# Problem_Set_4_Carlos_Sathler

*Carlos Sathler*

*3/2/2019*

## Contents

## Problem 1

```
# code adapted from class lecture by Dr. Luen
# tally observed counts
kids = 0:10
n = c(398, 455, 575, 227, 65, 28, 9, 3, 0, 1, 0)
kids.tally = data.frame(kids, n)
kids.tally
```

```
##    kids   n
## 1     0 398
## 2     1 455
## 3     2 575
## 4     3 227
## 5     4  65
## 6     5  28
## 7     6   9
## 8     7   3
## 9     8   0
## 10    9   1
## 11   10   0
```

```
# find total number of observations
N = sum(n)
N
```

```
## [1] 1761
```

```
# estimate lambda for Poisson distribution (H0: Poisson model is good fit for observed data)
lambda.obs = weighted.mean(kids.tally$kids, kids.tally$n)
lambda.obs
```

```
## [1] 1.57297
```

```
# find expected counts based on N and lambda
expected = N * dpois(kids.tally$kids, lambda.obs)
round(expected, 1)
```

```
##  [1] 365.3 574.6 451.9 236.9  93.2  29.3   7.7   1.7   0.3   0.1   0.0
```

```
# calculate chi-squared
observed = kids.tally$n
round(observed,1)
```

```
##  [1] 398 455 575 227  65  28   9   3   0   1   0
```

```
chi.stat = sum((observed - expected)^2 / expected) + (N - sum(expected))  # second term is total "extra
chi.stat
```

```
## [1] 86.77279
```

Need to use simulation because we have infinite number of categories (possible number of kids), therefore cannot know k to calculate degrees of freedom (df = k-d-1) to use in R function pschisq.

```
# function simulate sample drawn from poisson distribution with lambda calculated from original sample
# code adapted from class lecture by Dr. Luen
poisson.chisq.sim = function(N, lambda.obs){
  sim.sample = data.frame(x = rpois(N, lambda.obs))
  lambda.sim = mean(sim.sample$x)
  x.tally = tally(group_by(sim.sample, x))
  observed = x.tally$n
  expected = N * dpois(x.tally$x, lambda.sim)
  extra = N - sum(expected)
  chi.stat = sum((observed - expected)^2/ expected) + extra
  return(chi.stat)
}
```

Run it 10,000 times and find a P-value:

```
# code adapted from class lecture by Dr. Luen
# run function 10000 times, to get 10000 chi-squared stats to compare to the one from original sample
chi.list = replicate(10000, poisson.chisq.sim(N, lambda.obs))
# count percentage of time we found simulated stat was more extreme than stat from original sample
pvalue = mean(chi.list >= chi.stat)
pvalue
```

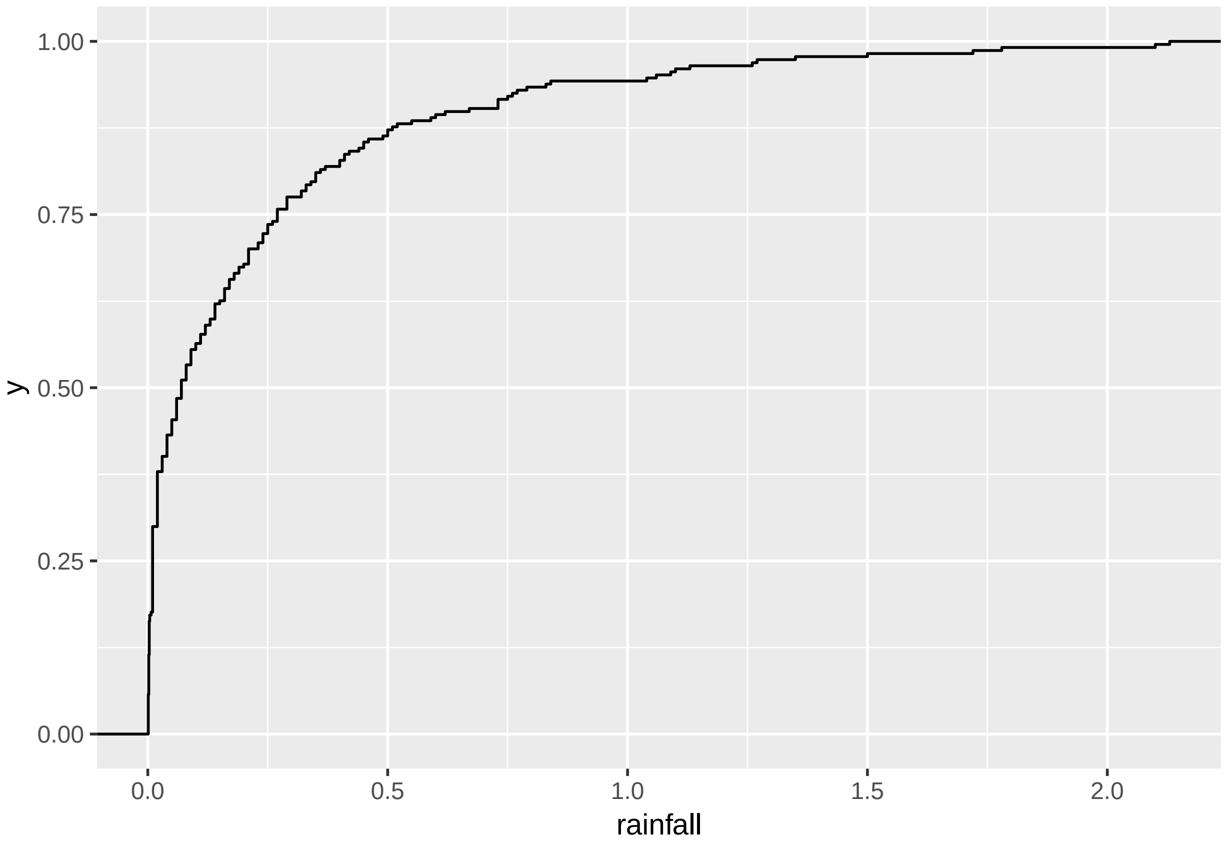
```
## [1] 0.0094
```

P-value = 0.0094

P-value is low (less than 5%), indicating that there is evidence to reject the null that the Poisson distribution is a good parametric distribution to model the observed data. In other words, the data cannot be well modeled by a Poisson distribution.

## Problem 2

```r
# code adapted from class lecture by Dr. Luen
rainfall = scan("illinois-rainstorms.txt")
F.hat = ecdf(rainfall)
ggplot(data.frame(rainfall), aes(x = rainfall)) + stat_ecdf()
```



Pointwise 95% band estimation accross range of rainstorm values.
Populate array with upper and lower point estimation values, for all observed points.

```r
# code adapted from class lecture by Dr. Luen
n = length(rainfall)
# create and initialize grid of x values to be ploted
x.grid = seq(0.01, max(rainfall), 0.01)
point.lower = rep(NA, length(x.grid))
point.upper = rep(NA, length(x.grid))
# for each x value in the grid
for(J in 1:length(x.grid)){
  # use the binomial to find probability of F(y) = P(X <= y) and it's confidence interval
  CI = binom.test(sum(rainfall <= x.grid[J]), n)$conf.int
  point.lower[J] = CI[1]
  point.upper[J] = CI[2]
}
```

Simultaneos 95% band estimation using DKW inequality, which calculates epsilon using alpha (alpha = 0.05 for a 95% band). Epsilon is used as the distance up and down the observed ECDF to draw the simultaneos band.

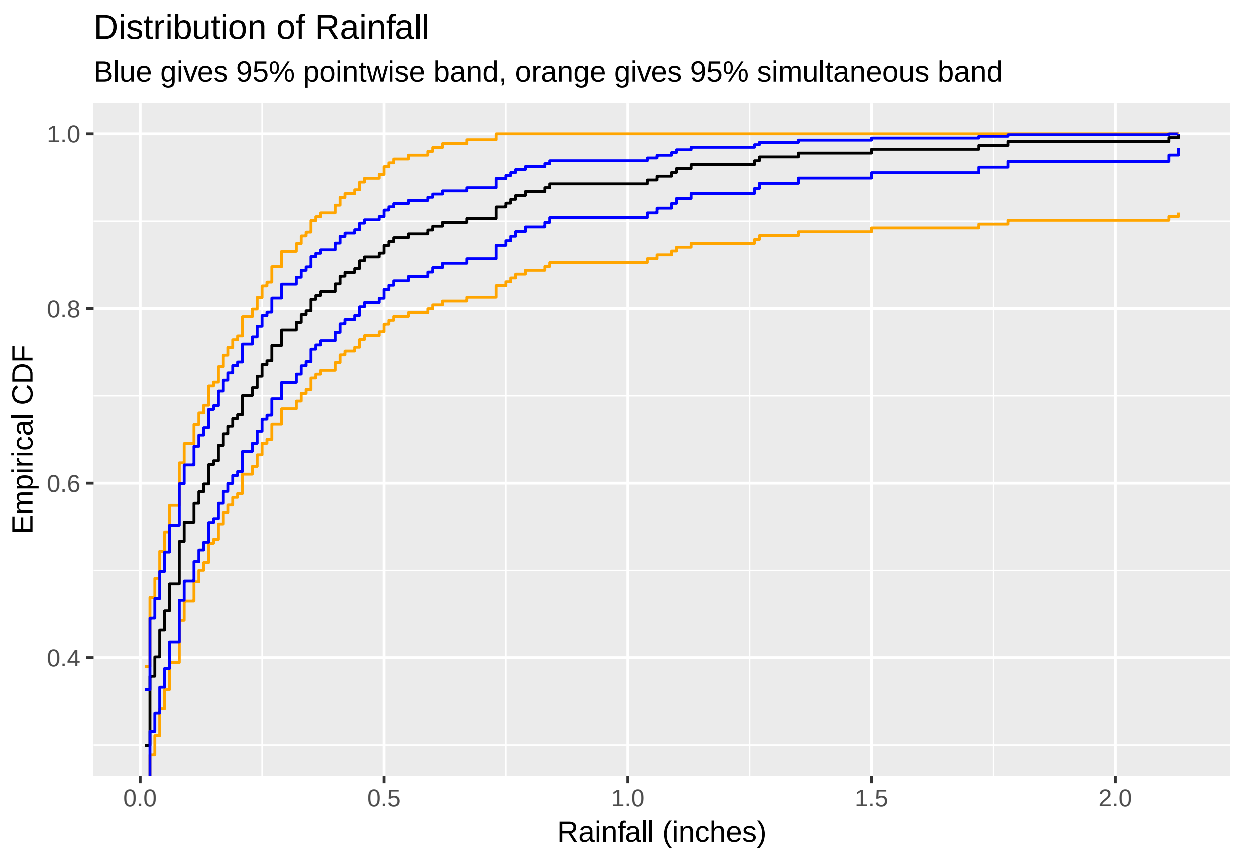
```r
# code adapted from class lecture by Dr. Luen

# calculate epsilon from alpha
alpha = 0.05
epsilon = sqrt(log(2 / alpha) / (2 * n))
epsilon
```

```
## [1] 0.09014036
```

```r
# Function to find lower bound
L = function(y) {
    raw = F.hat(y) - epsilon
    return(ifelse(raw < 0, 0, raw))
}
# Function to find upper bound
U = function(y) {
    raw = F.hat(y) + epsilon
    return(ifelse(raw > 1, 1, raw))
}

# get points for simultaneos 95% bands
# using the same grid created before
simul.lower = L(x.grid)
simul.upper = U(x.grid)
```

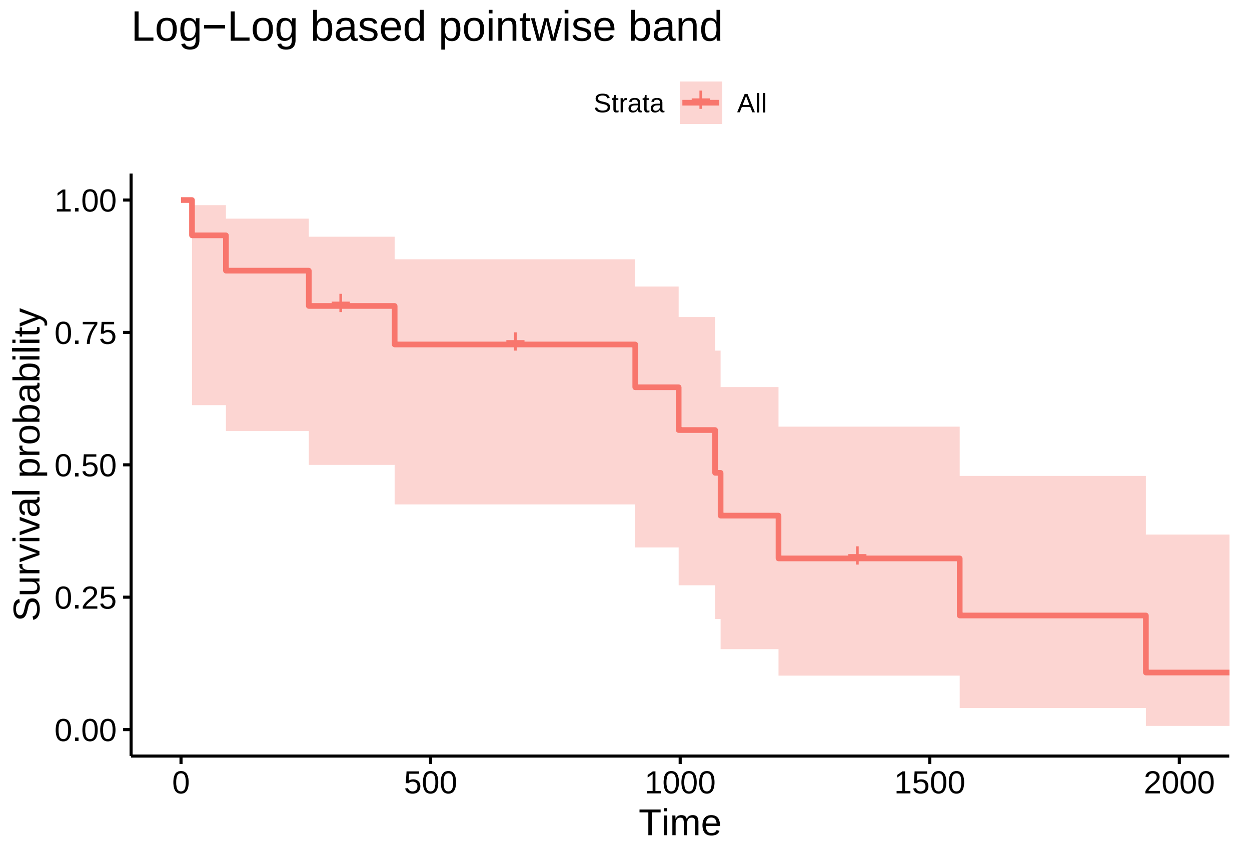
Plot 95% pointwise confidendence band for the CDF of rainfall in blue, and 95% simultaneous conndence band for the CDF of rainfall in orange. CDF for observed values appears in black.

```r
# code adapted from class lecture by Dr. Luen
y = F.hat(x.grid)
plot.df = data.frame(x = x.grid, y)
# plot observed ecdf
gg = ggplot(plot.df, aes(x, y)) + geom_step()
# plot simultaneous band in orange
gg = gg + geom_step(y = simul.lower, col = "orange") + geom_step(y = simul.upper, col = "orange")
# plot pointwise band in blue
gg = gg + geom_step(y = point.lower, col = "blue") + geom_step(y = point.upper, col = "blue")
gg = gg + ggtitle("Distribution of Rainfall")
gg = gg + labs(subtitle = "Blue gives 95% pointwise band, orange gives 95% simultaneous band")
gg + xlab("Rainfall (inches)") + ylab("Empirical CDF")
```


Distribution of Rainfall
Blue gives 95% pointwise band, orange gives 95% simultaneous band
Empirical CDF
1.0
0.8
0.6
0.4
0.0
0.5
1.0
1.5
2.0
Rainfall (inches)

**Problem 3**

```r
# code adapted from class lecture by Dr. Luen
library(survival)
library(survminer)
time  = c(22, 90, 256, 320, 428, 670, 910, 997, 1070, 1081, 1197, 1355, 1560, 1933, 2202)
event = c(1,   1,   1,   0,   1,   0,   1,   1,    1,    1,    1,    0,    1,    1,    1)
surv.data = Surv(time = time, event = event)
fit = survfit(surv.data ~ 1, conf.type = "log-log")
#data.frame(fit$time, fit$surv)
ggsurvplot(fit, data = surv.data) + ggtitle("Log-Log based pointwise band")
```



## Problem 4

Get sample estimates

```r
# get sample stats
rainfall = scan("illinois-rainstorms.txt")
n = length(rainfall)
x.bar = mean(rainfall)
x.sd = sd(rainfall)
x.var = var(rainfall)
x.cv = x.sd / x.bar
x.bar.mse = x.var / n
```

### (a)

```r
# sample mean
x.bar
```

```
## [1] 0.2243921
```

```r
set.seed(7)
# find bootstrap estimate of mean squared error for mean statistic
# steps 1, 2, 3 from class slides (slides-bootstrap-erros.pdf)
x.bar.boots = replicate(10000, mean(sample(rainfall, replace = TRUE)))
# step 4 from class slides
mse.x.bar = mean((x.bar.boots - x.bar)^2)
mse.x.bar
```

```
## [1] 0.000576657
```

Sample mean: 0.2244 MSE of sample mean, estimated from bootstrap: 0.0006

### (b)

```r
# sample sd
x.sd
```

```
## [1] 0.3658212
```

```r
set.seed(7)
# find bootstrap estimate of mean squared error for sd statistic
# steps 1, 2, 3 from class slides (slides-bootstrap-erros.pdf)
x.sd.boots = replicate(10000, sd(sample(rainfall, replace = TRUE)))
# step 4 from class slides
mse.x.sd = mean((x.sd.boots - x.sd)^2)
mse.x.sd
```

```
## [1] 0.001578371
```

Sample sd: 0.3658 MSE of sample sd, estimated from bootstrap: 0.0016

### (c)

```r
library(raster)
```

```
## Loading required package: sp

##
## Attaching package: 'raster'

## The following object is masked from 'package:ggpubr':
##
##     rotate

## The following object is masked from 'package:magrittr':
##
##     extract

## The following object is masked from 'package:dplyr':
##
##     select
```

```r
# sample cv
x.cv
```

```
## [1] 1.630277
```

```r
set.seed(7)
# find bootstrap estimate of mean squared error for cv statistic
# steps 1, 2, 3 from class slides (slides-bootstrap-erros.pdf)
x.cv.boots = replicate(10000, cv(sample(rainfall, replace = TRUE)))
# step 4 from class slides
mse.x.cv = mean((x.cv.boots/100 - x.cv)^2)
mse.x.cv
```

```
## [1] 0.009870811
```

Coefficient of variation: 1.6303 MSE of coefficient of variation, estimated from bootstrap: 0.0099

## Problem 5

Read data and initialize sample stats

```
easinophil = scan("rabbits2.txt")
summary(easinophil)
```

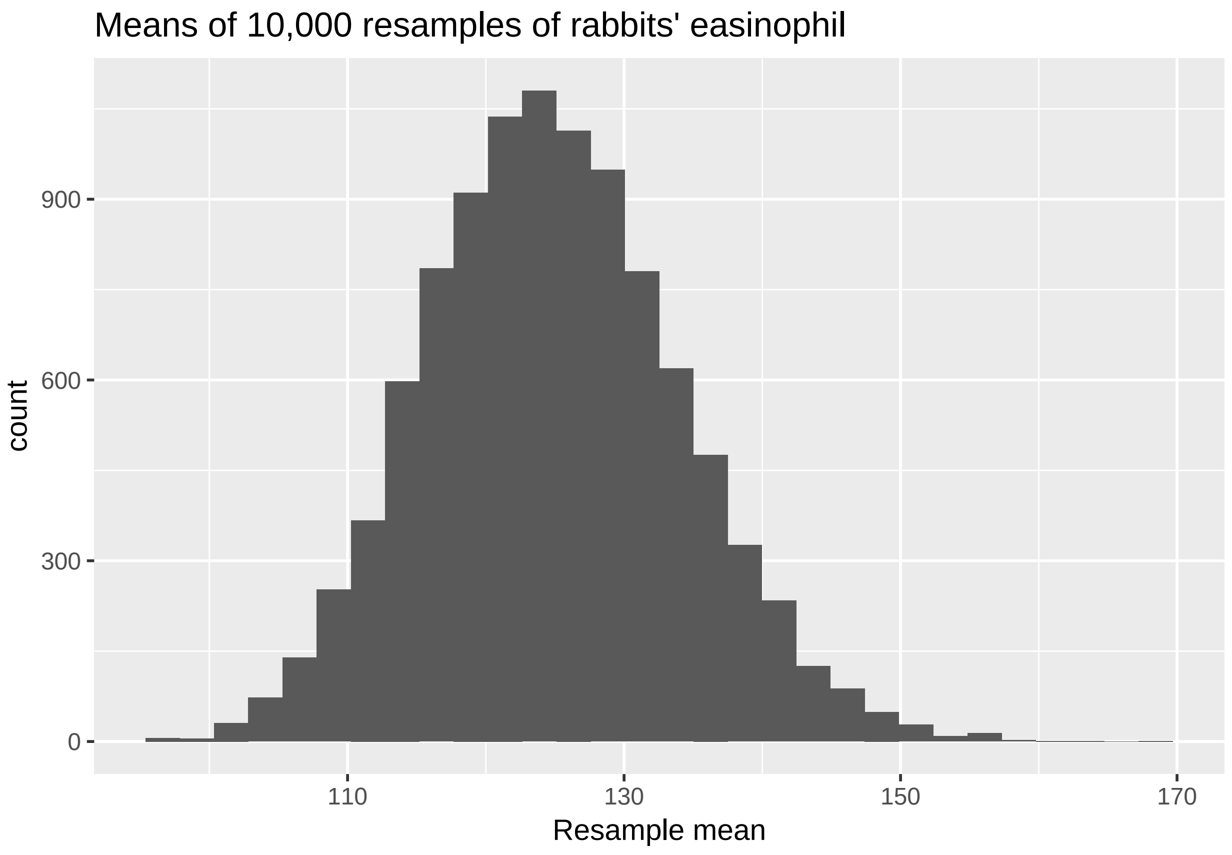
```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   45.00   78.75  112.50  124.78  152.75  299.00
```

```
n = length(easinophil)
x.bar = mean(easinophil)
x.sd = sd(easinophil)
```

Plot histogram of 1,000 bootstrap means

```
# code from class lecture
library(ggplot2)
set.seed(7)
mean.boot = replicate(10000, mean(sample(easinophil, replace = TRUE)))
ggplot(data.frame(mean.boot), aes(x = mean.boot)) +
  geom_histogram() +
  xlab("Resample mean") +
  ggtitle("Means of 10,000 resamples of rabbits' easinophil")
```

```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```



**(a)**

```r
set.seed(7)

# code adapted from class lecture by Dr. Luen

### Studentized bootstrap for the mean
t.pivot = function(x) {
  x.boot = sample(x, replace = TRUE)
  t.boot = (mean(x.boot) - mean(x)) / (sd(x.boot) / sqrt(length(x)))
  return(t.boot)
}

t.boot = replicate(10000, t.pivot(easinophil))

x.bar - quantile(t.boot, c(0.975, 0.025)) * x.sd / sqrt(n)
```

```
##    97.5%     2.5%
## 108.0661 146.8284
```

The 95% Studentized t-pivot confidence interval for the mean eosinophil count of rabbits is [108.0661, 146.8284].

**(b)**

```r
# code adapted from class lecture by Dr. Luen

set.seed(7)
x = easinophil
p0 = mean(mean.boot <= mean(x))
z0 = qnorm(p0)
mean.jack = rep(NA, length(x))
for(J in 1:length(x)){
  mean.jack[J] = mean(x[-J])
}
a = sum((mean(mean.jack) - mean.jack)^3) / 6 /
  sum((mean(mean.jack) - mean.jack)^2) ^ 1.5
zp = qnorm(.975)
z.lower = z0 + (z0 - zp) / (1 - a * (z0 - zp))
z.upper = z0 + (z0 + zp) / (1 - a * (z0 + zp))
q.lower = pnorm(z.lower)
q.upper = pnorm(z.upper)
#c(q.lower, q.upper)

quantile(mean.boot, c(q.lower, q.upper))
```

```
## 3.802101%  98.4984%
##  109.1293  146.1428
```

The 95% BCa confidence interval for the mean eosinophil count of rabbits is [109.1293, 146.1428].

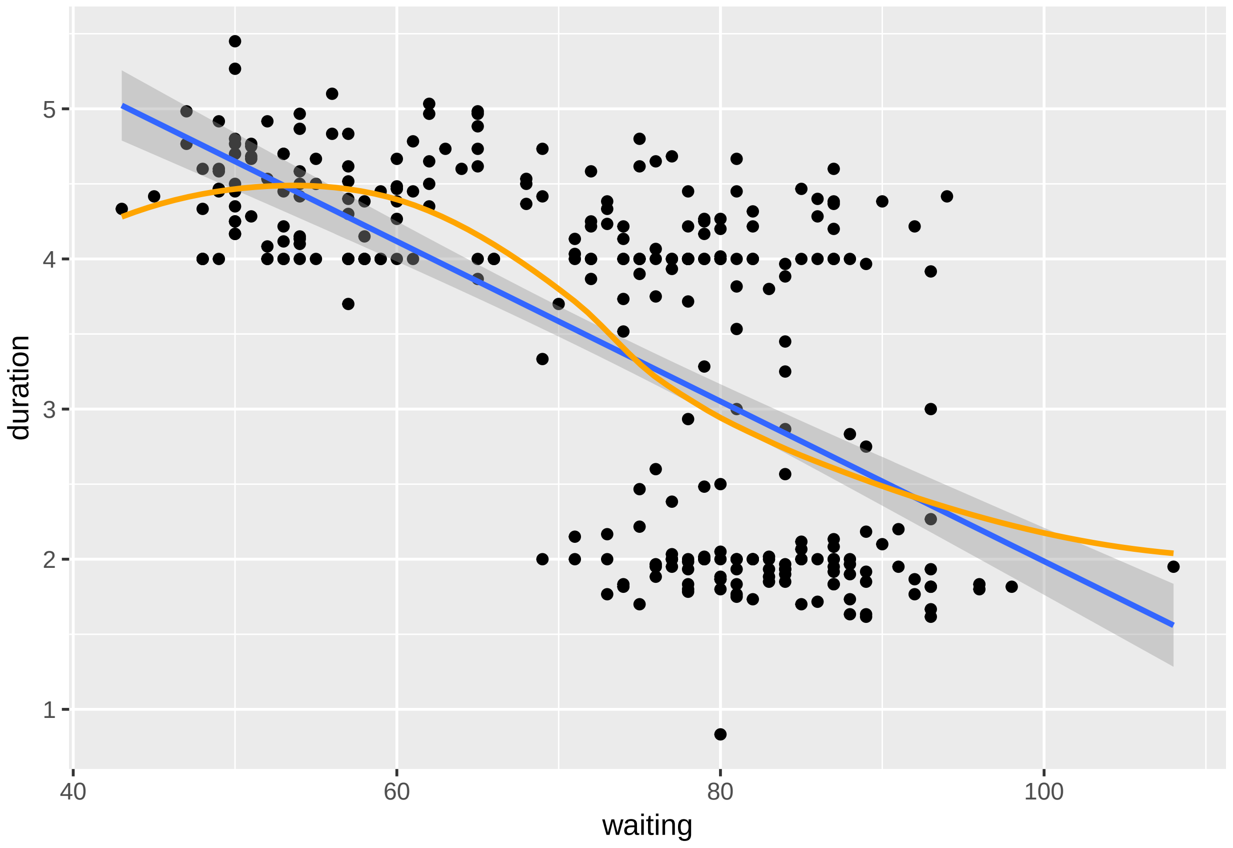
### Problem 6

#### (a)

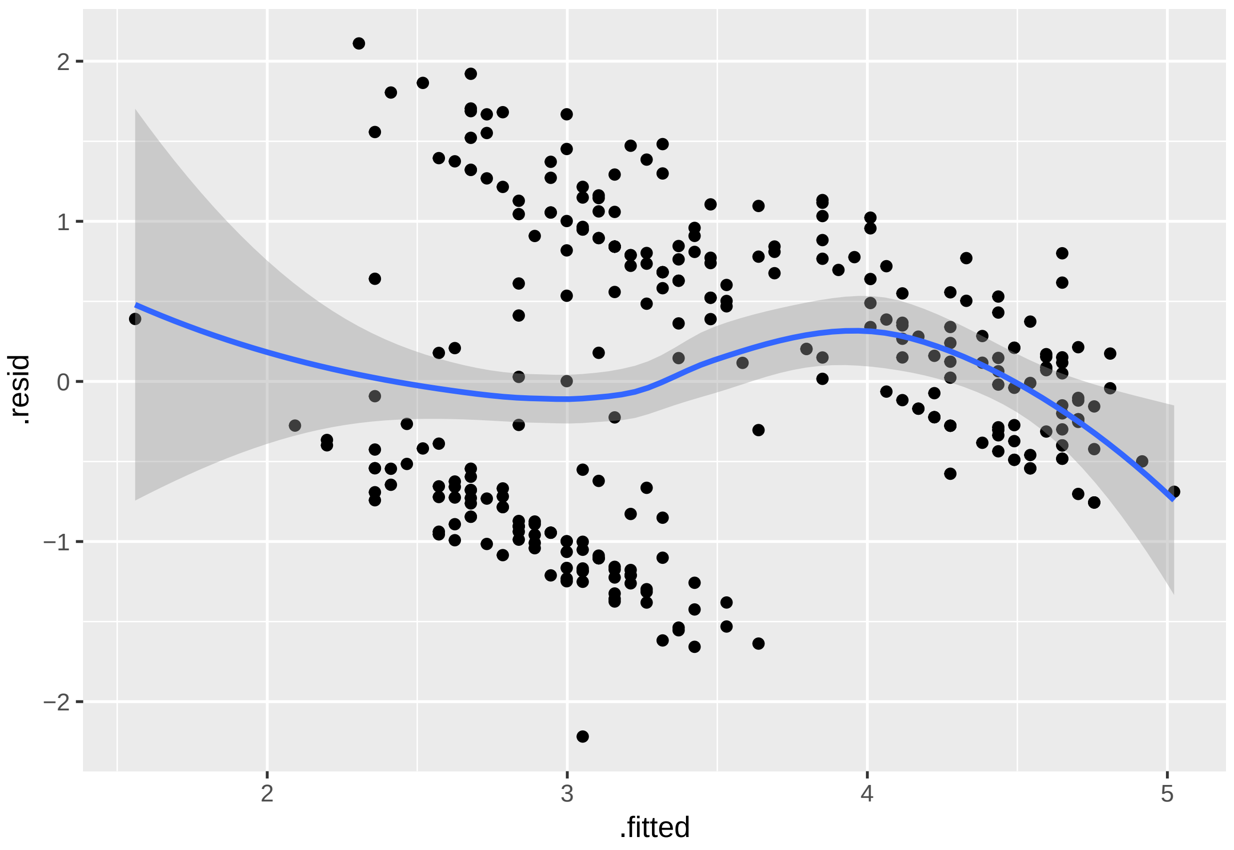
```
geyser = read.table('geyser.txt', header=T)
summary(geyser)
```

```
##     waiting          duration
##  Min.   : 43.00   Min.   :0.8333
##  1st Qu.: 59.00   1st Qu.:2.0000
##  Median : 76.00   Median :4.0000
##  Mean   : 72.31   Mean   :3.4608
##  3rd Qu.: 83.00   3rd Qu.:4.3833
##  Max.   :108.00   Max.   :5.4500
```

```
# plot duration as function of waiting
gg = ggplot(geyser, aes(y=duration, x=waiting)) + geom_point()
gg = gg + geom_smooth(method='lm') + geom_smooth(method='loess', se=F, color='orange')
gg
```



```
# plot residuals as function of fitted values
model = lm(duration ~ waiting, data=geyser)
model.df = augment(model)
gg1 = ggplot(model.df, aes(y=.resid, x=.fitted)) + geom_point() + geom_smooth(method='loess')
gg1
```



I the first plot, the linar smoother (blue) and the loess smoother (orange) are clearly NOT closely colocated. The relationship between the variables is not linear.

In the second plot, residuals don't appear randomly distributed. Additionally, the curvy loess line show evidence of heteroskedasticity.

In conclusion, both key assumptions of classical liner regression inference are violated: the relationship between variables doesn't appear to be linear and residuals don't show homoskedasticity.

### (b)

I will use the pairs bootstrap since the relationship between the variables is only somewhat linear.

```r
# code adapted from examples from class lectures

# function returns slope coefficient of lm model using resample with replacement (indices par)
lm.pairs.boot = function(data, indices){
  newdata = data[indices,]
  return(lm(newdata[,2] ~ newdata[,1])$coef[2])
}

# calls slope function 1000 time; boot will generate indices for the function
boot.lm.dist = boot(cbind(geyser$waiting, geyser$duration), lm.pairs.boot, R = 10000)
boot.lm.dist
```

```
##
## ORDINARY NONPARAMETRIC BOOTSTRAP
##
```

```
##
## Call:
## boot(data = cbind(geyser$waiting, geyser$duration), statistic = lm.pairs.boot,
##     R = 10000)
##
##
## Bootstrap Statistics :
##        original        bias    std. error
## t1* -0.05327198 -1.050758e-05 0.002980386
```

```r
boot.ci(boot.lm.dist)
```

```
## Warning in boot.ci(boot.lm.dist): bootstrap variances needed for
## studentized intervals
```

```
## BOOTSTRAP CONFIDENCE INTERVAL CALCULATIONS
## Based on 10000 bootstrap replicates
##
## CALL :
## boot.ci(boot.out = boot.lm.dist)
##
## Intervals :
## Level      Normal              Basic
## 95%   (-0.0591, -0.0474 )   (-0.0591, -0.0473 )
##
## Level     Percentile            BCa
## 95%   (-0.0592, -0.0474 )   (-0.0589, -0.0472 )
## Calculations and Intervals on Original Scale
```

Either interval can be used. Each has advantages and disadvantages. I will use the Basic 95% interval: (-0.0591, -0.0475). The interval means that 95% of the time we produce an estimate for the slope coeficient for this linear regression we expect it will fall within this interval.

## Problem 7

```r
set.seed(100)

# code adapted from example from class lectures

bootmv = function(x, indices){
  m = mean(x[indices])
  n = length(indices)
  v = var(x[indices]) / n # or plug-in
  return(c(m, v))
}

exp.competition = function(n){
  x = rchisq(n, 1)
  bootmv.dist = boot(x, bootmv, 1000)
  cis = boot.ci(bootmv.dist, type = "all")
  # expected value of chi-square disty with k=1 is 1
  x.bar = 1
  perc = (cis$perc[4]<=x.bar) & (cis$perc[5]>=x.bar)
  norm = (cis$norm[2]<=x.bar) & (cis$norm[3]>=x.bar)
  basic = (cis$basic[4]<=x.bar) & (cis$basic[5]>=x.bar)
  bca = (cis$bca[4]<=x.bar) & (cis$bca[5]>=x.bar)
  stud = (cis$stud[4]<=x.bar) & (cis$stud[5]>=x.bar)
  return(c(norm, perc, basic, bca, stud))
}
results = replicate(1000, exp.competition(20))
scores = apply(results, 1, sum)
names(scores) = c("norm", "perc", "basic", "bca", "stud")
scores
```

```
##  norm  perc basic   bca  stud
##   869   880   854   899   937
```

### (a) The percentile bootstrap

88%

### (b) The residual (basic) bootstrap

85.4%

### (c) The BCa bootstrap

89.9%

### (d) The Studentized (t-pivot) bootstrap

93.7%

## Problem 8

### (a)

```
library(MASS)
```

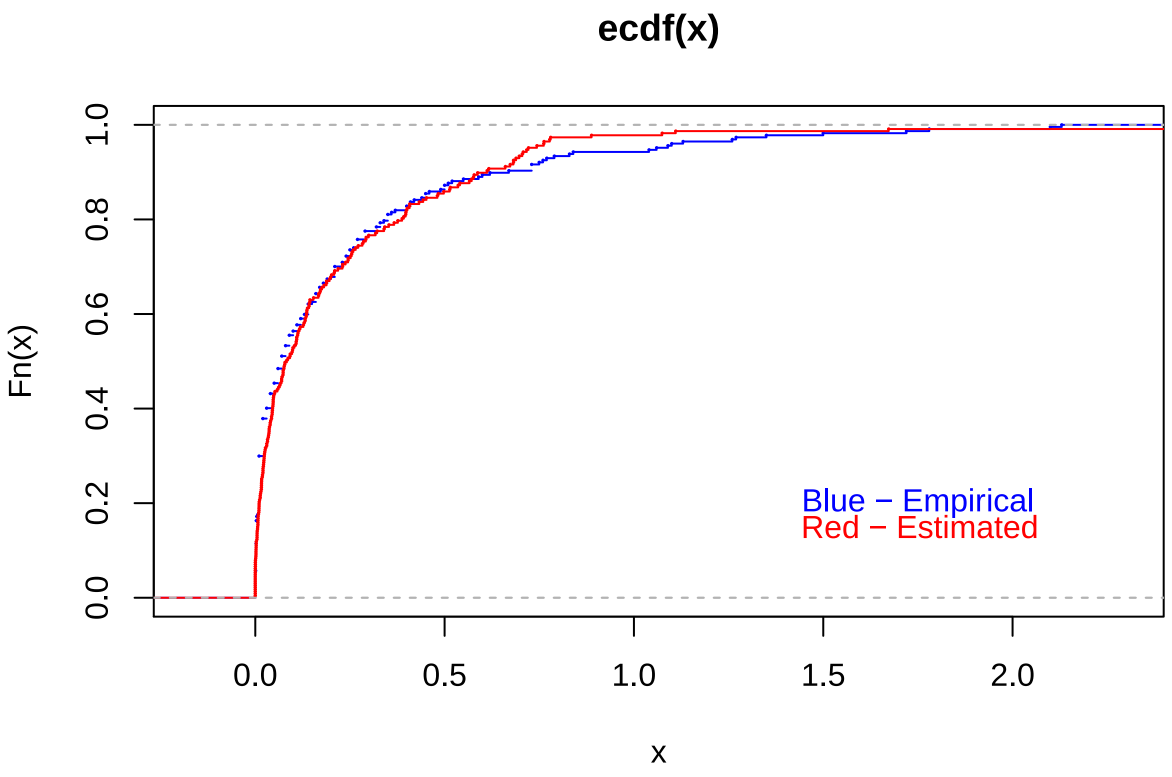
```
##
## Attaching package: 'MASS'

## The following object is masked _by_ '.GlobalEnv':
##
##     geyser

## The following objects are masked from 'package:raster':
##
##     area, select

## The following object is masked from 'package:dplyr':
##
##     select
```

```
fit = fitdistr(rainfall, "gamma")
fit$estimate
```

```
##     shape      rate
## 0.4408017 1.9647629
```

### (b)

```
# plot cdf of empirical data
plot.ecdf(rainfall, col='blue', cex=0.1)
# get the estimated distribution using mle parameters from part (a) above
estimated = rgamma(length(rainfall), shape = fit$estimate[1], rate = fit$estimate[2])
# plot cdf of estimated gamma
plot.ecdf(estimated, col='red', cex=0.1, add=T)
text(1.75, 0.2, 'Blue - Empirical', col='blue')
text(1.755, 0.15, 'Red - Estimated', col='red')
```


ecdf(x)
Fn(x)
1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.5
1.0
1.5
2.0
x
Blue − Empirical
Red − Estimated

## Problem 9

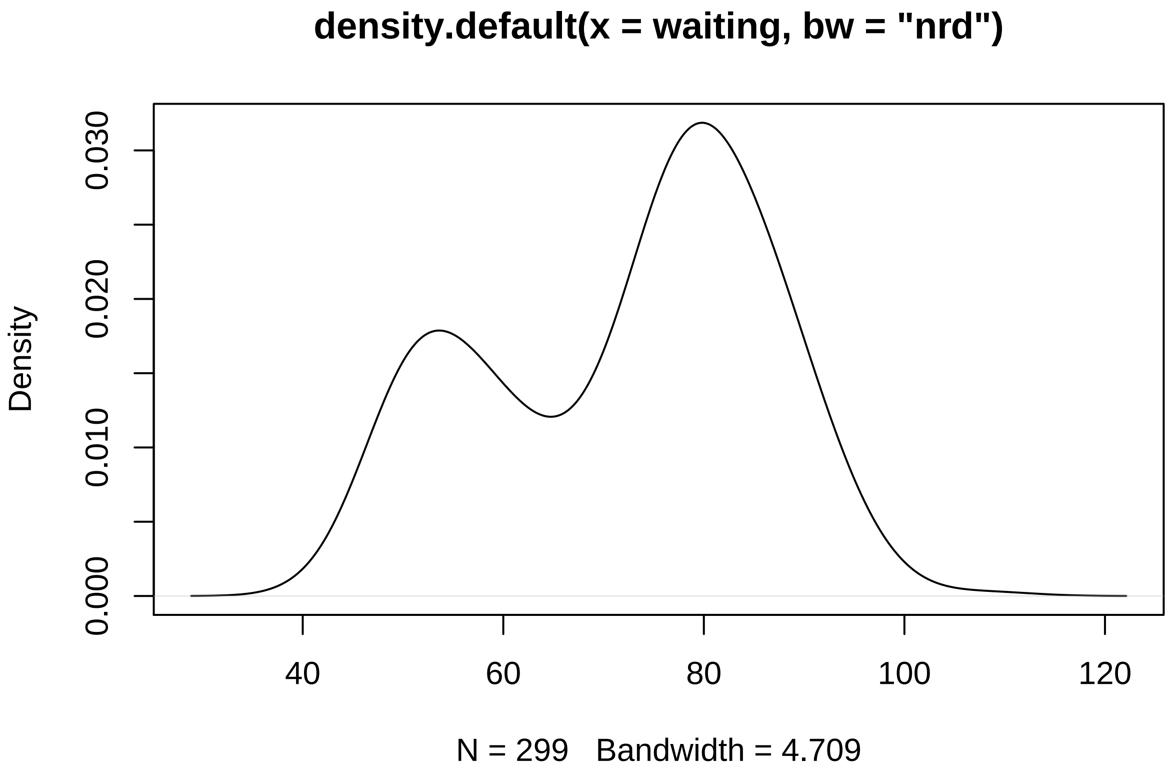
### (a)

I will use the R density function with bw parameter set by the Gaussian reference rule. The optimal bandwidth for this kernel according to class notes is:

$$1.06 * \sigma * n^{-1/5}$$

. According to R documentation (copied below), "ndr" is similar to R default "ndr0": * bw.nrd0 implements a rule-of-thumb for choosing the bandwidth of a Gaussian kernel density estimator. It defaults to 0.9 times the minimum of the standard deviation and the interquartile range divided by 1.34 times the sample size to the negative one-fifth power (= Silverman's 'rule of thumb', Silverman (1986, page 48, eqn (3.31))) unless the quartiles coincide when a positive result will be guaranteed. *bw.nrd is the more common variation given by Scott (1992), using factor 1.06.

```r
geyser = read.table("geyser.txt", header = T)
waiting = geyser$waiting
fit.bw = density(waiting, bw = 'nrd')
plot(fit.bw)
```

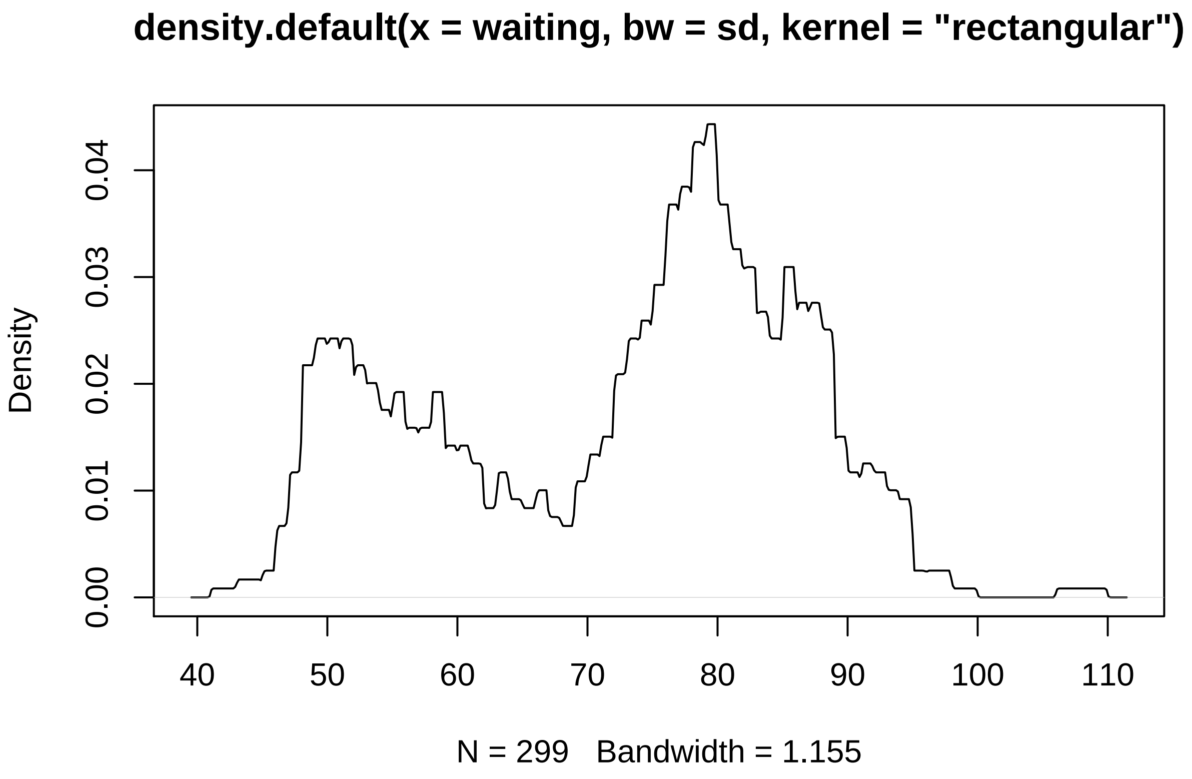


### (b)

I will use density function with kernel set to "regular" and bw set to sd of uniform distribution with "(b-a)=4". Per R documentation as well as class notes: "In density() the bandwidth is the standard deviation of the kernel."

```r
# I will use density function with kernel set to "regular"
# and bw set to sd of uniform distribution with "(b-a)=4"
sd = 4 * 12^(-0.5)
```

```
fit.bw = density(waiting, bw = sd, kernel="rectangular")
plot(fit.bw)
```



## Problem 10

### (a)

```
# code adapted from lecture code from Dr. Luen

# craete conditional distribution using package np
library(np)
```

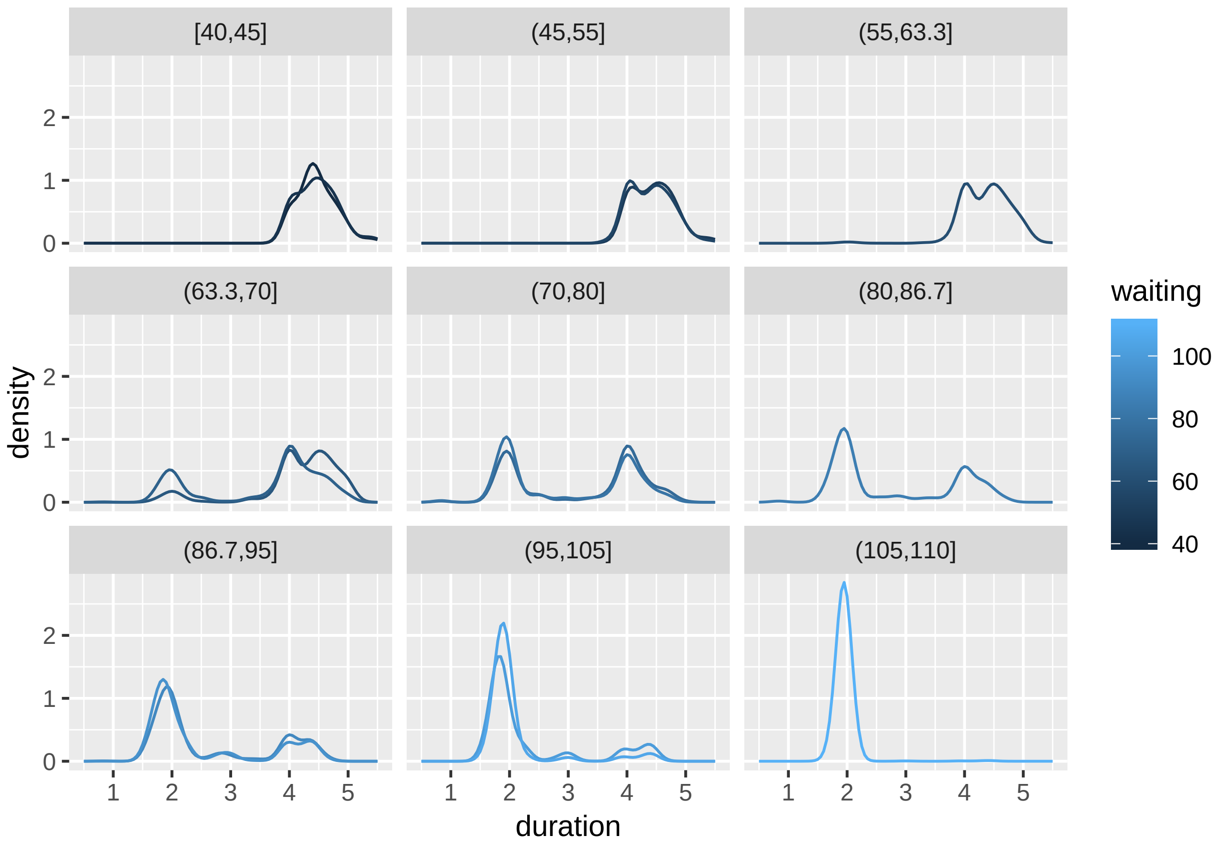
```
## Nonparametric Kernel Methods for Mixed Datatypes (version 0.60-9)
## [vignette("np_faq",package="np") provides answers to frequently asked questions]
## [vignette("np",package="np") an overview]
## [vignette("entropy_np",package="np") an overview of entropy-based methods]
```

```
geyser.cdens = npcdens(duration ~ waiting, data = geyser)
```

```
##
Multistart 1 of 2 |
Multistart 1 of 2 |
Multistart 1 of 2 |
Multistart 1 of 2 /
Multistart 1 of 2 -
Multistart 1 of 2 |
Multistart 1 of 2 |
Multistart 2 of 2 |
Multistart 2 of 2 |
Multistart 2 of 2 /
Multistart 2 of 2 -
Multistart 2 of 2 |
Multistart 2 of 2 |
```

```
grid = expand.grid(waiting = seq(40, 110, 5), duration = seq(0.5, 5.5, 0.05))
f.hat = predict(geyser.cdens, newdata = grid)
```

```
# plot density lines for duration conditioned on waiting
predict.df = data.frame(grid, density = f.hat)
gg = ggplot(predict.df, aes(x = duration, y = density, group = waiting, color = waiting)) + geom_line()
gg1 = gg + facet_wrap(~ cut_number(waiting, n=9))
gg1
```



The plot shows that for shorter waiting times, eruption duration is longer (around 5 minutes on avarage). As waiting time starts getting higher than 63.3 minutes, some eruptions have shorter duration (around 2 minutes). The number of shorter duration eruptions increases consistently thereafter as waiting time increases. Between 70 and 80 minutes, shorter duration (average ~2 min) and longer duration (average ~4 min) eruptions happen in about the same frequence. After that, little by little, as waiting time increases, short duration eruptions become progressively more frequent, while long duration ones become progressively less frequent. The patern continues until waiting time reaches 105 minutes, after which all eruptions have short duration, around 2 minutes.

**(b)**

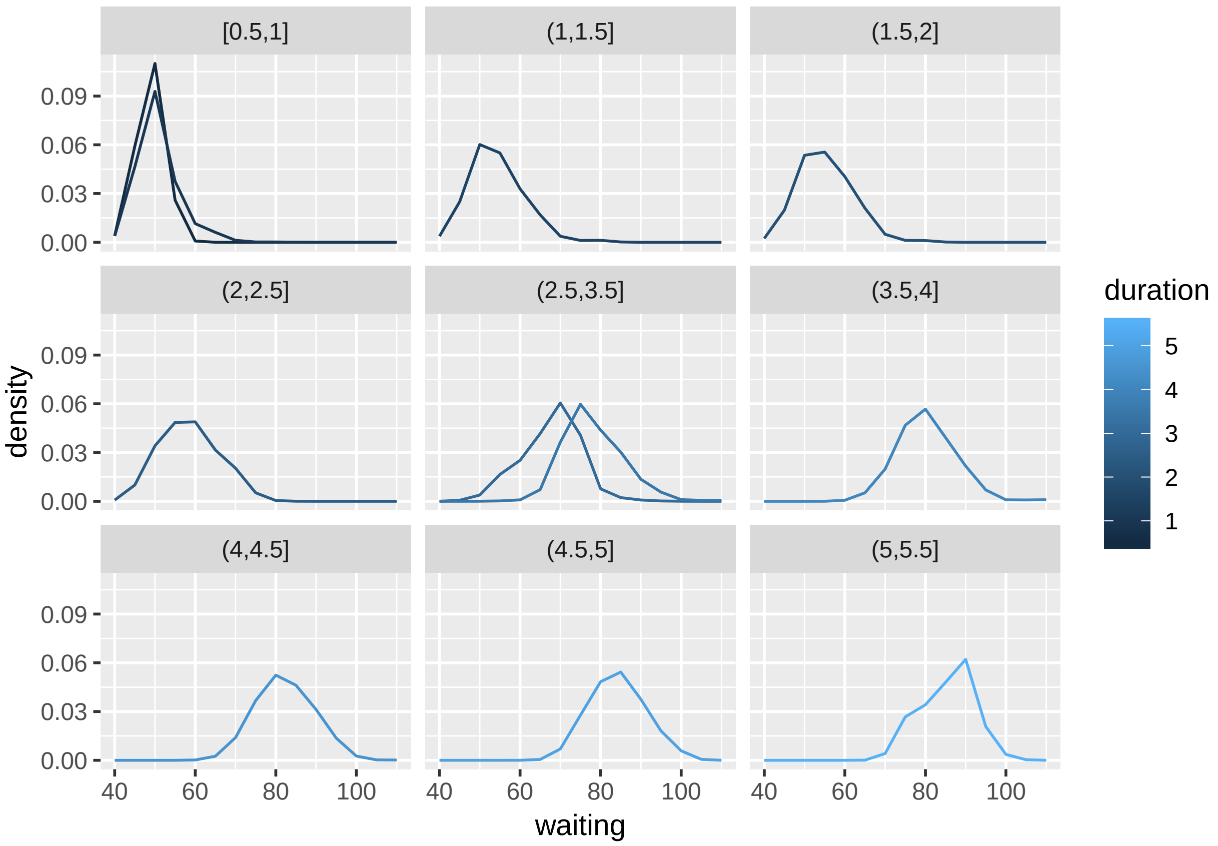
```r
# create copy of the database, moving columns, and delete elements
# to align previus duration with waiting time
geyser2 = data.frame(geyser$duration[-299], geyser$waiting[-1])
names(geyser2) = c('duration', 'waiting')
```

```r
# code adapted from lecture code from Dr. Luen

# craete condition distribution using package np
library(np)
geyser2.cdens = npcdens(waiting ~ duration, data = geyser2)
```

```
##
Multistart 1 of 2 |
Multistart 1 of 2 |
Multistart 1 of 2 |
Multistart 1 of 2 /
Multistart 1 of 2 -
Multistart 1 of 2 \
Multistart 1 of 2 |
Multistart 1 of 2 |
Multistart 2 of 2 |
Multistart 2 of 2 |
Multistart 2 of 2 /
Multistart 2 of 2 -
Multistart 2 of 2 |
Multistart 2 of 2 |
Multistart 2 of 2 /
Multistart 2 of 2 -
```

```
grid2 = expand.grid(waiting = seq(40, 110, 5), duration = seq(0.5, 5.5, 0.5))
f2.hat = predict(geyser2.cdens, newdata = grid2)
```

```
# plot density lines for duration conditioned on waiting
predict2.df = data.frame(grid2, density = f2.hat)
gg = ggplot(predict2.df, aes(x = waiting, y = density, group = duration, color = duration)) + geom_line
gg1 = gg + facet_wrap(~ cut_number(duration, n=9))
gg1
```



As eruption duration increases, waiting time following the eruption also increases. Shortest duration explosions (less than 1 min) are followed by short waiting periods averaging 50 minutes. Eruptions lasting between 1 and 2.5 minutes are followed by waiting periods averaging between 50 and 60 minutes. After 2.5 min long explosions, waiting times increase progressively, avareging around 80 minutes after 3.5 minutes long explosions. Waiting time reaches 85 minutes on average following explosions lasting longer than 4.5 minutes, and even a bit higher after explosions lasting longer than 5.5 minutes in duration.