Ordinal Predicted Variable

Tim Frasier

Goals and General Idea

Goals
When would we use this type of analysis?

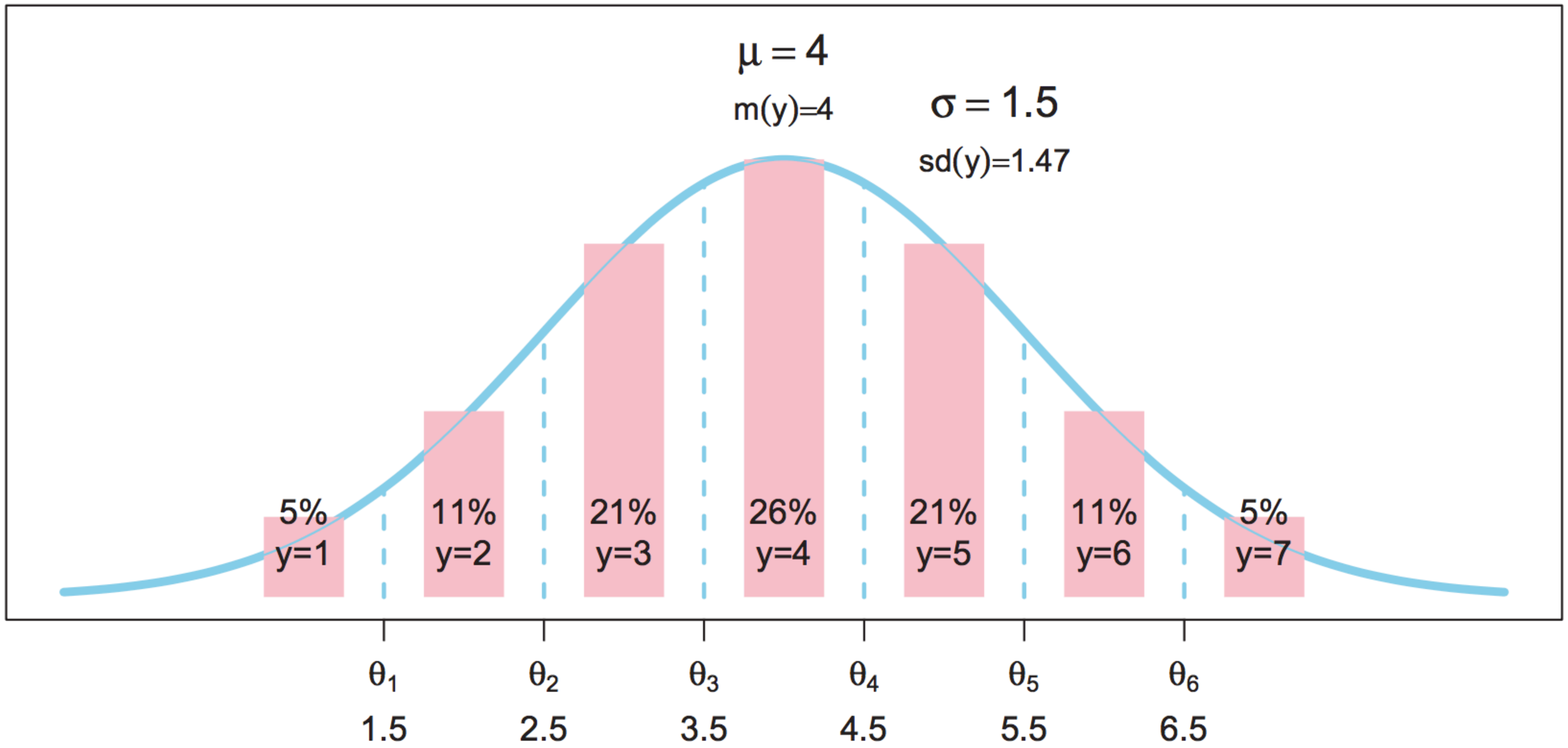
- When the predicted variable is ordinal!
  - Places in a race (1st, 2nd, 3rd, etc.)
  - Surveys on a Likert scale (5 = strongly agree, 4 = agree, 3 = neutral 2 = disagree, 1 = strongly disagree)
  - Scaled responses (good, mediocre, bad)
  - etc.

Characteristics

- Know order, but not necessarily equally spaced
  - How much do you like fish (1-hate to 5-love)?
    - May be harder to go from 1→2 than 4→5
- As predictor variables “increase”, should sequentially step through predicted values
  - How can we ensure this happens?

Characteristics

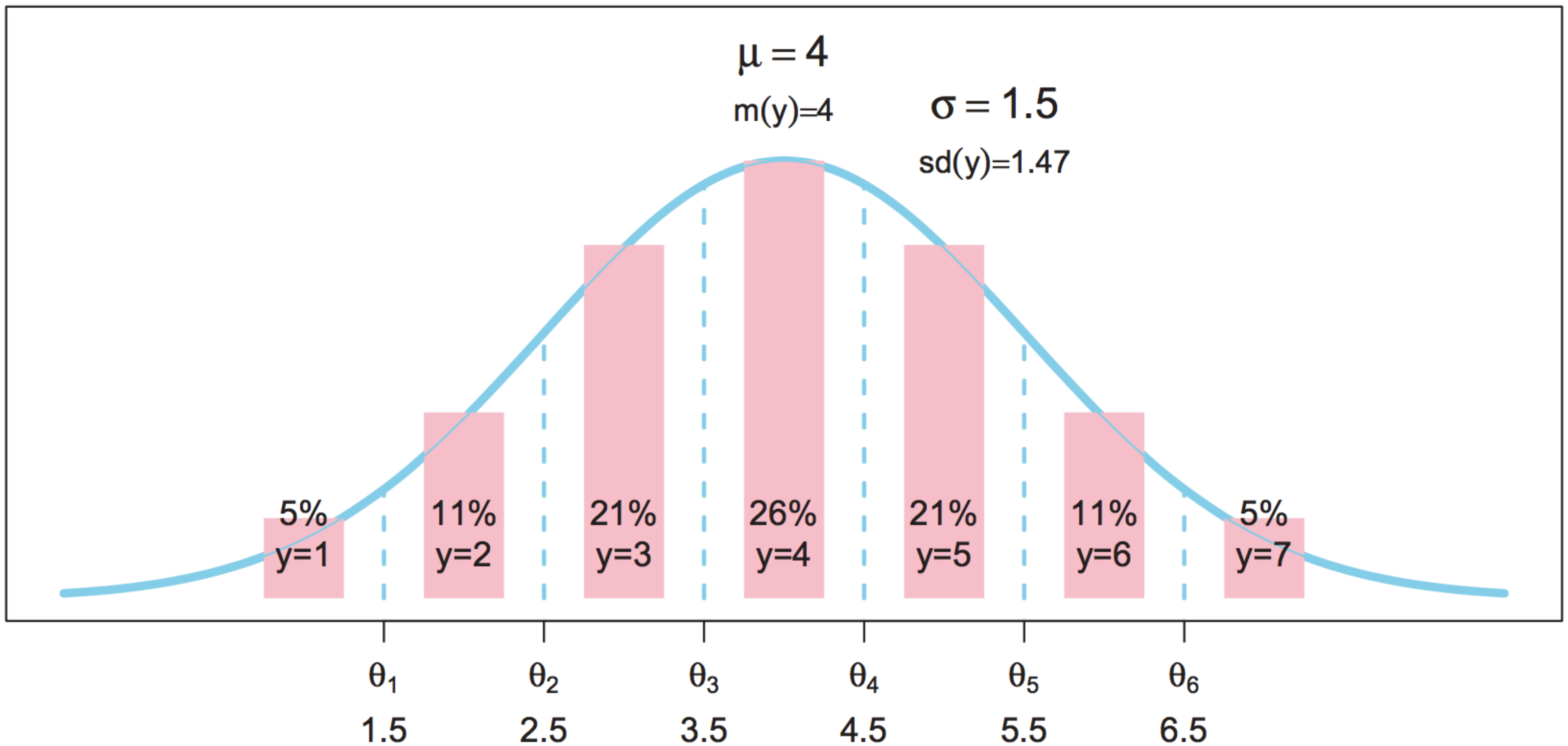
- Suppose ordinal data with 7 "levels"
- There will be cut-off points (thresholds) between levels, indicating where it switches from one to another (indicated here as θs)
- If there are k levels, there will be k-1 of these thresholds



From Kruschke (2015) p. 673

Characteristics

- How do we get probabilities for each level?
  - Cumulative distribution



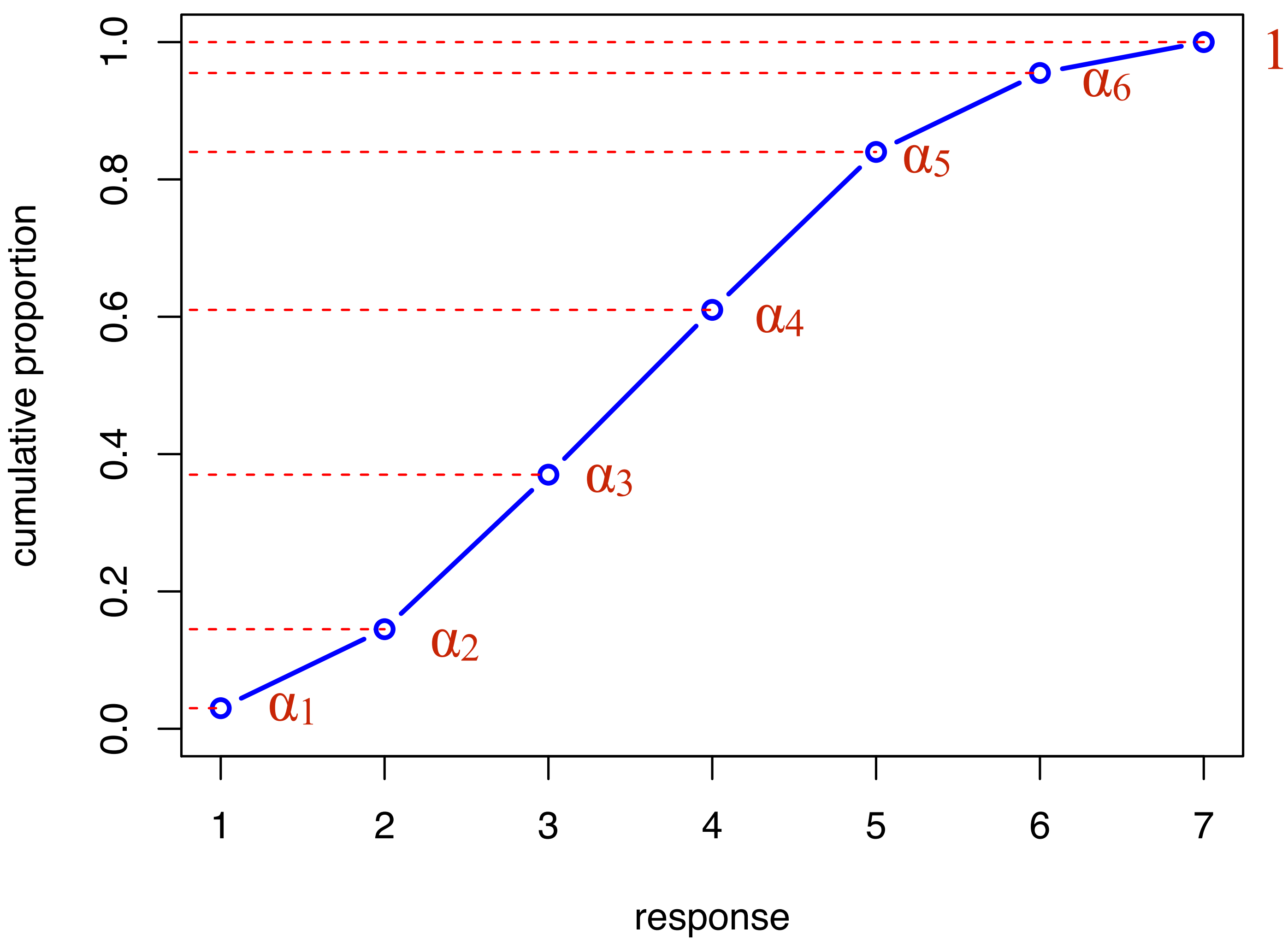
From Kruschke (2015) p. 673

Characteristics

- Now values range from 0 to 1

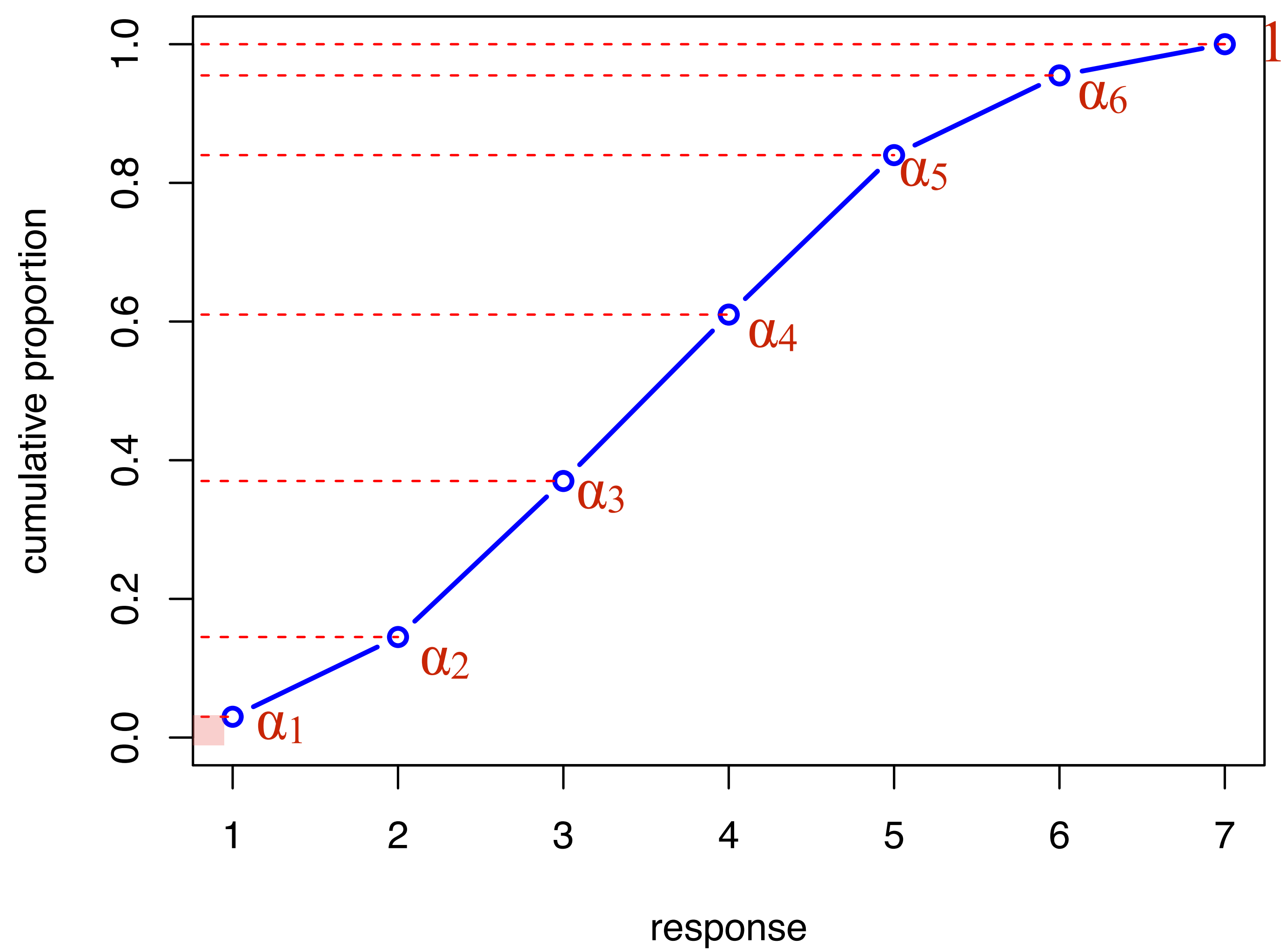
- Probability for each level is the cumulative area up to the threshold just above that level minus the cumulative area up to the threshold just below that level

- Call each threshold point an α value



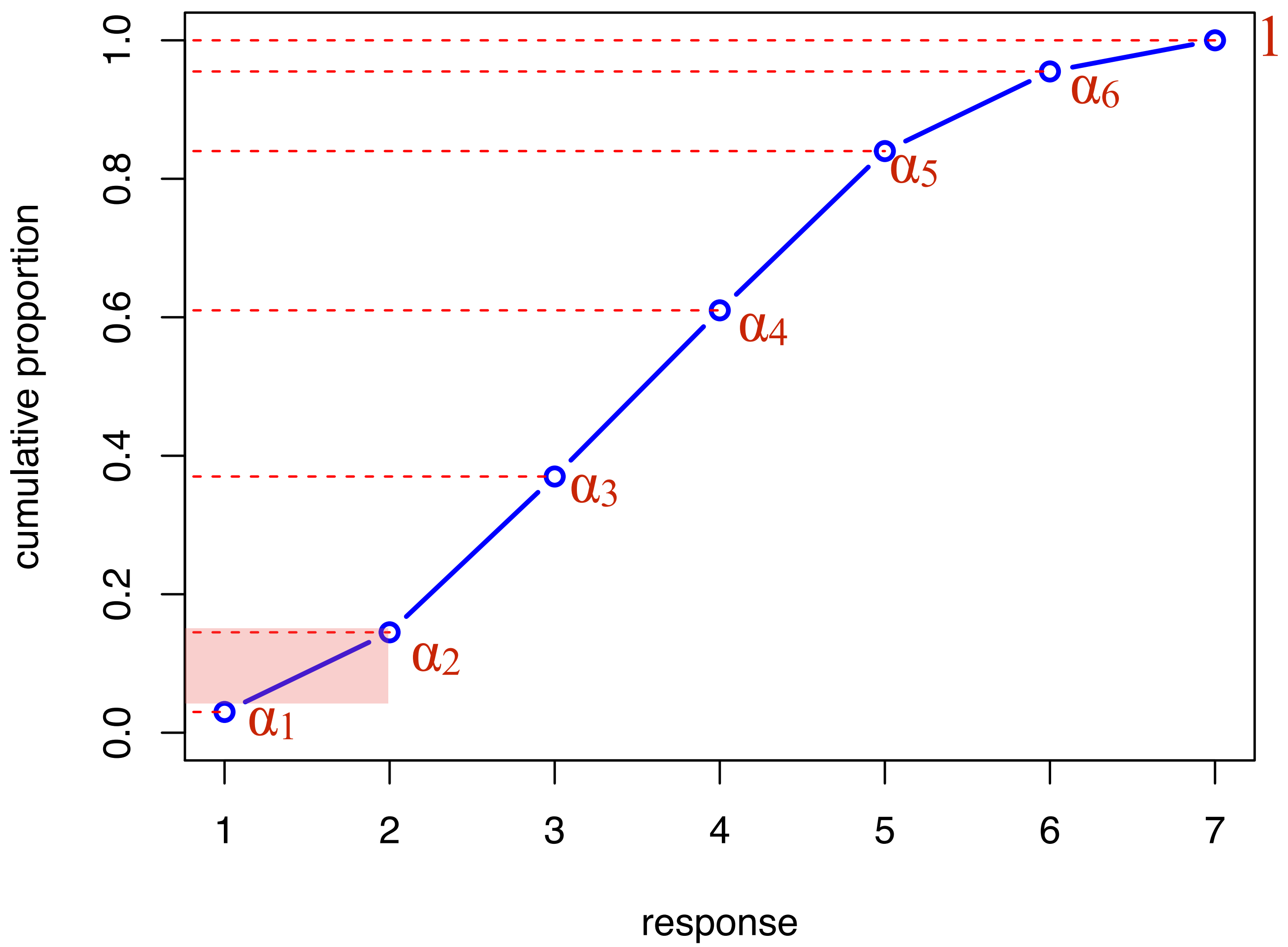
Characteristics

- For first category, probability is cumulative probability for that value, minus zero
  - Considering the mean and sd of the underlying distribution



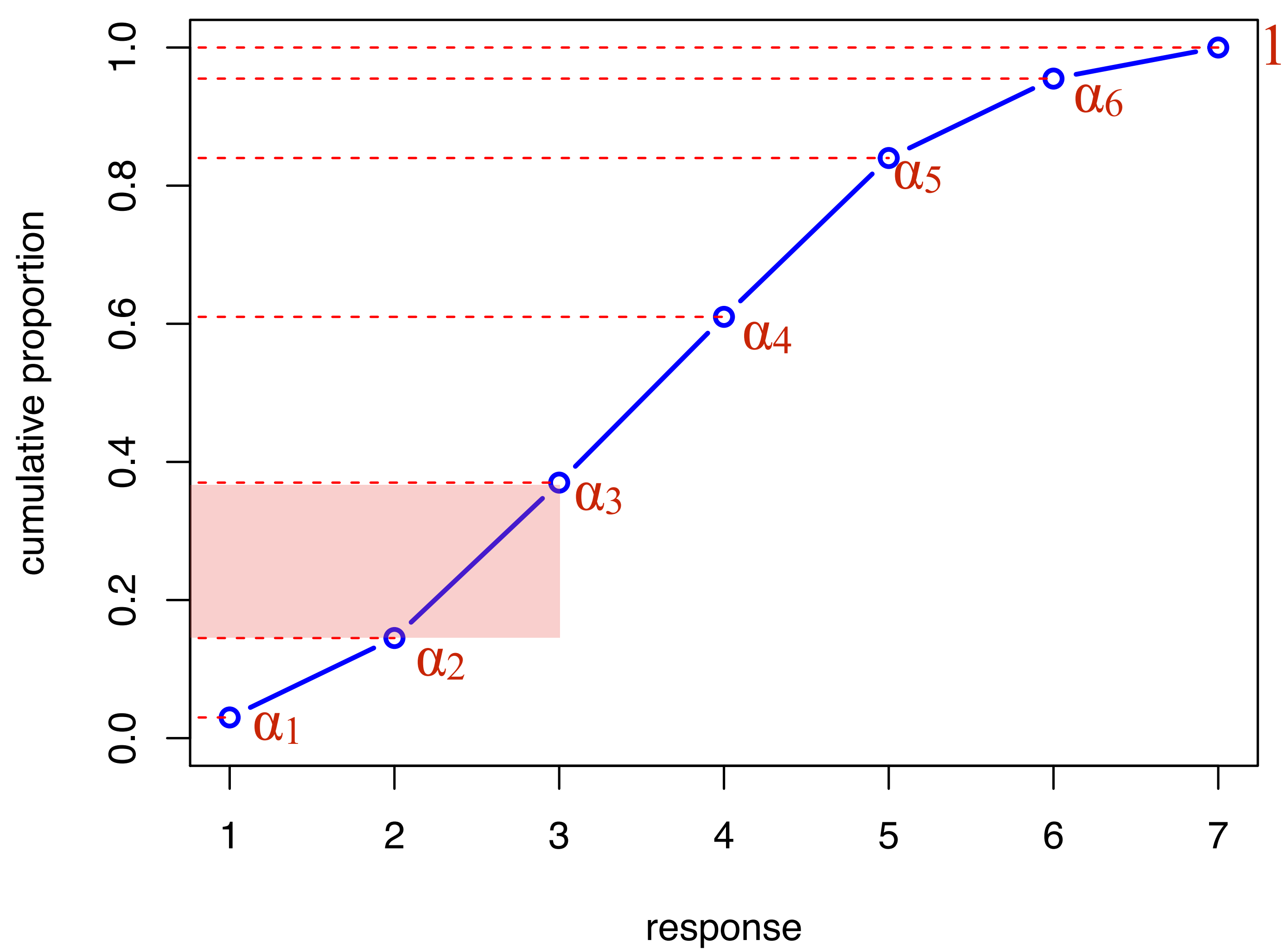
Characteristics

- For second category, probability is cumulative probability for that value, minus that for the first category



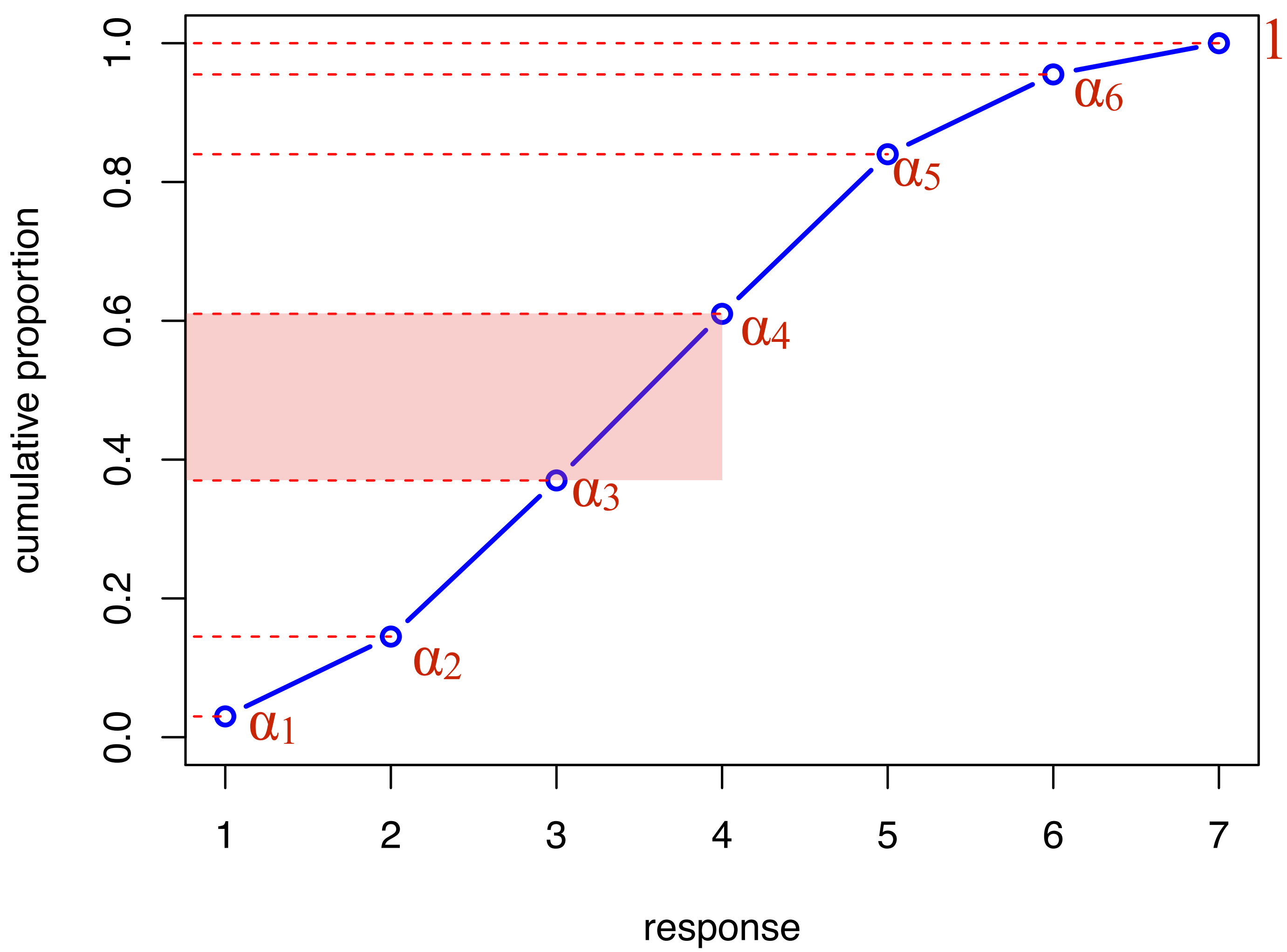
Characteristics

- For third category, probability is cumulative probability for that value, minus that for the second category



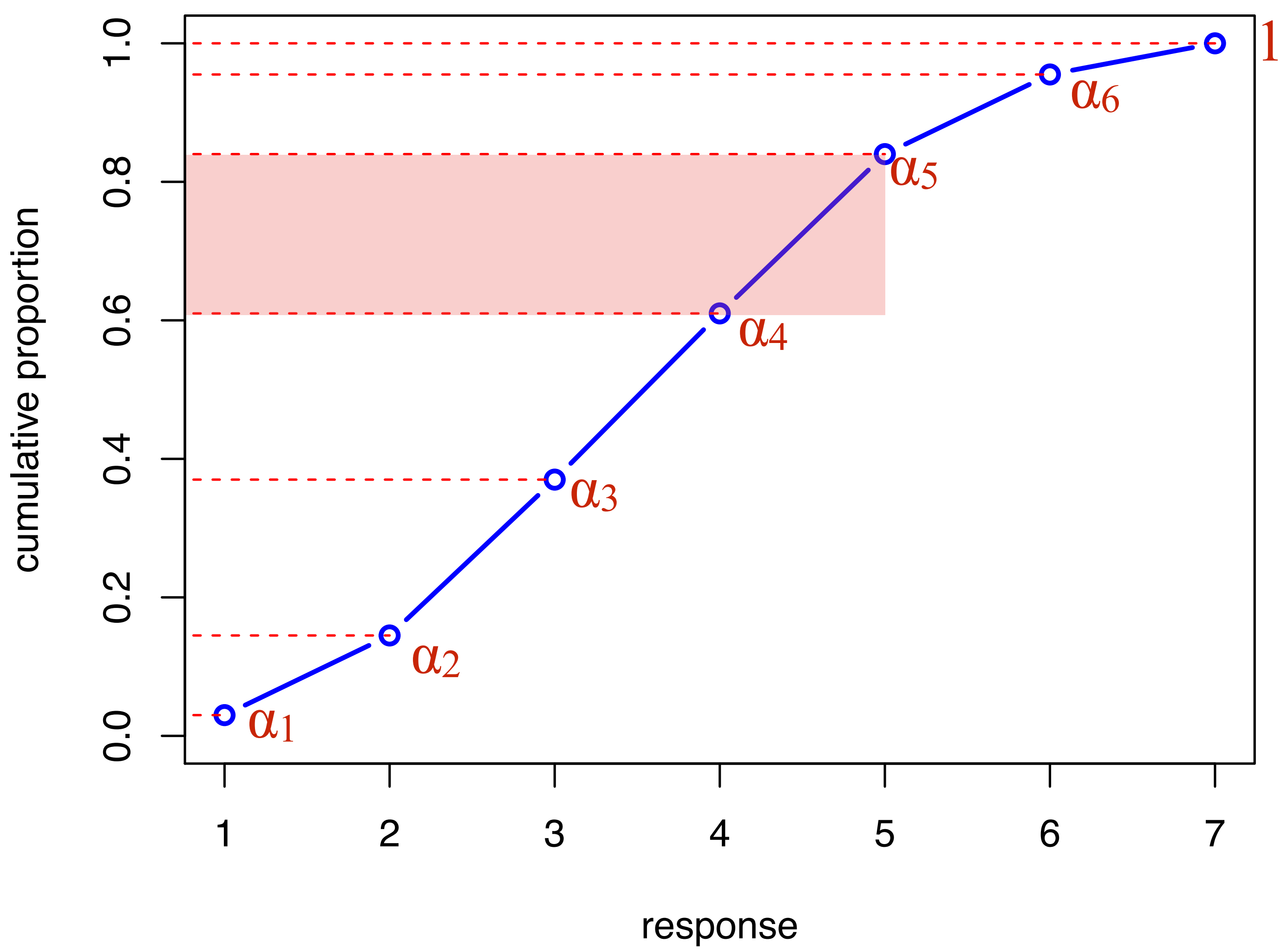
Characteristics

- For fourth category, probability is cumulative probability for that value, minus that for the third category



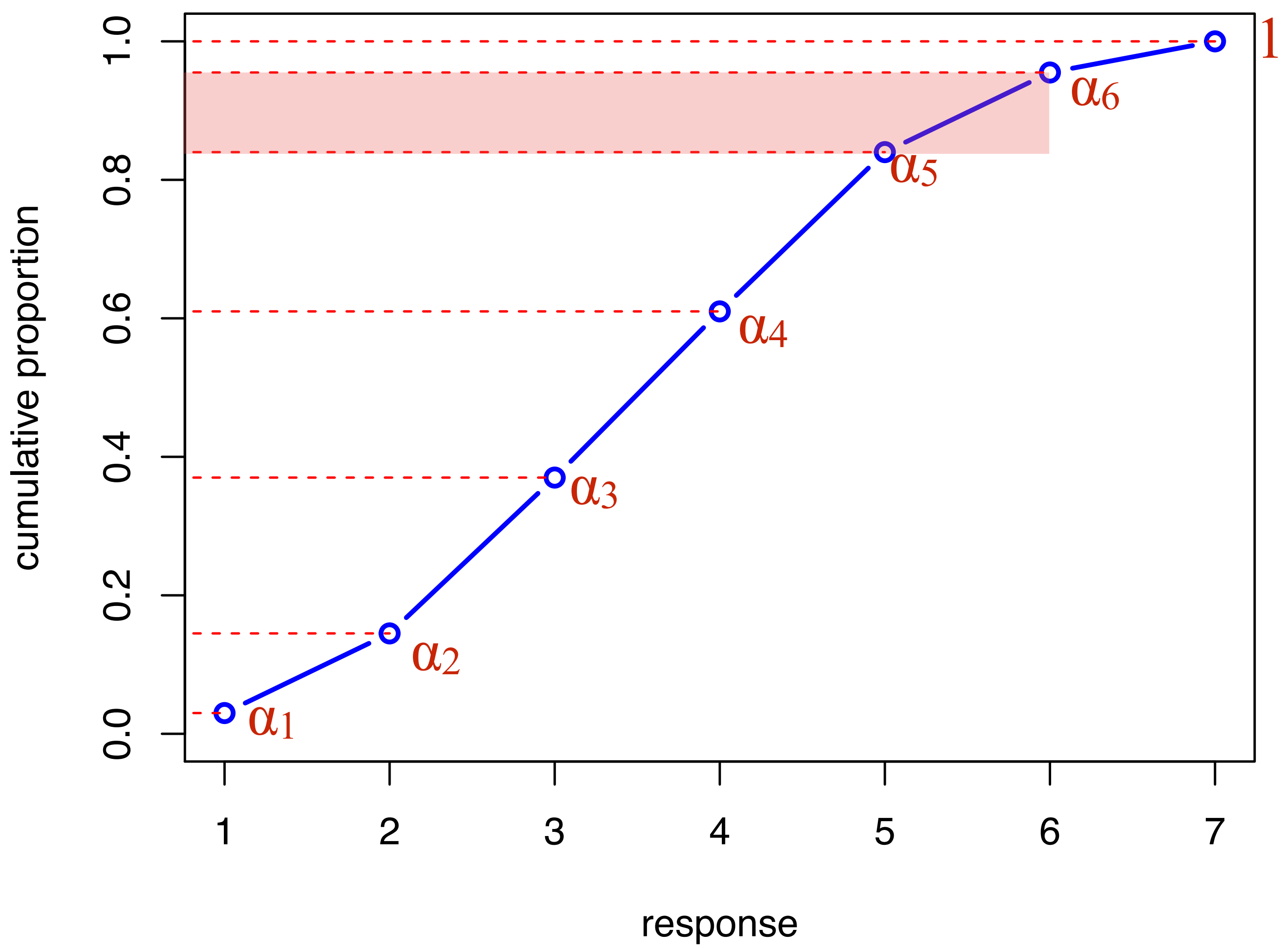
Characteristics

- For fifth category, probability is cumulative probability for that value, minus that for the fourth category



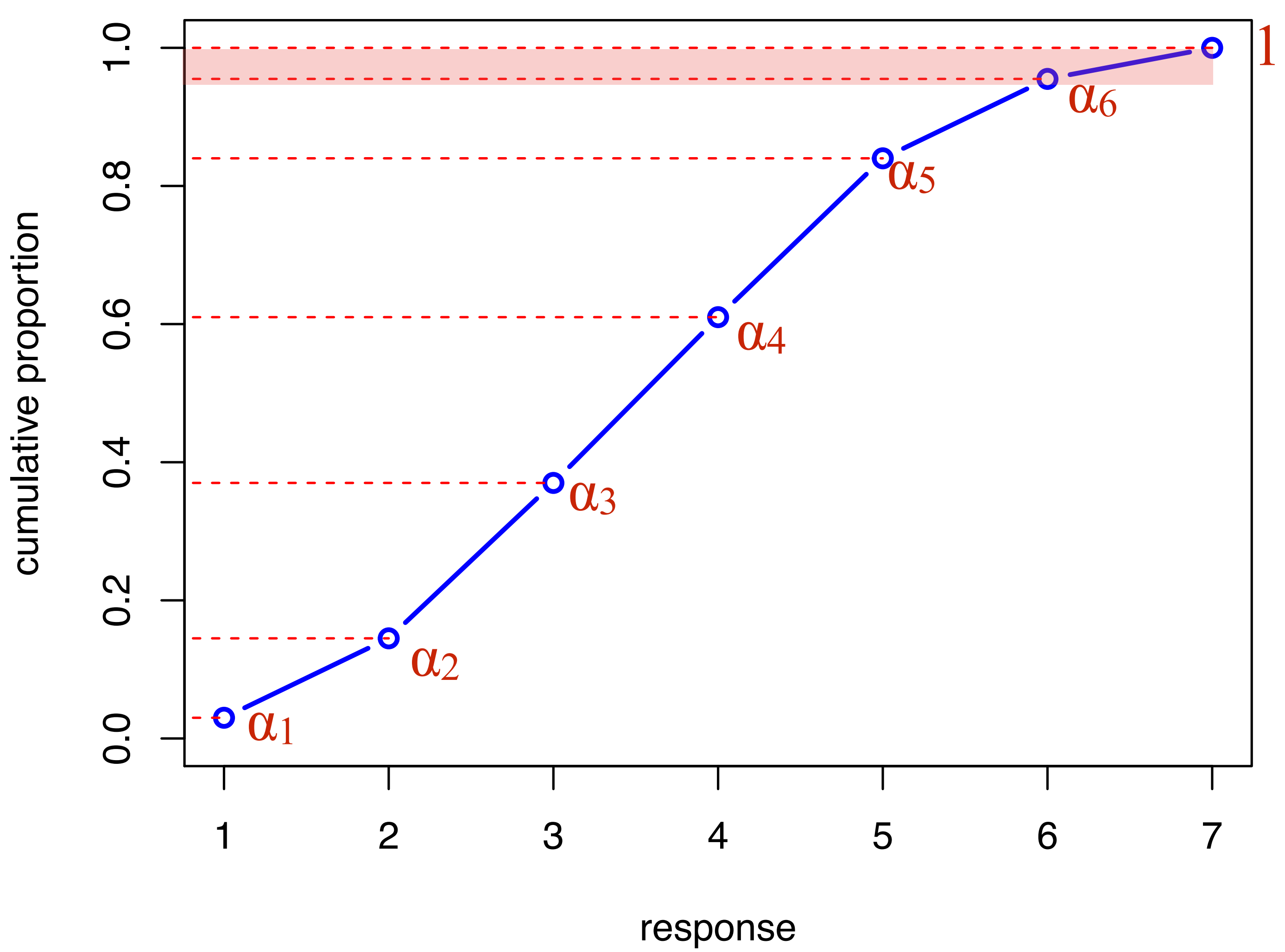
Characteristics

- For sixth category, probability is cumulative probability for that value, minus that for the fifth category



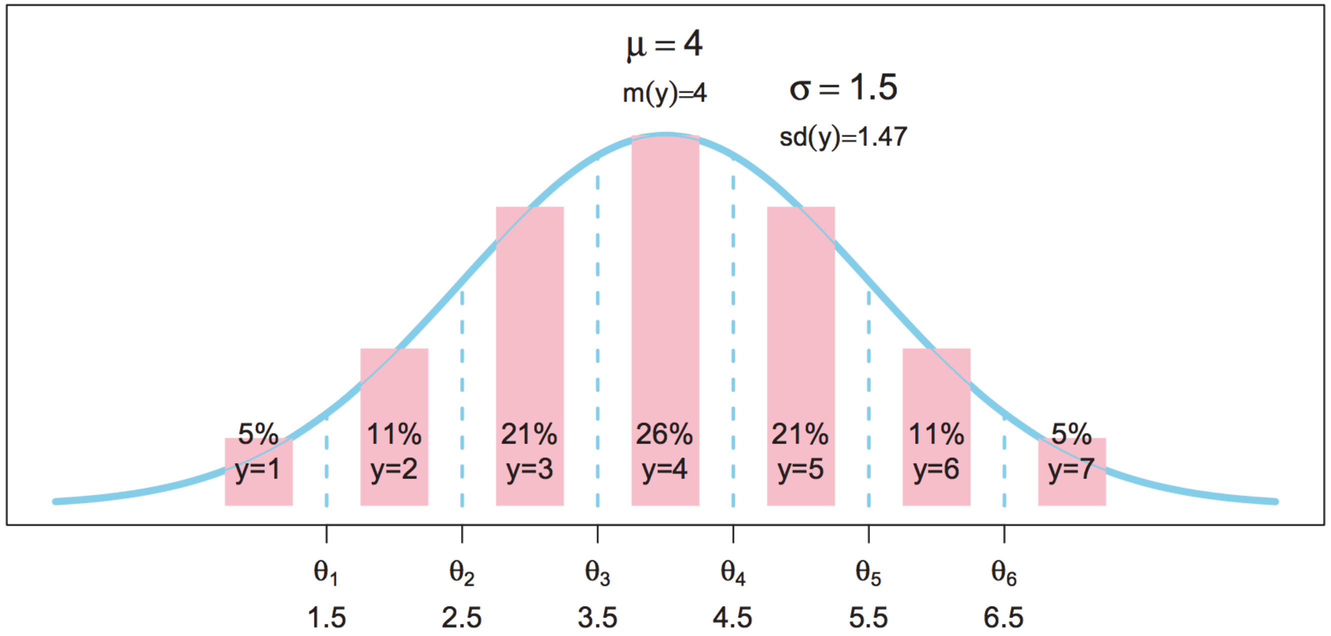
Characteristics

- For seventh category, probability is one, minus the cumulative probability for the 6th category



Characteristics

- The mean (*mu*) of this distribution is the result of the additive effect of our predictor variables
  - Our "standard" equation for the effects of the predictor variables goes into this *mu*



Characteristics

- What we are estimating:
  1. The α values for all but the first and last thresholds
  2. The mean (μ) of the underlying distribution (based on the additive effect of the predictor variables)
  3. The standard deviation (σ) of the underlying distribution
  4. Other appropriate distribution parameters if not using the normal distribution

The Data

Data

- Fake data generated from code in Kruschke (2011)

```
ord = read.table("ordinalData.csv", header = TRUE, sep = ",")
```

Y	X1	X2
4	1962.355	86.57301
5	1966.074	109.66692
5	1953.017	98.04524
4	1960.928	89.13965
3	1968.705	80.63023
7	1932.752	118.00796
3	1976.787	75.07696
5	1965.596	118.49754
3	1959.245	76.51348
3	1962.996	75.61261
3	1952.087	64.32620

Data

- Fake data generated f

Ordinal predicted variable

```
ord = read.table("ordinalData.csv", header = TRUE, sep = ",")
```

Y	X1	X2
4	1962.355	86.57301
5	1966.074	109.66692
5	1953.017	98.04524
4	1960.928	89.13965
3	1968.705	80.63023
7	1932.752	118.00796
3	1976.787	75.07696
5	1965.596	118.49754
3	1959.245	76.51348
3	1962.996	75.61261
3	1952.087	64.32620

Data

- Fake data generated from code in Kruschke (2011)

Two metric predictor variables

```
ord
ader = TRUE, sep = ",")
```

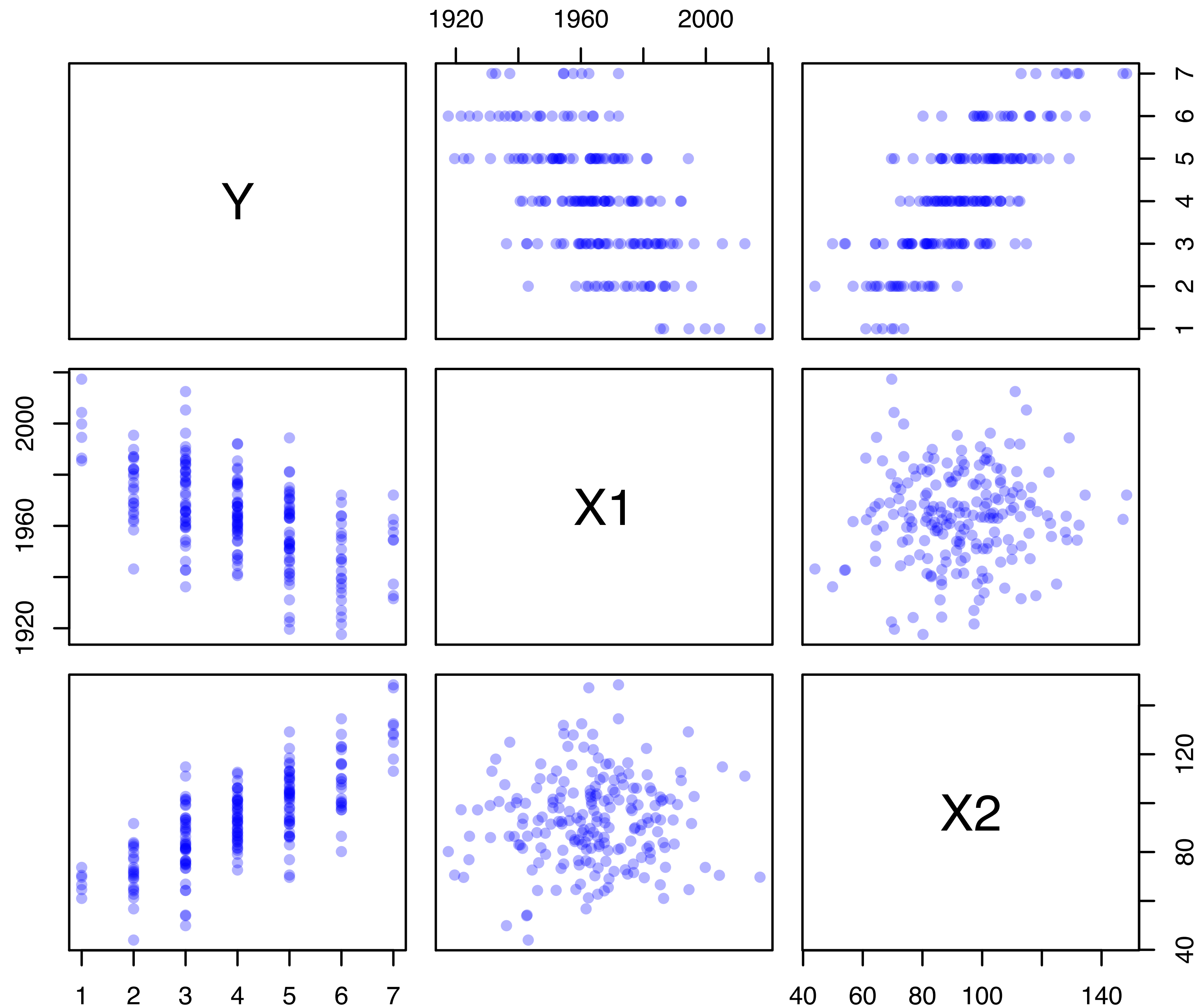
Y	X1	X2
4	1962.355	86.57301
5	1966.074	109.66692
5	1953.017	98.04524
4	1960.928	89.13965
3	1968.705	80.63023
7	1932.752	118.00796
3	1976.787	75.07696
5	1965.596	118.49754
3	1959.245	76.51348
3	1962.996	75.61261
3	1952.087	64.32620

Data

- Can use the `pairs` function to plot the data, and get some idea of potential patterns (keeping in mind the issue of interactions)

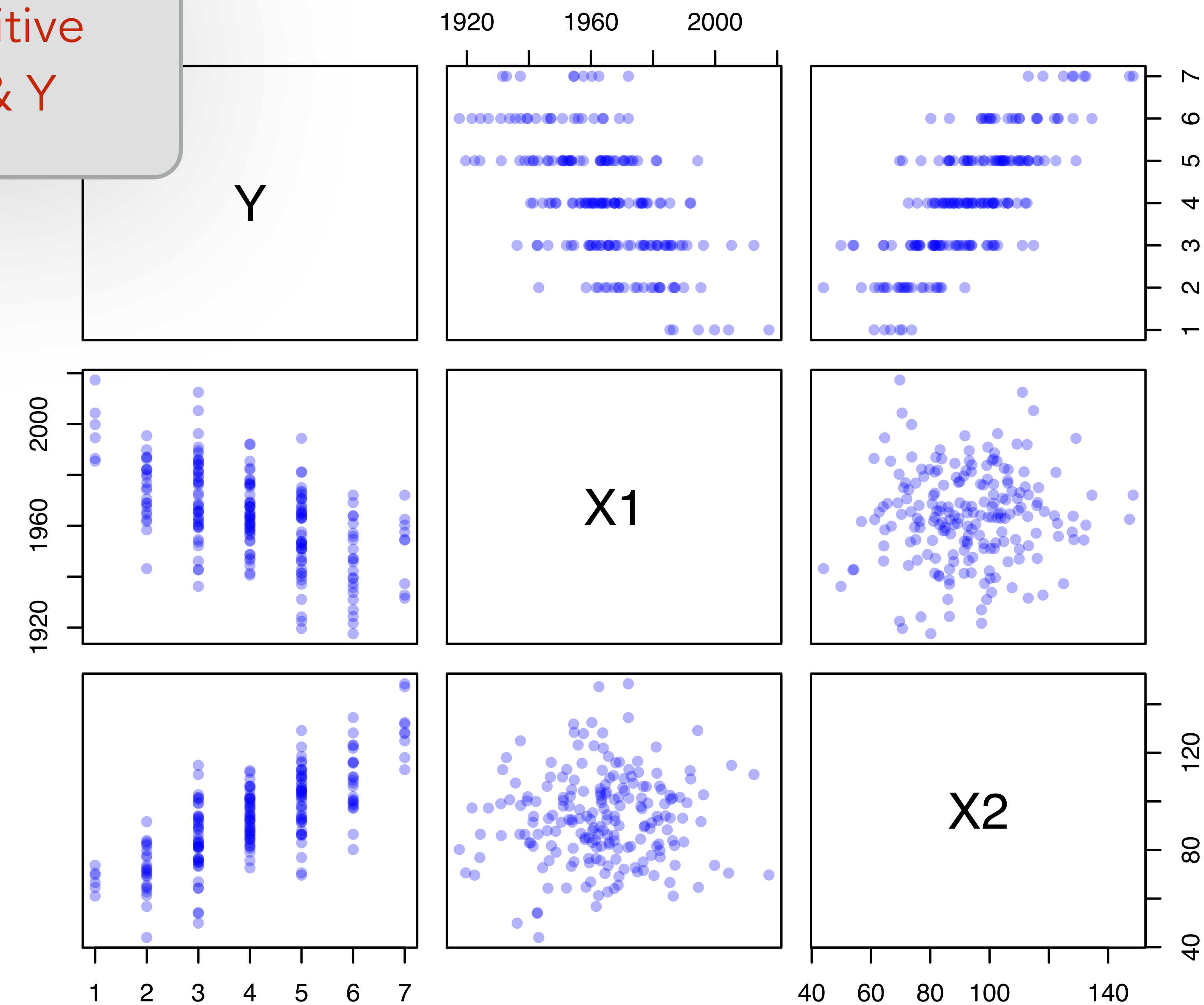
```
pairs(ord, pch = 16, col = rgb(0, 0, 1, 0.3))
```

Data Exploration



Data Exploration

Looks like a negative relationship between X1 & Y, and a positive relationship between X2 & Y



Data Exploration

- Use the `table` function to get frequencies for each ordinal response

```
yTable = table(ord$Y)

yTable

 1  2  3  4  5  6  7
 6 23 45 48 46 23  9
```

Data Exploration

- Make as a data frame and format properly

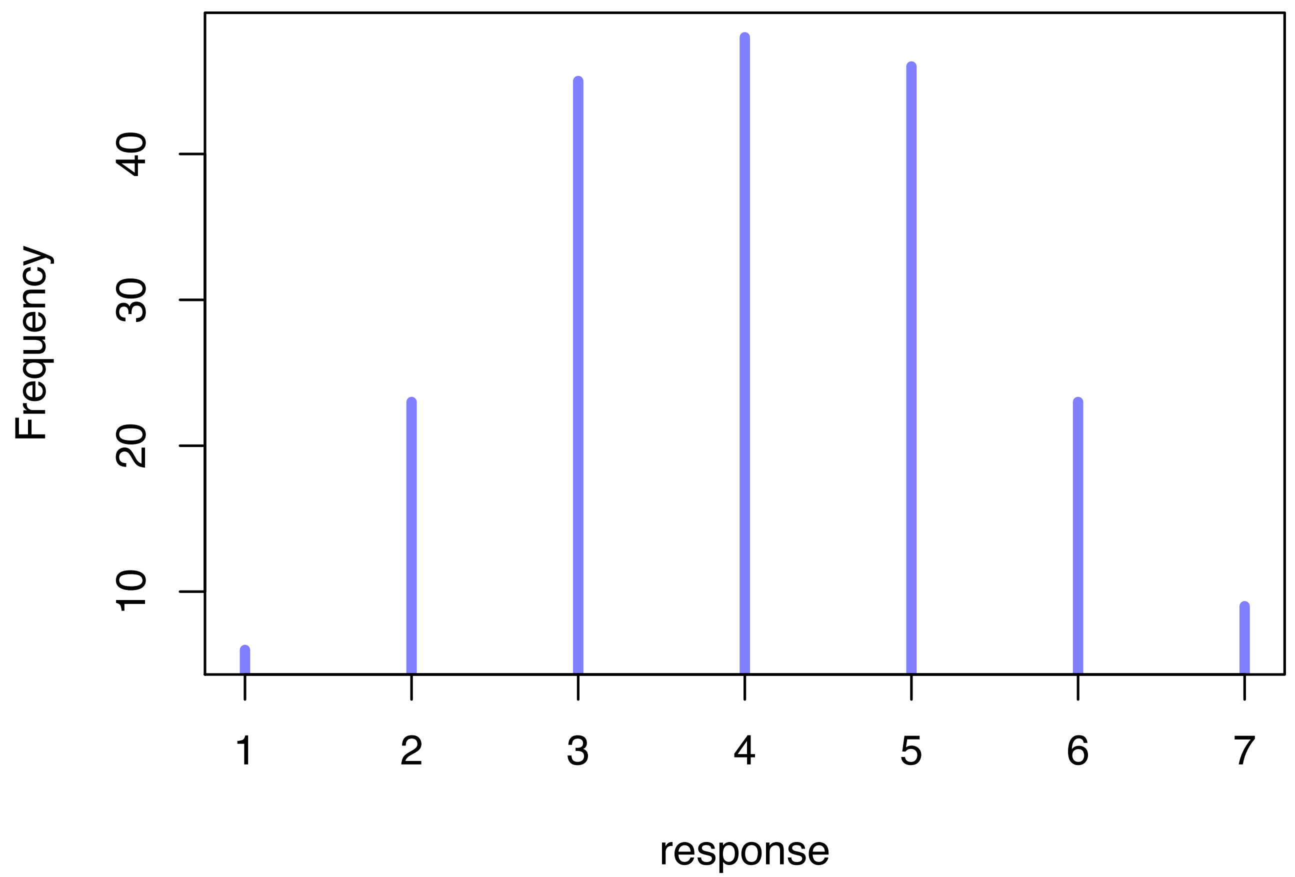
```
yTable.df = as.data.frame(yTable)

yTable.df[, 1] = as.numeric(as.character(yTable.df[, 1]))
```

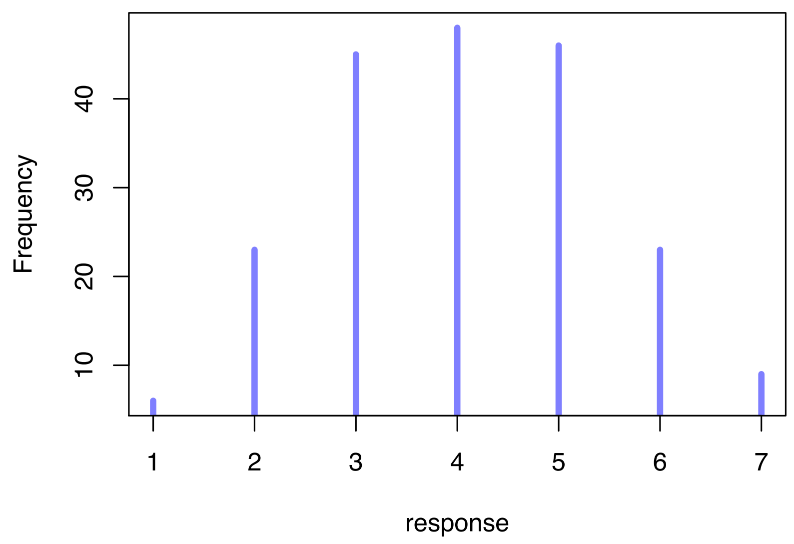
Data Exploration

- Plot the data

```
plot(yTable.df[, 1], yTable.df[, 2], type = "h", ylab = "Frequency",
     xlab = "response", lwd = 4, col = rgb(0, 0, 1, 0.5))
```



Data Exploration

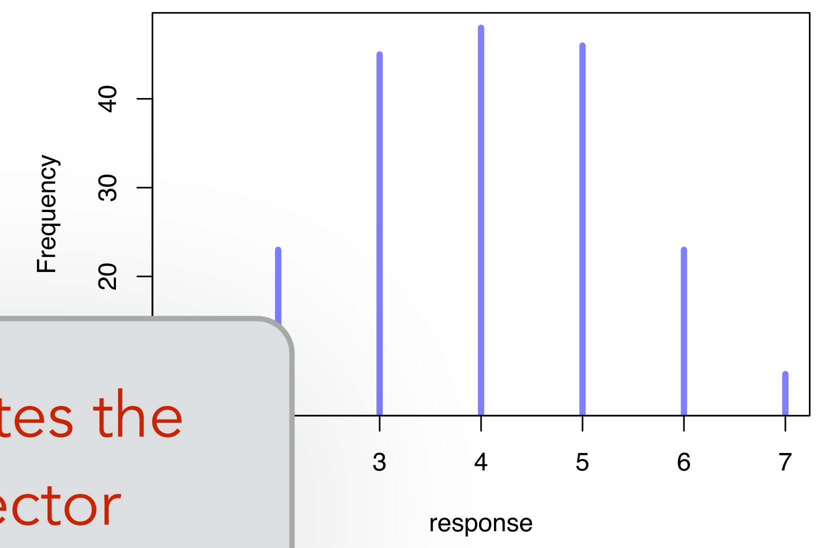


- Can also transpose this to the cumulative distribution of your data, if you want to

```
# Get proportions
pr_y = yTable / nrow(ord)

# Get cumulative proportions
cum_pr_y = cumsum(pr_y)
```

Data Exploration



- Can also transpose this to the cumulative distribution of your d

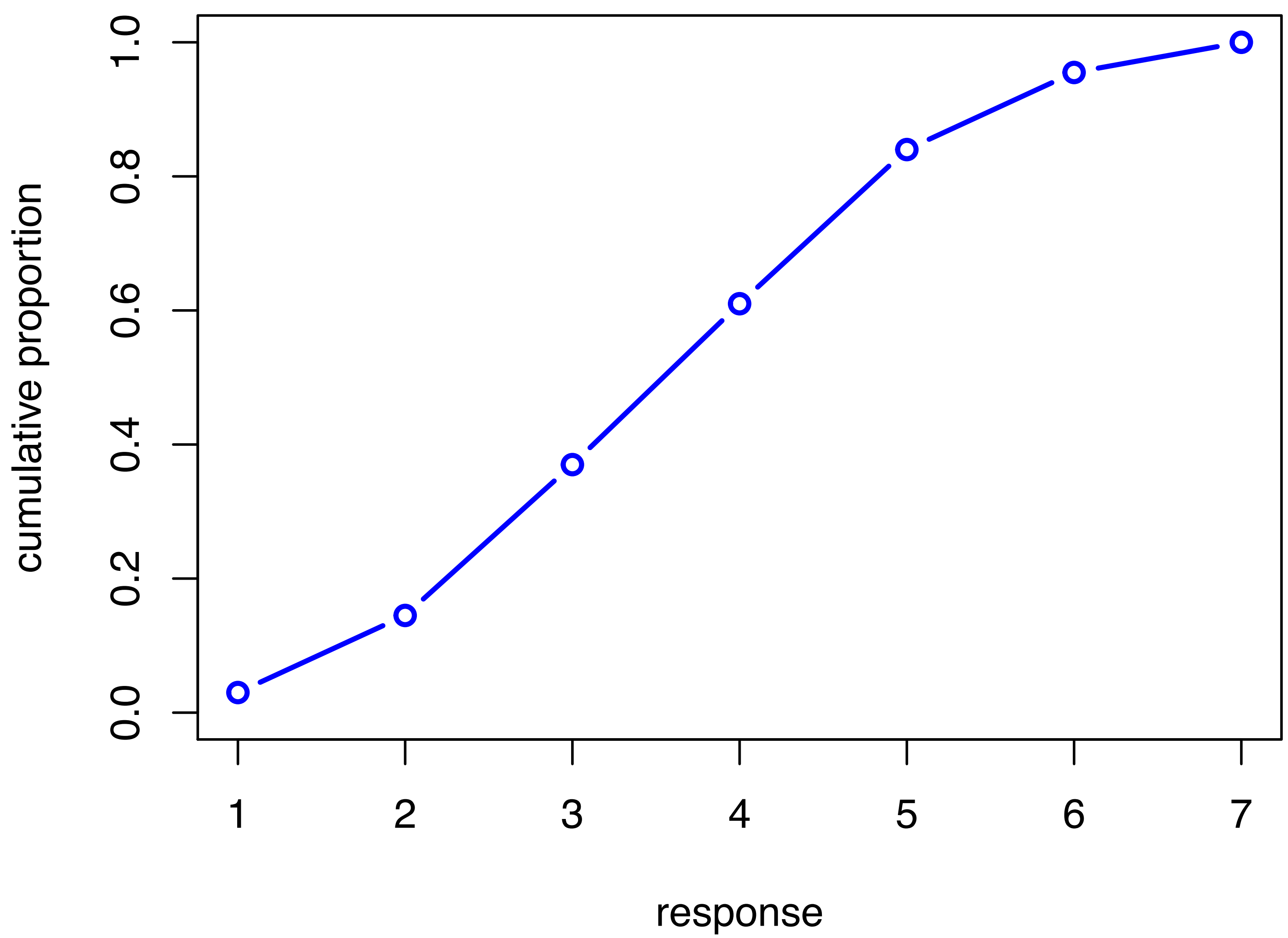
An R function that calculates the cumulative sums of a vector

```
# Get proportions
pr_y = yTable / nrow(ord)

# Get cumulative proportions
cum_pr_y = cumsum(pr_y)
```

Data Exploration

```
plot(yTable.df[, 1], cum_pr_y, type = "b", lwd = 2, ylab = “cumulative
     proportion", xlab = "response", ylim = c(0, 1), col = “blue”)
```



Bayesian Approach

Load Libraries & Functions

```
library(rstan)
library(ggplot2)
source("plotPost.R")
```

Organize the Data

```
y = ord$Y
N = length(y)
nLevels = length(unique(y))

x1 = ord$X1
x2 = ord$X2
```

Organize the Data

```
y = ord$Y
N = length(y)
nLevels = length(unique(y))

x1 = ord$X1
x2 = ord$X2
```

Making a variable with the number of response levels will make your code more generic

Standardize the Metric Variables

```
x1Mean = mean(x1)
x1SD = sd(x1)
zx1 = (x1 - x1Mean) / x1SD

x2Mean = mean(x2)
x2SD = sd(x2)
zx2 = (x2 - x2Mean) / x2SD
```

Make Data List For Stan

```
dataList = list(
  N = N,
  K = nLevels,
  y = y,
  x1 = zx1,
  x2 = zx2
)
```

Define the Model

- The **data** block

```
modelstring = "
  data {
    int<lower=0> N;                  // Sample size
    int<lower=2> K;                  // Number of ordered outcomes
    int<lower=1, upper=K> y[N];      // Ordered predicted variable
    vector[N] x1;                    // First predictor variable
    vector[N] x2;                    // Second predictor variable
  }
```

Define the Model

- The **parameters** block

```
parameters {
    real b0;
    real b1;                  // Effect of first predictor variable
    real b2;                  // Effect of second predictor variable
    ordered[K-1] c;           // Vector of cutpoints
  }
```

Define the Model

- The **parameters** block

```
parameters {
    real b0;
    real b1;                    // Effect of first predictor variable
    real b2;                    // Effect of second predictor variable
    ordered[K-1] c;             // Vector of cutpoints
  }
```

Stan is going to estimate our cut points for us (with JAGS you had to do this yourself!!!)

Define the Model

- The **model** block

```
model {
    // Definitions
    vector[N] mu;

    // Likelihood
    for (i in 1:N) {
      mu[i] = b0 + (b1 * x1[i]) + (b2 * x2[i]);
      y[i] ~ ordered_logistic(mu[i], c);
    }

    // Priors
    b0 ~ normal(0, 1);
    b1 ~ normal(0, 1);
    b2 ~ normal(0, 1);
  }
```

Define the Model

- The **model** block

The “black box” into which we can put any equations that we have dealt with before (or more)

```
model {
    // Definitions
    vector[N] mu;

    // Likelihood
    for (i in 1:N) {
      mu[i] = b0 + (b1 * x1[i]) + (b2 * x2[i]);
      y[i] ~ ordered_logistic(mu[i], c);
    }

    // Priors
    b0 ~ normal(0, 1);
    b1 ~ normal(0, 1);
    b2 ~ normal(0, 1);
  }
```

Define the Model

- The **model** block

An ordered logistic likelihood with mean; cut points estimates from the data.

```
model {
    // Definitions
    vector[N] mu;

    // Likelihood
    for (i in 1:N) {
      mu[i] = b0 + (b1 * x1[i]) + (b2 * x2[i]);
      y[i] ~ ordered_logistic(mu[i], c);
    }

    // Priors
    b0 ~ normal(0, 1);
    b1 ~ normal(0, 1);
    b2 ~ normal(0, 1);
  }
```

Define the Model

- The **generated quantities** block

```
generated quantities {
    // Definitions
    vector[N] mu_pred;
    vector[N] y_pred;

    for (i in 1:N) {
      mu_pred[i] = b0 + (b1 * x1[i]) + (b2 * x2[i]);
      y_pred[i] = ordered_logistic_rng(mu_pred[i], c);
    }
  }
"
writeLines(modelstring, con = "model.stan")
```

Run the Model

```
stanFit = stan(file = "model.stan",
               data = dataList,
               pars = c("b0", "b1", "b2", "y_pred"),
               warmup = 2000,
               iter = 7000,
               chains = 3)
```

Check MCMC Performance

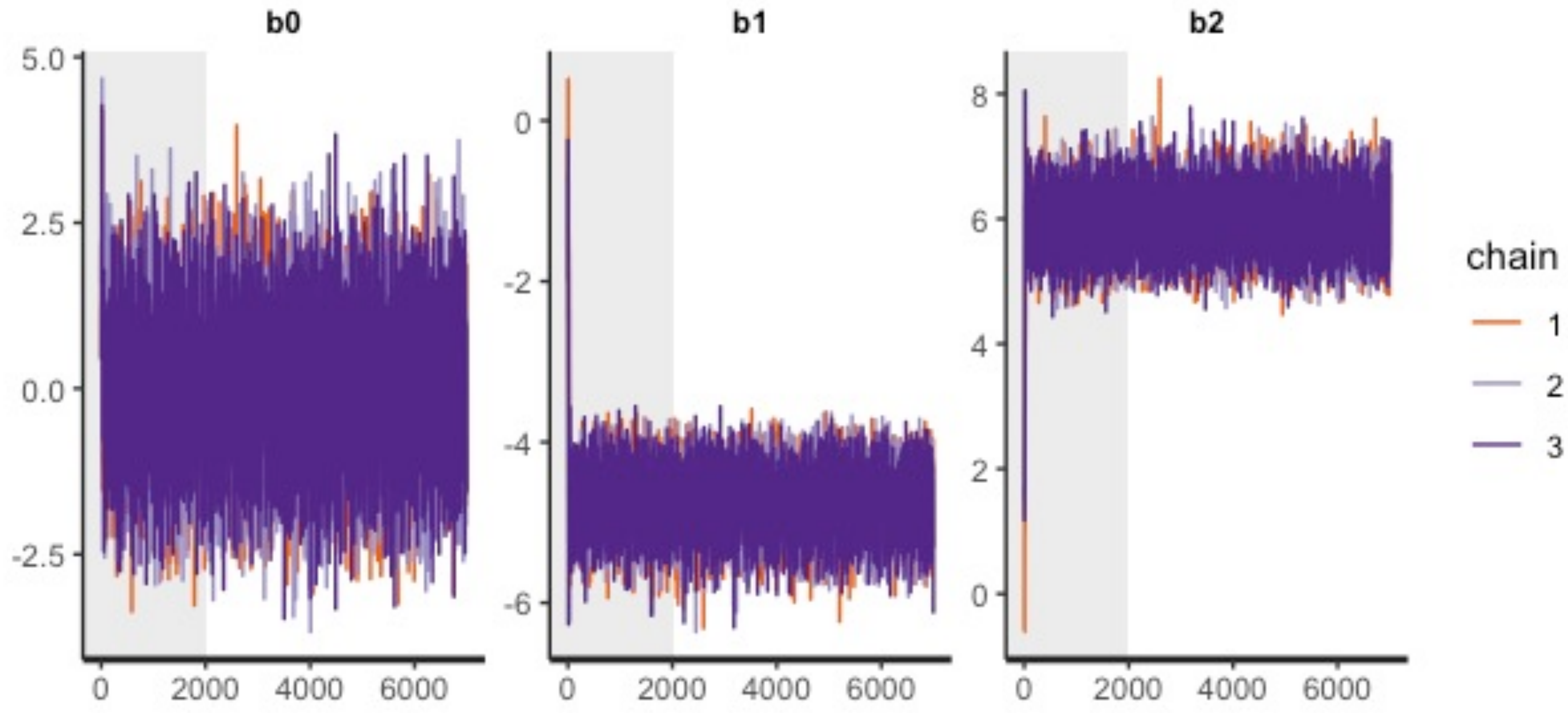
```
print(stanFit)

Inference for Stan model: model.
3 chains, each with iter=7000; warmup=2000; thin=1;
post-warmup draws per chain=5000, total post-warmup draws=15000.

              mean se_mean   sd  2.5%   25%   50%   75%  97.5% n_eff Rhat
b0           -0.01    0.01 1.01 -1.99 -0.69 -0.02  0.68   1.98 11679    1
b1           -4.75    0.00 0.37 -5.49 -4.99 -4.74 -4.49  -4.05  5648    1
b2            5.97    0.01 0.45  5.13  5.66  5.96  6.27   6.88  5693    1
y_pred[1]     3.74    0.00 0.48  3.00  3.00  4.00  4.00   4.00 15324    1
y_pred[2]     5.00    0.00 0.38  4.00  5.00  5.00  5.00   6.00 14732    1
...
```

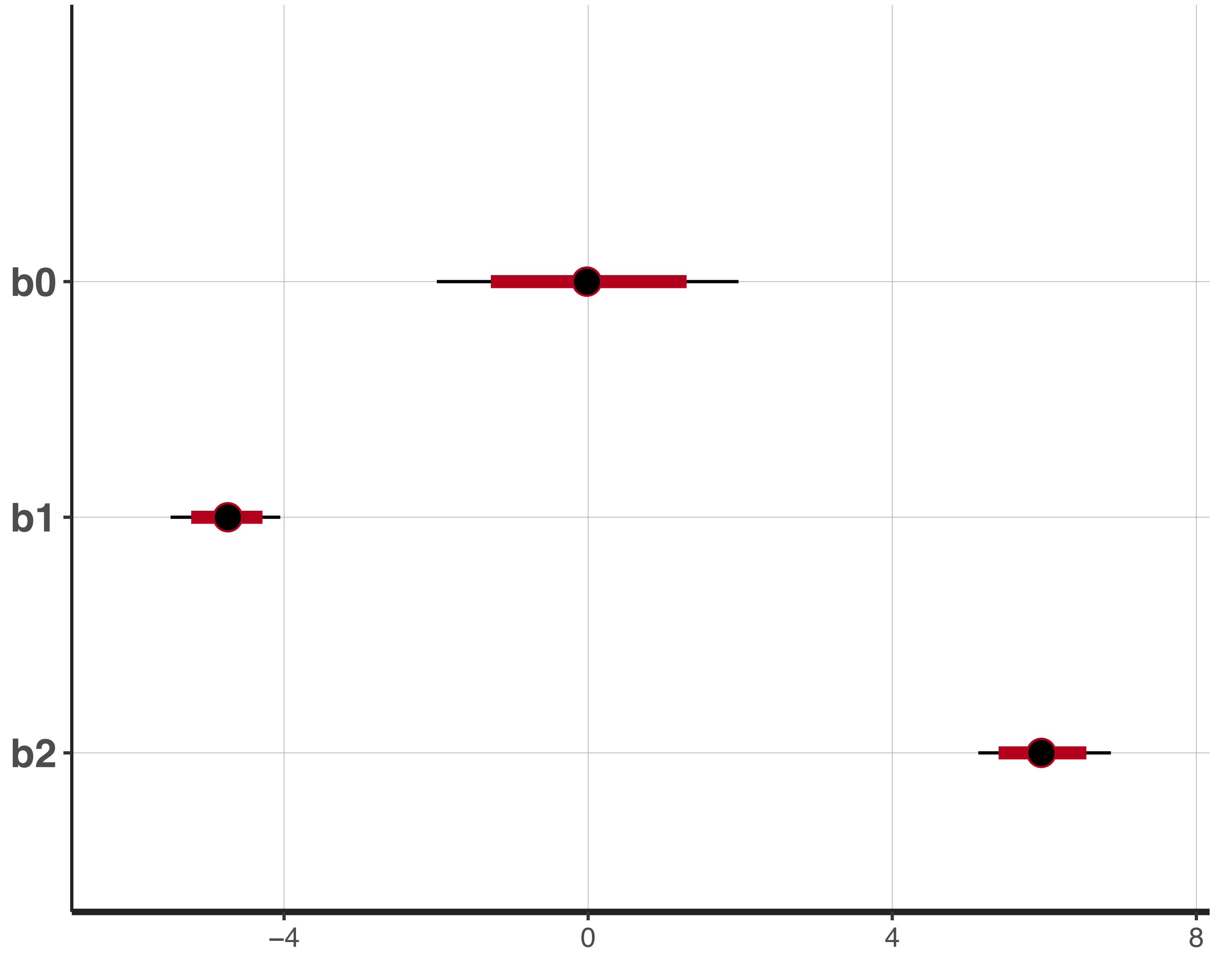
Check MCMC Performance

```
stan_trace(stanFit, pars = c("b0", "b1", "b2"), inc_warmup = TRUE)
```



Evaluate Results

```
stan_plot(stanFit, par = c("b0", "b1", "b2"))
```



Evaluate Results

- Parse out the data

```
mcmcChain = as.data.frame(stanFit)

zb0 = mcmcChain[, "b0"]
zb1 = mcmcChain[, "b1"]
zb2 = mcmcChain[, "b2"]
```

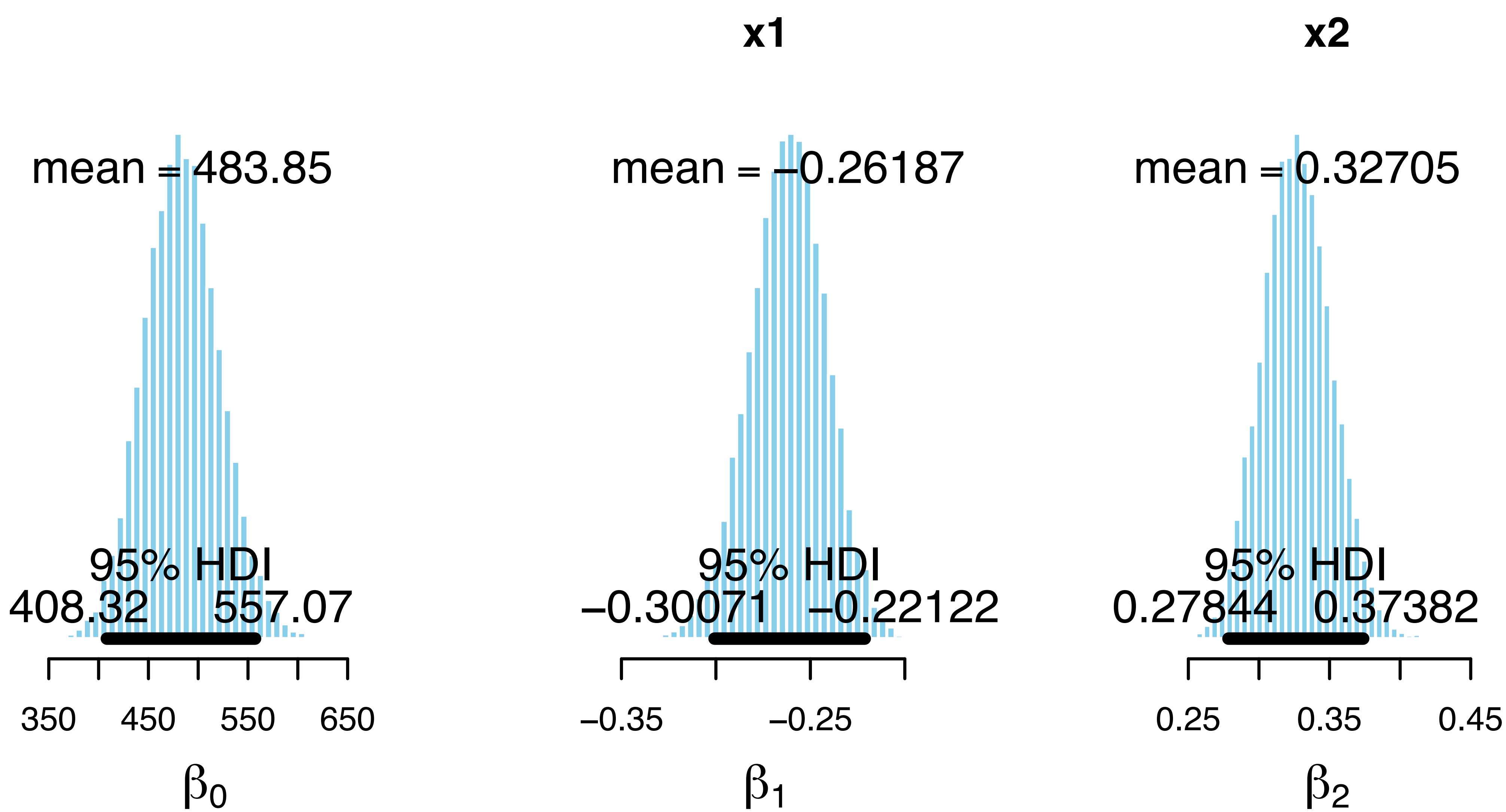
Evaluate Results

- Convert to original scale

```
b1 = zb1 / x1SD
b2 = zb2 / x2SD
b0 = zb0 - (((zb1 * x1Mean) / x1SD) + ((zb2 * x2Mean) / x2SD))
```

Evaluate Results

```
par(mfrow = c(1, 3))
histInfo = plotPost(b0, xlab = bquote(beta[0]))
histInfo = plotPost(b1, xlab = bquote(beta[1]), main = "x1")
histInfo = plotPost(b2, xlab = bquote(beta[2]), main = “x2")
```



Posterior
Predictive Check

Posterior Predictive Check

- Get the predicted values

```
chainLength = length(mcmcChain[, 1])

zypred = matrix(0, ncol = N, nrow = chainLength)

for (i in 1:N) {
  zypred[, i] = mcmcChain[, paste("y_pred[", i, "]", sep = "")]
}
```

Posterior Predictive Check

- Create a table of counts of each category in each step

```
yPredCounts = matrix(0, ncol = nLevels, nrow = chainLength)

for (i in 1:chainLength) {
  yPredCounts[i, ] = as.integer(table(zypred[i, ]))
}
```

Posterior Predictive Check

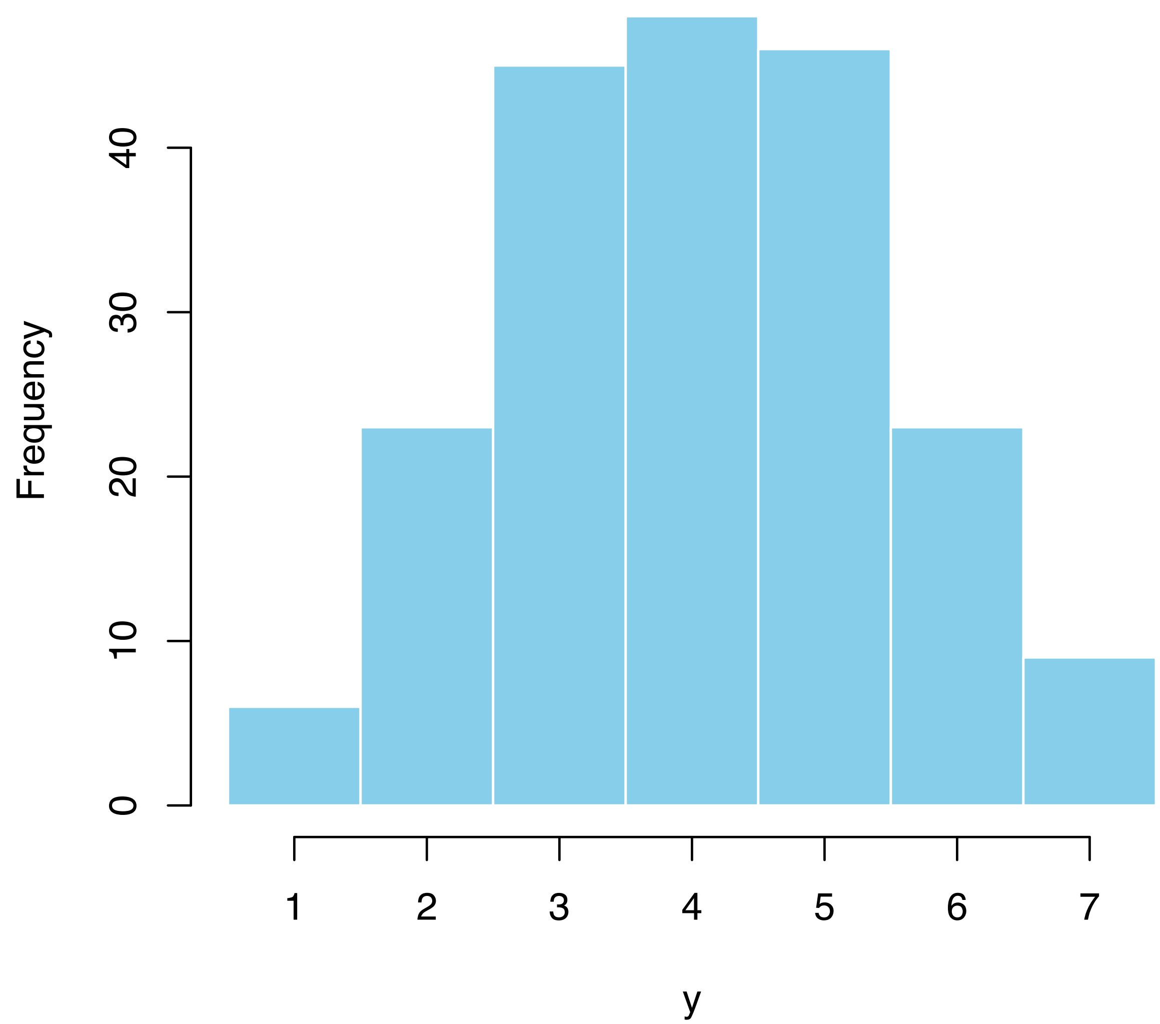
```
#--- Calculate the mean counts ---#
zypredMean = apply(yPredCounts, 2, mean)

#--- Upper and lower expected 95% HDI for each visit ---#
zypredLow  = apply(yPredCounts, 2, quantile, probs = 0.025)
zypredHigh = apply(yPredCounts, 2, quantile, probs = 0.975)
```

Posterior Predictive Check

- Plot observed data

```
par(mfrow = c(1, 1))
hist(y, breaks = c(0.5, (1:nLevels + 0.5)), main = "", col = "skyblue",
     border = "white")
```

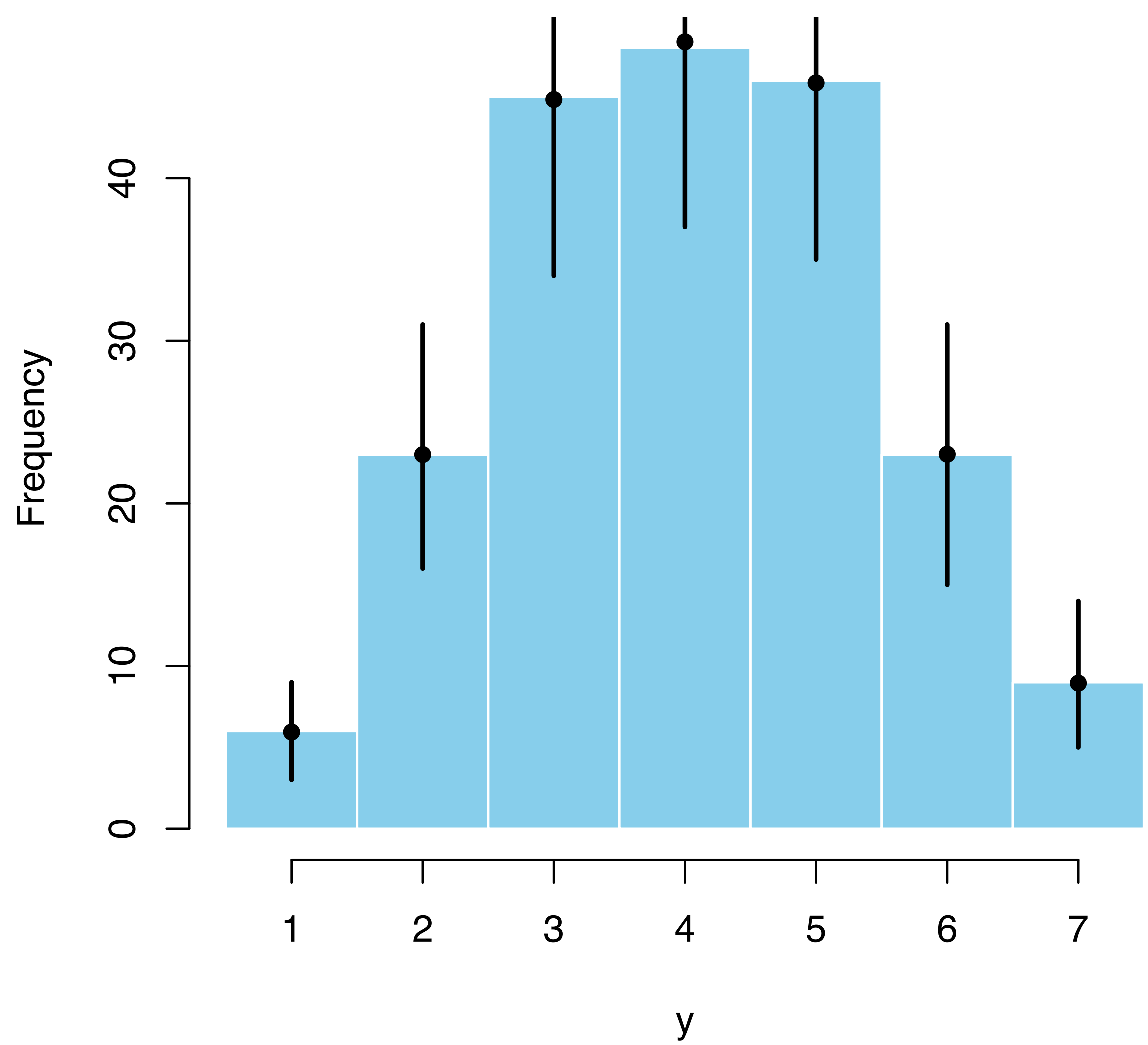


Posterior Predictive Check

- Add predicted values and HDIs

```
points(x = 1:nLevels, y = zypredMean, pch = 16)

segments(x0 = 1:nLevels, y0 = zypredLow, x1 = 1:nLevels, y1 = zypredHigh, lwd = 2)
```



Questions?