StatR 101: Winter 2013

Homework 02

Rod Doe

Sunday, February 3, 2013 (Super Bowl Sunday)

Cool Trick of the Week

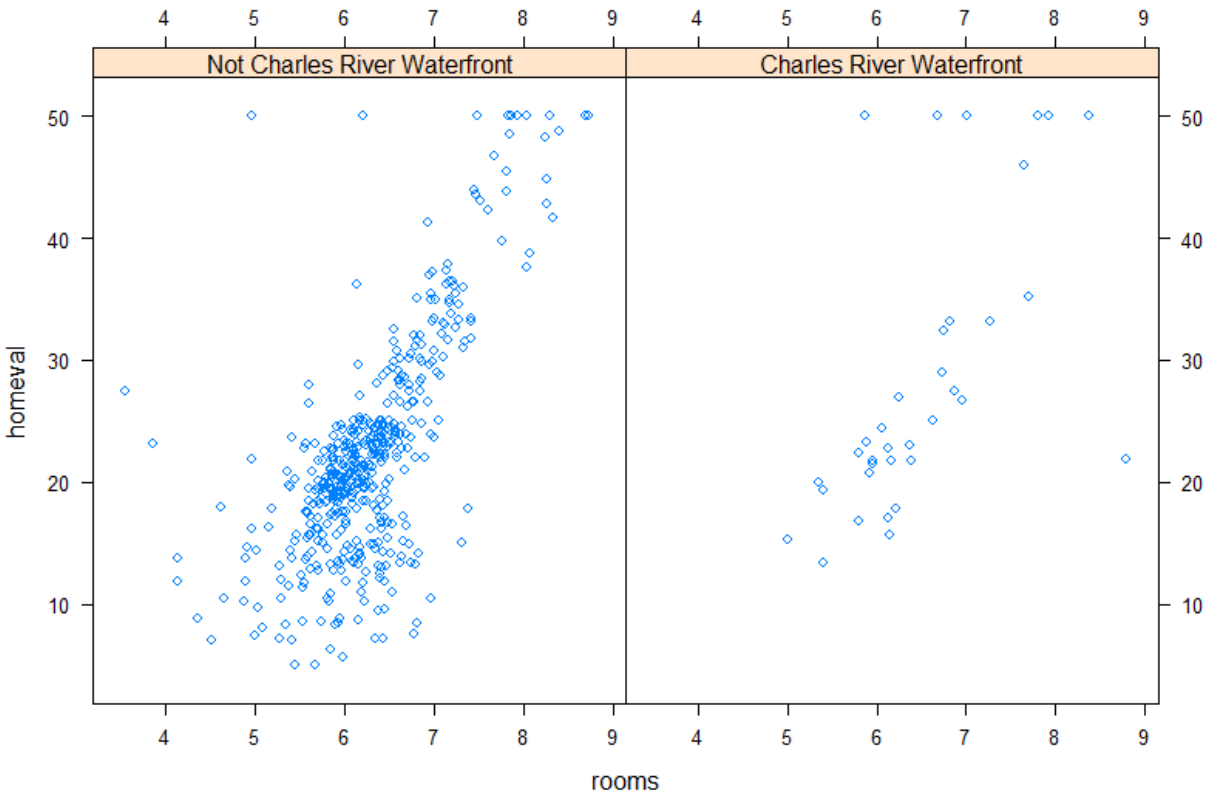
This appears in the graph for Metro on a Snowy Day problem. Here is the code:

```
buswait = function(x) { dexp(x, rate=(1/35)) }
curve(buswait, 0, 120,
        main=bquote(paste("Exponential distribution with estimated rate:  ", hat(lambda), " = 1/35" ) ),
        xlab="Waiting time [Minutes]",
        ylab="Probability of bus arrival" )
```

Annoyance of the Week

Thanks to Alan Bruce for this one!

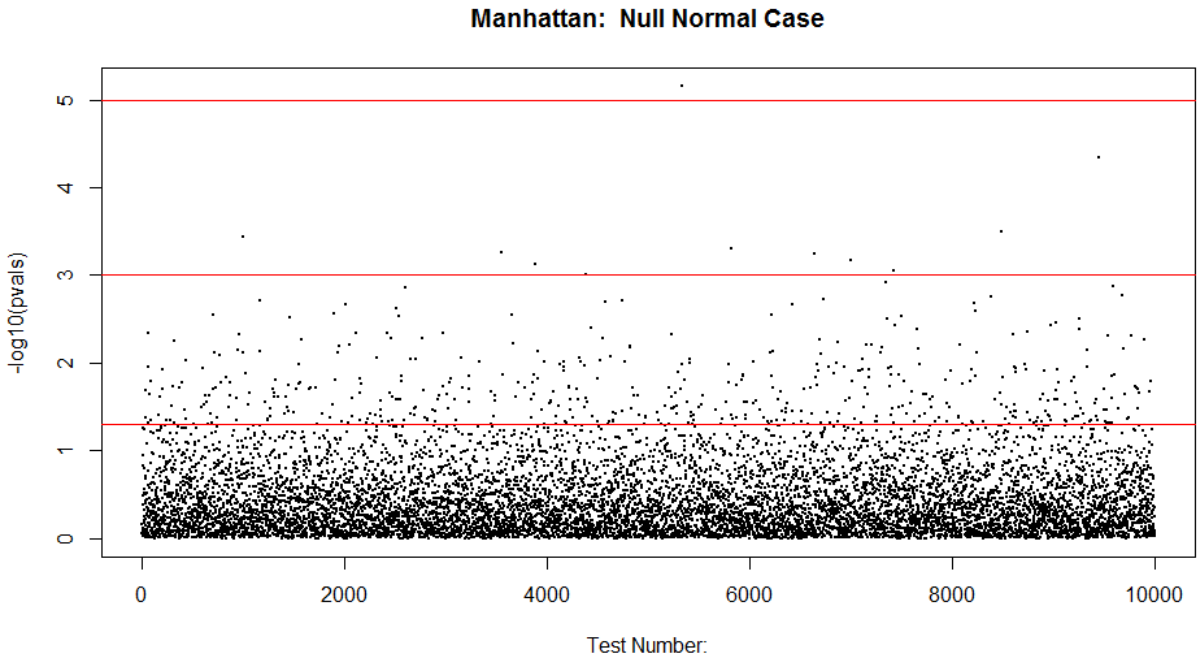
```
require(lattice)

xyplot(homeval ~ rooms | factor(river, labels = c("Not Charles River Waterfront", "Charles River
Waterfront")), pch = 1, data=boston.reduced, scales = list(alternating = 3))
```



Linear Regression Simulation

Changing the sample size from 100 to 10 had the largest effect.

Sample Size	**Y = Random Noise**	**N < 0.05**
100	Norm	50
100	t/sqrt(2)	50
100	exp	41
100	cauchy/2	50
10	Norm	9
10	t/sqrt(2)	3
10	exp	4
10	cauchy/2	3




```
x=matrix(rnorm(10000), nrow=10)
y=matrix(rnorm(10000), nrow=10)
betas=mapply(function(z,w) {summary(lm(w~z))$coef[2,]}, split(x,col(x)),split(y,col(y)))
b = betas[4,]
length(which(b < 0.05))
[1] 50
#x=matrix(rt(10000, 4)/sqrt(2), nrow=10)
y=matrix(rt(10000, 4)/sqrt(2), nrow=10)
betas=mapply(function(z,w) {summary(lm(w~z))$coef[2,]}, split(x,col(x)),split(y,col(y)))
b = betas[4,]
length(which(b < 0.05))
[1] 50
#x=matrix(rexp(10000, 4), nrow=10)
y=matrix(rexp(10000, 4), nrow=10)
betas=mapply(function(z,w) {summary(lm(w~z))$coef[2,]}, split(x,col(x)),split(y,col(y)))
b = betas[4,]
length(which(b < 0.05))
[1] 41
#x=matrix(rcauchy(10000)/2, nrow=10)
y=matrix(rcauchy(10000)/2, nrow=10)
betas=mapply(function(z,w) {summary(lm(w~z))$coef[2,]}, split(x,col(x)),split(y,col(y)))
b = betas[4,]
length(which(b < 0.05))
```

```
[1] 50

x=matrix(rnorm(10000), nrow=100)
y=matrix(rnorm(10000), nrow=100)
betas=mapply(function(z,w) {summary(lm(w~z))$coef[2,]}, split(x,col(x)),split(y,col(y)))
b = betas[4,]
length(which(b < 0.05))
[1] 9
#x=matrix(rt(10000, 4)/sqrt(2), nrow=100)
y=matrix(rt(10000, 4)/sqrt(2), nrow=100)
betas=mapply(function(z,w) {summary(lm(w~z))$coef[2,]}, split(x,col(x)),split(y,col(y)))
b = betas[4,]
length(which(b < 0.05))
[1] 3
#x=matrix(rexp(10000, 4), nrow=100)
y=matrix(rexp(10000, 4), nrow=100)
betas=mapply(function(z,w) {summary(lm(w~z))$coef[2,]}, split(x,col(x)),split(y,col(y)))
b = betas[4,]
length(which(b < 0.05))
[1] 4
#x=matrix(rcauchy(10000)/2, nrow=100)
y=matrix(rcauchy(10000)/2, nrow=100)
betas=mapply(function(z,w) {summary(lm(w~z))$coef[2,]}, split(x,col(x)),split(y,col(y)))
b = betas[4,]
length(which(b < 0.05))
[1] 3
```

Now we add the noise onto a 2X signal.

Sample Size	**Y = Signal + noise**	**Power**	**Bias**	**Interval Coverage Violations**
10	2x + Norm	0.978	-0.006	0
10	2x + t/sqrt(2)	0.966	0.005	1
10	2x + Exp	1	0.002	0
10	2x + cauchy/2	0.64	0.22	59
100	2x + Norm	1	-0.007	0
100	2x + t/sqrt(2)	1	0.001	
100	2x + Exp	1	-0.002	0
100	2x + cauchy/2	0.69	0.36	18

```
> x=matrix(rnorm(10000), nrow=10)
> y=2*x + matrix(rnorm(10000), nrow=10)
```

```
> betas=mapply(function(z,w) {summary(lm(w~z))$coef[2,]}, split(x,col(x)),split(y,col(y)))
> estimates = betas[1, ]
> stdErrs = betas[2, ]
> tvals = betas[3, ]
> pvals = betas[4, ]
> pwr = length(which(tvals > qt(0.975, 8))) / length(tvals)
> bias = mean(estimates) - 2
> nTooLow = length(which(estimates + qt(0.975, 8) < 2))
> nTooHigh = length(which(estimates - qt(0.975, 8) > 2))
> (pwr)
[1] 0.978
> (bias)
[1] -0.006979612
> (nTooLow)
[1] 0
> (nTooHigh)
[1] 0
>
> #x=matrix(rt(10000, 4)/sqrt(2), nrow=10)
> y=2*x + matrix(rt(10000, 4)/sqrt(2), nrow=10)
> betas=mapply(function(z,w) {summary(lm(w~z))$coef[2,]}, split(x,col(x)),split(y,col(y)))
> estimates = betas[1, ]
> stdErrs = betas[2, ]
> tvals = betas[3, ]
> pvals = betas[4, ]
> pwr = length(which(tvals > qt(0.975, 8))) / length(tvals)
> bias = mean(estimates) - 2
> nTooLow = length(which(estimates + qt(0.975, 8) < 2))
> nTooHigh = length(which(estimates - qt(0.975, 8) > 2))
> (pwr)
[1] 0.966
> (bias)
[1] 0.004715597
> (nTooLow)
[1] 0
> (nTooHigh)
[1] 1
>
> #x=matrix(rexp(10000, 4), nrow=10)
> y=2*x + matrix(rexp(10000, 4), nrow=10)
> betas=mapply(function(z,w) {summary(lm(w~z))$coef[2,]}, split(x,col(x)),split(y,col(y)))
> estimates = betas[1, ]
```

```
> stdErrs = betas[2, ]
> tvals = betas[3, ]
> pvals = betas[4, ]
> pwr = length(which(tvals > qt(0.975, 8))) / length(tvals)
> bias = mean(estimates) - 2
> nTooLow = length(which(estimates + qt(0.975, 8) < 2))
> nTooHigh = length(which(estimates - qt(0.975, 8) > 2))
> (pwr)
[1] 1
> (bias)
[1] 0.00269087
> (nTooLow)
[1] 0
> (nTooHigh)
[1] 0
>
> #x=matrix(rcauchy(10000)/2, nrow=10)
> y=2*x + matrix(rcauchy(10000)/2, nrow=10)
> betas=mapply(function(z,w) {summary(lm(w~z))$coef[2,]}, split(x,col(x)),split(y,col(y)))
> estimates = betas[1, ]
> stdErrs = betas[2, ]
> tvals = betas[3, ]
> pvals = betas[4, ]
> pwr = length(which(tvals > qt(0.975, 8))) / length(tvals)
> bias = mean(estimates) - 2
> nTooLow = length(which(estimates + qt(0.975, 8) < 2))
> nTooHigh = length(which(estimates - qt(0.975, 8) > 2))
> (pwr)
[1] 0.64
> (bias)
[1] 0.2182648
> (nTooLow)
[1] 59
> (nTooHigh)
[1] 58
```

The Metro on a Snowy Day Problem

The scenario here is that I arrive at my bus stop. The bus is obviously off schedule, so I ask the regulars how long they've been waiting. Given their input, I calculate the most likely estimate of how long we will need to wait for the bus, or, more realistically, based upon my experience, when to start to walk home. Luckily, it's only about 7 miles, and I've done it many times on nice days.

Modeled solution

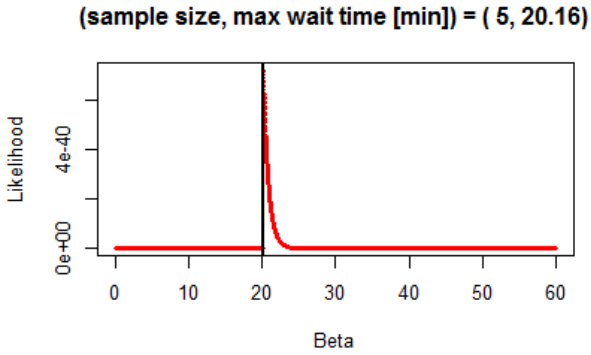
The interesting part here is that the maximum wait time is likely to increase as a function of sample size. I think that this is due to the fact that with a larger sample size, it is more probable to get a larger maximum waiting time.

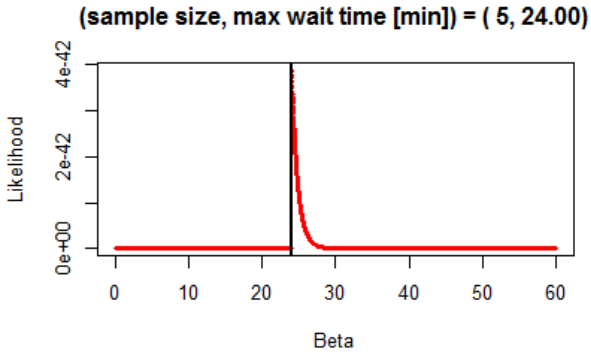
Here is the essential R code:

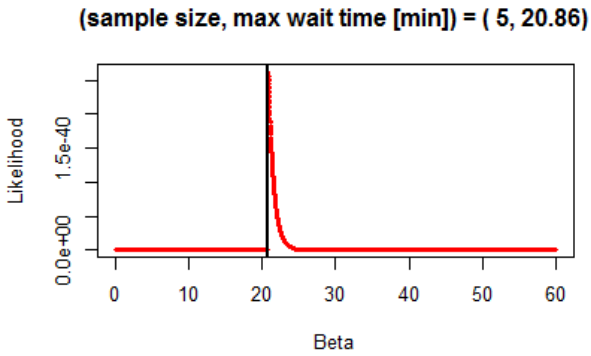
```
par(mfrow=c(2,2))

for (i in 1:4) {
        # Beta is the period [minutes] between consecutive buses
        # as they leave the base.  In my case, that is about 30 minutes.
        beta = 30

        # Let's assume that there are n fellow passengers waiting.
        nWaiting = 5  # AKA sample size
        y = runif(nWaiting, 0, beta)

        # Generate data on the x axis up to two intervals of beta.
        x = seq(0, 2 * beta, 0.01)

        # Plot all data less than the maximum wait experienced as 0.  Why?
        # Because if the bus came before then, all the passengers would be on it.
        plot(x, ifelse( x < max(y), 0, x^(-beta)), pch=19, col=2, cex=0.5, xlab='Beta', ylab='Likelihood',
                main=sprintf("(sample size, max wait time [min]) = (%2d, %5.2f)", nWaiting, max(y)))
        abline(v = max(y), lwd = 2)
}
```

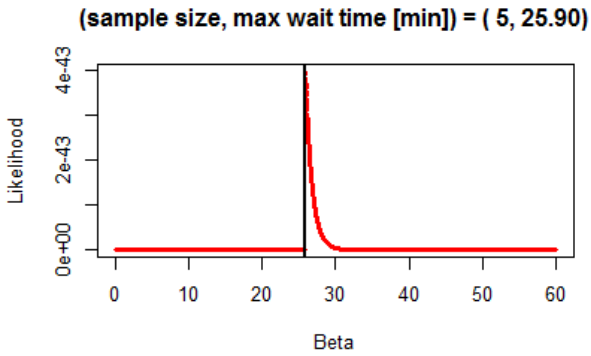
The graphs from simulations with sample size of (5, 25, 125) appear below:

For sample size = 5

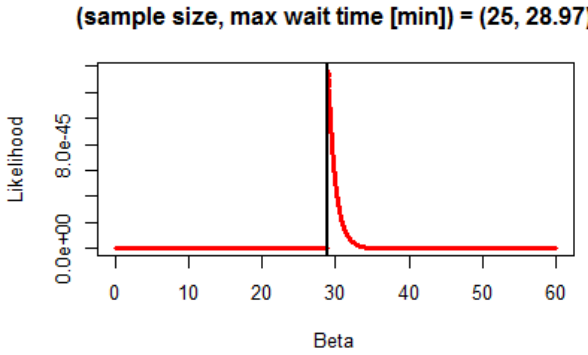


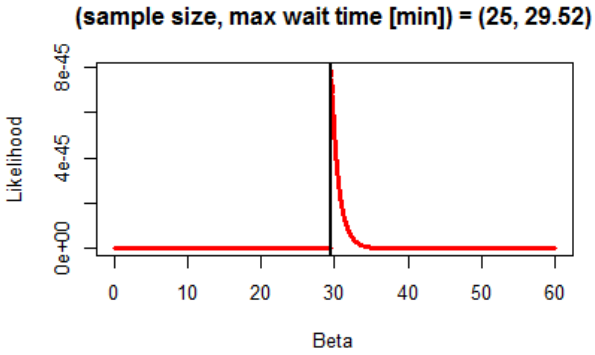


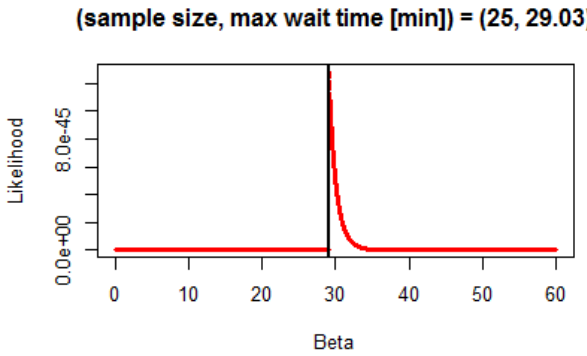


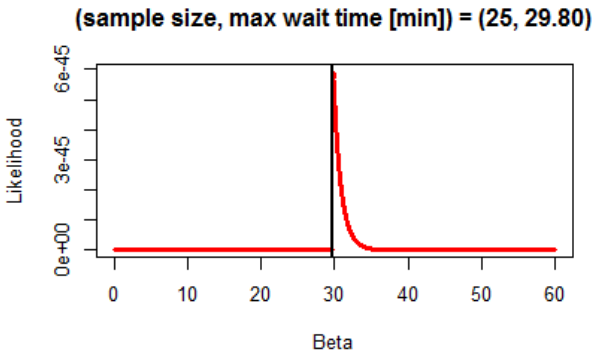


For sample size = 25

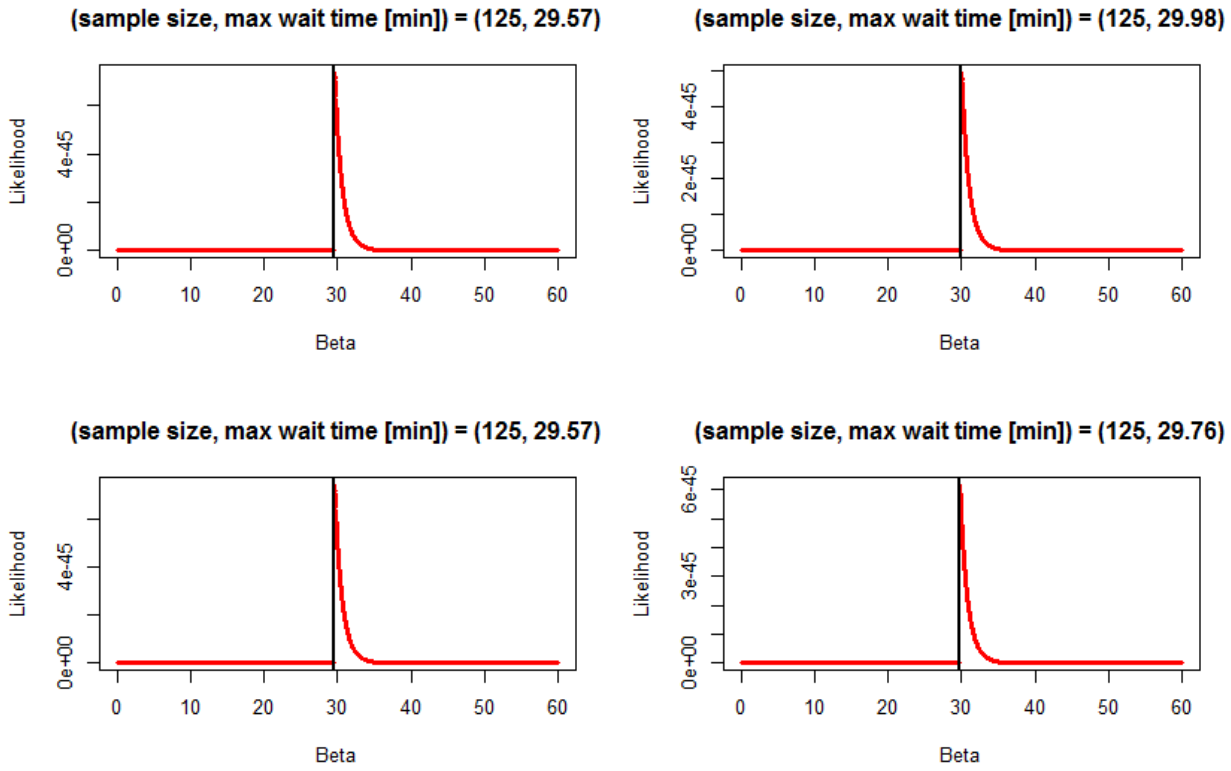








For sample size = 125



The modeled solution troubles me because I think it is dubious science. It seems to assume that the buses will leave the snow-free terminal garage at N minute intervals, and does not allow for a wait time greater than the departure frequency. The streets are covered with snow, the buses may be delayed or in fact, they may be stopped. As such, a longer wait time should be a possibility.

Algebraic solution

For this solution, wait times are estimated using the exponential distribution:

$f(t) = \lambda e^{-\lambda t,}$ where, in this instance, λ (which is unknown) has units of time^{-1}.

The Maximum Likelihood Estimator L is given by:

$$L(\lambda) = (\lambda e^{-\lambda t1})(\lambda e^{-\lambda t2})\ldots(\lambda e^{-\lambda tn})$$

$$L(\lambda) = \lambda^{n} e^{-\lambda(t1 + t2 + ..tn)}$$

This simplifies to $L(\lambda) = \lambda^{n} e^{-\lambda n(\text{mean}(t))}$

The value of λ is what must be determined based upon all we that we know about this situation, which is that some people have been waiting independently in the snow for a bus for different lengths of time.

Taking the natural log of both sides:

$$\ln(L(\lambda)) = n\ln(\lambda) - \lambda n t\bar{t}$$

Taking the first derivative with respect to λ and setting it equal to zero yields:

$$n/\lambda - n\bar{t} = 0$$

This yields $\hat{\lambda} = 1/\bar{t}$.

The second derivative is $-n/\lambda^2$, which is negative for all values of λ, so we know that this is a maximum.

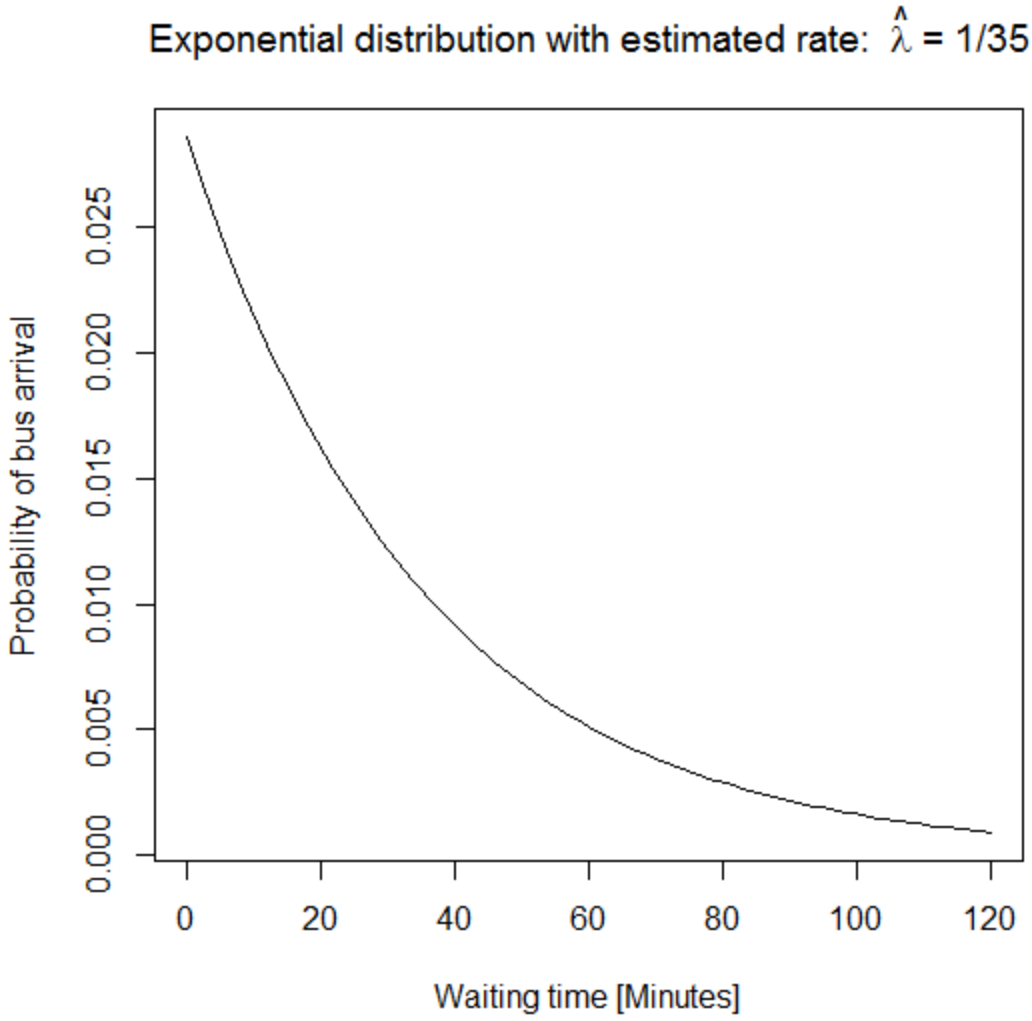
Personal note:

This was a fun exercise. My son is a high school senior, taking calculus, and I asked him to check the above calculations. I think he enjoyed the involvement, and was maybe surprised that the old man isn't quite as dumb as he looks.

To apply this, I ask all my fellow bus riders how long they have been waiting for the bus, take the mean of their responses, and take the reciprocal. For demonstration purposes, let us assume that the responses are (40, 30, 50, 20) minutes, so some quick math (140/4) reveals that we are stuck in an exponential distribution with λ = 1/35. I then pull out my laptop, fire up R, and enter the following code:

```
buswait = function(x) { dexp(x, rate=(1/35)) }
curve(buswait, 0, 120,
main=bquote(paste("Exponential distribution with estimated rate:  ",
hat(lambda), " = 1/35" ) ),
xlab="Waiting time [Minutes]",
ylab="Probability of bus arrival" )
```

Then I show the waiting passengers this chart:



One of them asks, “That’s just wonderful, but what does it mean? Tell me something useful, like when will the bus come?”

So, I do a couple of quick integations:

```
integrate(buswait, 0, 60)
integrate(buswait, 0, 120)
```

I tell them that based upon their reported wait times, there is an 82% probability that the bus will arrive in an hour. They groan, so I try to bolster their spirits by telling them that there is a 96% probability that the bus will come in two hours. That doesn’t work - they groan more. I remind them that I’m just the messenger, and while it may feel good to punish the messenger for bad news, that vengeance is misplaced. They groan even more, and start to wander off to Starbuck’s. I do another quick integration:

```
integrate(buswait, 0, 10)
```

I point out to them, “On the other hand, there is a 25% probability that the bus will come in the next 10 minutes, so you might not want to go too far…”

The big guy with the pierced head and the Sounders FC emblem tattoo across his face asks me if I would like to become a one-handed statistician. I reply, “Probably not.” I had no idea that statistics could be so fraught with danger.

The Likelihood Always Goes Up

A likelihood ratio test is used to compare the fit of two models. Reviewing Wikipedia content, the ratio can be calculated as:

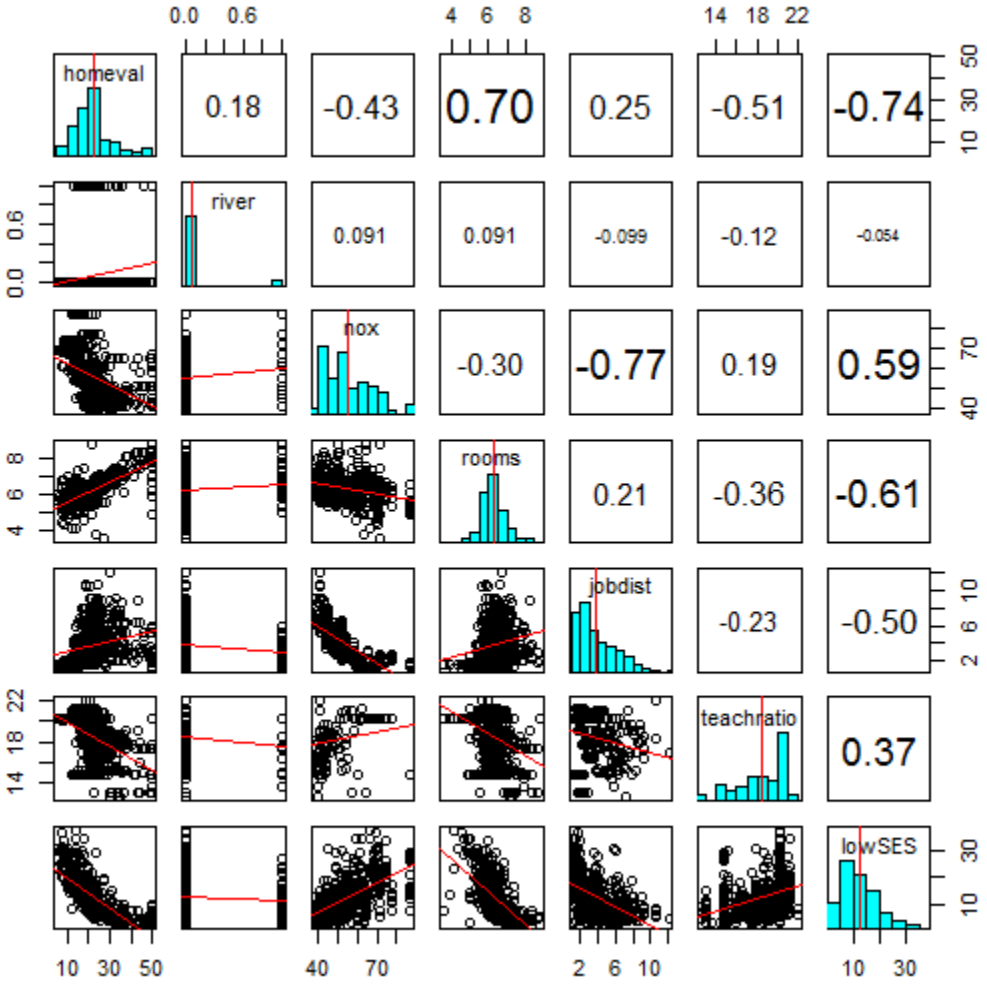
D = -2 ln(likelihood for null model) + 2 ln(likelihood for alternative model)

This compares the likelihood of a general model with that of a saturated model. Given N observations, the saturated model will have N parameters. The fit of the saturated model is perfect, but it has zero explanatory power. In contrast, the minimally adequate model could have N parameters, but ideally it would have less parameters than the N observations. It would have a fit that is perhaps less than the saturated model, but an explanatory power R^2 = SSR/SSY.

Back to Boston

```
# Back to Boston

setwd("C:/Users/Rod/SkyDrive/R/201/Week02")
boston = read.csv("Boston.csv")

full.model = lm(homeval ~ crime + biglots + indust + river + nox + rooms + old + jobdist + tax + teachratio
+ lowSES, data=boston)
summary(full.model)

step.model = step(full.model, direction="backward")
summary(step.model)

reduced.model = lm(homeval ~ river + nox + rooms + jobdist + teachratio + lowSES, data=boston)
summary(reduced.model)
```

```
boston.reduced = data.frame(cbind(boston$homeval, boston$river, boston$nox, boston$rooms,
boston$jobdist, boston$teachratio, boston$lowSES))

names(boston.reduced) = c('homeval', 'river', 'nox', 'rooms', 'jobdist', 'teachratio', 'lowSES')
pairsPlus(boston.reduced)
```



Build the Model

```
cor(boston.reduced)
            homeval      river        nox      rooms    jobdist teachratio    lowSES
homeval   1.0000000  0.17526018 -0.42732077  0.69535995  0.24992873 -0.5077867 -0.7376627
river     0.1752602  1.00000000  0.09120281  0.09125123 -0.09917578 -0.1215152 -0.0539293
nox      -0.4273208  0.09120281  1.00000000 -0.30218819 -0.76923011  0.1889327  0.5908789
rooms     0.6953599  0.09125123 -0.30218819  1.00000000  0.20524621 -0.3555015 -0.6138083
jobdist   0.2499287 -0.09917578 -0.76923011  0.20524621  1.00000000 -0.2324705 -0.4969958
teachratio -0.5077867 -0.12151517  0.18893268 -0.35550149 -0.23247054  1.0000000  0.3740443
```

lowSES -0.7376627 -0.05392930 0.59087892 -0.61380827 -0.49699583 0.3740443 1.0000000

By inspection, the covariate with the highest correlation to homeval is lowSES.

I created a series of models, adding a covariate to each successive model:

```
fit1 = lm(homeval ~ lowSES, data = boston.reduced)
fit2 = lm(homeval ~ lowSES + rooms, data = boston.reduced)
fit3 = lm(homeval ~ lowSES + rooms + teachratio, data = boston.reduced)
fit4 = lm(homeval ~ lowSES + rooms + teachratio + nox, data = boston.reduced)
fit5 = lm(homeval ~ lowSES + rooms + teachratio + nox + jobdist, data = boston.reduced)
fit6 = lm(homeval ~ lowSES + rooms + teachratio + nox + jobdist + river, data = boston.reduced)
```

I then ran waldtest on the series:

```
> waldtest(fit1, fit2, fit3, fit4, fit5, fit6)
Wald test

Model 1: homeval ~ lowSES
Model 2: homeval ~ lowSES + rooms
Model 3: homeval ~ lowSES + rooms + teachratio
Model 4: homeval ~ lowSES + rooms + teachratio + nox
Model 5: homeval ~ lowSES + rooms + teachratio + nox + jobdist
Model 6: homeval ~ lowSES + rooms + teachratio + nox + jobdist + river
  Res.Df Df       F    Pr(>F)
1    504
2    503  1 131.3942 < 2.2e-16 ***
3    502  1  62.5791 1.645e-14 ***
4    501  1   0.9072 0.3413103
5    500  1  49.4744 6.642e-12 ***
6    499  1  13.4920 0.0002655 ***
---
```

I noticed something odd about the p-value for fit4 as compared to fit3. I assume that means that the addition of nox to the model does not have a significant effect.

I got the same result from lrtest:

```
> lrtest(fit1, fit2, fit3, fit4, fit5, fit6)
Likelihood ratio test

Model 1: homeval ~ lowSES
Model 2: homeval ~ lowSES + rooms
```

Model 3: homeval ~ lowSES + rooms + teachratio
Model 4: homeval ~ lowSES + rooms + teachratio + nox
Model 5: homeval ~ lowSES + rooms + teachratio + nox + jobdist
Model 6: homeval ~ lowSES + rooms + teachratio + nox + jobdist + river

	#Df	LogLik	Df	Chisq	Pr(>Chisq)	
1	3	-1641.5				
2	4	-1582.8	1	117.4326	< 2.2e-16	***
3	5	-1553.0	1	59.4450	1.258e-14	***
4	6	-1552.6	1	0.9155	0.3386716	
5	7	-1528.7	1	47.7432	4.859e-12	***
6	8	-1522.0	1	13.4996	0.0002386	***

In summary, I am not certain how to interpret these results. Could we go over this in class?