Advanced models for count data

Frank Edwards

Count data

- Counts are cumulative totals of the number of incidences of some event, generally across time or place

Count data

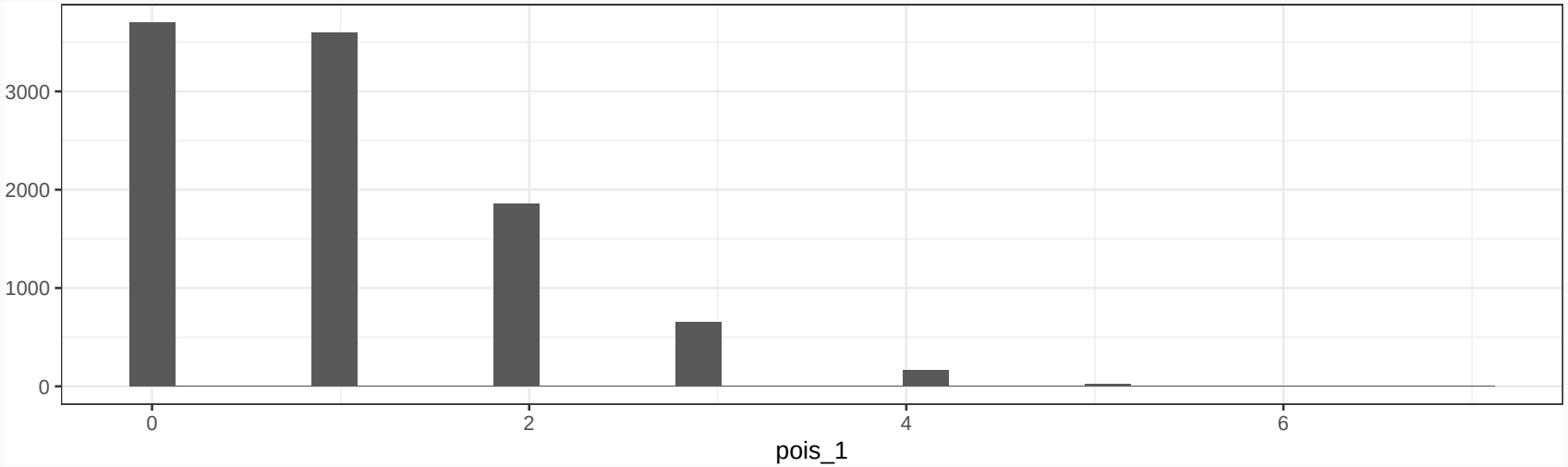
- Counts are cumulative totals of the number of incidences of some event, generally across time or place
- Counts are positive integers $\in [0, \infty]$

Count data

- Counts are cumulative totals of the number of incidences of some event, generally across time or place
- Counts are positive integers $\in [0, \infty]$
- We can model count variables using the Poisson distribution

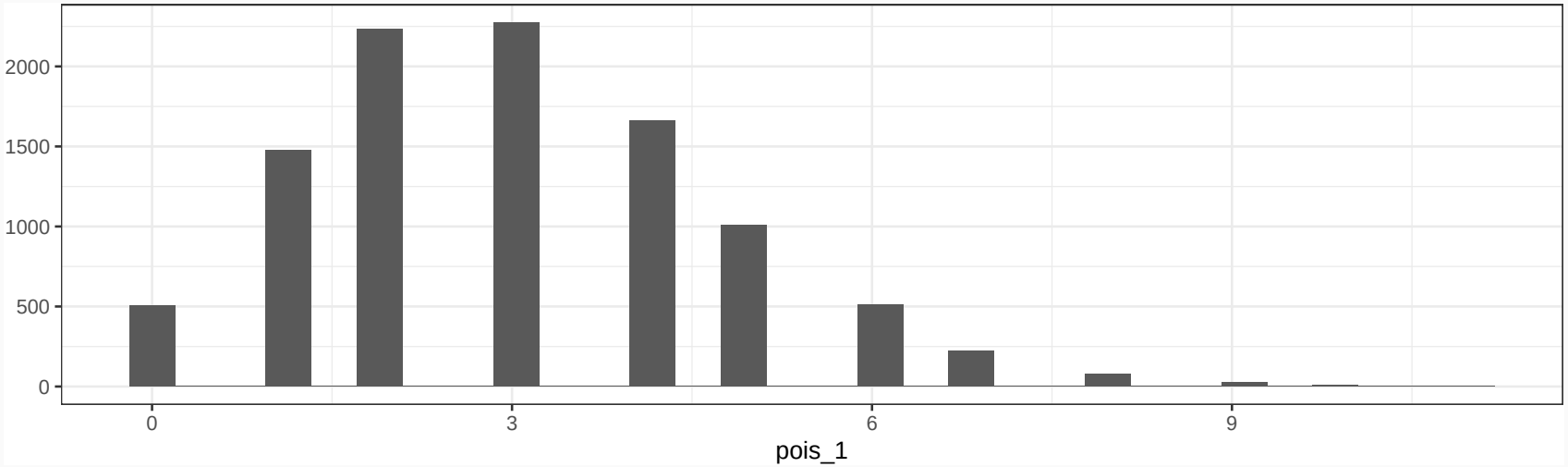
The Poisson distribution (lambda = 1)

```
## pois_1
##    0    1    2    3    4    5    6    7
## 3700 3599 1862  653  161   22    2    1
```



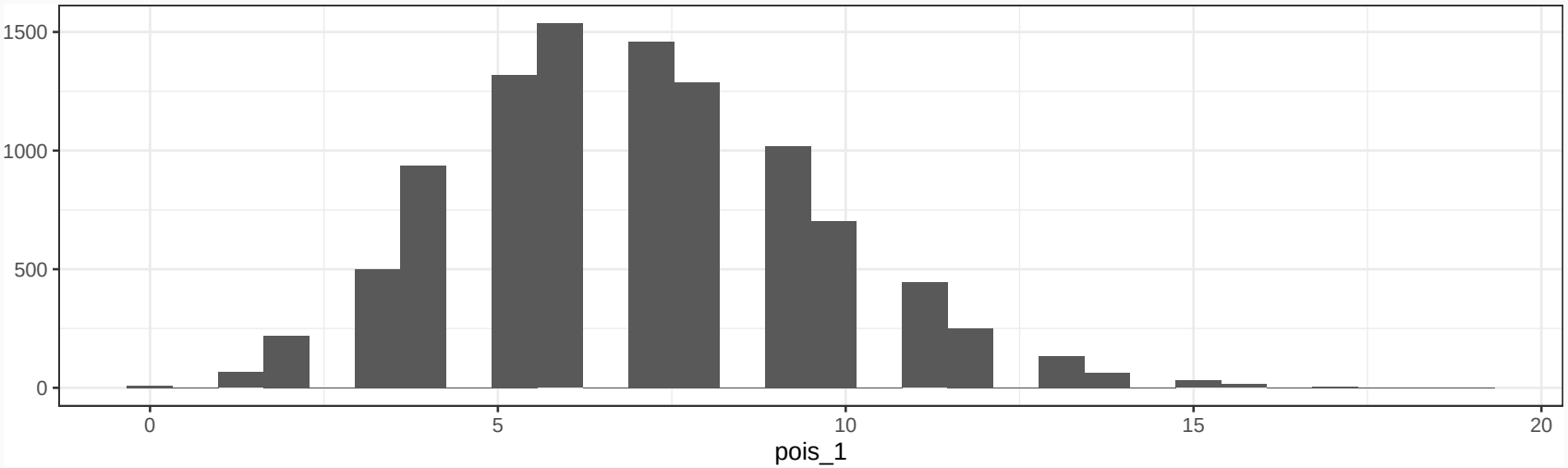
The Poisson distribution (lambda = 3)

```
## pois_1
##    0    1    2    3    4    5    6    7    8    9   10   11
##  505 1476 2231 2273 1662 1009  509  220   80   26    7    2
```

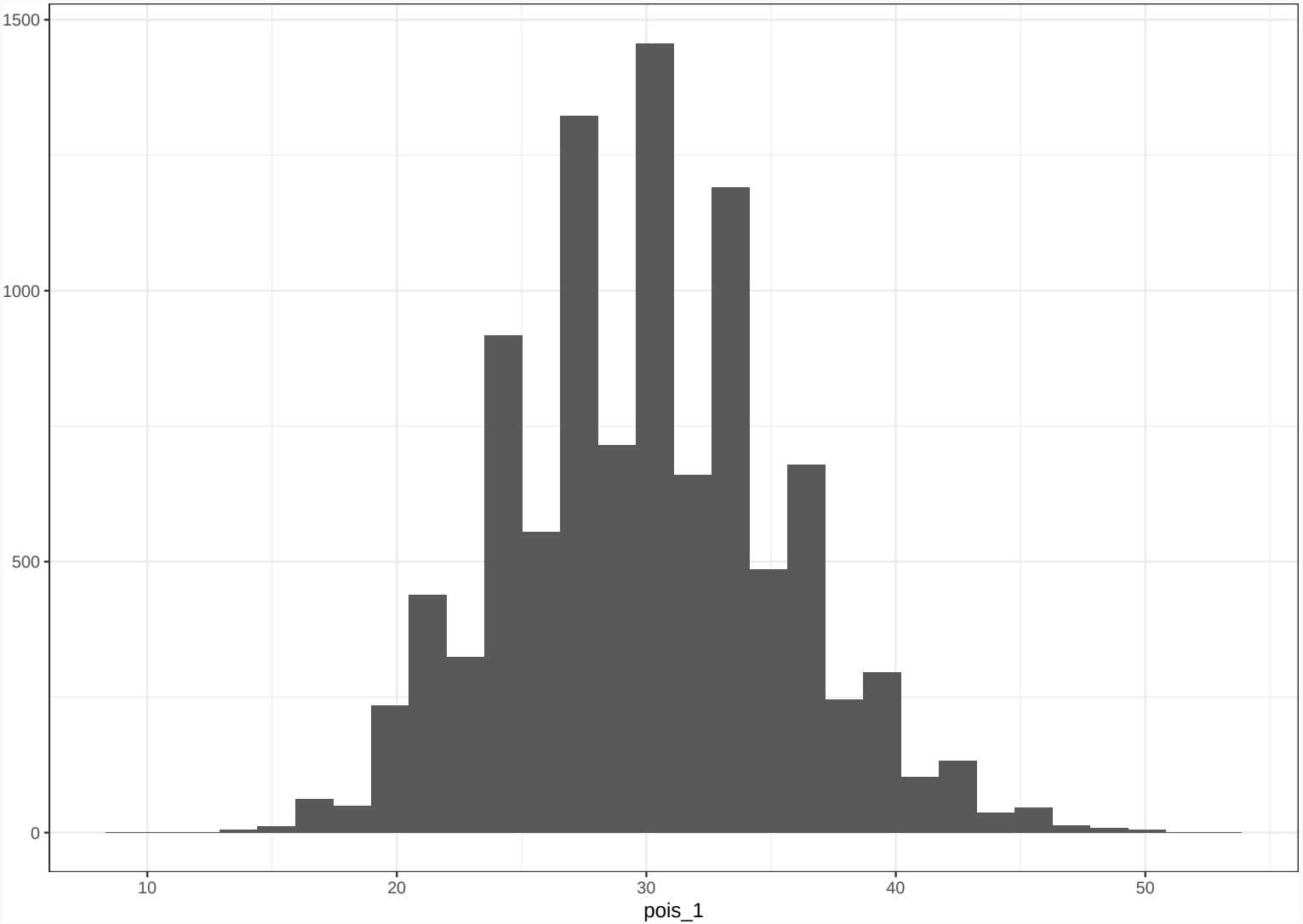


The Poisson distribution (lambda = 7)

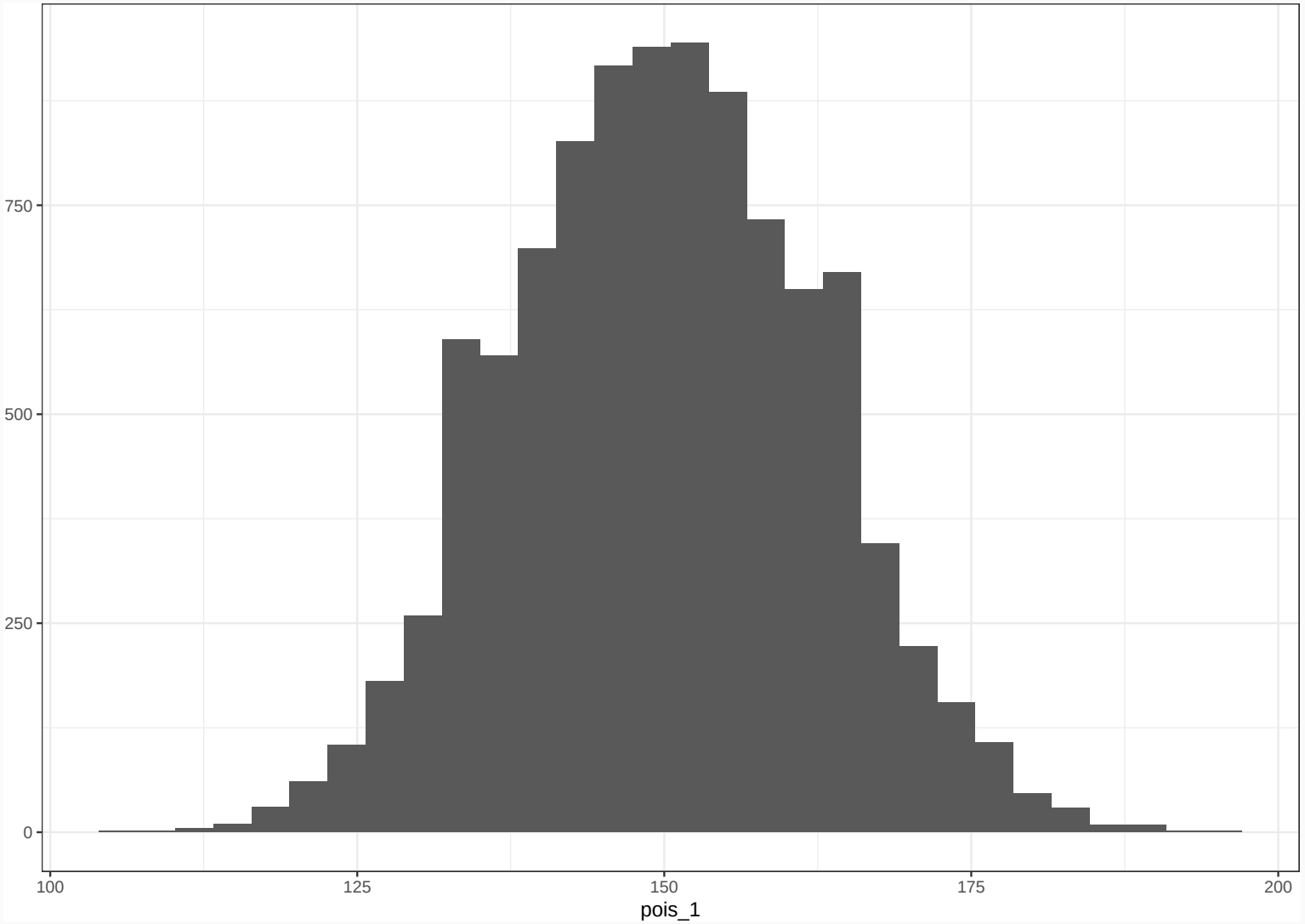
```
## pois_1
##    0    1    2    3    4    5    6    7    8    9   10   11   12   13   14   15
##    9   66  219  500  938 1319 1536 1459 1288 1018  703  444  250  133   63   31
##   16   17   18   19
##   15    6    2    1
```



The Poisson distribution (lambda = 30)



The Poisson distribution (lambda = 150)



Special properties of the Poisson

- The variance and mean of a Poisson variable with parameter λ are both equal to λ

Special properties of the Poisson

- The variance and mean of a Poisson variable with parameter λ are both equal to λ

```
### draw a sample of 10,000 from a Poisson with lambda = 2.3
pois_demo <- rpois(10000, lambda = 2.3)
table(pois_demo)
```

```
## pois_demo
##    0    1    2    3    4    5    6    7    8    9
## 1068 2309 2615 1999 1160  533  219   70   25    2
```

Special properties of the Poisson

- The variance and mean of a Poisson variable with parameter λ are both equal to λ

```
### draw a sample of 10,000 from a Poisson with lambda = 2.3
pois_demo <- rpois(10000, lambda = 2.3)
table(pois_demo)
```

```
## pois_demo
##    0    1    2    3    4    5    6    7    8    9
## 1068 2309 2615 1999 1160  533  219   70   25    2
```

```
mean(pois_demo)
```

```
## [1] 2.2863
```

```
var(pois_demo)
```

```
## [1] 2.345167
```

How well does a Poisson distribution fit our soccer data?

- Is the mean / variance assumption of a Poisson reasonable for our NWSL data on goal scoring?

```
### Load in NWLS data
library(nwslR)
data("fieldplayer_overall_season_stats")
nwsl_stats <- fieldplayer_overall_season_stats
### Check if mean == variance
mean(nwsl_stats$gls)
```

```
## [1] 1.42963
```

```
var(nwsl_stats$gls)
```

```
## [1] 6.091041
```

Comparing distributions

```
### Draws from a Poisson with nrow() observations and lambda = mean(nwls$goals)
sim1 <- rpois(nrow(nwsl_stats), lambda = mean(nwsl_stats$gls))
### simulation
table(sim1)
```

```
## sim1
##   0   1   2   3   4   5   6
## 321 461 337 144  67  15   5
```

```
### observed
table(nwsl_stats$gls)
```

```
##
##   0   1   2   3   4   5   6   7   8   9  10  11  12  13  14  15  16  17  18
## 738 237 106  71  61  40  26  16  19  13   5   6   3   3   1   1   2   1   1
```

What’s going on here?

- Problem 1: Player position is associated with goal scoring, right? Defenders don’t score many (or any) goals.

What’s going on here?

- Problem 1: Player position is associated with goal scoring, right? Defenders don’t score many (or any) goals.

What's going on here?

- Problem 1: Player position is associated with goal scoring, right?

  Defenders don't score many (or any) goals.

```
nwsl_stats %>%
    group_by(pos) %>%
    summarize(gls_mn = mean(gls))
```

```
## # A tibble: 6 x 2
##   pos   gls_mn
##   <chr>  <dbl>
## 1 DF     0.334
## 2 DF,FW  0.925
## 3 DF,MF  0.837
## 4 FW     2.45
## 5 FW,MF  3.17
## 6 MF     1.11
```

So would simulating by position help yield a better fit?

```
### compute mean, variance, and N for each position
positions <- nwsl_stats %>%
    group_by(pos) %>%
    summarize(obs_mn = mean(gls), obs_var = var(gls), n_obs = n())

positions
```

```
## # A tibble: 6 x 4
##   pos   obs_mn obs_var n_obs
##   <chr>  <dbl>   <dbl> <int>
## 1 DF     0.334   0.558   353
## 2 DF,FW  0.925   2.38     40
## 3 DF,MF  0.837   3.40    129
## 4 FW     2.45    8.63    334
## 5 FW,MF  3.17   14.8     146
## 6 MF     1.11    3.37    348
```

So would simulating by position help yield a better fit?

Does $E(goals|position) = var(goals|position)$?

```
### now simulate 10000 player - season totals by position
positions <- positions %>%
    group_by(pos) %>%
    mutate(sim_mn = mean(rpois(n_obs, obs_mn)), sim_var = var(rpois(n_obs, obs_mn)))

positions
```

```
## # A tibble: 6 x 6
## # Groups:   pos [6]
##   pos   obs_mn obs_var n_obs sim_mn sim_var
##   <chr>  <dbl>   <dbl> <int>  <dbl>   <dbl>
## 1 DF     0.334   0.558   353  0.323   0.282
## 2 DF,FW  0.925   2.38     40  0.925   0.640
## 3 DF,MF  0.837   3.40    129  0.822   0.785
## 4 FW     2.45    8.63    334  2.18    2.48
## 5 FW,MF  3.17   14.8     146  3.27    3.69
## 6 MF     1.11    3.37    348  1.06    1.28
```

Overdispersion

Overdispersion is the presence of greater variability than we would expect from a given statistical model

Overdispersion

Overdispersion is the presence of greater variability than we would expect from a given statistical model

For a Poisson model, we assume that

$$var(x) = \bar{x} = \lambda$$

Overdispersion

Overdispersion is the presence of greater variability than we would expect from a given statistical model

For a Poisson model, we assume that

$$var(x) = \bar{x} = \lambda$$

If $var(x) > \bar{x}$, then the data are *overdispersed* relative to predictions from the Poisson model.

Clustering and overdispersion

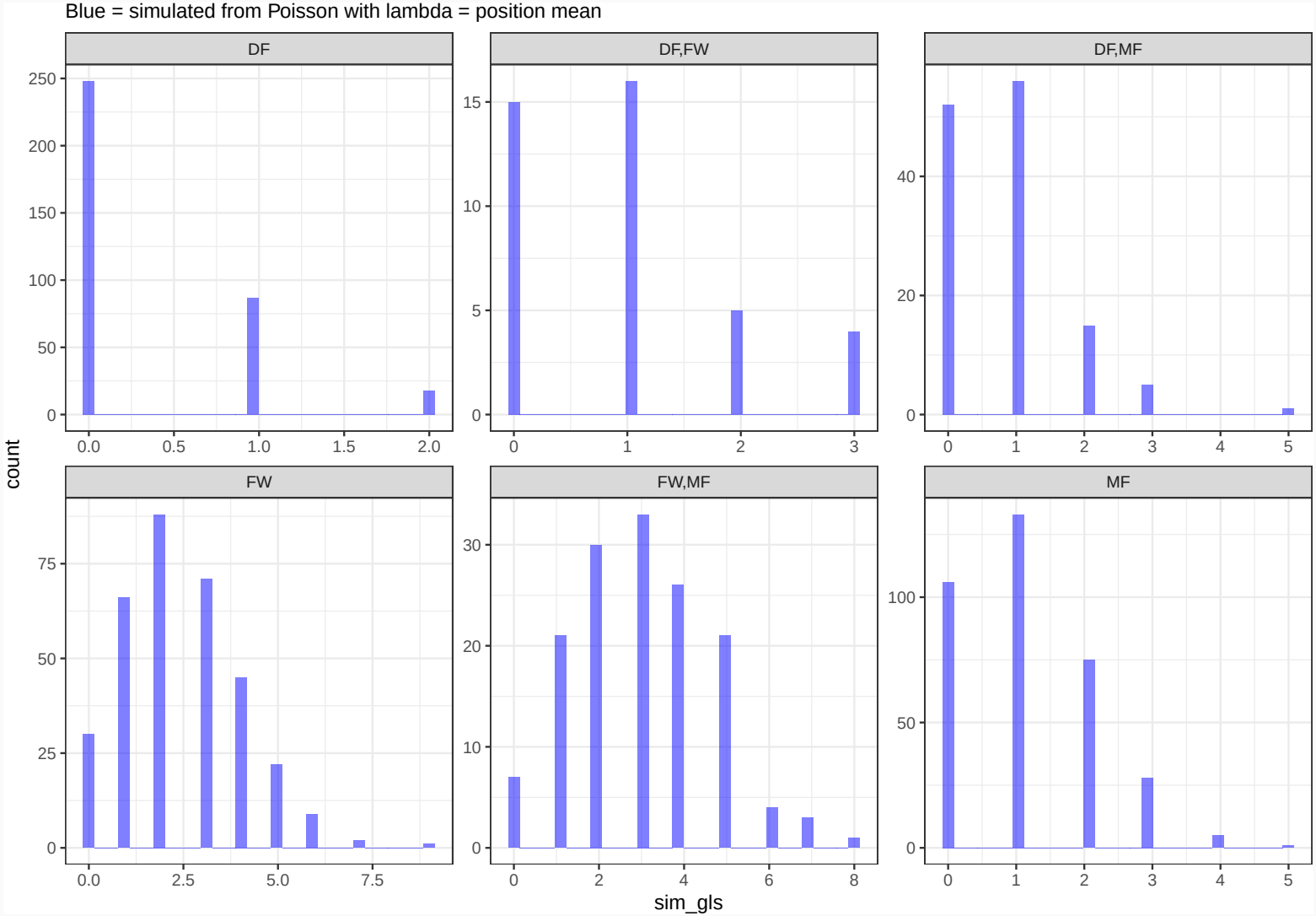
When data come from distinct sub-populations, or clusters, they can have different underlying *data generating processes* (the real world process that produces our observations that we try to approximate using a statistical model).

Clustering and overdispersion

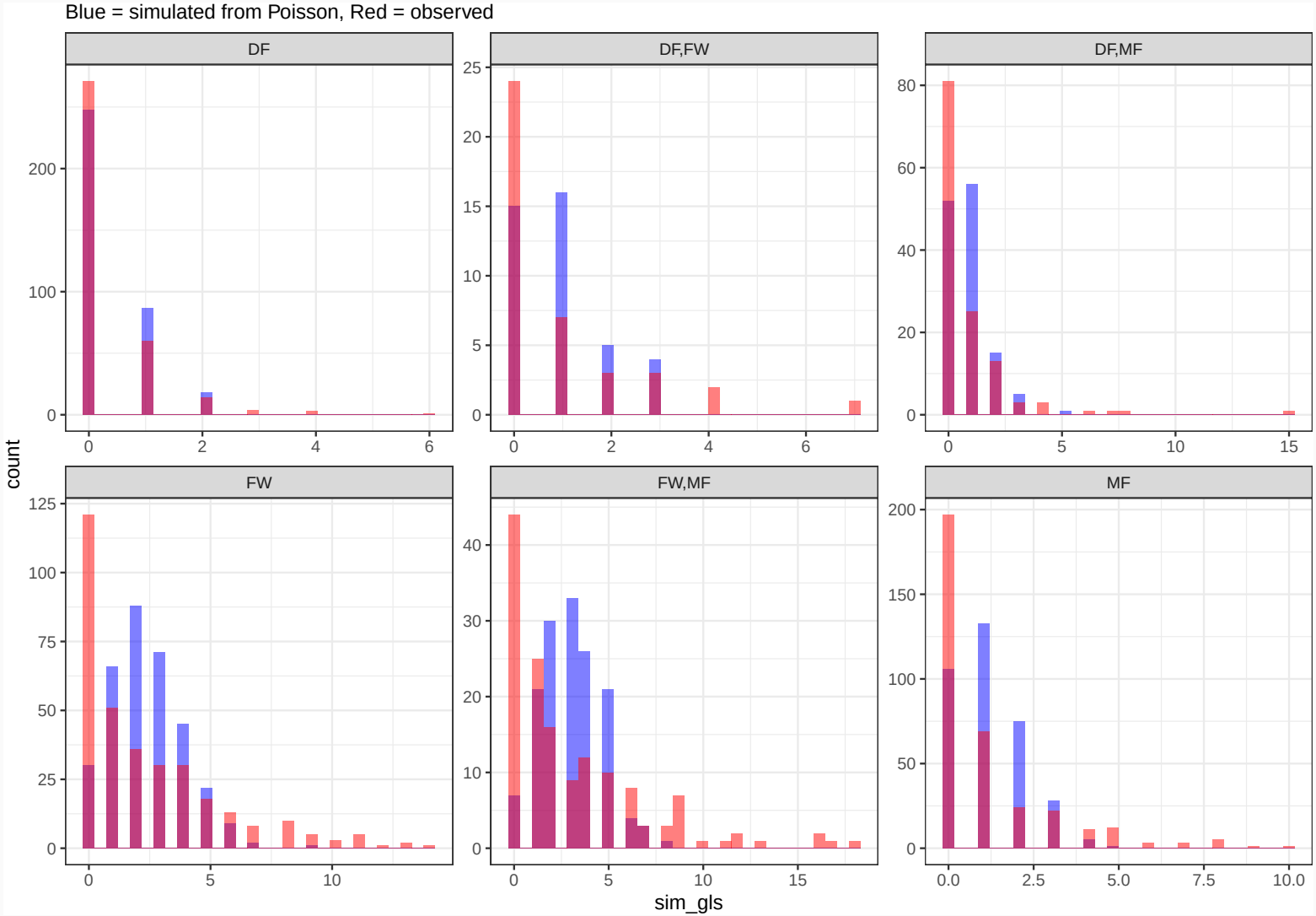
When data come from distinct sub-populations, or clusters, they can have different underlying *data generating processes* (the real world process that produces our observations that we try to approximate using a statistical model).

These differences in data generating processes often result in a) different expected values across sub-groups, and b) different levels of variability across sub-groups

Poisson expectation



Overdispersion



Overdispersion!

```
nwsl_sim %>%
    group_by(pos) %>%
    summarize(obs_var = var(gls), sim_var = var(sim_gls))
```

```
## # A tibble: 6 x 3
##   pos   obs_var sim_var
##   <chr>   <dbl>   <dbl>
## 1 DF      0.558   0.330
## 2 DF,FW   2.38    0.921
## 3 DF,MF   3.40    0.773
## 4 FW      8.63    2.42
## 5 FW,MF  14.8     2.73
## 6 MF      3.37    1.00
```

Modeling overdispersion: adding a shape parameter

We can relax the $var(x) = \bar{x}$ assumption of the Poisson likelihood with a *quasi-Poisson* likelihood that has the following properties:

$$E(x) = \lambda$$

$$var(x) = \theta\lambda$$

Modeling overdispersion: adding a shape parameter

We can relax the $var(x) = \bar{x}$ assumption of the Poisson likelihood with a *quasi-Poisson* likelihood that has the following properties:

$$E(x) = \lambda$$

$$var(x) = \theta\lambda$$

We call θ a dispersion or shape parameter. Higher values of θ result in more variability, lower values of θ result in more concentration.

The Negative Binomial model

The negative binomial model is very similar to the quasi-poisson. It includes a mean parameter μ and a shape parameter θ.

We can define a negative binomial likelihood as

$$x \sim \text{Negative Binomial}(\mu, \theta)$$

The Negative Binomial model

The negative binomial model is very similar to the quasi-poisson. It includes a mean parameter μ and a shape parameter θ.

We can define a negative binomial likelihood as

$$x \sim \text{Negative Binomial}(\mu, \theta)$$

With an expected value

$$\bar{x} = \mu$$

and variance

$$var(x) = \mu + \frac{\mu^2}{\theta}$$

Let’s see if these likelihoods generate different results

```r
options(mc.cores = parallel::detectCores(logical = FALSE))
goals_poisson <- stan_glm(gls ~ pos, family = "poisson", data = nwsl_stats, refresh = 0)

goals_negbin <- stan_glm(gls ~ pos, family = "neg_binomial_2", data = nwsl_stats,
    refresh = 0)
```

What do we notice about the results?

```
goals_poisson
```

```
## stan_glm
##  family:       poisson [log]
##  formula:      gls ~ pos
##  observations: 1350
##  predictors:   6
## ------
##             Median MAD_SD
## (Intercept) -1.1    0.1
## posDF,FW     1.0    0.2
## posDF,MF     0.9    0.1
## posFW        2.0    0.1
## posFW,MF     2.2    0.1
## posMF        1.2    0.1
##
## ------
## * For help interpreting the printed output see ?print.stanreg
## * For info on the priors used see ?prior_summary.stanreg
```

What do we notice about the results?

```
goals_negbin
```

```
## stan_glm
##  family:       neg_binomial_2 [log]
##  formula:      gls ~ pos
##  observations: 1350
##  predictors:   6
## ------
##             Median MAD_SD
## (Intercept) -1.1    0.1
## posDF,FW     1.0    0.3
## posDF,MF     0.9    0.2
## posFW        2.0    0.1
## posFW,MF     2.3    0.2
## posMF        1.2    0.1
##
## Auxiliary parameter(s):
##                       Median MAD_SD
## reciprocal_dispersion 0.6    0.0
##
## ------
## * For help interpreting the printed output see ?print.stanreg
## * For info on the priors used see ?prior_summary.stanreg
```

Posterior distributions of Beta for forwards: setting up to plot

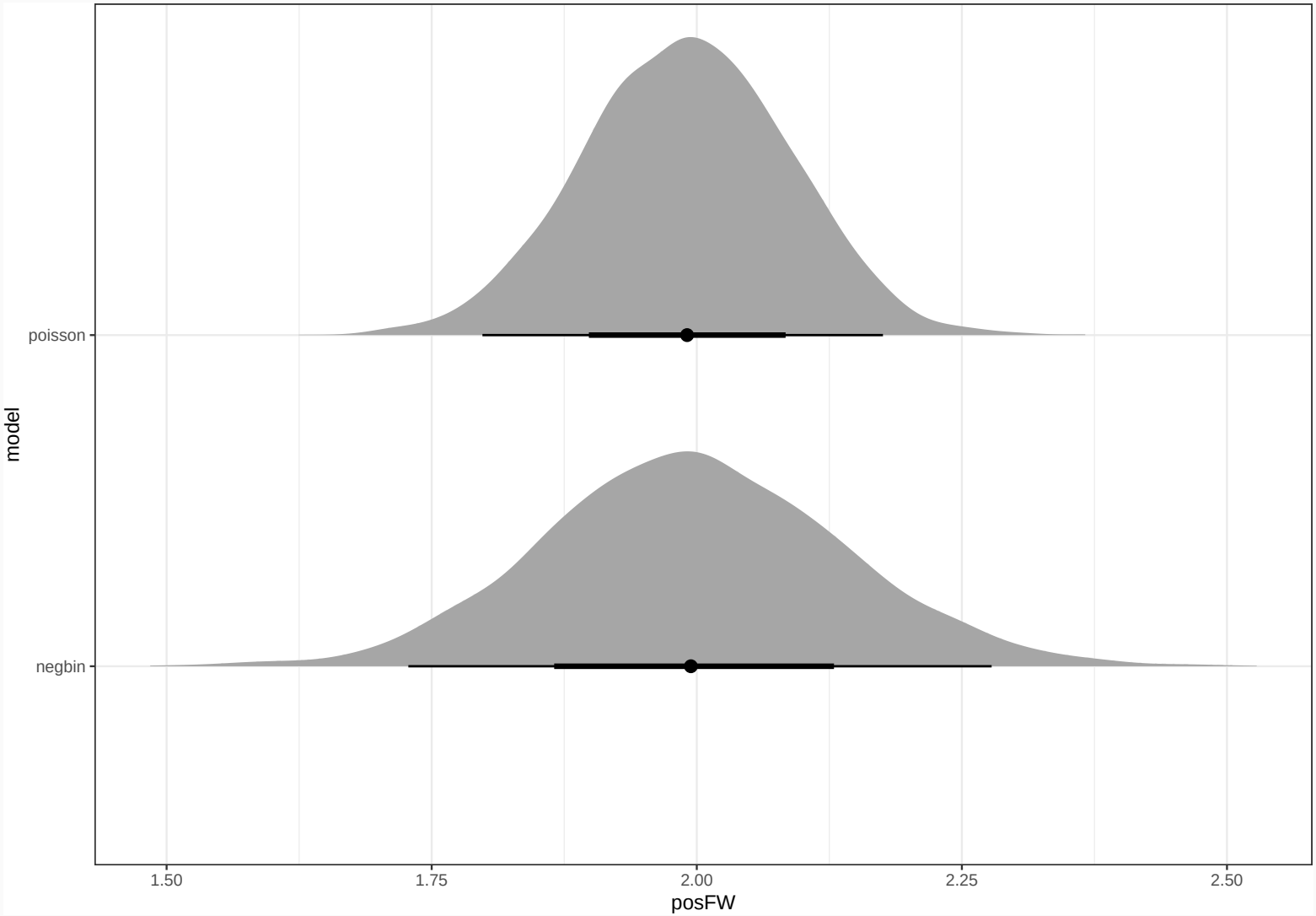
```
p_post <- data.frame(goals_poisson) %>%
    mutate(model = "poisson")
n_post <- data.frame(goals_negbin) %>%
    mutate(model = "negbin")

plot_dat <- bind_rows(p_post, n_post)
head(plot_dat)
```

```
##   X.Intercept.  posDF.FW  posDF.MF    posFW posFW.MF    posMF   model
## 1    -1.142532 0.8653415 1.0272324 2.081668 2.293969 1.239101 poisson
## 2    -1.195308 1.0110497 0.9218240 2.057276 2.389170 1.362630 poisson
## 3    -1.265993 0.9828361 1.1992549 2.159849 2.477450 1.280035 poisson
## 4    -1.279030 0.9434385 1.2634680 2.113728 2.500908 1.315258 poisson
## 5    -1.238423 0.9822504 0.8831236 2.192809 2.409898 1.357905 poisson
## 6    -1.185544 1.0623632 1.1963970 2.098625 2.305386 1.266539 poisson
##   reciprocal_dispersion
## 1                    NA
## 2                    NA
## 3                    NA
## 4                    NA
## 5                    NA
## 6                    NA
```

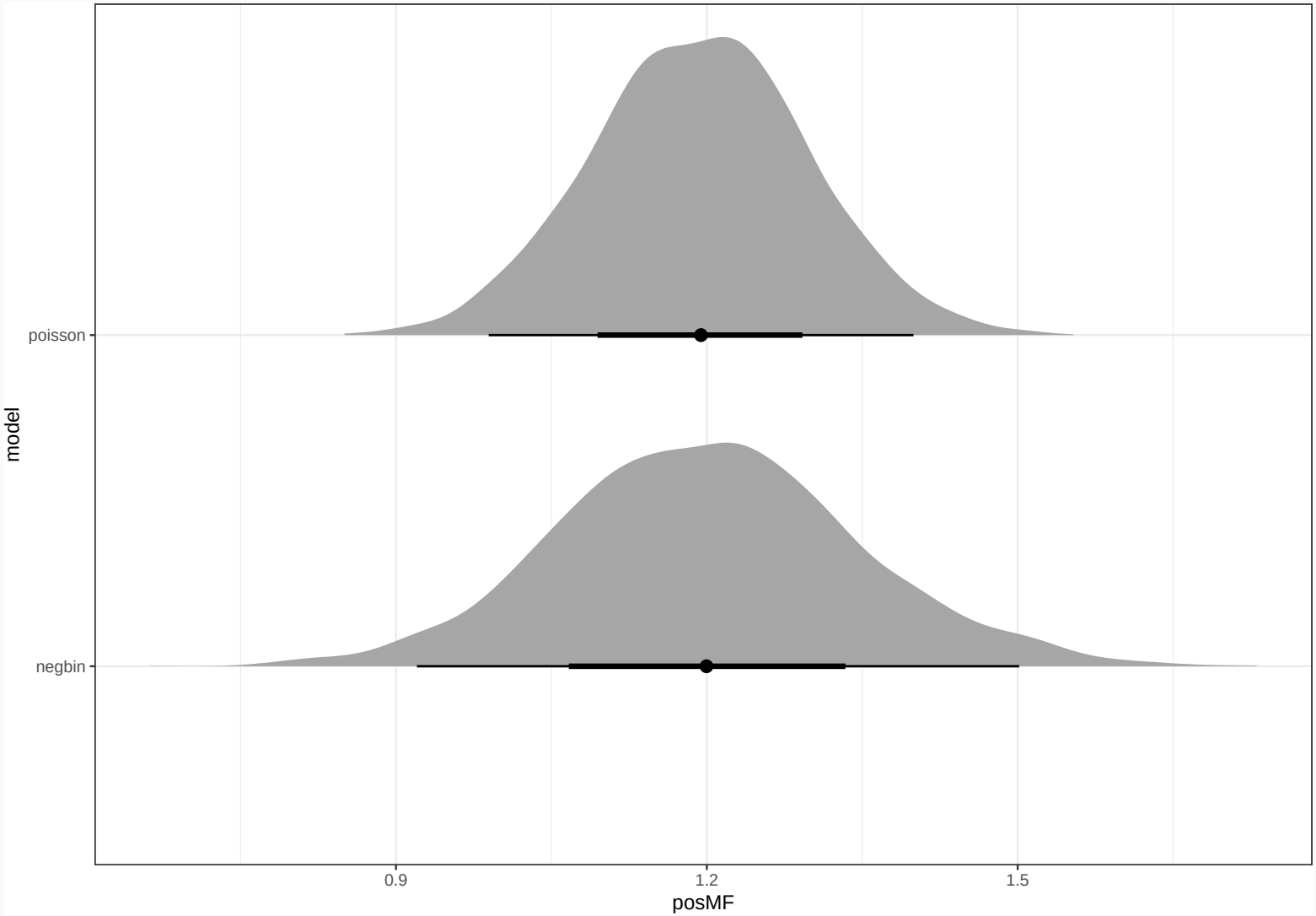
Posterior distributions of Beta: Forwards

```
ggplot(plot_dat, aes(x = posFW, y = model)) + stat_halfeye()
```



Posterior distributions of Beta: Midfielders

```
ggplot(plot_dat, aes(x = posMF, y = model)) + stat_halfeye()
```



Overdispersion and count models

Poisson likelihoods nearly always underestimate standard errors in complex social processes (especially under clustering).

Our models *must* account for overdispersion if we want reasonable uncertainty estimates (standard errors, t-tests, prediction error, etc).

Negative binomial handles this problem well. Other approaches can work too!

Let's evaluate predictive performance against the observed: Compute 90 percent predictive posterior intervals and empirical interval for goals

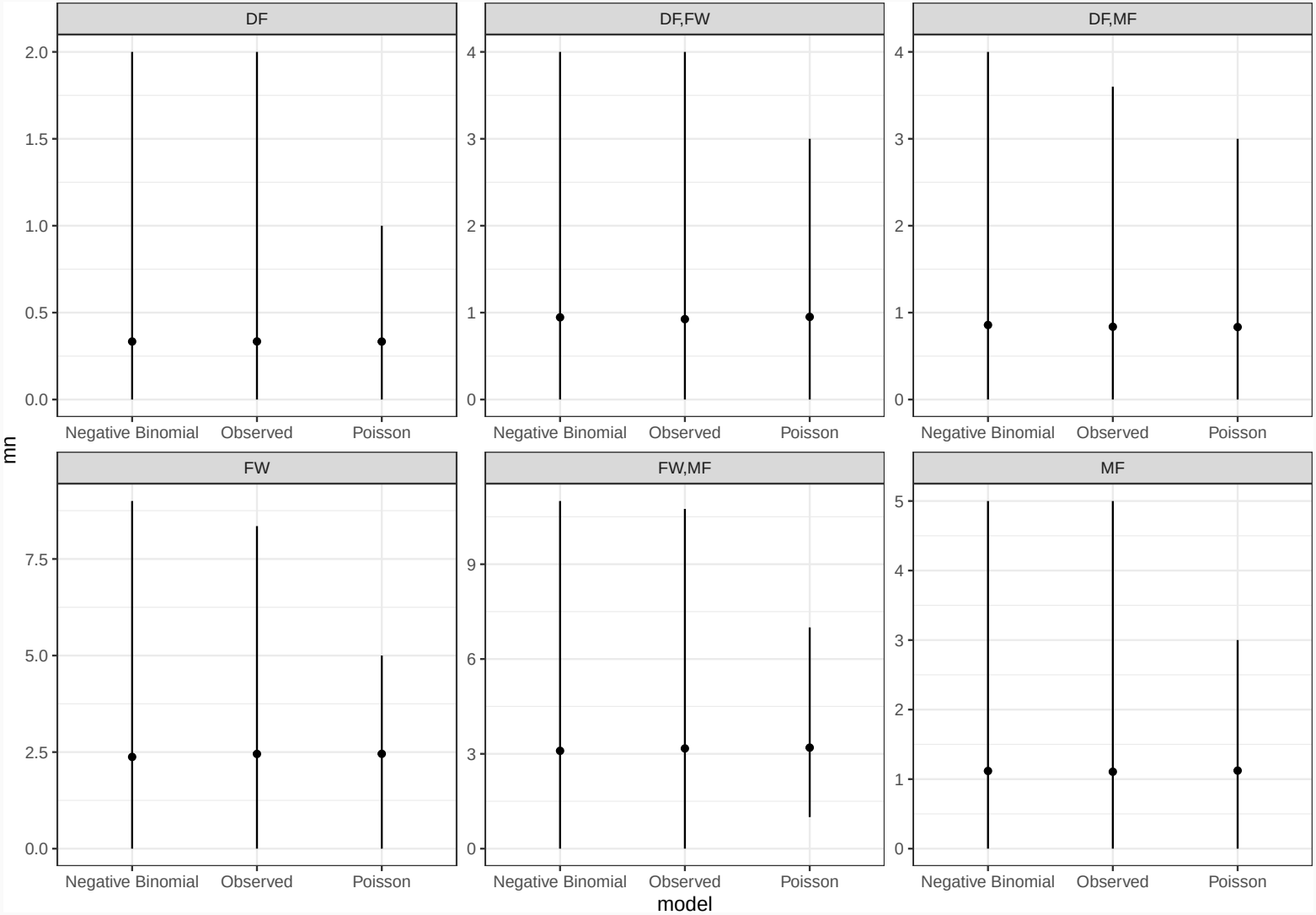
```
pos <- data.frame(pos = unique(nwsl_stats$pos))
pos_pois <- pos %>%
    add_predicted_draws(goals_poisson) %>%
    summarize(mn = mean(.prediction), upr = quantile(.prediction, 0.95), lwr = quantile(.prediction,
        0.05)) %>%
    mutate(model = "Poisson")

pos_negbin <- pos %>%
    add_predicted_draws(goals_negbin) %>%
    summarize(mn = mean(.prediction), upr = quantile(.prediction, 0.95), lwr = quantile(.prediction,
        0.05)) %>%
    mutate(model = "Negative Binomial")

pos_obs <- nwsl_stats %>%
    group_by(pos) %>%
    summarize(mn = mean(gls), upr = quantile(gls, 0.95), lwr = quantile(gls, 0.05)) %>%
    mutate(model = "Observed")

plot_dat <- bind_rows(pos_pois, pos_negbin, pos_obs)
```

Let’s visualize the difference in model predictions



Offsets in event count models

Offsets can improve model fit

Goals are a function of position, sure, but also a function of how many games a player appeared in.

Offsets can improve model fit

Goals are a function of position, sure, but also a function of how many games a player appeared in.

An *offset* term can be added to our model to convert our count into a rate.

Offsets can improve model fit

Goals are a function of position, sure, but also a function of how many games a player appeared in.

An *offset* term can be added to our model to convert our count into a rate.

Here, we can add `mp` as a measure of time

The model for the mean

Using matches played as our offset variable

$$\text{goals} \sim \text{NegBin}(\mu, \theta)$$

$$\log(\mu) = \beta \times \text{position} + \log(\text{mp})$$

A generic form

$$\log(\mu) = \beta X + \log(\text{offset})$$

A generic form

$$\log(\mu) = \beta X + \log(\text{offset})$$

Because $log(x) - log(y) = log(x/y)$ This can be rewritten as

$$\log\left(\frac{\mu}{\text{offset}}\right) = \beta X$$

A generic form

$$\log(\mu) = \beta X + \log(\text{offset})$$

Because $log(x) - log(y) = log(x/y)$ This can be rewritten as

$$\log\left(\frac{\mu}{\text{offset}}\right) = \beta X$$

That's a rate! With the inverse of the link function (e here), we can write this as

$$\frac{\mu}{\text{offset}} = e^{\beta X}$$

Let’s fit the model again

```
goals_negbin_offset <- stan_glm(gls ~ pos, data = nwsl_stats, offset = log(mp), family = "neg_binomial_2",
    refresh = 0)
```

Now we can compare model fits

```
loo_compare(loo(goals_negbin), loo(goals_negbin_offset))
```

```
##                      elpd_diff se_diff
## goals_negbin_offset    0.0       0.0
## goals_negbin        -191.6      11.8
```

The offset dramatically improves our model fit. Let's see how this works.

The regression parameters

Let's compute the estimated number of goals under each model for a forward

$$\log(\text{goals}) = \beta_0 + \beta_1 \times \text{position}$$

```
preds_1 <- add_epred_draws(goals_negbin, newdata = data.frame(pos = "FW"))
```

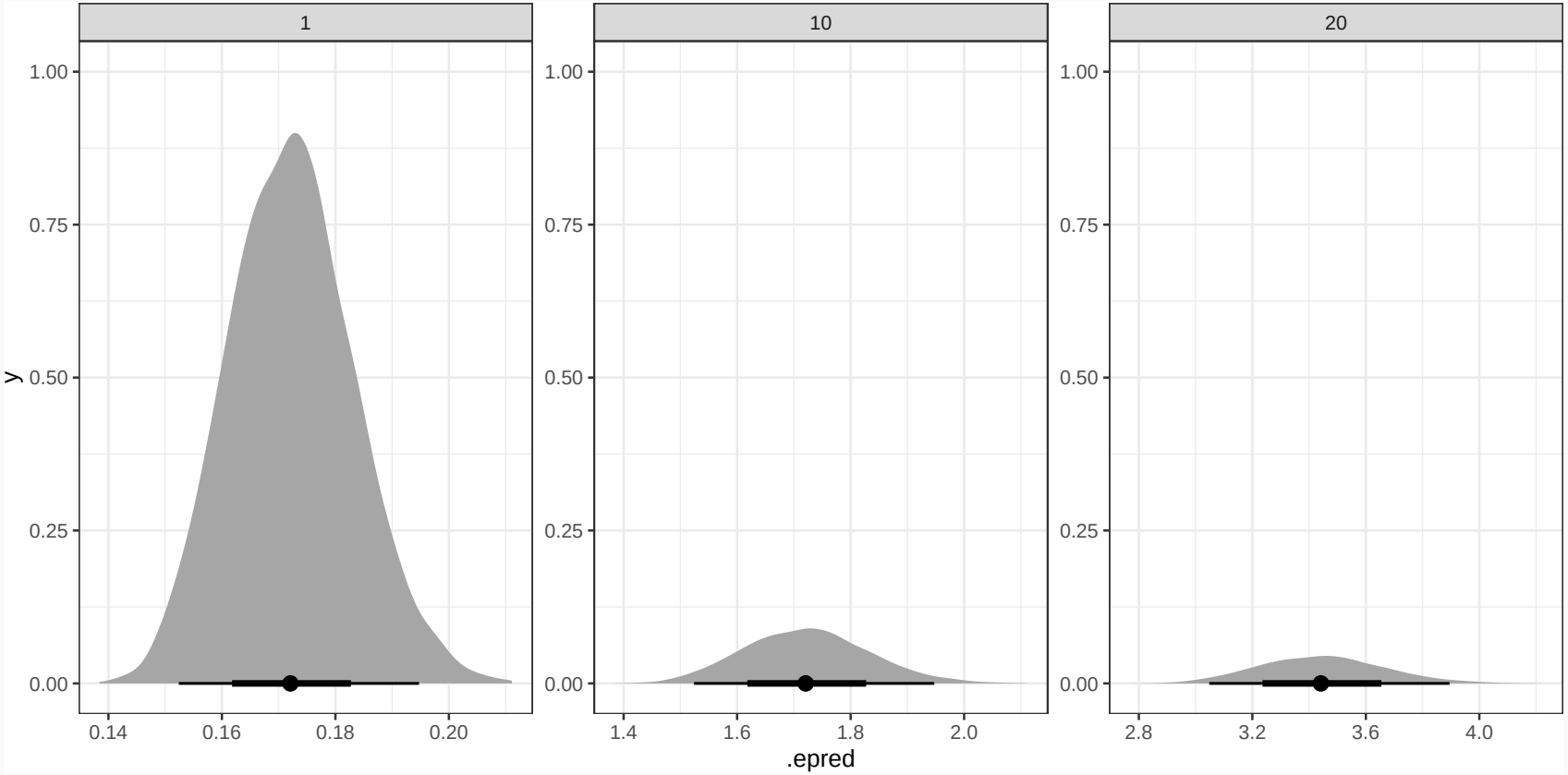
The regression parameters: offset model

$$E\left[\log(\frac{\text{goals}}{\text{games}})\right] = \beta_0 + \beta_1 \times \text{position}$$

```
preds_2 <- add_epred_draws(goals_negbin_offset, newdata = data.frame(pos = rep("FW",
    3), mp = c(1, 10, 20)), offset = log(c(1, 10, 20)))
```

And visualize the posterior expected values

```
ggplot(preds_2, aes(x = .epred)) + stat_halfeye() + facet_wrap(~mp, scales = "free")
```



The difference between an offset and a predictor

```
goals_negbin2 <- stan_glm(gls ~ pos + mp, data = nwsl_stats, family = "neg_binomial_2",
    refresh = 0)

coef(goals_negbin_offset)
```

```
## (Intercept)    posDF,FW    posDF,MF       posFW    posFW,MF       posMF
##  -3.6947551   1.0602888   0.7541335   1.9357007   2.1267232   1.1586680
```

```
coef(goals_negbin2)
```

```
## (Intercept)    posDF,FW    posDF,MF       posFW    posFW,MF       posMF
##  -3.2857103   1.1166067   0.7342340   1.9963171   2.1934796   1.1557684
##          mp
##   0.1366723
```

Compare fits

```
loo_compare(loo(goals_negbin2), loo(goals_negbin_offset), loo(goals_negbin))
```

```
##                      elpd_diff se_diff
## goals_negbin2           0.0       0.0
## goals_negbin_offset    -8.8       8.8
## goals_negbin         -200.4      17.3
```