Graphics

James M. Flegal

Agenda

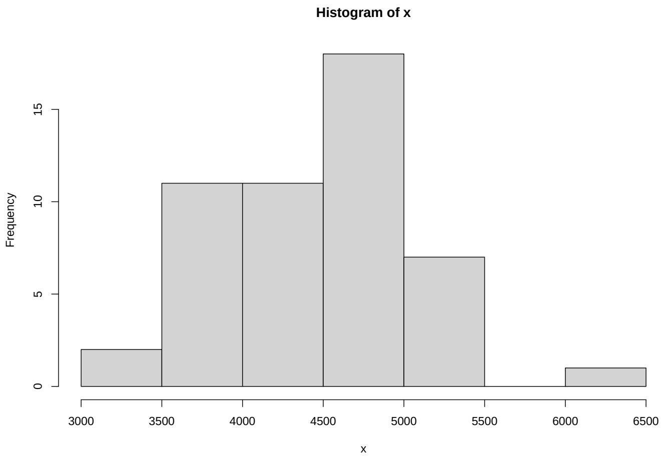
- High-level graphics
- Custom graphics
- Layered graphics in `ggplot2`

Functions for graphics

- The functions `hist()`, `boxplot()`, `plot()`, `points()`, `lines()`, `text()`, `mtext()`, `axis()`, etc. form a suite that plot graphs and add features to the graph
- Each of these functions have various options, to learn more about them, use the help
- `par()` can be used to set or query graphical parameters

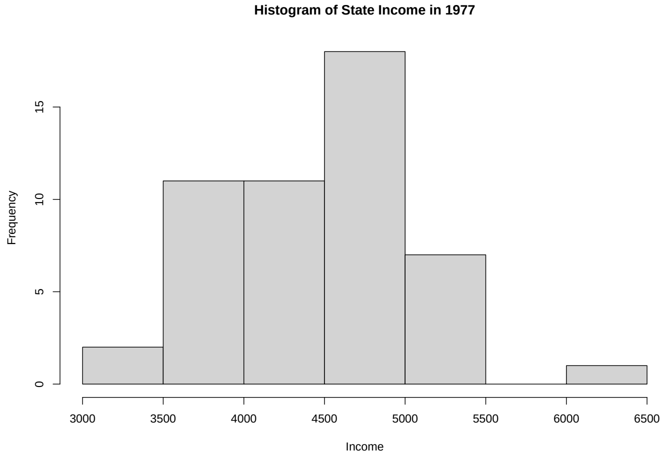
Univariate data: Histogram

```r
x = state.x77[ , 2]                        # 50 average state incomes in 1977
hist(x)
```



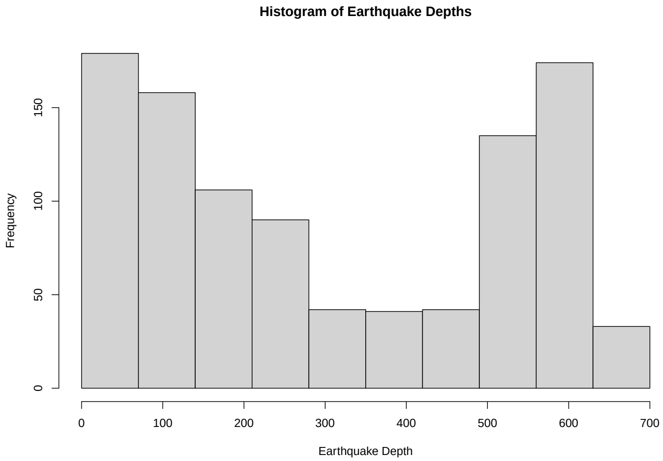
Univariate data: Histogram

```r
hist(x, breaks = 8, xlab="Income", main="Histogram of State Income in 1977")
```



Univariate data: Histogram

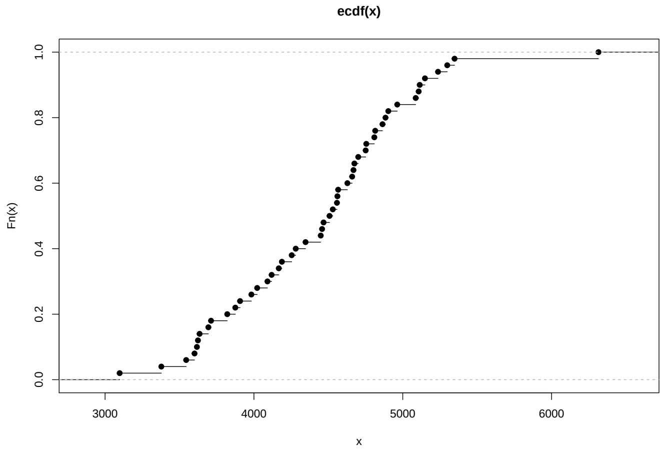
```r
y = quakes$depth                               # 1000 earthquake depths
hist(y, seq(0, 700, by = 70), xlab="Earthquake Depth", main="Histogram of Earthquake Depths")
```



Empirical CDF

Function `ecdf()` provides data for empirical cdf

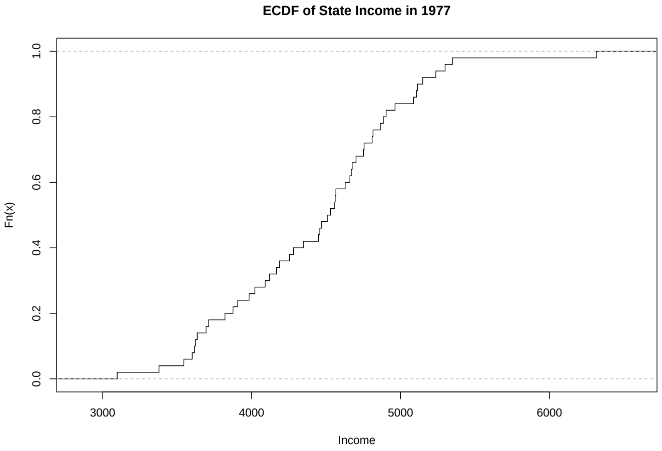
```r
plot.ecdf(x)
```



Empirical CDF

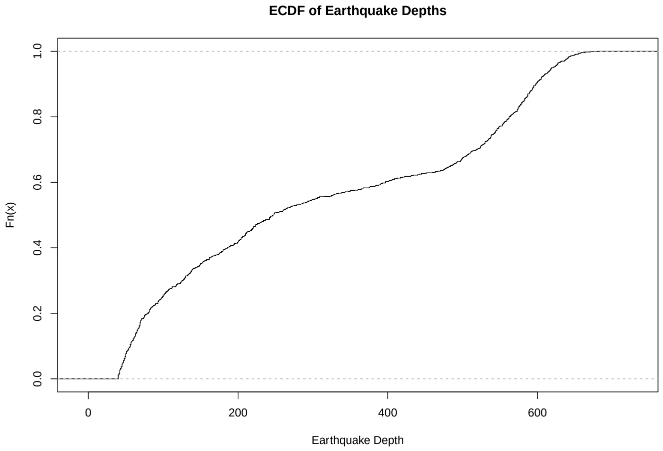
Can add vertical lines and remove dots

```
plot.ecdf(x, verticals = T, pch = "", xlab="Income", main="ECDF of State Income in 1977")
```



Empirical CDF

```
plot.ecdf(y, verticals = T, pch = "", xlab="Earthquake Depth",
          main="ECDF of Earthquake Depths")
```

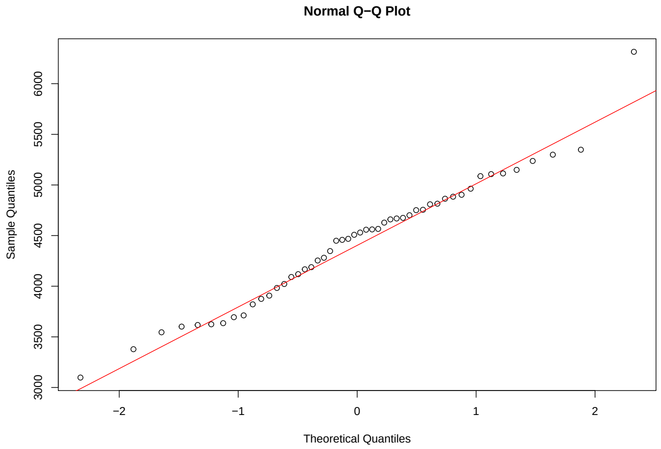


qqnorm() and qqplot()

- ‘qqnorm() plots the quantiles of a data set against the quantiles of a Normal distribution
- ‘qqplot() plots the quantiles of a first data set against the quantiles of a second data set

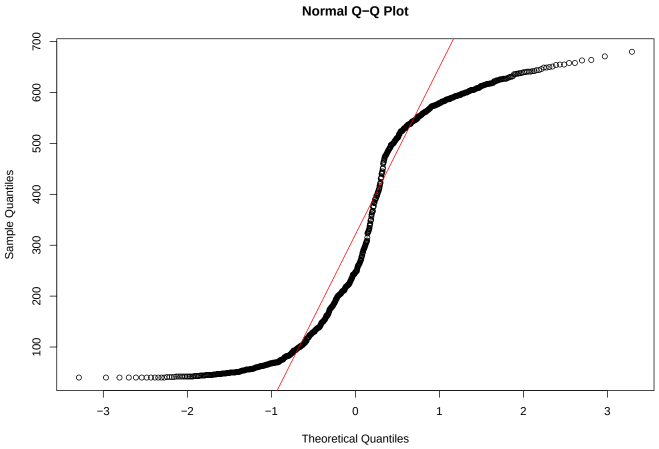
qqnorm() and qqplot()

```r
qqnorm(x)                                # qq plot for the earthquake depths
qqline(x, col = "red")                   # red reference line
```



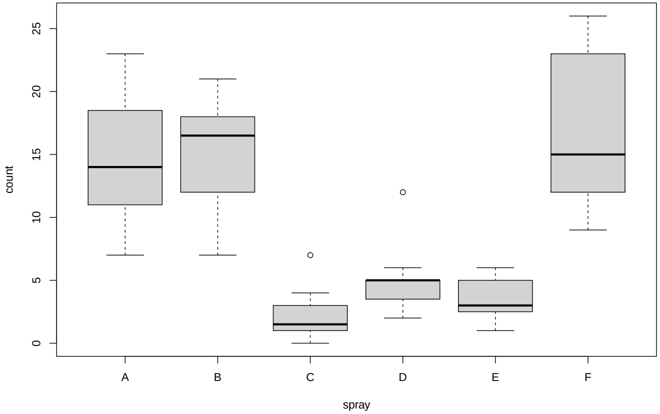
qqnorm() and qqplot()

```r
qqnorm(y)                            # qq plot for the earthquake depths
qqline(y, col = "red")               # red reference line
```



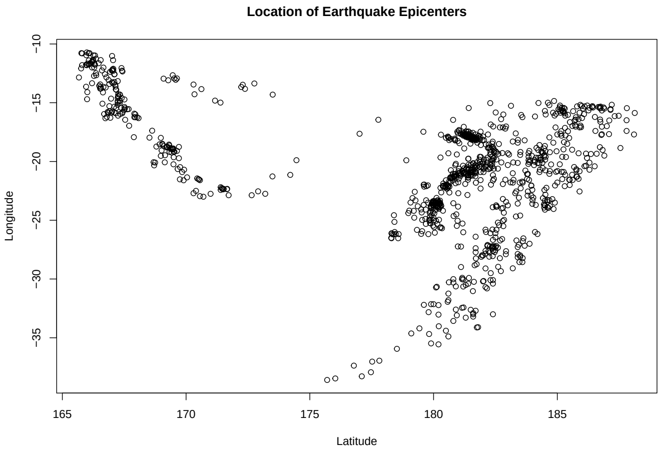
Box plots

```r
boxplot(count ~ spray, data = InsectSprays)
```



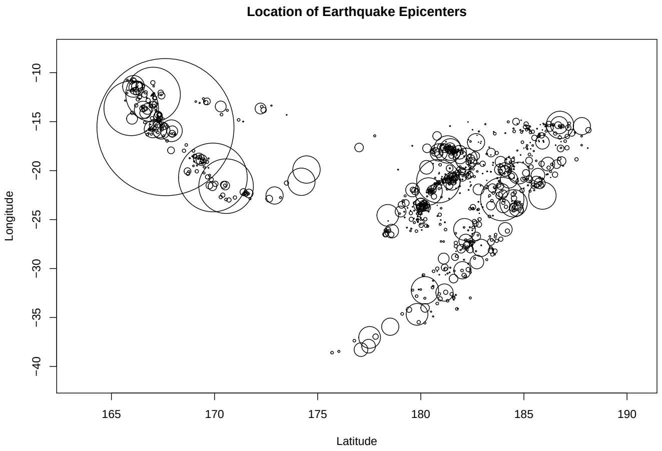
Scatterplots: `plot(x, y)`

```r
plot(quakes$long, quakes$lat, xlab="Latitude", ylab="Longitude",
     main="Location of Earthquake Epicenters")
```



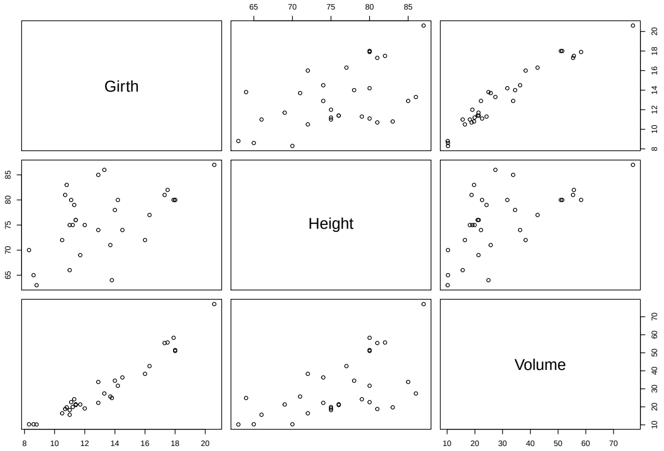
Scatterplots: `plot(x, y)`

```r
symbols(quakes$long, quakes$lat, circles = 10 ^ quakes$mag, xlab="Latitude",
        ylab="Longitude", main="Location of Earthquake Epicenters")
```



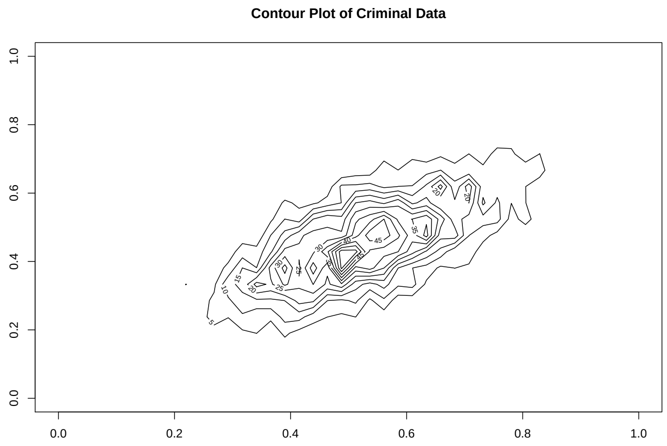
Three-dimensional data: `pairs(x)`

```
pairs(trees)
```



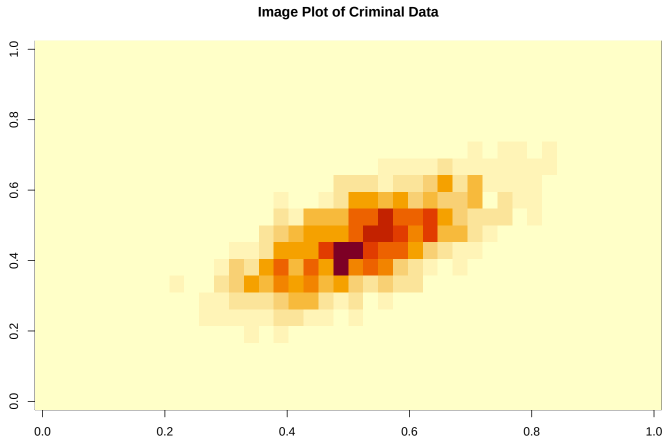
Three dimensional plots

```
contour(crimtab, main="Contour Plot of Criminal Data")
```



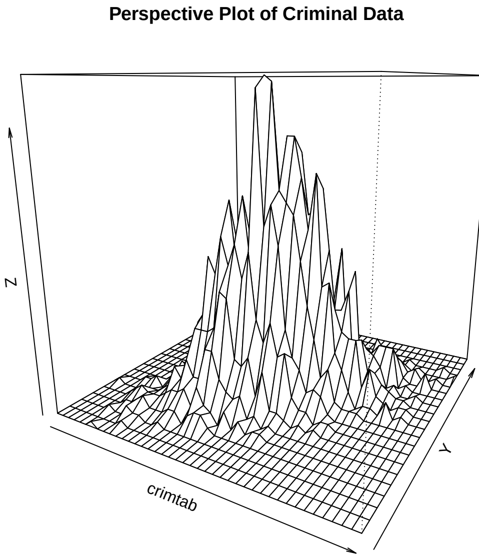
Three dimensional plots

```r
image(crimtab, main="Image Plot of Criminal Data")
```



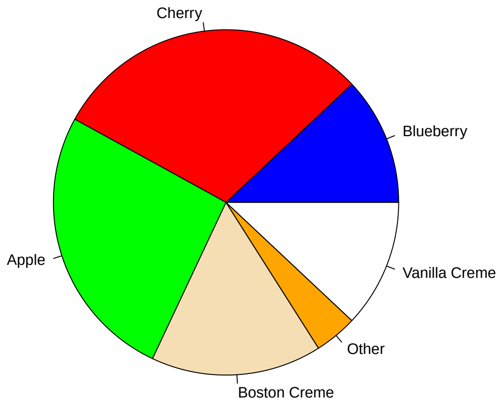
Three dimensional plots

```
persp(crimtab, theta=30, main="Perspective Plot of Criminal Data")
```



Categorical data: Pie charts

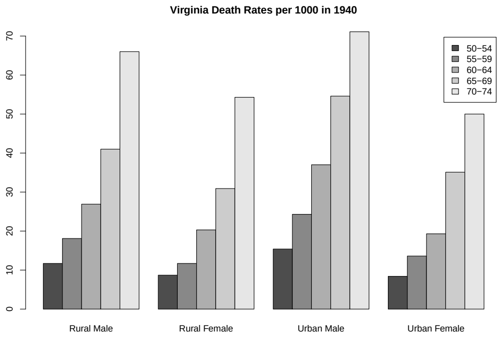
```r
pie.sales = c(0.12, 0.30, 0.26, 0.16, 0.04, 0.12)
names(pie.sales) = c("Blueberry", "Cherry", "Apple", "Boston Creme",
                        "Other", "Vanilla Creme")
pie(pie.sales, col = c("blue", "red", "green", "wheat", "orange", "white"))
```



Categorical data: Pie charts

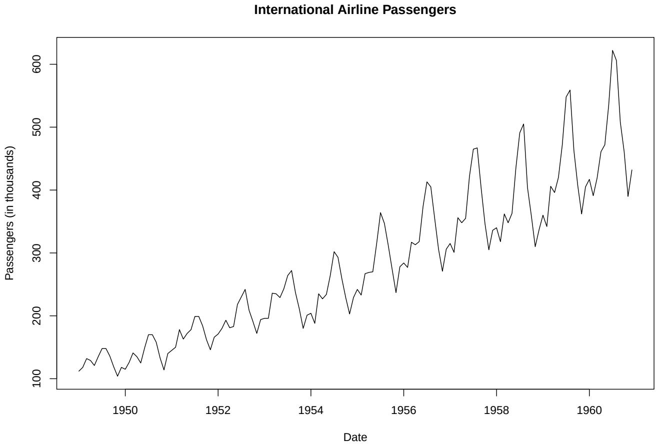
dotchart() and barplot() also available

```
barplot(VADeaths, beside = T, legend = T,
        main = "Virginia Death Rates per 1000 in 1940")
```



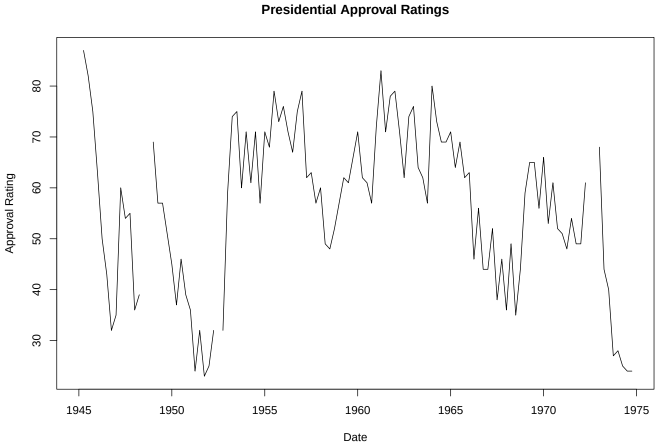
Time series plots

```r
ts.plot(AirPassengers, xlab="Date", ylab="Passengers (in thousands)"
        , main="International Airline Passengers")
```



Time series plots

```r
ts.plot(presidents, xlab="Date", ylab="Approval Rating"
        , main="Presidential Approval Ratings")
```



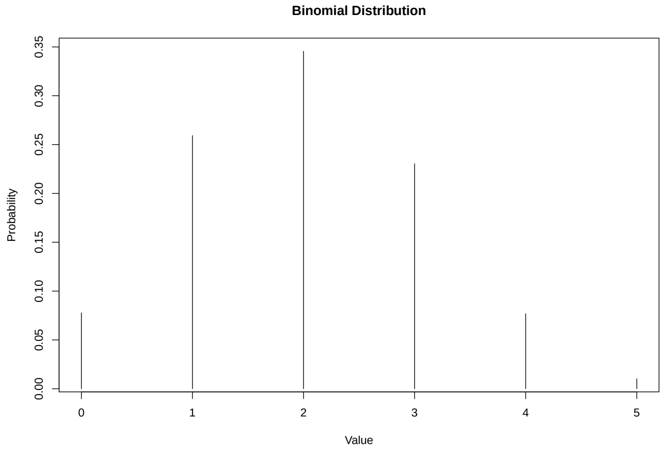
Custom graphics

- `par()` can be used to set or query graphical parameters
- We've already used some of these
  - `adj`: text justification
  - `bg`: background color
  - `col`, `col.axis`, `col.lab`, `...`: color specification
  - `lty`: line type, e.g. dashed, dotted, solid (default), longdash, `...`
  - `lwd`: line width (helpful to increase for presentation plots)
  - `mfcol` and `mfrow`: subsequent figures will be drawn in an nr-by-nc array on the device
  - `pch`: point types
  - `xlog`: plots to log scale if `TRUE`
  - `...`

Binomial distribution

Plot of binomial distribution with $n = 5$ and $p = .4$

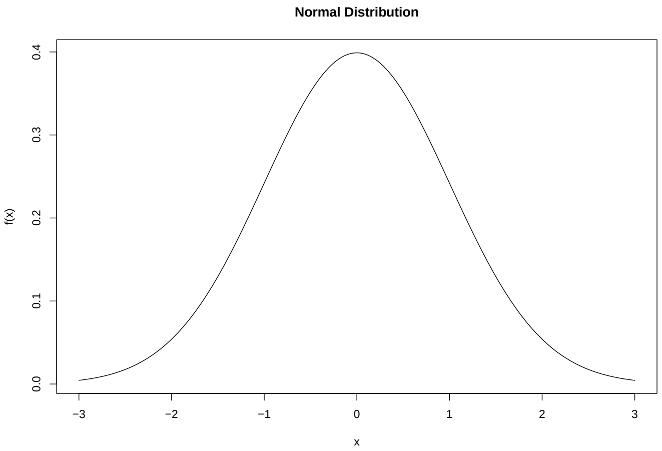
```r
x = 0:5
y = dbinom(x, 5, 2 / 5)
plot(x, y, type = "h", main="Binomial Distribution", xlab="Value", ylab="Probability")
```



Normal distribution

Probability density function for the standard Normal distribution from -3 to 3

```r
x = seq(-3, 3, by = 0.01)
y = dnorm(x)
plot(x, y, type = "l", main="Normal Distribution", ylab="f(x)")
```

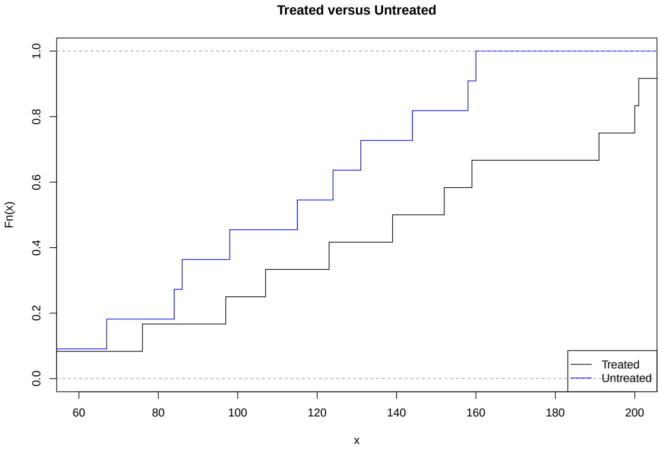


Two empirical cdfs: `Puromycin` dataset

```
x = Puromycin$rate[Puromycin$state == "treated"]
y = Puromycin$rate[Puromycin$state == "untreated"]
```

Two empirical cdfs: `Puromycin` dataset

```r
plot.ecdf(x, verticals = TRUE, pch = "", xlim = c(60, 200), main="Treated versus Untreated")
lines(ecdf(y), verticals = TRUE, pch = "", xlim = c(60, 200), col="blue")
legend("bottomright", c("Treated", "Untreated"), pch = "", col=c("black", "blue"), lwd = 1)
```



Saving a plot to a file

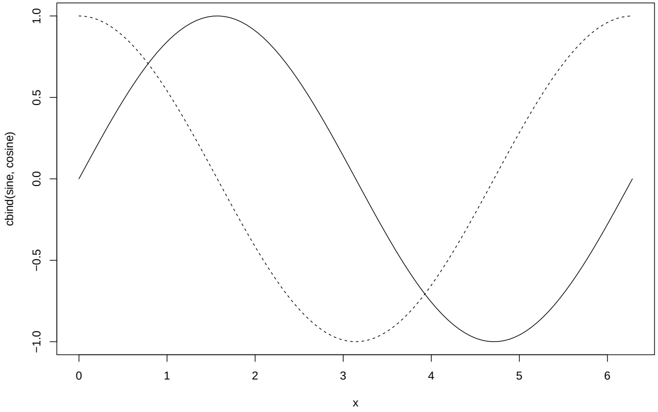
- Begin with functions `postscript()`, `pdf()`, `tiff()`, `jpeg()`, ...
- ... put all your plotting commands here ...
- Finish with `dev.off()`

```
pdf("2cdfs.pdf", width=6, height=4)
plot.ecdf(x, verticals = TRUE, pch = "", xlim = c(60, 200), main="Treated versus Untreated")
lines(ecdf(y), verticals = TRUE, pch = "", xlim = c(60, 200), col="blue")
legend("bottomright", c("Treated", "Untreated"), pch = "", col=c("black", "blue"), lwd = 1)
dev.off()
```

```
## pdf
##   2
```

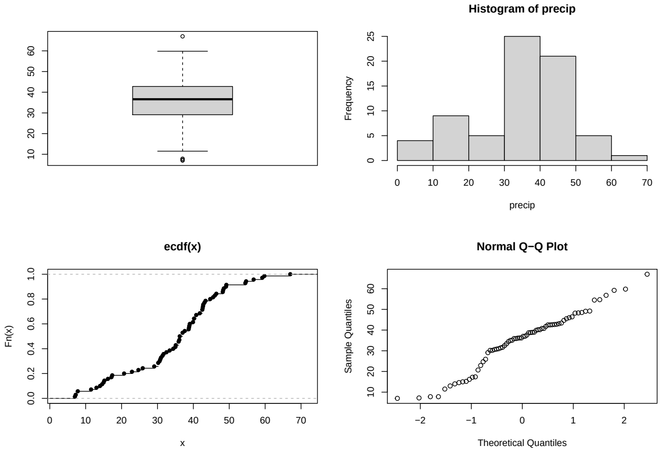
Multiple plots on one set of axes

```r
x = seq(0, 2 * pi, length = 100)
sine = sin(x)
cosine = cos(x)
matplot(x, cbind(sine, cosine), col = c(1, 1), type = "l")
```



Multiple frame plots

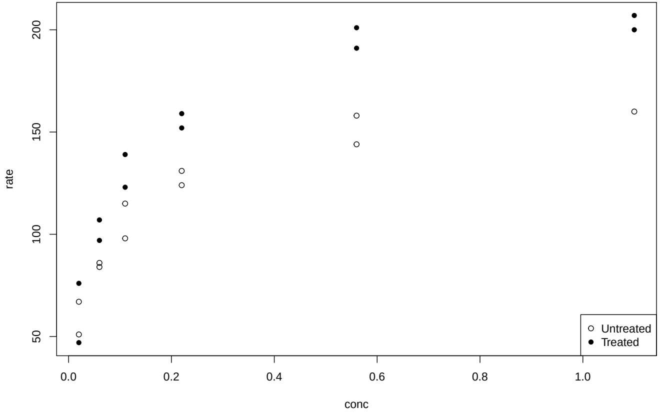
```r
par(mfrow = c(2, 2))
boxplot(precip)
hist(precip)
plot.ecdf(precip)
qqnorm(precip)
```



```r
par(mfrow = c(1, 1))
```

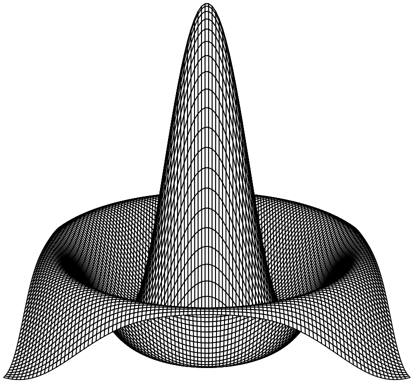
Plot using statistical model

```r
plot(rate ~ conc, data = Puromycin, pch = 15 * (state == "treated") + 1)
legend("bottomright", legend = c("Untreated", "Treated"), pch = c(1, 16))
```

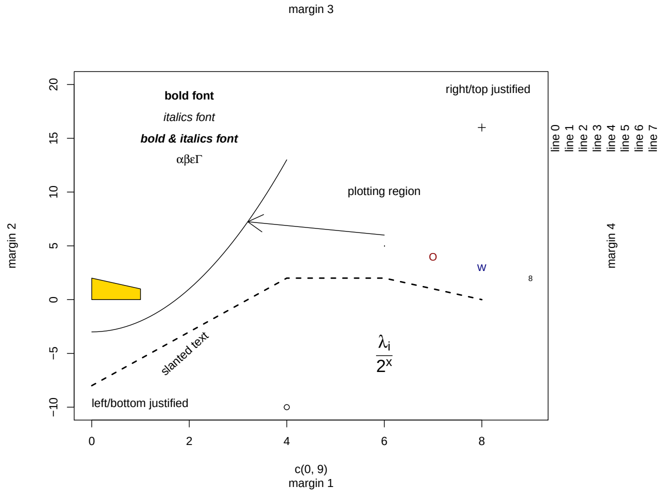


Plot using `persp()` for wire mesh

```
x = seq(-8, 8, length = 100)
y = x
f = function(x, y) sin(sqrt(x ^ 2 + y ^ 2)) / (sqrt (x ^ 2 + y ^ 2))
z = outer(x, y, f)
persp(x, y, z, xlab = "", ylab = "", zlab = "", axes = F, box = F)
```



Custom plot: Leemis Chapter 21



ggplot2

- Everything so far has been part of base R
- The `ggplot2` package is a popular by Hadley Wickham
- Based on the grammar of graphics, which tries to take the good parts of `base` and `lattice` graphics and none of the bad parts
- http://ggplot2.org

Forced Expiratory Volume (FEV) Data

Explore a data on the relationship between smoking and pulmonary function from Rosner (1999) using layered graphics created with `ggplot2`. The data consists of a sample of 654 youths, aged 3 to 19, in the area of East Boston during middle to late 1970's. Our main interest is in the relationship between smoking and FEV. Use the following commands to load the data and `ggplot2`.

```
load(url("http://www.faculty.ucr.edu/~jflegal/fev.RData"))
library(ggplot2)
```

```
## Warning: package 'ggplot2' was built under R version 4.0.2
```

```
str(fevdata)
```

```
## 'data.frame':    654 obs. of  5 variables:
##  $ age   : int  9 8 7 9 9 8 6 6 8 9 ...
##  $ fev   : num  1.71 1.72 1.72 1.56 1.9 ...
##  $ height: num  57 67.5 54.5 53 57 61 58 56 58.5 60 ...
##  $ sex   : Factor w/ 2 levels "female","male": 1 1 1 2 2 1 1 1 1 1 ...
##  $ smoke : Factor w/ 2 levels "nonsmoker","smoker": 1 1 1 1 1 1 1 1 1 1 ...
```

Layered graphics in `ggplot2`

`ggplot2` allows you to construct multi-layered graphics. A plot in `ggplot2` consists of several components:

- Defaults
- Layers
- Scales
- Coordinate system

Layers consist of:

- Data
- Mapping
- Geom
- Stat
- Position

`ggplot2` uses the + operator to build up a plot from these components. The basic plot definition looks like this:

```
ggplot(data, mapping) +
layer(
  stat = "",
  geom = "",
  position = "",
  geom_parms = list(),
  stat_params = list(),
)
```

We usually won't write out the full specification of layer, but use shortcuts like:

```
geom_point()
stat_summary()
```

Every geom has a default stat and every stat has a default geom.

- Usually, data and mappings are the same for all layers and so they can be stored as defaults:

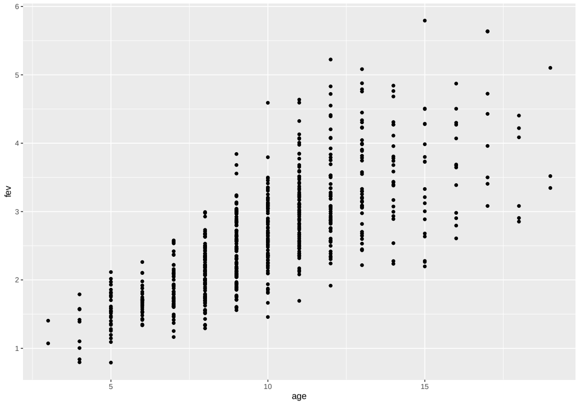
```
ggplot(data, mapping = aes(x = x, y = y))
```

- All layers use the default values of data and mapping unless explicitly you override them explicitly. The `aes()` function describes the mapping that will be used for each layer. You must specify a default, but you can also specify per layer mappings and data:

```
ggplot(data, mapping(aes(x = x, y = y)))
  + geom_point(aes(color = z))
  + geom_line(data = another_data)
```

- Scatterplot of age versus `fev`:

```r
s <- ggplot(fevdata, aes(x = age, y = fev))
s + geom_point()
```



- What do you see?
- Try coloring the points according to sex or smoker at home. Are there any apparent differences?

Layering

- Can add additional layers to a plot with the + operator.
- Let's try adding a line that shows the average value of `fev` for each age.
- One way to do this is to construct an additional data frame with columns corresponding to age and average value of `fev` and then add a layer with this data.
- Do this with the `dplyr` package. Don't worry about how it works for now. First, you will need to install `dplyr` if it is not already installed:

```r
install.packages("dplyr")
```

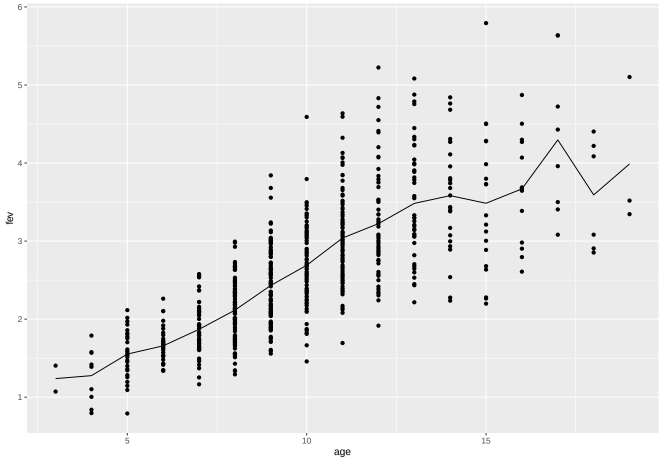
```r
library(dplyr)
```

```
## Warning: package 'dplyr' was built under R version 4.0.2

##
## Attaching package: 'dplyr'

## The following objects are masked from 'package:stats':
##
##     filter, lag

## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

```
fev_mean <- summarize(group_by(fevdata, age), fev = mean(fev))
```

```
## `summarise()` ungrouping output (override with `.groups` argument)
```

```
s + geom_point() + geom_line(data = fev_mean)
```



Smoothing

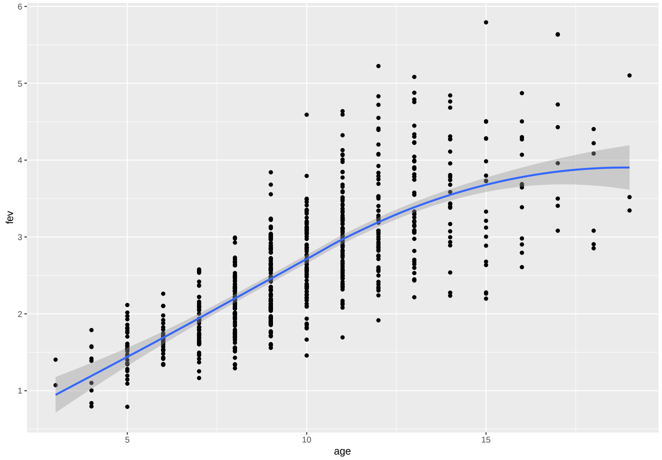
- Similarly, we can add a smoother to the scatterplot by first computing the smooth and storing it in data frame. Then add a layer with that data. Since smoothers are so useful, this operation is available in `ggplot2` as a stat.
- `stat_smooth()` provides a smoothing transformation. It creates a new data frame with the values of the smooth and by default uses `geom="ribbon"` so that both the smooth curve and error bands are shown.

```
s + geom_point() + stat_smooth()
```

- The default smoother is loess. This is the name given to the locally weighted quadratic regression smoother with tricubic weight function. Its bandwidth can be specified indirectly with the span parameter.

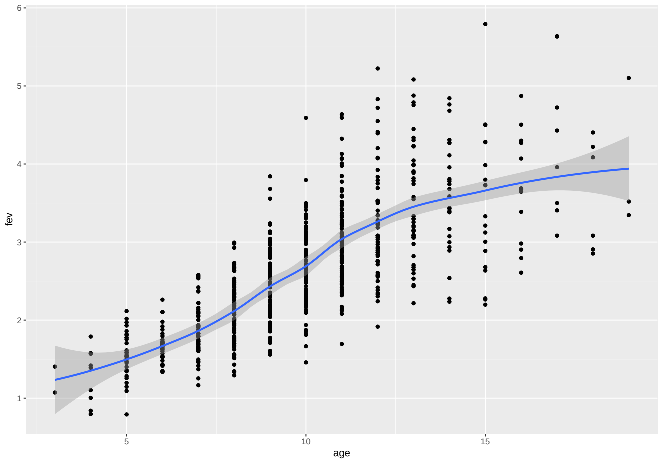
```r
s + geom_point() + stat_smooth(span = 1)
```

```
## `geom_smooth()` using method = 'loess' and formula 'y ~ x'
```



```r
s + geom_point() + stat_smooth(span = 1/2)
```

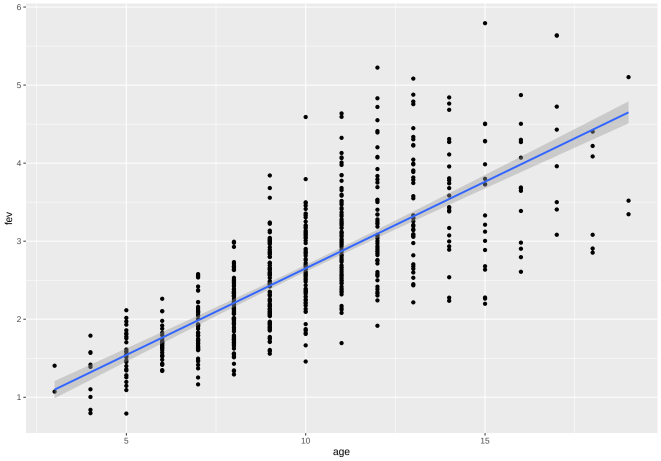
```
## `geom_smooth()` using method = 'loess' and formula 'y ~ x'
```



- We could also use linear regression by specifying `lm`:

```r
s + geom_point() + stat_smooth(method = 'lm')
```

```
## `geom_smooth()` using formula 'y ~ x'
```

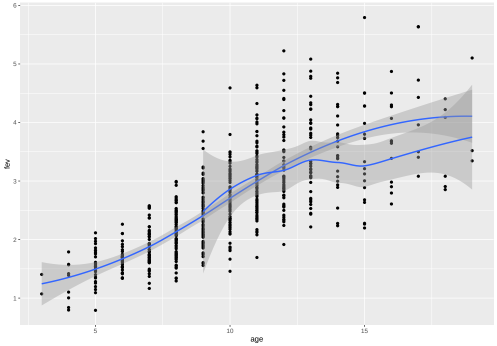


Grouping

- Clearly, we can see `age` and `fev` are highly correlated. What else is correlated with `age` and `fev`?
- How can we compare the relationship between `age` and `fev` among smokers and non-smokers? One way is to use two separates smoothers: one for smokers and one for non-smokers.
- Can do this using the group aesthetic. It specifies that we wish to group the data according by some variable before layering.

```r
p <- ggplot(fevdata, aes(x = age, y = fev, group = smoke))
p + geom_point() + stat_smooth()
```

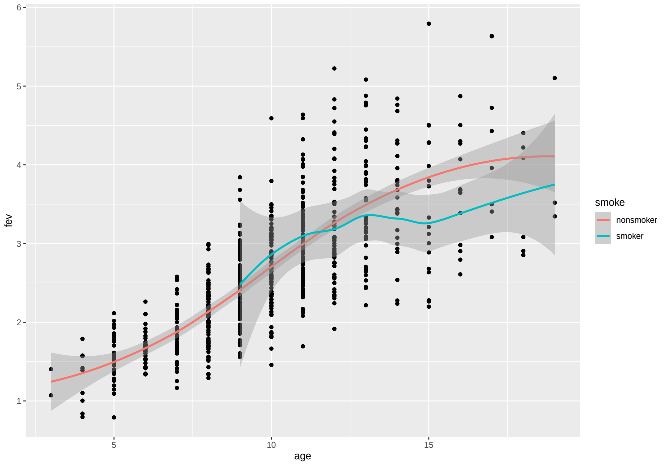
```
## `geom_smooth()` using method = 'loess' and formula 'y ~ x'
```



- Problem with this plot is that we can't tell which group each curve corresponds to.

```
p + geom_point() + stat_smooth(aes(color = smoke))
```

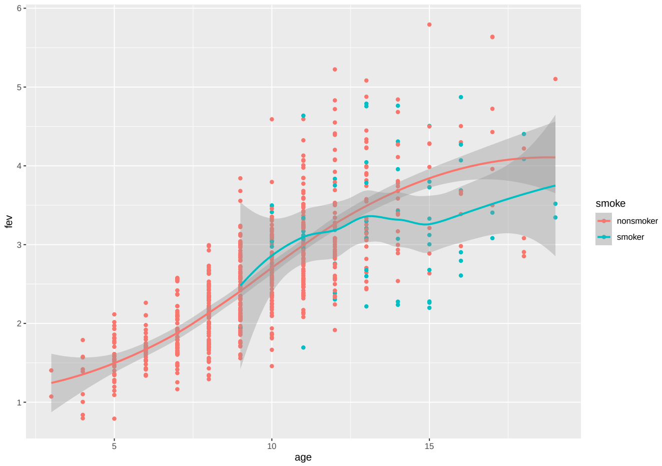
```
## `geom_smooth()` using method = 'loess' and formula 'y ~ x'
```



- Or for points and smooths

```r
p <- ggplot(fevdata, aes(x = age, y = fev, group = smoke, color = smoke))
p + geom_point() + stat_smooth()
```

```
## `geom_smooth()` using method = 'loess' and formula 'y ~ x'
```

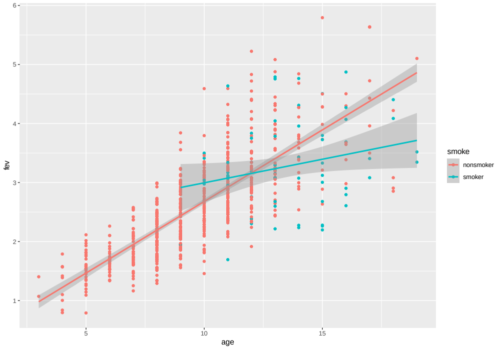


- What do these plots suggest?
- What can't be seen in these plots?
- Could there be lurking confounders?

- How is the following plot different from the others in terms of the conclusions you might draw about the relation between `smoke` and `fev`?

```
p + geom_point() + stat_smooth(method = 'lm')
```

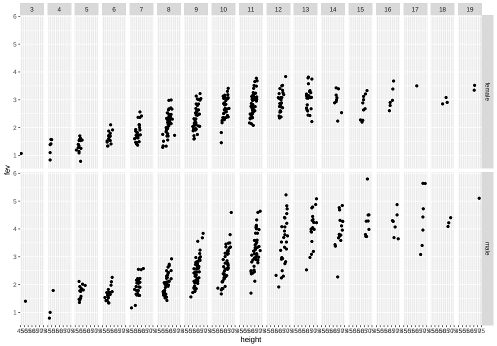
```
## `geom_smooth()` using formula 'y ~ x'
```



Faceting

- Faceting is a technique for constructing multiple subplots that show different subsets of data.
- `ggplot2` has two ways to facet: `facet_wrap` and `facet_grid`.
    - Grid faceting allows you use specify up to two factor variables: one for rows and one for columns of the grid.
    - Wrap faceting allows you to specify one factor variable. We can use faceting to examine the relationship between `fev` and `height` for different combinations of `age` and `sex`. This will allow us to view four variables simultaneously.

```
p <- ggplot(fevdata, aes(x = height, y = fev)) + facet_grid(sex ~ age)
p + geom_point()
```

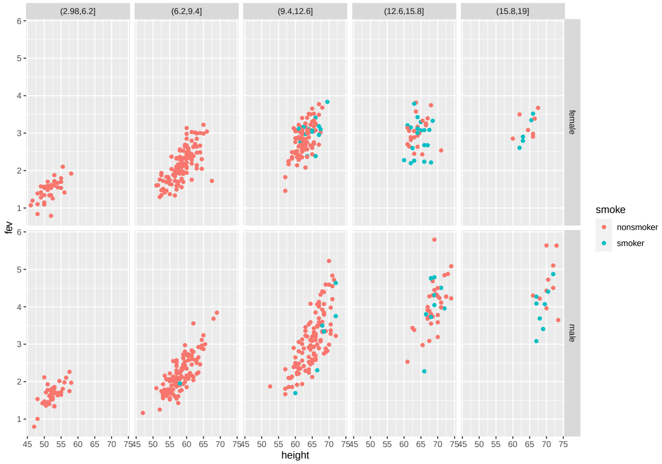


- One problem with the previous plot is that there are too many ages and relatively few observations at each age.
- Can instead try dividing age into a smaller number of groups using the `cut()` function. `cut()` creates a new factor variable by cutting its input. Here we cut age into 5 intervals of equal length:

```r
fevdata <- transform(fevdata, age_group = cut(age, breaks = 5))
```

- Then make new plots

```r
p <- ggplot(fevdata, aes(x = height, y = fev)) + facet_grid(sex ~ age_group)
p + geom_point(aes(color = smoke))
```



Summary

- R has strong graphic capabilities
- Graphing is an iterative process; Don't rely on the default options
- Avoid gimmicks, use the minimum amount of ink to get your point across
- A small table can be better than a large graph
- Carefully consider the size and shape of your graph, bigger is not always better