

Date Science with R

Statistical models

Peng Zhang

School of Mathematical Sciences, Zhejiang University

2025/06/30

Agenda

- Using data frames for statistical purposes
- Manipulation of data into more convenient forms
- (Re-)Introduction to linear models and the model space

So You've Got A Data Frame. What can we do with it?

- Plot it: examine multiple variables and distributions
- Test it: compare groups of individuals to each other
- Check it: does it conform to what we'd like for our needs?

Test Case: Birth weight data

Included in R already:

```
library(tidyverse)
library(lubridate)
library(MASS)
data(birthwt)
summary(birthwt)
```

	low	age	lwt	race
## Min.	:0.0000	Min. :14.00	Min. : 80.0	Min. :1.000
## 1st Qu.	:0.0000	1st Qu.:19.00	1st Qu.:110.0	1st Qu.:1.000
## Median	:0.0000	Median :23.00	Median :121.0	Median :1.000
## Mean	:0.3122	Mean :23.24	Mean :129.8	Mean :1.847
## 3rd Qu.	:1.0000	3rd Qu.:26.00	3rd Qu.:140.0	3rd Qu.:3.000
## Max.	:1.0000	Max. :45.00	Max. :250.0	Max. :3.000
	smoke	ptl	ht	ui
## Min.	:0.0000	Min. :0.0000	Min. :0.00000	Min. :0.0000
## 1st Qu.	:0.0000	1st Qu.:0.0000	1st Qu.:0.00000	1st Qu.:0.0000
## Median	:0.0000	Median :0.0000	Median :0.00000	Median :0.0000
## Mean	:0.3915	Mean :0.1958	Mean :0.06349	Mean :0.1481
## 3rd Qu.	:1.0000	3rd Qu.:0.0000	3rd Qu.:0.00000	3rd Qu.:0.0000
## Max.	:1.0000	Max. :3.0000	Max. :1.00000	Max. :1.0000
	ftv	bwt		
## Min	:0.0000	Min. : 709		

Make it readable!

```
colnames(birthwt)

## [1] "low"     "age"      "lwt"      "race"     "smoke"    "ptl"      "ht"       "ui"       "ftv"
## [10] "bwt"

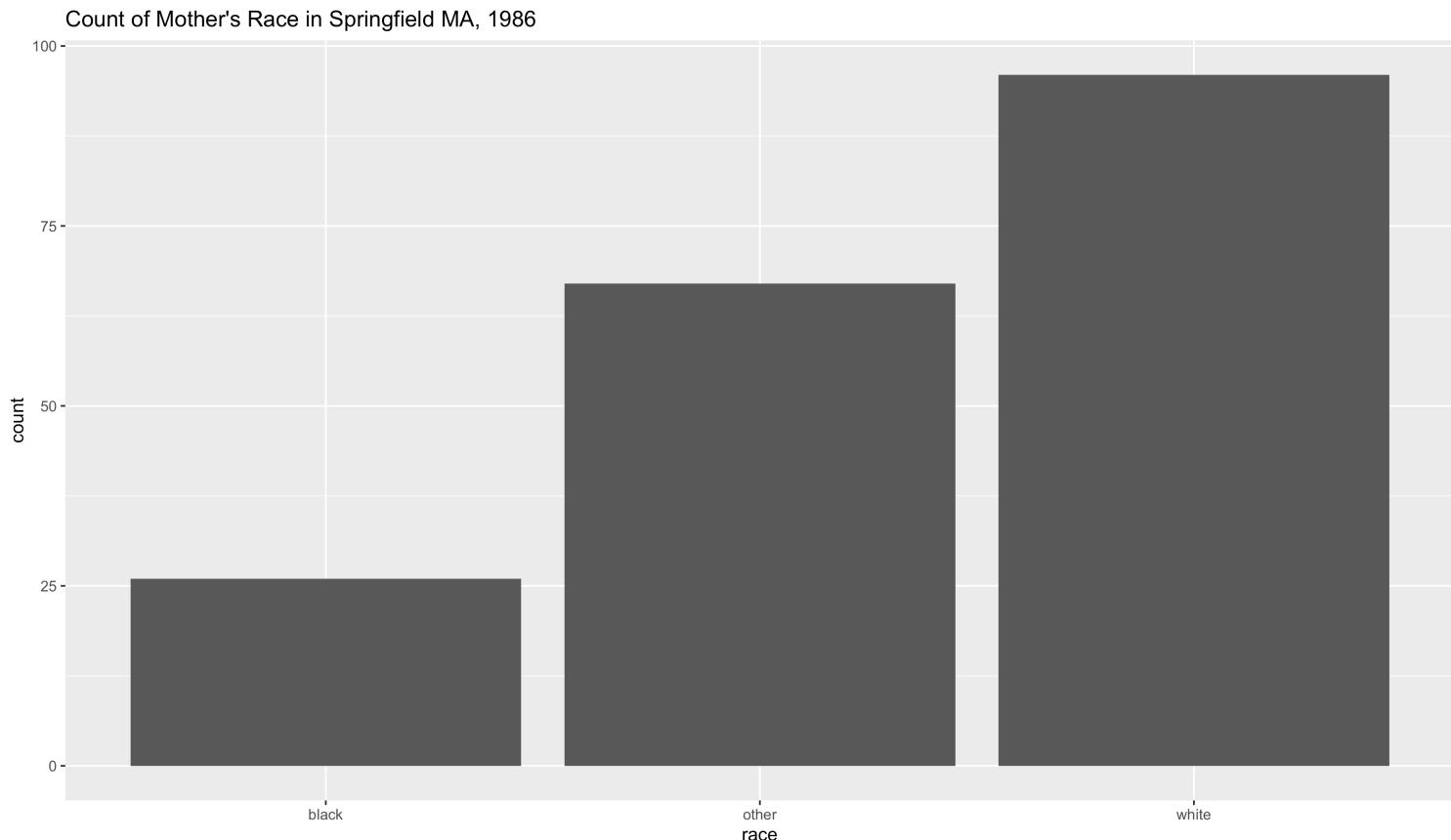
colnames(birthwt) <- c("birthwt.below.2500", "mother.age",
                      "mother.weight", "race",
                      "mother.smokes", "previous.prem.labor",
                      "hypertension", "uterine.irr",
                      "physician.visits", "birthwt.grams")
```

Let's make all the factors more descriptive.

```
birthwt$race <- factor(c("white", "black", "other"))[birthwt$race]
birthwt$mother.smokes <- factor(c("No", "Yes"))[birthwt$mother.smokes]
birthwt$uterine.irr <- factor(c("No", "Yes"))[birthwt$uterine.irr + 1]
birthwt$hypertension <- factor(c("No", "Yes"))[birthwt$hypertension + 1]
```

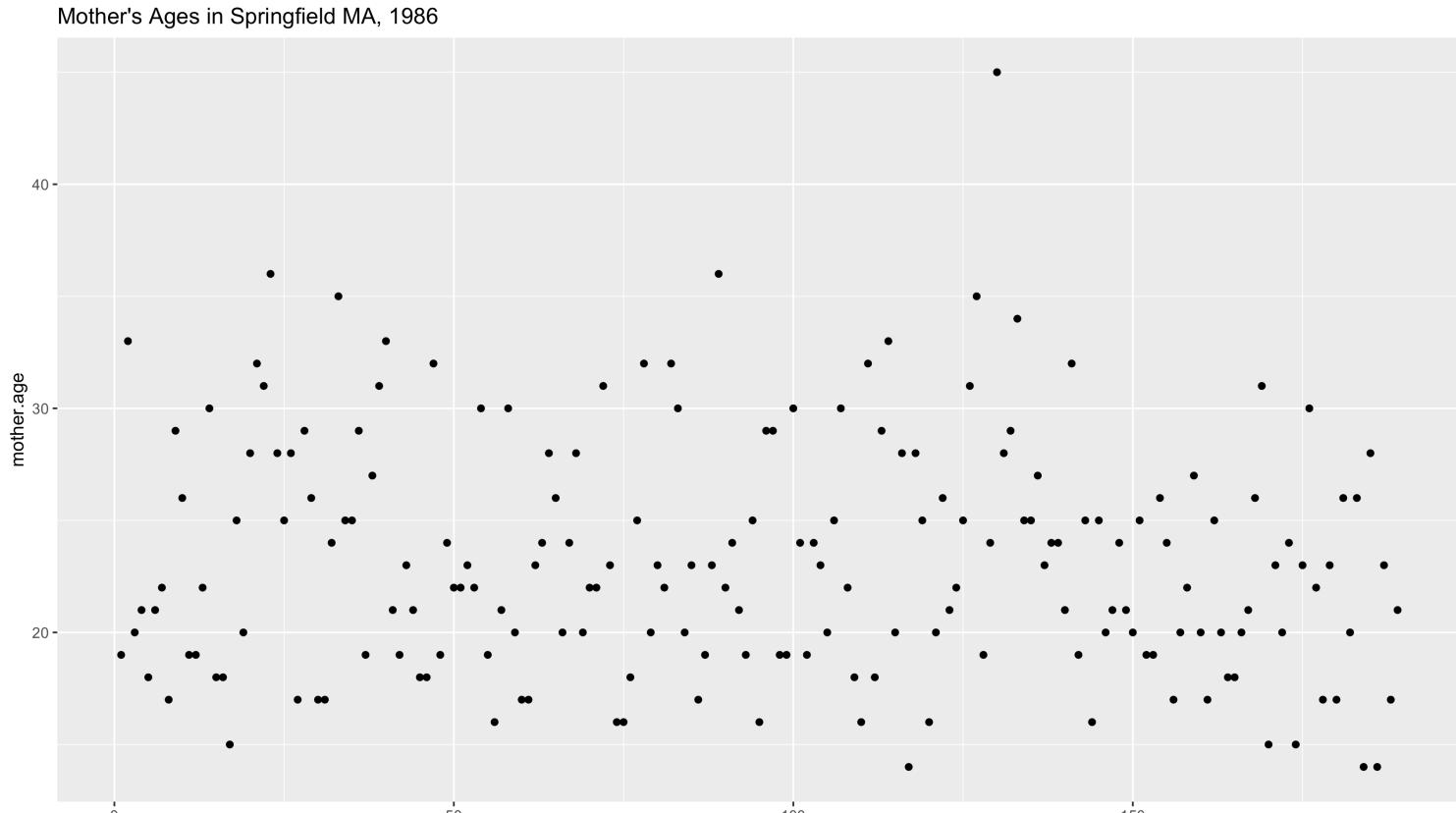
Bar plot for race

```
birthwt |> ggplot(aes(x = race))+
  geom_bar()+
  labs(title = "Count of Mother's Race in Springfield MA, 1986")
```



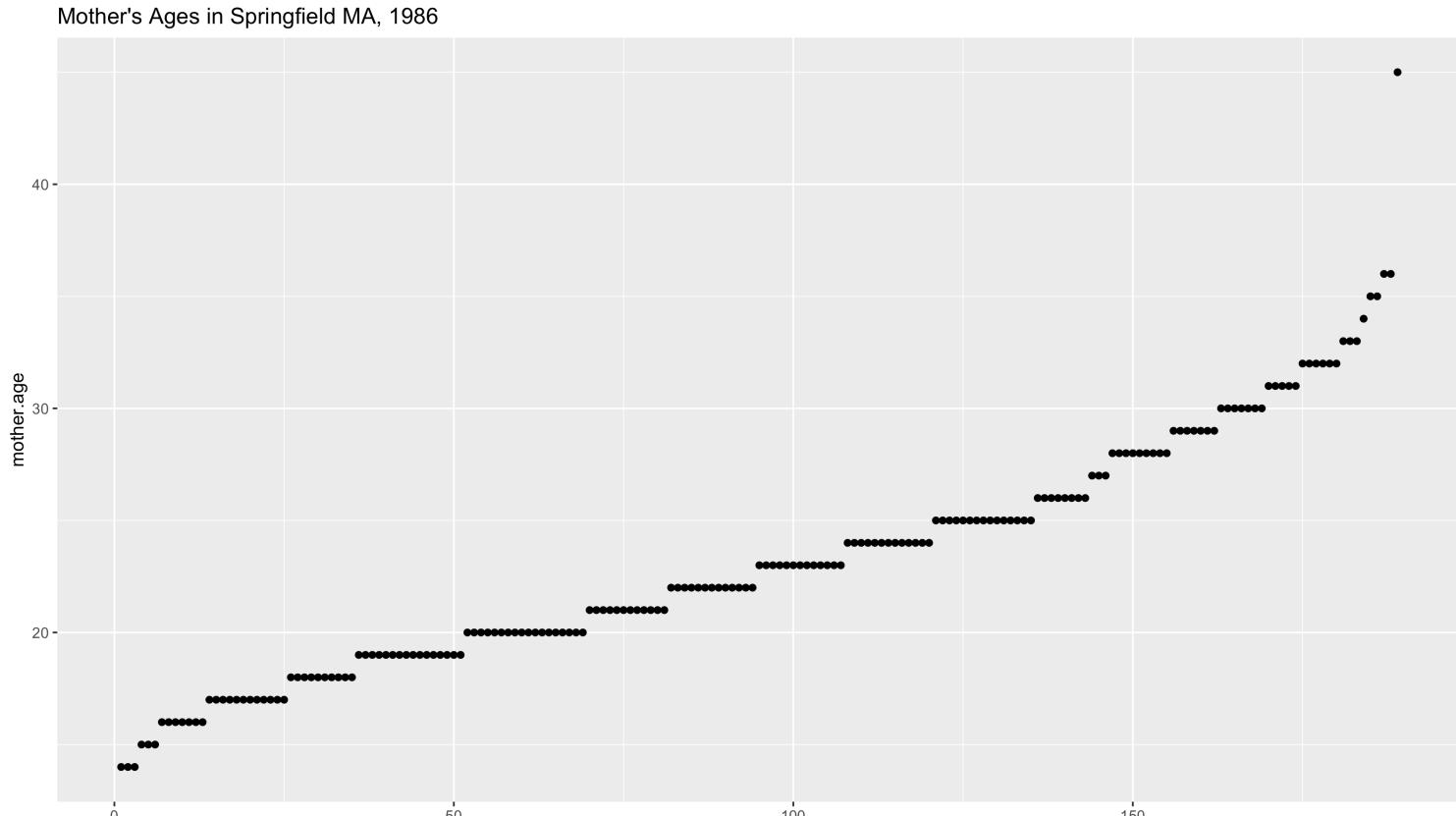
Scatter plot for mother's ages

```
birthwt |> ggplot(aes(x = 1:nrow(birthwt), y = mother.age))+
  geom_point()+
  labs(x = 'number', title = "Mother's Ages in Springfield MA, 1986")
```



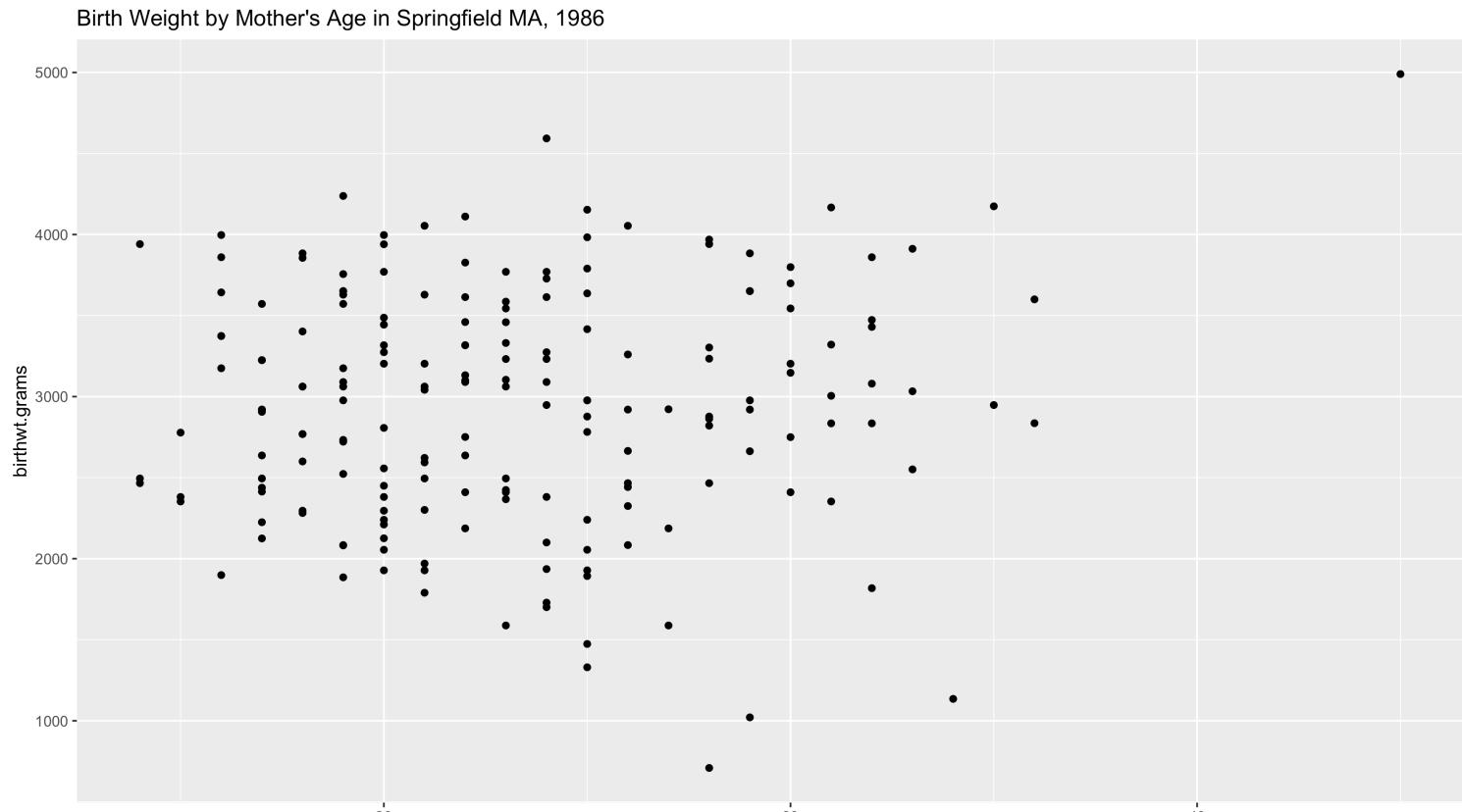
Sorted mother's ages

```
birthwt |> arrange(mother.age) |> ggplot(aes(x = 1:nrow(birthwt), y =  
geom_point() +  
labs(x = 'number', title = "Mother's Ages in Springfield MA, 1986")
```



Birth weight versus mother's ages

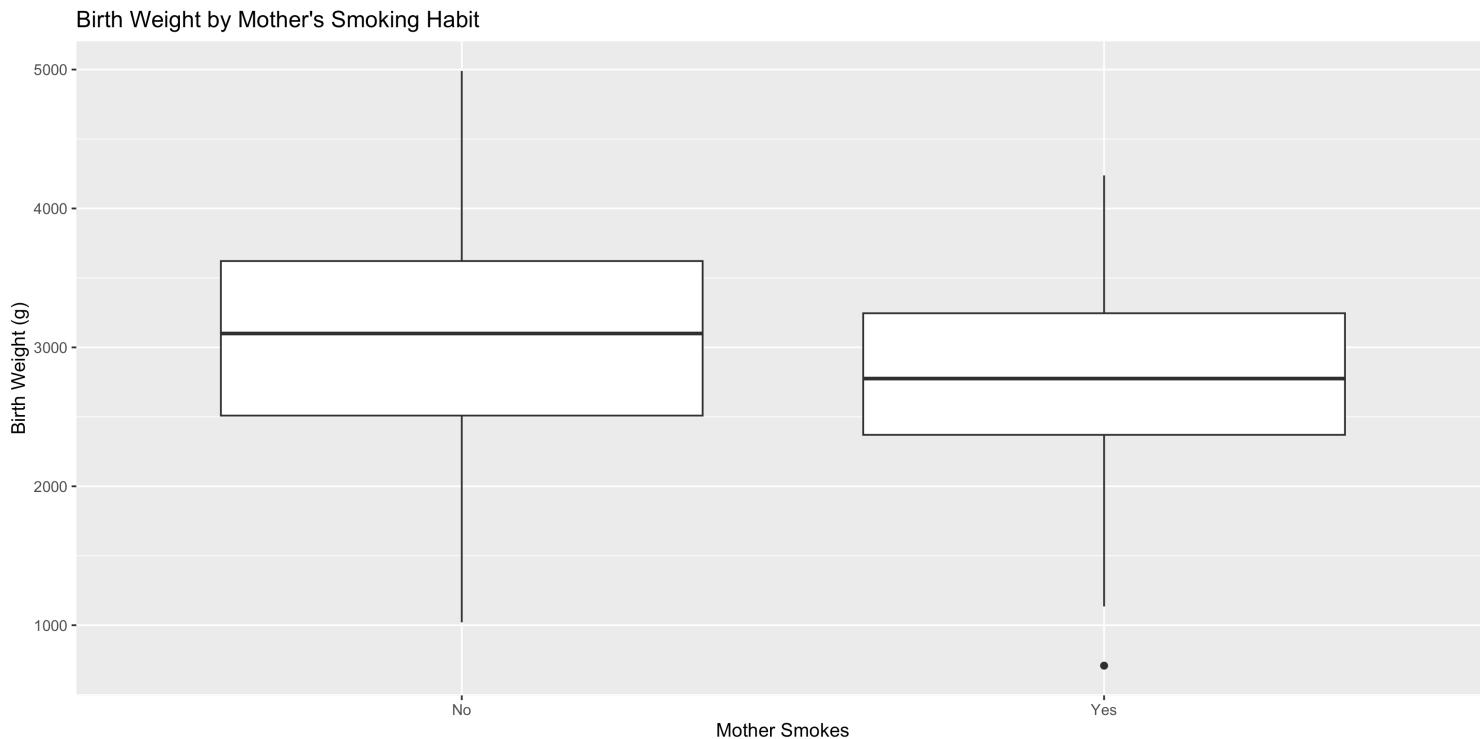
```
birthwt |> ggplot(aes(x = mother.age, y = birthwt.grams))+
  geom_point()+
  labs(title = "Birth Weight by Mother's Age in Springfield MA, 1986")
```



Boxplot

Let's fit some models to the data pertaining to our outcome(s) of interest.

```
birthwt |> ggplot(aes(x = mother.smokes, y = birthwt.grams))+
  geom_boxplot()+
  labs(title = "Birth Weight by Mother's Smoking Habit", y = "Birth Weight (g)")
```



Basic statistical testing

Tough to tell! Simple two-sample t-test:

```
t.test (birthwt$birthwt.grams[birthwt$mother.smokes == "Yes"],  
        birthwt$birthwt.grams[birthwt$mother.smokes == "No"], var.eq  
  
##  
##      Two Sample t-test  
##  
## data: birthwt$birthwt.grams[birthwt$mother.smokes == "Yes"] and birthwt$b  
## t = -2.6529, df = 187, p-value = 0.008667  
## alternative hypothesis: true difference in means is not equal to 0  
## 95 percent confidence interval:  
## -494.79735 -72.75612  
## sample estimates:  
## mean of x mean of y  
## 2771.919 3055.696
```

Does this difference match the linear model?

```
linear.model.1 <- lm (birthwt.grams ~ mother.smokes, data=birthwt)
summary(linear.model.1)
```

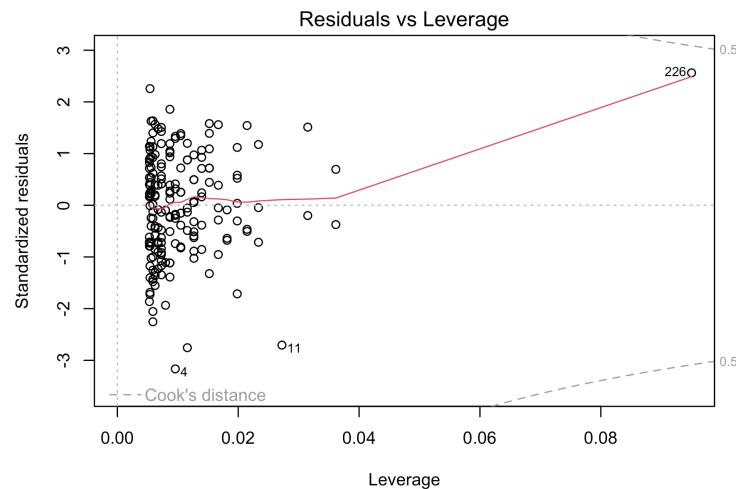
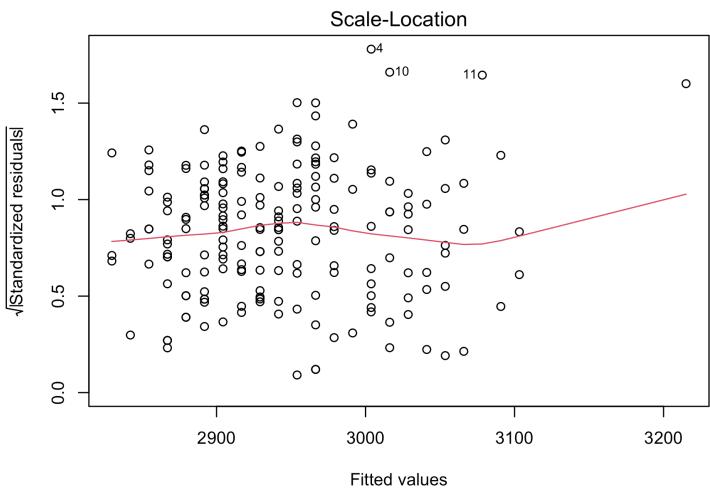
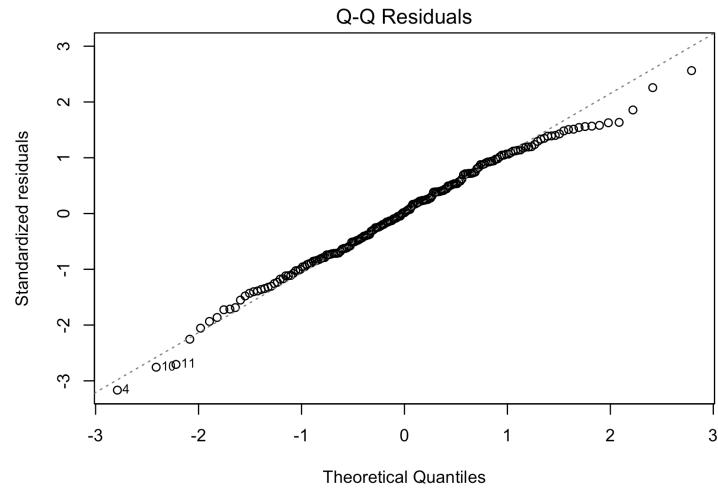
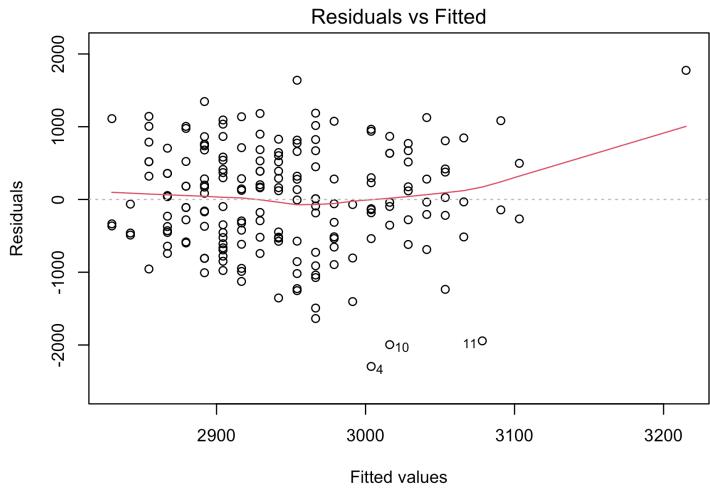
```
##
## Call:
## lm(formula = birthwt.grams ~ mother.smokes, data = birthwt)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -2062.9  -475.9    34.3   545.1  1934.3
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)    
## (Intercept) 3055.70     66.93  45.653 < 2e-16 ***
## mother.smokesYes -283.78    106.97  -2.653  0.00867 ** 
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 717.8 on 187 degrees of freedom
## Multiple R-squared:  0.03627,    Adjusted R-squared:  0.03112 
## F-statistic: 7.038 on 1 and 187 DF,  p-value: 0.008667
```

Basic statistical testing

Does this difference match the linear model?

```
linear.model.2 <- lm (birthwt.grams ~ mother.age, data=birthwt)
summary(linear.model.2)
```

```
##
## Call:
## lm(formula = birthwt.grams ~ mother.age, data = birthwt)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -2294.78  -517.63    10.51   530.80  1774.92
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) 2655.74     238.86   11.12  <2e-16 ***
## mother.age   12.43      10.02    1.24    0.216
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 728.2 on 187 degrees of freedom
## Multiple R-squared:  0.008157,    Adjusted R-squared:  0.002853
## F-statistic: 1.538 on 1 and 187 DF,  p-value: 0.2165
```



Detecting Outliers

These are the default diagnostic plots for the analysis. Note that our oldest mother and her heaviest child are greatly skewing this analysis as we suspected.

```
birthwt.noout <- birthwt |> filter(mother.age <= 40)
linear.model.3 <- lm (birthwt.grams ~ mother.age, data=birthwt.noout)
summary(linear.model.3)
```

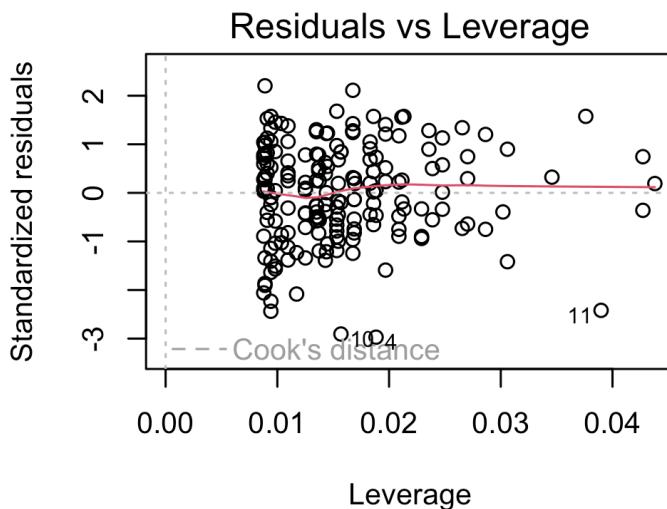
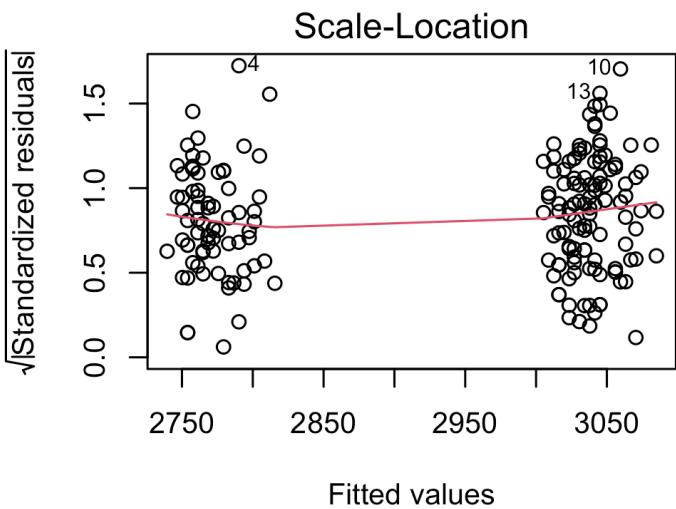
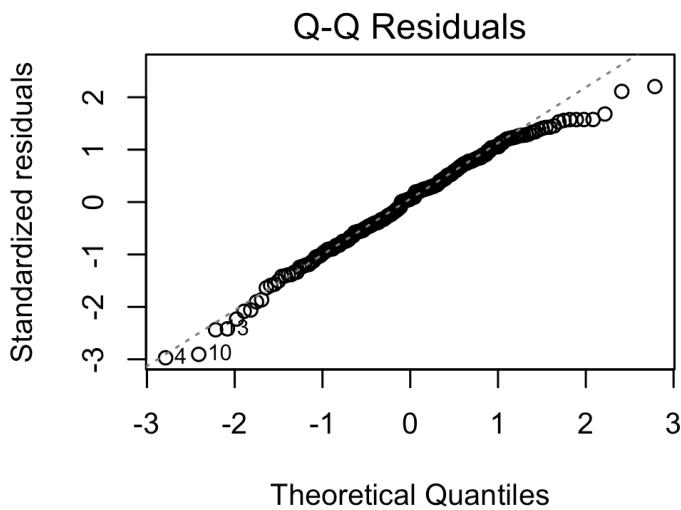
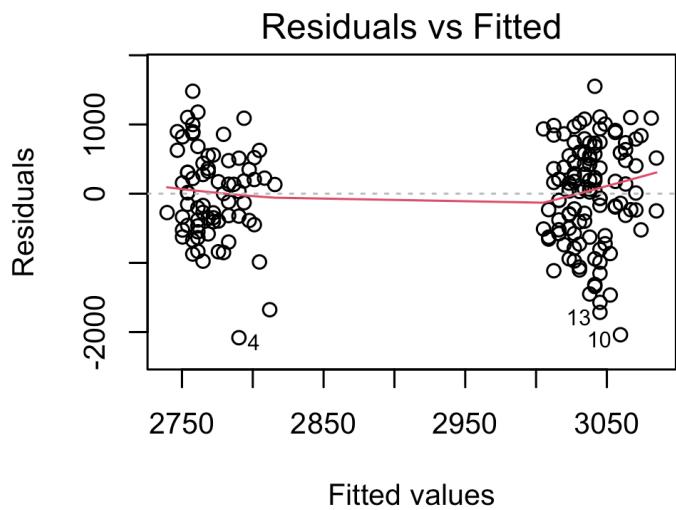
```
## 
## Call:
## lm(formula = birthwt.grams ~ mother.age, data = birthwt.noout)
## 
## Residuals:
##      Min       1Q   Median       3Q      Max 
## -2245.89  -511.24    26.45   540.09  1655.48 
## 
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)    
## (Intercept) 2833.273   244.954   11.57   <2e-16 ***
## mother.age    4.344    10.349    0.42     0.675    
## ---    
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

More complex models

Add in smoking behavior:

```
linear.model.3a <- lm (birthwt.grams ~ + mother.smokes + mother.age,  
summary(linear.model.3a)
```

```
##  
## Call:  
## lm(formula = birthwt.grams ~ +mother.smokes + mother.age, data = birthwt.r  
##  
## Residuals:  
##      Min       1Q   Median       3Q      Max  
## -2081.22  -459.82    43.56   548.22  1551.51  
##  
## Coefficients:  
##                               Estimate Std. Error t value Pr(>|t|)  
## (Intercept)            2954.582   246.280 11.997 <2e-16 ***  
## mother.smokesYes -265.756    105.605 -2.517  0.0127 *  
## mother.age           3.621     10.208  0.355  0.7232  
## ---  
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1  
##  
## Residual standard error: 707.1 on 185 degrees of freedom  
## Multiple R-squared:  0.03401,    Adjusted R-squared:  0.02357  
## F-statistic: 3.257 on 2 and 185 DF,  p-value: 0.04072
```

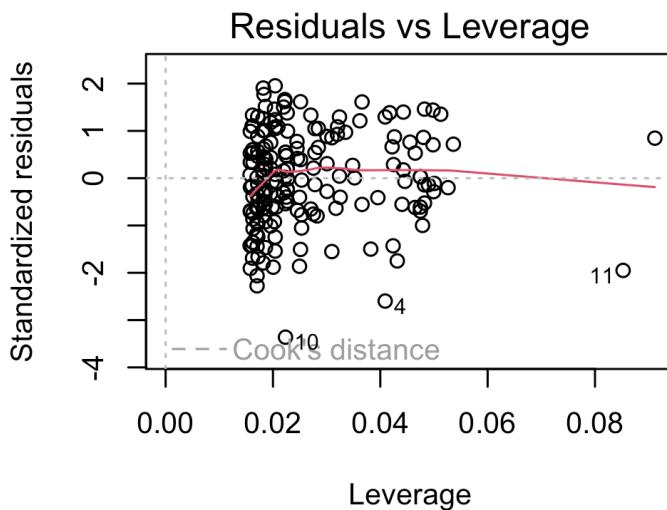
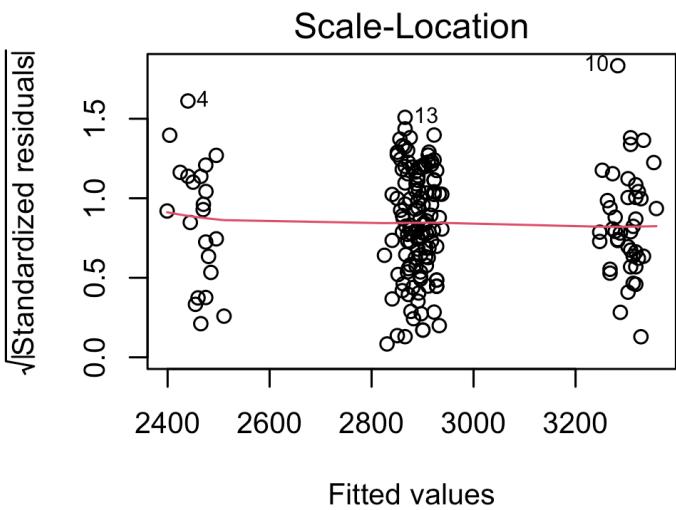
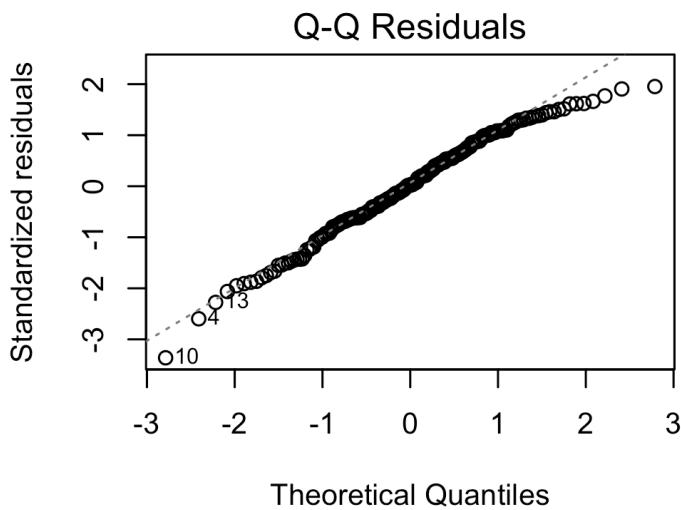
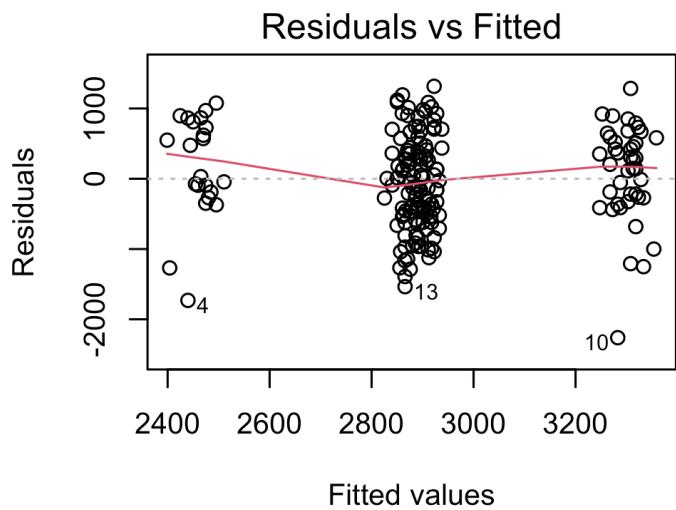


More complex models

Add in smoking behavior:

```
linear.model.3b <- lm (birthwt.grams ~ mother.age + mother.smokes + r  
summary(linear.model.3b)
```

```
##  
## Call:  
## lm(formula = birthwt.grams ~ mother.age + mother.smokes + race,  
##      data = birthwt.noout)  
##  
## Residuals:  
##      Min       1Q   Median       3Q      Max  
## -2261.76  -422.49    15.98   512.00  1315.40  
##  
## Coefficients:  
##                               Estimate Std. Error t value Pr(>|t|)  
## (Intercept)            2986.405   260.897 11.447 < 2e-16 ***  
## mother.age             -5.050    10.056 -0.502 0.616136  
## mother.smokesYes -410.656   108.635 -3.780 0.000212 ***  
## raceother              5.487   158.918  0.035 0.972492  
## racewhite              442.799   154.023  2.875 0.004521 **  
## ---  
## Signif. codes: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```



Everything Must Go (ln)

Let's do a kitchen sink model on this new data set:

```
linear.model.4 <- lm (birthwt.grams ~ ., data=birthwt.noout)
summary(linear.model.4)
```

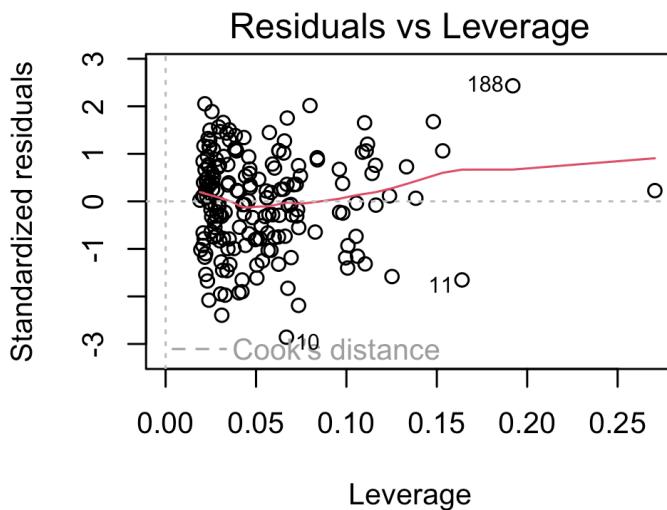
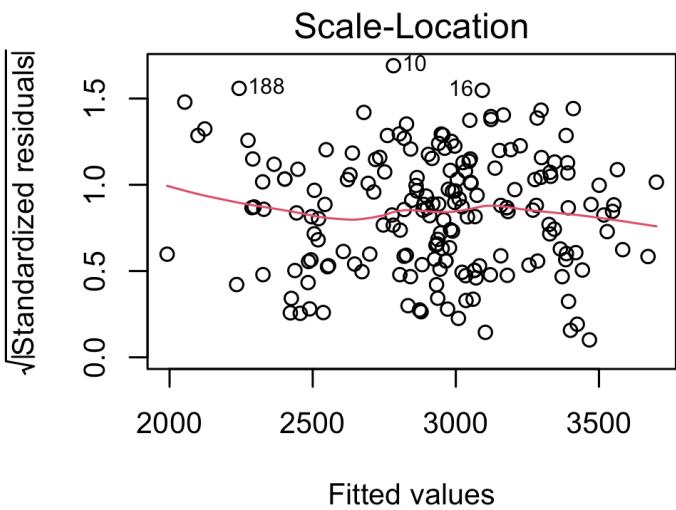
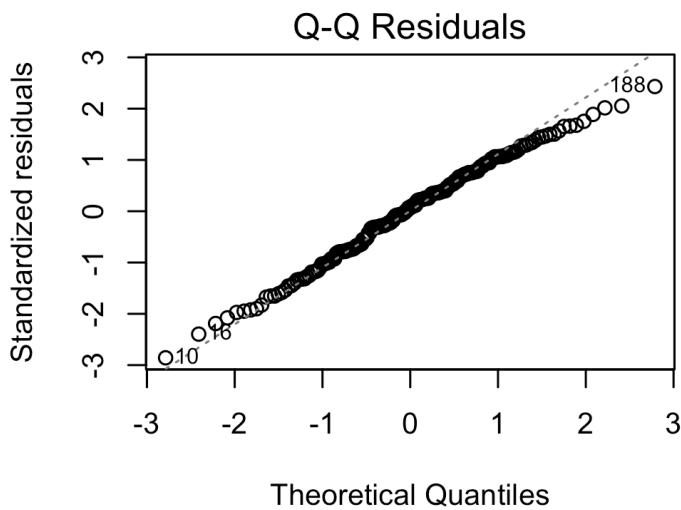
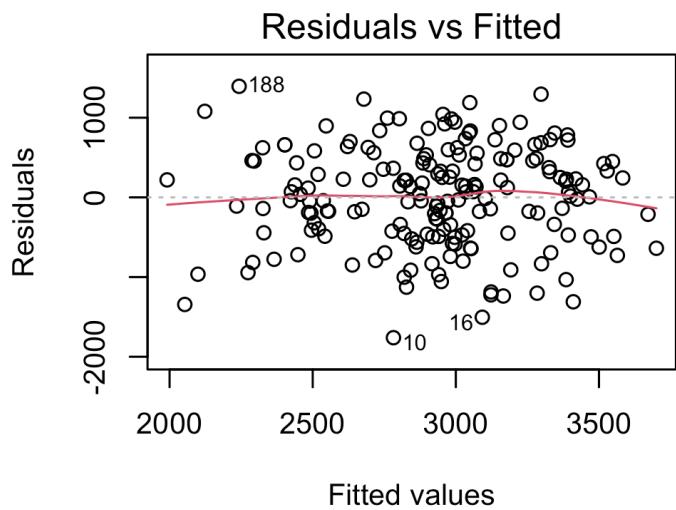
```
##
## Call:
## lm(formula = birthwt.grams ~ ., data = birthwt.noout)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -985.04 -274.13  -13.87  262.53 1146.50
##
## Coefficients:
##                               Estimate Std. Error t value Pr(>|t|)
## (Intercept)            3360.5163   215.4112  15.600 < 2e-16 ***
## birthwt.below.2500    -1116.3933    70.8578 -15.755 < 2e-16 ***
## mother.age             -16.0321     6.4159  -2.499 0.013373 *
## mother.weight          1.9317      1.1208   1.723 0.086545 .
## raceother              68.8145    101.4451   0.678 0.498441
## racewhite              247.0241    96.4935   2.560 0.011302 *
## mother.smokesYes      -157.7041    68.6205  -2.298 0.022719 *
## previous.prem.labor    95.9825    65.3329   1.469 0.143573
## hypertensionYes        -185.2778   131.0126  -1.414 0.159060
```

Everything Must Go (In), Except What Must Not

Whoops! One of those variables was birthwt.below.2500 which is a function of the outcome.

```
linear.model.4a <- lm (birthwt.grams ~ . - birthwt.below.2500, data=k
summary(linear.model.4a)
```

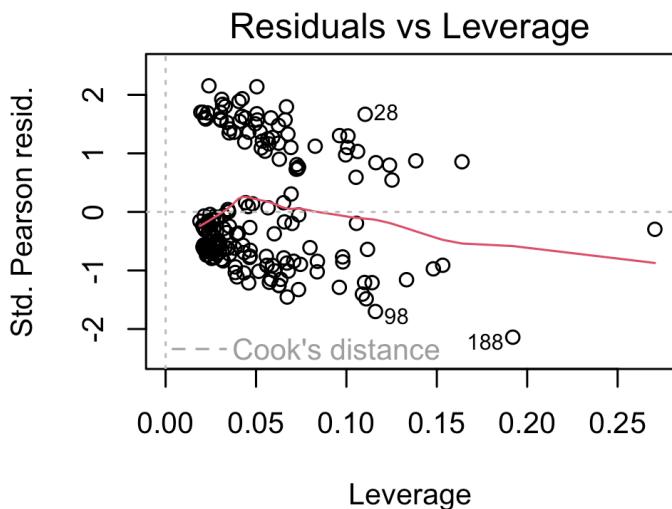
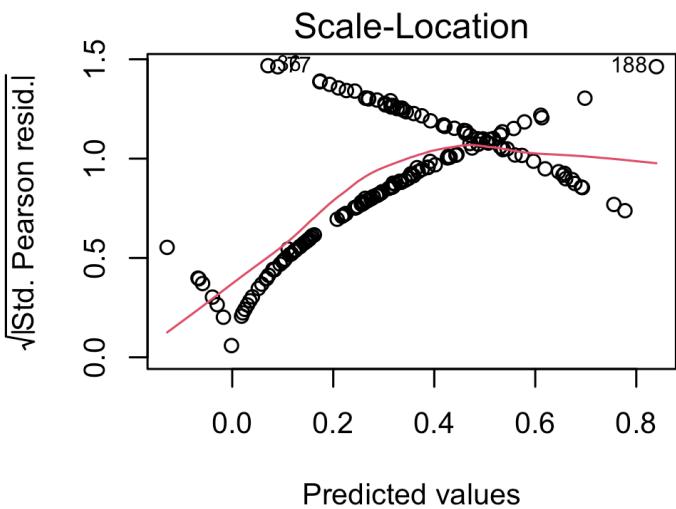
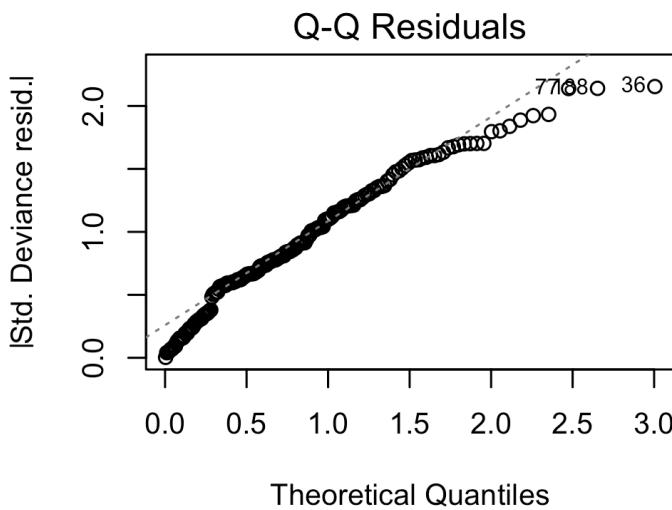
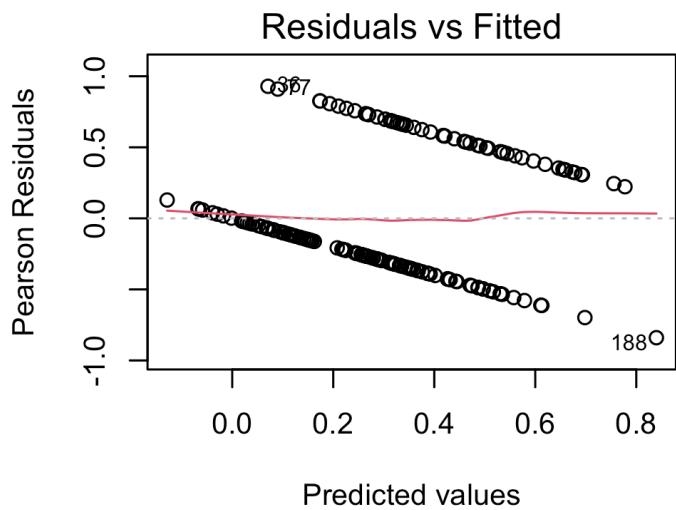
```
##  
## Call:  
## lm(formula = birthwt.grams ~ . - birthwt.below.2500, data = birthwt.noout)  
##  
## Residuals:  
##      Min       1Q   Median       3Q      Max  
## -1761.10  -454.81    46.43   459.78  1394.13  
##  
## Coefficients:  
##                               Estimate Std. Error t value Pr(>|t|)  
## (Intercept)              2545.584   323.204   7.876 3.21e-13 ***  
## mother.age                -12.111     9.909  -1.222 0.223243  
## mother.weight                 4.789     1.710   2.801 0.005656 **  
## raceother                  155.605   156.564   0.994 0.321634  
## racewhite                  494.545   147.153   3.361 0.000951 ***  
## mother.smokesYes          -335.793   104.613  -3.210 0.001576 **
```



Generalized Linear Models

Maybe a linear increase in birth weight is less important than if it's below a threshold like 2500 grams (5.5 pounds). Let's fit a generalized linear model instead:

```
glm.0 <- glm (birthwt.below.2500 ~ . - birthwt.grams, data=birthwt.no
```

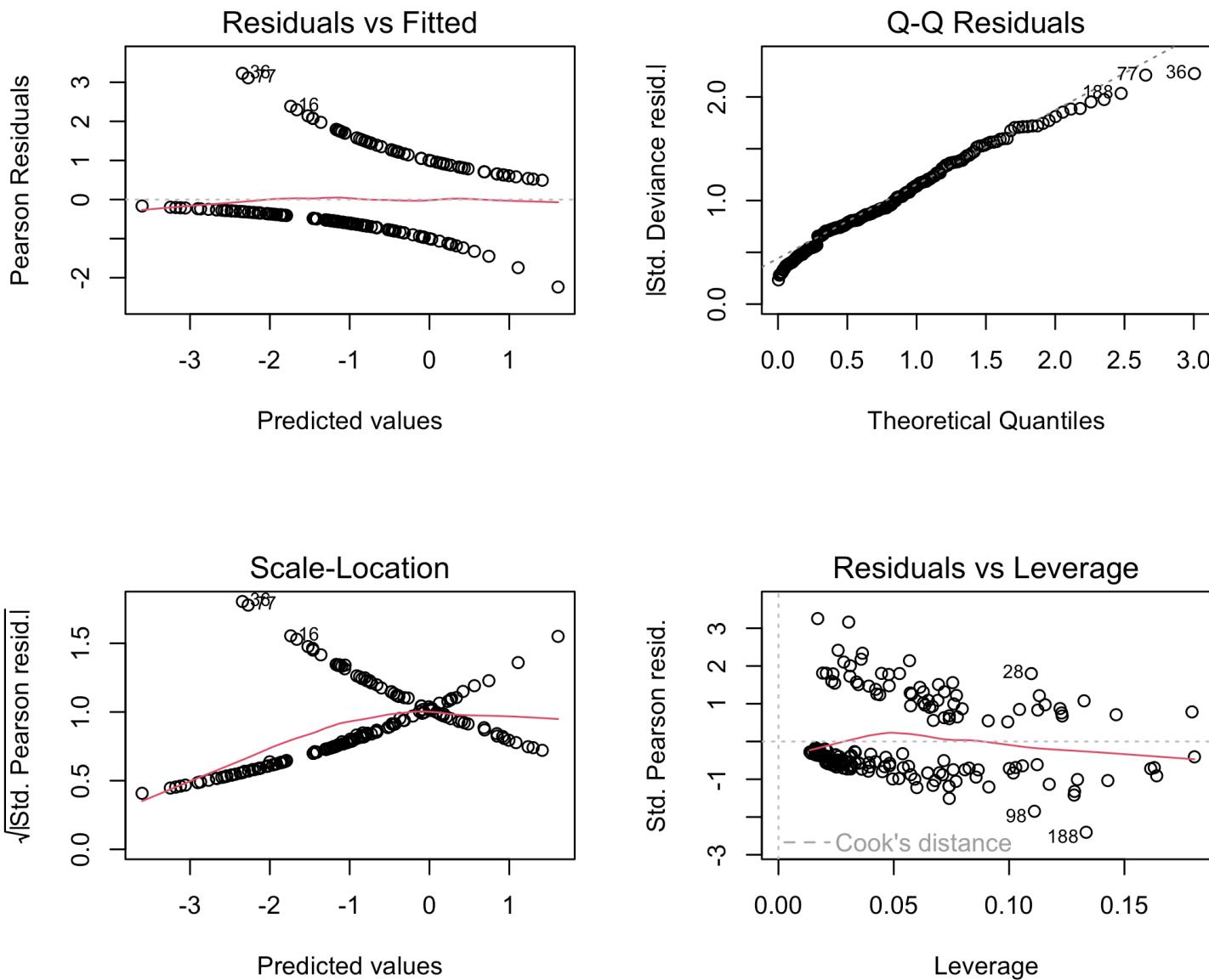


Generalized Linear Models

The default value is a Gaussian model (a standard linear model). Change this:

```
glm.1 <- glm (birthwt.below.2500 ~ . - birthwt.grams, data=birthwt.noout)
summary(glm.1)
```

```
## 
## Call:
## glm(formula = birthwt.below.2500 ~ . - birthwt.grams, family = binomial(link = "logit"),
##      data = birthwt.noout)
## 
## Coefficients:
##                               Estimate Std. Error z value Pr(>|z|)
## (Intercept)                1.721830   1.258897   1.368  0.17140
## mother.age                 -0.027537   0.037718  -0.730  0.46534
## mother.weight               -0.015474   0.006919  -2.237  0.02532 *
## raceother                  -0.395505   0.537685  -0.736  0.46199
## racewhite                  -1.269006   0.527180  -2.407  0.01608 *
## mother.smokesYes            0.931733   0.402359   2.316  0.02058 *
## previous.prem.labor        0.539549   0.345413   1.562  0.11828
## hypertensionYes              1.860521   0.697502   2.667  0.00764 **
## uterine.irrYes               0.766517   0.458951   1.670  0.09489 .
## physician.visits             0.063402   0.172431   0.368  0.71310
##
```



What Do We Do With This, Anyway?

Let's take a subset of this data to do predictions.

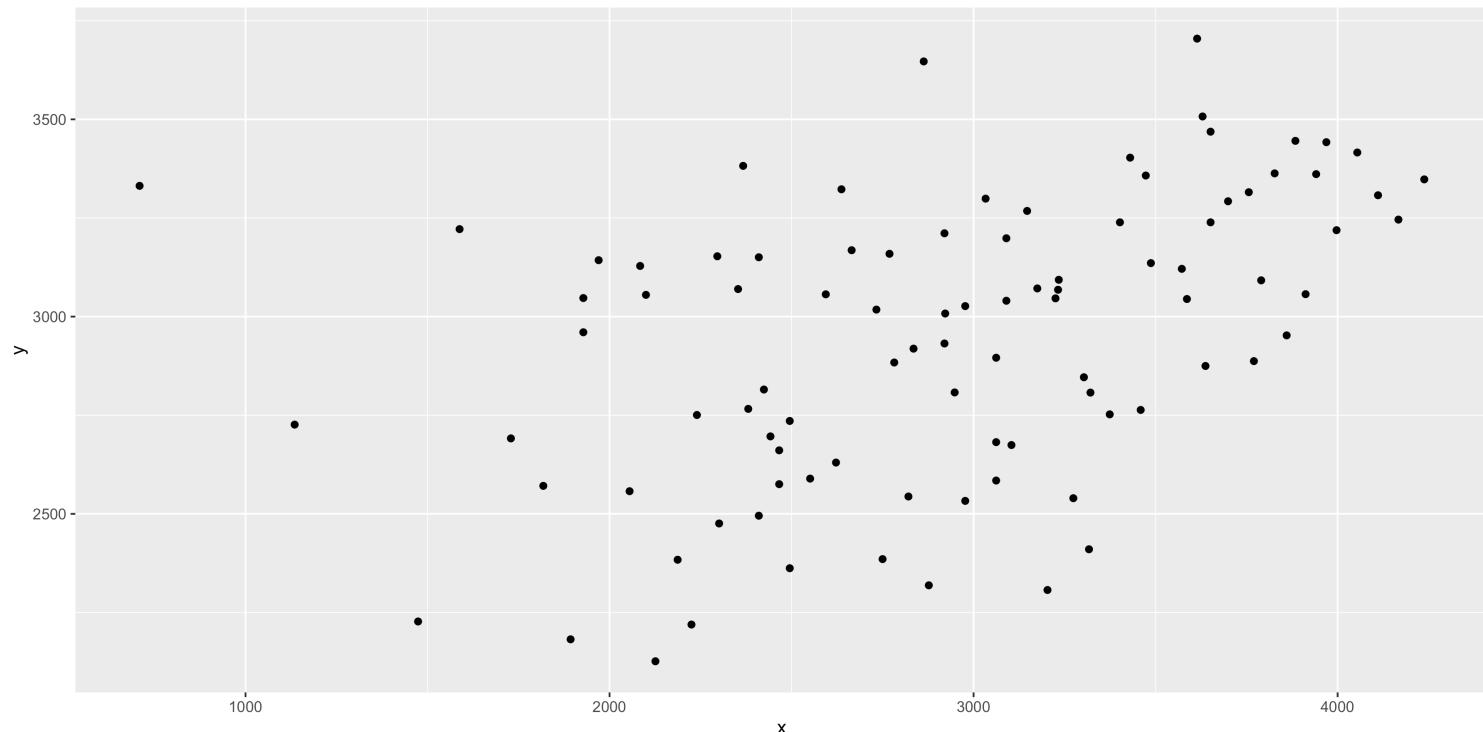
```
odds <- seq(1, nrow(birthwt.noout), by=2)
birthwt.in <- birthwt.noout[odds,]
birthwt.out <- birthwt.noout[-odds,]
linear.model.half <- lm (birthwt.grams ~ . - birthwt.below.2500, data = birthwt.in)
summary (linear.model.half)

## 
## Call:
## lm(formula = birthwt.grams ~ . - birthwt.below.2500, data = birthwt.in)
##
## Residuals:
##      Min       1Q   Median       3Q      Max 
## -1705.17  -303.11    26.48   427.18  1261.57 
## 
## Coefficients:
##                               Estimate Std. Error t value Pr(>|t|)    
## (Intercept)              2514.891   450.245   5.586 2.81e-07 ***
## mother.age                  7.052    14.935   0.472  0.63801    
## mother.weight                 2.683     2.885   0.930  0.35501    
## raceother                   113.948   224.519   0.508  0.61312    
## .lin  
 100.010    204.007   0.477  0.63710
```

```
birthwt.predict <- predict (linear.model.half)
cor (birthwt.in$birthwt.grams, birthwt.predict)
```

```
## [1] 0.508442
```

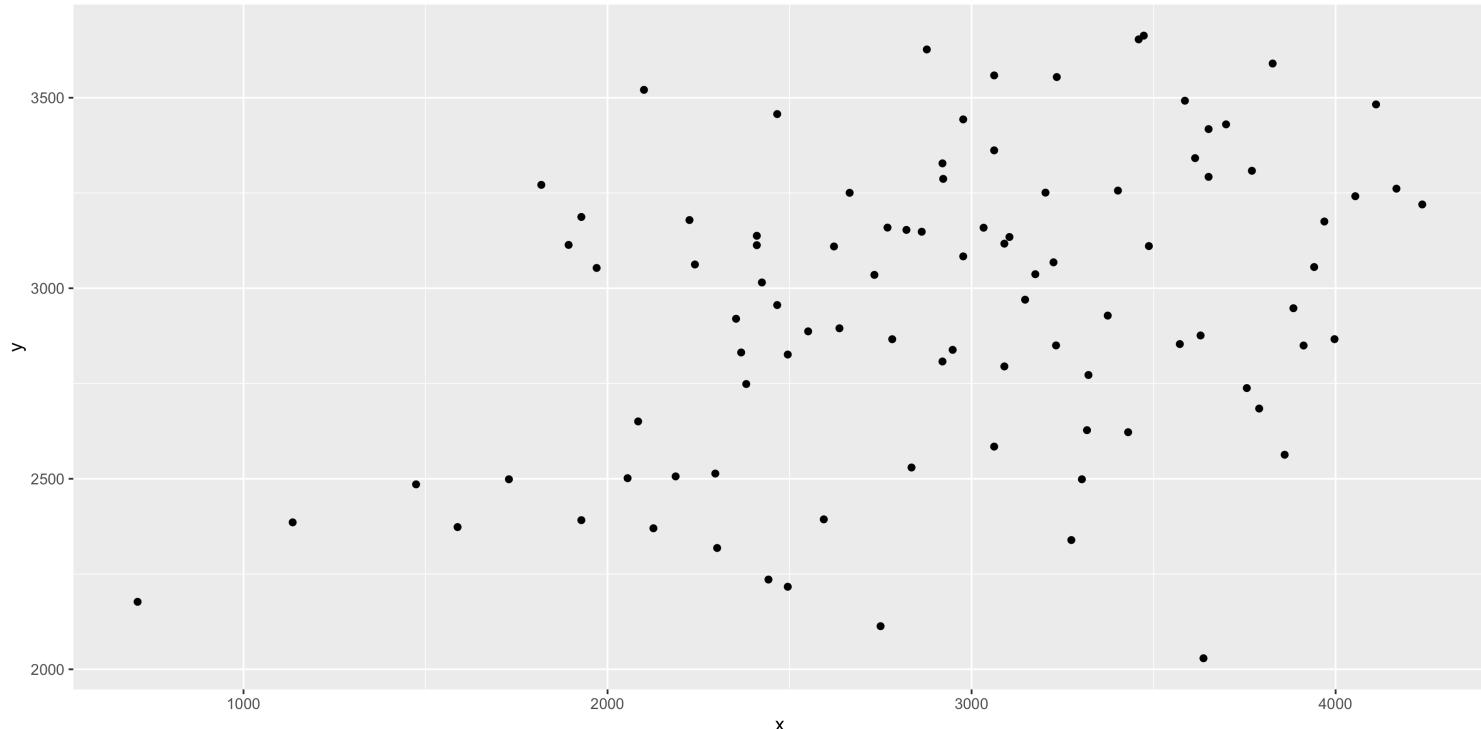
```
tibble(x = birthwt.out$birthwt.grams, y = birthwt.predict) |>
  ggplot (aes(x = x, y = y)) + geom_point()
```



What Do We Do With This, Anyway?

```
birthwt.predict.out <- predict (linear.model.half, birthwt.out)
cor (birthwt.out$birthwt.grams, birthwt.predict.out)
```

```
## [1] 0.3749431
```



Random number generators

- We made reference to random number generation without going under the hood.
- How *does* R get "random" numbers?
- It doesn't, really -- it uses a trick that should be indistinguishable from the real McCoy

Pseudorandom generators produce a deterministic sequence that is indistinguishable from a true random sequence if you don't know how it started.

Example: `runif`, where we know where it started

```
runif(1:10)
```

```
## [1] 0.69731331 0.49422049 0.77964656 0.98497476 0.03847107 0.59479792
## [7] 0.61858268 0.89822554 0.50665171 0.26237437
```

```
set.seed(10)
runif(1:10)
```

```
## [1] 0.50747820 0.30676851 0.42690767 0.69310208 0.08513597 0.2254366283
## [7] 0.27453052 0.27230507 0.61582931 0.42967153
```

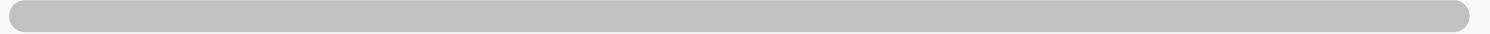
Basic version: Linear Congruential Generator

```
seed <- 10
new.random <- function (a=5, c=12, m=16) {
  out <- (a*seed + c) %% m
  seed <~- out
  return(out)
}
out.length <- 20
variates <- rep (NA, out.length)
for (kk in 1:out.length) variates[kk] <- new.random()
variates
##  [1] 14  2   6 10 14  2   6 10 14  2   6 10 14  2   6 10
```

Try again

Period 8:

```
variates <- rep (NA, out.length)
for (kk in 1:out.length) variates[kk] <- new.random(a=131, c=7, m=16)
variates
## [1] 5 6 9 2 13 14 1 10 5 6 9 2 13 14 1 10 5 6 9 2
```



Try again, again

Period 16:

```
variates <- rep (NA, out.length)
for (kk in 1:out.length) variates[kk] <- new.random(a=129, c=7, m=16)
variates
## [1] 9 0 7 14 5 12 3 10 1 8 15 6 13 4 11 2 9 0 7 14
```



Try again, at last

Numerical Recipes uses

```
variates <- rep (NA, out.length)
for (kk in 1:out.length) variates[kk] <- new.random(a=1664545, c=1013)
variates
```



```
## [1] 1037207853 2090831916 4106096907 768378826 3835752553 1329121000
## [7] 2125006663 2668506502 3581687205 2079234980 2067291011 2197025090
## [13] 3748878561 2913996384 758844863 4029469438 2836748829 1458315036
## [19] 2399149563 2766656186
```

How To Distinguish Non-Randomness

- We've covered period: if it's missing some values, it's distinguishable
- Uniformity of distribution in the limitx
- Autocorrelation
- Dimensional distribution -- not a problem for 1-D distributions, but can be for 2+-D

How does R get everything we need?

A few distributions of interest:

- Uniform(0,1)
- Bernoulli(p)
- Binomial(n,p)
- Gaussian(0,1)
- Exponential(1)
- Gamma(a)

In R: everything we need

Suppose we were working with the Exponential distribution.

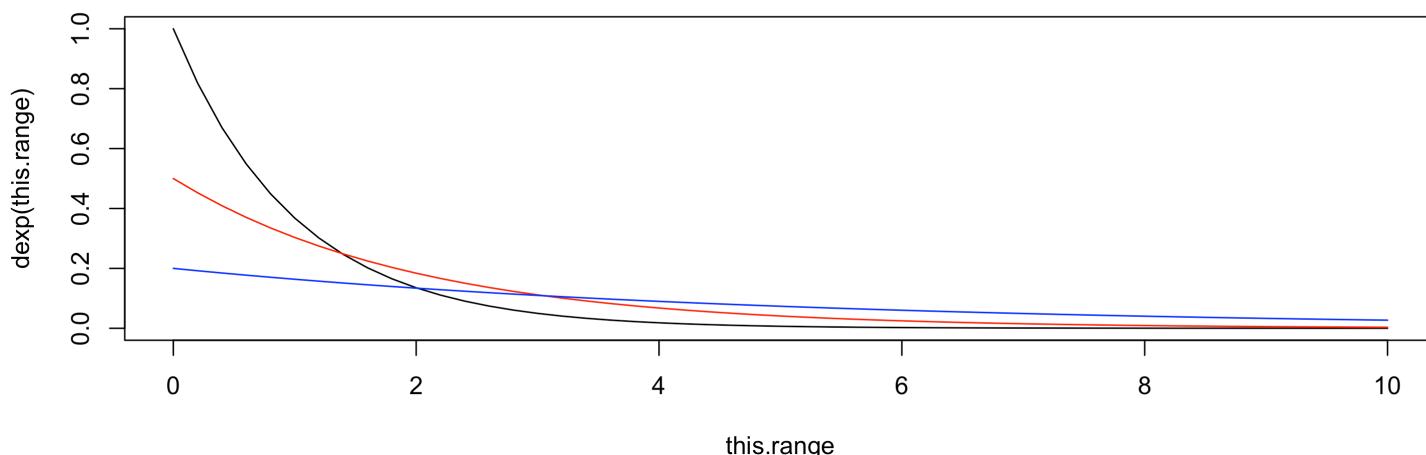
- `rexp()` generates variates from the distribution.
- `dexp()` gives the probability density function.
- `pexp()` gives the cumulative distribution function.
- `qexp()` gives the quantiles.

dexp()

```
dexp(0:5)
```

```
## [1] 1.000000000 0.367879441 0.135335283 0.049787068 0.018315639 0.00673794
```

```
this.range <- 0:50/5
plot (this.range, dexp(this.range), ty="l")
lines (this.range, dexp(this.range, rate=0.5), col="red")
lines (this.range, dexp(this.range, rate=0.2), col="blue")
```

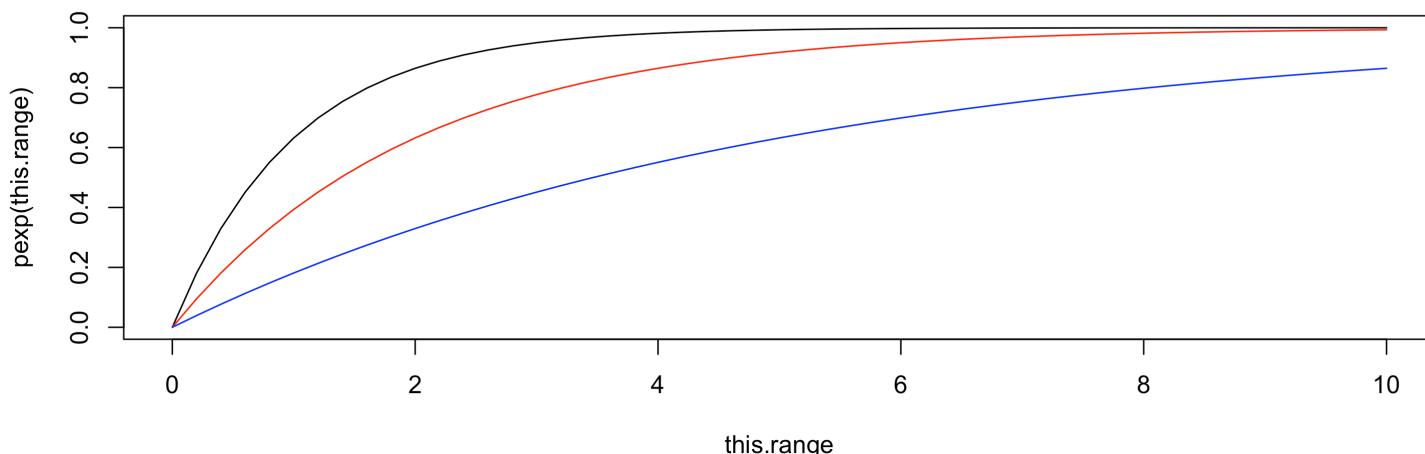


pexp()

```
pexp(0:5)
```

```
## [1] 0.0000000 0.6321206 0.8646647 0.9502129 0.9816844 0.9932621
```

```
this.range <- 0:50/5
plot (this.range, pexp(this.range), ty="l")
lines (this.range, pexp(this.range, rate=0.5), col="red")
lines (this.range, pexp(this.range, rate=0.2), col="blue")
```



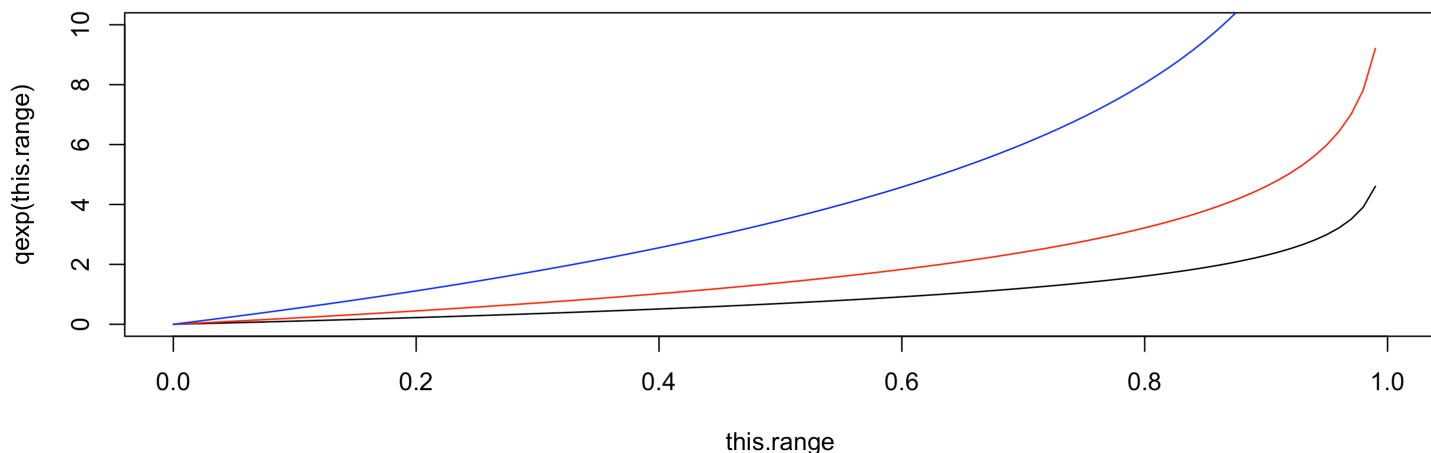
qexp()

```
qexp(0:5)
```

```
## Warning in qexp(0:5): NaNs produced
```

```
## [1] 0 Inf NaN NaN NaN NaN
```

```
this.range <- seq(0,1,by=0.01)
plot (this.range, qexp(this.range), ylim = c(0, 10), ty="l")
lines (this.range, qexp(this.range, rate=0.5), col="red")
lines (this.range, qexp(this.range, rate=0.2), col="blue")
```

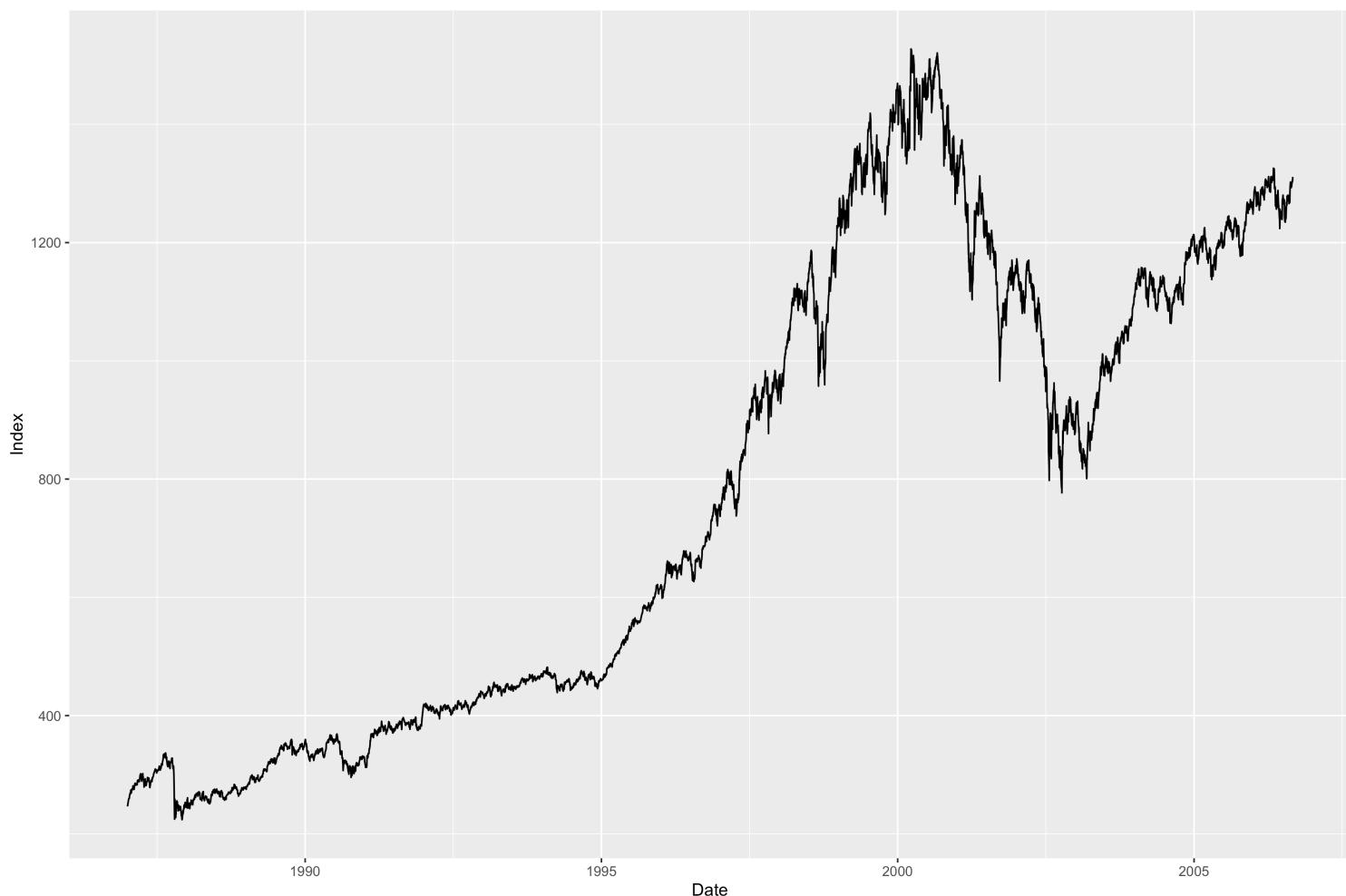


Let's Grab some Data

The Standard and Poor's 500, or simply the S&P 500, is a stock market index tracking the stock performance of 500 large companies listed on exchanges in the United States. It is one of the most commonly followed equity indices.

```
library(readxl)
SP <- read_excel("data/Stock_Bond.xls") |> dplyr::select(Date, `S&P_/
  rename(Index = `S&P_AC`)
```

```
SP |> ggplot(aes(x = Date, y = Index)) + geom_line()
```



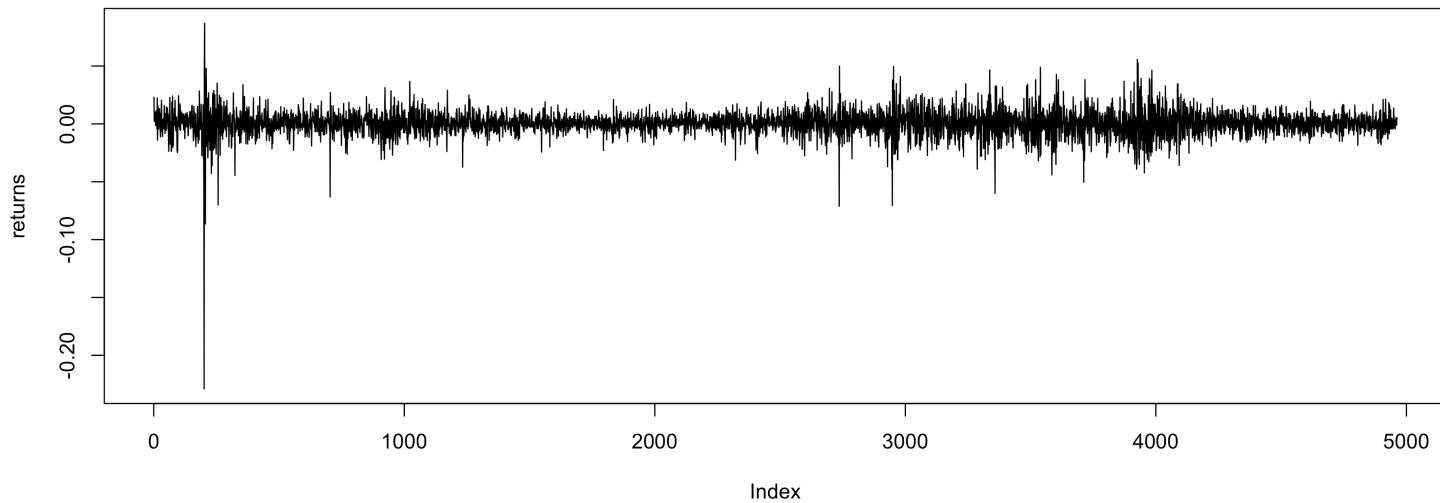
Let's Transform Some Data

The price p_t doesn't matter, what matters are the returns $r_t = \log(p_t/p_{t-1})$

```
returns <- na.omit(as.vector(diff(log(SP$Index))))  
summary(returns)
```

```
##           Min.     1st Qu.      Median       Mean     3rd Qu.       Max.  
## -0.2289972 -0.0046537  0.0004976  0.0003368  0.0056195  0.0870888
```

```
plot(returns, type="l")
```



The Data's Distribution

`quantile(x, probs)` calculates the quantiles at `probs` from `x`

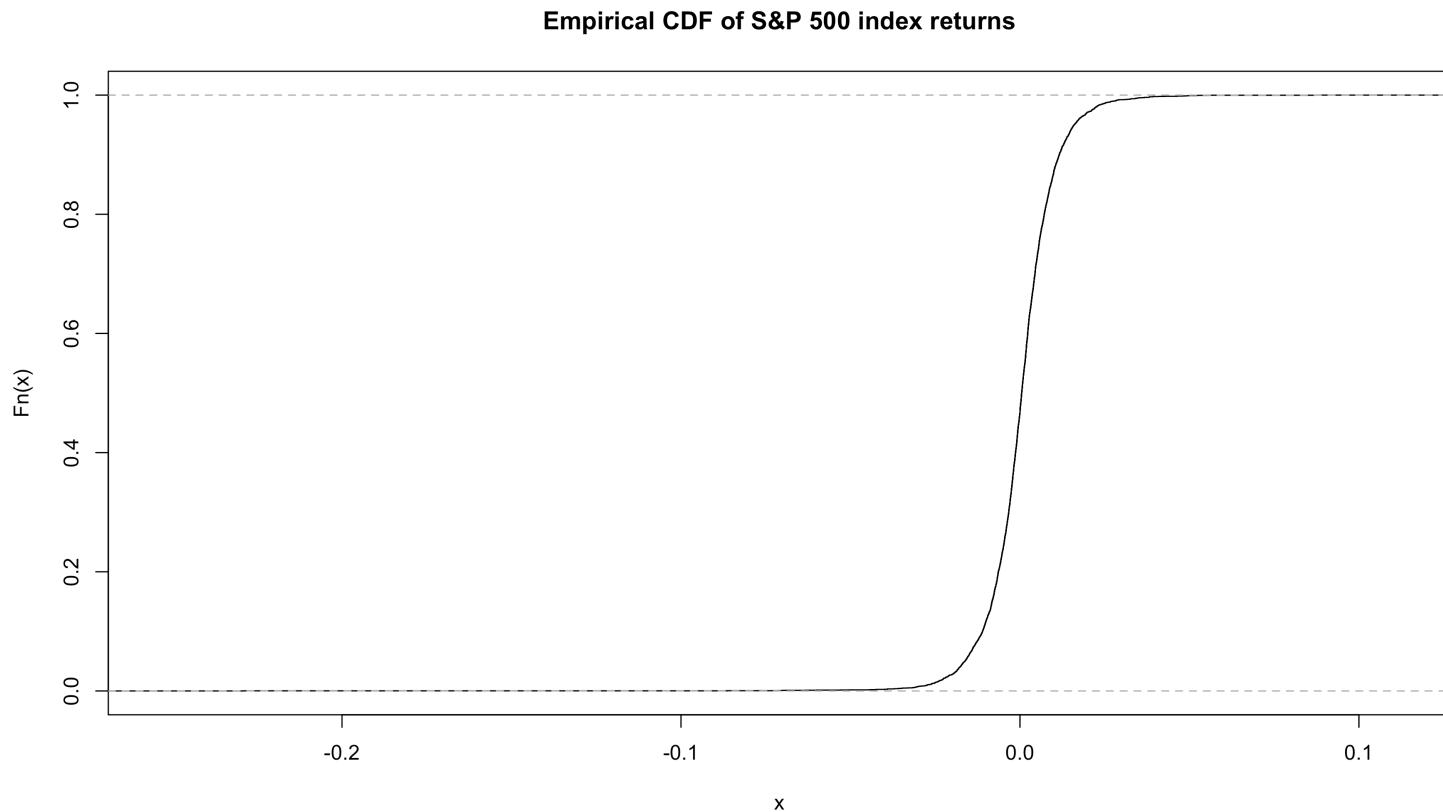
```
quantile(returns,c(0.25,0.5,0.75))
```

```
##           25%          50%          75%
## -0.0046537538  0.0004976042  0.0056195438
```

`ecdf()` - empirical cumulative distribution function; no assumptions but also no guess about distribution between the observations

In math, ECDF is often written as \hat{F} or \hat{F}_n

```
plot(ecdf(returns), main="Empirical CDF of S&P 500 index returns")
```



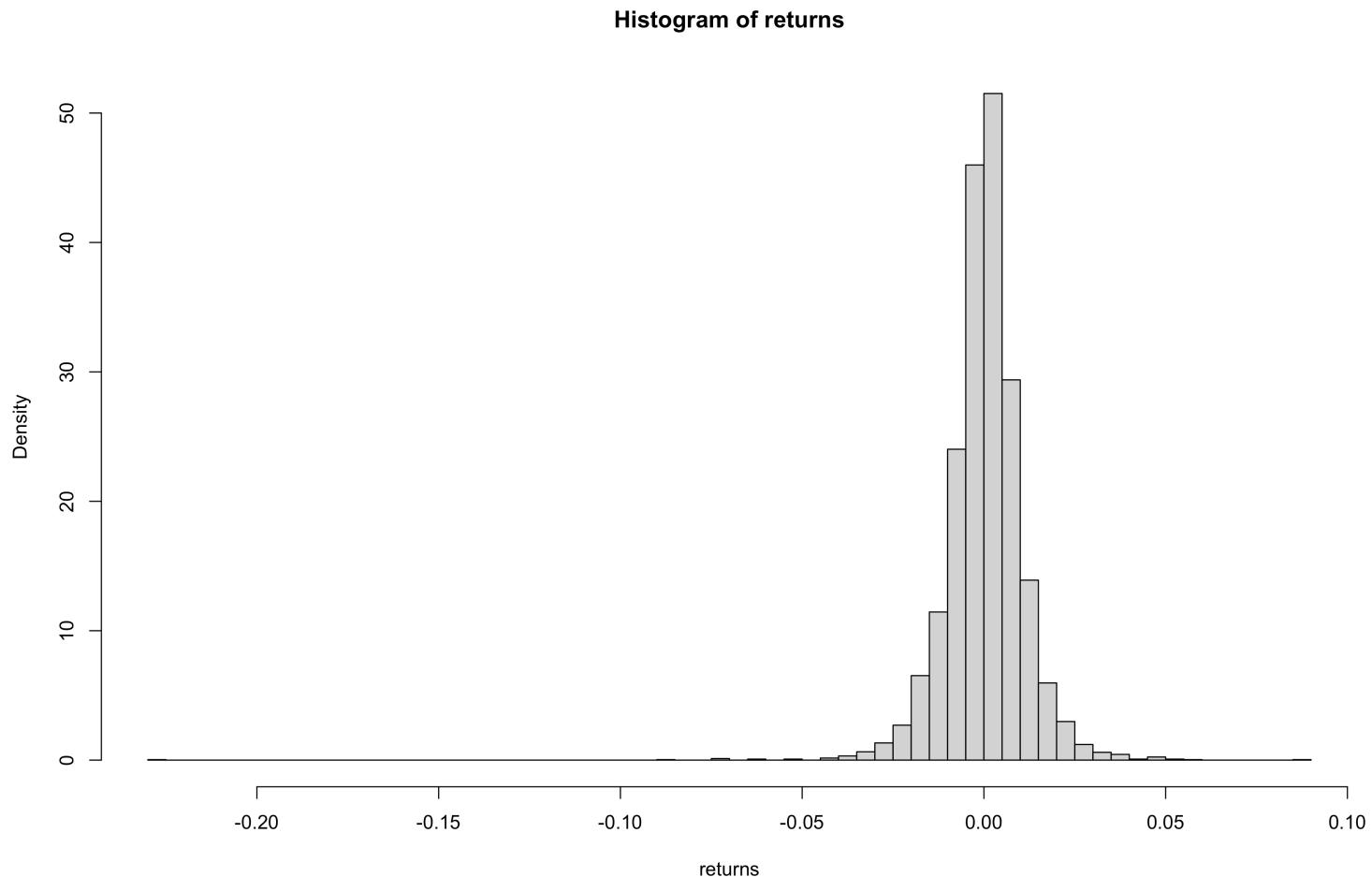
Conceptually, `quantile` and `ecdf` are inverses to each other

Getting Probability Densities from Data

`hist(x)` calculates a histogram from `x`

- divide the data range up into equal-width bins and *count* how many fall into each bin
- *Or* divide bin counts by (total count)*(width of bin), and get an estimate of the probability density function (pdf)
Produces plot as a default side-effect

```
hist(returns,n=101,probability=TRUE)
```



Probability Densities from Data (cont'd.)

`density(x)` estimates the density of x by counting how many observations fall in a little window around each point, and then smoothing

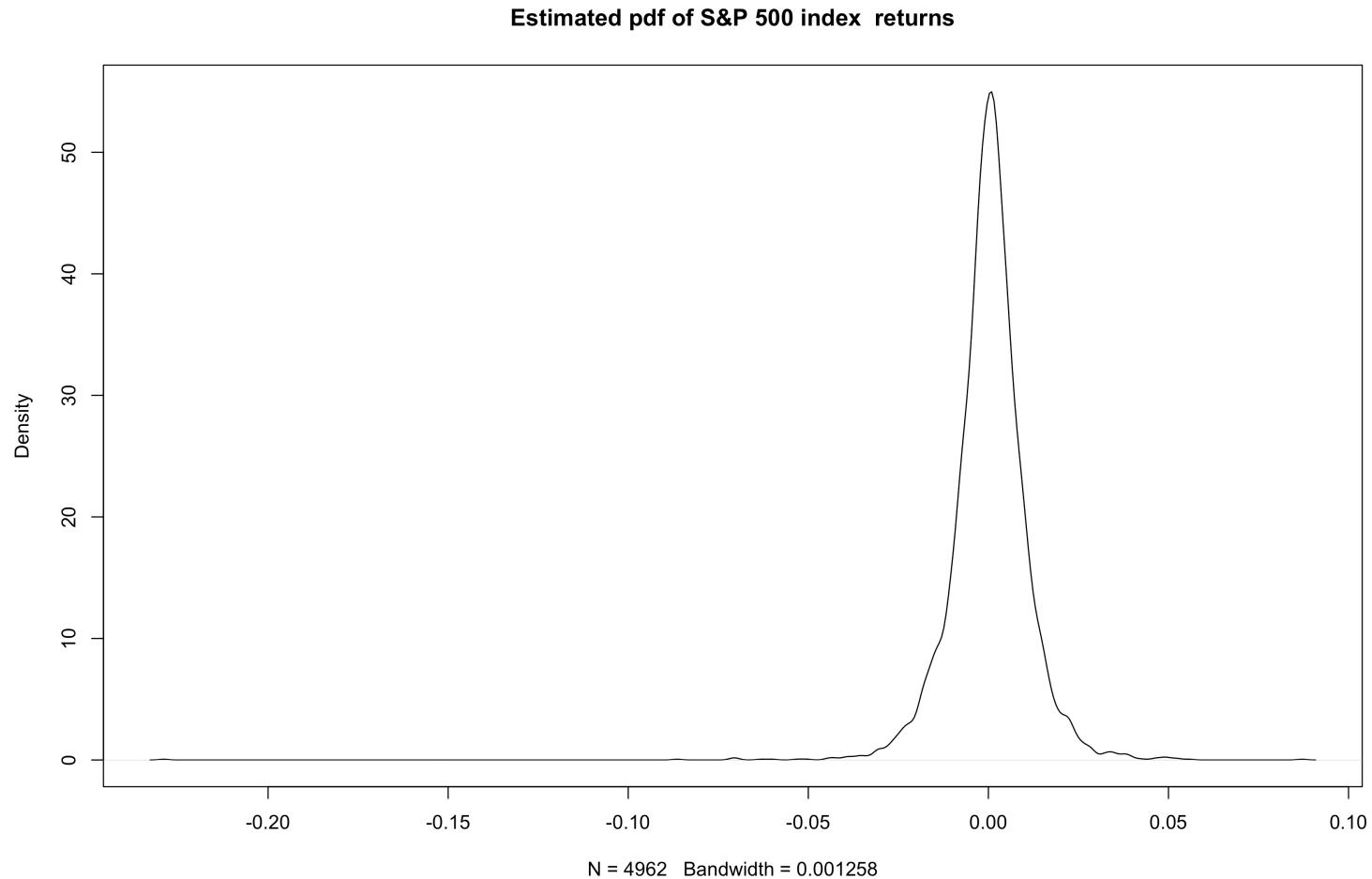
"Bandwidth" \approx width of window around each point

Technically, a "kernel density estimate"

Remember: `density()` is an *estimate* of the pdf, not The Truth

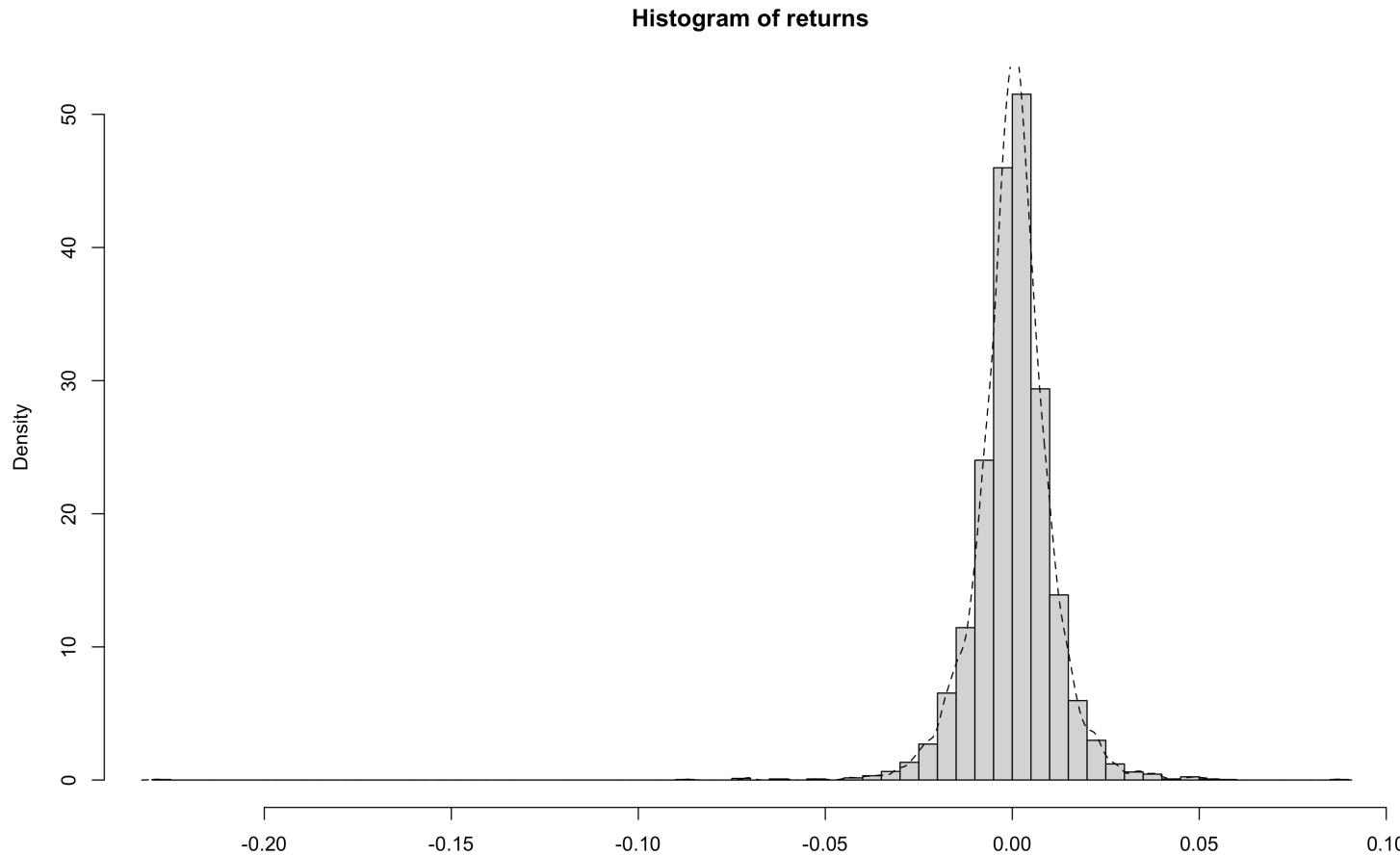
`density` returns a collection of x, y values, suitable for plotting

```
plot(density(returns),main="Estimated pdf of S&P 500 index returns")
```



Probability Densities from Data (cont'd.)

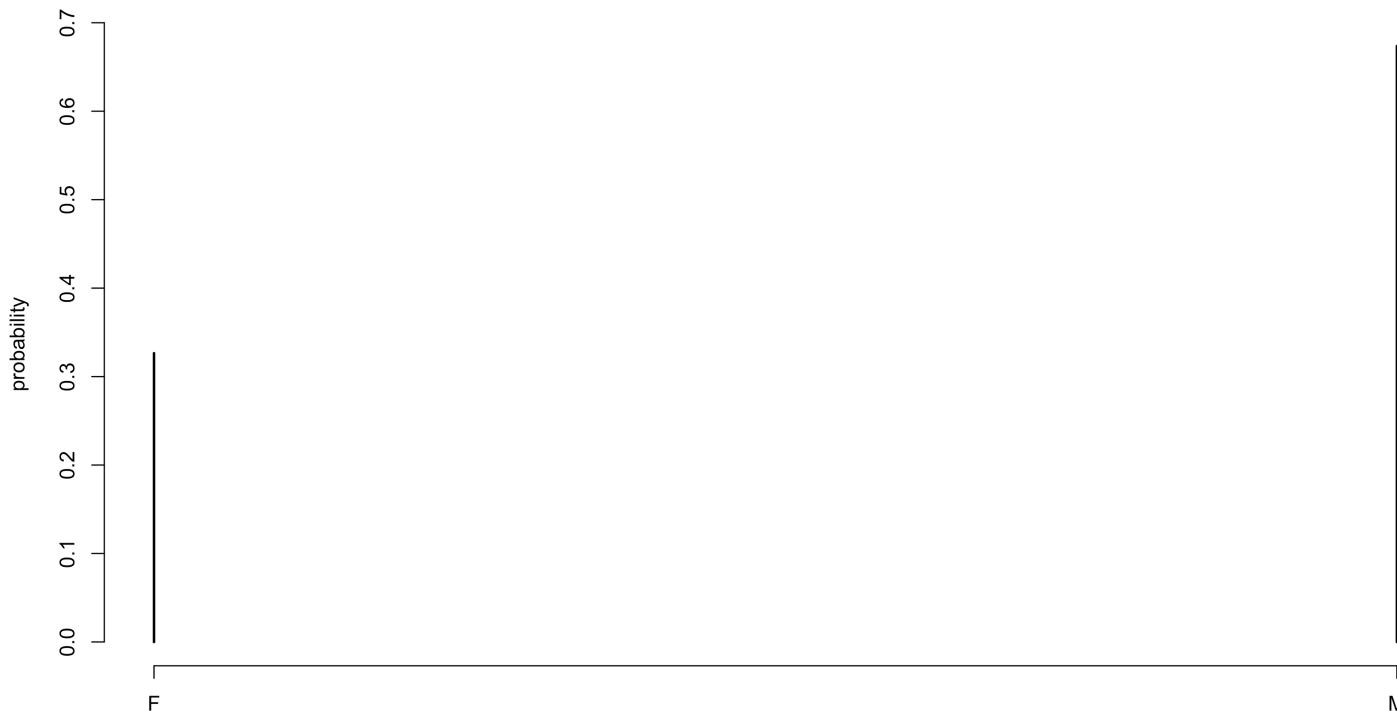
```
hist(returns,n=101,probability=TRUE)  
lines(density(returns),lty="dashed")
```



Getting distributions from data (cont'd.)

`table()` - tabulate outcomes, most useful for discrete spaces; remember to normalize if you want probabilities

```
plot(table(cats$Sex)/nrow(cats), ylab="probability")
```



Who Cares About the Distribution of the Data?

- Overly detailed: every single observation recorded as a separate tick
 - *Too much information*
- The exact set of samples would never repeat if we re-ran things anyway
 - *That information is wrong*
- Try to *summarize* what will *generalize* to other situations
 - Use a model, remember the model's parameters

R commands for distributions

- $dfoo$ = the probability density (if continuous) or probability mass function of foo (pdf or pmf)
- $pfoo$ = the cumulative probability function (CDF)
- $qfoo$ = the quantile function (inverse to CDF)
- $rfoo$ = draw r random numbers from foo (first argument always the number of draws)

?Distributions to see which distributions are built in

If you write your own, follow the conventions

Examples

```
dnorm(x=c(-1,0,1),mean=1,sd=0.1)
```

```
## [1] 5.520948e-87 7.694599e-22 3.989423e+00
```

```
pnorm(q=c(2,-2)) # defaults to mean=0, sd=1
```

```
## [1] 0.97724987 0.02275013
```

```
dbinom(5,size=7,p=0.7,log=TRUE)
```

```
## [1] -1.146798
```

```
qchisq(p=0.95,df=5)
```

```
## [1] 11.0705
```

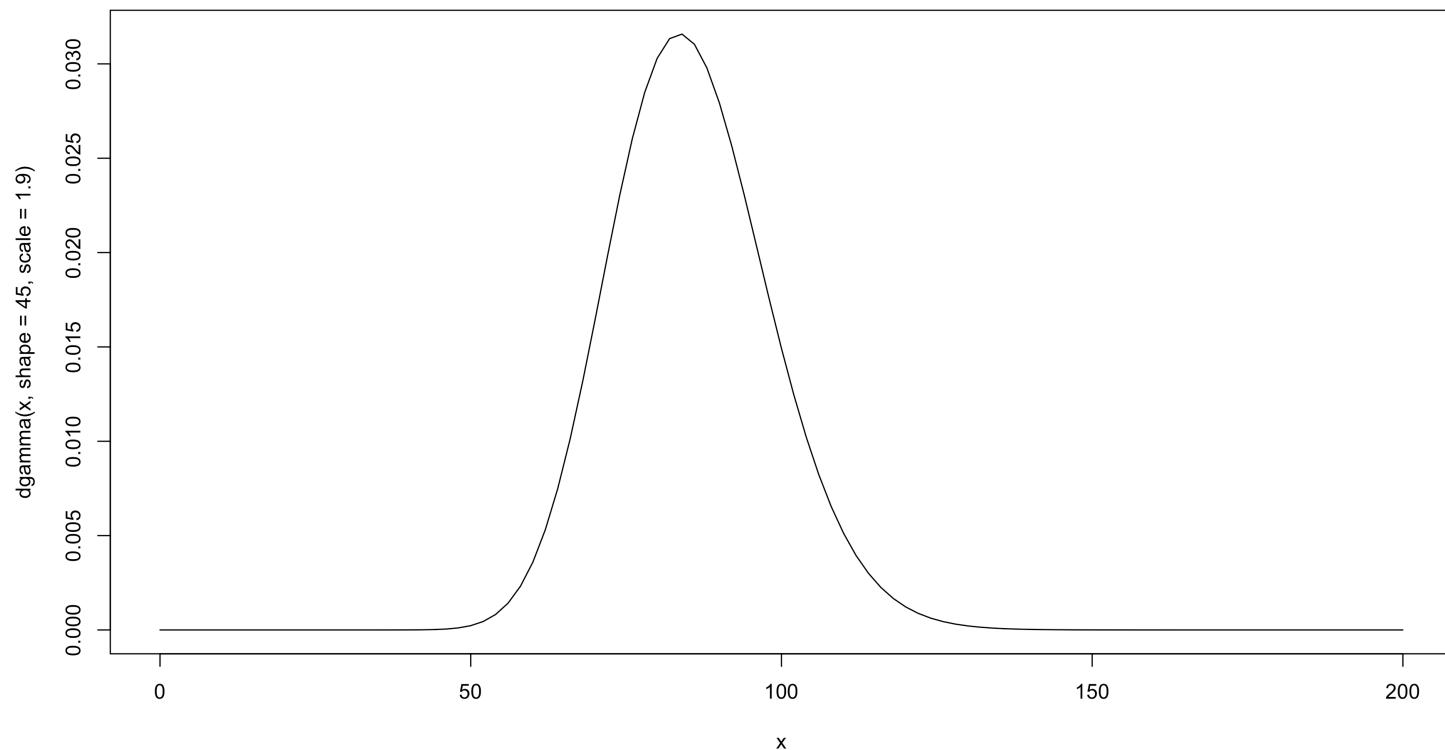
```
rt(n=4,df=2)
```

```
## [1] 0.4397369 1.1463390 -0.3783058 -1.3546069
```

Displaying Probability Distributions

curve is very useful for the d, p, q functions:

```
curve(dgamma(x,shape=45,scale=1.9),from=0,to=200)
```



How Do We Fit Distributional Models to the Data?

- Match moments (mean, variance, etc.)
- Match other summary statistics
- Maximize the likelihood

Method of Moments (MM), Closed Form

- Pick enough moments that they **identify** the parameters
 - At least 1 moment per parameter; algebraically independent
- Write equations for the moments in terms of the parameters
e.g., for gamma

$$\mu = as, \sigma^2 = as^2$$

- Do the algebra by hand to solve the equations

$$a = \mu^2 / \sigma^2, s = \sigma^2 / \mu$$

- Code up the formulas (did this in lab 3)

```
gamma.est_MM <- function(x) {  
  m <- mean(x); v <- var(x)  
  return(c(shape=m^2/v, scale=v/m))  
}
```

MM, Numerically

- Write functions to get moments from parameters (usually algebra)
- Set up the difference between data and model as another function

```
gamma.mean <- function(shape,scale) { return(shape*scale) }  
gamma.var <- function(shape,scale) { return(shape*scale^2) }  
gamma.discrepancy <- function(shape,scale,x) {  
  return((mean(x)-gamma.mean(shape,scale))^2 + (var(x)-gamma.var(shape,scale)) )  
}
```

- Minimize it

More Generally...

- Nothing magic about moments
- Match other data summaries, say the median
 - Or even more complicated things, like ratios of quantiles
 - You did this in lab
- If you can't solve exactly for parameters from the summaries, set up a discrepancy function and minimize it
 - You are doing this in the HW
- The summaries just have to converge on population values

Maximum Likelihood

- Usually we think of the parameters as fixed and consider the probability of different outcomes, $f(x; \theta)$ with θ constant and x changing
- **Likelihood** of a parameter value = $L(\theta)$ = what probability does θ give to the data?
 - For continuous variables, use probability density
 - $f(x; \theta)$ but letting θ change while data constant
 - *Not* the probability of θ , if that even makes sense
- **Maximum likelihood** = guess that the parameter is whatever makes the data most likely
- Most likely parameter value = **maximum likelihood estimate** = MLE

Likelihood in Code

- With independent data points x_1, x_2, x_n , likelihood is

$$L(\theta) = \prod_{i=1}^n f(x_i; \theta)$$

- Multiplying lots of small numbers is numerically bad; take the log:

$$\ell(\theta) = \sum_{i=1}^n \log f(x_i; \theta)$$

- In pseudo-code:

```
loglike.foo <- function(params, x) {  
  sum(dfoo(x=x, params, log=TRUE))  
}
```

What Do We Do with the Likelihood?

- We maximize it!
- Sometimes we can do the maximization by hand with some calculus
 - For Gaussian, MLE = just match the mean and variance
 - For Pareto, MLE $\hat{a} = 1 + \overline{\log(x/x_{\min})}$
- Doing numerical optimization
 - Stick in a minus sign if we're using a minimization function

Why Use the MLE?

- Usually (but not always) *consistent*: converges on the truth as we get more data
- Usually (but not always) *efficient*: converges on the truth at least as fast as anything else
- There are some parameters where the maximum isn't well-defined (e.g. x_{\min} for a Pareto)
- Sometimes the data is too aggregated or mangled to use the MLE (as with the income data in lab 5)

fitdistr

MLE for one-dimensional distributions can be done through `fitdistr` in the MASS package

It knows about most the standard distributions, but you can also give it arbitrary probability density functions and it will try to maximize them
A starting value for the optimization is optional for some distributions, required for others (including user-defined densities)

Returns the parameter estimates and standard errors
SEs come from large n approximations so use cautiously

fitdistr Examples

Fit the gamma distribution to the cats' hearts:

```
require(MASS)
fitdistr(cats$Hwt, densfun="gamma")
```

```
##          shape            rate
##    20.2998092    1.9095724
##  ( 2.3729252) ( 0.2259942)
```

Returns: estimates above, standard errors below

Fit the Students *t* distribution to the returns:

```
fitdistr(returns, "t")
```

```
##          m            s            df
##    0.0005103602  0.0071003759  3.5882142134
##  (0.0001206711) (0.0001290377) (0.2076614177)
```

Here parameters are location (m), scale (s) and degrees of freedom (very heavy tails)

fitdistr Examples (cont'd.)

User-defined density:

```
dpareto <- function(x,exponent,xmin,log=FALSE) {  
  f <- (exponent-1)/xmin * (x/xmin)^(-exponent)  
  f <- ifelse(x<xmin,NA,f)  
  if(!log) { return(f) } else (return(log(f)))  
}  
# Fit pareto to large absolute returns  
# Parameters given outside the "start" list are fixed  
fitdistr(abs(returns)[abs(returns)>0.05], densfun=dpareto,  
         start=list(exponent=2.5), xmin=0.05)
```

```
## Warning in stats:::optim(x = c(0.0529756421537337, 0.228997226565671, 0.051  
## use "Brent" or optimize() directly  
  
##      exponent  
##      3.9960938  
##      (0.8309668)
```

Checking Your Estimator

- simulate, then estimate; estimates should converge as the sample grows

```
gamma.est_MM(rgamma(100,shape=19,scale=45))
```

```
##      shape      scale  
## 20.72856 41.69062
```

```
gamma.est_MM(rgamma(1e5,shape=19,scale=45))
```

```
##      shape      scale  
## 19.00673 44.88559
```

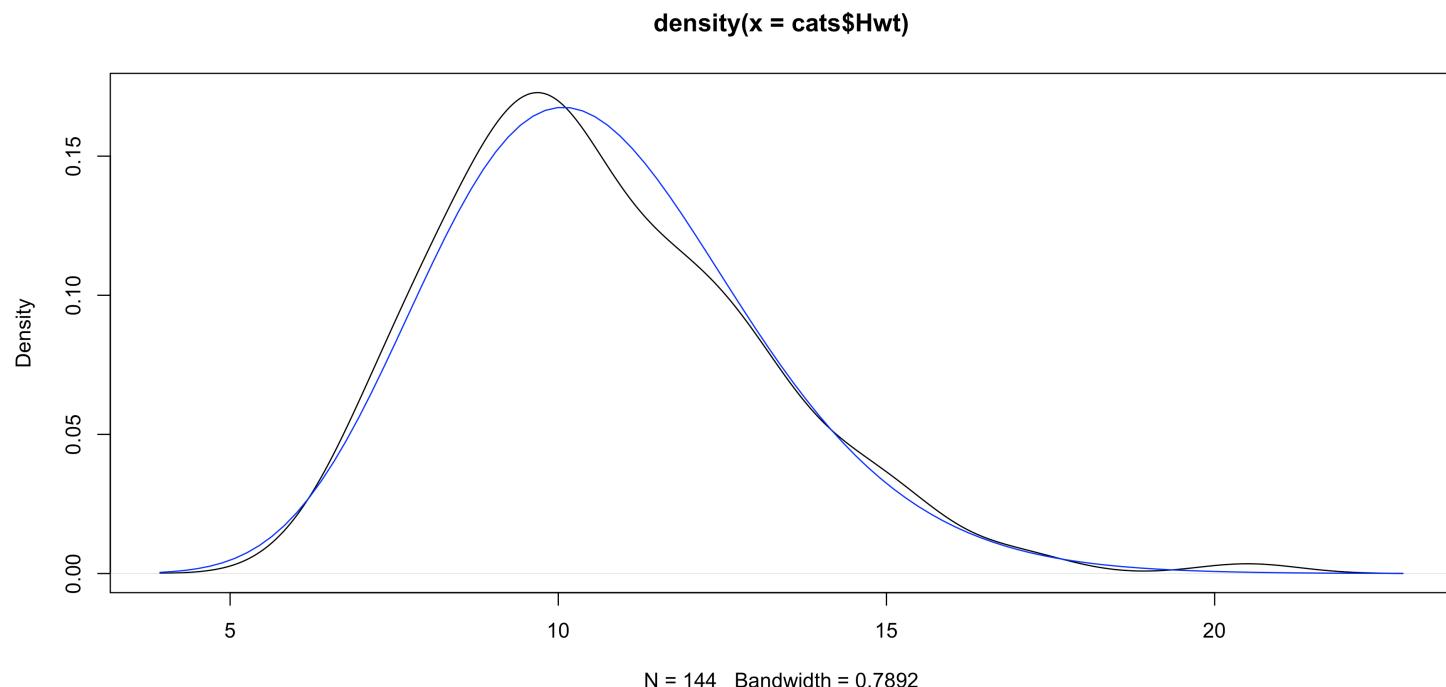
```
gamma.est_MM(rgamma(1e6,shape=19,scale=45))
```

```
##      shape      scale  
## 18.99762 45.01647
```

Checking the Fit

Use your eyes: Graphic overlays of theory vs. data

```
plot(density(cats$Hwt))
cats.gamma <- gamma.est_MM(cats$Hwt)
curve(dgamma(x,shape=cats.gamma["shape"],scale=cats.gamma["scale"]),a
```



Checking the Fit (cont'd.)

- Calculate summary statistics *not* used in fitting, compare them to the fitted model

```
# Really bad and good days for index fund holders, per model:  
qnorm(c(0.01,0.99),mean=mean(returns),sd=sd(returns))
```

```
## [1] -0.02487668  0.02555036
```

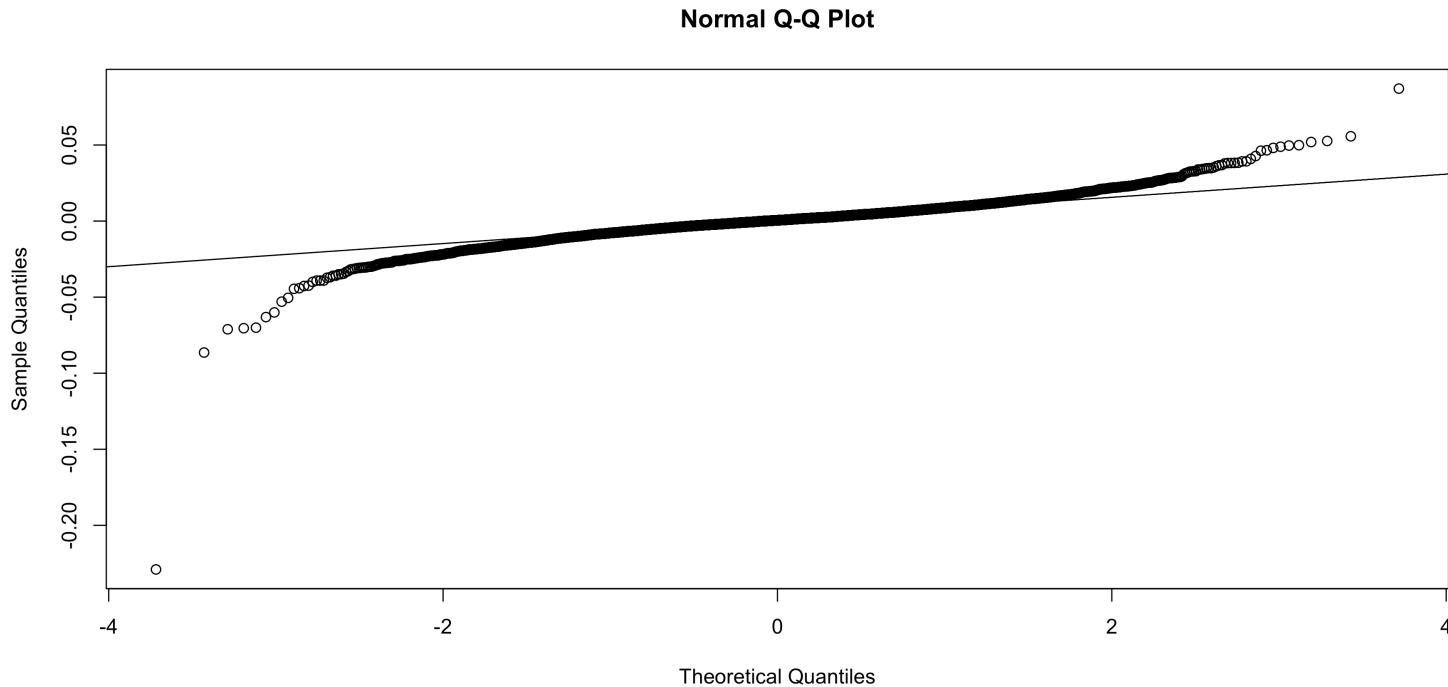
```
# As it happened:  
quantile(returns,c(0.01,0.99))
```

```
##           1%          99%  
## -0.02730130  0.02765495
```

Quantile-Quantile (QQ) Plots

- Plot theoretical vs. actual quantiles
- or plot quantiles of two samples against each other
- Ideally, a straight line when the distributions are the same
- `qqnorm`, `qqline` are specialized for checking normality

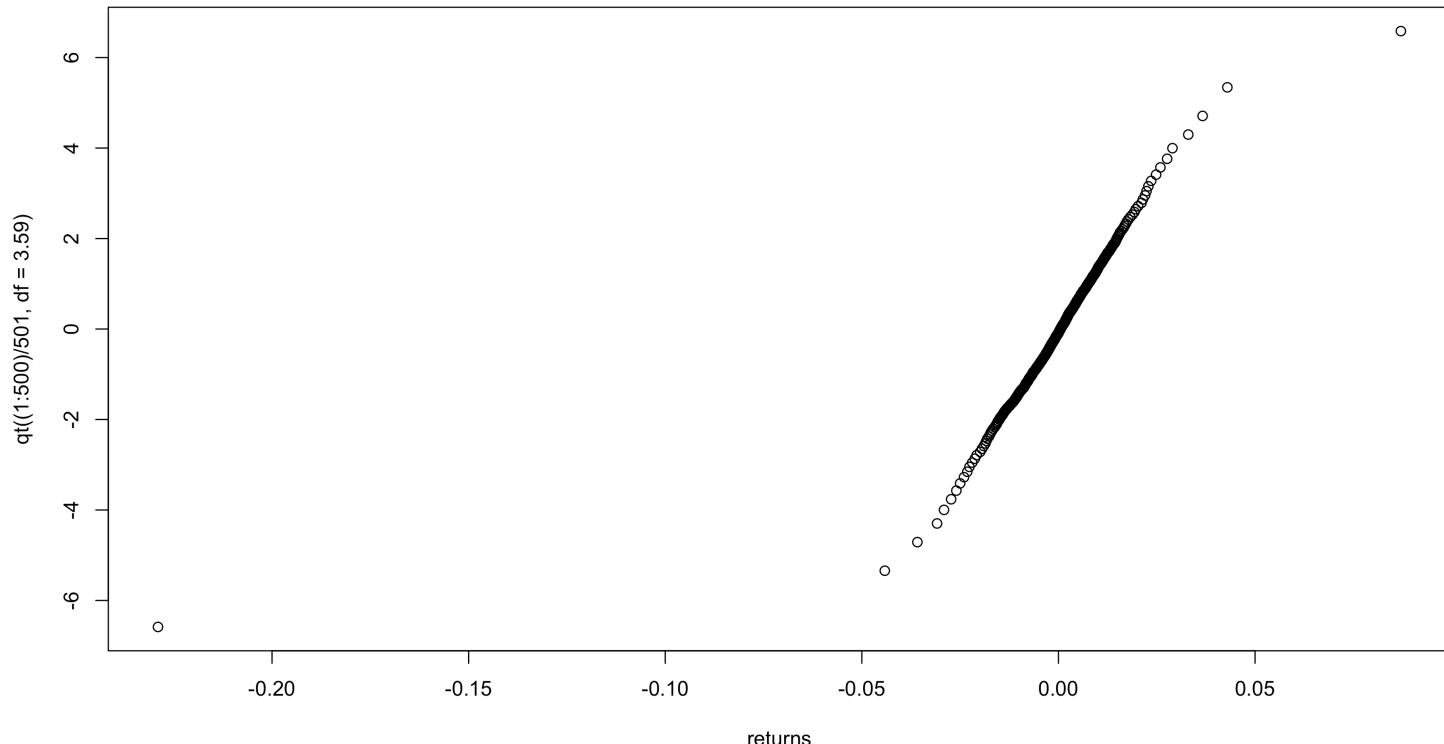
```
qqnorm(returns); qqline(returns)
```



QQ Plots (cont'd)

- `qqplot(x, y)` will do a Q-Q plot of one vector against another

```
qqplot(returns, qt((1:500)/501, df=3.59))
```

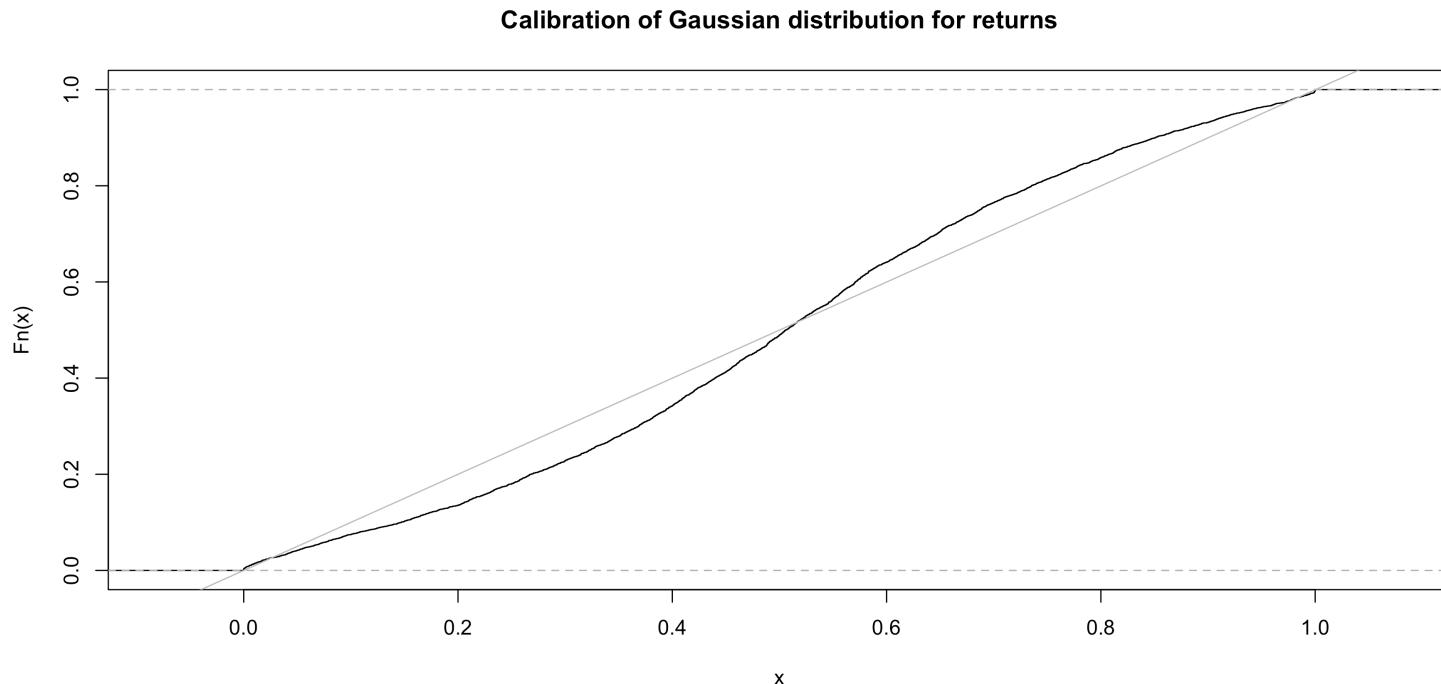


Calibration Plots

- If the distribution is right, 50% of the data should be below the median, 90% should be below the 90th percentile, etc.
- Special case of **calibration** of probabilities: events with probability $p\%$ should happen about $p\%$ of the time, not more and not less
- We can look at calibration by calculating the (empirical) CDF of the (theoretical) CDF and plotting
 - Ideal calibration plot is a straight line up the diagonal
 - Systematic deviations are a warning sign

Making a Calibration Plot

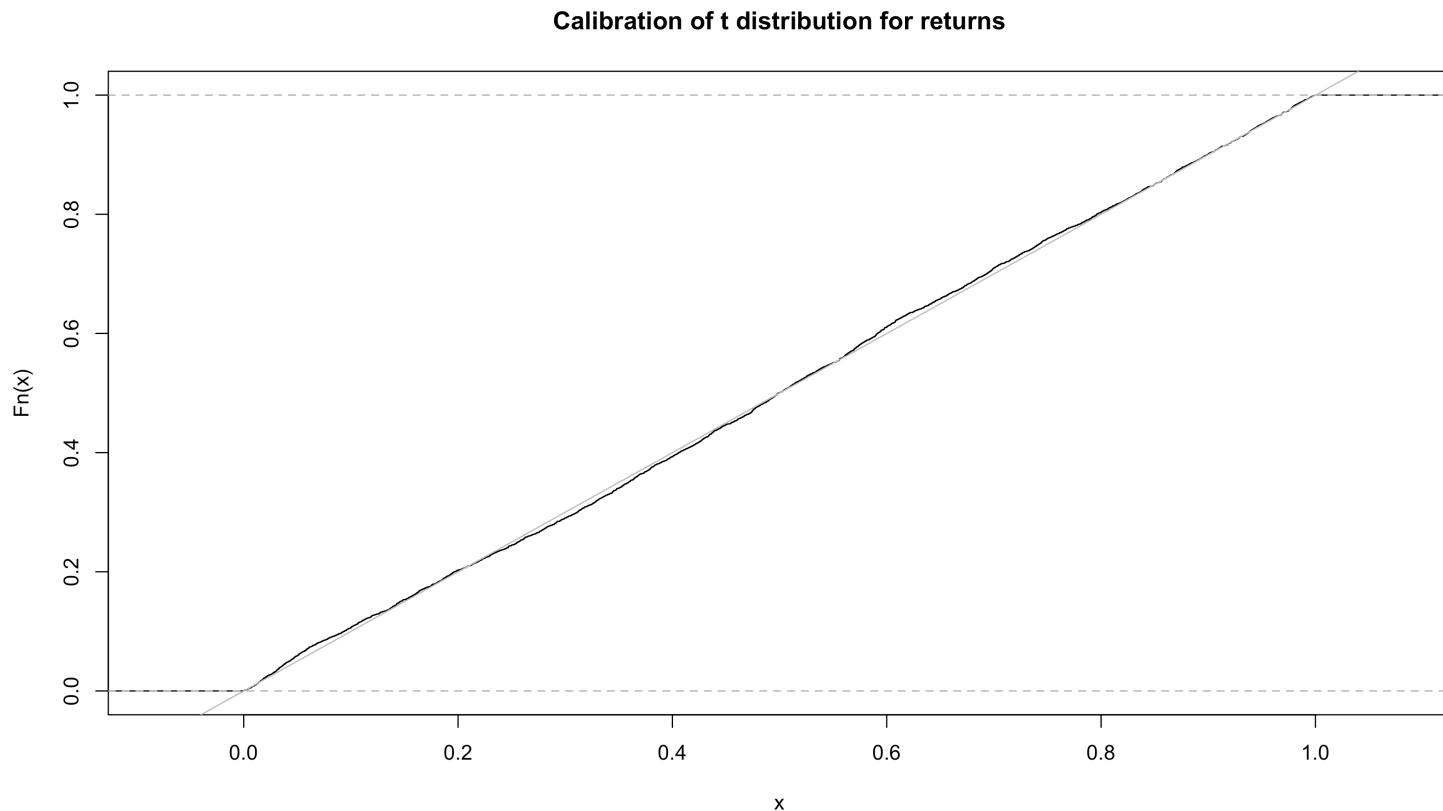
```
plot(ecdf(pnorm(returns, mean=mean(returns), sd=sd(returns))),  
      main="Calibration of Gaussian distribution for returns")  
abline(0,1,col="grey")
```



Again, way too many large changes (in either direction)

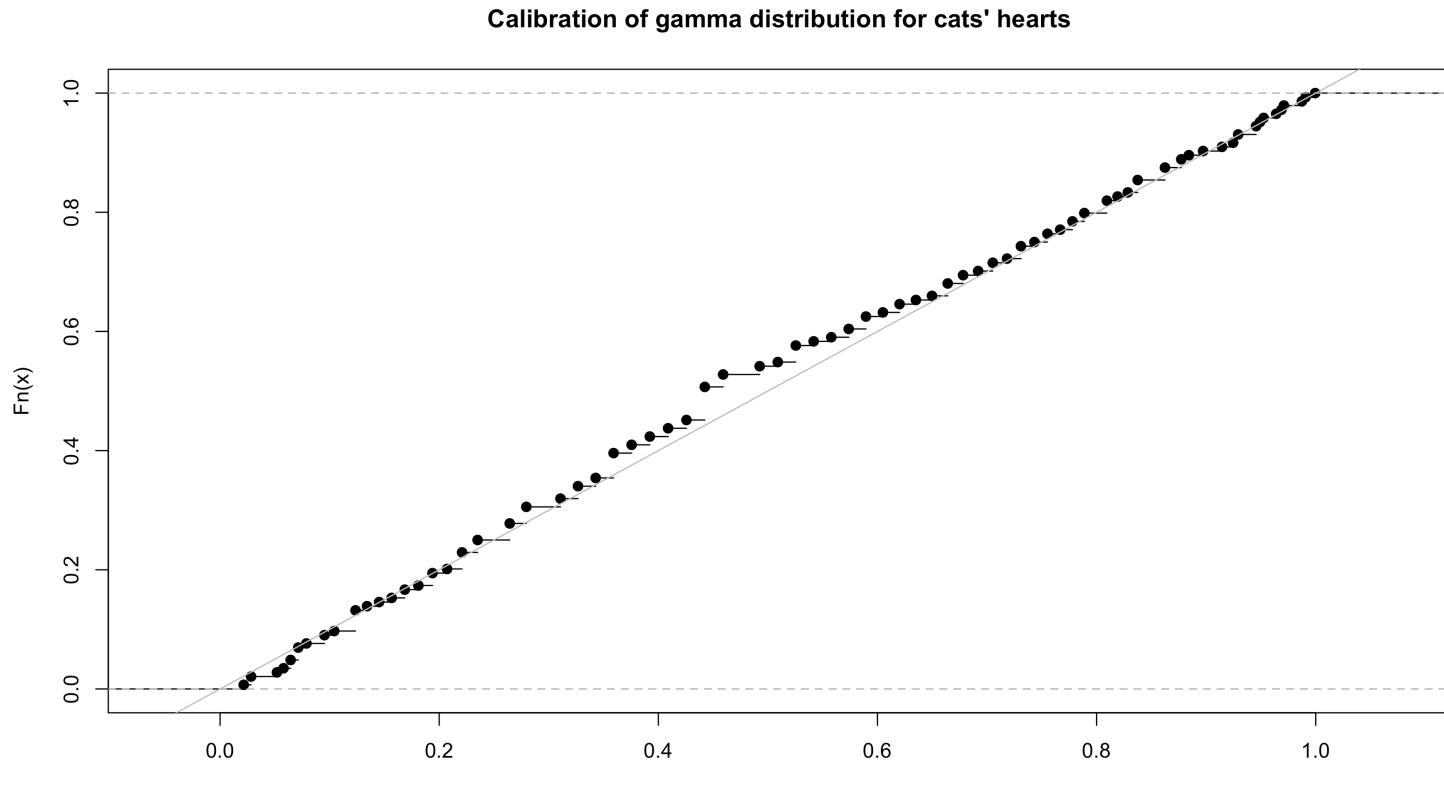
Calibration Plots (cont'd.)

```
SP.t <- coefficients(fitdistr(returns, "t"))
plot(ecdf(pt((returns-SP.t[1])/SP.t[2]), df=SP.t[3])),
     main="Calibration of t distribution for returns")
abline(0,1,col="grey")
```



Calibration Plots (cont'd.)

```
plot(ecdf(pgamma(cats$Hwt, shape=cats.gamma["shape"], scale=cats.gamr  
main="Calibration of gamma distribution for cats' hearts")  
abline(0,1,col="grey")
```



Calibration Plots (cont'd.)

Challenge: Write a general function for making a calibration plot, taking a data vector, a cumulative probability function, and a parameter vector

Kolmogorov-Smirnov Test

- How much should the QQ plot or calibration plot wiggle around the diagonal?
- Answer a different question...
- Biggest gap between theoretical and empirical CDF:

$$D_{KS} = \max_x |F(x) - \hat{F}(x)|$$

- Useful because D_{KS} always has the same distribution *if* the theoretical CDF is fixed and correct, and K+S calculated this back in the day
- Also works for comparing the empirical CDFs of two samples, to see if they came from the same distribution

KS Test, Data vs. Theory

```
ks.test(returns, pnorm, mean=0, sd=0.0125)
```

```
##  
##      Asymptotic one-sample Kolmogorov-Smirnov test  
##  
## data: returns  
## D = 0.10893, p-value < 2.2e-16  
## alternative hypothesis: two-sided
```

- More complicated (and *not* properly handled by built-in R) if parameters are estimated
 - Estimating parameters makes the fit look *better* than it really is, so it doesn't help save the model when it gets really rejected (like this one is)

Hack: Estimate using (say) 90% of the data, and then check the fit on the remaining 10%

```
train <- sample(1:length(returns), size=round(0.9*length(returns)))
SP.t_train <- coefficients(fitdistr(returns[train], "t"))
returns.test_standardized <- (returns[-train]-SP.t_train[1])/SP.t_train[3]
ks.test(returns.test_standardized, pt, df=SP.t_train[3])
```

```
##  
##      Asymptotic one-sample Kolmogorov-Smirnov test  
##  
## data: returns.test_standardized  
## D = 0.037817, p-value = 0.4772  
## alternative hypothesis: two-sided
```

- Can also test whether two samples come from same distribution

```
n <- length(returns)
half <- round(n/2)
ks.test(returns[1:half], returns[(half+1):n])
```

```
##  
##      Asymptotic two-sample Kolmogorov-Smirnov test  
##  
## data: returns[1:half] and returns[(half + 1):n]
## D = 0.099154, p-value = 5.103e-11
## alternative hypothesis: two-sided
```

Chi-Squared Test for Discrete Distributions

Compare an actual table of counts to a hypothesized probability distribution
e.g., as many up days as down?

```
up_or_down <- ifelse(returns > 0, 1, -1)
# 1936 down days, 1772 up days
chisq.test(table(up_or_down), p=c(1/2, 1/2))
```

```
## Chi-squared test for given probabilities
## data: table(up_or_down)
## X-squared = 20.896, df = 1, p-value = 4.85e-06
```

Chi-Squared Test: Degrees of Freedom

- The p -value calculated by `chisq.test` assumes that all the probabilities in p were *fixed*, not estimated from the data used for testing, so $df = \text{number of cells in the table} - 1$
- If we estimate q parameters, we need to subtract q degrees of freedom

Chi-Squared Test for Continuous Distributions

- Divide the range into bins and count the number of observations in each bin; this will be `x` in `chisq.test()`
- Use the CDF function `p foo` to calculate the theoretical probability of each bin; this is `p`
- Plug in to `chisq.test`
- If parameters are estimated, adjust
- `hist()` gives us break points and counts:

```
cats.hist <- hist(cats$Hwt, plot=FALSE)
cats.hist$breaks
```

```
## [1] 6 8 10 12 14 16 18 20 22
```

```
cats.hist$counts
```

```
## [1] 20 45 42 23 11 2 0 1
```

Chi-Squared for Continuous Data (cont'd.)

Use these for a χ^2 test:

```
# Why the padding by -Inf and Inf?  
p <- diff(pgamma(c(-Inf,cats.hist$breaks,Inf),shape=cats.gamma["shape"],  
           scale=cats.gamma["scale"]))  
# Why the padding by 0 and 0?  
x2 <- chisq.test(c(0,cats.hist$counts,0),p=p)$statistic  
# Why +2? Why -length(cats.gamma)?  
pchisq(x2,df=length(cats.hist$counts)+2 - length(cats.gamma))
```

```
## X-squared  
## 0.854616
```

Don't need to run `hist` first; can also use `cut` to discretize (see `?cut`)

- This is all a bit old-school
 - Loss of information from discretization
 - Lots of work just to use χ^2
- Try e.g. `ks.test` with an independent test set

Summary

- Visualizing and computing empirical distribution
- Parametric distributions are models
- Methods of fitting: moments, generalized moments, likelihood
- Methods of checking: visual comparisons, other statistics, tests, calibration

Aside: Some Math for MM and GMM

- Parameter θ is a p -dimensional vector, true value = θ^*
- Introduce $q \geq p$ **functionals** g_1, \dots, g_q , which we can calculate either from the parameter θ or from the data $x_{1:n}$
- Assume that for each i , $g_i(x_{1:n}) \rightarrow g_i(\theta^*)$
- Define

$$\hat{\theta}_{GMM} = \operatorname{argmin}_{\theta} \sum_{i=1}^q (g_i(\theta) - g_i(x_{1:n}))^2$$

Math for MM and GMM (cont'd.)

- Shouldn't be hard to believe that $\hat{\theta}_{GMM} \rightarrow \theta^*$
- But why give equal attention to every functional?
 - More weight on the more-precisely-measured functionals
 - More weight on the more-sensitive-to θ functionals
 - Less weight on partially-redundant functionals
- Abbreviate $g(\theta)$ for $(g_1(\theta), \dots, g_q(\theta))$, and likewise $g(x_{1:n})$, so

$$\hat{\theta}_{GMM} = \operatorname{argmin}_{\theta} (g(\theta) - g(x_{1:n}))^T (g(\theta) - g(x_{1:n}))$$

- Generalize by introducing any positive-definite matrix Ω :

$$\hat{\theta}_{GMM} = \operatorname{argmin}_{\theta} (g(\theta) - g(x_{1:n}))^T \Omega (g(\theta) - g(x_{1:n}))$$

- Optimal Ω turns out to be the variance matrix of $g(\theta^*)$
- Iterative approximation: start with no weighting, estimate that variance matrix, re-do the estimate with weights, etc.

Aside: Some Math for the MLE

- More convenient to work with the mean log likelihood:

$$\Lambda(\theta) = \frac{1}{n} \sum_{i=1}^n \log f(X_i; \theta)$$

- This is a sample average so the law of large numbers applies:

$$\Lambda(\theta) \rightarrow \mathbf{E}[\Lambda(\theta)] = \lambda(\theta)$$

- The true parameter has higher average log-likelihood than anything else: if $\theta \neq \theta^*$

$$\theta \neq \theta^* \Rightarrow \lambda(\theta) < \lambda(\theta^*)$$

- Some extra conditions are needed for

$$\hat{\theta}_{MLE} \rightarrow \theta^*$$