Stat 405/705
Class 12

Statistical computing with R

Richard P. Waterman

Wharton

Table of contents I

Today's module

Topics to be covered in this module:

- Last time
- Graphics in R
  1. Low level graphics primitives
  2. Basic plotting
  3. High level presentation graphics
- Next time

Last time

- The R-ecosystem
- Extending R through packages

Low level graphics

- R lets you have complete control over graphics because it contains graphics primitives
- The primitives are the building blocks of any graphics
  1. Points
  2. Lines
  3. Polygons
  4. Curves

Graphics primitives

Begin with a blank canvas. The command for this is `plot`:

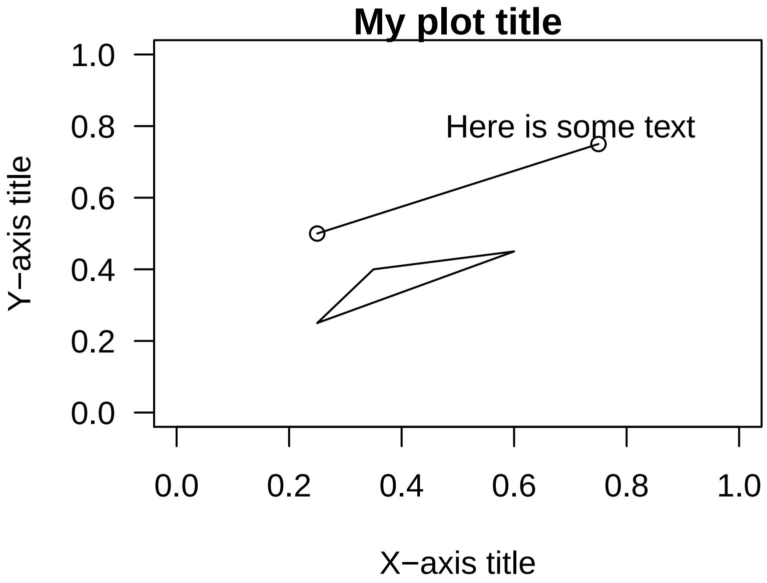
```r
plot(x = c(0,1), y = c(0,1), # this gives the unit square as a canvas
     main = "My plot title", # add a title
     xlab = "X-axis title",  # add an x-axis label
     ylab = "Y-axis title",  # add a y-axis label
     type = "n")             # don't actually plot anything yet
```

Adding objects to the canvas

The most basic features are points, lines, polygons and text.
All these elements need to be positioned with coordinates. They can also take additional graphical parameters to change the way they are rendered. For example, to draw points as squares instead of circles, dashed lines rather than solid, and so on.

```r
points(x = c(0.25,0.75), y = c(0.5,0.75)) # Add points to the graph
lines(x=c(0.25,0.75), y = c(0.5,0.75))    # Join them up with a line
polygon(x = c(0.25,0.6,0.35), y = c(0.25, 0.45, 0.4)) # A polygon
text(x = c(0.7),y = c(0.8), labels = c("Here is some text")) # Add text
```

Graphics primitives

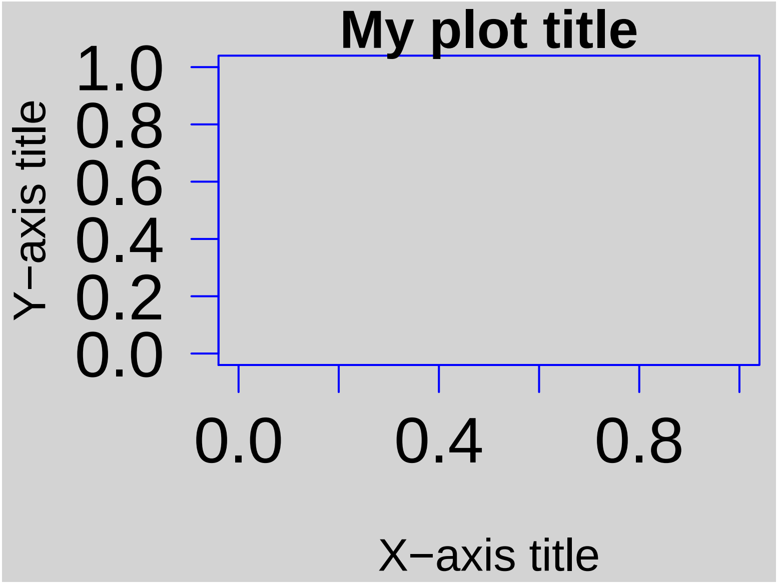


The par command

The par command gives you almost complete control over the canvas with the use of graphical parameters. We most often use it for controlling margins and axes

```r
par(bg = "lightgray", # Set the background color
    cex = 1.4,        # Larger sized symbols
    cex.axis = 1.4,   # Change the axis annotation magnification
    fg = "blue")      # Set the foreground color

plot(x = c(0,1), y = c(0,1), # this gives the unit square as a canvas
     main = "My plot title", # add a title
     xlab = "X-axis title",  # add an x-axis label
     ylab = "Y-axis title",  # add a y-axis label
     type = "n")             # don't actually plot anything yet
```

The `par` command

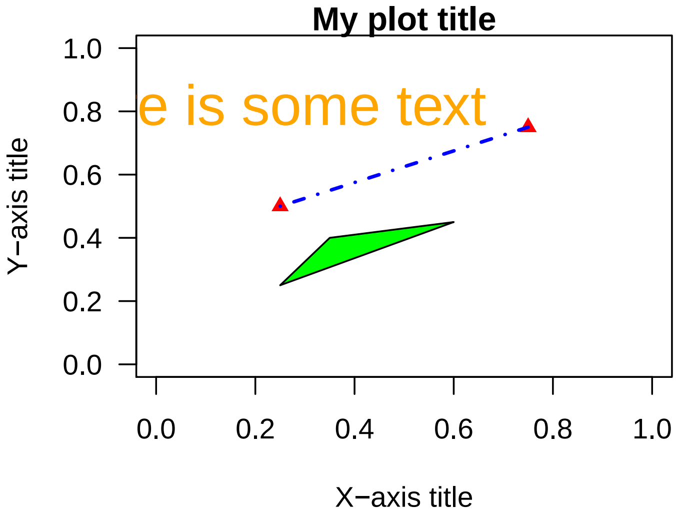


Add graphics attributes directly to the primitives

That was ugly so we should reset the par() options.
The best way to do that is save them at the beginning of the session, so we can always reset the original values.

```
old.par <- par(no.readonly=TRUE) # Save the current set of graphics paramet

par(old.par) # Reset the old graphics parameters

points(x = c(0.25,0.75), y = c(0.5,0.75),
       pch=17,col="red") # use red triangles for points
lines(x=c(0.25,0.75), y = c(0.5,0.75),
       lwd=2,lty=4,col="blue") # Change the line type
polygon(x = c(0.25,0.6,0.35), y = c(0.25, 0.45, 0.4),
        col="green") # Fill the polygon in green
text(x = c(0.7),y = c(0.8), labels = c("Here is some text"),
     col="orange",cex=2,pos=2) # Position to the left of the coordinates
```

Add graphics attributes directly to the primitives

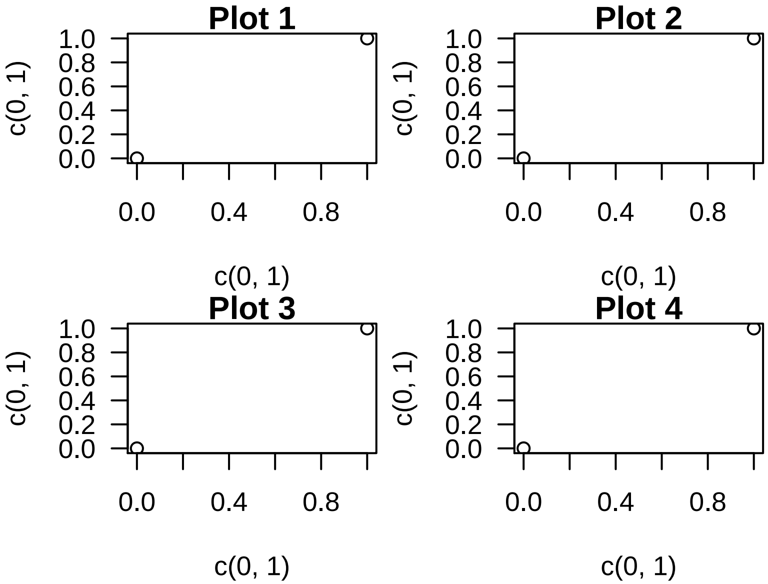


Multiple plots on the same page

The command to do this is the option to `par`, `mfrow`.

```r
par(mfrow=c(2,2)) # Set up a two-by-two grid of plots

plot(x = c(0,1), y = c(0,1), # this gives the unit square as a canvas
     main = "Plot 1")
plot(x = c(0,1), y = c(0,1), # this gives the unit square as a canvas
     main = "Plot 2")
plot(x = c(0,1), y = c(0,1), # this gives the unit square as a canvas
     main = "Plot 3")
plot(x = c(0,1), y = c(0,1), # this gives the unit square as a canvas
     main = "Plot 4")
```

Multiple plots on the same page

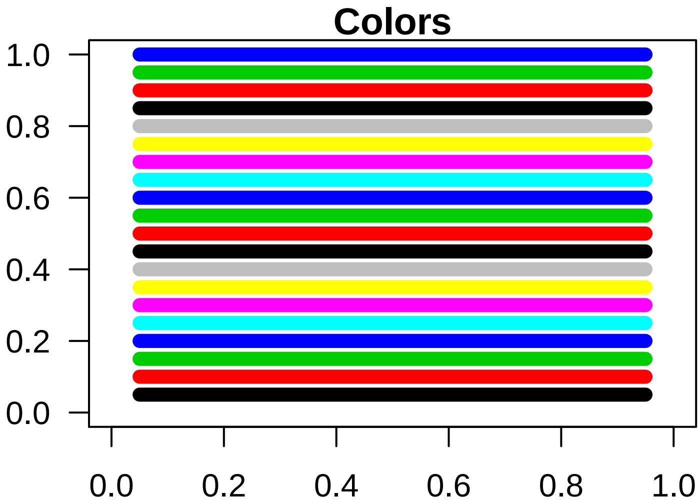


Colors

There are built in colors with expected names, and a default numbered set of eights colors. You can also make your own colors.

```
plot(x = c(0,1), y = c(0,1), # this gives the unit square as a canvas
     xlab="",ylab="",main = "Colors",type="n")
for(i in 1 :20){
  lines(x = c(0.05,0.95),
        y = c(i/20,i/20),
                col=i, # The numbered colors
                lwd=7) # line width
}
```

Colors by number



Named colors

You can see the named colors with the command `colors()`:

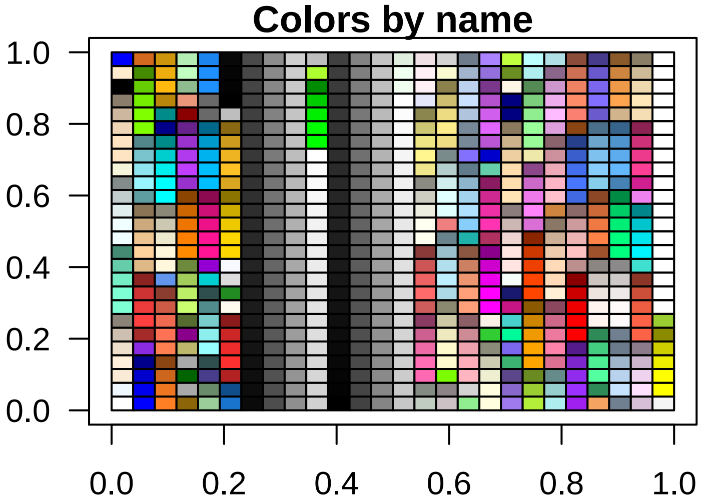
```
length(colors())  # There are plenty of colors

## [1] 657

colors()[1:30] #Here are the names of the first 30

##  [1] "white"          "aliceblue"      "antiquewhite"
##  [4] "antiquewhite1"  "antiquewhite2"  "antiquewhite3"
##  [7] "antiquewhite4"  "aquamarine"     "aquamarine1"
## [10] "aquamarine2"    "aquamarine3"    "aquamarine4"
## [13] "azure"          "azure1"         "azure2"
## [16] "azure3"         "azure4"         "beige"
## [19] "bisque"         "bisque1"        "bisque2"
## [22] "bisque3"        "bisque4"        "black"
## [25] "blanchedalmond" "blue"           "blue1"
## [28] "blue2"          "blue3"          "blue4"
```

Colors

```r
#### View all the colors available by name:
plot(x = c(0,1), y = c(0,1), # this gives the unit square as a canvas
     xlab="",ylab="",main = "Colors by name",type="n")
counter <- 1 # Keep track of the latest color
for(i in 0:25){ # Loop over the x-axis
  for(j in 0:25){ # Loop over the y-axis
    polygon(  # We will fill a square with each color in turn
                x = c(i/26,(i+1)/26, (i+1)/26, i/26),
            y = c(j/26, j/26,(j+1)/26, (j+1)/26),
            col=colors()[counter] # Specificy the color by name
            )
    counter <- counter + 1
  }
}
```

Colors

Colors can be accessed by name, for example "red", "lightgreen" etc.

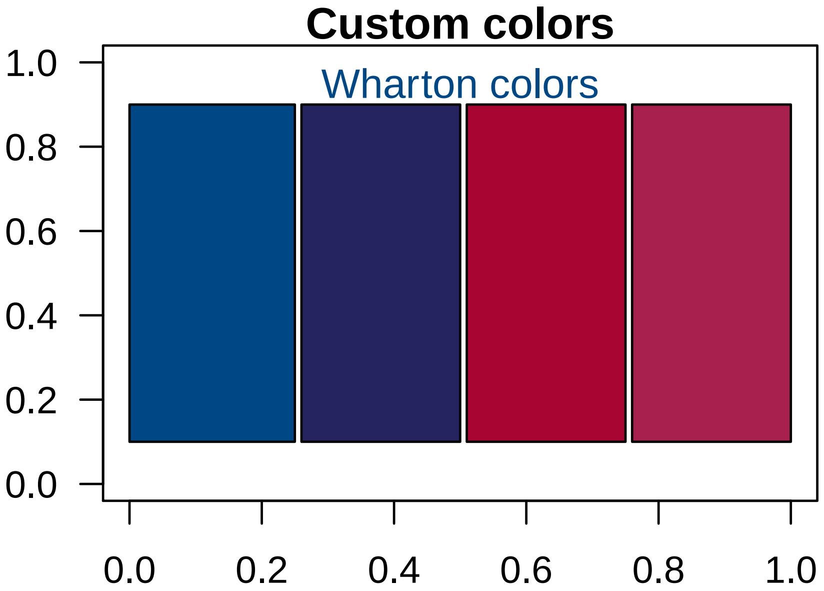


Making your own colors

You can create your own colors using the `rgb` command (there is a transparency argument too):

```
wb1 <- rgb(red=    0/255, green = 71/255, blue = 133/255,
           names="wharton-blue-one")
wb2 <- rgb(red =  38/255, green = 36/255, blue =  96/255,
           names="wharton-blue-two")
wr1 <- rgb(red = 169/255, green =  5/255, blue =  51/255,
           names="wharton-red-one")
wr2 <- rgb(red = 168/255, green = 32/255, blue =  78/255,
           names="wharton-red-two")

plot(x = c(0,1), y = c(0,1), # this gives the unit square as a canvas
     xlab="",ylab="", main = "Custom colors",type="n")
polygon(x = c(0,0.25,0.25,0), y = c(0.1,0.1,0.9,0.9), col=wb1)
polygon(x = c(0.26,0.5,0.5,0.26), y = c(0.1,0.1,0.9,0.9), col=wb2)
polygon(x = c(0.51,0.75,0.75,0.51), y = c(0.1,0.1,0.9,0.9), col=wr1)
polygon(x = c(0.76,1.0,1.0,0.76), y = c(0.1,0.1,0.9,0.9), col=wr2)
text(x=0.5,0.95,"Wharton colors",col=wb1,cex=1.1)
```

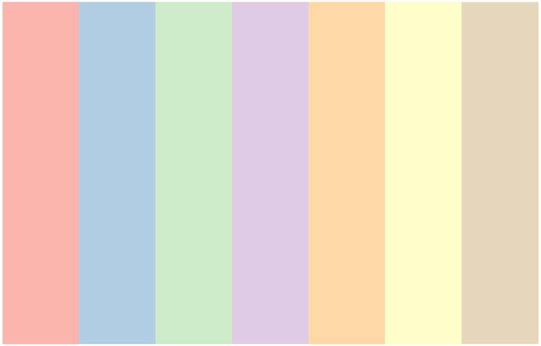
Making your own colors



Predefined palletes

Another option is to use a predefined color palette. The package `RColorBrewer` facilitates this:

```
library(RColorBrewer) # Load the color brewer library
display.brewer.pal(n=7, name="Pastel1") # Display a palette with 7 colors
my.palette <- brewer.pal(7,name="Pastel1") # Save it for future use
```



Basic plotting commands

The common plots include:

- Histograms
- Scatterplots
- Bar charts
- Pie charts

Get a dataset ready for plotting

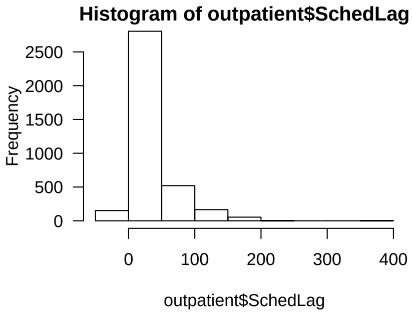
Preparing the outpatient dataset:

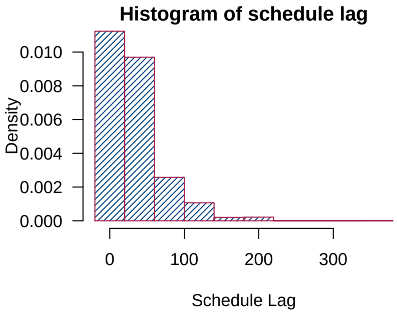
```
outpatient <- read.csv( # Read in the outpatient data
"C:\\Users\\richardw\\Dropbox (Penn)\\Teaching\\705s2019\\Data\\Outpatient.

outpatient$SchedDate <- strptime(outpatient$SchedDate, format="%m/%d/%Y")
outpatient$ApptDate <- strptime(outpatient$ApptDate, format="%m/%d/%Y")

# The difference between the two dates (measured in seconds)
# This will be a new column in the data frame
outpatient$SchedLag <- (outpatient$ApptDate - outpatient$SchedDate)
# Turn it into days
outpatient$SchedLag <- as.numeric(outpatient$SchedLag)/(60 * 60 * 24)
```

Histogram

```r
hist(x = outpatient$SchedLag) # A basic histogram

hist(x = outpatient$SchedLag, # A more custom histogram
     breaks=seq(-20,400,40),  # Choose break points
     freq=FALSE,              # Plot probabilities
     density=24,              # Control shading
     col=wb1,                 # Specify color
     border=wr2,              # Specify border
     xlab="Schedule Lag",     # Create x-axis label
     main = "Histogram of schedule lag" # Plot title
)
```

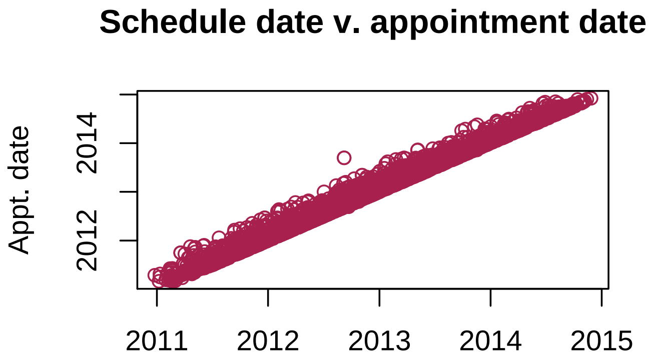
Histogram


Histogram of outpatient$SchedLag
Frequency
2500
2000
1500
1000
500
0
0
100
200
300
400
outpatient$SchedLag


Histogram of schedule lag
Density
0.010
0.008
0.006
0.004
0.002
0.000
0
100
200
300
Schedule Lag

Scatterplots

```
plot(x = outpatient$SchedDate, y = outpatient$ApptDate,
     main="Schedule date v. appointment date",
         xlab="Schedule date",
         ylab = "Appt. date",
     col=wr2)
```

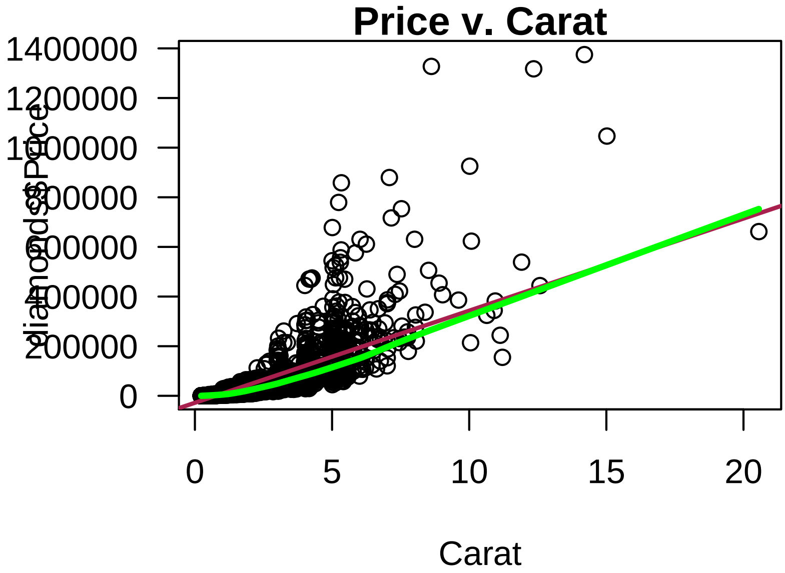


Adding lines and smooths to the plot

The commands `abline` and `lowess` will add lines and a smooth to a scatterplot.

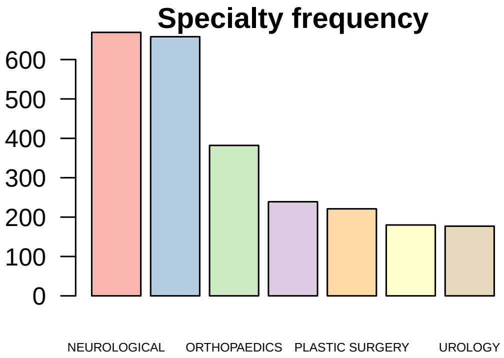
```
diamonds <- read.csv( # Read in the diamonds data
"C:\\Users\\richardw\\Dropbox (Penn)\\Teaching\\705s2019\\Data\\Diamonds200

plot(diamonds$Carat,diamonds$Price,main="Price v. Carat") #Plot the data

abline(lm(Price ~ Carat, data = diamonds),col=wr2,lwd=2) # Add the LS line
lines(lowess(diamonds$Carat,diamonds$Price), # A scatterplot smoother
      col=wb2,
      lwd=3) # Add the smooth to the plot
```

Adding lines and smooths to the plot



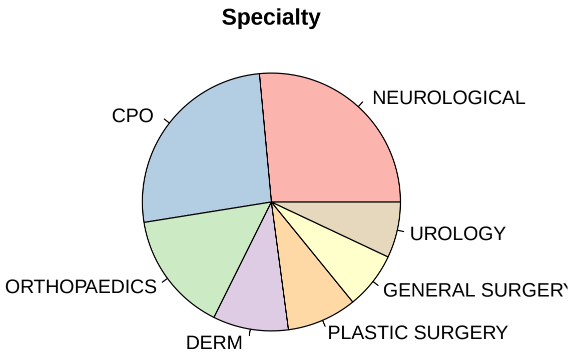
Bar plots

```
barplot(sort(table(outpatient$Dept),decreasing=TRUE)[1:7], # The top 7 spec
        cex.names=0.5,  # Make the names smaller so they fit
        col=my.palette, # Specify the color palette
        main="Specialty frequency") # Add a title
```



Bar plots

```
pie(sort(table(outpatient$Dept),decreasing=TRUE)[1:7], # The top 7 specialt
        col=my.palette, # Specify the color palette
        main="Specialty") # Add a title
```



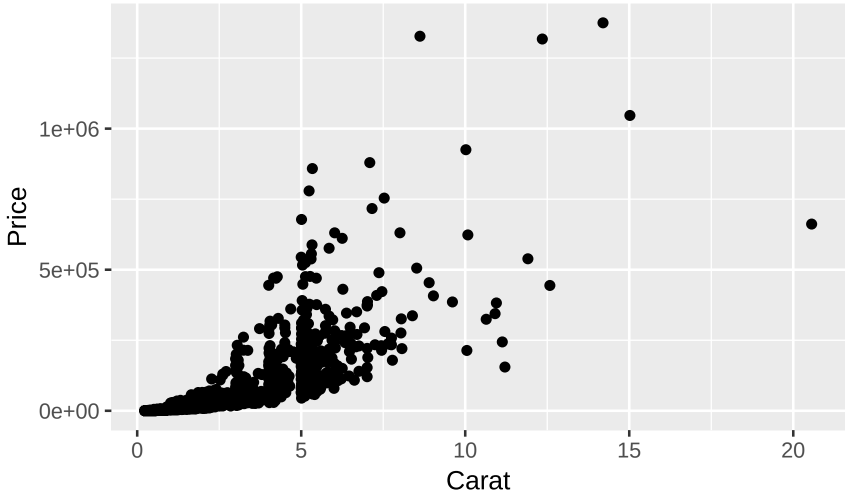
High level graphics: `ggplot2`

- The `ggplot2` package is is the state of the art in plotting. It often needs the `plyr` package to initially shape data.
- There is a quick plotting command `qplot` and a more fine tuned one `ggplot2`.
- A useful cheat sheet: https://www.rstudio.com/wp-content/uploads/2015/03/ggplot2-cheatsheet.pdf

```
# Read in the library
library(ggplot2)

##
## Attaching package:  'ggplot2'
## The following object is masked _by_ '.GlobalEnv':
##
##     diamonds
```

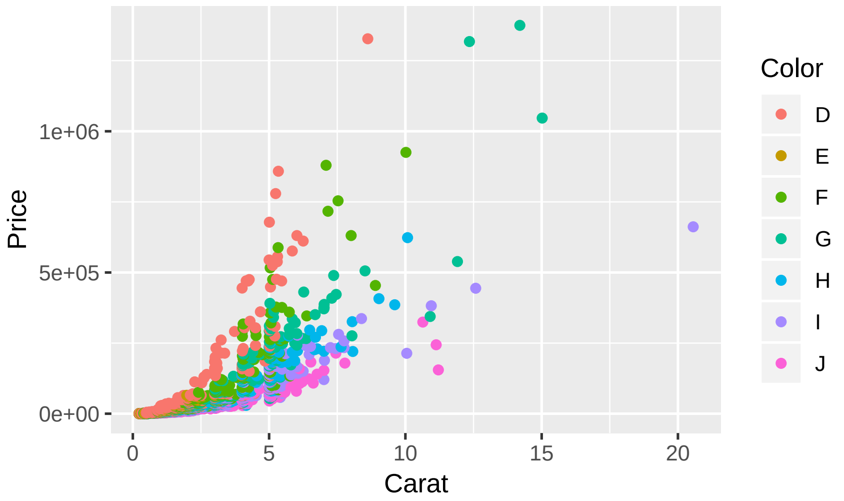
A basic plot

```r
qplot(x = Carat, y = Price, data = diamonds,  geom="point")
```



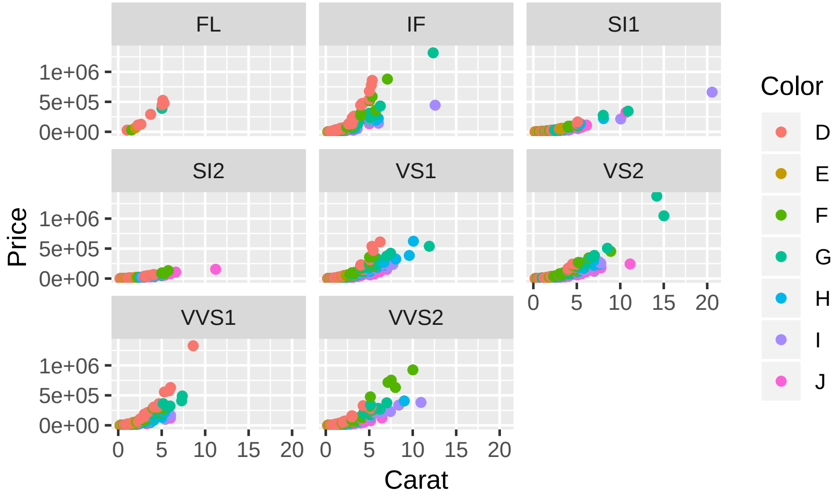
Add some aesthetics

```r
qplot(x = Carat, y = Price, color = Color, data = diamonds,  geom="point")
```



Add a facet (conditioning variable)

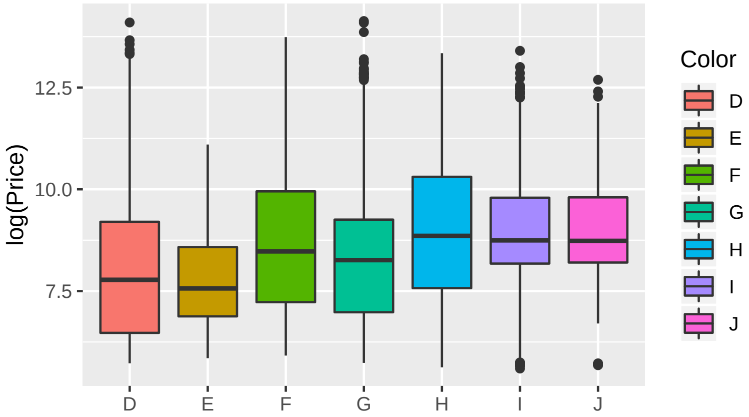
```
qplot(x = Carat, y = Price, color = Color,
      facets = ~ Clarity, data = diamonds,  geom="point")
```



You get more flexibility with ggplot

The ggplot command doesn't show anything until you add a "layer".

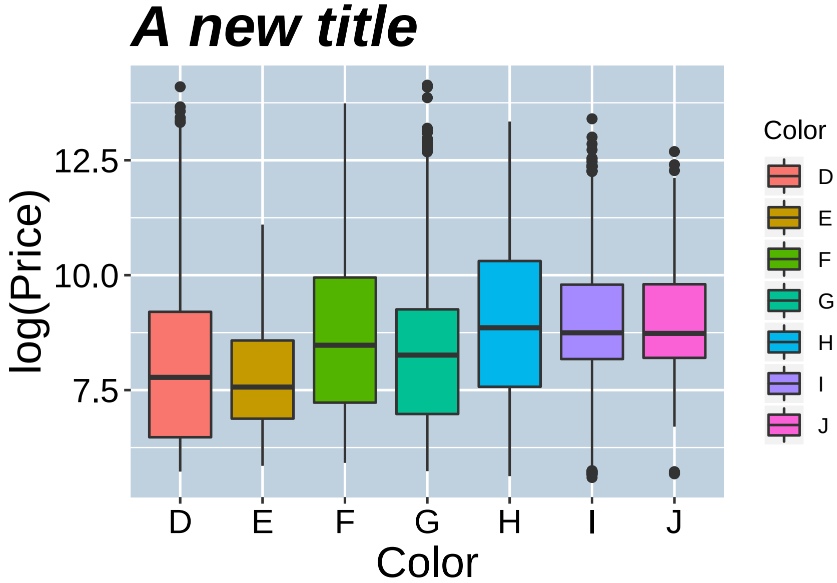
```
# This defines the features in the plot (but shows nothing)
my.plot <- ggplot(data = diamonds,
                  aes(x = Color, y = log(Price), fill = Color))
my.plot + geom_boxplot() # Now add a layer
```



Adding more layers

```
# We will add elements to the plot.
my.plot +
  geom_boxplot() + # Now add a layer and some thematic elements
  ggtitle("A new title") +
  theme(plot.title=element_text(face="bold.italic", size="24", color="black
  theme(axis.title=element_text(size="18", color="black")) +
  theme(axis.text=element_text(size="14", color="black")) +
  theme(panel.background = element_rect(fill = "#BFD0DF"))
```

Adding more layers



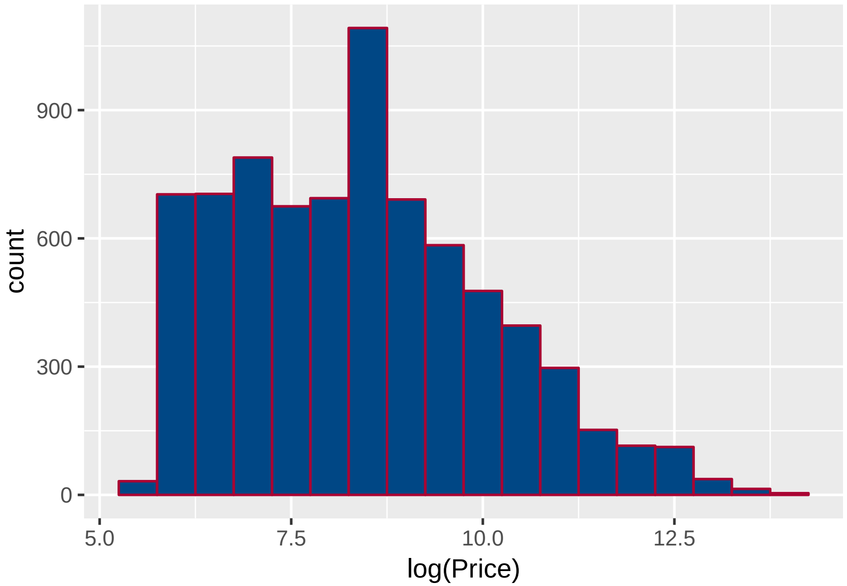
Plotting continuous variables

```r
# A basic histogram
ggplot(data = diamonds, aes(x = log(Price))) +
       geom_histogram(binwidth=0.5,col=wr1,fill=wb1)

#Overlaying a transparent density
ggplot(data = diamonds, aes(x = log(Price))) +
      geom_histogram(binwidth=0.5,col=wr1,fill=wb1,aes(y=..density..)) +
      geom_density(alpha=.2, fill=wb2)

# Stacked histogram
ggplot(data = diamonds, aes(x = log(Price),fill=Color)) +
      geom_histogram(binwidth=0.5)
```

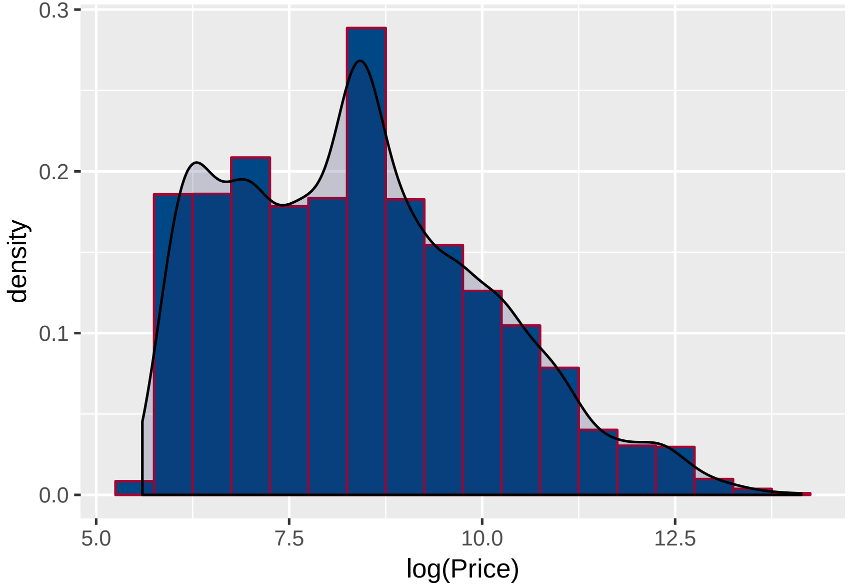
Plotting continuous variables

```
# Price densities for each color
ggplot(data = diamonds, aes(x = log(Price),fill=Color)) +
       geom_density(alpha=.3)

# Add Cut as a facet
 ggplot(data = diamonds, aes(x = log(Price),fill=Color)) +
       geom_density(alpha=.3) +
           facet_grid(~ Cut)


# Add Cut and Polish as a facet
 ggplot(data = diamonds, aes(x = log(Price),fill=Color)) +
       geom_density(alpha=.3) +
           facet_grid(Polish ~ Cut)
```

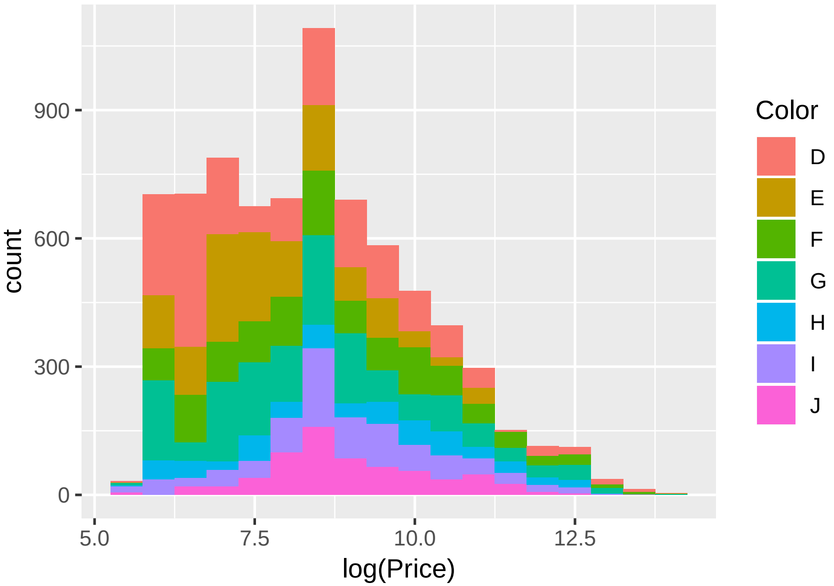
Plotting continuous variables



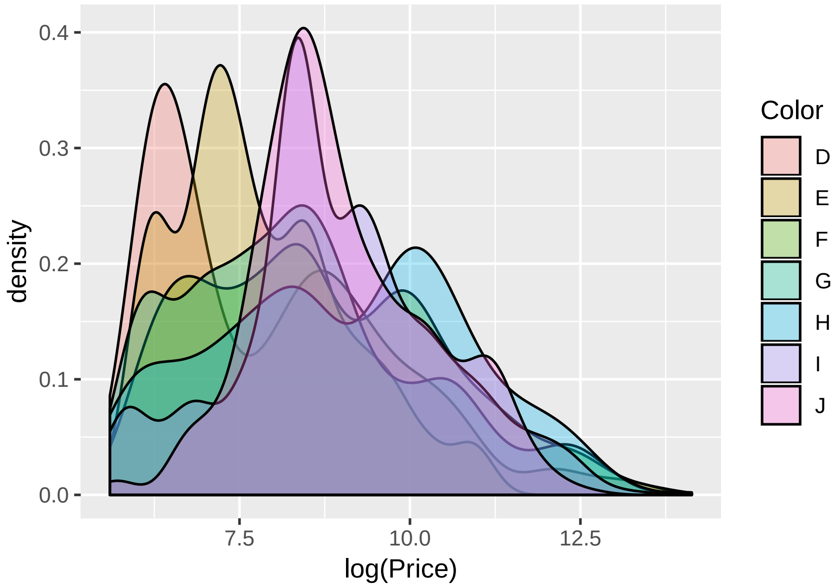
Plotting continuous variables



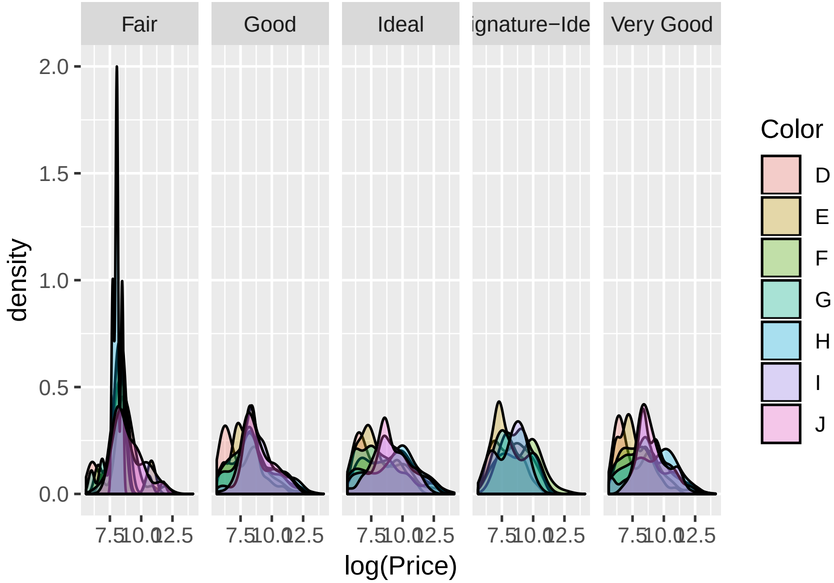
Plotting continuous variables



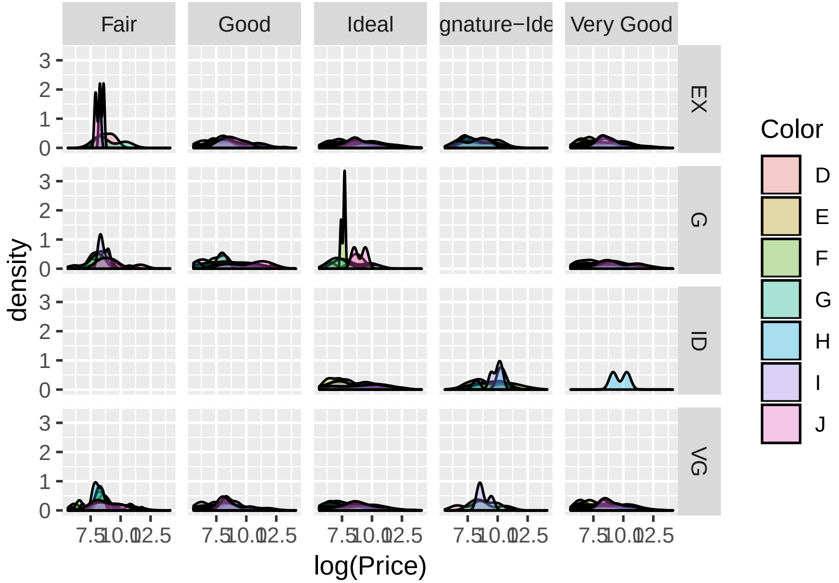
Price densities for each color



Add Cut as a facet



Add Cut and Polish as a facet



Saving a plot from within R

```
my.graph <- ggplot(data = diamonds, aes(x = log(Price),fill=Color)) +
       geom_density(alpha=.3) +
           facet_grid(~ Cut)

# Open a graphics device to write to
png(filename="C:\\Users\\richardw\\Dropbox (Penn)\\Teaching\\705s2019\\Note
             width=720,
             height=480)
my.graph
dev.off()       # Turn the graphics device off

## pdf
##   2
```

Module summary

Topics covered today include:

- Low-level graphics
- Basic graphics
- High-level presentation graphics

Next time

- Automatic report generation

Today's function list

Do you know what each of these functions does?

```
abline
barplot
brewer.pal
display.brewer.pal
colors
ggplot
hist
lines
lowess
par
pie
plot
points
polygon
qplot
rgb
text
```