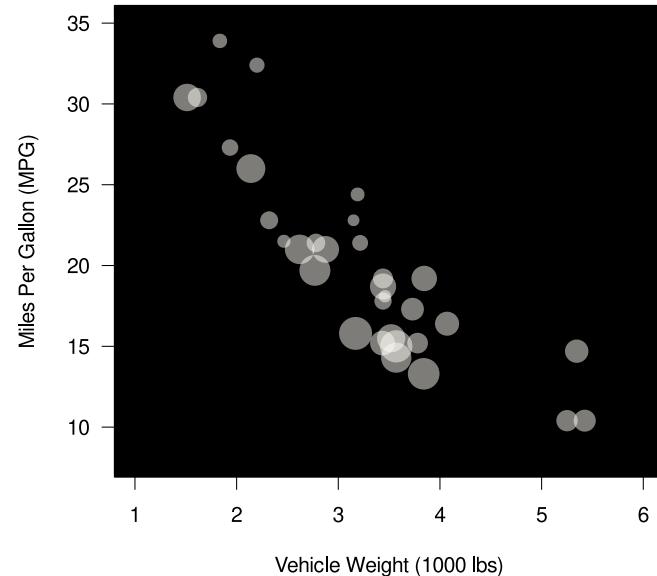


Plot Background

To change the background of *just the plot region* (i.e., where the points go), you need to:

1. Create an empty plot
2. Create a rectangle using `rect()` of the plotting region
3. Create points (and anything else you want to plot) over the rectangle

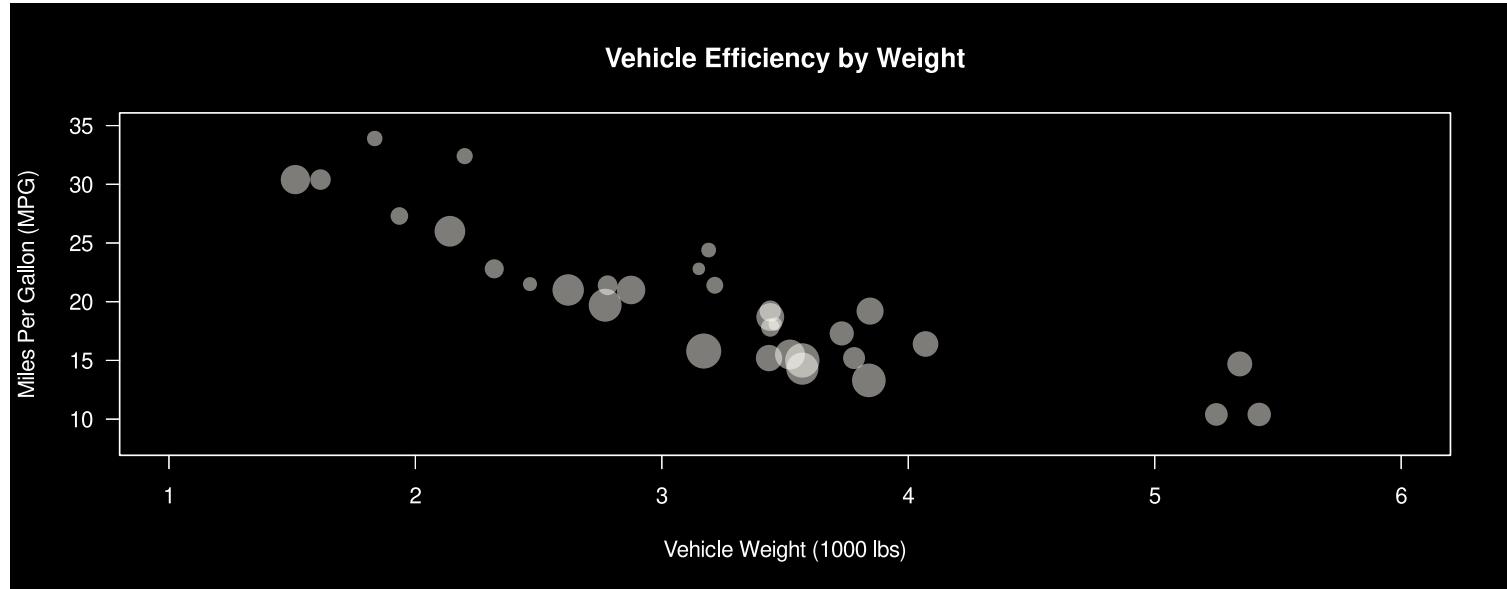
```
plot(..., type = "n")  
  
rect(xleft = par("usr")[1],  
      ybottom = par("usr")[3],  
      xright = par("usr")[2],  
      ytop = par("usr")[4],  
      col = "black")  
  
points(x = mtcars[order(mtcars$qsec), c("wt"  
                                         pch = 16,  
                                         cex = seq(3.5, 1.25, length.out = nr  
                                         col = rgb(255, 252, 245, 255*.5, max
```



Plotting Area Background

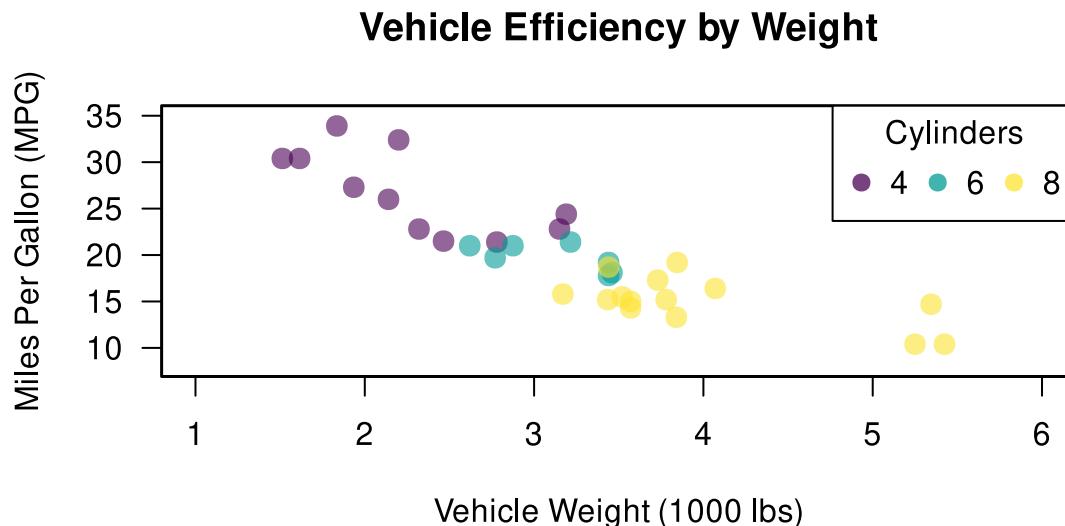
Changing the background of the entire plotting area is much easier than changing just the plotting region

```
par("bg" = "black", # Change background color  
    "fg" = "white") # Change foreground color (box, axes, tick marks)  
  
plot(...,  
      col = "white",      # Points  
      col.main = "white", # Main title  
      col.lab = "white",  # Axis labels  
      col.axis = "white") # Tick labels
```



Legends

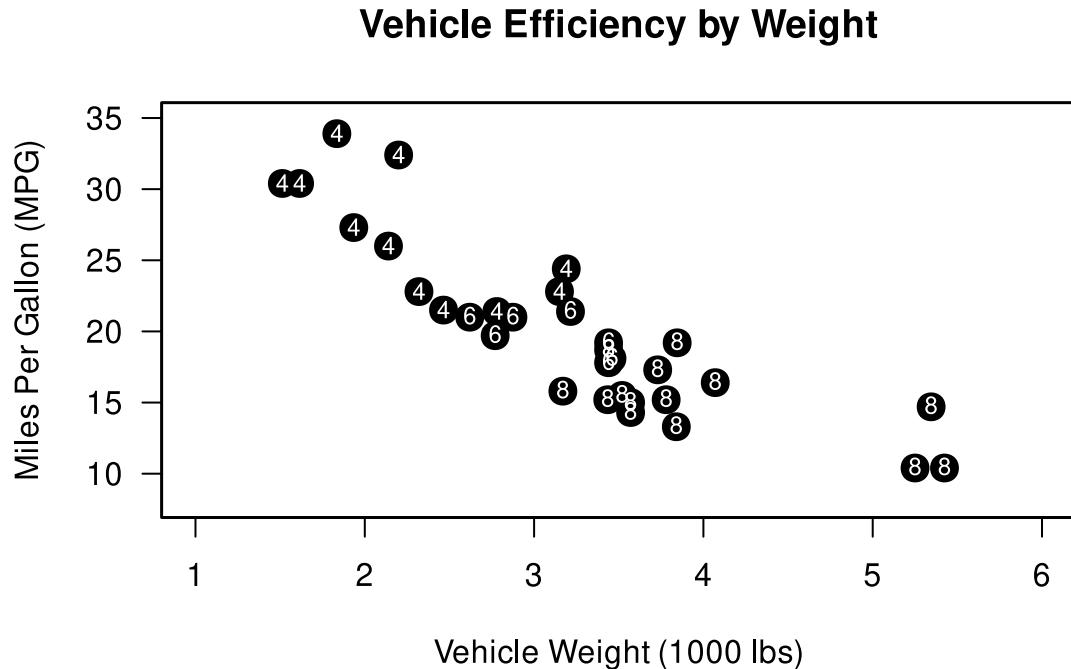
```
plot(x = mtcars[order(mtcars$cyl), c("wt", "mpg")], # order rows by cyl  
      col = rep(hcl.colors(3, alpha = .6), times = table(mtcars$cyl)), # color by cyl  
      ...) # color by cyl  
  
legend(x = "topleft", # takes keywords OR x, y coordinates  
       title = "Cylinders", # legend title  
       legend = seq(4, 8, 2), # values inside legend  
       col = hcl.colors(3, alpha = .75), # colors corresponding with values  
       horiz = T, # plot legend horizontally  
       pch = 16) # shape of legend point
```



Adding Text to Plots

```
plot(...)

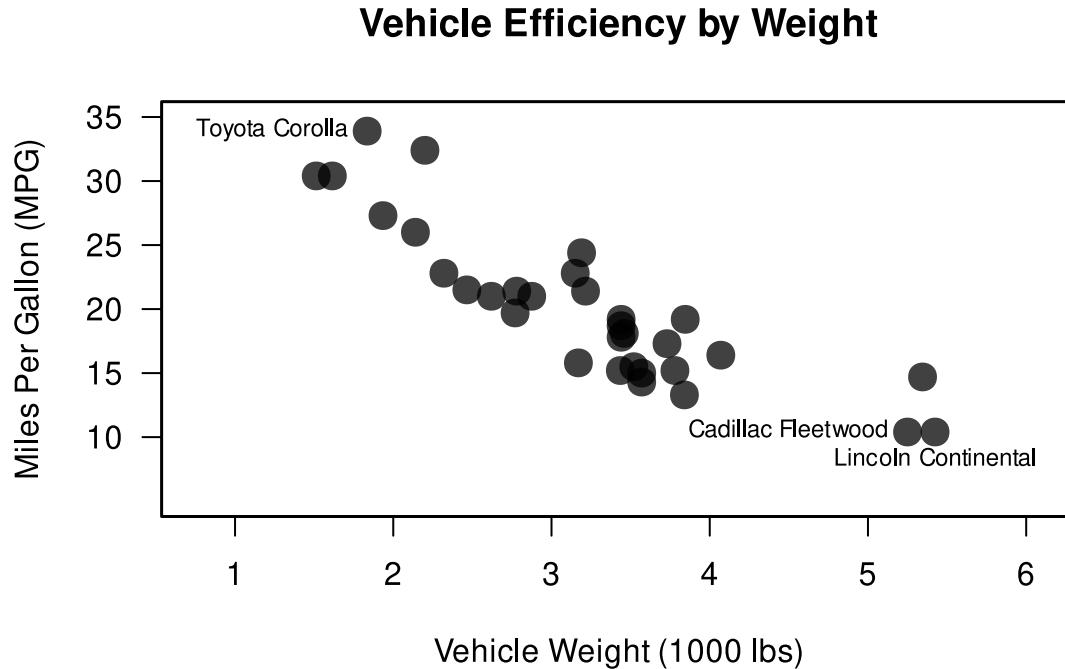
text(x = mtcars[, c("wt", "mpg")], # x,y coordinates of labels
     labels = mtcars$cyl,
     col = "white",
     cex = .75)
```



Labeling Specific Points

```
plot(...)

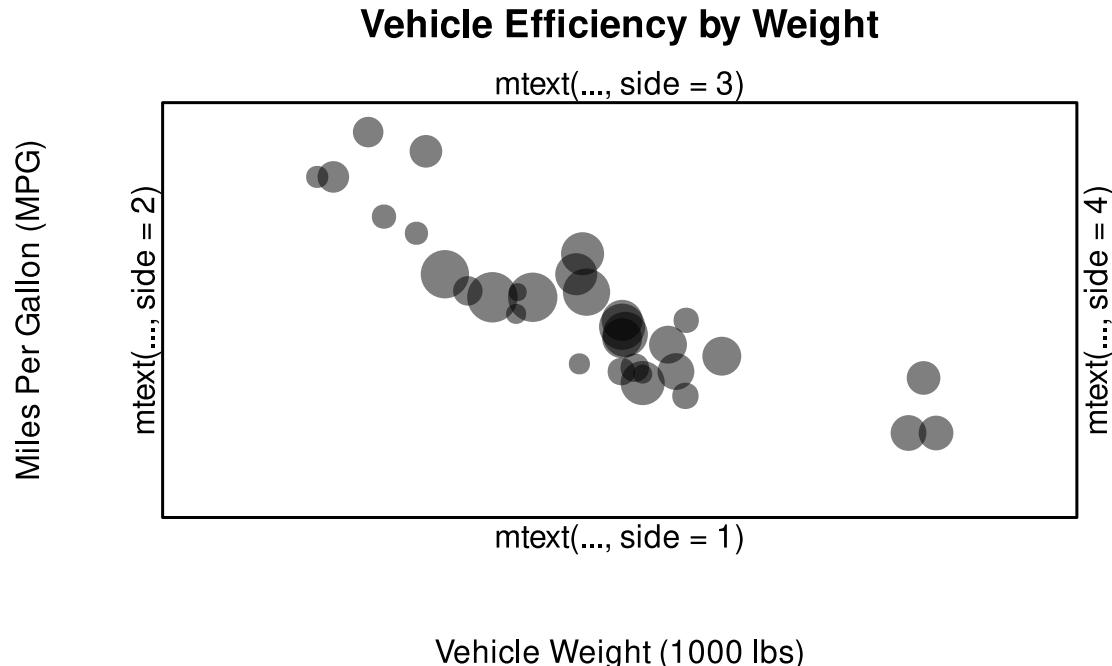
text(x = mtcars[mtcars$mpg %in% c(min(mtcars$mpg), max(mtcars$mpg)), c("wt", "mpg")],
     labels = rownames(mtcars)[mtcars$mpg %in% c(min(mtcars$mpg), max(mtcars$mpg))],
     cex = .75,
     pos = c(2, 1, 2))
```



Margin text with `mtext()`

Sometimes you want to put text in the margins of the plot (e.g., when you have multiple plots and you want to give them all one title). For that you can use the `mtext()` function (for **margin text**).

```
for(i in 1:4){  
  mtext(paste0("mtext(..., side = ", i, ")"), side = i)  
}
```



Changing Fonts

Changing font size and style is easy, but changing font family is a bit trickier because it depends on the fonts you have installed on your operating system

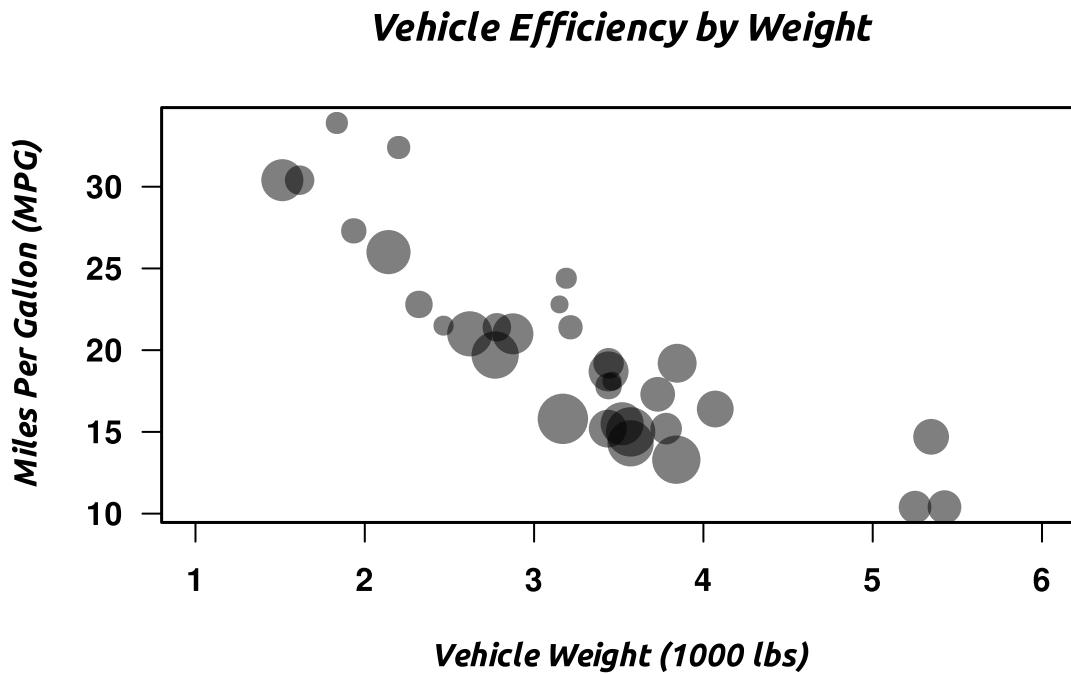
The `extrafont` package extends the fonts available for plotting in R:

1. Install the package `install.packages("extrafont")`
 2. Import the fonts: `extrafont::font_import()`
-

Font `par` arguments:

- `font`: Integer which specifies which font style to use for text
 - 1 = plain
 - 2 = **bold**
 - 3 = *italic*
 - 4 = **bold italic**
- `font.axis`: Integer which specifies which font to use for axis annotation
- `font.lab`: Integer which specifies which font to use for x and y labels (axis labels)
- `font.main`: Integer which specifies which font to use for main titles
- `font.sub`: Integer which specifies which font to use for subtitles

```
plot(...,  
    family = "Ubuntu", # Ubuntu font  
    font.main = 4, # Title (bold, italic)  
    font.axis = 2, # Axis tick mark labels (bold)  
    font.lab = 4) # Axis labels (bold italic)
```



Multiple Plots in Same Window

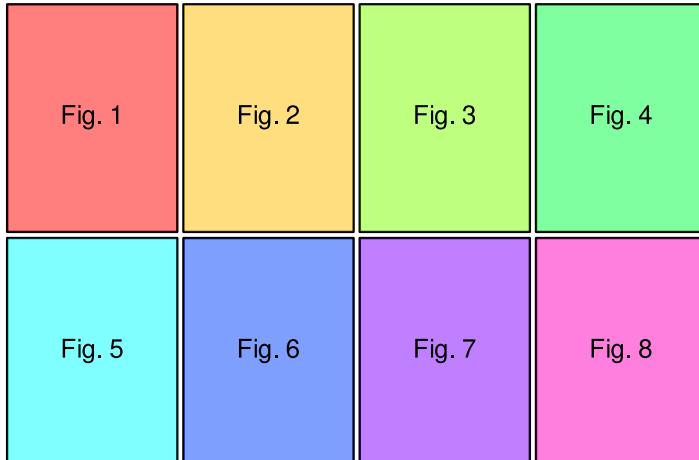
There are two primary ways of creating multiple figures within the same window in R:

- `par`'s `mfrow` and `mfcol` arguments
- the `layout()` function

`mfrow` and `mfcol`

These functions take a vector of two elements (nrow, ncol) and draw a grid on the graphing screen that is filled with figures by row (`mfrow`) or by column (`mfcol`)

```
par(mfrow = c(2, 4))
```

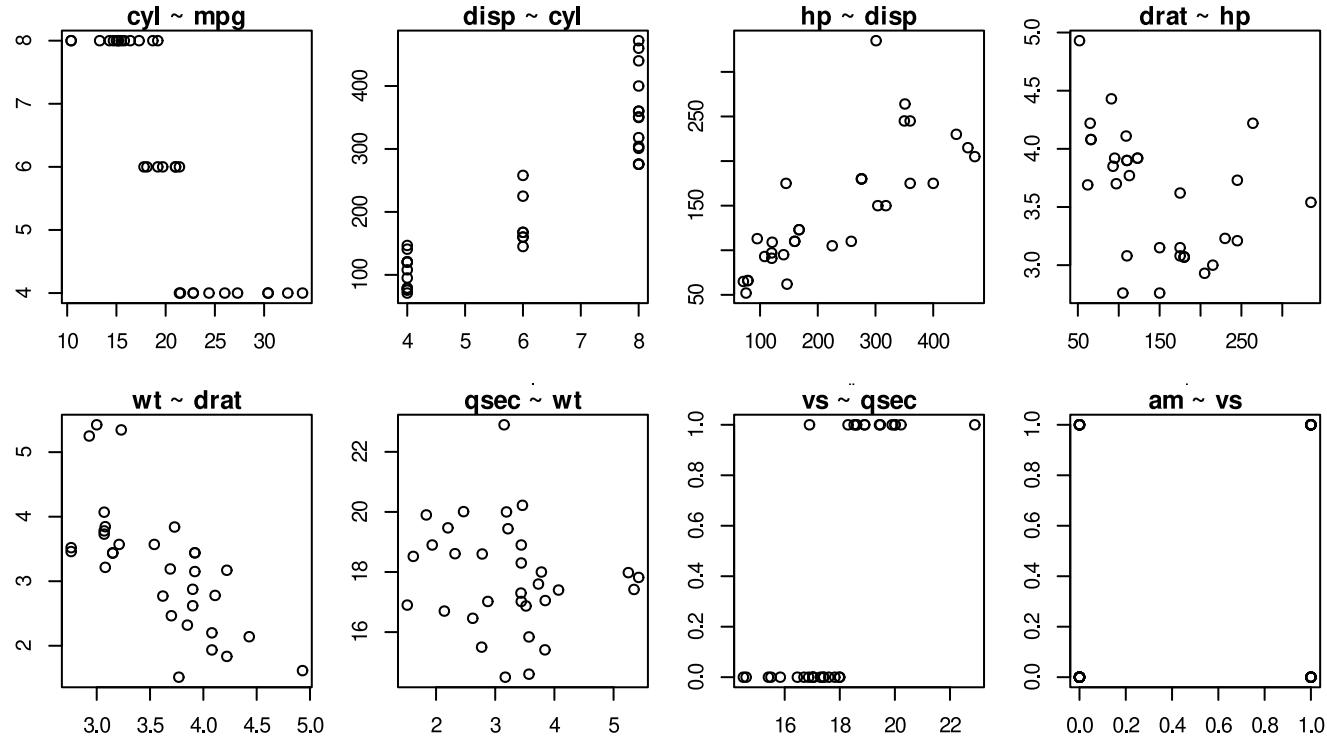


```
par(mfcol = c(2, 4))
```



mfrow/mfcol example:

```
par(mfrow = c(2, 4))
for(i in 1:8){
  plot(mtcars[, c(i, i+1)],
       main = paste(colnames(mtcars[, c(i+1, i)]),
                     collapse = " ~ "))
}
```

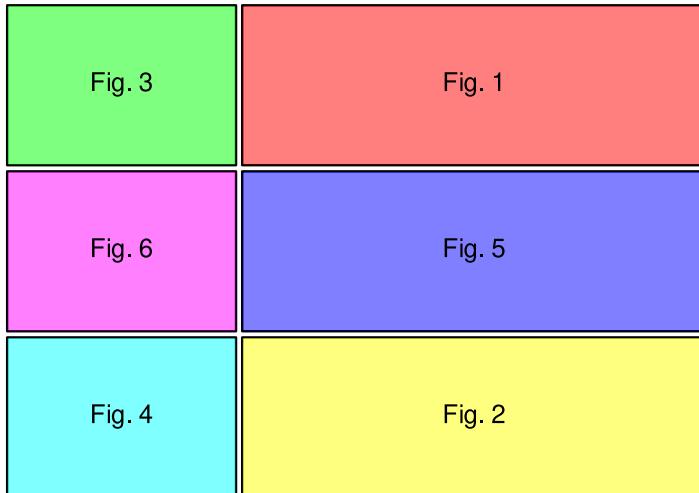


Multiple figures with `layout()`

The `layout` function takes a matrix that specifies the location of the next N figures created **and** the order in which they will be placed. The `widths` and `heights` arguments take the relative (or in centimeters if you prefer) row/column widths. For example:

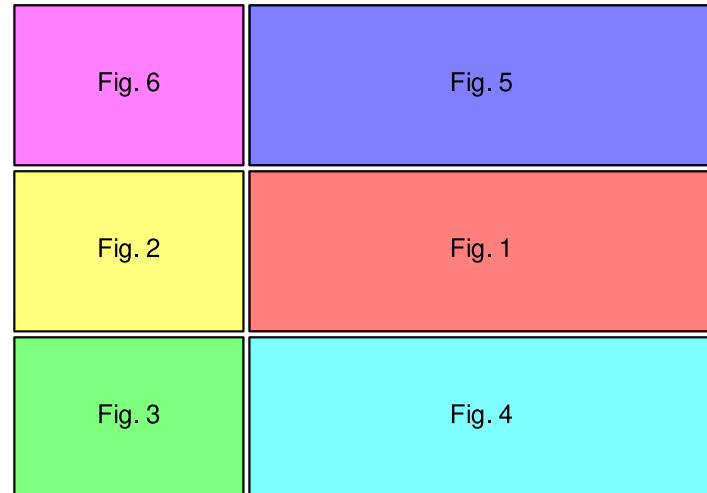
```
layout_mat <- matrix(c(3, 1,
                      6, 5,
                      4, 2),
                     nrow = 3,
                     byrow = T)

layout(layout_mat, c(1, 2), c(1, 1, 1))
```



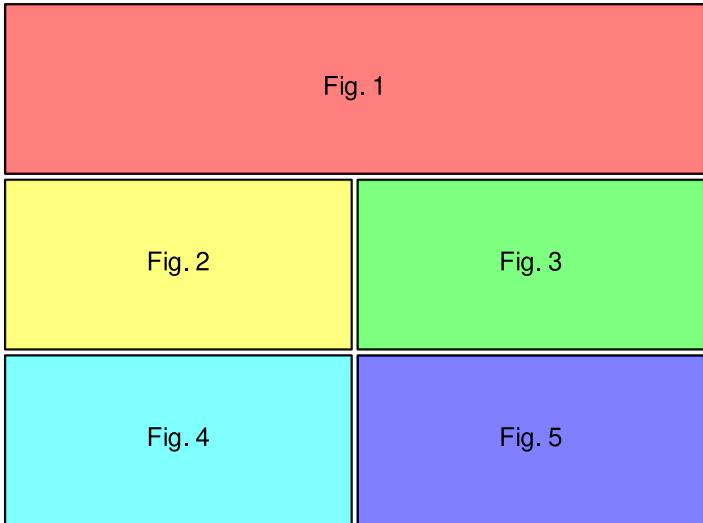
```
layout_mat <- matrix(c(6, 5,
                      2, 1,
                      3, 4),
                     nrow = 3,
                     byrow = T)

layout(layout_mat, c(1, 2), c(1, 1, 1))
```

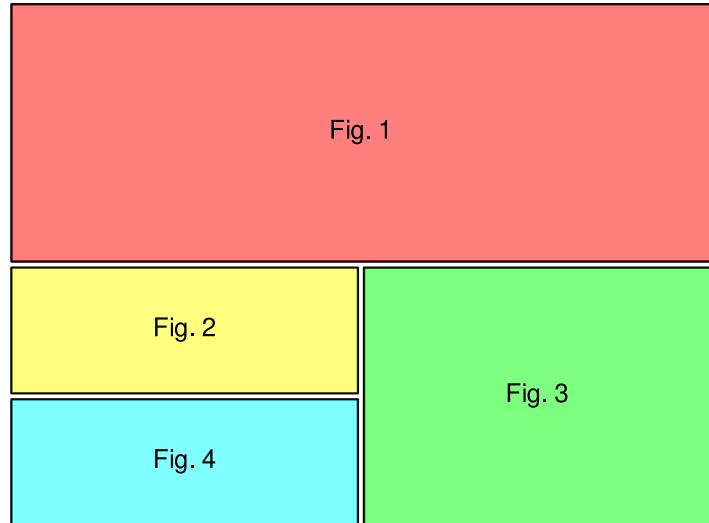


Complex layouts with `layout()`

```
layout_mat <- matrix(c(1, 1,  
                      2, 3,  
                      4, 5),  
                      nrow = 3,  
                      byrow = T)  
  
layout(layout_mat, c(1, 1), c(1, 1))
```

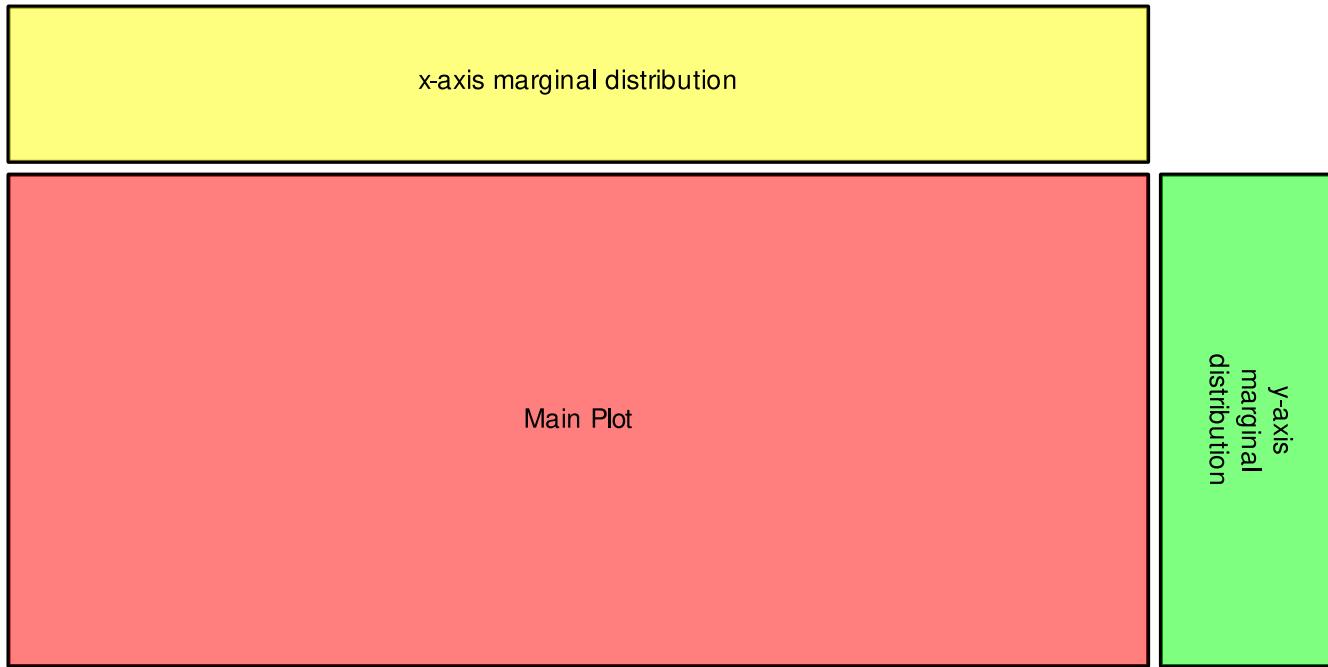


```
layout_mat <- matrix(c(1, 1,  
                      2, 3,  
                      4, 3),  
                      nrow = 3,  
                      byrow = T)  
  
layout(layout_mat, c(1, 1), c(2, 1))
```



Example: Adding Marginal Distributions

```
layout_mat <- matrix(c(2, 0,  
                      1, 3),  
                      nrow = 2 byrow = T)  
  
layout(mat = layout_mat,  
       widths = c(3, 0.5),  
       heights = c(1, 3))
```



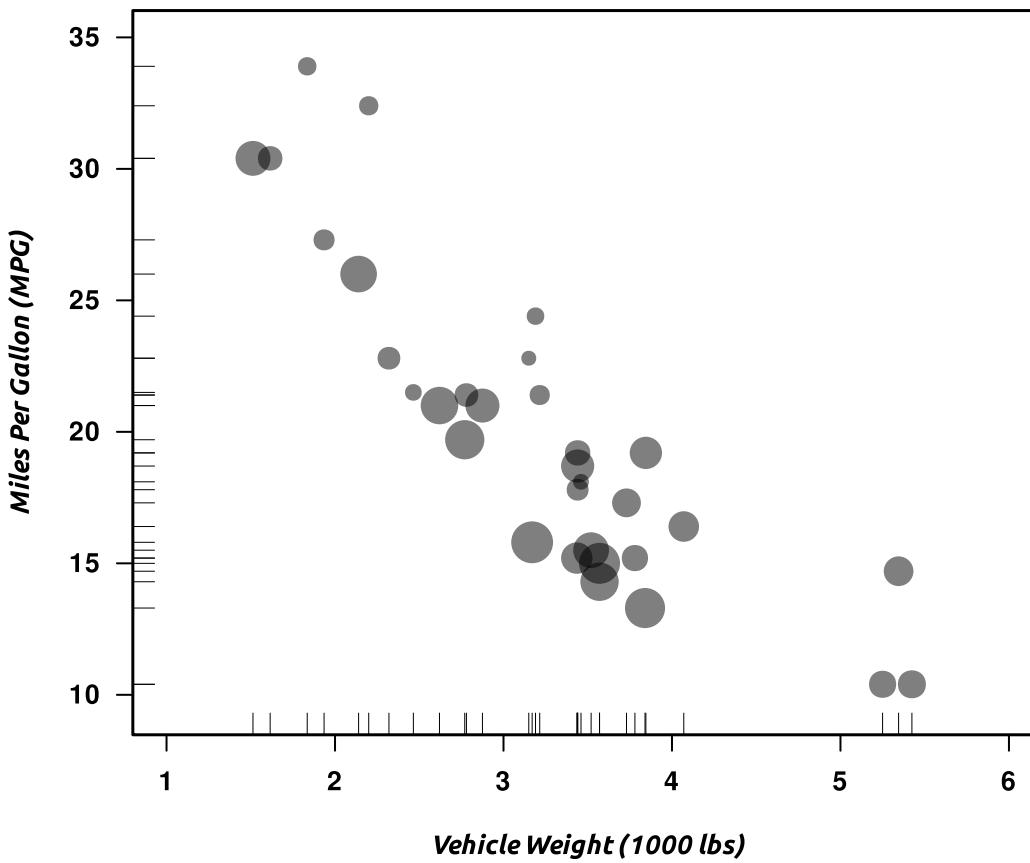
```
# Plot main scatterplot
par(mar = c(5, 4, 1, 1) + 0.1)
plot(...)

# Add marginal rugs to x and y axes
rug(mtcars$wt, side = 1)
rug(mtcars$mpg, side = 2)

# Get densities of `wt` and `mpg`
d_wt <- density(mtcars[order(mtcars$qsec), "wt"])
d_mpg <- density(mtcars[order(mtcars$qsec), "mpg"])

par(mar = c(0,3,1,.1))
plot(d_wt, axes = F, main = "", xlab= "", ylab = "", lwd = 2)

par(mar = c(4.25,0,1,1))
plot(d_mpg$y, d_mpg$x, type="l", axes = F, main = "",
     xlab= "", ylab = "", lwd = 2)
```



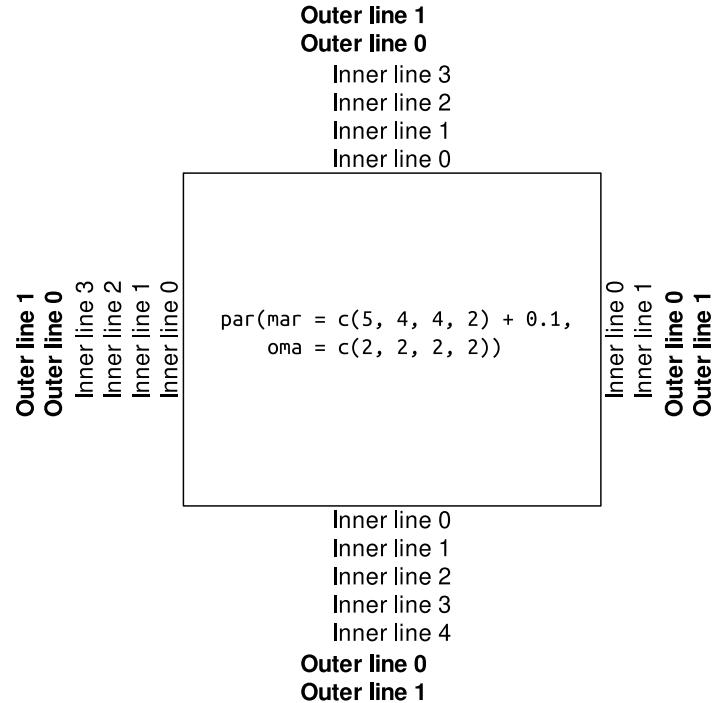
Plot Margins

Inner Margins

Inner margins refer to the margins on each axis

`par("mar")` is a numerical vector corresponding with `c(bottom, left, top, right)` that specifies the number of lines of margin on each side of the plot (default = `c(5, 4, 4, 2) + 0.1`)

`par("mai")` is similar to `mar`, except the margins are specified in inches (default = `c(1.02, 0.82, 0.82, 0.42)`)



Outer Margins

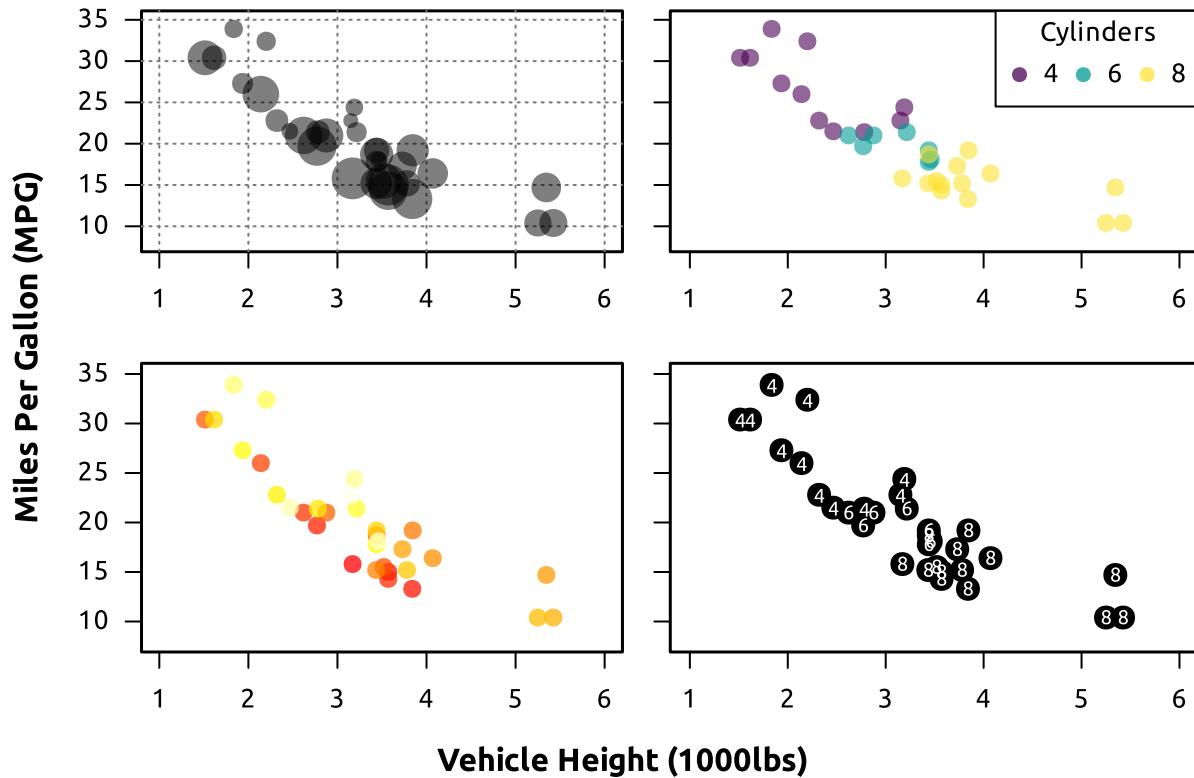
Outer margins correspond with the entire plotting region, not just the axes

`par("oma")` is a numerical vector corresponding with `c(bottom, left, top, right)` that specifies the number of lines of margin on each side of the plot (default is no margin)

`par("omi")` is similar to `oma`, except the margins are specified in inches

Example: Outer Margin Labels

```
par(..., oma = c(2, 2, 0, 4), family = "Ubuntu")  
mtext(text="Vehicle Height (1000lbs)", side = 1, line = 0, outer = TRUE, font = 2)  
mtext(text="Miles Per Gallon (MPG)", side = 2, line = 0, outer = TRUE, font = 2)
```



Other Types of Plots

Line Graph

The Theoph dataset in Base R has data from an experiment on the pharmacokinetics of theophylline (a medication for lung diseases like COPD). Let's plot the mean theophylline concentration (mg/L) over time (within-subjects) by dose administered (between-subjects).

Data need to be in long form for line graphs

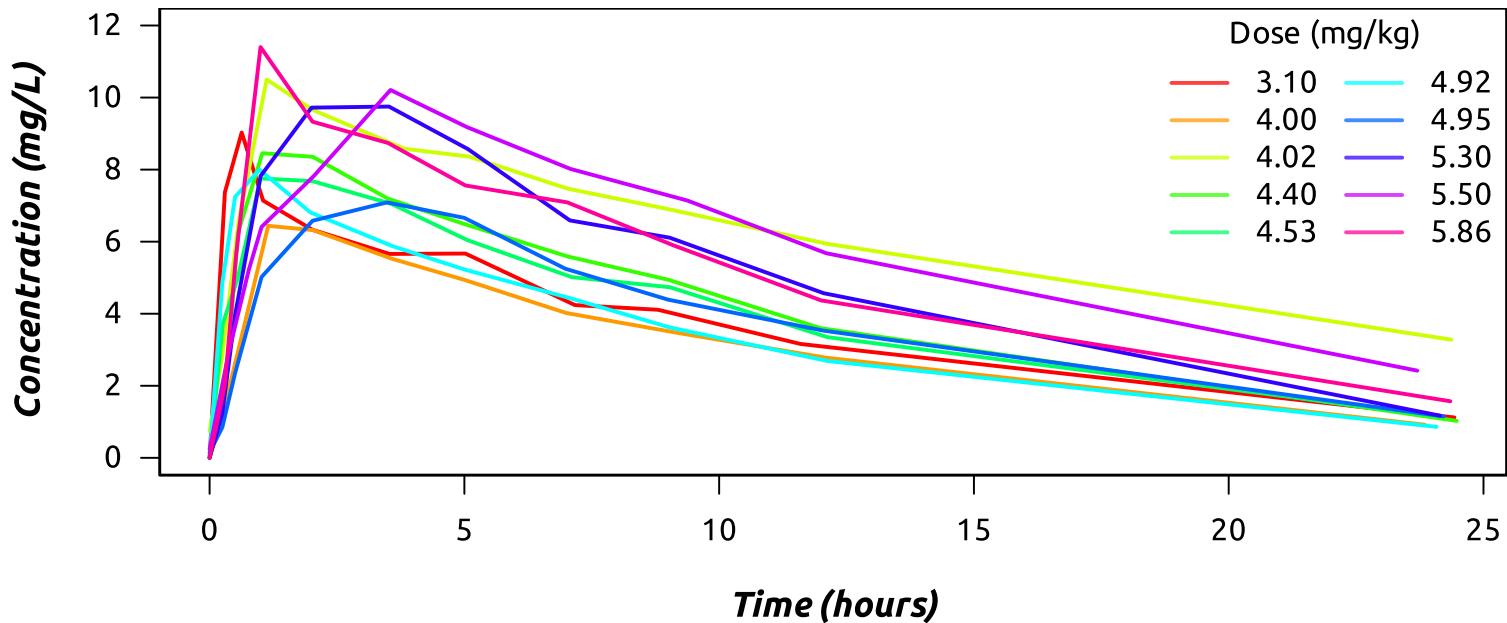
```

par(mar = c(5, 4, 0, 0))
par(family = "Ubuntu",
  font.lab = 4,
  cex.lab = 1.15)

# Plot empty plot with correct dimensions
plot(x = conc_data$Time, y = conc_data$conc, type = "n", axes = F,
      xlab = "Time (hours)", ylab = "Concentration (mg/L)",
      ylim = c(0, 12))
box(lwd = 1.5)
axis(side = 1, at = axTicks(1), labels = axTicks(1), lwd = 0, lwd.ticks = 1)
axis(side = 2, at = seq(0, 12, 2), labels = seq(0, 12, 2), lwd = 0, lwd.ticks = 1, las = 1)

# Plot line of each dose over time
# lines() is similar to points(type = "l")

```

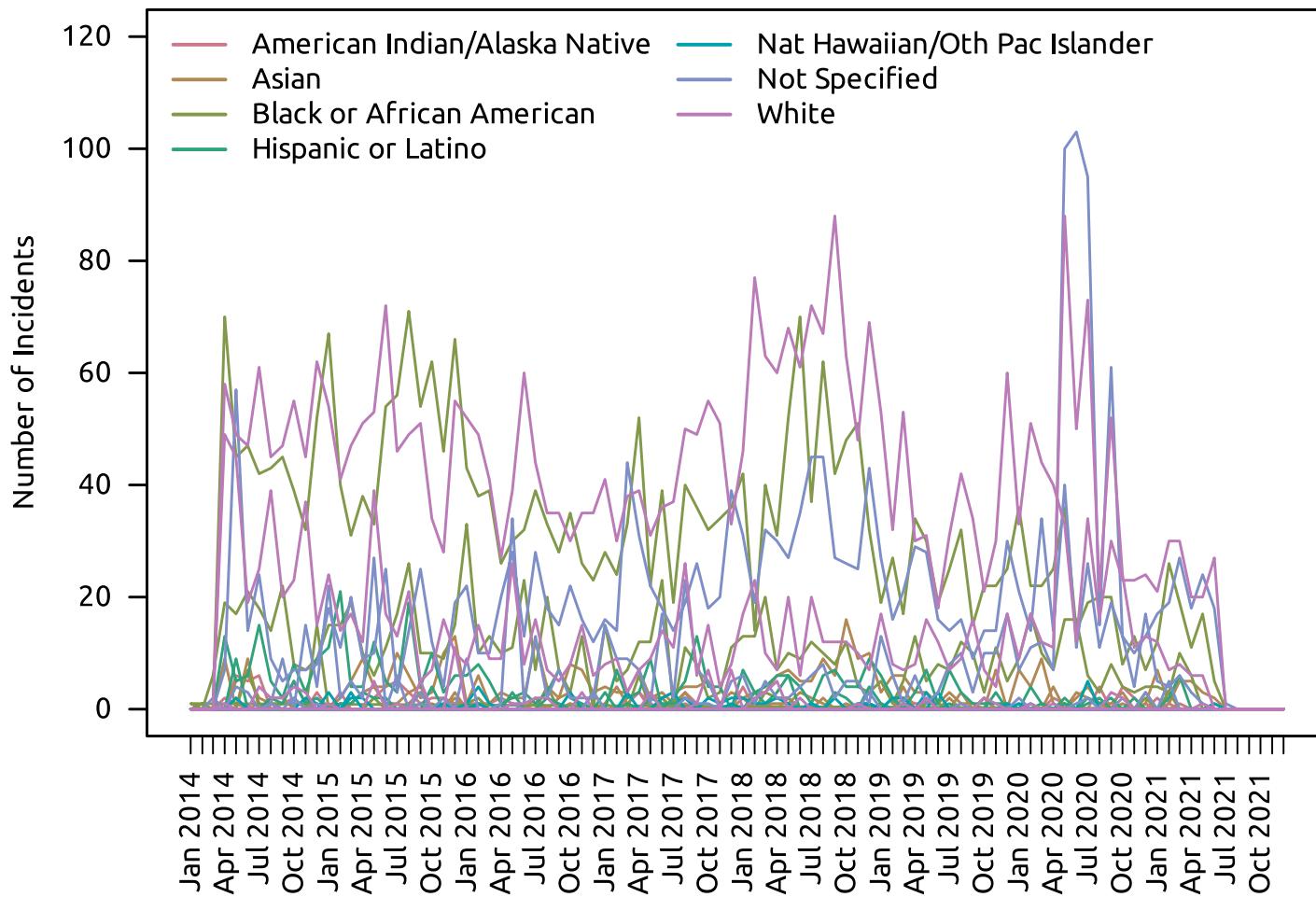


Plotting dates on the x-axis

When you have a `Date` column, R's `plot()` will automatically plot the dates properly on the x-axis. If you want to change the x-axis at all (labels, tick marks, other aesthetics), your best bet is to use the special `axis.Date()` function

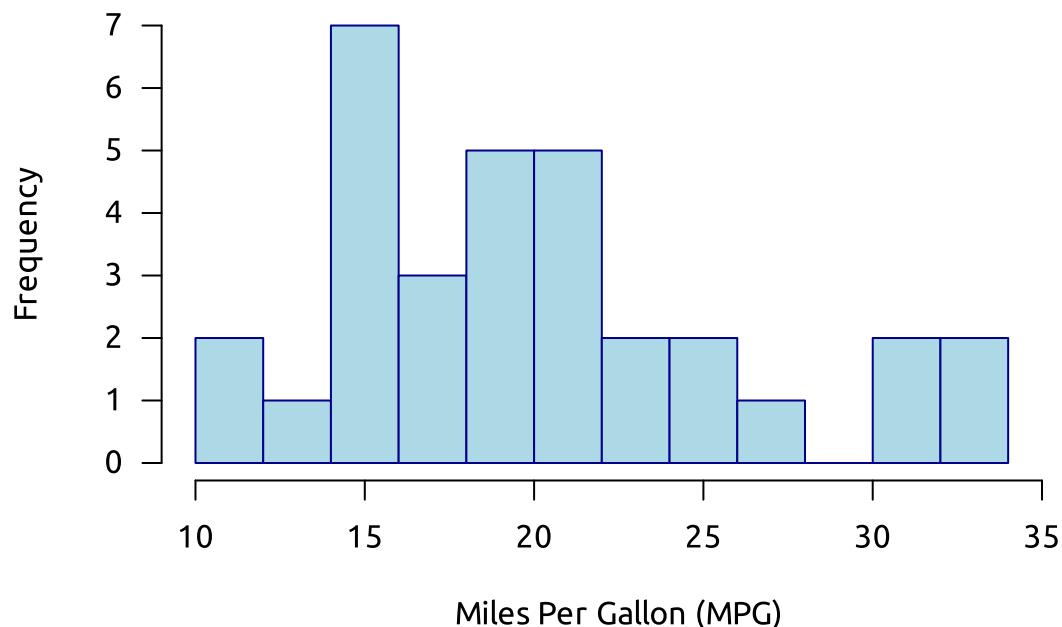
```
# Axis ticks at each month Jan - Dec
axis.Date(
  # x-axis
  side = 1,
  
  # Date object to create axis
  x = counts$Date,
  
  # Ticks from Jan to Dec each year in the data
  at = seq.Date(min(counts$Date),
                max(counts$Date),
                by = "month"),
  
  # Labels from Jan to Dec each year in the data
  # formatted to Year and abbreviated month name
  # (e.g., 2021 Aug)
  labels = format(seq.Date(min(counts$Date),
                           max(counts$Date),
                           by = "month"), "%b %Y"),
  
  # Rotate text -90 degrees
  las = 2
)
```

SPD Use-of-Force Incidents by Victim Race



Histogram

```
hist(x = mtcars$mpg, # data to plot  
      breaks = 15, # change default number of bars  
      xlim = c(10, 35), # change size of x-axis  
      main = "", # no main title  
      xlab = "Mile Per Gallon (MPG)", # x-axis title  
      las = 1, # y-axis ticks horizontal  
      border = "darkblue", # bar border color  
      col = "lightblue") # bar fill color
```



Frequency Polygon

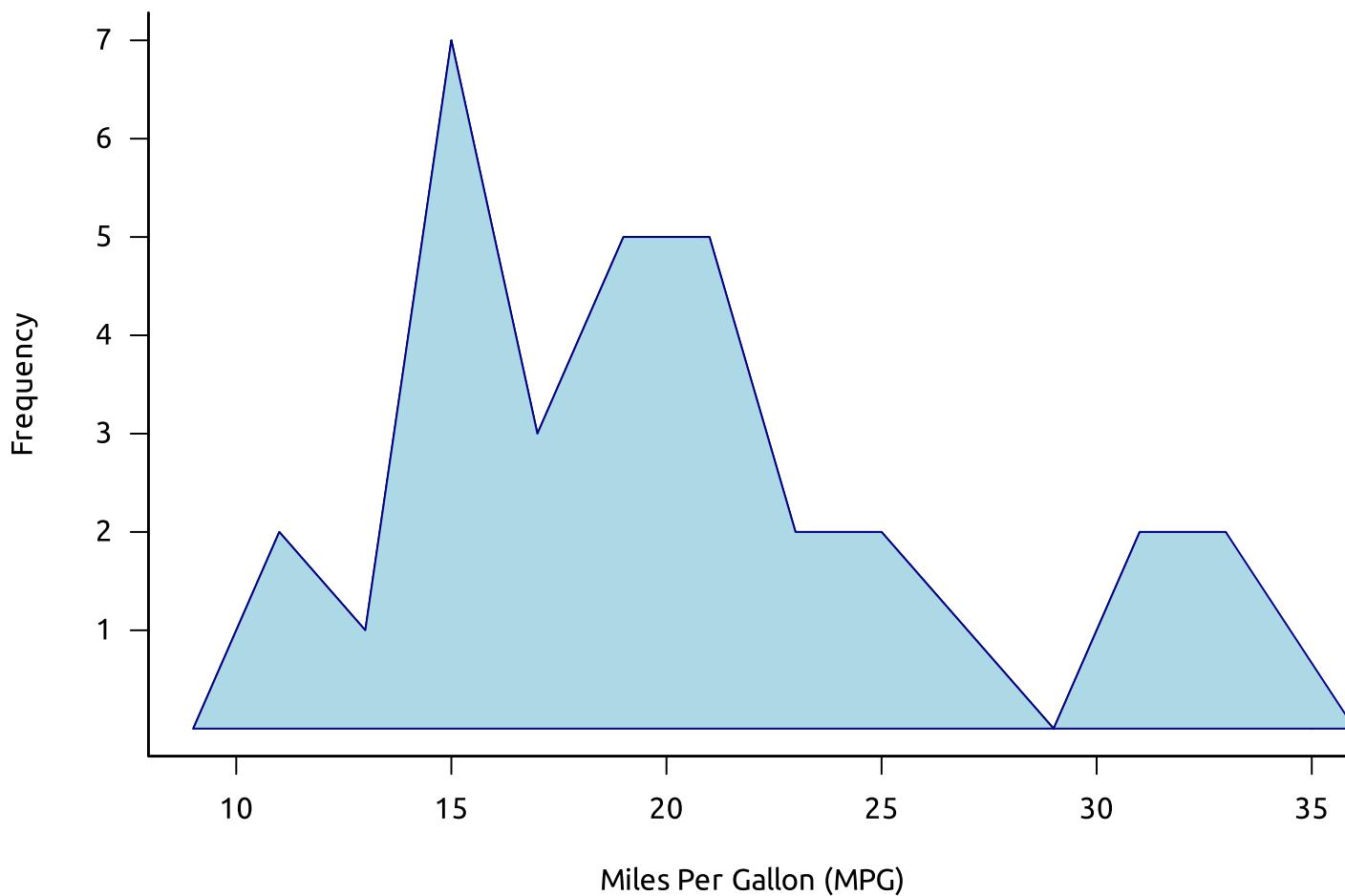
```
# Get histogram parameters without plotting it
p <- hist(x = mtcars$mpg, breaks = 15, plot = F)
str(p)

## List of 6
## $ breaks  : int [1:13] 10 12 14 16 18 20 22 24 26 28 ...
## $ counts   : int [1:12] 2 1 7 3 5 5 2 2 1 0 ...
## $ density  : num [1:12] 0.0312 0.0156 0.1094 0.0469 0.0781 ...
## $ mids     : num [1:12] 11 13 15 17 19 21 23 25 27 29 ...
## $ xname    : chr "mtcars$mpg"
## $ equidist: logi TRUE
## - attr(*, "class")= chr "histogram"

# Plot the midpoints and associated frequencies
plot(x = c(min(p$mids)-2, p$mids, max(p$mids)+2),
      y = c(0, p$counts, 0),
      type = "l",
      xlab = "Miles Per Gallon (MPG)",
      ylab = "Frequency",
      axes = F)

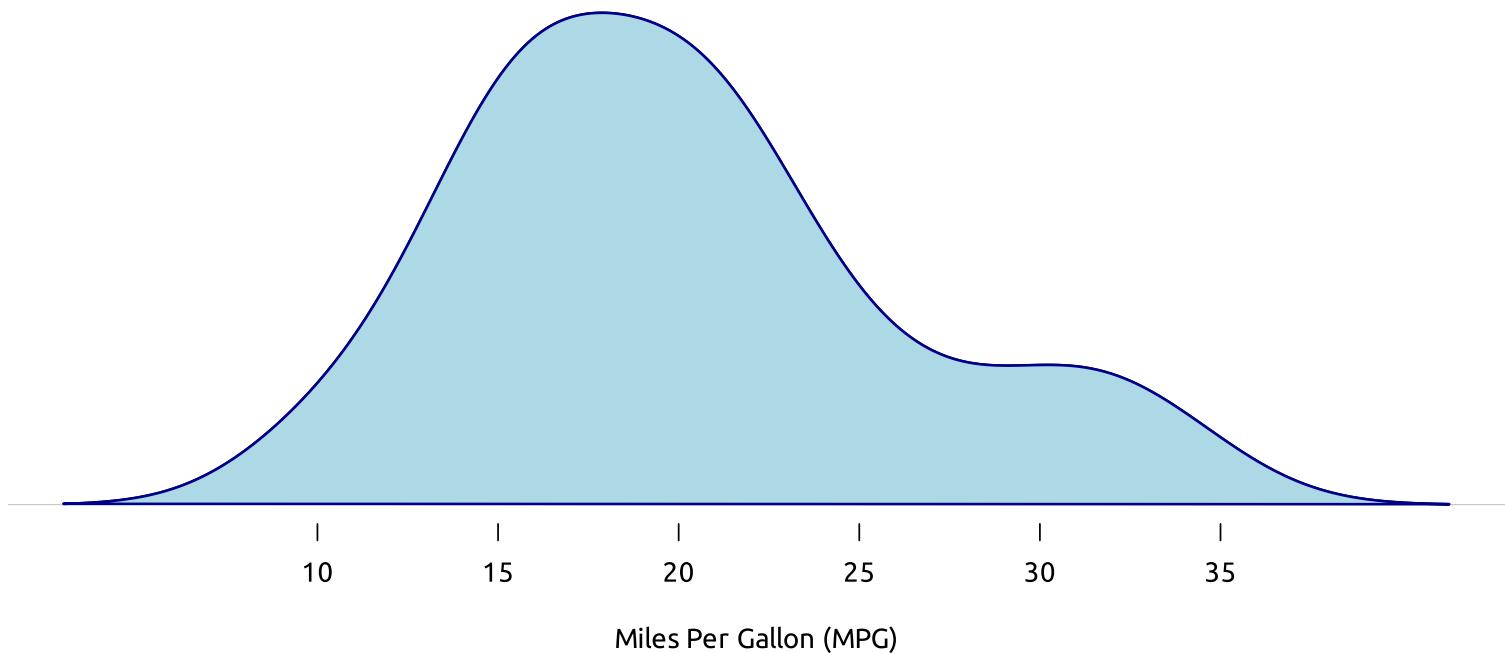
... # axes, box

# Create polygon to fill in area below curve
polygon(x = c(min(p$mids)-2, p$mids, max(p$breaks)+2), # create 0 min and max
         y = c(0, p$counts, 0),
         col = "lightblue",
         border = "darkblue")
```



Density plot

```
# Get density vals  
dens <- density(mtcars$mpg)  
  
# Plot density object (with other plotting args)  
plot(x = dens, ...)  
axis(side = 1, ...) # no side 2  
  
polygon(dens, col = "lightblue", border = "darkblue")
```



The Normal Curve

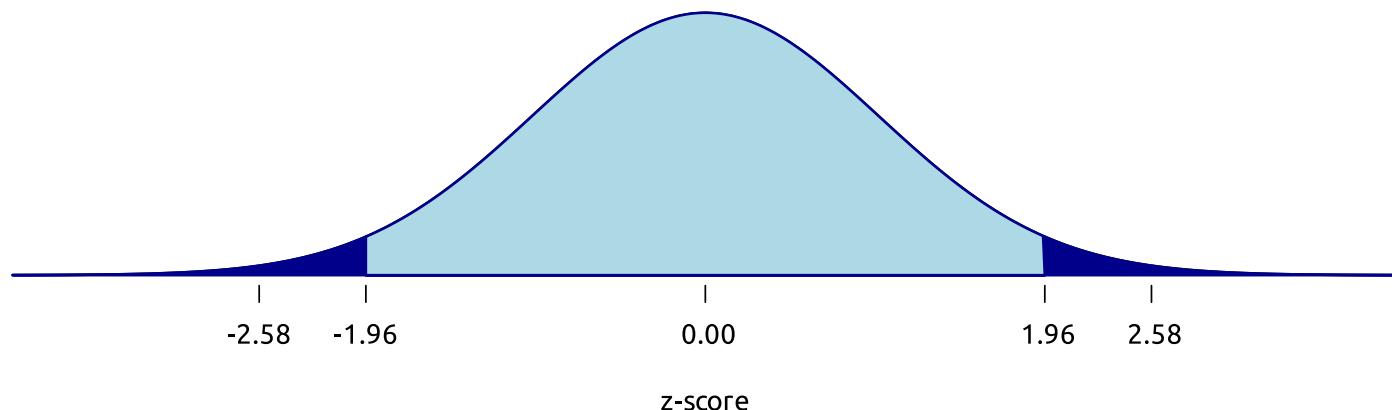
The `polygon()` function can be used to plot any area(s) you want. For example:

```
par(mar=c(4,0,0,0),
  family = "Ubuntu")

curve(expr = dnorm(x, mean = 0, sd = 1),
  xlim = c(-4, 4),
  xlab = "z-score",
  ylab = "",
  lwd = 1.5,
  axes = FALSE)

axis(side = 1, at = qnorm(c(0.005, 0.025, .50, 0.975, .995)),
  labels = format(qnorm(c(0.005, 0.025, .50, 0.975, .995)), nsmall = 2, digits = 3),
  lwd = 0, lwd.ticks = 1)

# Z scores to draw polygon (tails)
```



Barplots

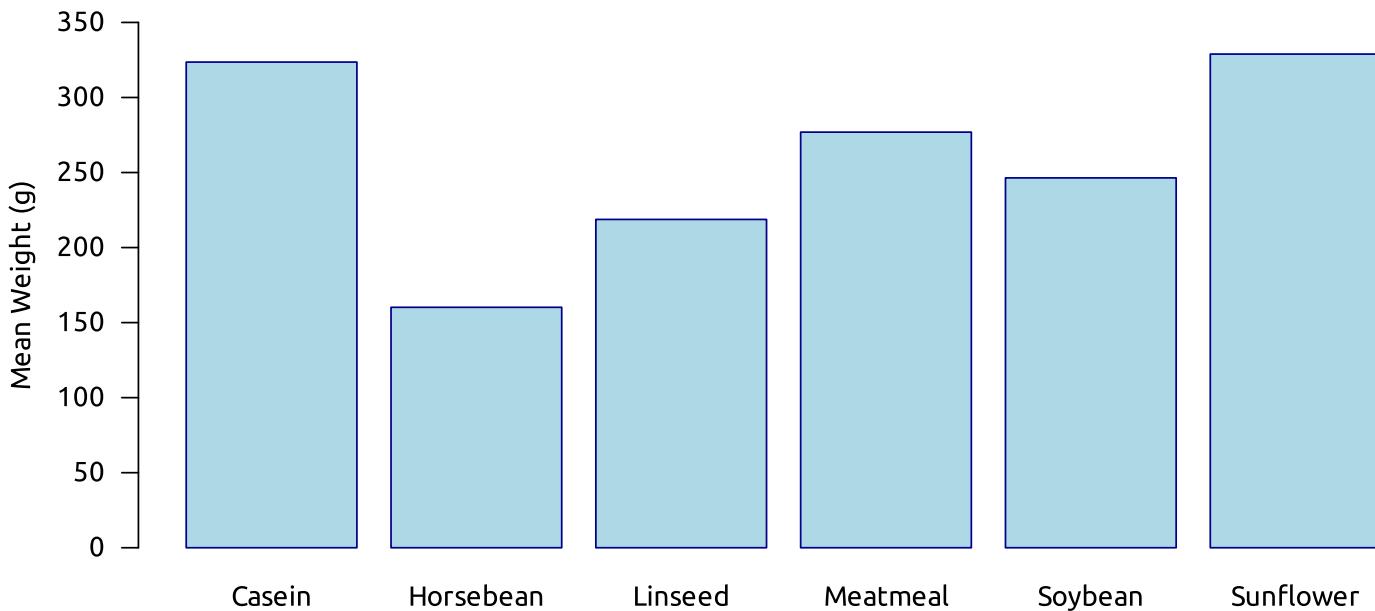
R's `chickwts` data comes from a between-subjects experiment on the effect of chicken feed implements on chicken growth rate at 6 weeks old:

```
## 'data.frame':    71 obs. of  2 variables:  
##   $ weight: num  179 160 136 227 217 168 108 124 143 140 ...  
##   $ feed   : Factor w/ 6 levels "casein","horsebean",...: 2 2 2 2 2 2 2 2 2 2 ...
```

```
# Calculate means and info for CI error bars (n, SD)  
chickwts_desc <- chickwts %>%  
  group_by(feed) %>%  
  summarize(n = n(),  
            mean_weight = mean(weight, na.rm = T),  
            sd_weight = sd(weight, na.rm = T))
```

feed	n	mean_weight	sd_weight
casein	12	323.5833	64.43384
horsebean	10	160.2000	38.62584
linseed	12	218.7500	52.23570
meatmeal	11	276.9091	64.90062
soybean	14	246.4286	54.12907
sunflower	12	328.9167	48.83638

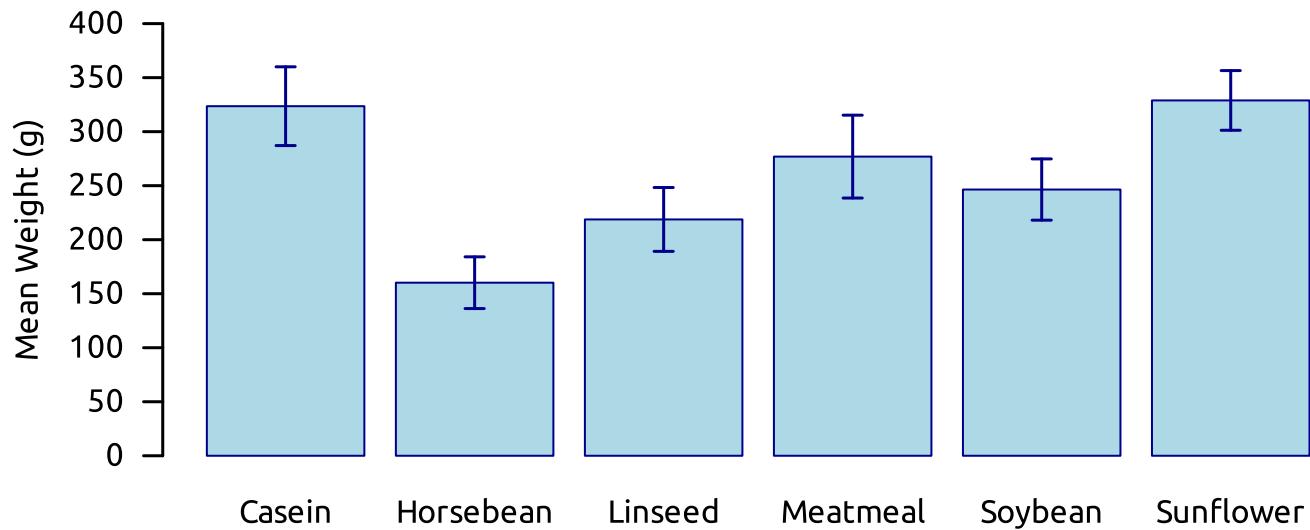
```
barplot(chickwts_desc$mean_weight,
        names.arg = stringr::str_to_title(chickwts_desc$feed),
        ylim = c(0, 350),
        las = 1,
        ylab = "Mean Weight (g)",
        border = "darkblue",
        col = "lightblue")
```



Error Bars

```
# Save barplot x values
bp <- barplot(...)

# with() lets you reference columns without subsetting each time
with(chickwts_desc,
    # draw arrows with flat lines on each head
    arrows(x0 = bp,
           x1 = bp,
           y0 = mean_weight - qnorm(.025, lower.tail = F) * (sd_weight / sqrt(n)),
           y1 = mean_weight + qnorm(.025, lower.tail = F) * (sd_weight / sqrt(n)),
           lwd = 1.5, angle = 90, code = 3, length = 0.05, col = "darkblue")
)
```



Pie Chart

The `pie()` function uses a table of counts (i.e. relative proportions) to build the initial pie chart

The `table()` and `tapply()` functions from base R are both very helpful for this

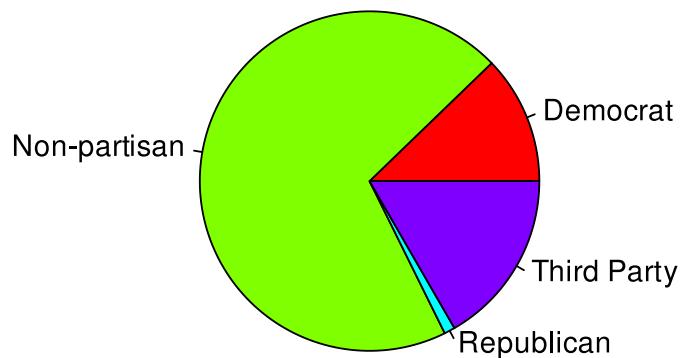
`group_by()` and `summarize(n = n())` from `dplyr` are also very helpful

From the King County 2016 elections data:

```
counts <- d %>%
  group_by(Party_Simplified) %>%
  summarize(n = n())
```

```
## # A tibble: 4 × 2
##   Party_Simplified     n
##   <chr>              <int>
## 1 Democrat            31931
## 2 Non-partisan        182983
## 3 Republican          2683
## 4 Third Party         43518
```

```
pie(x = counts$n,
  labels = counts$Party_Simplified,
  col = rainbow(4))
```

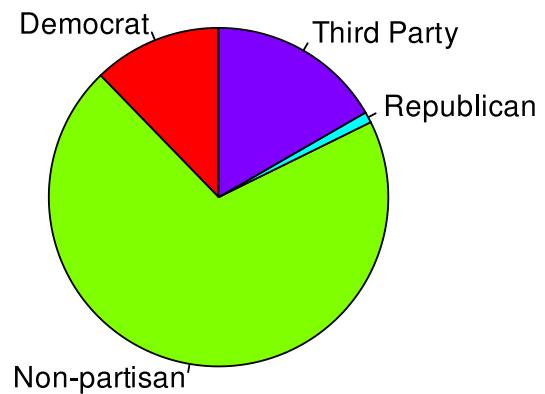
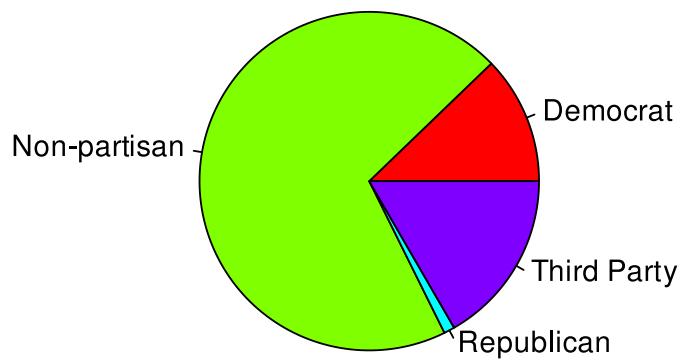


Use the `init.angle` argument to control the initial rotation of the pie chart

See `help("pie")` for more pie chart plotting options

```
pie(x = counts$n,  
     labels = counts$Party_Simplified,  
     col = rainbow(4))
```

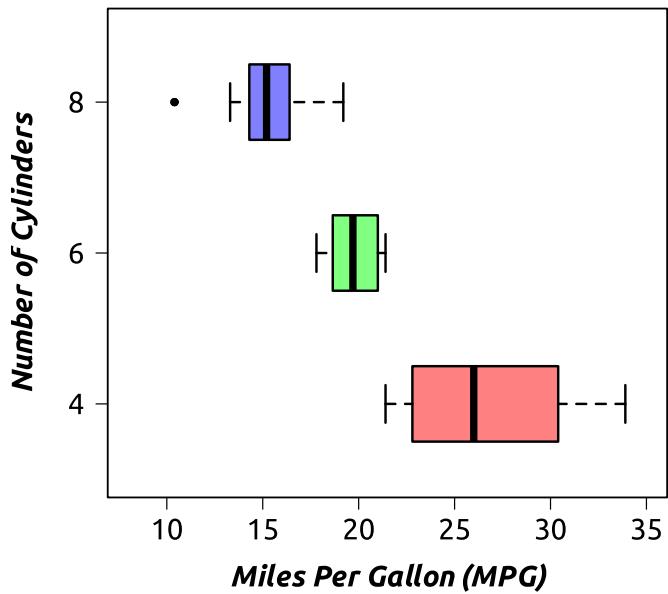
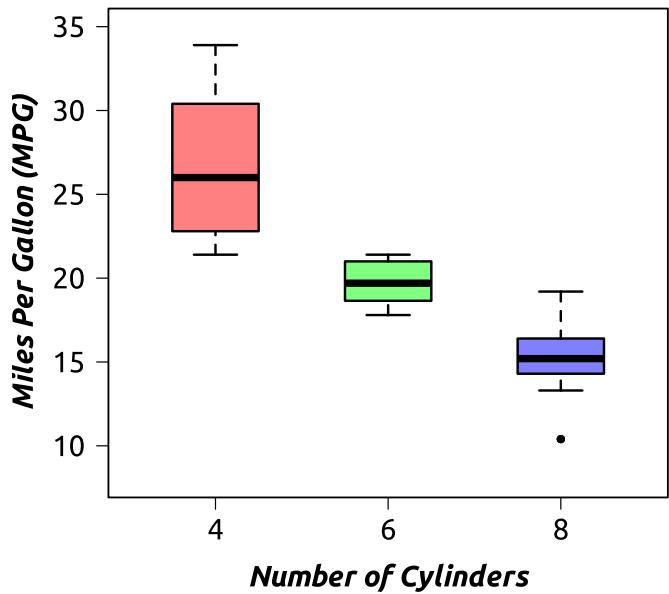
```
pie(x = counts$n,  
     labels = counts$Party_Simplified,  
     col = rainbow(4),  
     init.angle = 90)
```



Boxplot

```
# See `help("bxp")` for boxplot options  
boxplot(mpg ~ cyl, data = mtcars,  
        xlab="Number of Cylinders",  
        ylab="Miles Per Gallon",  
        pch = 20,  
        col = rainbow(3, .5),  
        boxwex = 0.5)
```

```
boxplot(mpg ~ cyl, data = mtcars,  
        ylab="Number of Cylinders",  
        xlab="Miles Per Gallon",  
        pch = 20,  
        col = rainbow(3, .5),  
        boxwex = 0.5,  
        horizontal = TRUE)
```



Violin Plot

You *can* make violin plots in base R, but it would require a lot of work and there's [a package](#) to make your life easier (while still using base R graphics) works just like the code for boxplot

```
vioplot(mpg ~ cyl, data = mtcars,
        xlab="Number of Cylinders",
        ylab="Miles Per Gallon (MPG)",
        pch = 20,
        ylim = c(8, 35),
        col = rainbow(3, .5),
        las = 1)
```

```
## Loading required package: sm

## Package 'sm', version 2.2-5.6: type help(
## Loading required package: zoo

## 
## Attaching package: 'zoo'

## The following objects are masked from 'pa
## 
##     as.Date, as.Date.numeric
```

```
vioplot(mpg ~ cyl, data = mtcars,
        xlab="Number of Cylinders",
        ylab="Miles Per Gallon (MPG)",
        pch = 20, las = 1,
        ylim = c(8, 35),
        col = rainbow(3, .5),
        horizontal = TRUE)
```

