STAT 40001/STAT 50001 Statistical Computing

Lecture 5

Department of Mathematics and Statistics

Outline

- **Creating graphs in R**
- **Adding features in the graphs**
- **Saving R graphs**
- **grammer of graphics-ggplot2**

R Graphics

Producing high-quality graphics is one of the fundamental parts of any statistical computing. Graphics are often the starting point for statistical analysis. The particular plot function you need will depend on the number of variables you want to plot and the pattern you wish to highlight. One of the most attractive aspects of the R system is its capacity to produce state-of-the -art statistical graphics.

Four graphics systems in R

- Base graphics: Basic still powerful
- Grid graphics: Modules for building other tools
- Lattice graphics: General purpose for grid graphs
- ggplot2: The grammar of graphics

Common Graphs

Sometimes, if the reading doesn't function correctly, the error may stem from the text file itself. Below is a list of some of the most common problems

- plot(x,y) # scatter Plot
- boxplot(y)# Box Plot
- barplot(y)# Bar Plot
- pie(y)# Pie Chart
- hist(y)# Histogram
- stem(y)# Stem and Leaf Plot
- ts.plot(y)# Time Series Plot
- stripchart(y)#Dot Plot

Scatter Plots

The **plot** function draws axes and adds a scatter plot of points. Two extra functions, points and lines, add extra points or lines to an existing plot. There are two ways of specifying plot, points and lines and you should choose whichever you prefer: Customize graphs (line style, symbols, color, etc) can be drawn by specifying graphical parameters.

```
## Default S3 method:
plot(x, y = NULL, type = "p",  xlim = NULL, ylim = NULL,
     log = "", main = NULL, sub = NULL, xlab = NULL, ylab =
     ann = par("ann"), axes = TRUE, frame.plot = axes,
     panel.first = NULL, panel.last = NULL, asp = NA, ...)
```

Plotting a Graph

We can build up a graph in stages by issuing a series of commands

The Coordinate System

We want to establish the dimensions of the figure before plotting anything

The most important point but perhaps is obvious that both variable x and y must be of the same length.

Axes:

It is possible to turn off the axes, to adjust the coordinate space by using the *xlim* and *ylim* options

To create the labels for the axes. **axes=** Allows us to control whether the axes appear in the figure or not.

We may select **axes=F** and then create own labels using **xlim=, ylim=**

xlab="", ylab="" Creates labels for the x- and y-axis

Plot Types in R

We now want to plot these series, but the plot function allows for different types of plots. The different types that one can include within the generic plot function include:

```
"p" for points,
"l" for lines,
"b" for both,
"c" for the lines part alone of "b",
"o" for both overplotted,
"h" for histogram like (or high-density) vertical lines
"s" for stair steps,
"S" for other steps,
"n" for no plotting.
```

Styles in R

There are a number of options to adjust the style in the figure, including changes in the line type, line weight, color, point style, and more.

lty= Selects the type of line (solid, dashed, short-long dash, etc.)

lwd= Selects the line width (fat or skinny lines)

pch= Selects the plotting symbol, can either be a numbered symbol (pch=1) or a letter (pch="R")

col= Selects the color of the lines/points in the figure

Colors in R

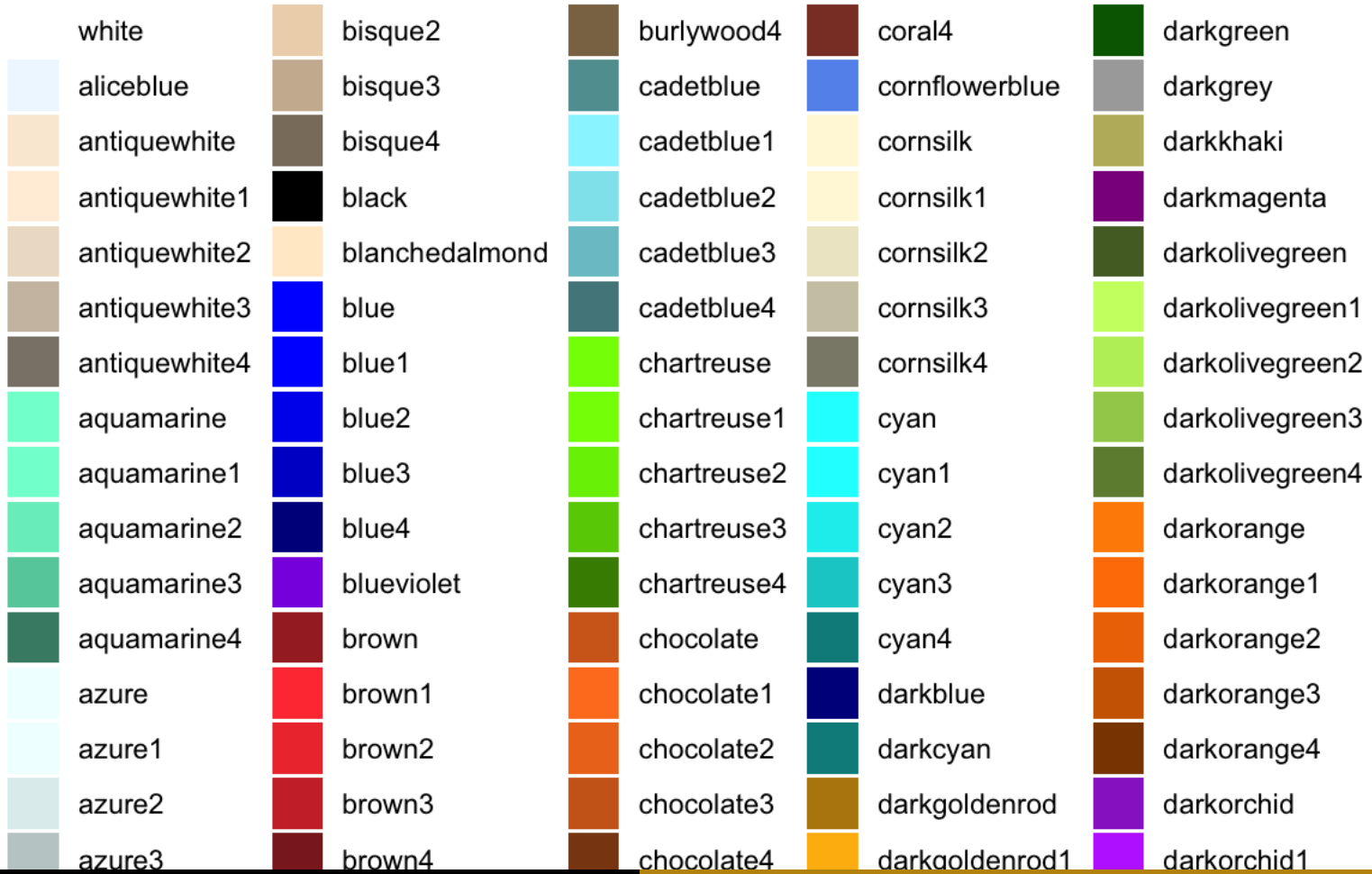
There exists different options to specify a color in R:

- Using numbers from 1 to 8, e.g. col = 1
- Specifying the color name, e.g. col = "blue",
- Using the HEX value of the color, e.g. col = "#0000FF"
- Using the RGB value making use of the rgb function, e.g. col = rgb(0, 0, 1).
- The "rgb" option also allows us to set the transparency of the color, if needed, with the alpha argument, which ranges from 0 (completely transparent) to 1.
- Using rainbow option , e.g. col=rainbow(7)

Awesome list of 657 different colors can be displayed using

```
> colors()
```

R colors


white
aliceblue
antiquewhite
antiquewhite1
antiquewhite2
antiquewhite3
antiquewhite4
aquamarine
aquamarine1
aquamarine2
aquamarine3
aquamarine4
azure
azure1
azure2
azure3
bisque2
bisque3
bisque4
black
blanchedalmond
blue
blue1
blue2
blue3
blue4
blueviolet
brown
brown1
brown2
brown3
brown4
burlywood4
cadetblue
cadetblue1
cadetblue2
cadetblue3
cadetblue4
chartreuse
chartreuse1
chartreuse2
chartreuse3
chartreuse4
chocolate
chocolate1
chocolate2
chocolate3
chocolate4
coral4
cornflowerblue
cornsilk
cornsilk1
cornsilk2
cornsilk3
cornsilk4
cyan
cyan1
cyan2
cyan3
cyan4
darkblue
darkcyan
darkgoldenrod
darkgoldenrod1
darkgreen
darkgrey
darkkhaki
darkmagenta
darkolivegreen
darkolivegreen1
darkolivegreen2
darkolivegreen3
darkolivegreen4
darkorange
darkorange1
darkorange2
darkorange3
darkorange4
darkorchid
darkorchid1

Examples

lty= Selects the type of line (solid, dashed, short-long dash, etc.)

```
>plot(c(0,1), c(0,0), type="l", axes=FALSE, xlab=NA, ylab=NA, lty=1)
```

lwd= Selects the line width (fat or skinny lines)

```
>plot(c(0,1), c(0,0), type="l", axes=FALSE, xlab=NA, ylab=NA, lwd=2)
```

pch= *Selects the plotting symbol, can either be a numbered symbol (pch=1) or a letter (pch="R")*

```
>plot(c(1,2,3), c(3,5,7), pch=5)
```

```
>plot(c(1,2,3), c(3,5,7), pch="A")
```

col= Selects the color of the lines/points in the figure

bg= 'background' color

col= color of lines and data symbols

col.axis= color of axis tick labels

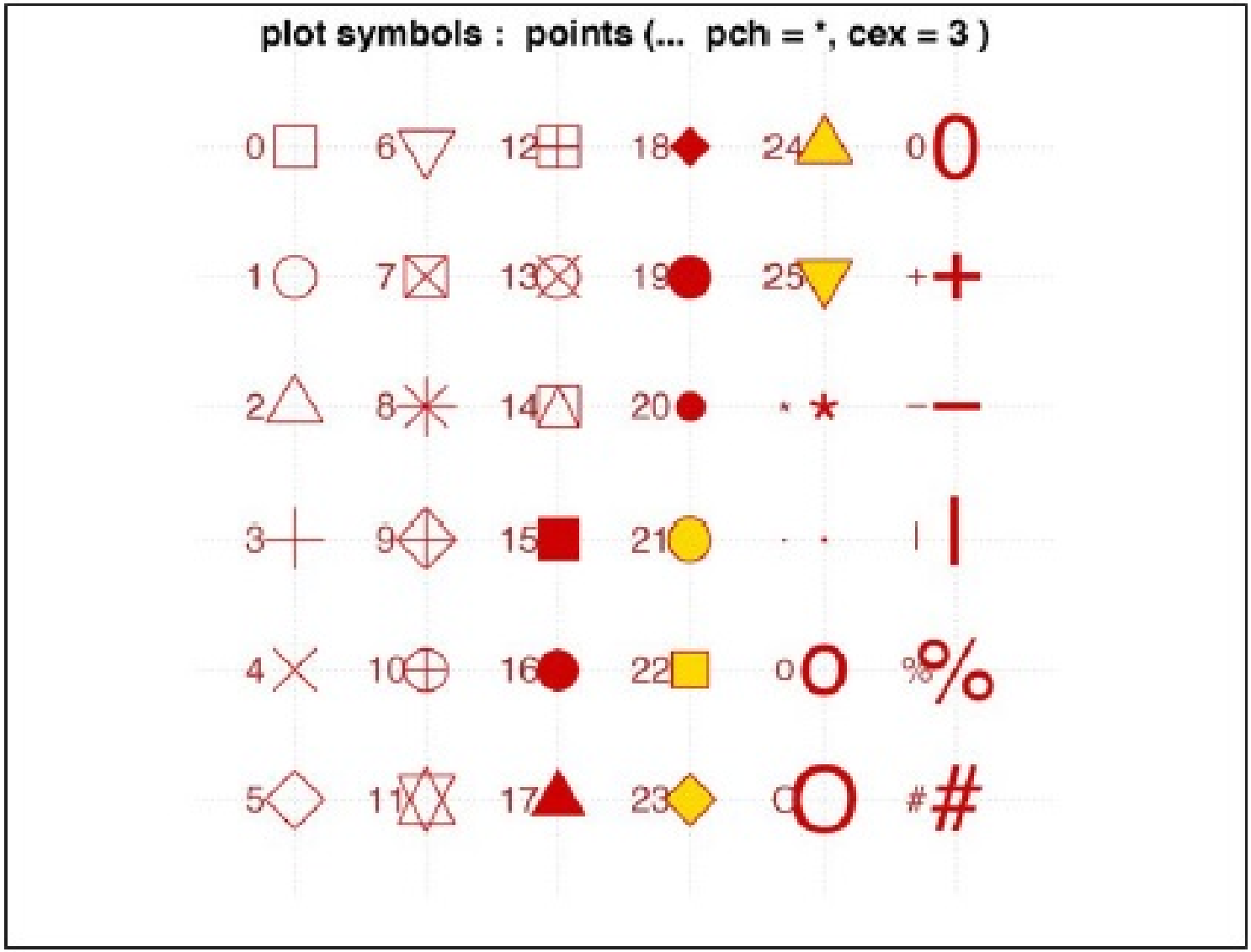
col.lab= color of axis labels

col.main= color of plot title

col.sub= color of plot sub-title

par(bg=4)
plot(c(1,2,3), c(3,5,7),col=2)

Plot Symbols



Graphic Parameters, *par()*

The function **par()** is used to set or get graphical parameters. **par** allows us to plot multiple (x, y)'s in a single graphic. This is accomplished by selecting par(new=T) following each call to plot.

```
x<-seq(-5,5,0.1)
y1<-dnorm(x)
y2<-dcauchy(x)
y3<-0.5*dexp(abs(x))
yrange<-range(y1,y2,y3)
plot(x,y1,xlab="x",ylab="f(x)",lty=1, type="l",xlim=c(-5,5),ylim=yrange,col=1)
par(new=TRUE)
plot(x,y2,xlab="",ylab="",lty=3,type="l",xlim=c(-5,5),ylim=yrange,col=2)
par(new=TRUE)
plot(x,y3,xlab="",ylab="",lty=2,type="l",xlim=c(-5,5),ylim=yrange,col=4)
legend(1,.5,legend=c("N(0,1)","C(0,1)","L(0,1)"),lty=c(1,3,2),col=c(1,2,4))
title(cex=1,"probability density functions of standard Normal, standard Cauchy and
\n standard Laplace distributions")
```

Add-on Functions in R Graphics

```
arrows(x1, y1, x2, y2): Create arrows within the plot
text(x1, x2, "text"): Create text within the plot
lines(x, y): Create a plot that connects lines
points(x, y): Create a plot of points
polygon(): Create a polygon of any shape (rectangles, triangles, etc.
legend(x, y, at = c("", "",), labels=c("", ""))): Create a legend to id
mtext( ):  Insert text in the figure and outer margins
title( ): Add figure title or outer title
abline( ): Add horizontal and vertical lines or a single line
box( ): Draw a box around the current plot
rect(): Draw a rectangle
segments(x0,y0,x1,y1): Draw line segments from (x0,y0)to(x1,y1)
trans3d(): Add 2-D components to a 3-D plot
main= : Overall title for the plot
sub= : A subtitle for the plot
```

Text and Symbol Size

The following options can be used to control text and symbol size in graphs.
cex: Number indicating the amount by which plotting text and symbols should be scaled relative to the default. cex=1 is default, cex=1.5 is 50% larger, cex=0.5 is 50% smaller, etc.
cex.axis magnification of axis annotation relative to cex
cex.lab magnification of x and y labels relative to cex
cex.main magnification of titles relative to cex
text(0,1,"R") places text at that location inside graph
text(0,1,expression(theta)) places symbol in that location in the graph
`plot(x,y, main=substitute(y==Psi*zeta-sum(beta^2,gamma)))`
place math title

Example:

Link below provides the number of Atlantic hurricane from 1870 to 2010
http://people.sc.fsu.edu/~jburkardt/datasets/time_series/hurricanes.txt
We will import the subject data and plot a graph

```
>hurricane<-"http://people.sc.fsu.edu/~jburkardt/datasets/time_series/hurricanes.txt"
>data=read.table(hurricane)
>x<-data$V1
>y<-data$V2
```

Example:

The Duncan data frame has 45 rows and 4 columns. Data on the prestige and other characteristics of 45 U. S. occupations in 1950. The data is in the library "car" we will access the data as below

```
> library(car)
> data(Duncan)
> attach(Duncan)
> head(Duncan, n=5)

> plot(education)
> plot(prestige)

> plot(education, prestige)
```

Arguments in R Graphics

```
> library(graphics);
>text(x,y,"mytext") #text in plot at the coordinate (x, y)
>mtext("mytext",side=4)
#gives text in margins of plot with side choices
#(1=bottom, 2=left, 3=top, 4=right)

>library(ISLR)#  baseball hitters salaries etc. 1987-88
>h1=hist(Hitters$HmRun)
>h1
>pie(h1$counts)
>names(h1$counts)=h1$breaks[1:8]
>pie(h1$counts);
> title("Baseball Hitter Home Run counts")

library(plotrix)# this allows 3D pie charts
pie3D(h1$counts,explode=0.1, theta=pi/3)
```

How to draw more than one plot

One way to display several graphics at a time is using the option mfrow as follows. The option 'mar' refers to margin in the sequence: bottom, left, up, and right. Try this:

```
par(mfrow=c(2,2), mar=c(4,4,4,4))
plot(AirPassengers)
plot(log(AirPassengers))
plot(diff(AirPassengers))
plot(diff(AirPassengers, lag=12))
```

Saving Graphs

Since R runs on so many different operating systems, and supports so many different graphics formats, it's not surprising that there are a variety of ways of saving your plots, depending on what operating system you are using, what you plan to do with the graph, and whether you're connecting locally or remotely.

Format	Driver
JPG	jpeg
PNG	png
WMF	win.metafile
PDF	pdf
Postscript	postscript

Viewing Several Graphs

Creating a new graph by issuing a high level plotting command (plot, hist, boxplot, etc.) will typically overwrite a previous graph. To avoid this, open a new graph window before creating a new graph. To open a new graph window use one of the functions below.

windows()

Graphics with ggplot2

The ggplot2 package, offers a powerful graphics language for creating elegant and complex plots.ggplot2 is based on the grammar of graphics , the idea that you can build every graph from the same few components: a data set, a set of geoms—visual marks that represent data points, and a coordinate
Use install.packages()

```
> install.packages(''ggplot2'')
```

Please explore the `plotly` package.
Making plots is a very repeatetive: draw this line, add these colored points, then add these, etc. Instead of re-using the same code over and over, ggplot implements them using a high-level but very expressive API. The result is less time spent creating your charts, and more time interpreting what they mean.
If you're planning on using ggplot, it's best to keep your data in DataFrames. Think of a DataFrame as a tabular data object. Example, diamonds dataset in "ggplot2".

ggplot2

ggplot function accepts two arguments:

- Data set to be plotted
- Aesthetic mappings provided by aes function

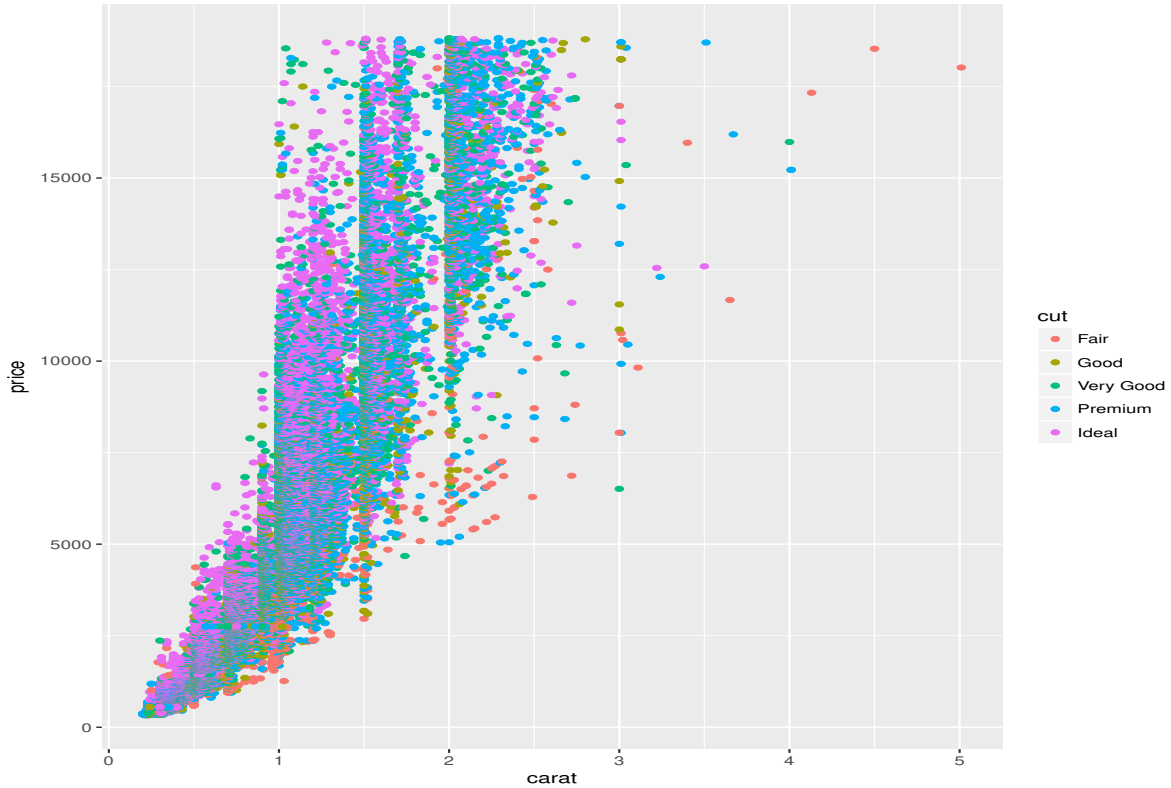
Aesthetics describe how your data will relate to your plots. Some common aesthetics are: x, y, and color.
Aesthetics are specific to the type of plot (or layer) you're adding to your visual. For example, a scatterplot (geompoint) and a line (geomline) will share x and y, but only a line chart has a linetype aesthetic.

ggplot-syntex

```
Noun:  Data
     - ggplot(data = df)

Verb: ‘‘geom_" + Plot Type
 - ggplot(data = df) + geom_bar()

Adjectives:  Aesthetics (‘‘aes") (x, y, fill, colour, linetype)
     - ggplot(data = df,aes(x=categorical.var, fill=group.var))
     + geom_bar()

Adverb: stat (e.g. identity, bin)
     -ggplot(data = df, aes(x=categorical.var, fill=group.var))
       + geom_bar(stat = "bin")

Preposition :  position (e.g. fill, dodge, identity)
    -ggplot(data = df, aes(x=categorical.var, fill=group.var)) +
        geom_bar(stat="bin", position = "identity", binwidth=5)
```

ggplot - Diamond Data

```
> library(ggplot2)
> data (diamonds)
> ggplot(diamonds, aes(x=carat, y=price, color=cut))
  +geom_point()
```



Data Visualization

```
grade<-c(12,20,10,3,5)
names(grade)=c("A", "B", "C", "D", "F")
barplot(grade,col=c("1","2","3","4","5"), main="Grade Distribution")
```

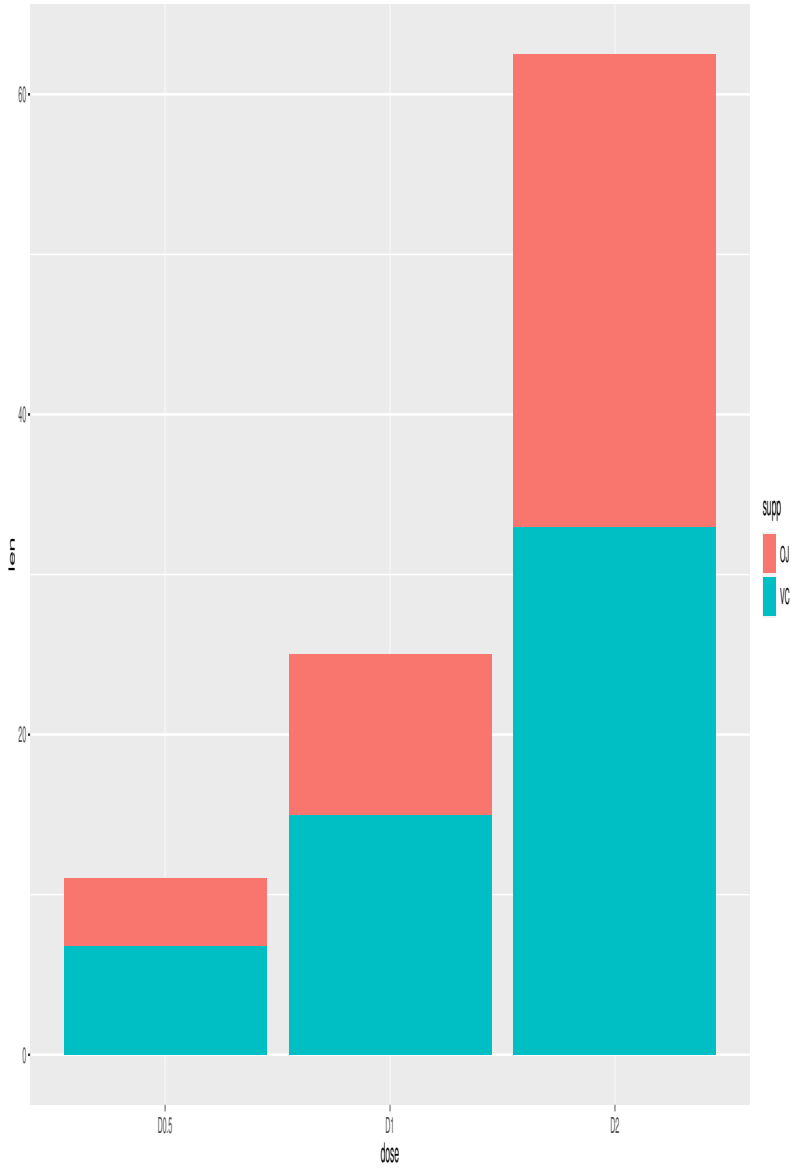
Using ggplot

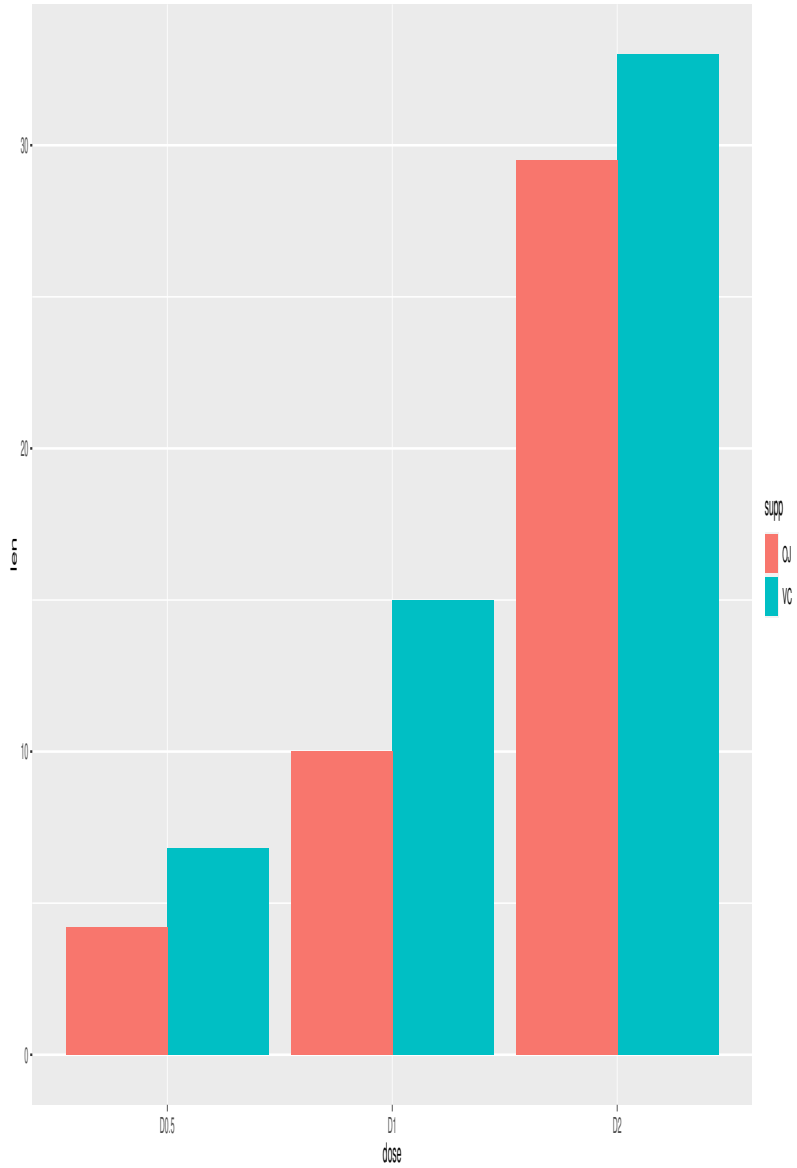
```
> library(ggplot2)
> grade=c("A", "B", "C", "D", "F")
> freq=c(12,20,10,3,5)
> bardata=data.frame(grade,freq)
> attach(bardata)
> ggplot(data=bardata,aes(x=grade,y=freq))
  +geom_bar(stat="identity", width=0.5)
> ggplot(data=bardata,aes(x=grade,y=freq))
  +geom_bar(stat="identity", fill=c(1,2,3,4,5))
```

Bar Graphs

```
library(ggplot2)
df <- data.frame(supp=rep(c("VC", "OJ"), each=3),
                 dose=rep(c("D0.5", "D1", "D2"),2),
                 len=c(6.8, 15, 33, 4.2, 10, 29.5))
head(df)
ggplot(data=df, aes(x=dose, y=len, fill=supp)) +
  geom_bar(stat="identity")
# Use position=position_dodge()
ggplot(data=df, aes(x=dose, y=len, fill=supp)) +
geom_bar(stat="identity", position=position_dodge())
```

Bar Graphs


60
40
20
0
len
D0.5
D1
D2
dose
supp
OJ
VC

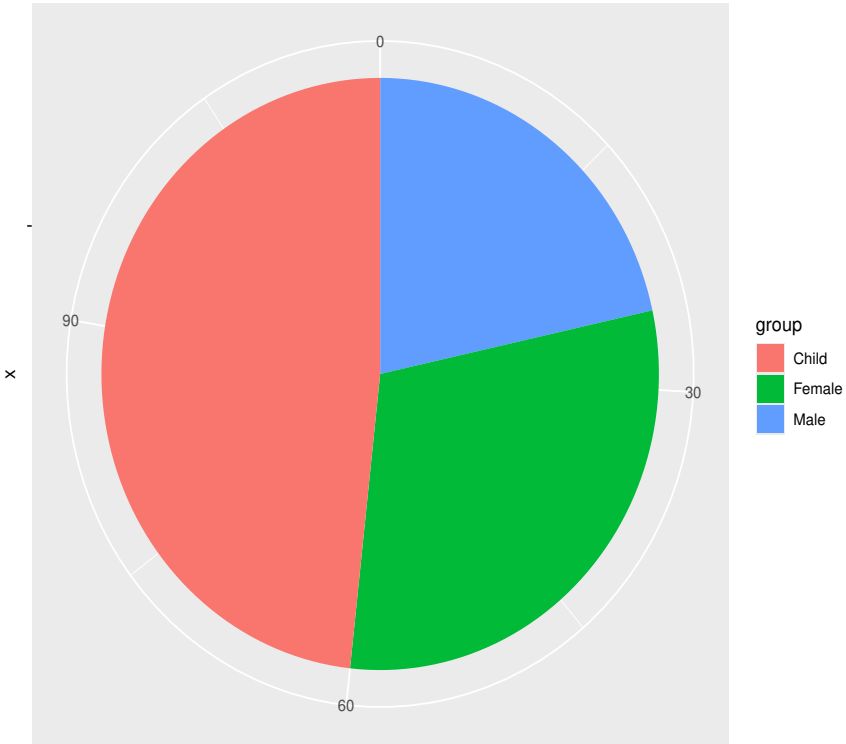

30
20
10
0
len
D0.5
D1
D2
dose
supp
OJ
VC

Pie Chart

```
students<-c(120, 200, 350, 230,125)
names(students)=c("Maths","Sci","Eng","Edu","Tech")
pie(students,col=c("1","2","3","4","5"))
######
library(plotrix) # To create a 3D plot
pie3D(students)
pie3D(students, height = 0.2,labels = students)

lab <- paste0(round(students/sum(students) * 100, 2), "%")
pie3D(students, height = 0.2,labels = lab)
pie3D(students, height = 0.2,labels = lab, explode=0.2)
```

Pie chart

```
library(ggplot2)
group=c("Male", "Female", "Child")
freq=c(25, 35, 56)
df=data.frame(group,freq)
df
ggplot(df, aes(x="", y=freq, fill=group))+
geom_bar(width = 1,stat ="identity")+ coord_polar("y",start=0)
```



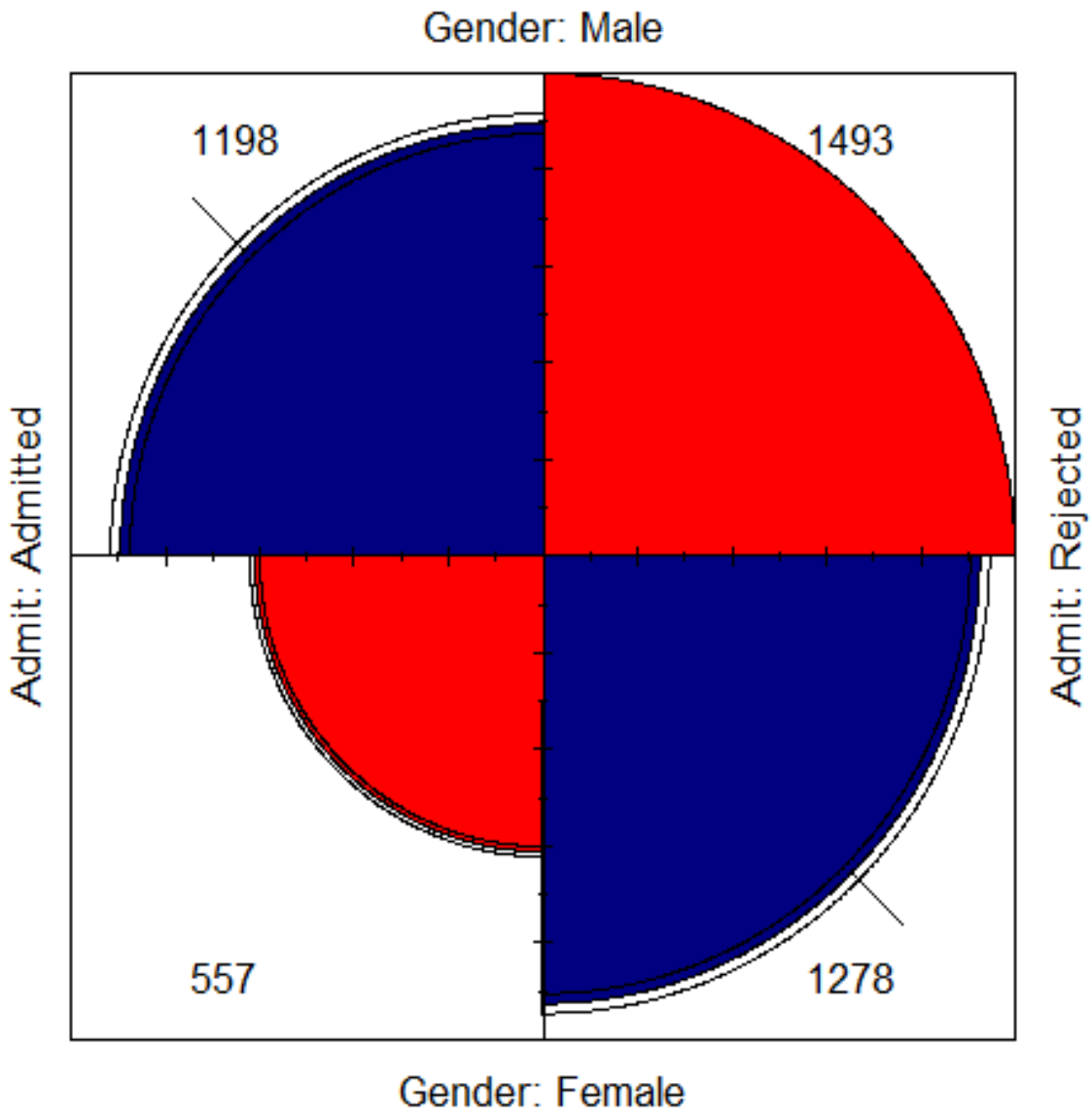
Mosaic Plot

```
data(HairEyeColor)
library(graphics) #requires(graphics) also works
## Full mosaic
mosaicplot(HairEyeColor)
mosaicplot(HairEyeColor, col=c(1,2))
## Aggregate over sex
newdata <- apply(HairEyeColor, c(1, 2), sum)
newdata
mosaicplot(newdata, col=c(1,2,3,4))
```

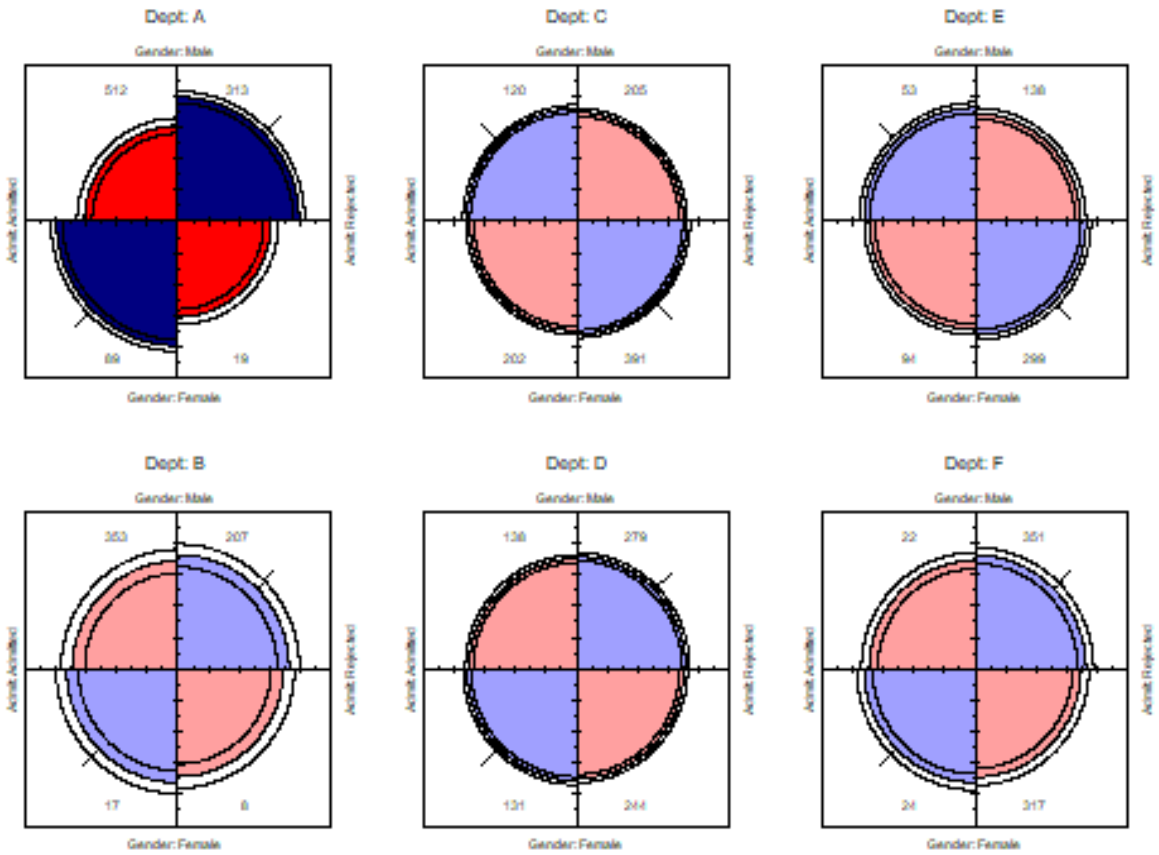
Multi-fold Plot

```
> library(vcd)
> data("UCBAdmissions")
> UCB <- aperm(UCBAdmissions, c(2,1,3))
#marginal table, collapsing over Dept
> UCB2 <- margin.table(UCB, c(1,2))
> fourfold(UCB2)
##
> fourfold(UCB2, std="ind.max")
> fourfold(UCB, mfrow=c(2,3))
```

Fourfold graphs



Fourfold graphs

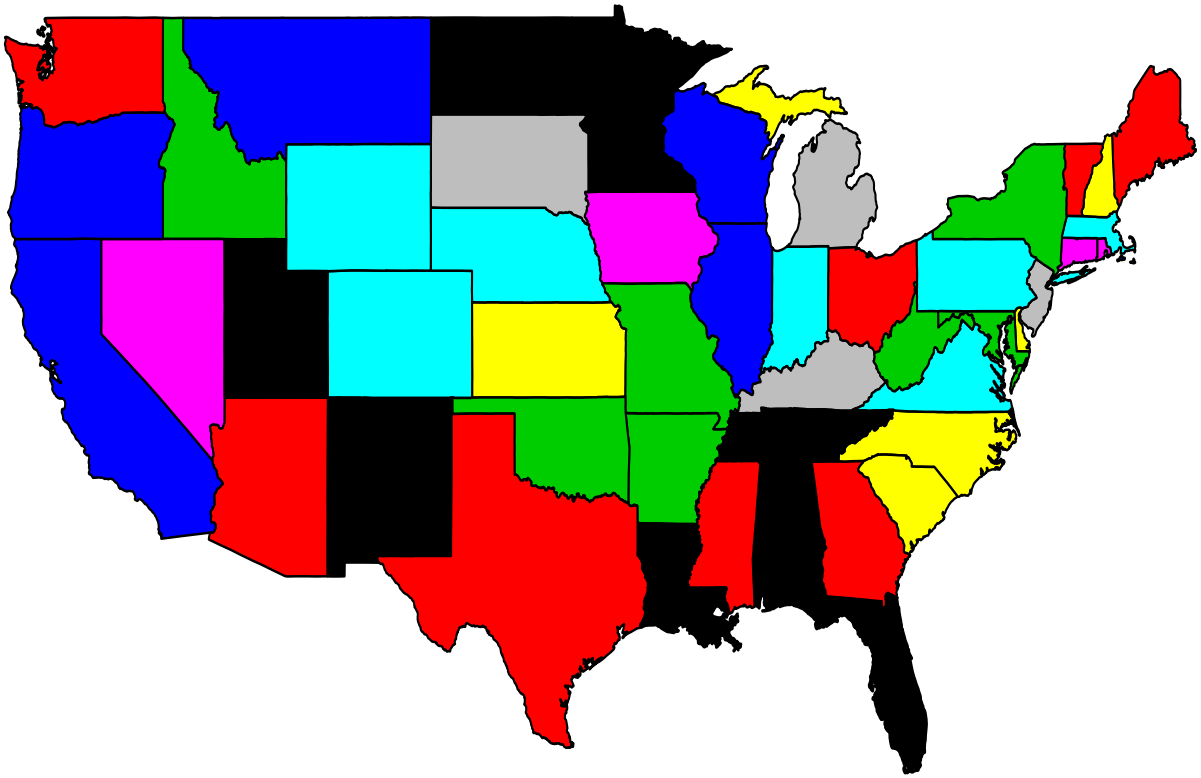


Creating Maps

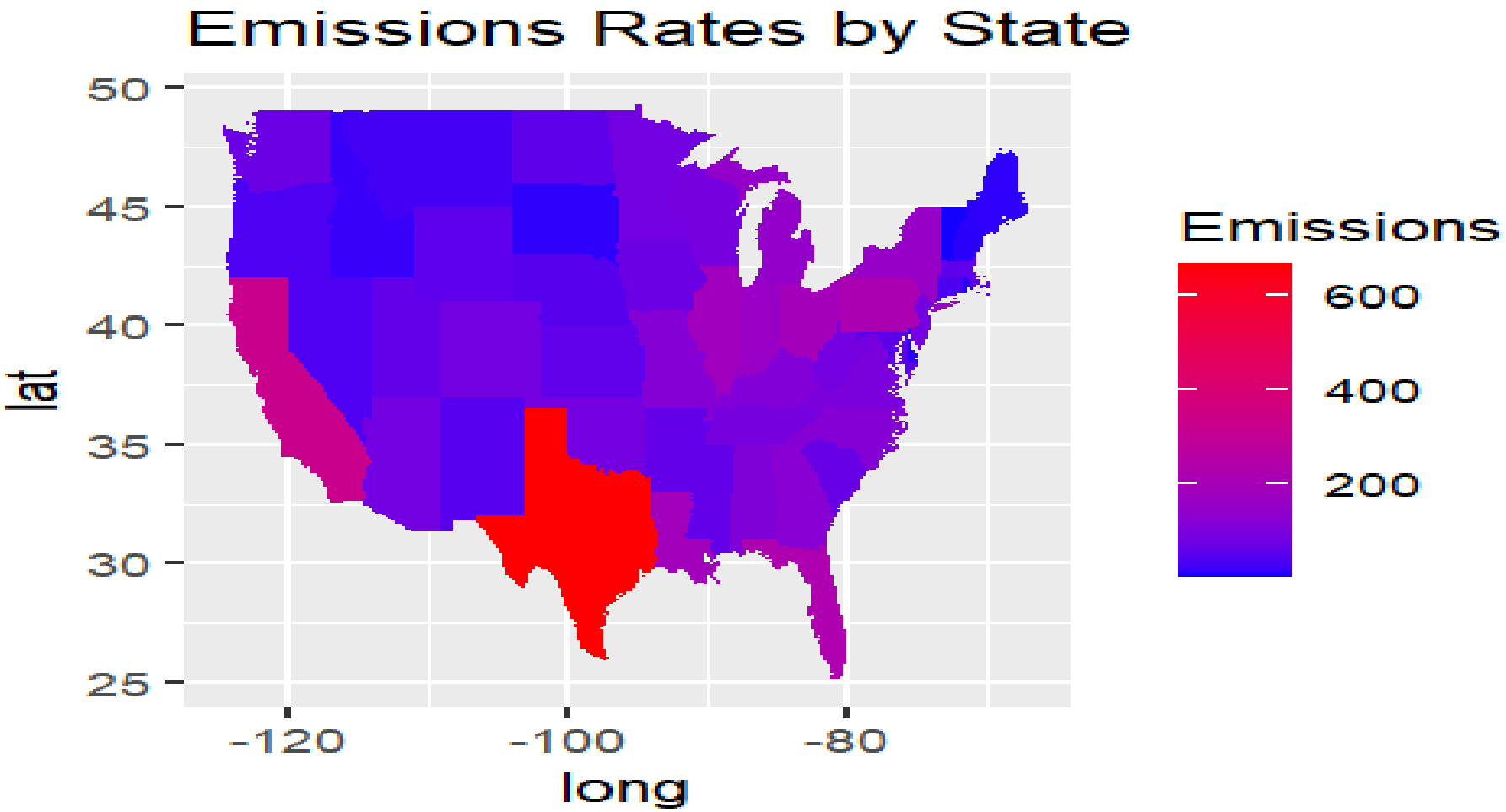
We can retrieve map data from the *maps* package and draw it with `geom_polygon()` (which can have a color fill) or `geom_path()` (which can't have a fill). By default, the latitude and longitude will be drawn on a Cartesian coordinate plane, but you can use `coord_map()` and specify a projection. The default projection is "mercator", which, unlike the Cartesian plane, has a progressively changing spacing for latitude lines

```
> library(maps)
> world_map <- map("world")
> map("world", wrap=c(0,360))
> states_map <- map("state")
> states_map <- map("state", regions="Indiana")
#  three USA databases (usa, state, county)
> states_map <- map("state", fill=TRUE, col=1:50)
> map("world", "China")
> map("world", "India")
> map("world", "Nepal")
```

Maps- USA mainland



Maps- Carbon emission (in megatons) by states



```
library(maps)
library(mapproj)
library(tidyverse)

# Define the emissions data

Z <- c(Alabama = 108.4, Alaska = 38.9, Arizona = 83.0, Arkansas = 62.0, California = 324.0,
Colorado = 85.4,Connecticut = 36.6,Delaware = 13.0, "District of Columbia" = 2.5, Florida = 226.3,
  Georgia = 124.1, Hawaii = 17.3, Idaho = 20.5, Illinois = 184.2, Indiana = 166.4, Iowa = 73.1,
  Kansas = 59.8, Kentucky = 111.3, Louisiana = 188.6, Maine = 14.4, Maryland = 52.6,
  Massachusetts = 56.1, Michigan = 147.8, Minnesota = 83.2, Mississippi = 63.1, Missouri = 117.0,
  Montana = 28.5, Nebraska = 47.2, Nevada = 39.4,  "New Hampshire" = 13.3, "New Jersey" = 89.1,
  "New Mexico" = 45.9, "New York" = 156.0, "North Carolina" = 115.6, "North Dakota" = 56.5,
  Ohio = 194.0, Oklahoma = 87.8, Oregon = 38.5, Pennsylvania = 213.5, "Rhode Island" = 10.6,
  "South Carolina" = 69.3, "South Dakota" = 15.2, Tennessee = 92.7, Texas = 663.5,
  Utah = 62.1, Vermont = 5.6, Virginia = 98.0, Washington = 73.8,
  "West Virginia" = 88.4, Wisconsin = 92.5, Wyoming = 54.6)
```

```
# Combine emissions data with state names
> emissions_data <-data.frame(region = tolower(names(Z)),emissions = Z)

# Get state map data
states <- map_data("state")

# Merge emissions data with state map data
states <- merge(states, emissions_data, by = "region", all.x = TRUE)

# Plot
ggplot(states, aes(long, lat, group = group, fill = emissions)) +
geom_polygon() + labs(title = "Emissions Rates by State")+
scale_fill_gradient(name="Emissions in Megatons",low="blue",high="red")
```