Day 4 - Exponential Family

David Carlson

October 7, 2020

More advanced plotting, introductory simulations

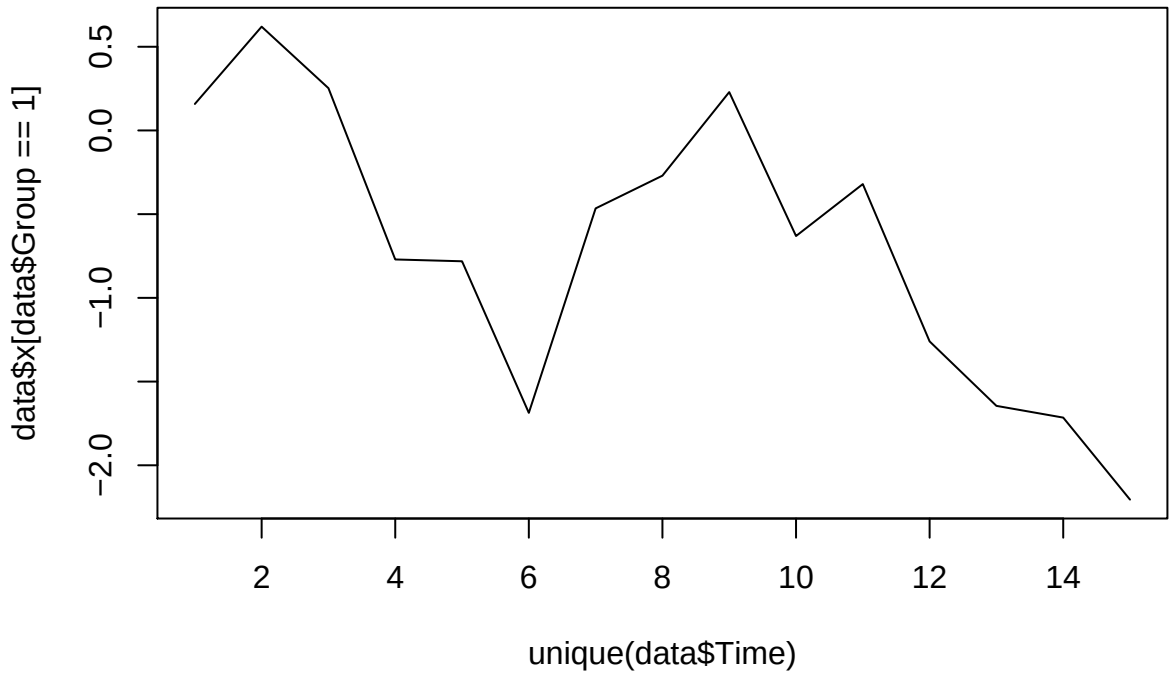
```
setwd('~/QPMRFall2021/inclass/day4Exp')
```

Simulate TSCS data

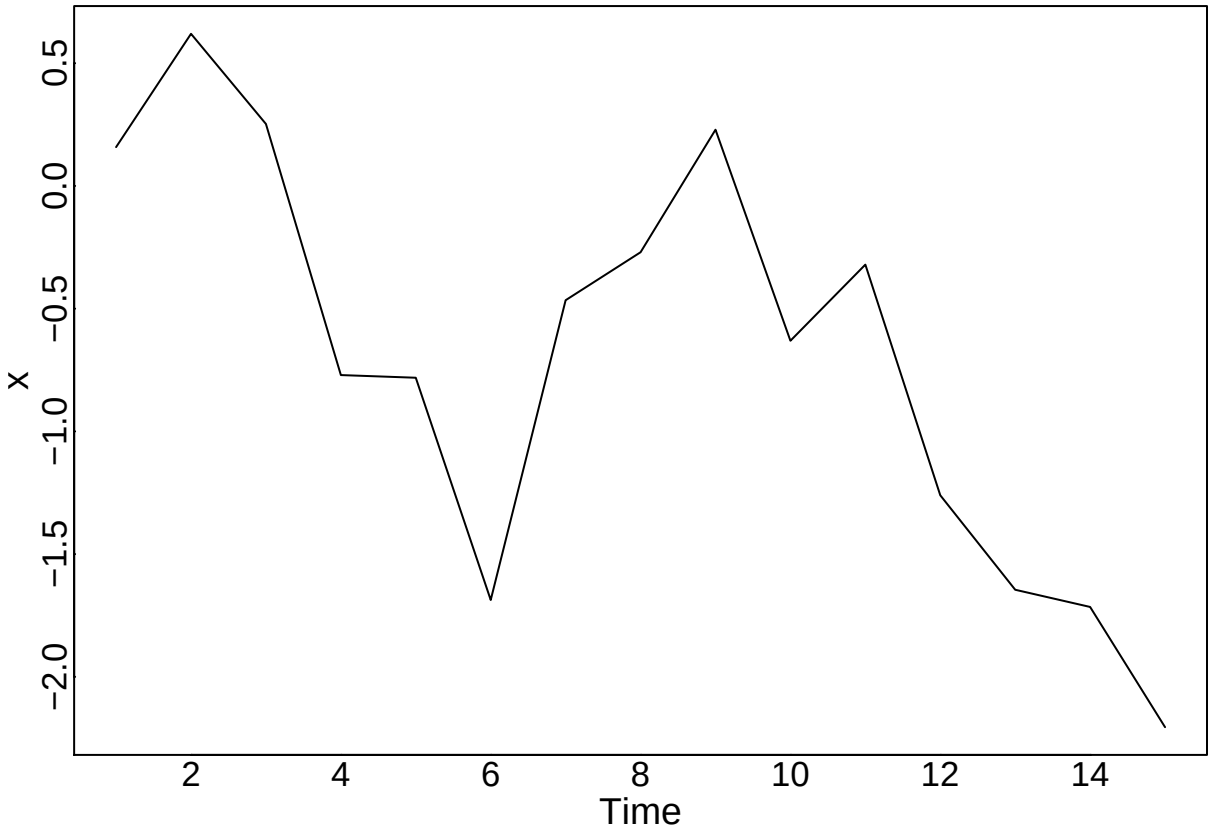
```
# lets create a function to generate one tscs sample with auto-correlation
tscsGen = function(rho, n.groups = 15, n.obs = 15, seed = 1523){
  set.seed(seed)
  returnMat = expand.grid(1:n.groups, 1:n.obs)
  colnames(returnMat) = c('Group', 'Time')
  returnMat$x = NA
  returnMat$x[returnMat$Time == 1] = rnorm(n.groups)
  err = numeric(n.obs*n.groups)
  err[returnMat$Time == 1] = rnorm(n.groups)

  for(g in 1:n.groups){
    for(t in 2:n.obs){
      returnMat$x[returnMat$Group == g & returnMat$Time == t] = rho * returnMat$x[returnMat$Group == g &
      err[returnMat$Group == g & returnMat$Time == t] = rho * err[returnMat$Group == g & returnMat$Time
    }
  }
  err = scale(err)
  g.ef = rnorm(n.groups)
  returnMat$y = -1 + rep(g.ef, times = n.groups) + .4*returnMat$x + err
  return(returnMat)
}
```

Generate one TSCS dataset at $\rho = 0.8$ and plot the variable relationships

```
data = tscsGen(rho = .8)
#the goal is to plot every trend in x on one window
plot(data$x[data$Group == 1] ~ unique(data$Time), type = 'l')
```

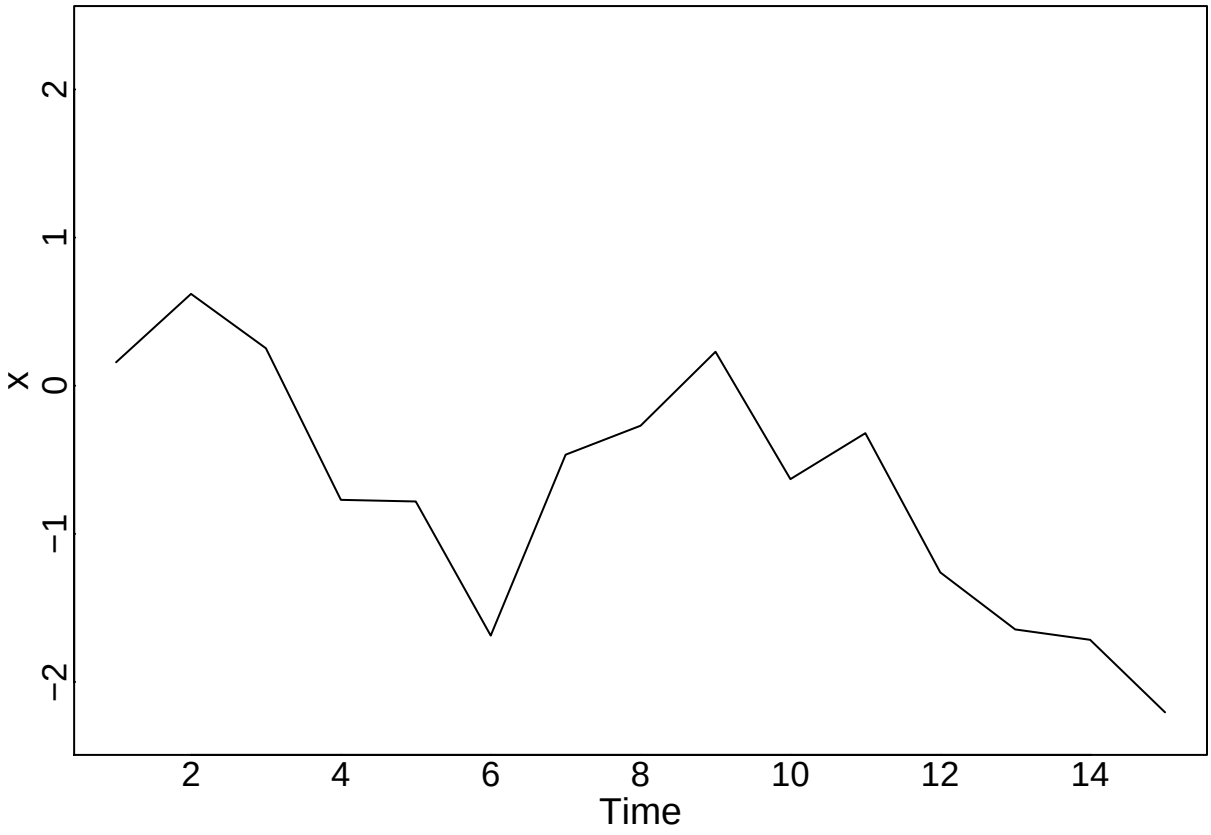



```
#add axis titles
plot(data$x[data$Group == 1] ~ unique(data$Time), type = 'l',
     xlab = 'Time', ylab = 'x')
#lets pretty it up
par(mfrow=c(1,1), mgp=c(1,0,0), tcl=0, mar=c(2,2,1,1), cex.lab=1.2, cex.axis=1.1)
plot(data$x[data$Group == 1] ~ unique(data$Time), type = 'l',
     xlab = 'Time', ylab = 'x')
```

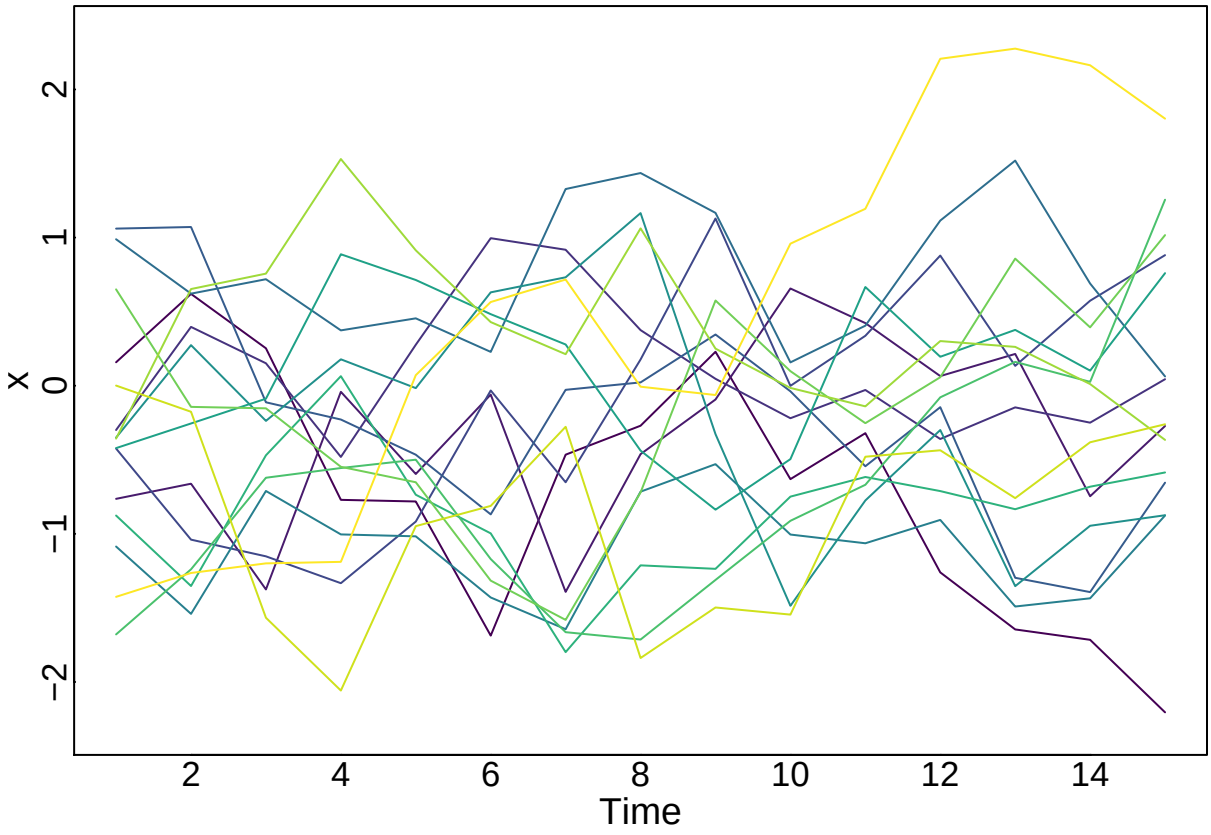



```
#well need to add other lines, and have to change the dim of the plot window accordingly
plot(data$x[data$Group == 1] ~ unique(data$Time), type = 'l',
     xlab = 'Time', ylab = 'x', ylim = c(min(data$x)-.1, max(data$x) +.1))
#lets add color to separate the lines, and add lines
library(viridis)
```

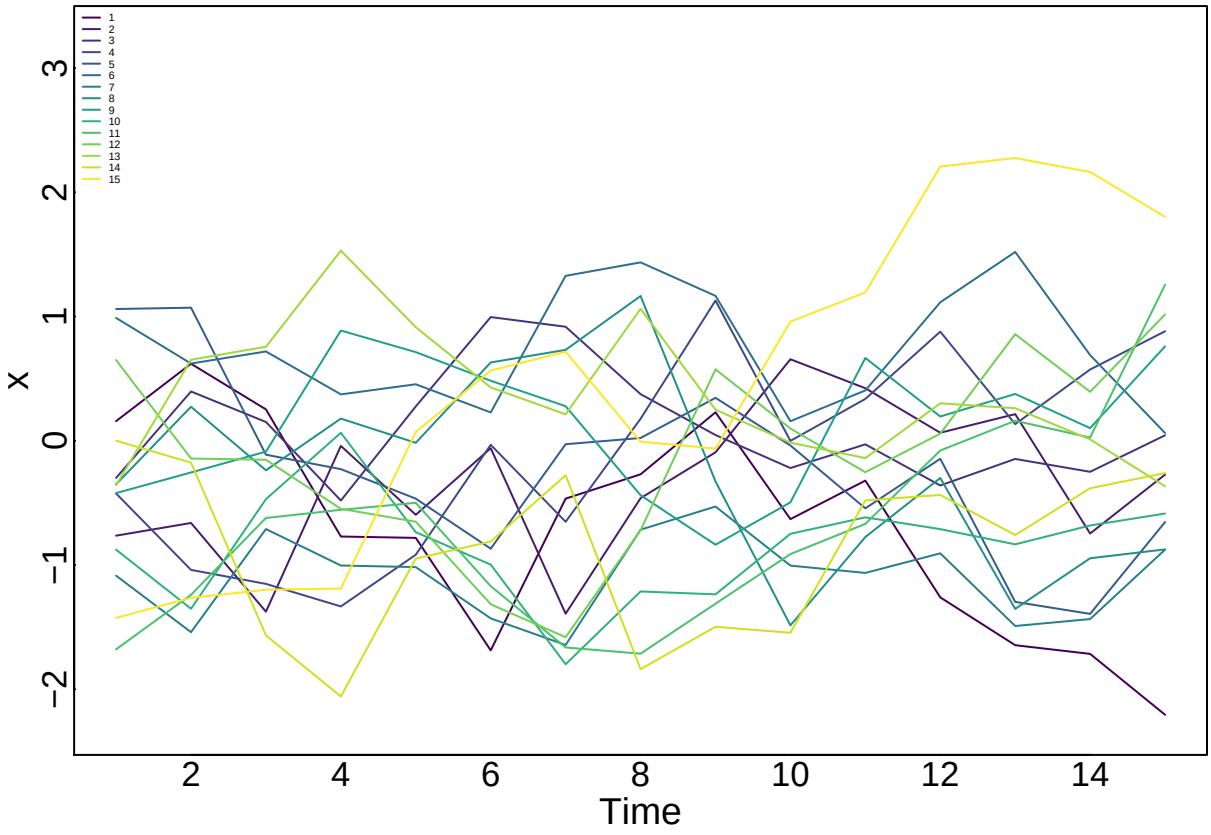
```
## Loading required package: viridisLite
```



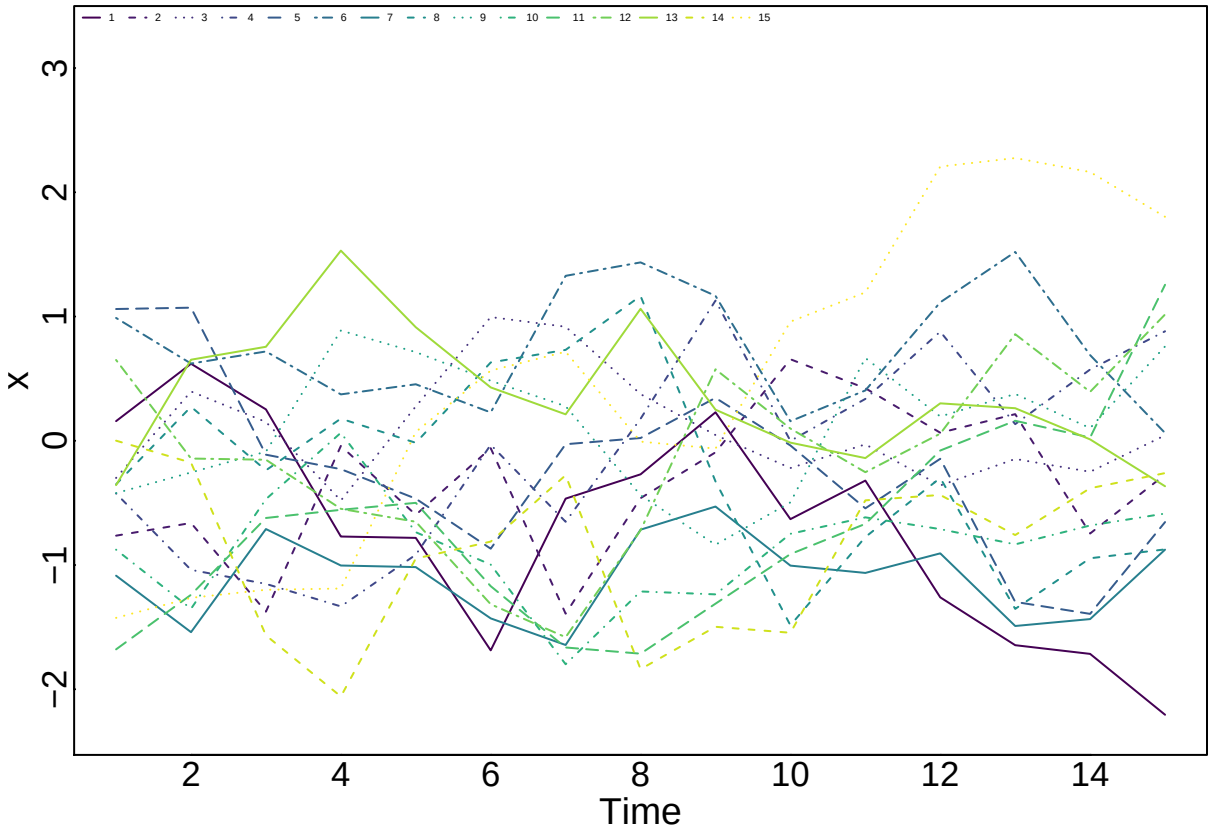

```
col = viridis(15)
plot(data$x[data$Group == 1] ~ unique(data$Time), type = 'l',
     xlab = 'Time', ylab = 'x', ylim = c(min(data$x)-.1, max(data$x) +.1),
     col = col[1])
for(i in 2:max(data$Group)) points(data$x[data$Group == i] ~ unique(data$Time), col = col[i], type = 'l
```



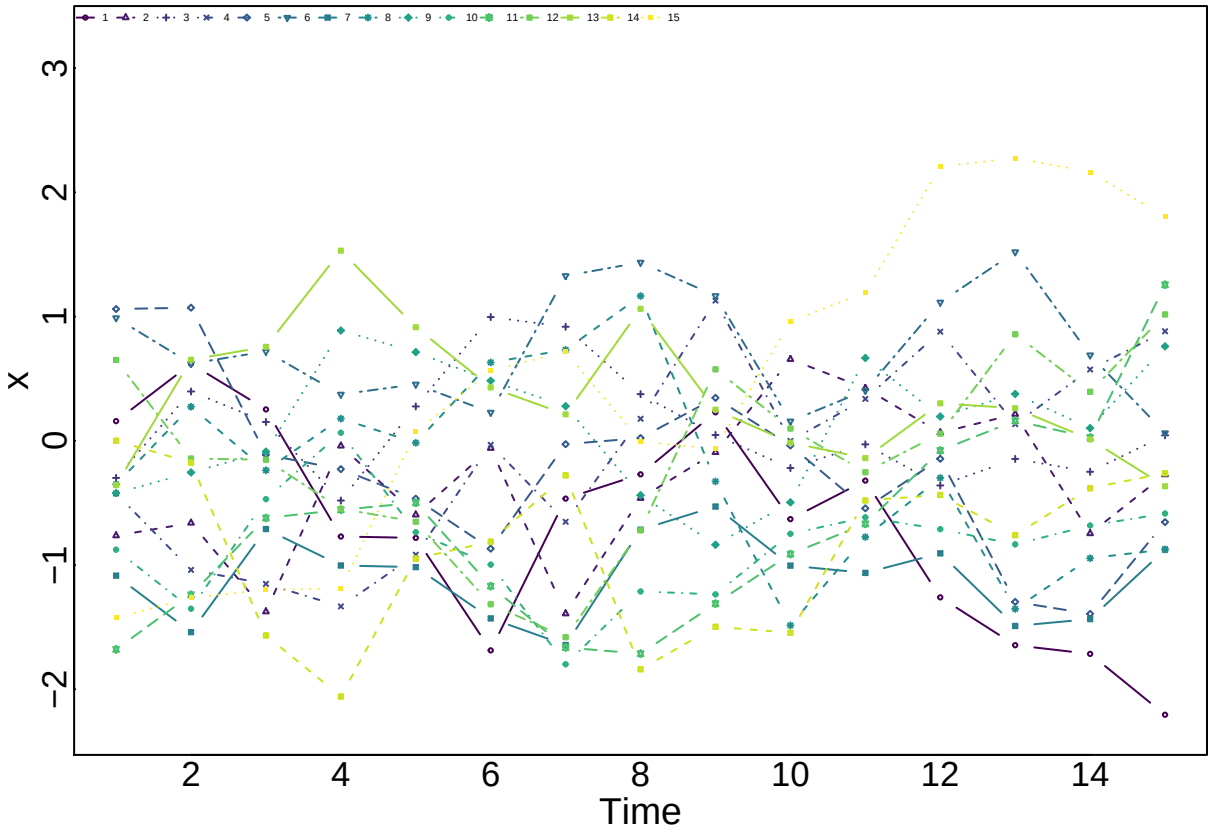

```r
#lets increase the size of the window to fit a legend
plot(data$x[data$Group == 1] ~ unique(data$Time), type = 'l',
     xlab = 'Time', ylab = 'x', ylim = c(min(data$x)-.1, max(data$x) +1),
     col = col[1])
for(i in 2:max(data$Group)) points(data$x[data$Group == i] ~ unique(data$Time), col = col[i], type = 'l
legend('topleft', legend = 1:max(data$Group), col = col, lty = 1, cex = .3, bty = 'n')
```



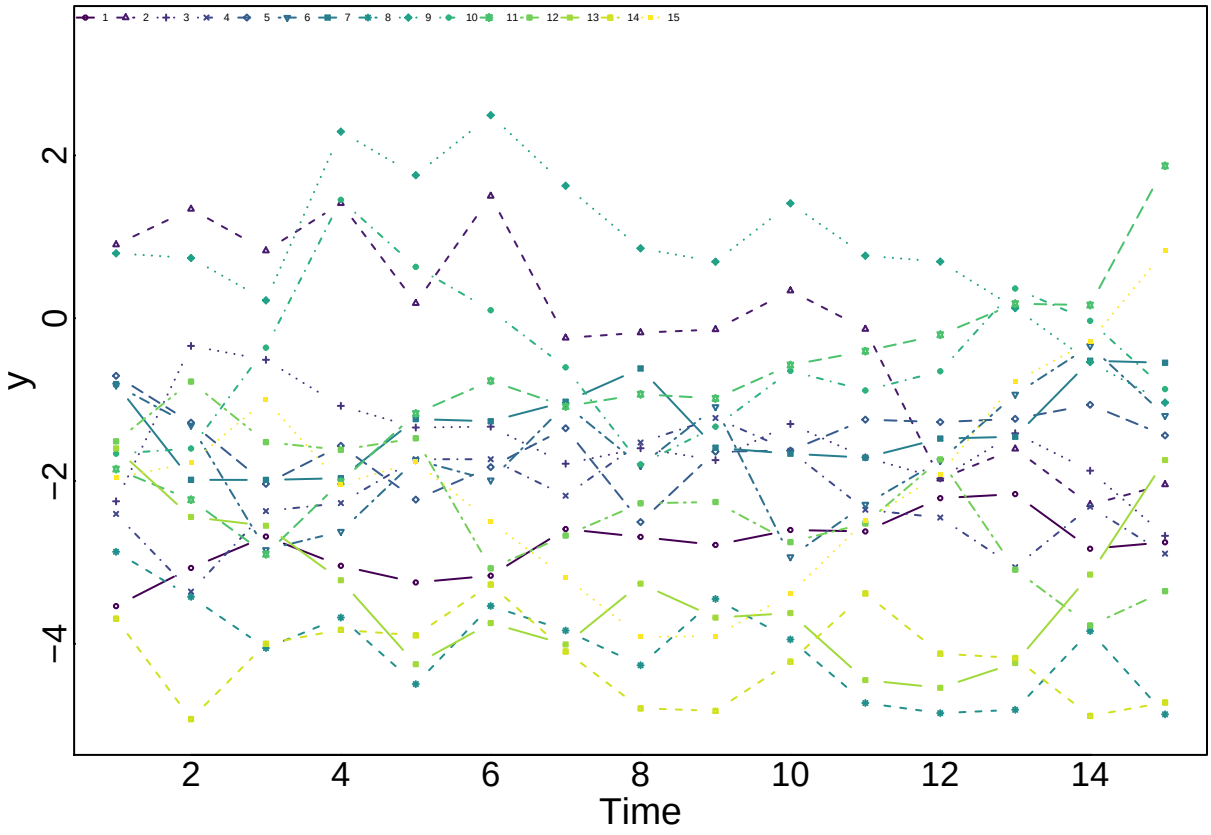

```
#what about line types for greyscale?
plot(data$x[data$Group == 1] ~ unique(data$Time), type = 'l',
     xlab = 'Time', ylab = 'x', ylim = c(min(data$x)-.1, max(data$x) +1),
     col = col[1])
for(i in 2:max(data$Group)) points(data$x[data$Group == i] ~ unique(data$Time), col = col[i], type = 'l
legend('topleft', legend = 1:max(data$Group), col = col, lty = 1:max(data$Group), cex = .3, bty = 'n', l
```



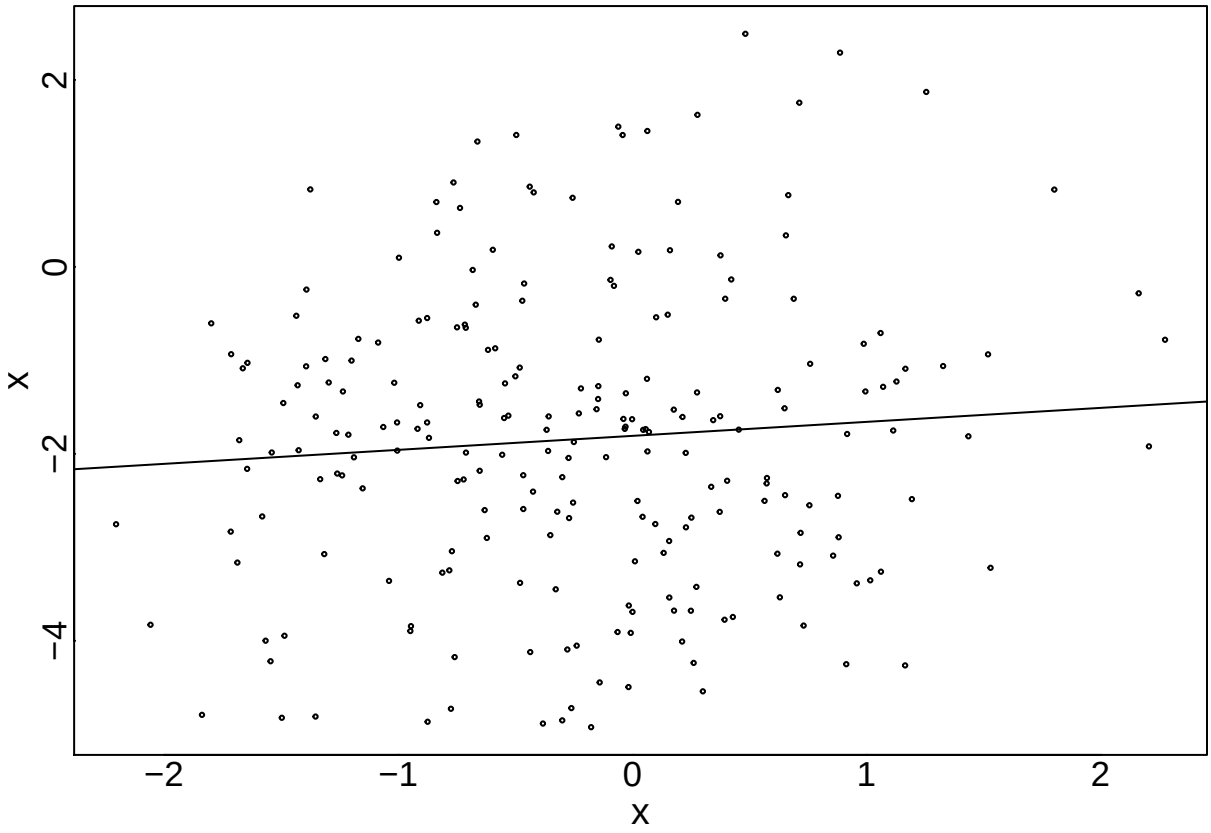

```
#notice the recycling of lty - lets add points as well
plot(data$x[data$Group == 1] ~ unique(data$Time), type = 'b',
     xlab = 'Time', ylab = 'x', ylim = c(min(data$x)-.1, max(data$x) +1),
     col = col[1], pch = 1, cex = .3)
for(i in 2:max(data$Group)) points(data$x[data$Group == i] ~ unique(data$Time), col = col[i], type = 'b
legend('topleft', legend = 1:max(data$Group), col = col, lty = 1:max(data$Group), pch = 1:max(data$Grou
```



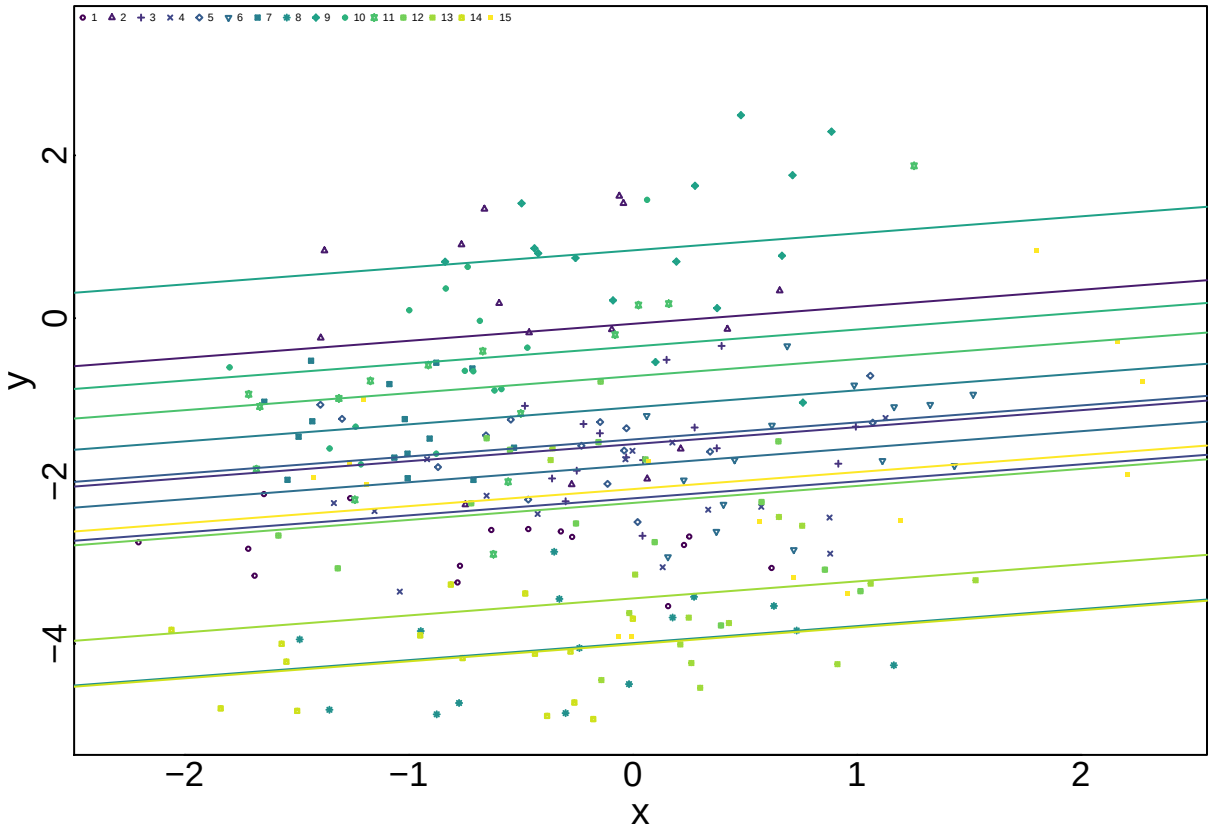

```
#lets repeat for y to investigate the auto-correlation in the outcome
plot(data$y[data$Group == 1] ~ unique(data$Time), type = 'b',
     xlab = 'Time', ylab = 'y', ylim = c(min(data$y)-.1, max(data$y) +1),
     col = col[1], pch = 1, cex = .3)
for(i in 2:max(data$Group)) points(data$y[data$Group == i] ~ unique(data$Time), col = col[i], type = 'b
legend('topleft', legend = 1:max(data$Group), col = col, lty = 1:max(data$Group), pch = 1:max(data$Grou
```



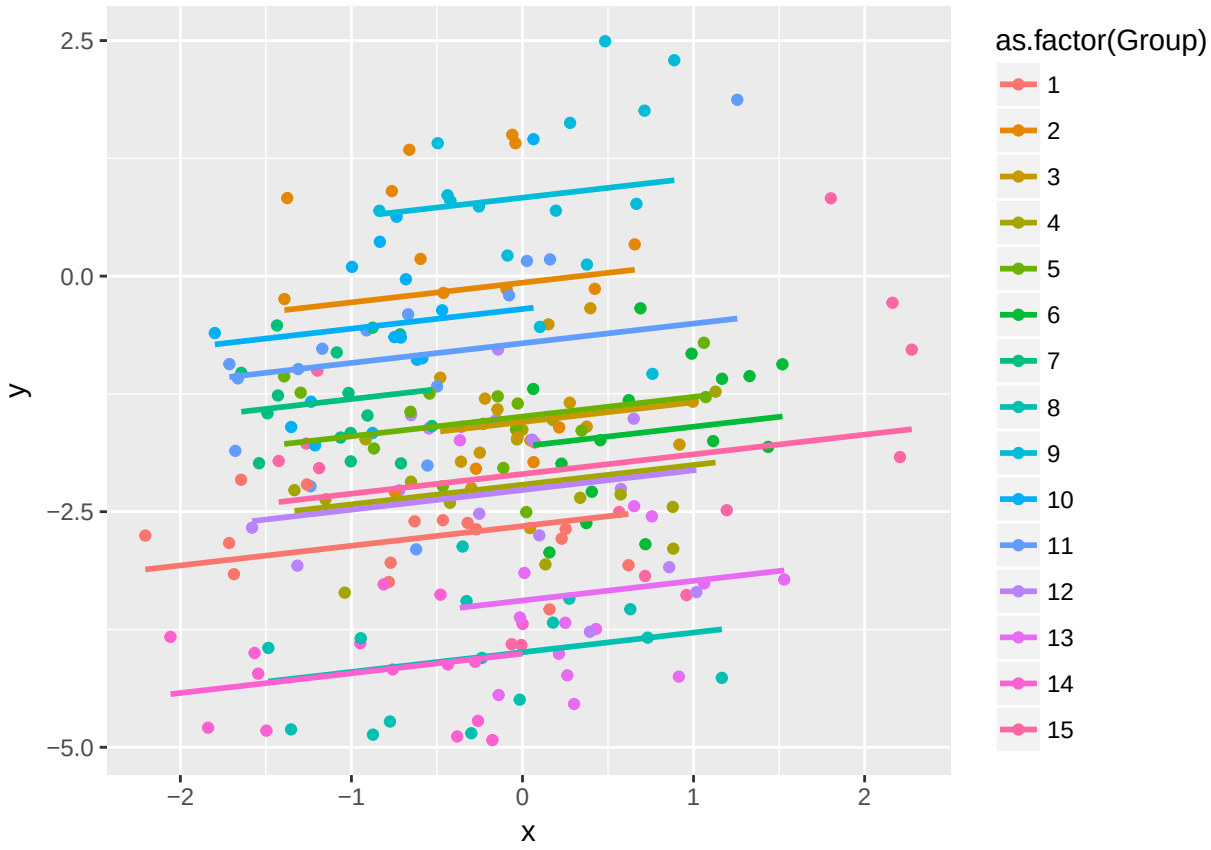

```
#notice there are also different baselines
#now lets plot y as a function of x, first all points
plot(data$y ~ data$x, type = 'p',
     xlab = 'x', ylab = 'x', cex = .3)
#add a line of best fit
abline(lm(y ~ x, data))
```



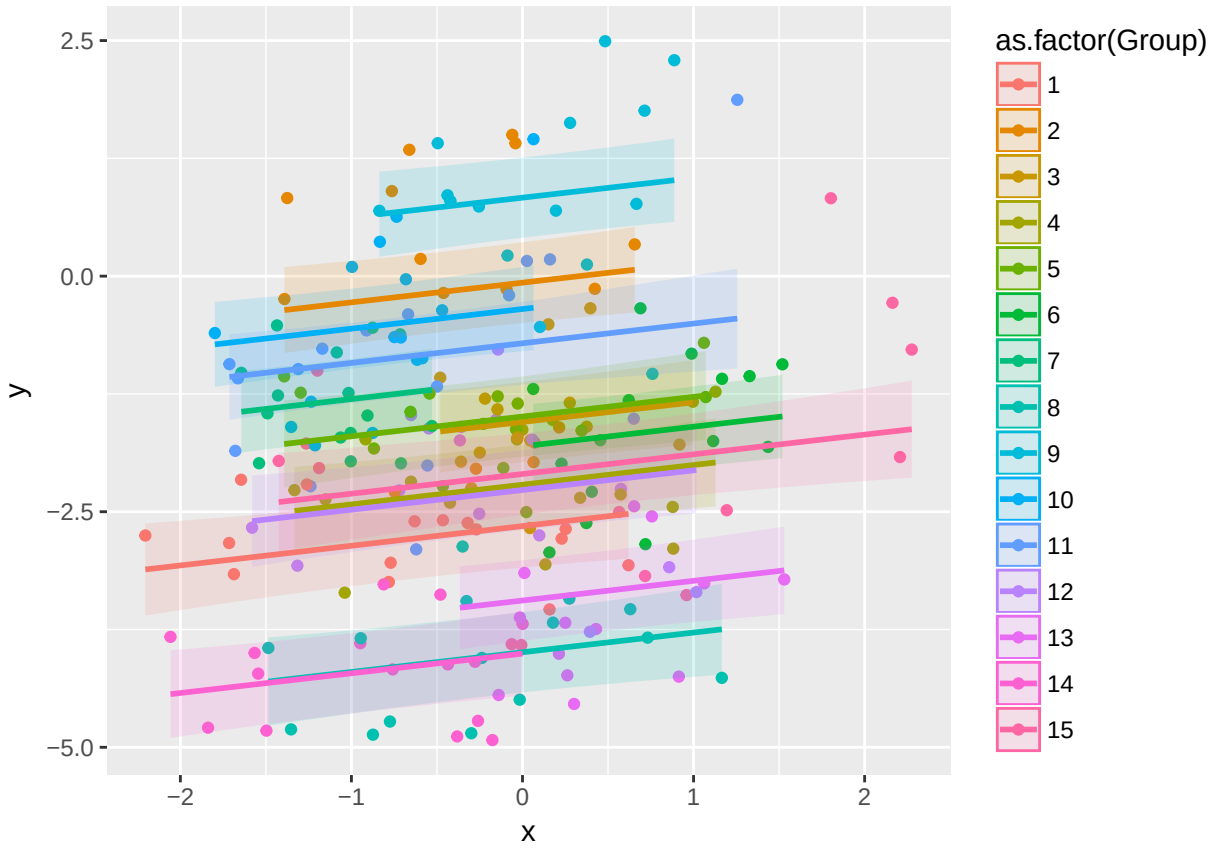

```
#now lets allow different baselines
#notice the as.factor
mod = lm(y ~ x + as.factor(Group) - 1, data)
plot(data$y[data$Group == 1] ~ data$x[data$Group == 1], type = 'p',
     xlab = 'x', ylab = 'y', ylim = c(min(data$y)-.1, max(data$y) +1),
     xlim = c(min(data$x)-.1, max(data$x) +.1),
     col = col[1], pch = 1, cex = .3)
for(i in 2:max(data$Group)) points(data$y[data$Group == i] ~ data$x[data$Group == i], col = col[i], typ
legend('topleft', legend = 1:max(data$Group), col = col,
       pch = 1:max(data$Group), cex = .3, bty = 'n', horiz = T)
#add the lines
for(i in 2:max(data$Group)) abline(a = coef(mod)[i+1], b = coef(mod)[1], col = col[i])
```




```
#now lets use ggplot2
data$pred = predict(mod)
library(ggplot2)
ggplot(data, aes(x = x, y = y, color = as.factor(Group)) ) +
     geom_point() +
     geom_line(aes(y = pred), size = 1)
```




```
#lets add confidence intervals
predslm = predict(mod, interval = "confidence")
data = cbind(data, predslm)
ggplot(data, aes(x = x, y = y, color = as.factor(Group)) ) +
     geom_point() +
     geom_ribbon( aes(ymin = lwr, ymax = upr, fill = as.factor(Group), color = NULL), alpha = .15) +
     geom_line( aes(y = fit), size = 1)
```

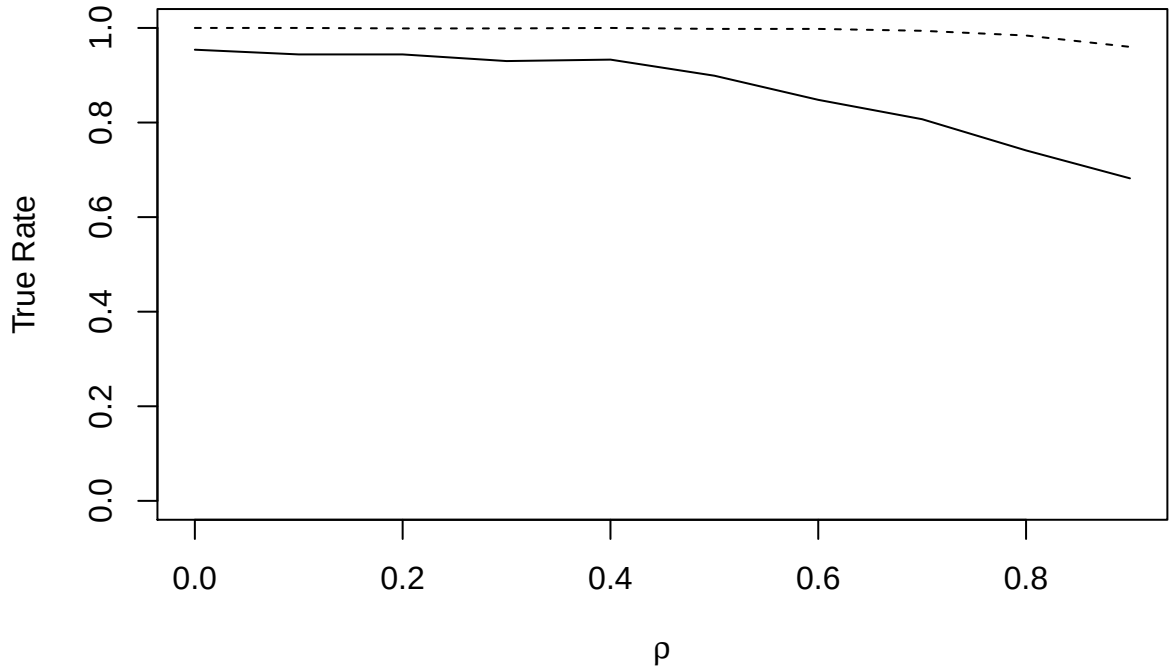


Analyze the generated TSCS data and plot the results

```
#we want to simulate a bunch of data sets at a range of rho, run a linear model, and check for coverage
simAnalysis = function(rho = seq(0, .9, by = .1),
                       n.datasets = 1000, ...){ #the ... notation allows additional arguments to be sen
  datas = lapply(1:n.datasets, function(y){
    lapply(rho, function(x){
      dd = tscsGen(x, seed = 100*y + 10*x, ...)
      mod = lm(y ~ x + as.factor(Group) - 1, dd)
      cov = confint(mod)[1,1] < .4 & confint(mod)[1,2] > .4
      pos = confint(mod)[1,1] > 0
      return(list(cov, pos))
  })})
  datas = unlist(datas)
  #so, it looped through each rho nested within each dataset
  datas.cov = datas[seq(1, length(datas), by = 2)]
  datas.pos = datas[seq(2, length(datas), by = 2)]
  covsByRho = numeric(length(rho))
  possByRho = numeric(length(rho))
  for(i in 1:length(rho)){
    covsByRho[i] = mean(datas.cov[seq(i, length(datas.cov), by = length(rho))])
    possByRho[i] = mean(datas.pos[seq(i, length(datas.pos), by = length(rho))])
  }
  return(list(covsByRho, possByRho))
} #hw - parallelize this code
```

```
toAnalyze = simAnalysis()
save(toAnalyze, file = 'sims.Rdata')
```

```
load('sims.Rdata')
plot(toAnalyze[[1]] ~ seq(0, .9, by=.1), ylim = c(0,1), type = 'l',
     xlab = expression(rho), ylab = 'True Rate')
points(toAnalyze[[2]] ~ seq(0, .9, by=.1), type='l', lty = 2)
```



Coefficient plots

```
set.seed(99)
X = matrix(rnorm(1000*3), ncol = 3, nrow = 1000)
y = -1.5 + X%*%c(.5, -1.2, .3) + rnorm(1000)
mod = lm(y ~ X)
coef(mod)
```

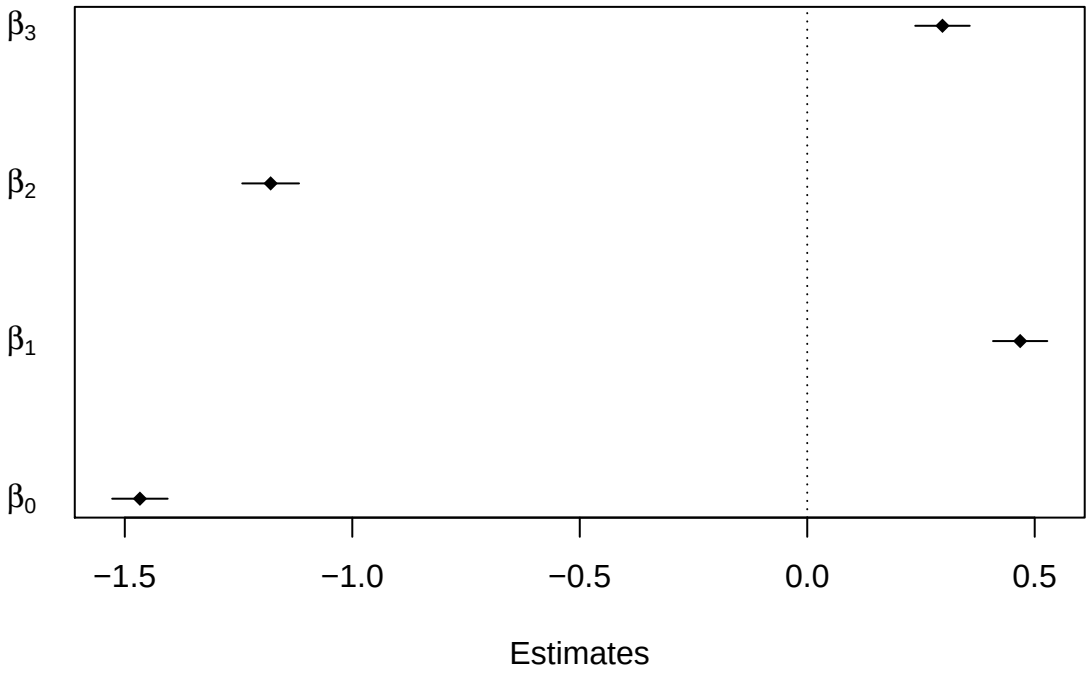
```
## (Intercept)          X1          X2          X3
##  -1.4667980   0.4680407  -1.1794593   0.2972580
```

```
confint(mod)
```

```
##                 2.5 %     97.5 %
## (Intercept) -1.5277685 -1.4058275
## X1           0.4083956  0.5276857
## X2          -1.2418564 -1.1170621
## X3           0.2373861  0.3571299
```

```
plot(coef(mod), 1:4, yaxt = 'n', pch = 18,
     xlim = c(min(confint(mod)), max(confint(mod))),
     ylab = '', xlab = 'Estimates')
segments(x0 = confint(mod)[,1], x1 = confint(mod)[,2],
         y0 = 1:4, y1 = 1:4)
```

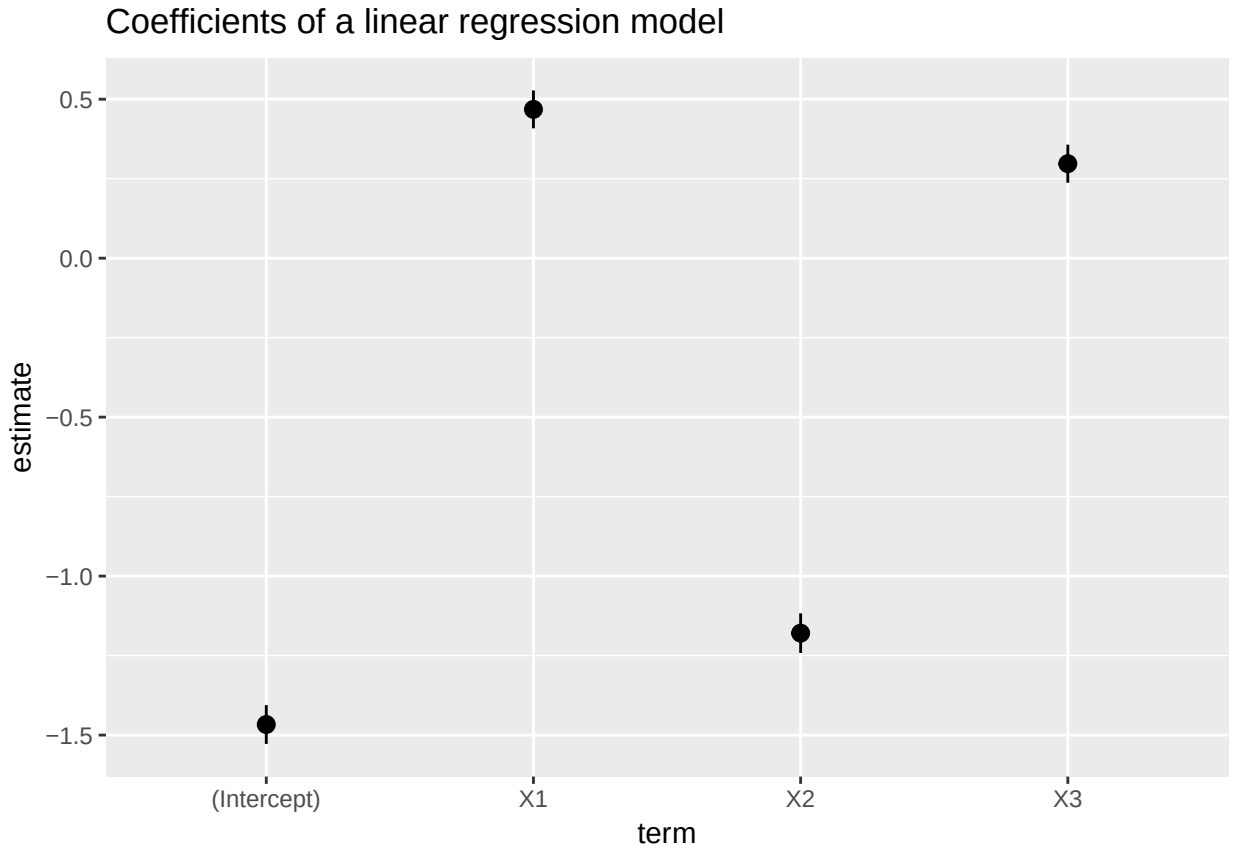
```
abline(v=0, lty = 3)
axis(2, at = 1:4,
     tick = F, labels = c(expression(beta['0']),
                          expression(beta['1']),
                          expression(beta['2']),
                          expression(beta['3'])),
     las = 1)
```




```
library(broom)
coef = tidy(mod, conf.int = T)
coef
```

```
##          term   estimate  std.error  statistic       p.value   conf.low
## 1 (Intercept) -1.4667980 0.03107017 -47.209209 2.440994e-256 -1.5277685
## 2          X1  0.4680407 0.03039474  15.398738  3.725627e-48  0.4083956
## 3          X2 -1.1794593 0.03179718 -37.093208 7.165481e-190 -1.2418564
## 4          X3  0.2972580 0.03051034   9.742863  1.738102e-21  0.2373861
##    conf.high
## 1 -1.4058275
## 2  0.5276857
## 3 -1.1170621
## 4  0.3571299
```

```
ggplot(coef, aes(term, estimate))+
  geom_point()+
  geom_pointrange(aes(ymin = conf.low, ymax = conf.high))+
  labs(title = "Coefficients of a linear regression model")
```



Exponential family

- Fundamental to generalized linear models
- Puts PDF's in a generalized form
- Subfunctions are contained within the exponent component of the natural exponential function
- The isolated subfunctions quite naturally produce a small number of statistics that compactly summarize even large datasets without any loss of information
- Sufficient statistic: For some parameter contains all the information available in a given dataset about that parameter
- Suppose we consider a one-parameter conditional probability density function or probability mass function for the random variable Z of the form $F(z|\zeta)$, it is exponential if it can be written as:

$$\begin{aligned} f(z|\zeta) &= \exp[t(z)u(\zeta)]r(z)s(\zeta) \\ &= \exp\left[u(\zeta)\sum_{i=1}^{n} t(z_i) + \sum_{i=1}^{n}\log(r(z_i)) + n\log(s(\zeta))\right] \end{aligned}$$

- Canonical form:

$$f(y|\theta) = \exp[y\theta - b(\theta) + c(y)]$$

- Simply sum over dimension of θ for higher dimensionality
- Derive the exponential family form of the following:
  - Poisson

$$f(y|\mu) = \frac{e^{-u}\mu^y}{y!}$$

– Binomial (with known n)

$$f(y|n,p) = \binom{n}{p} p^y (1-p)^{n-y}$$

– Normal (for both μ and σ^2)

$$f(y|\mu,\sigma^2) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left[-\frac{1}{2\sigma^2}(y-\mu)^2\right]$$

– Gamma

$$f(y|\mu,\delta) = \left(\frac{\delta}{\mu}\right)^{\delta} \frac{1}{\Gamma(\delta)} y^{\delta-1} \exp\left[\frac{-\delta y}{\mu}\right]$$

– Negative binomial

$$f(y|r,p) = \binom{r+y-1}{y} p^r (1-p)^y$$

– Multinomial (assume three outcomes)

$$p(\mathbf{Y}_i = r|\mathbf{X}) = \frac{\exp(\mathbf{X}_i\beta_r)}{1 + \sum_{s=1}^{k-1} \exp(\mathbf{X}\beta_s))}$$