MATH 3050 – Predictive Analytics

UNC CHARLOTTE

Topic 1: The R Programming Language – Part 4

- Exploratory Analysis
  - One-Dimension Plots
  - Multiple Dimension Plot
  - Other Plotting Considerations
- Interactions
- Interaction Plots

1

Objectives of this Lesson:

By the end of this lesson you should be able to:

- Exploit features of one-dimension summaries
- Exploit features of multiple-dimension summaries
- Exploit common plot parameters
- Distinguish the three plotting systems in R
- Print to various graphical devices
- Manipulate color in graphical plots
- Create smooth scatter plots

2

One-Dimension Plots

For one dimensional summaries, there are number of options in R.

1. **Five-number summary**: This gives the minimum, 25th percentile, median, 75th percentile, maximum of the data and is a quick check on the distribution of the data. The `fivenum()` function can be used and it takes a vector of numbers. It produces the same results as `summary()`, less the mean.
2. **Boxplots**: Boxplots are a visual representation of the five-number summary plus a bit more information. In particular, boxplots commonly plot outliers that go beyond the bulk of the data. This is implemented via the `boxplot()` function
3. **Barplot**: Barplots are useful for visualizing categorical data, with the number of entries for each category being proportional to the height of the bar. Think "pie chart" but actually useful. The barplot can be made with the `barplot()` function.
4. **Histograms**: Histograms show the complete empirical distribution of the data, beyond the five data points shown by the boxplots. Here, you can easily check skewwness of the data, symmetry, multi-modality, and other features. The `hist()` function makes a histogram and a handy function to go with it sometimes is the `rug()` function.
5. **Density plot**: The `density()` function computes a non-parametric estimate of the distribution of a variables.

Application to Pollution Data

The U.S. Environmental Protection Agency (EPA) sets national air quality standards for outdoor air pollution. One of the national ambient air quality standards in the U.S. concerns the long-term average level of fine particle pollution, also referred to as PM2.5.

The standard says that the "annual mean, averaged over 3 years" cannot exceed 12 micrograms per cubic meter. Data on daily PM2.5 are available from the U.S. EPA web site, or specifically, the EPA Air Quality System web site.

pm25:	Grams of particulate matter 2.5 micrometers in diameter.
fips:	Numbers which uniquely identify geographic areas.
region:	East or West
longitude:	The angular distance of a place east or west of the meridian at Greenwich, England.
latitude:	The angular distance of a place north or south of the earth's equator.

```
#Dataset = avgpm25.csv
pollution<-avgpm25
attach(pollution)
str(pollution)
head(pollution)
```

```
'data.frame':   576 obs. of  5 variables:
 $ pm25     : num  9.77 9.99 10.69 11.34 12.12 ...
 $ fips     : int  1003 1027 1033 1049 1055 1069 1073
 $ region   : Factor w/ 2 levels "east","west": 1 1 1
 $ longitude: num  -87.7 -85.8 -87.7 -85.8 -86 ...
 $ latitude : num  30.6 33.3 34.7 34.5 34 ...
```

```
       pm25 fips region longitude latitude
1  9.771185 1003   east -87.74826 30.59278
2  9.993817 1027   east -85.84286 33.26581
3 10.688618 1033   east -87.72596 34.73148
4 11.337424 1049   east -85.79892 34.45913
5 12.119764 1055   east -86.03212 34.01860
6 10.827805 1069   east -85.35039 31.18973
```

The Class of a Variable

The "class" of an object defines the data type of the object.

For example:

```
bubba <- c(1,2,3)
bubba
[1] 1 2 3

class(bubba)
[1] "numeric"
```

R object classes:

1. Numeric
2. Character
3. Factor
4. Logical

5

The Class of a Variable

Numeric Class Examples:

```
class(12)
class(c(1, 1.5, 2))
class(1:5)

#Coerce numbers as integer
as.integer(c(1, 1.5, 2))
```

Character Class Examples:

```
class("United States")
as.numeric("United States")

#A numeric and a character become character
class(c(1, "test"))
a <- 1:4 class(a)
class(a) <- "character"
class(a)
```

Factor Class Examples:

```
factor(1:3)
factor(c(1, 2, 1, 2, 1, 2))
levels(factor(1:3))
factor(c("a", "b", "b", "c"))
```

Logical Class Examples:

```
k <- 4 > 3
class(k)

isTRUE(k)
!isTRUE(k)

one <- c(TRUE, FALSE, TRUE, FALSE)
two <- c(FALSE, TRUE, TRUE, TRUE)
one & two
```


6

The Class of a Variable

The List Class

The list class is a very flexible class, and thus, very useful. You can put anything inside a list, such as numbers:

```
list1 <- list(3, 2)
```

Or other lists constructed with c():

```
list2 <- list(c(1, 2), c(3, 4))
```

You can also put objects of different classes in the same list:

```
list3 <- list(3, c(1, 2), "lists are amazing!")
```

And of course create list of lists:

```
my_lists <- list(list1, list2, list3)
str(my_lists)
```


The Class of a Variable

The List Class

You can also create named lists:

```
list4 <- list("a" = 2, "b" = 8, "c" = "this is a named list")
```

And you can access the elements in two ways:

```
list4[[1]]
```

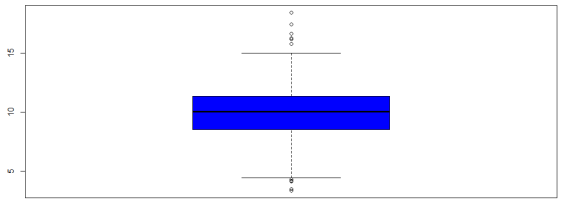
Or, for named lists:

```
list4$c
```

One Dimension Summaries

```
#Change fips to Character Class for Session
class(fips)<-"character"
class(fips)

fivenum(pollution$pm25)
summary(pollution$pm25)
boxplot(pollution$pm25, col = "blue")
```

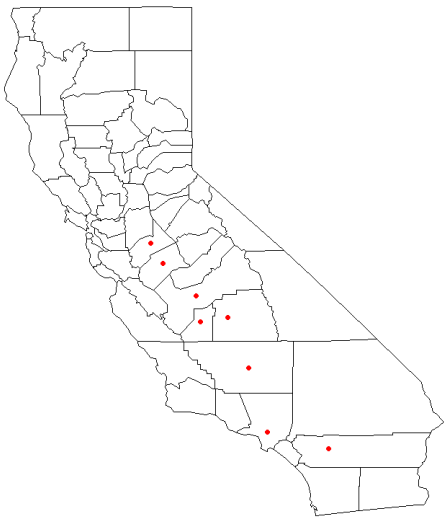
```
> fivenum(pollution$pm25)
[1]  3.382626  8.547590 10.046697 11.356829 18.440731
> summary(pollution$pm25)
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
  3.383   8.549  10.047   9.836  11.356  18.441
```



9

One Dimension Summaries

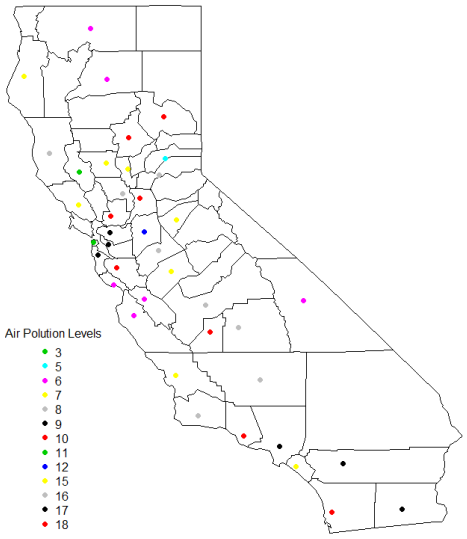
```
#Using Maps
install.packages("maps")
library(maps)
library(dplyr)
map("county", "california")
with(filter(pollution, pm25 > 15), points(longitude, latitude,
pch=16, col="red"))
```



10

One Dimension Summaries

```
#Using Maps
CA<-filter(pollution, State.Name=="California")
CA

map("county", "california")
with(CA, points(longitude, latitude, pch=16,col=CA$pm25_rounded))
legend("bottomleft", horiz=FALSE, xjust=0, title ="Air Pollution
Levels",legend = paste(unique(CA$pm25_rounded)),
     col = unique(CA$pm25_rounded), pch = 19, bty = "n")
```



Reference for legend parameters:
https://stat.ethz.ch/R-manual/R-devel/library/graphics/html/legend.html

One Dimension Summaries

Homework

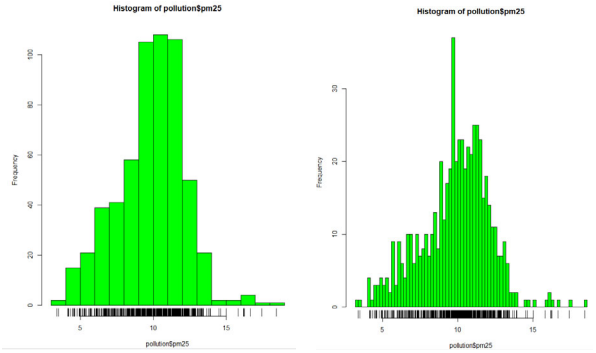
Create a pollution map for your state. If your state is California, pick your next favorite state.

Make sure you play around with the legend parameters, so your legend does not overlap your map. You can find more information on legend parameters at the reference below.

Reference for legend parameters:
https://stat.ethz.ch/R-manual/R-devel/library/graphics/html/legend.html

One Dimension Summaries

```
#Histogtrams
hist(pollution$pm25, col = "green")
rug(pollution$pm25)

#Breaks
hist(pollution$pm25, col = "green", breaks = 100)
rug(pollution$pm25)
```



One Dimension Summaries

Density estimation for continuous variables

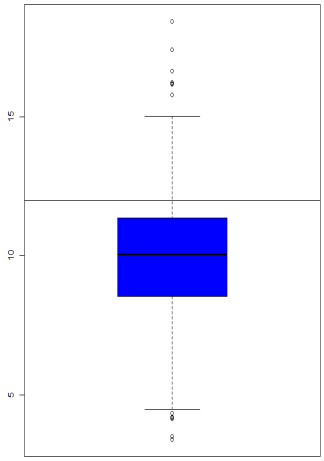
The problems associated with drawing histograms of continuous variables are much more challenging. Bandwidth must be estimated.

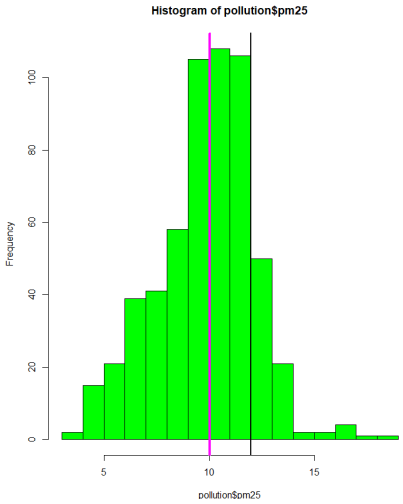
The rule of thumb for bandwidth is

$$b = \frac{\max(x) - \min(x)}{2(1 + \log_2 n)}$$

One Dimension Summaries

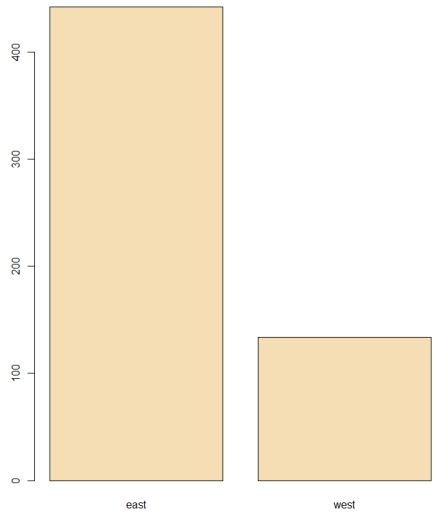
```
#Overlaying Features
boxplot(pollution$pm25, col = "blue")
abline(h = 12)

hist(pollution$pm25, col = "green")
abline(v = 12, lwd = 2) #lwd = line width
abline(v = median(pollution$pm25),
    col = "magenta", lwd = 4)
```





One Dimension Summaries

```
#Barplot
library(dplyr)
table(pollution$region) %>% barplot(col
= "wheat")
```



Multiple Dimension Summaries

There is an array of multiple dimension tools. Some of the key approaches are:

1. **Multiple or Overlayed 1-D plots** (Lattice/ggplot2): Using multiple boxplots or multiple histograms can be useful for seeing the relationship between two variables, especially when one is naturally categorical.

2. **Scatterplots**: Scatterplots are the natural tool for visualizing two continuous variables. Transformations of the variables (e.g. log or square-root transformation) may be necessary for effective visualization.

3. **Smooth scatterplots**: Similar in concept to scatterplots but rather plots a 2-D histogram of the data. Can be useful for scatterplots that may contain many, many data points.

17

Multiple Dimension Summaries

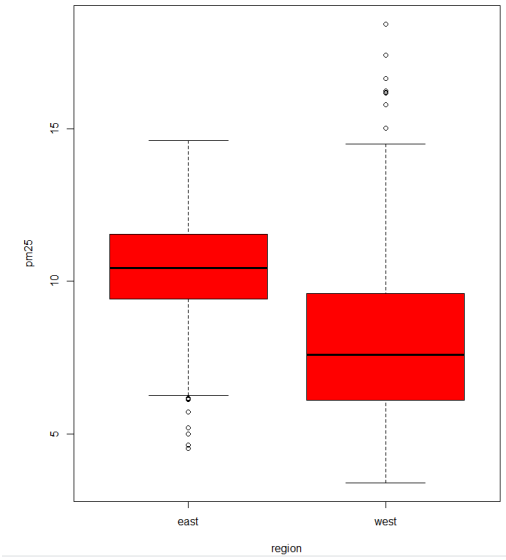
There is an array of multiple dimension tools. Some of the key approaches are:

4. **Overlayed or Multiple 2-D plots, conditioning plots (coplots)**: A conditioning plot, or coplot, shows the relationship between two variables as a third (or more) variable changes. For example, you might want to see how the relationship between air pollution levels and mortality changes with the season of the year. Here, air pollution and mortality are the two primary variables and season is the third variable varying in the background.
5. **Use color, size, shape to add dimensions**: Plotting points with different colors or shapes is useful for indicating a third dimension, where different colors can indicate different categories or ranges of something. Plotting symbols with different sizes can also achieve the same effect when the third dimension is continuous.
6. **Spinning/interactive plots**: Spinning plots can be used to simulate 3-D plots by allowing the user to essentially quickly cycle through many different 2-D projections so that the plot feels 3-D. These are sometimes helpful to capture unusual structure in the data, but I rarely use them.

18

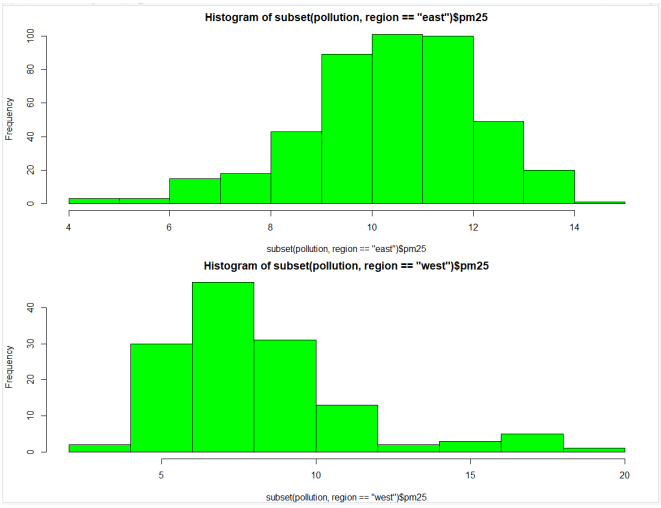
Multiple Dimension Plots

```
#Multiple Boxplots
boxplot(pm25 ~ region, data = pollution, col = "red")
```



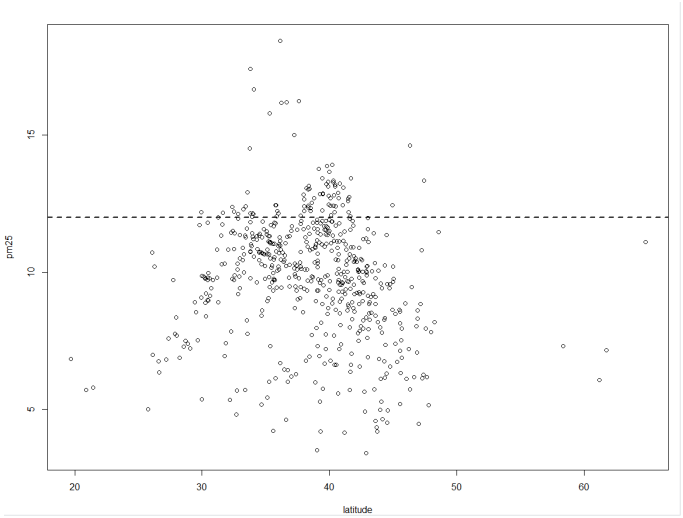
Multiple Dimension Plots

```
#Multiple Histograms
par(mfrow = c(2, 1), mar = c(4, 4, 2, 1))
hist(subset(pollution, region == "east")$pm25,
     col = "green")
hist(subset(pollution, region == "west")$pm25,
     col = "green")
```



Multiple Dimension Plots

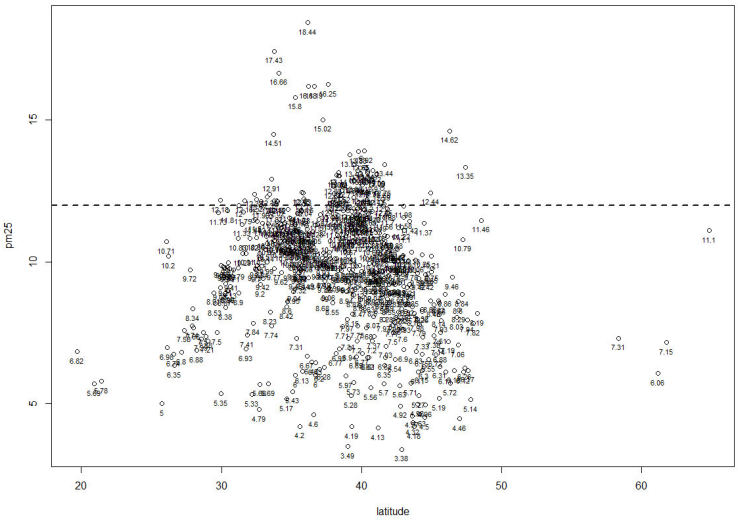
```
#Scatterplots
with(pollution, plot(latitude, pm25))
abline(h = 12, lwd = 2, lty = 2)
```



21

Multiple Dimension Plots

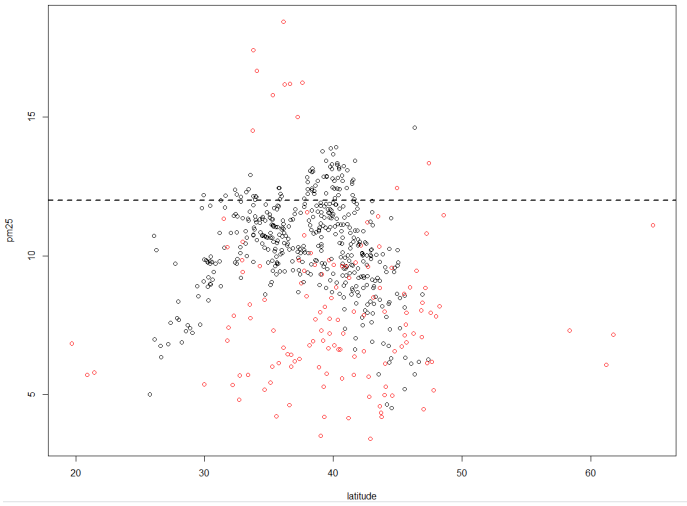
```
#Scatterplots
with(pollution, plot(latitude, pm25))
abline(h = 12, lwd = 2, lty = 2)

#Add labels to points
text(latitude, pm25, labels=round(pm25, 2),
pos=1, offset=0.5,cex=0.7)
```



22

Multiple Dimension Plots

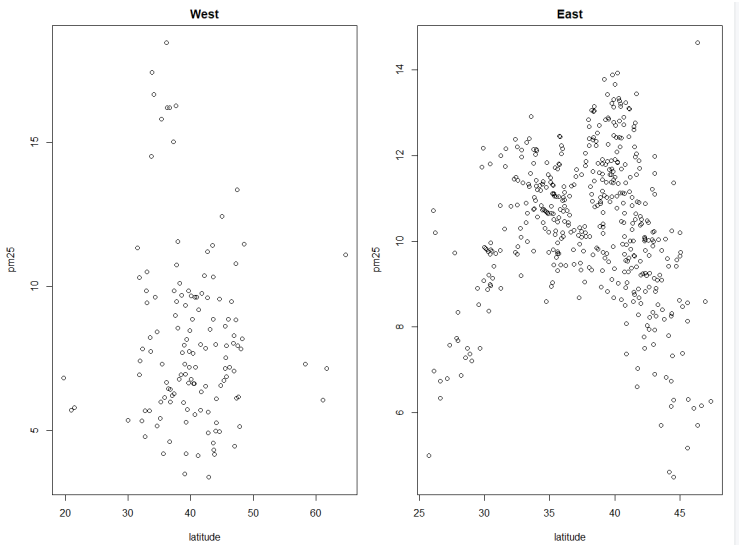
```
#Scatterplots with Color
with(pollution, plot(latitude, pm25, col = region))
abline(h = 12, lwd = 2, lty = 2)
```



23

Multiple Dimension Plots

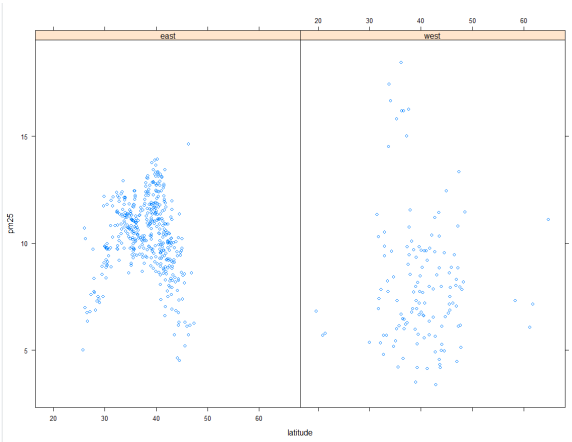
```
#Multiple Scatterplots
par(mfrow = c(1, 2), mar = c(5, 4, 2, 1))

with(subset(pollution, region == "west"),
plot(latitude, pm25, main = "West"))

with(subset(pollution, region == "east"),
plot(latitude, pm25, main = "East"))
```



24

Multiple Dimension Plots

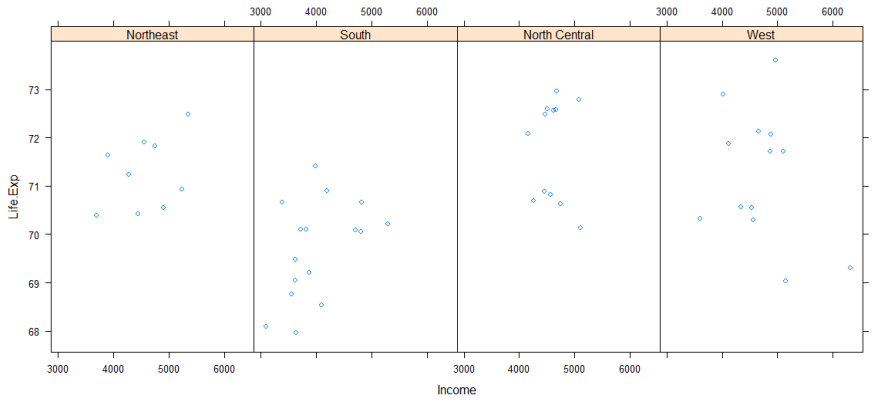
```
#Lattice
library(lattice)
xyplot(pm25 ~ latitude | region, data = pollution)
```



25

Multiple Dimension Plots

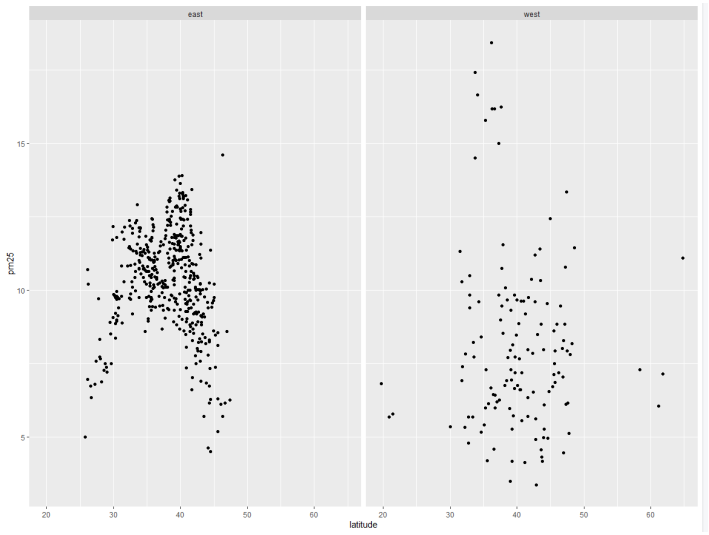
```
#Lattice
library(lattice)
state <- data.frame(state.x77, region = state.region)
xyplot(Life.Exp ~ Income | region, data = state, layout = c(4, 1))
```



26

Multiple Dimension Plots

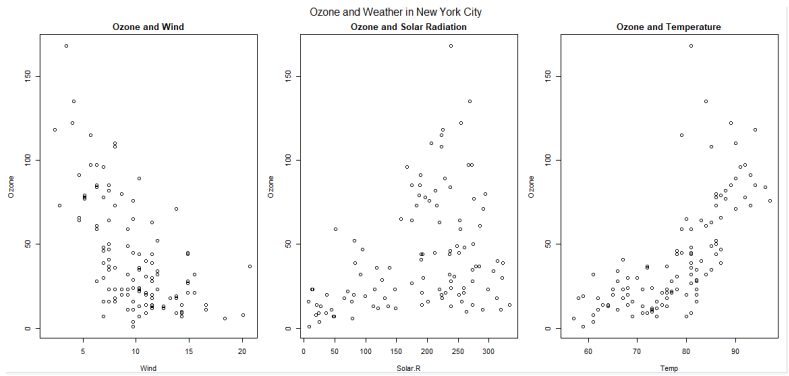
```
#ggplot2
library(ggplot2)
qplot(latitude, pm25, data = pollution,
    facets = . ~ region)
```



27

Multiple Dimension Plots

```
par(mfrow = c(1, 3), mar = c(4, 4, 2, 1), oma = c(0, 0, 2, 0))
with(airquality, {
        plot(Wind, Ozone, main = "Ozone and Wind")
        plot(Solar.R, Ozone, main = "Ozone and Solar Radiation")
        plot(Temp, Ozone, main = "Ozone and Temperature")
        mtext("Ozone and Weather in New York City", outer = TRUE)
        })
```



28

Summary of Plot Parameters

- par()
- points()
- col =
- breaks =
- abline()
- lwd =
- lty =
- mfrow()
- mfcol()
- mar =
- main =
- pch()
- xlab =
- ylab =
- type =
- offsets()
- bty =
- pos
- las = A numeric value indicating the orientation of the tick mark labels
- bg = background
- oma =
- title=
- mtext
- legend()
- outer=
- xlim =
- ylim=
- lines()
- cex = Scaling factor for text and characters relative to default. Default = 1.0
- mai = A numerical vector of the form c(bottom, left, top, right) which gives the margin size specified in inches
- new =
- labels

You can see the default values for global graphics parameters by calling the par() function and passing the name of the parameter in quotes.

```
> par("lty")
[1] "solid"
> par("col")
[1] "black"
> par("pch")
[1] 1
> par("bg")
[1] "white"
> par("mar")
[1] 5.1 4.1 4.1 2.1
> par("mfrow")
[1] 1 1
```

Source for more parameter (par()) values and definitions:
https://www.rdocumentation.org/packages/graphics/versions/3.6.2/topics/par

Graphics Devices

A graphics device is something where you can make a plot appear. Examples include:

1. A window on your computer (screen device)
2. A PDF file (file device)
3. **A PNG or JPEG file (file device)**
4. A scalable vector graphics (SVG) file (file device)

The list of devices supported by your installation of R is found in ?Devices. The screen is the default device.

Device Commands:
1. window()
2. device.off()
3. device.cur()
4. device.set(<integer>)
5. device.copy
6. dev.copy2pdf

```
>install.packages("grDevices")

Warning in install.packages :  package
'grDevices' is a base package, and
should not be updated

>library(grDevices)

> library(datasets)
>
> ## Create plot on screen device
> with(faithful, plot(eruptions, waiting))
>
> ## Add a main title
> title(main = "Old Faithful Geyser data")
>
> ## Copy my plot to a PNG file
> dev.copy(png, file = "geyserplot.png")
>
> ## Don't forget to close the PNG device!
> dev.off()
```

Graphics Devices

There are two basic types of file devices to consider: *vector* and *bitmap* devices. Some of the key vector formats are

1. pdf: useful for line-type graphics, resizes well, usually portable, not efficient if a plot has many objects/points
2. svg: XML-based scalable vector graphics; supports animation and interactivity, potentially useful for web-based plots
3. win.metafile: Windows metafile format (only on Windows)
4. postscript: older format, also resizes well, usually portable, can be used to create encapsulated postscript files; Windows systems often don't have a postscript viewer

```
> # Open PDF device; create 'myplot.pdf' in my working directory
> pdf(file = "myplot.pdf")
>
> # Create plot and send to a file (no plot appears on screen)
> with(faithful, plot(eruptions, waiting))
>
> # Annotate plot; still nothing on screen
> title(main = "Old Faithful Geyser data")
>
> # Close the PDF file device
> dev.off()
>
> # Now you can view the file 'myplot.pdf' on your computer
```


Graphics Devices

Some examples of bitmap formats are

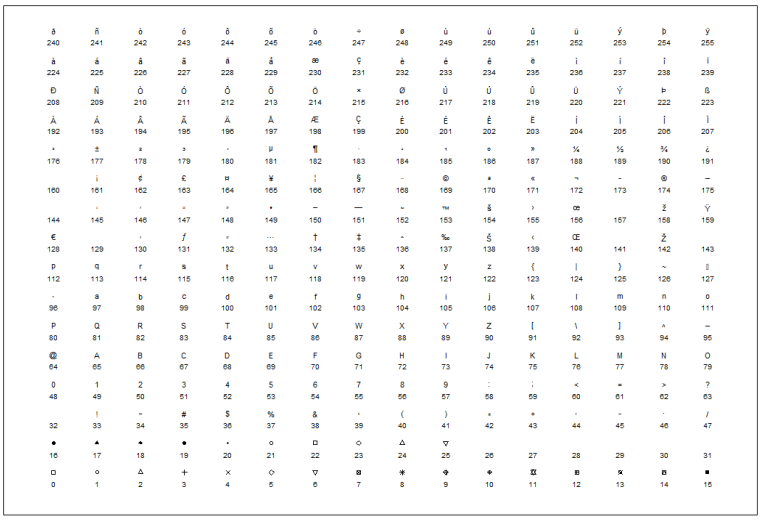
1. png: bitmapped format, good for line drawings or images with solid colors, uses lossless compression (like the old GIF format), most web browsers can read this format natively, good for plotting many points, does not resize well
2. jpeg: good for photographs or natural scenes, uses lossy compression, good for plotting many many many points, does not resize well, can be read by almost any computer and any web browser, not great for line drawings
3. tiff: Creates bitmap files in the TIFF format; supports lossless compression
4. bmp: a native Windows bitmapped format

```
# 1. Open jpeg file
jpeg("rplot.jpg", width = 350,
         height = "350")
# 2. Create the plot
plot(x = my_data$wt, y = my_data$mpg,
         pch = 16, frame = FALSE,
         xlab = "wt", ylab = "mpg",
         col = "#2E9FDF")
# 3. Close the file dev.off()
# 4. File prints to working directory
```

Plotting Symbols: pch (plotting character)

There are 256 different plotting symbols available in Windows (0 to 255). Here is a graphic showing all of them in sequence, from bottom left to top right. These symbols can be used for almost anything but to also change the graph markers.

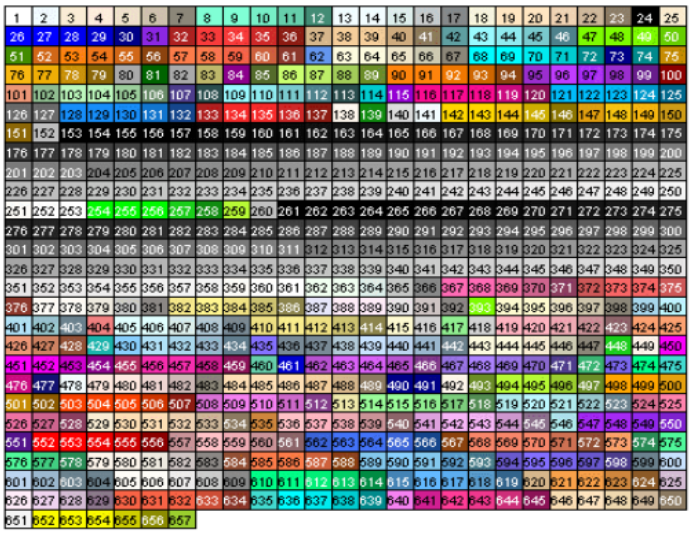
The default value for graphs, `pch=1`, is a small open circle in black.



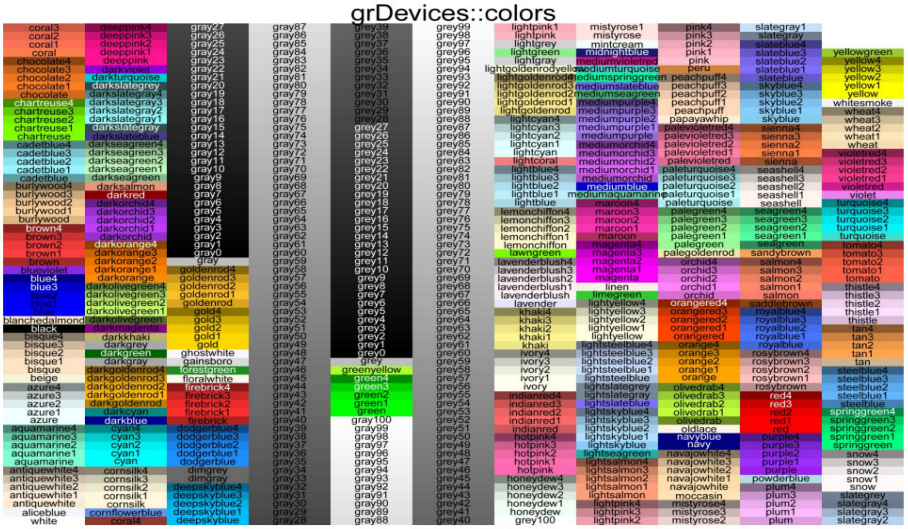
Plotting Specifications

The Color Chart for col =

Google "Colors in R" to find information on colors in R.



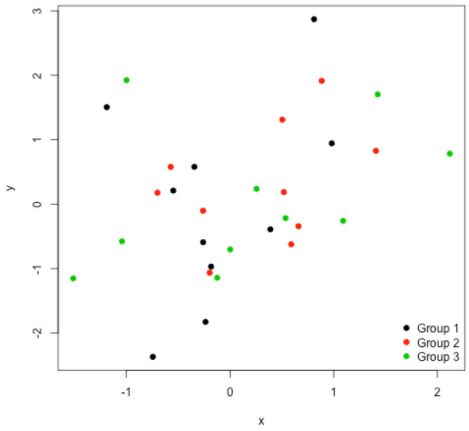
Colors in R



Plotting with Color in R

```
>set.seed(19)
> x <- rnorm(30)
> y <- rnorm(30)
> plot(x, y, col = rep(1:3, each = 10), pch = 19)
> legend("bottomright", legend = paste("Group", 1:3),
         col = 1:3, pch = 19, bty = "n")
```

Note: The "col =" parameter still randomizes the color despite it appearing we are assigning the colors 1, 2, and 3.



Default colors in R

Color Utilities in R

R has a number of utilities for dealing with colors and color palettes in your plots. For starters, the grDevices package has two functions

1. **colorRamp**: Take a palette of colors and return a function that takes valeus between 0 and 1, indicating the extremes of the color palette (e.g. see the gray() function)

2. **colorRampPalette**: Take a palette of colors and return a function that takes integer arguments and returns a vector of colors interpolating the palette (like **heat.colors()** or **topo.colors()**)

Both of these functions take palettes of colors and help to interpolate between the colors on the palette. They differ only in the type of object that they return.

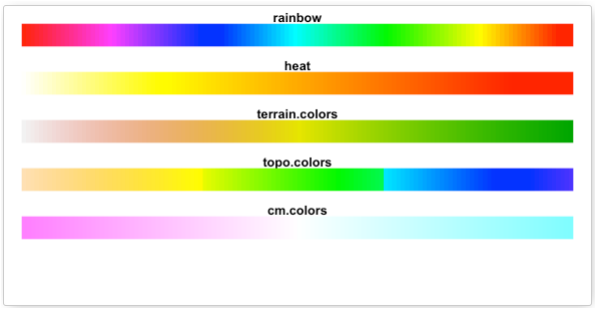
Finally, the function **colors()** lists the names of colors you can use in any plotting function. Typically, you would specify the color in a (base) plotting function via the **col** argument.

Plotting with Color in R

Predefined Color Palettes in R

There are 5 R base functions that can be used to generate a vector of n contiguous colors:

1. rainbow(n)
2. heat.colors(n)
3. terrain.colors(n)
4. topo.colors(n)
5. cm.colors(n)



colorRamp()

Both colorRamp() and colorRampPalette() handle that "mixing" process for you. Let's start with a simple palette of "red" and "blue" colors and pass them to colorRamp().

```
> pal <- colorRamp(c("red", "blue"))
> pal(0)
     [,1] [,2] [,3]
[1,]  255    0    0

> ## blue
> pal(1)
     [,1] [,2] [,3]
[1,]    0    0  255
>
> ## purple-ish
> pal(0.5)
      [,1] [,2]  [,3]
[1,] 127.5    0 127.5
```


colorRamp()

You can also pass a sequence of numbers to the pal() function.

```
> pal(seq(0, 1, len = 10))
           [,1] [,2]      [,3]
 [1,] 255.00000    0   0.00000
 [2,] 226.66667    0  28.33333
 [3,] 198.33333    0  56.66667
 [4,] 170.00000    0  85.00000
 [5,] 141.66667    0 113.33333
 [6,] 113.33333    0 141.66667
 [7,]  85.00000    0 170.00000
 [8,]  56.66667    0 198.33333
 [9,]  28.33333    0 226.66667
[10,]   0.00000    0 255.00000
```

The idea here is that colorRamp() gives you a function that allows you to interpolate between the two colors red and blue. You do not have to provide just two colors in your initial color palette; you can start with multiple colors and colorRamp() will interpolate between all of them.

colorRampPalette()

The colorRampPalette() function in manner similar to colorRamp((), however the function that it returns gives you a fixed number of colors that interpolate the palette.

```
> pal <- colorRampPalette(c("red", "yellow"))
```

Again we have a function pal() that was returned by colorRampPalette(), this time interpolating a palette containing the colors red and yellow. But now, the pal() function takes an integer argument specifing the number of interpolated colors to return.

```
> ## Just return red and yellow
> pal(2)
[1] "#FF0000" "#FFFF00"
```

#Note: The colors are represented in hexadecimal.

```
> ## Return 10 colors in between red and yellow
> pal(10)
[1] "#FF0000" "#FF1C00" "#FF3800" "#FF5500" "#FF7100" "#FF8D00" "#FFAA00"
[8] "#FFC600" "#FFE200" "#FFFF00"
```


rgb() Function

The **rgb()** function can be used to produce any color via red, green, blue proportions and return a hexadecimal representation.

```
> rgb(0, 0, 234, maxColorValue = 255)
[1] "#0000EA"
```

RColorBrewer Package()

One package on CRAN that contains interesting and useful color palettes is the RColorBrewer32 package.

The RColorBrewer package offers three types of palettes

1. Sequential: for numerical data that are ordered
2. Diverging: for numerical data that can be positive or negative, often representing deviations from some norm or baseline
3. Qualitative: for qualitative unordered data

RColorBrewer Package()

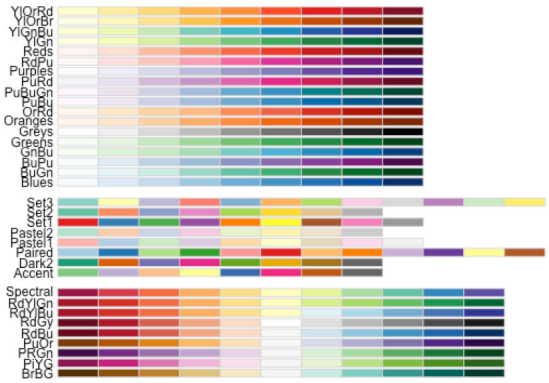
All of these palettes can be used in conjunction with the colorRamp() and colorRampPalette().

Here is a display of all the color palettes available from the RColorBrewer package.

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

The only real function in the RColorBrewer package is the **brewer.pal()** function which has two arguments

1. name: the name of the color palette you want to use
2. n: the number of colors you want from the palette (integer)



RColorBrewer palettes

RColorBrewer Package()

Volcano Data

```
str(volcano)
 num [1:87, 1:61] 100 101 102 103 104 105 105 106 107 108 ...
>
```

Topographic Information on Auckland's Maunga Whau Volcano

Description
Maunga Whau (Mt Eden) is one of about 50 volcanos in the Auckland volcanic field. This data set gives topographic information for Maunga Whau on a 10m by 10m grid.

Format
A matrix with 87 rows and 61 columns, rows corresponding to grid lines running east to west and columns to grid lines running south to north.

Source
Digitized from a topographic map by Ross Ihaka. These data should not be regarded as accurate.

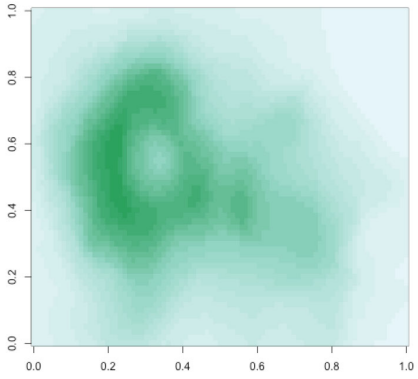
volcano

Filter | Cols: « < 1 - 50 > »

	V1	V2	V3	V4	V5	V6	V7	V8	V9	V10	V11
1	100	100	101	101	101	101	101	100	100	100	101
2	101	101	102	102	102	102	102	101	101	101	102
3	102	102	103	103	103	103	103	102	102	102	103
4	103	103	104	104	104	104	104	103	103	103	103
5	104	104	105	105	105	105	105	104	104	103	104
6	105	105	105	106	106	106	106	105	105	104	104
7	105	106	106	107	107	107	107	106	106	105	105
8	106	107	107	108	108	108	108	107	107	106	106
9	107	108	108	109	109	109	109	108	108	107	108
10	108	109	109	110	110	110	110	109	109	108	110
11	109	110	110	111	111	111	111	110	110	110	112
12	110	110	111	113	112	111	113	112	112	114	116
13	110	111	113	115	114	113	114	114	115	117	119
14	111	113	115	117	116	115	116	117	117	119	121
15	114	115	117	117	117	118	119	119	120	121	124

RColorBrewer Package()

```
> library(RColorBrewer)
> cols <- brewer.pal(3, "BuGn")
> cols
[1] "#E5F5F9" "#99D8C9" "#2CA25F"

> pal <- colorRampPalette(cols)
> image(volcano, col = pal(20))
```



Volcano data with color ramp palette

Plotting with Color in R

The **image()** Function

You have a matrix in R, and you want to visualize it – say, for example, with each cell colored according to the value in the cell. Not a heatmap, per se, as that requires clustering; just a simple, visual image.

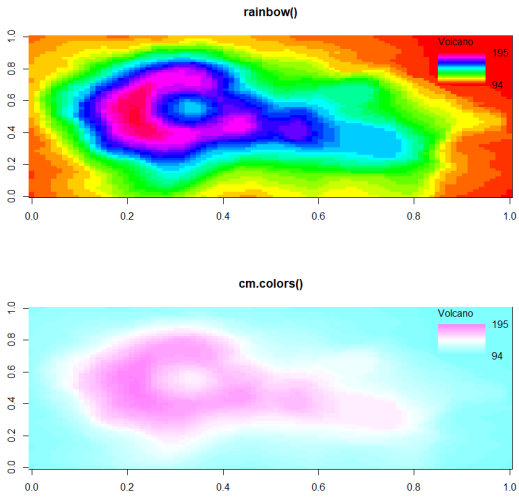
Creates and displays a gradient legend on a plot or image file. The place and size of the legend is defined by coordinates, previously identified.

```
image(x, y, z, zlim, xlim, ylim,
    col = heat.colors(100), add = FALSE,
    xaxs = "i", yaxs = "i", xlab, ylab,
    breaks, oldstyle = FALSE, useRaster, …)
```

See url for more documentation:
https://www.rdocumentation.org/packages/graphics/versions/3.6.2/topics/image

Plotting with Color in R

```
par(mfrow = c(2, 1))
image(volcano, col = rainbow(30), main = "rainbow()")
LegLoc   = cbind(x =c(0.85,0.95, 0.95, 0.85),c(0.9, 0.9, 0.7, 0.7))
legend.gradient(pnts=LegLoc, cols=rainbow(30),
          limits=c(min(volcano), max(volcano)), title = "Volcano")

image(volcano, col = cm.colors(30), main = "cm.colors()")
LegLoc   = cbind(x =c(0.85,0.95, 0.95, 0.85),c(0.9, 0.9, 0.7, 0.7))
legend.gradient(pnts=LegLoc, cols=cm.colors(30),
          limits=c(min(volcano), max(volcano)), title = "Volcano")
```



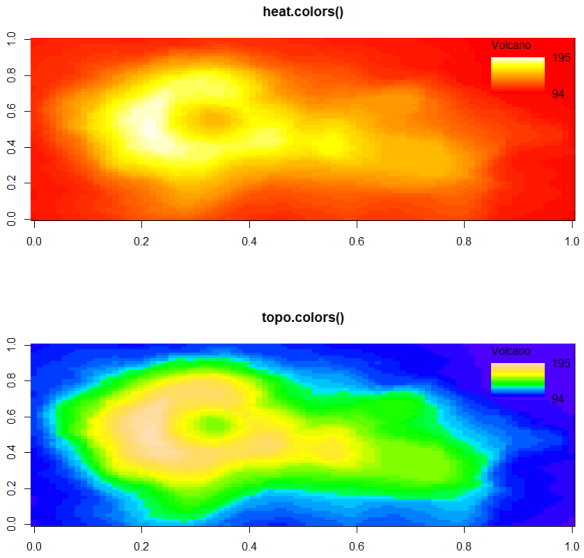
legend.gradient(pnts, cols = heat.colors(100), limits = c(0, 1), title = "Legend", ...)
creates and displays a gradient legend on a plot or image file. The place and size of the legend is defined by coordinates, previously identified.

Arguments

pnts:	x and y coordinates of the gradient location in the plot **#LegLoc - Legend Location**
cols:	a set of 2 or more colors used in the image, to create the gradient
limits:	to label the min and max values of the gradient in the legend
title:	to specify the title of the legend
...	other graphical parameters defined by image() or plot()

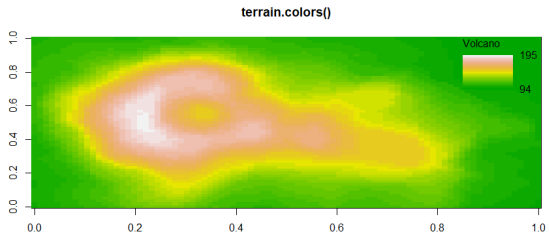
Plotting with Color in R

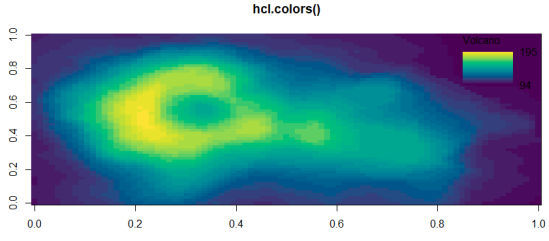
```
par(mfrow = c(2, 1))
image(volcano, col = heat.colors(30), main = "heat.colors()")
LegLoc   = cbind(x =c(0.85,0.95, 0.95, 0.85),c(0.9, 0.9, 0.7, 0.7))
legend.gradient(pnts=LegLoc, cols=heat.colors(30),
          limits=c(min(volcano), max(volcano)), title = "Volcano")

image(volcano, col = topo.colors(30), main = "topo.colors()")
LegLoc   = cbind(x =c(0.85,0.95, 0.95, 0.85),c(0.9, 0.9, 0.7, 0.7))
legend.gradient(pnts=LegLoc, cols=topo.colors(30),
          limits=c(min(volcano), max(volcano)), title = "Volcano")
```



Plotting with Color in R

```
par(mfrow = c(2, 1))
image(volcano, col = terrain.colors(30), main = "terrain.colors()")
LegLoc   = cbind(x =c(0.85,0.95, 0.95, 0.85),c(0.9, 0.9, 0.7, 0.7))
legend.gradient(pnts=LegLoc, cols=terrain.colors(30),
          limits=c(min(volcano), max(volcano)), title = "Volcano")

image(volcano, col = hcl.colors(30), main = "hcl.colors()")
LegLoc   = cbind(x =c(0.85,0.95, 0.95, 0.85),c(0.9, 0.9, 0.7, 0.7))
legend.gradient(pnts=LegLoc, cols=hcl.colors(30),
          limits=c(min(volcano), max(volcano)), title = "Volcano")
```





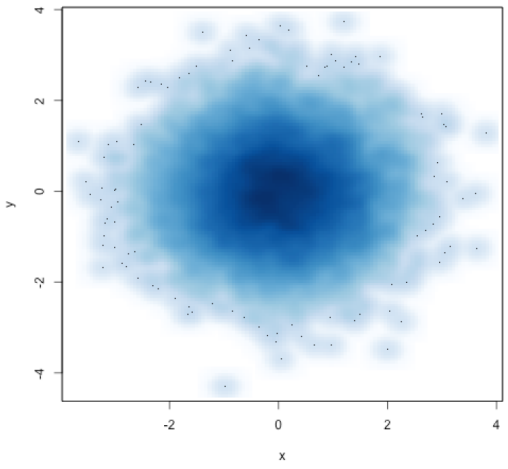
Smooth Scatterplots

The smoothScatter() function

A function that takes advantage of the color palettes in RColorBrewer is the smoothScatter() function, which is very useful for making scatterplots of very large datasets.

The smoothScatter() function essentially gives you a 2-D histogram of the data using a sequential palette (here "Blues").

```
> set.seed(1)
> x <- rnorm(10000)
> y <- rnorm(10000)
> smoothScatter(x, y)
```



smoothScatter function

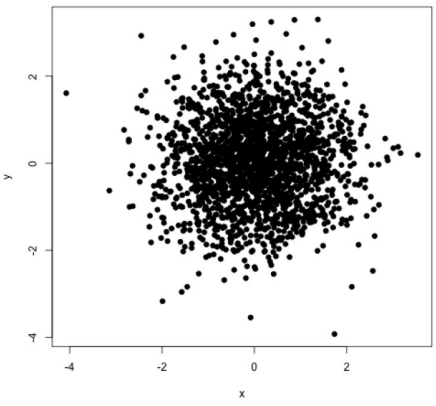
Adding Transparency

```
> rgb(1, 0, 0, 0.1) #The color red with a high level of transparency
[1] "#FF00001A"
```

Transparency can be useful when you have plots with a high density of points or lines.

For example, the scatterplot below has a lot of overplotted points and it's difficult to see what's happening in the middle of the plot region.

```
> set.seed(2)
> x <- rnorm(2000)
> y <- rnorm(2000)
> plot(x, y, pch = 19)
```

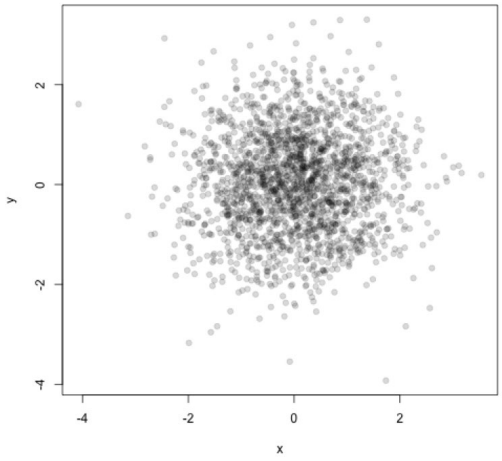


Scatterplot with no transparency

Adding Transparency

If we add some transparency to the black circles, we can get a better sense of the varying density of the points in the plot.

```
> plot(x, y, pch = 19, col = rgb(0, 0, 0, 0.15))
```



Scatterplot with transparency

Homework

Watch the following videos:

1. https://www.youtube.com/watch?v=ma6-0PSNLHo&feature=youtu.be
2. https://www.youtube.com/watch?v=UyopqXQ8TTM&feature=youtu.
3. https://www.youtube.com/watch?v=7QaR91TCc3k&feature=youtu.be
4. https://www.youtube.com/watch?v=3Q1ormd5oMA&feature=youtu.be
5. https://www.youtube.com/watch?v=4JJfGpVHTq4&feature=youtu.be
6. https://www.youtube.com/watch?v=9-cn0auNV58&feature=youtu.be

Interaction Plots

Interaction plot
An interaction plot is a visual representation of the interaction between the effects of two factors, or between a factor and a numeric variable. It is suitable for experimental data.

You can create an interaction plot with the **interaction.plot()** function. The command takes the general form:

```
> interaction.plot(dataset$var1, dataset$var2, dataset$response)
```

where var1 and var2 are the names of the explanatory variables and response is the name of the response variable.

If one of the explanatory variables is numeric and the other is a factor, list the numeric variable first and the factor second. This way the numeric variable is displayed along the x-axis and the factor is represented by separate lines on the plot.

Interaction Plots

Example: Interaction plot with ToothGrowth data

Consider the ToothGrowth dataset, which is included with R. The dataset gives the results of an experiment to determine the effect of two supplements (Vitamin C and Orange Juice), each at three different doses (0.5, 1 or 2 mg) on tooth length in guinea pigs.

The len variable gives the tooth growth, the supp variable gives the supplement type and the dose variable gives the supplement dose.

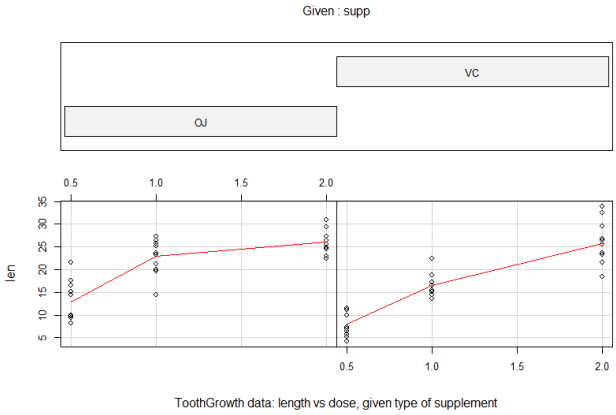
You can view more information about the ToothGrowth dataset by entering help(ToothGrowth).

Interaction Plots

Let's create an exploratory plot first:

```
> require(graphics)
coplot(len ~ dose | supp, data = ToothGrowth,
       panel = panel.smooth,
       xlab = "ToothGrowth data: length vs dose,
             given type of supplement")
```

The response is the length of odontoblasts (cells responsible for tooth growth) in 60 guinea pigs. Each animal received one of three dose levels of vitamin C (0.5, 1, and 2 mg/day) by one of two delivery methods, orange juice or ascorbic acid (a form of vitamin C and coded as VC).

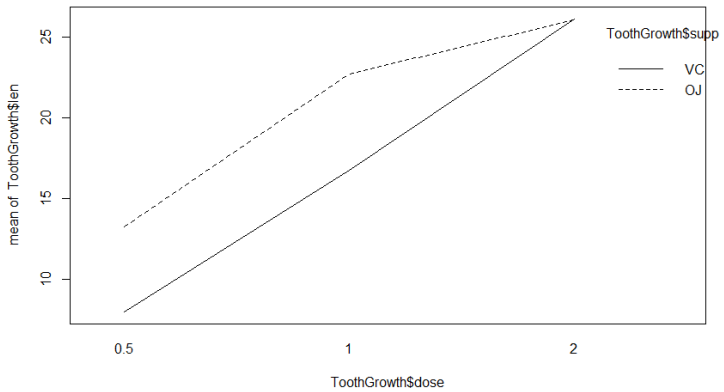


Interaction Plots

To create an interaction plot illustrating the interaction between supplement type and supplement dose, use the command:

```
> interaction.plot(ToothGrowth$dose, ToothGrowth$supp,
       ToothGrowth$len)
```

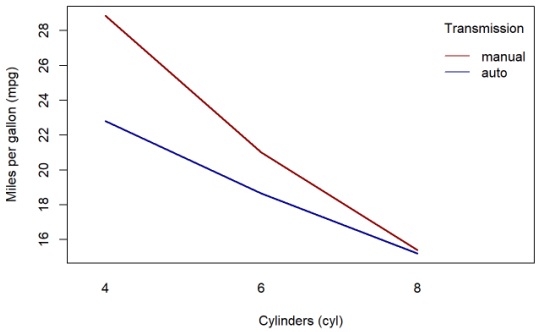
Source of Vitamin C Matters at Low Doses



Interaction Plots

Let's consider miles-per-gallon as a function of the number of cylinders (4, 6 or 8) and transmission type (automatic, manual) for the vehicles included in the Motor Trend "mtcars" dataset; itself included as part of the base installation "datasets" package.

```
attach(mtcars)
# coerce "am" variable to factor with appropriate labels/levels. For the purposes
of the plot, this is necessary only to ease labeling of the 'traces' for the legend.
mtcars$am <- factor(x = mtcars$am, labels = c("auto", "manual"))

interaction.plot(x.factor = mtcars$cyl,    # variable to plot on x-axis
        trace.factor = mtcars$am, # variable to specify "traces"; here, lines
        response = mtcars$mpg,    # variable to plot on y-axis
        fun = median,  # summary statistic to be plotted for response variable
        type = "l",     # type of plot, here "l" for lines
        ylab = "Miles per gallon (mpg)",
        xlab = "Cylinders (cyl)",
        col = c("blue4", "red4"),
        lty = 1,  # line type
        lwd = 2,  # line width
        trace.label = "Transmission",  # label for legend
        xpd = FALSE)  # 'clip' legend at border
```



What does this interaction tell you?

Interaction Plots

```
>attach(mtcars)
>reg1<-lm(mpg~am + cyl)
>summary(reg1)

Call:
lm(formula = mpg ~ am + cyl)

Residuals:
   Min     1Q Median     3Q    Max
-5.6856 -1.7172 -0.2657  1.8838  6.8144

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)  34.5224     2.6032  13.262 7.69e-14 ***
ammanual      2.5670     1.2914   1.988   0.0564 .
cyl          -2.5010     0.3608  -6.931 1.28e-07 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 3.059 on 29 degrees of freedom
Multiple R-squared:  0.759,  Adjusted R-squared:  0.7424
F-statistic: 45.67 on 2 and 29 DF,  p-value: 1.094e-09
```

```
>reg2<-lm(mpg~am * cyl)  # Alternate:  reg2<-lm(mpg~am + cyl + am:cyl)
>summary (reg2)

Call:
lm(formula = mpg ~ am * cyl)

Residuals:
    Min      1Q  Median      3Q     Max
-6.5255 -1.2820 -0.0191  1.6301  5.9745

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)  30.8735     3.1882   9.684 1.95e-10 ***
ammanual     10.1754     4.3046   2.364 0.025258 *
cyl          -1.9757     0.4485  -4.405 0.000141 ***
ammanual:cyl -1.3051     0.7070  -1.846 0.075507 .
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 2.939 on 28 degrees of freedom
Multiple R-squared:  0.7852, Adjusted R-squared:  0.7621
F-statistic: 34.11 on 3 and 28 DF,  p-value: 1.73e-09
```

Interaction Plots

Main Effects No Interaction

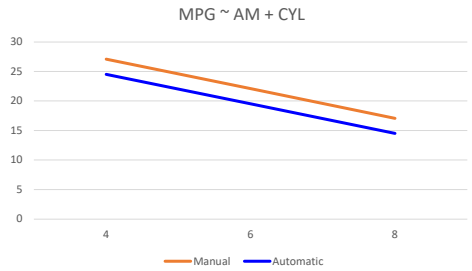
MPG = 34.5224 + 2.567AM_Manual + -2.5010 CYL

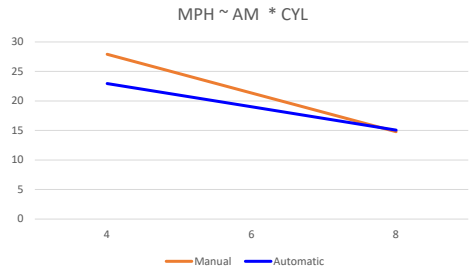
CYL	MPG Manual	MPG Automatic
4	27.0854	24.5184
6	22.0834	19.5164
8	17.0814	14.5144

Main Effects With Interaction

MPG = 30.8735 + 10.1754AM_Manual -1.9757 CYL - 1.3051AM_Manual:CYL

CYL	MPG Manual	MPG Automatic
4	27.9257	22.9707
6	21.3641	19.0193
8	14.8025	15.0679





61

Interaction Plots

Homework

Using interaction.plot(), explore the following relationships, provide interpretations of the plots, and construct regression models with and without the interaction terms. Finally, state whether or not you would include the interaction term in your regression and why based on your analysis. Create any additional plots that will help you understand the data.

1. mpg as a function of cyl across vs types
2. disp as a function of cyl across vs types
3. mpg as a function of gear across vs types
4. hp as a function of cyl across am types

62

Interaction Plots

Homework

This data is from the famous Framingham, Massachusetts Heart Study.
Framingham <- read.csv("http://publicifsv.sund.ku.dk/~tag/Teaching/share/data/Framingham.txt",sep=" ")

Framingham data

Description

Planned as a 20 years cohort study of residents aged 30-59 in Framingham town, Massachusetts, in 1948. Here: Data of 1406 persons (age > 44) and CHOL not missing at exam 1.

Variable Names

In order from left to right: 'data.frame': 1406 obs. of 9 variables:

Variable	Explanation (Unit)
ID	subject id
SEX	gender (1 for males, 2 for females)
AGE	Age in years
FRW	``Framingham relative weight'' (pct.) at baseline (52-222; 11 persons have missing values)
SBP	systolic blood pressure at baseline mmHg (90-300)
DBP	diastolic blood pressure at baseline mmHg (50-160)
CHOL	cholesterol at baseline mg/100ml (96-430)
CIG	cigarettes per day at baseline (0-60; 1 person has missing value)
CHD	0 if no ``coronary heart disease'' during follow-up, 1 if ``coronary heart disease'' at baseline (prevalent cases), x=2-10 if ``coronary heart disease'' was diagnosed at follow-up no. x

Interaction Plots

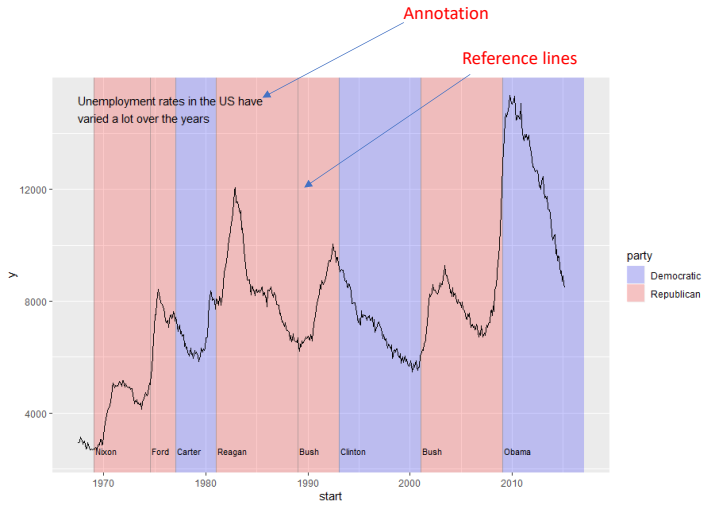
Homework

Create the following plots:

1. Create a smooth scatter plot of y = SBP vs. x = DBP
2. A transparency plot of y = SBP vs. x = DBP (Hint: Slide 48.)
3. A transparency plot of y = SBP vs. x = DBP
   - Coloring points by gender.
   - Add a legend to indicate gender
   - Add a gender label to the points using the SEX variable
4. What does this graph tell you about gender and blood pressure?

GGPlot Toolbox of Functions

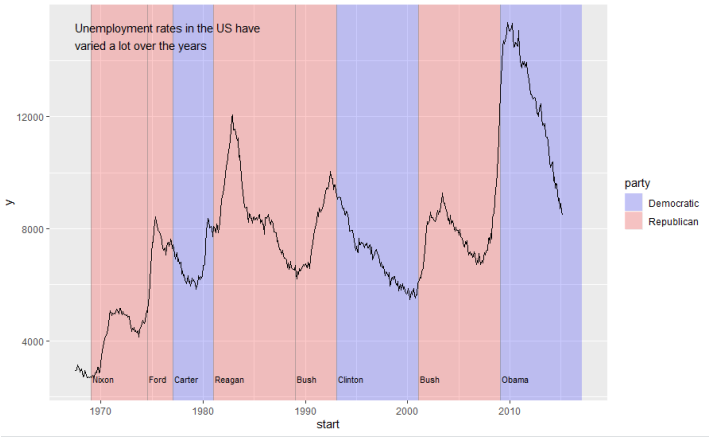
```
presidential <- subset(presidential, start > economics$date[1])
yrng <- range(economics$unemploy)
xrng <- range(economics$date)
caption <- paste(strwrap("Unemployment rates in the US have
                varied a lot over the years", 40), collapse = "\n")
ggplot(economics) +
 geom_rect(
  aes(xmin = start, xmax = end, fill = party),
  ymin = -Inf, ymax = Inf, alpha = 0.2, #alpha is a transparency parameter
  data = presidential
 ) +
 geom_vline(
  aes(xintercept = as.numeric(start)),
  data = presidential,
  color = "grey50", alpha = 0.5
 ) +
 geom_text(
  aes(x = start, y = 2500, label = name),
  data = presidential,
  size = 3, vjust = 0, hjust = 0, nudge_x = 50
 ) +
 geom_line(aes(date, unemploy)) +
 scale_fill_manual(values = c("blue", "red")
 ) +
 geom_text(
  aes(x, y, label = caption),
  data = data.frame(x = xrng[1], y = yrng[2], caption = caption),
  hjust = 0, vjust = 1, size = 4
 )
```



65

GGPlot Toolbox of Functions

```
presidential <- subset(presidential, start > economics$date[1])

yrng <- range(economics$unemploy)
xrng <- range(economics$date)
caption <- paste(strwrap("Unemployment rates in the US have
                varied a lot over the years", 40), collapse = "\n")
ggplot(economics) +
 geom_rect(
  aes(xmin = start, xmax = end, fill = party),
  ymin = -Inf, ymax = Inf, alpha = 0.2,
  data = presidential
 ) +
 geom_vline(
  aes(xintercept = as.numeric(start)),
  data = presidential,
  color = "grey50", alpha = 0.5
 ) +
 geom_text(
  aes(x = start, y = 2500, label = name),
  data = presidential,
  size = 3, vjust = 0, hjust = 0, nudge_x = 50
 ) +
 geom_line(aes(date, unemploy)) +
 scale_fill_manual(values = c("blue", "red")
 ) +
 annotate("text", x = xrng[1], y = yrng[2], label = caption,
       hjust = 0, vjust = 1, size = 4
 )
```

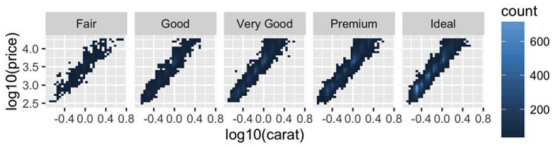


66

GGPlot Toolbox of Functions

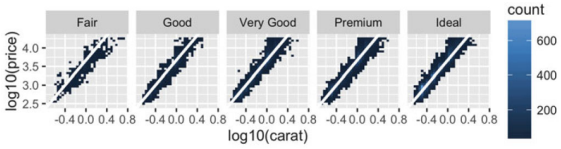
Heat Map with Facet Wrap

```
ggplot(diamonds, aes(log10(carat), log10(price))) +
geom_bin2d() +
facet_wrap(~cut, nrow = 1)
```



Heat Map with Abline & Facet Wrap

```
mod_coef <- coef(lm(log10(price) ~ log10(carat), data =
diamonds))
ggplot(diamonds, aes(log10(carat), log10(price))) +
geom_bin2d() +
geom_abline(intercept = mod_coef[1], slope = mod_coef[2],
colour = "white", size = 1) +
facet_wrap(~cut, nrow = 1)
```

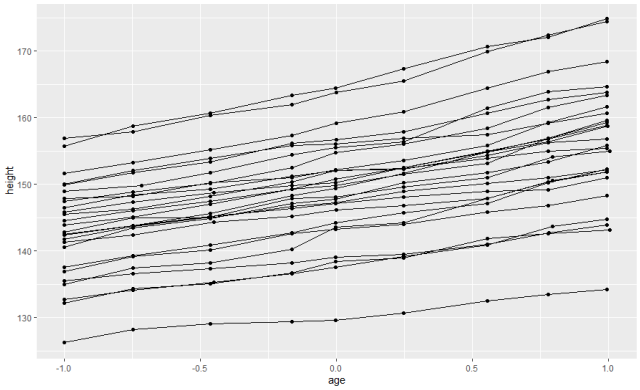


Possible modification to define bin size: geom_bin2d(binwidth = c(0.2, 0.2))

GGPlot Toolbox of Functions

Examining Groups

```
data(Oxboys, package = "nlme")
head(Oxboys, 100)
str(Oxboys)

ggplot(Oxboys, aes(age, height, group = Subject)) +
  geom_point() +
  geom_line()
```



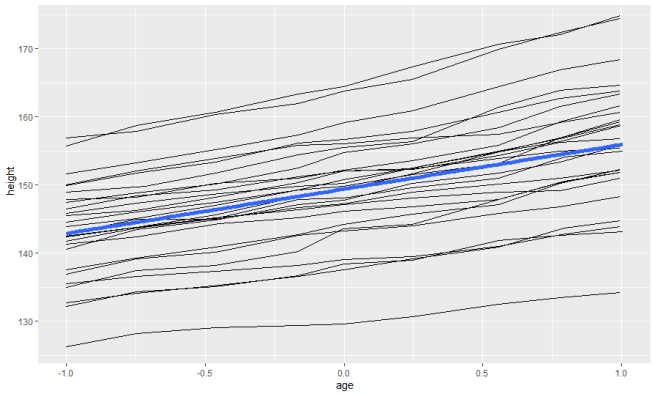
GGPlot Toolbox of Functions

Add an Overall Trend Line

```
ggplot(Oxboys, aes(age, height)) +
geom_line(aes(group = Subject)) +
geom_smooth(method = "lm", size = 2, se = FALSE)
```

Alternatively,

```
h <- ggplot(nlme::Oxboys, aes(age, height))
h <- h + geom_line(aes(group = Subject))
h <- h + geom_smooth(aes(group = 1), size = 2,
            method = "lm", se = FALSE)
h
```

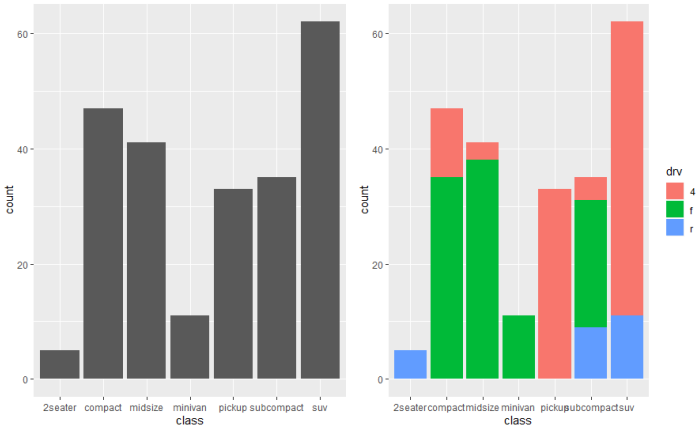


69

GGPlot Toolbox of Functions

The Fill Parameter

```
#Discrete Fill Variable
install.packages("gridExtra")
library(gridExtra)
par(mfrow = c(1,2))

P1 <- ggplot(mpg, aes(class)) +
  geom_bar()

P2 <- ggplot(mpg, aes(class, fill = drv)) +
  geom_bar()

grid.arrange(p1, p2, nrow = 2)
```

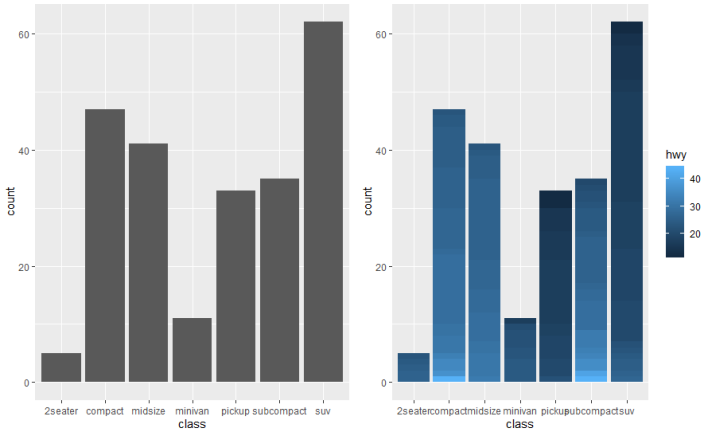


70

GGPlot Toolbox of Functions

The Fill Parameter

```
#Continuous Fill Variable
install.packages("gridExtra")
library(gridExtra)

ggplot(mpg, aes(class, fill = hwy)) +
  geom_bar()

ggplot(mpg, aes(class, fill = hwy, group = hwy)) +
  geom_bar()
```

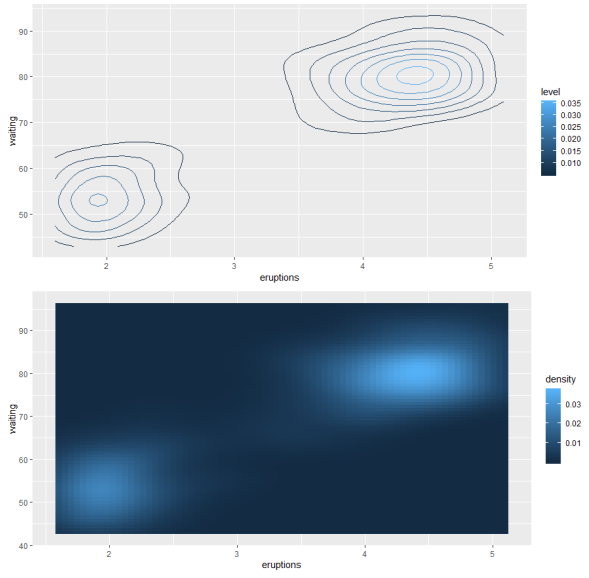


GGPlot Toolbox of Functions

Surface Plots

ggplot2 does not support true 3d surfaces. However, it does support many common tools for representing 3d surfaces in 2d: contours, colored tiles and bubble plots. These all work similarly, differing only in the aesthetic used for the third dimension.

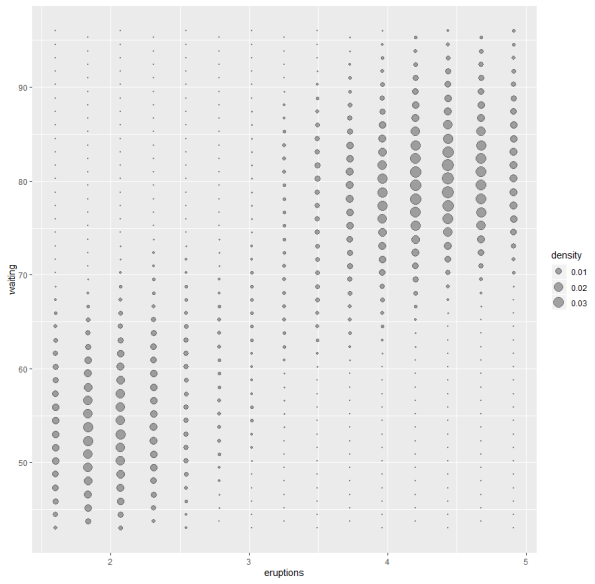
```
p1 <-ggplot(faithfuld, aes(eruptions, waiting)) +
  geom_contour(aes(z = density, color = ..level..))

p2 <-ggplot(faithfuld, aes(eruptions, waiting)) +
  geom_raster(aes(fill = density))

grid.arrange(p1, p2, nrow = 2)
```



GGPlot Toolbox of Functions

Bubble Plots

```
# Bubble plots work better with fewer observations
small <- faithfuld[seq(1, nrow(faithfuld), by = 10), ]
ggplot(small, aes(eruptions, waiting)) +
geom_point(aes(size = density), alpha = 1/3) +
scale_size_area()
```



73

GGPlot Toolbox of Functions

Creating Maps

There are four types of map data you might want to visualize:

1. vector boundaries
2. point metadata
3. area metadata
4. raster images

74

GGPlot Toolbox of Functions

Vector Boundaries

Vector boundaries are defined by a data frame with one row for each "corner" of a geographical region like a country, state, or county. It requires four variables:

1. **Latitude and Longitude**, giving the location of a point.
2. **Group**, a unique identifier for each contiguous region.
3. **ID**, the name of the region.

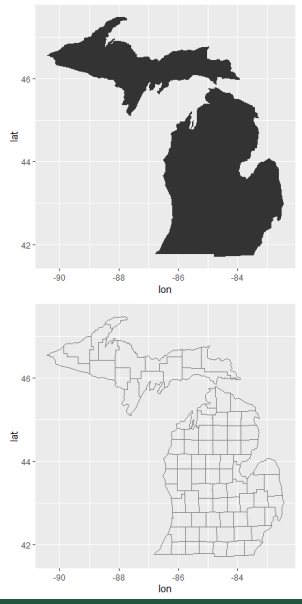
Vector boundaries based on mathematical formulas that define geometric primitives such as polygons, lines, curves, circles and rectangles.

Separate group and id variables are necessary because sometimes a geographical unit isn't a contiguous polygon. For example, Hawaii is composed of multiple islands that can't be drawn using a single polygon.

GGPlot Toolbox of Functions

Vector Boundaries

```
library(dplyr) #Package needed for the %>% function
mi_counties <- map_data("county", "michigan") %>%
  select(lon = long, lat, group, id = subregion)

p1 <-ggplot(mi_counties, aes(lon, lat)) +
  geom_polygon(aes(group = group)) +
  coord_quickmap()

p2 <-ggplot(mi_counties, aes(lon, lat)) +
  geom_polygon(aes(group = group), fill = NA, colour =
"grey50") +
  coord_quickmap()

grid.arrange(p1, p2, nrow = 2)
```



GGPlot Toolbox of Functions

Sources of Vector Boundary Data

1. **The USAboundaries package**, https://github.com/ropensci/USAboundaries which contains state, county and zip code data for the US. As well as current boundaries, it also has state and county boundaries going back to the 1600s.

2. **The tigris package**, https://github.com/walkerke/tigris, makes it easy to access the US Census TIGRIS shapefiles. It contains state, county, zipcode, and census tract boundaries, as well as many other useful datasets.

3. **The rnaturalearth package** bundles up the free, high-quality data from http://naturalearthdata.com/. It contains country borders, and borders for the top-level region within each country (e.g. states in the USA, regions in France, counties in the UK).

4. **The osmar package**, https://cran.r-project.org/package=osmar wraps up the OpenStreetMap API so you can access a wide range of vector data including individual streets and buildings

GGPlot Toolbox of Functions

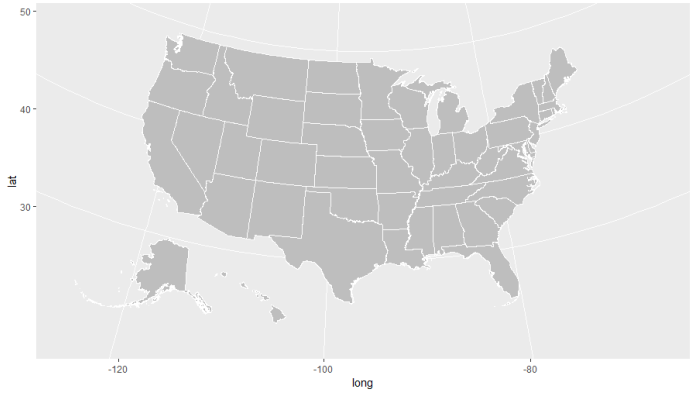
Creating a map

```
r = getOption("repos")
r["CRAN"] = "http://cran.us.r-project.org"
options(repos = r)
install.packages(c("readxl", "ggplot2", "tidyverse", "choroplethrMaps", "devtools", "dplyr", "yaml", "knitr"))

devtools::install_github("UrbanInstitute/urbnmapr")

library(readxl)
library(ggplot2)
library(tidyverse)
library(devtools)
library(choroplethrMaps)
library(urbnmapr)
library(dplyr)
library(yaml)
library(knitr)
```

GGPlot Toolbox of Functions

Creating a Country Map

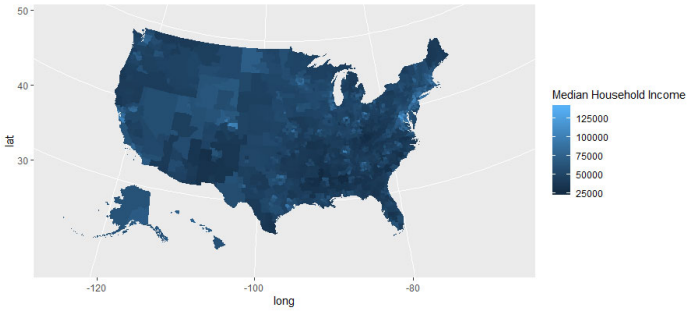
```
ggplot() +
  geom_polygon(data = urbnmapr::states, mapping = aes(x = long, y = lat, group =
group), fill = "grey", color = "white") +
  coord_map(projection = "albers", lat0 = 39, lat1 = 45)
```



GGPlot Toolbox of Functions

Creating a Country Gradient Map

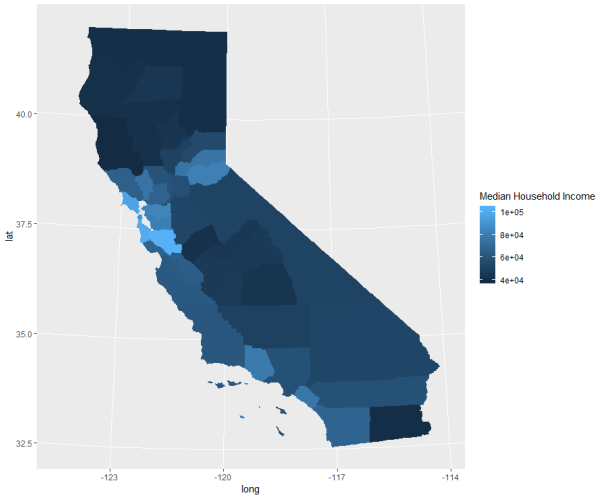
```
data(state.map)
data("state.regions")
data("statedata")

ggplot(state.map, aes(long, lat, group=group)) + geom_polygon()
household_data <- left_join(countydata, counties, by = "county_fips")
household_data %>%
  ggplot(aes(long, lat, group = group, fill = medhhincome)) +
  geom_polygon(color = NA) +
  coord_map(projection = "albers", lat0 = 39, lat1 = 45) +
  labs(fill = "Median Household Income")
```



GGPlot Toolbox of Functions

Creating a Country Gradient Map

```
state.map.CA <- subset(state.map, region=="california")
ggplot(state.map.CA, aes(long, lat, group=group)) + geom_polygon()
household_data_CA <-subset(household_data, state_abbv=="CA")
household_data_CA %>%

ggplot(aes(long, lat, group = group, fill = medhhincome)) +
 geom_polygon(color = NA) +
 coord_map(projection = "albers", lat0 = 39, lat1 = 45) +
 labs(fill = "Median Household Income")
```



81

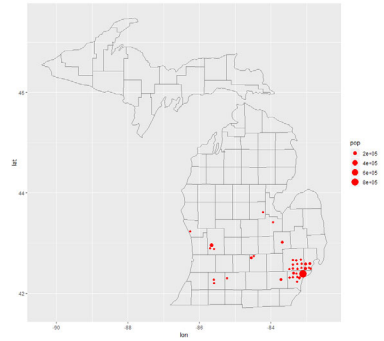
GGPlot Toolbox of Functions

Point Metadata

Point metadata connects locations (defined by latitude and longitude) with other variables.

```
mi_cities <- maps::us.cities %>%
  tbl_df() %>%
  filter(country.etc =="MI") %>%
  select(-country.etc, lon = long) %>%
  arrange(desc(pop))
mi_cities

ggplot(mi_cities, aes(lon, lat)) +
  geom_polygon(aes(group = group), mi_counties, fill = NA, colour = "grey50") +
  geom_point(aes(size = pop), colour = "red") +
  scale_size_area() +
  coord_quickmap()
```

```
> mi_cities
# A tibble: 36 x 5
   name                pop   lat   lon capital
   <chr>             <int> <dbl> <dbl>   <int>
 1 Detroit MI       871780  42.4 -83.1       0
 2 Grand Rapids MI  193006  43.0 -85.7       0
 3 Warren MI        132537  42.5 -83.0       0
 4 Sterling Heights MI 127027  42.6 -83.0       0
 5 Lansing MI       117236  42.7 -84.6       2
 6 Flint MI         115691  43.0 -83.7       0
 7 Ann Arbor MI     113716  42.3 -83.7       0
 8 Clinton MI       100517  42.6 -82.9       0
 9 Livonia MI        97722  42.4 -83.4       0
10 Dearborn MI       94681  42.3 -83.2       0
# … with 26 more rows
```



82

GGPlot Toolbox of Functions

Area Metadata

Area metadata is associated with an area not with a point. For example, we might want to add census data to a map.

```
library(dplyr)
library(maps)
library(ggplot2)
library(gridExtra)

View(ggplot2::midwest)

mi_census <- ggplot2::midwest %>%
  tbl_df() %>%
  filter(state =="MI") %>%
  mutate(county = tolower(county)) %>%
  select(county, area, poptotal, percwhite, percblack)
```

```
> head(mi_census, 20)
# A tibble: 20 x 5
   county      area poptotal percwhite percblack
   <chr>      <dbl>    <int>     <dbl>     <dbl>
 1 alcona     0.041    10145      98.8     0.266
 2 alger      0.051     8972      93.9     2.37
 3 allegan    0.049    90509      95.9     1.60
 4 alpena     0.034    30605      99.2     0.114
 5 antrim     0.031    18185      98.4     0.126
 6 arenac     0.021    14931      98.4     0.0670
 7 baraga     0.054     7954      87.6     0.616
 8 barry      0.034    50057      98.7     0.208
 9 bay        0.026   111723      96.4     1.11
10 benzie     0.02     12200      97.2     0.246
11 berrien    0.033   161378      82.6    15.4
12 branch     0.029    41502      97.1     1.70
13 calhoun    0.042   135982      87.3    10.6
14 cass       0.03     49477      90.6     7.53
15 charlevoix 0.022    21468      97.8     0.0792
16 cheboygan  0.048    21398      97.4     0.0701
17 chippewa   0.078    34604      81.9     6.31
18 clare      0.034    24952      98.8     0.160
19 clinton    0.034    57883      97.9     0.377
20 crawford   0.034    12260      96.3     2.15
```


GGPlot Toolbox of Functions

Area Metadata

GGPlot Midwest Dataset

```
> unique(ggplot2::midwest$state)
[1] "IL" "IN" "MI" "OH" "WI"
```

```
> str(ggplot2::midwest)
Classes 'tbl_df', 'tbl' and 'data.frame':       437 obs. of  28 variables:
 $ PID                       : int  561 562 563 564 565 566 567 568 569 570 ...
 $ county                    : chr  "ADAMS" "ALEXANDER" "BOND" "BOONE" ...
 $ state                     : chr  "IL" "IL" "IL" "IL" ...
 $ area                      : num  0.052 0.014 0.022 0.017 0.018 0.05 0.017 0.027 0.024 0.058 ...
 $ poptotal                  : int  66090 10626 14991 30806 5836 35688 5322 16805 13437 173025 ...
 $ popdensity                : num  1271 759 681 1812 324 ...
 $ popwhite                  : int  63917 7054 14477 29344 5264 35157 5298 16519 13384 146506 ...
 $ popblack                  : int  1702 3496 429 127 547 50 1 111 16 16559 ...
 $ popamerindian             : int  98 19 35 46 14 65 8 30 8 331 ...
 $ popasian                  : int  249 48 16 150 5 195 15 61 23 8033 ...
 $ popother                  : int  124 9 34 1139 6 221 0 84 6 1596 ...
 $ percwhite                 : num  96.7 66.4 96.6 95.3 90.2 ...
 $ percblack                 : num  2.575 32.9 2.862 0.412 9.373 ...
 $ percamerindan             : num  0.148 0.179 0.233 0.149 0.24 ...
 $ percasian                 : num  0.3768 0.4517 0.1067 0.4869 0.0857 ...
 $ percother                 : num  0.1876 0.0847 0.2268 3.6973 0.1028 ...
 $ popadults                 : int  43298 6724 9669 19272 3979 23444 3583 11323 8825 95971 ...
 $ perchsd                   : num  75.1 59.7 69.3 75.5 68.9 ...
 $ percollege                : num  19.6 11.2 17 17.3 14.5 ...
 $ percprof                  : num  4.36 2.87 4.49 4.2 3.37 ...
 $ poppovertyknown           : int  63628 10529 14235 30337 4815 35107 5241 16455 13081 154934 ...
 $ percpovertyknown          : num  96.3 99.1 95 98.5 82.5 ...
 $ percbelowpoverty          : num  13.15 32.24 12.07 7.21 13.52 ...
 $ percchildbelowpovert      : num  18 45.8 14 11.2 13 ...
 $ percadultpoverty          : num  11.01 27.39 10.85 5.54 11.14 ...
 $ percelderlypoverty        : num  12.44 25.23 12.7 6.22 19.2 ...
 $ inmetro                   : int  0 0 0 1 0 0 0 0 0 1 ...
 $ category                  : chr  "AAR" "LHR" "AAR" "ALU" ...
```

perchsd – percentage high school degree
inmetro – In a metro area

GGPlot Toolbox of Functions

Area Metadata

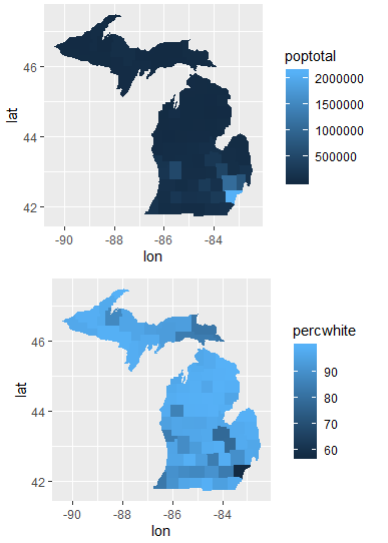
```
census_counties <- left_join(mi_census, mi_counties,
            by = c("county" ="id"))
census_counties
```

```
> census_counties
# A tibble: 1,472 x 8
   county  area poptotal percwhite percblack   lon   lat group
   <chr>  <dbl>    <int>     <dbl>     <dbl> <dbl> <dbl> <dbl>
 1 alcona 0.041    10145      98.8     0.266 -83.9  44.9     1
 2 alcona 0.041    10145      98.8     0.266 -83.4  44.9     1
 3 alcona 0.041    10145      98.8     0.266 -83.4  44.9     1
 4 alcona 0.041    10145      98.8     0.266 -83.3  44.8     1
 5 alcona 0.041    10145      98.8     0.266 -83.3  44.8     1
 6 alcona 0.041    10145      98.8     0.266 -83.3  44.8     1
 7 alcona 0.041    10145      98.8     0.266 -83.3  44.7     1
 8 alcona 0.041    10145      98.8     0.266 -83.3  44.7     1
 9 alcona 0.041    10145      98.8     0.266 -83.3  44.7     1
10 alcona 0.041    10145      98.8     0.266 -83.3  44.6     1
# … with 1,462 more rows
```


GGPlot Toolbox of Functions

Area Metadata

```
mi_census <- ggplot2::midwest %>%
  tbl_df() %>%
  filter(state =="MI") %>%
  mutate(county = tolower(county)) %>%
  select(county, area, poptotal, percwhite, percblack)
mi_census

mi_counties <- map_data("county", "michigan") %>%
  select(lon = long, lat, group, id = subregion)

census_counties <- left_join(mi_census, mi_counties, by = c("county" ="id"))
census_counties

p1 <-ggplot(census_counties, aes(lon, lat, group = county)) +
  geom_polygon(aes(fill = poptotal)) +
  coord_quickmap()

p2 <-ggplot(census_counties, aes(lon, lat, group = county)) +
  geom_polygon(aes(fill = percwhite)) +
  coord_quickmap()

grid.arrange(p1, p2, nrow = 2)
```



GGPlot Toolbox of Functions

Raster Images

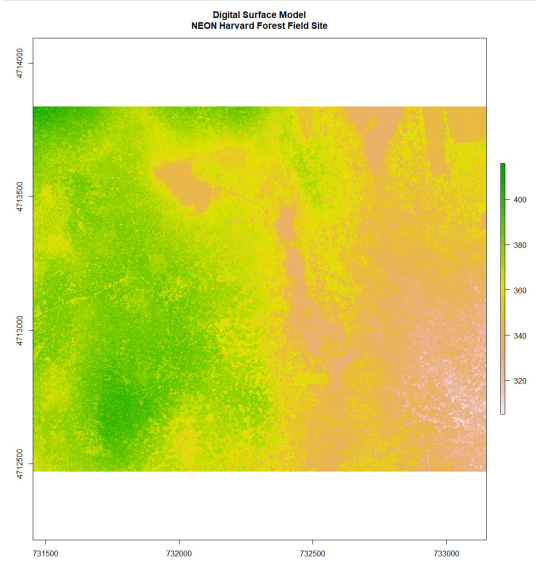
Raster graphics are bitmaps. A bitmap is a grid of individual pixels that collectively compose an image. Raster graphics render images as a collection of countless tiny squares.

```
install.packages("tiff")
library(tiff)
img <- readTIFF(system.file("img", "Rlogo.tiff", package="tiff"))
grid::grid.raster(img)
```

GGPlot Toolbox of Functions

Raster Images

```
library(rgdal)
library(raster)

# set working directory to ensure R can find the file we wish to import
# setwd("working-dir-path-here")

# import raster
DSM_HARV <- raster("C:/NEON-DS-Airborne-Remote-
Sensing/HARV/DSM/HARV_dsmCrop.tif")


# Plot raster object
plot(DSM_HARV,
    main="Digital Surface Model\nNEON Harvard Forest Field Site")
```



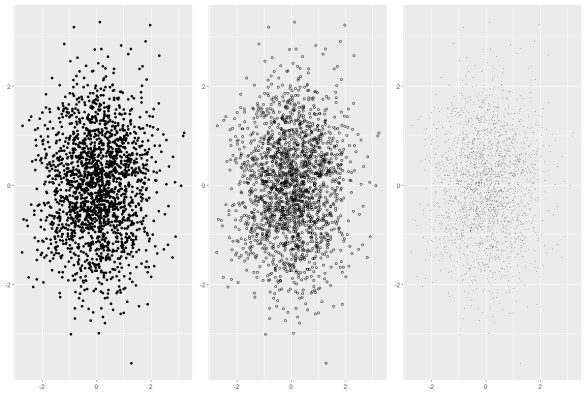
Dealing with Overplotting

When the data is large, points will be often plotted on top of each other, obscuring the true relationship, making any conclusions drawn from the graphic suspect. This problem is called **overplotting**.

There are a number of ways to deal with it depending on the size of the data and severity of the overplotting.

1. Making the points smaller or using hollow glyphs.

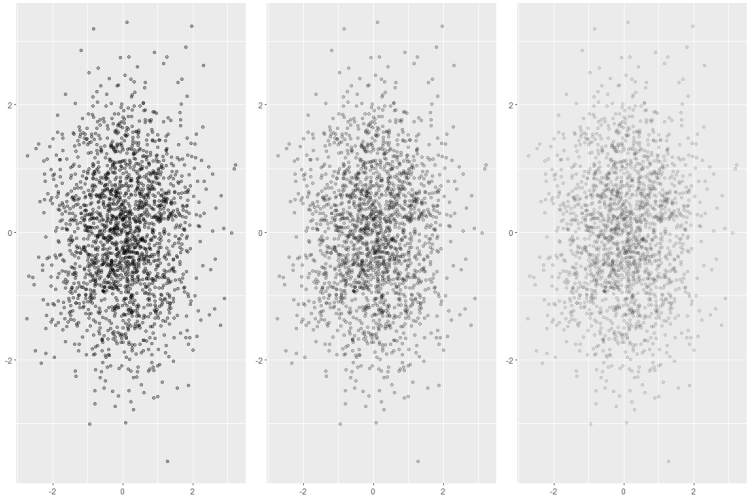
```
library(gridExtra)
df <- data.frame(x = rnorm(2000), y = rnorm(2000))

norm <- ggplot(df, aes(x, y)) + xlab(NULL) + ylab(NULL)
p1 <- norm + geom_point()
p2 <- norm + geom_point(shape = 1) # Hollow circles
p3 <- norm + geom_point(shape = ".") # Pixel sized

grid.arrange(p1, p2, p3,nrow = 1)
```



Dealing with Overplotting

2. For larger datasets with more overplotting, you can use alpha blending (transparency) to make the points transparent. If you specify alpha as a ratio, the denominator gives the number of points that must be overplotted to give a solid colour. Values smaller than ~1/500 are rounded down to zero, giving completely transparent points.

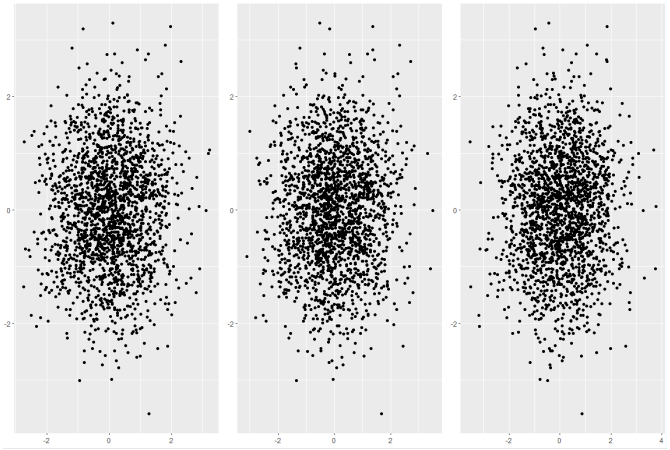
```
p1 <- norm + geom_point(alpha = 1 / 3)
p2 <-norm + geom_point(alpha = 1 / 5)
p3 <-norm + geom_point(alpha = 1 / 10)

grid.arrange(p1, p2, p3,nrow = 1)
```



Dealing with Overplotting

3. If there is some discreteness in the data, you can randomly jitter the points to alleviate some overlaps with geom jitter(). This can be particularly useful in conjunction with transparency. By default, the amount of jitter added is 40% of the resolution of the data, which leaves a small gap between adjacent regions. You can override the default with width and height arguments.

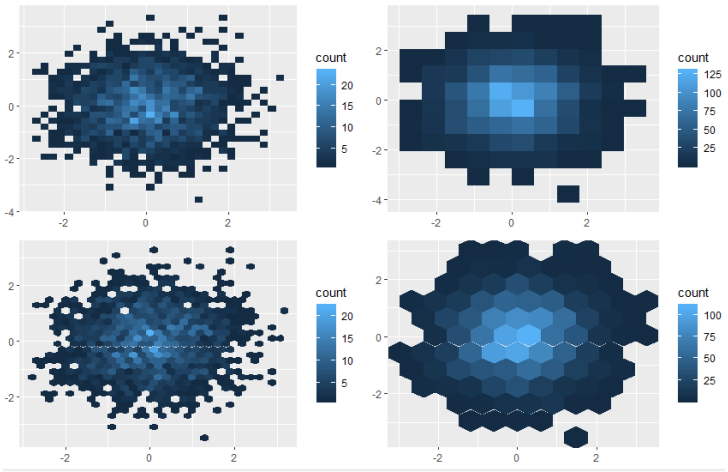
```
p1 <- norm + geom_jitter()
p2 <-norm + geom_jitter(width = .7)
p3 <-norm + geom_jitter(width = 0.9)

grid.arrange(p1, p2, p3,nrow = 1)
```



Dealing with Overplotting

4. We can think of overplotting as a 2d density estimation problem, which can be handled with geom_bin2d() or geom_hex().

```
install.packages("hexbin")
library(hexbin)

p1 <-norm + geom_bin2d()
p2 <-norm + geom_bin2d(bins = 10)

p3 <-norm + geom_hex()
p4 <-norm + geom_hex(bins = 10)

grid.arrange(p1, p2, p3, p4, nrow = 2)
```



Statistical Summaries

You can summarize data using statistical transformations, or stats. Without realizing it, we actually have already been using stat functions to build our plots. This is because the summary geoms (e.g. geom_boxplot), uses these functions natively (e.g. stat_boxplot). We have also used stat_bin(), stat_bbin2d() to create counts and densities for plotting.

The stat functions behind the various geoms that summarize data are useful to know, as they will provide more detail and information in the documentation. However, there are some stats that are separate from the basic geom functions we have been learning so far. For example,

1. stat_function(): computes y values from a function of x values.
2. stat_unique(): removes duplicate rows
3. stat_summary(): summarizes y values at distinct x values

Statistical Summaries

stat_summary operates on unique x; stat_summary_bin operates on binned x. They are more flexible versions of stat_bin(): instead of just counting, they can compute any aggregate.

```
stat_summary_bin(mapping = NULL, data = NULL, geom = "pointrange",
  position = "identity", ..., fun.data = NULL, fun.y = NULL,
  fun.ymax = NULL, fun.ymin = NULL, fun.args = list(), bins = 30,
  binwidth = NULL, breaks = NULL, na.rm = FALSE, show.legend = NA,
  inherit.aes = TRUE)
```

```
stat_summary(mapping = NULL, data = NULL, geom = "pointrange",
  position = "identity", ..., fun.data = NULL, fun.y = NULL,
  fun.ymax = NULL, fun.ymin = NULL, fun.args = list(),
  na.rm = FALSE, show.legend = NA, inherit.aes = TRUE)
```

fun.data
Complete summary function. Should take numeric vector as input and return data frame as output

fun.ymin
ymin summary function (should take numeric vector and return single number)

fun.y
y summary function (should take numeric vector and return single number)

fun.ymax
ymax summary function (should take numeric vector and return single number)

See documentation at:
https://ggplot2.tidyverse.org/reference/stat_summary.html

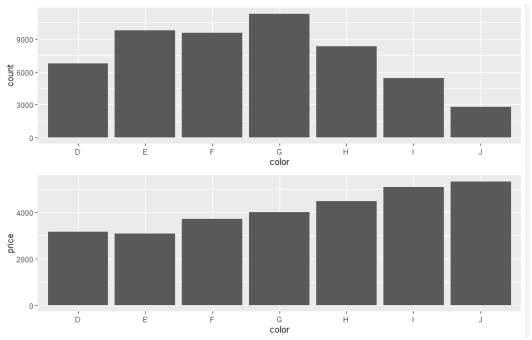
Statistical Summaries

We can use these parameters to change the metric that is plotted on the graph.

1. fun.data: Complete summary function. Should take numeric vector as input and return data frame as output
2. fun.ymin: ymin summary function (should take numeric vector and return single number)
3. fun.y: y summary function (should take numeric vector and return single number)
4. fun.ymax: ymax summary function (should take numeric vector and return single number)

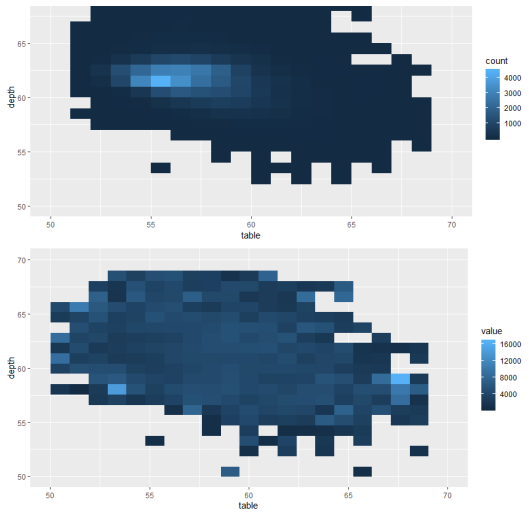
```
p1 <- ggplot(diamonds, aes(color)) +
  geom_bar()

p2 <- ggplot(diamonds, aes(color, price)) +
  geom_bar(stat = "summary_bin", fun.y = mean)

grid.arrange(p1, p2, nrow = 2)
```

Average Price



Statistical Summaries

```
p1 <- ggplot(diamonds, aes(table, depth)) +
  geom_bin2d(binwidth = 1, na.rm = TRUE) +
  xlim(50, 70) +
  ylim(50, 70)

p2 <- ggplot(diamonds, aes(table, depth, z = price)) +
  geom_raster(binwidth = 1, stat = "summary_2d", fun = mean,
              na.rm = TRUE) +
  xlim(50, 70) +
  ylim(50, 70)

grid.arrange(p1, p2, nrow = 2)
```



Statistical Summaries

Homework

Use the Chicago Data Set on Air Quality write a script to create two graph as follows:

1. Install and load "gridExtra" package
2. Add a variable called "year" to the dataset. *Hint: format(as.POSIXct(chicago$date),"%Y").* You have to re-attach dataset when you create new variables if you want to reference them.
3. Install and load "gridExtra" package
4. Create two bar charts using
   1. O3tmean2 as a function of year
   2. No2tmean2 as a function of year
5. Apply the "summary_bin" stat and define the y.fun as the mean
6. Plot the two graphs on the same plot with o3 stacked on top of the no2 graph.

Statistical Summaries

Homework

Using the actor_ages.csv dataset write a script to create the following plot:

1. Create line plots showing the relationship between actress ages as a function of actor ages.
2. Create one plot per actor and color code the plots using the default colors.
3. Create 4 rows of plots.
4. Which actors were consistently older than their leading ladies?
5. Analyzed these relationship as a function of film budget. Do film budget influence the gap in ages. Hint: Create a variable called "age_diff".
6. For which actors does budget have the greatest impact on age_diff?
7. Does race appear to influence the results? Why? Why not?

Statistical Summaries

Homework

Use the AutoClaims.csv dataset. This data set is from the Kaggle website and target variable of interest is total_claim_amount. Conduct an exploratory analysis using techniques from this chapter to determine if the following variables have potential to be predictive of total_claim_amount:

1. months_as_customer
2. insured_zip
3. insured_occupation
4. incident_location
5. auto_model
6. age
7. insured_sex
8. insured_hobbies
9. insured_education_level
10. witnesses

Unfortunately, the Kaggle site does not provide a data dictionary for this data set, buy the variables are pretty self explanatory.

Data Source: https://www.kaggle.com/buntyshah/auto-insurance-claims-data

GGPlot Toolbox of Functions

https://www.tidyverse.org/

R packages for data science

The tidyverse is an opinionated **collection of R packages** designed for data science. All packages share an underlying design philosophy, grammar, and data structures.

Install the complete tidyverse with:

```
install.packages("tidyverse")
```

Geom & Stat Reference
https://ggplot2.tidyverse.org/reference/