Question-01:

A researcher is interested in how variables, such as GRE (Graduate Record Exam scores), GPA (grade point average) and prestige of the undergraduate institution, effect admission into graduate school. The response variable, admit/don't admit, is a binary variable. The data set taken from https://stats.idre.ucla.edu/stat/data/binary.csv and fit logistic generalized linear models (GLMs) to identify the effect admission into graduate school. Interpret the result.

Answer-01:

For our data analysis below:

```
install.packages("aod")
install.packages("ggplot2")
library(aod)
library(ggplot2)
mydata <- read.csv("https://stats.idre.ucla.edu/stat/data/binary.csv")
## view the first few rows of the data
head(mydata)
```

```
##   admit gre  gpa rank
## 1     0 380 3.61    3
## 2     1 660 3.67    3
## 3     1 800 4.00    1
## 4     1 640 3.19    4
## 5     0 520 2.93    4
## 6     1 760 3.00    2
```

This dataset has a binary response (outcome, dependent) variable called admit. There are three predictor variables: gre, gpa and rank. We will treat the variables gre and gpa as continuous. The variable rank takes on the values 1 through 4. Institutions with a rank of 1 have the highest prestige, while those with a rank of 4 have the lowest. We can get basic descriptives for the entire data set by using summary. To get the standard deviations, we use sapply to apply the sd function to each variable in the dataset.

```
summary(mydata)
```

```
##      admit          gre          gpa          rank
## Min.   :0.000   Min.   :220   Min.   :2.26   Min.   :1.00
```

```
##  1st Qu.:0.000   1st Qu.:520   1st Qu.:3.13   1st Qu.:2.00
##  Median :0.000   Median :580   Median :3.40   Median :2.00
##  Mean   :0.318   Mean   :588   Mean   :3.39   Mean   :2.48
##  3rd Qu.:1.000   3rd Qu.:660   3rd Qu.:3.67   3rd Qu.:3.00
##  Max.   :1.000   Max.   :800   Max.   :4.00   Max.   :4.00
```

```
sapply(mydata, sd)
##   admit     gre     gpa    rank
##   0.466 115.517   0.381   0.944
```

```
## two-way contingency table of categorical outcome and predictors we want
## to make sure there are not 0 cells
xtabs(~admit + rank, data = mydata)
```

```
##      rank
## admit  1  2  3  4
##     0 28 97 93 55
##     1 33 54 28 12
```

Analysis methods you might consider:

- Below is a list of some analysis methods you may have encountered. Some of the methods listed are quite reasonable while others have either fallen out of favor or have limitations.

- Logistic regression, the focus of this page.
- Probit regression. Probit analysis will produce results similar logistic regression. The choice of probit versus logit depends largely on individual preferences.
- OLS regression. When used with a binary response variable, this model is known as a linear probability model and can be used as a way to describe conditional probabilities. However, the errors (i.e., residuals) from the linear probability model violate the homoskedasticity and normality of errors assumptions of OLS regression, resulting in invalid standard errors and hypothesis tests. For a more thorough discussion of these and other problems with the linear probability model, see Long (1997, p. 38-40).

- Two-group discriminant function analysis. A multivariate method for dichotomous outcome variables.
- Hotelling's T2. The 0/1 outcome is turned into the grouping variable, and the former predictors are turned into outcome variables. This will produce an overall test of significance but will not give individual coefficients for each variable, and it is unclear the extent to which each "predictor" is adjusted for the impact of the other "predictors."

Using the logit model:

The code below estimates a logistic regression model using the glm (generalized linear model) function. First, we convert rank to a factor to indicate that rank should be treated as a categorical variable.

```
mydata$rank <- factor(mydata$rank)
mylogit <- glm(admit ~ gre + gpa + rank, data = mydata, family = "binomial")
```

Since we gave our model a name (mylogit), R will not produce any output from our regression. In order to get the results we use the summary command:

```
summary(mylogit)
```

```
##
## Call:
## glm(formula = admit ~ gre + gpa + rank, family = "binomial",
##     data = mydata)
##
## Deviance Residuals:
##    Min      1Q  Median      3Q     Max
## -1.627  -0.866  -0.639   1.149   2.079
##
## Coefficients:
##             Estimate Std. Error z value Pr(>|z|)
## (Intercept) -3.98998    1.13995   -3.50  0.00047 ***
## gre          0.00226    0.00109    2.07  0.03847 *
## gpa          0.80404    0.33182    2.42  0.01539 *
## rank2       -0.67544    0.31649   -2.13  0.03283 *
## rank3       -1.34020    0.34531   -3.88  0.00010 ***
## rank4       -1.55146    0.41783   -3.71  0.00020 ***
```

```
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 499.98  on 399  degrees of freedom
## Residual deviance: 458.52  on 394  degrees of freedom
## AIC: 470.5
##
## Number of Fisher Scoring iterations: 4
```

In the output above, the first thing we see is the call, this is R reminding us what the model we ran was, what options we specified, etc.

Next we see the deviance residuals, which are a measure of model fit. This part of output shows the distribution of the deviance residuals for individual cases used in the model. Below we discuss how to use summaries of the deviance statistic to assess model fit.

The next part of the output shows the coefficients, their standard errors, the z-statistic (sometimes called a Wald z-statistic), and the associated p-values. Both gre and gpa are statistically significant, as are the three terms for rank. The logistic regression coefficients give the change in the log odds of the outcome for a one unit increase in the predictor variable.

- For every one unit change in gre, the log odds of admission (versus non-admission) increases by 0.002.
- For a one unit increase in gpa, the log odds of being admitted to graduate school increases by 0.804.
- The indicator variables for rank have a slightly different interpretation. For example, having attended an undergraduate institution with rank of 2, versus an institution with a rank of 1, changes the log odds of admission by -0.675.

Below the table of coefficients are fit indices, including the null and deviance residuals and the AIC. Later we show an example of how you can use these values to help assess model fit.

We can use the confint function to obtain confidence intervals for the coefficient estimates. Note that for logistic models, confidence intervals are based on the profiled log-likelihood function. We can also get CIs based on just the standard errors by using the default method.

```
## CIs using profiled log-likelihood
confint(mylogit)
```

```
##                2.5 %   97.5 %
## (Intercept) -6.271620 -1.79255
## gre          0.000138  0.00444
## gpa          0.160296  1.46414
## rank2       -1.300889 -0.05675
## rank3       -2.027671 -0.67037
## rank4       -2.400027 -0.75354
```

```
## CIs using standard errors
confint.default(mylogit)
```

```
##                2.5 %   97.5 %
## (Intercept) -6.22424 -1.75572
## gre          0.00012  0.00441
## gpa          0.15368  1.45439
## rank2       -1.29575 -0.05513
## rank3       -2.01699 -0.66342
## rank4       -2.37040 -0.73253
```

We can test for an overall effect of rank using the wald.test function of the aod library. The order in which the coefficients are given in the table of coefficients is the same as the order of the terms in the model. This is important because the wald.test function refers to the coefficients by their order in the model. We use the wald.test function. b supplies the coefficients, while Sigma supplies the variance covariance matrix of the error terms, finally Terms tells R which terms in the model are to be tested, in this case, terms 4, 5, and 6, are the three terms for the levels of rank.

```
wald.test(b = coef(mylogit), Sigma = vcov(mylogit), Terms = 4:6)
```

```
## Wald test:
## ----------
##
## Chi-squared test:
## X2 = 20.9, df = 3, P(> X2) = 0.00011
```

The chi-squared test statistic of 20.9, with three degrees of freedom is associated with a p-value of 0.00011 indicating that the overall effect of rank is statistically significant.

```
l <- cbind(0, 0, 0, 1, -1, 0)
wald.test(b = coef(mylogit), Sigma = vcov(mylogit), L = l)
```

```
## Wald test:
## ----------
##
## Chi-squared test:
## X2 = 5.5, df = 1, P(> X2) = 0.019
```

The chi-squared test statistic of 5.5 with 1 degree of freedom is associated with a p-value of 0.019, indicating that the difference between the coefficient for rank=2 and the coefficient for rank=3 is statistically significant.

```
## odds ratios only
exp(coef(mylogit))
```

```
## (Intercept)       gre       gpa     rank2     rank3     rank4
##      0.0185    1.0023    2.2345    0.5089    0.2618    0.2119
```

```
## odds ratios and 95% CI
exp(cbind(OR = coef(mylogit), confint(mylogit)))
```

```
##                OR   2.5 % 97.5 %
## (Intercept) 0.0185 0.00189  0.167
## gre         1.0023 1.00014  1.004
## gpa         2.2345 1.17386  4.324
## rank2       0.5089 0.27229  0.945
## rank3       0.2618 0.13164  0.512
## rank4       0.2119 0.09072  0.471
```

Now we can say that for a one unit increase in gpa, the odds of being admitted to graduate school (versus not being admitted) increase by a factor of 2.23.

```
newdata1 <- with(mydata, data.frame(gre = mean(gre), gpa = mean(gpa), rank = factor(1:4)))
```

```
## view data frame
newdata1
```

```
##   gre  gpa rank
## 1 588 3.39    1
## 2 588 3.39    2
## 3 588 3.39    3
## 4 588 3.39    4
```

We will start by calculating the predicted probability of admission at each value of rank, holding gre and gpa at their means. First we create and view the data frame.

```
newdata1$rankP <- predict(mylogit, newdata = newdata1, type = "response")
newdata1
```

```
##   gre  gpa rank rankP
## 1 588 3.39    1 0.517
## 2 588 3.39    2 0.352
## 3 588 3.39    3 0.219
## 4 588 3.39    4 0.185
```

In the above output we see that the predicted probability of being accepted into a graduate program is 0.52 for students from the highest prestige undergraduate institutions (rank=1), and 0.18 for students from the lowest ranked institutions (rank=4), holding gre and gpa at their means. We can do something very similar to create a table of predicted probabilities varying the value of gre and rank. We are going to plot these, so we will create 100 values of gre between 200 and 800, at each value of rank (i.e., 1, 2, 3, and 4).

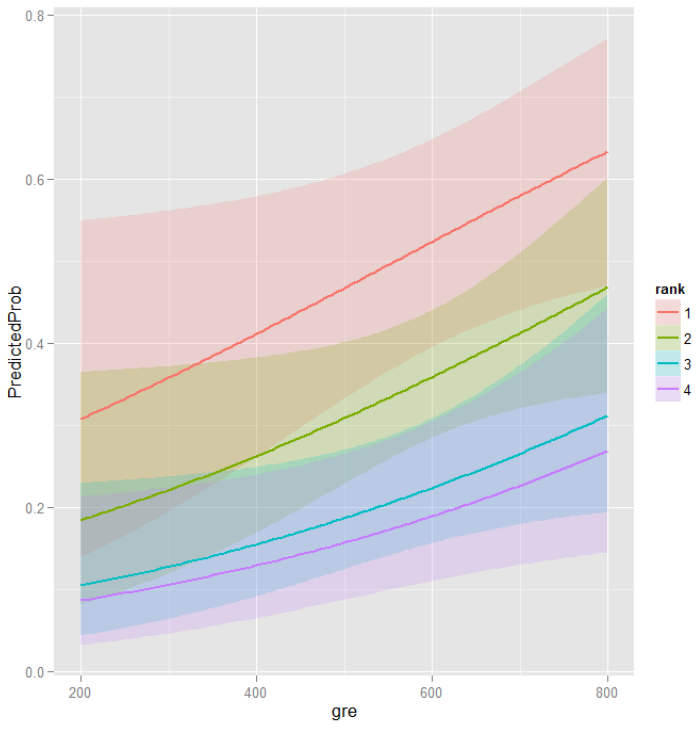
```
newdata2 <- with(mydata, data.frame(gre = rep(seq(from = 200, to = 800, length.out = 100),
  4), gpa = mean(gpa), rank = factor(rep(1:4, each = 100))))
newdata3 <- cbind(newdata2, predict(mylogit, newdata = newdata2, type = "link",
  se = TRUE))
newdata3 <- within(newdata3, {
  PredictedProb <- plogis(fit)
  LL <- plogis(fit - (1.96 * se.fit))
```

```
  UL <- plogis(fit + (1.96 * se.fit))
})

## view first few rows of final dataset
head(newdata3)
```

```
##   gre  gpa rank    fit se.fit residual.scale    UL    LL PredictedProb
## 1 200 3.39    1 -0.811  0.515              1 0.549 0.139         0.308
## 2 206 3.39    1 -0.798  0.509              1 0.550 0.142         0.311
## 3 212 3.39    1 -0.784  0.503              1 0.551 0.145         0.313
## 4 218 3.39    1 -0.770  0.498              1 0.551 0.149         0.316
## 5 224 3.39    1 -0.757  0.492              1 0.552 0.152         0.319
## 6 230 3.39    1 -0.743  0.487              1 0.553 0.155         0.322
```

It can also be helpful to use graphs of predicted probabilities to understand and/or present the model. We will use the ggplot2 package for graphing. Below we make a plot with the predicted probabilities, and 95% confidence intervals.

```
ggplot(newdata3, aes(x = gre, y = PredictedProb)) + geom_ribbon(aes(ymin = LL,
    ymax = UL, fill = rank), alpha = 0.2) + geom_line(aes(colour = rank),
    size = 1)
```



```
with(mylogit, null.deviance - deviance)
```

```
## [1] 41.5
```

The degrees of freedom for the difference between the two models is equal to the number of predictor variables in the mode, and can be obtained using:

```
with(mylogit, df.null - df.residual)
```

```
## [1] 5
```

Finally, the p-value can be obtained using:

```
with(mylogit, pchisq(null.deviance - deviance, df.null - df.residual, lower.tail = FALSE))
```

```
## [1] 7.58e-08
```

The chi-square of 41.46 with 5 degrees of freedom and an associated p-value of less than 0.001 tells us that our model as a whole fits significantly better than an empty model. This is sometimes called a likelihood ratio test (the deviance residual is -2*log likelihood). To see the model's log likelihood, we type:

```
logLik(mylogit)
```

```
## 'log Lik.' -229 (df=6)
```

Things to consider:

- Empty cells or small cells: You should check for empty or small cells by doing a crosstab between categorical predictors and the outcome variable. If a cell has very few cases (a small cell), the model may become unstable or it might not run at all.
- Separation or quasi-separation (also called perfect prediction), a condition in which the outcome does not vary at some levels of the independent variables. See our page FAQ: What is complete or quasi-complete separation in logistic/probit regression and how do we deal with them? for information on models with perfect prediction.
- Sample size: Both logit and probit models require more cases than OLS regression because they use maximum likelihood estimation techniques. It is sometimes possible to estimate models for binary outcomes in datasets with only a small number of cases using exact logistic regression. It is also important to keep in mind that when the outcome is rare, even if the overall dataset is large, it can be difficult to estimate a logit model.

- Pseudo-R-squared: Many different measures of psuedo-R-squared exist. They all attempt to provide information similar to that provided by R-squared in OLS regression; however, none of them can be interpreted exactly as R-squared in OLS regression is interpreted. For a discussion of various pseudo-R-squareds see Long and Freese (2006) or our FAQ page What are pseudo R-squareds?
- Diagnostics: The diagnostics for logistic regression are different from those for OLS regression. For a discussion of model diagnostics for logistic regression, see Hosmer and Lemeshow (2000, Chapter 5). Note that diagnostics done for logistic regression are similar to those done for probit regression.

Question-02:

The number of awards earned by students at one high school. Predictors of the number of awards earned include the type of program in which the student was enrolled (e.g., vocational, general or academic) and the score on their final exam in math. The data set is taken from **https://stats.idre.ucla.edu/stat/data/poisson_sim.csv** and fit the poisson generalized linear models (GLMs) to identify the factors associated with number of awards earned by students at one high school. Interpret the result.

Answer-02:

Let's start with loading the data and looking at some descriptive statistics.

```
install.packages("msm")
install.packages("sandwich")
require(ggplot2)
require(sandwich)
require(msm)
```

```
p <- read.csv("https://stats.idre.ucla.edu/stat/data/poisson_sim.csv")
p <- within(p, {
  prog <- factor(prog, levels=1:3, labels=c("General", "Academic",
                                            "Vocational"))
  id <- factor(id)
})
summary(p)
```

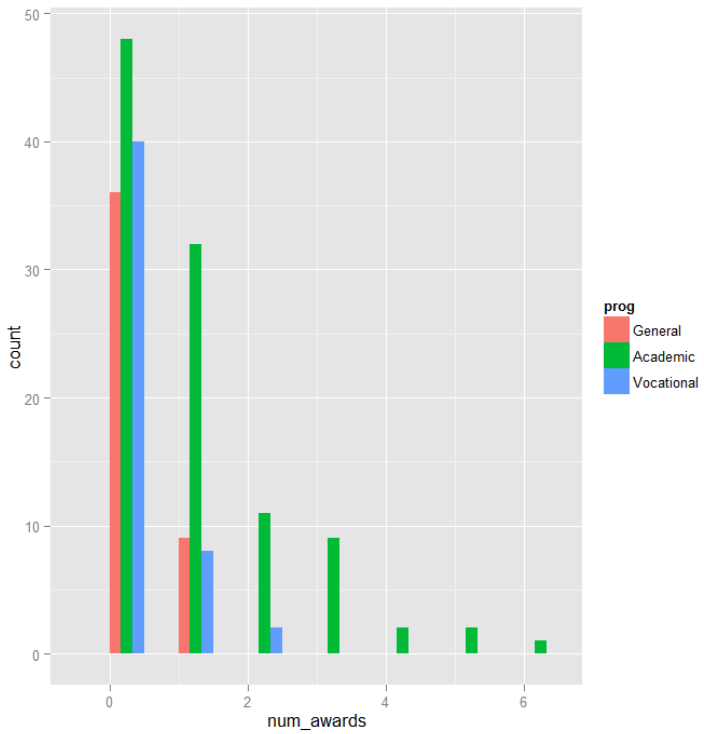
```
##        id         num_awards          prog         math
## 1      : 1   Min.   :0.00   General   : 45   Min.   :33.0
```

```
##  2     : 1   1st Qu.:0.00   Academic  :105   1st Qu.:45.0
##  3     : 1   Median :0.00   Vocational: 50   Median :52.0
##  4     : 1   Mean   :0.63                    Mean   :52.6
##  5     : 1   3rd Qu.:1.00                    3rd Qu.:59.0
##  6     : 1   Max.   :6.00                    Max.   :75.0
##  (Other):194
```

```
with(p, tapply(num_awards, prog, function(x) {
  sprintf("M (SD) = %1.2f (%1.2f)", mean(x), sd(x))
}))
```

```
##                 General                Academic                Vocational
## "M (SD) = 0.20 (0.40)" "M (SD) = 1.00 (1.28)" "M (SD) = 0.24 (0.52)"
```

```
ggplot(p, aes(num_awards, fill = prog)) +
  geom_histogram(binwidth=.5, position="dodge")
```



Analysis methods you might consider:

Below is a list of some analysis methods you may have encountered. Some of the methods listed are quite reasonable, while others have either fallen out of favor or have limitations.

- Poisson regression – Poisson regression is often used for modeling count data. Poisson regression has a number of extensions useful for count models.
- Negative binomial regression – Negative binomial regression can be used for over-dispersed count data, that is when the conditional variance exceeds the conditional mean. It can be considered as a generalization of Poisson regression since it has the same mean structure as Poisson regression and it has an extra parameter to model the over-dispersion. If the conditional distribution of the outcome variable is over-dispersed, the confidence intervals for coefficients in Negative binomial regression are likely to be wider as compared to those from a Poisson regression.
- Zero-inflated regression model – Zero-inflated models attempt to account for excess zeros. In other words, two kinds of zeros are thought to exist in the data, "true zeros" and "excess zeros". Zero-inflated models estimate two equations simultaneously, one for the count model and one for the excess zeros.
- OLS regression – Count outcome variables are sometimes log-transformed and analyzed using OLS regression. Many issues arise with this approach, including loss of data due to undefined values generated by taking the log of zero (which is undefined) and biased estimates.

Poisson regression:

At this point, we are ready to perform our Poisson model analysis using the glm function. We fit the model and store it in the object m1 and get a summary of the model at the same time.

```
summary(m1 <- glm(num_awards ~ prog + math, family="poisson", data=p))
```

```
##
## Call:
## glm(formula = num_awards ~ prog + math, family = "poisson", data = p)
##
## Deviance Residuals:
##    Min      1Q  Median      3Q     Max
## -2.204  -0.844  -0.511   0.256   2.680
##
## Coefficients:
```

```
##                Estimate Std. Error z value Pr(>|z|)
## (Intercept)     -5.2471     0.6585   -7.97  1.6e-15 ***
## progAcademic     1.0839     0.3583    3.03   0.0025 **
## progVocational   0.3698     0.4411    0.84   0.4018
## math             0.0702     0.0106    6.62  3.6e-11 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for poisson family taken to be 1)
##
##     Null deviance: 287.67  on 199  degrees of freedom
## Residual deviance: 189.45  on 196  degrees of freedom
## AIC: 373.5
##
## Number of Fisher Scoring iterations: 6
```

```
cov.m1 <- vcovHC(m1, type="HC0")
std.err <- sqrt(diag(cov.m1))
r.est <- cbind(Estimate= coef(m1), "Robust SE" = std.err,
"Pr(>|z|)"  = 2 * pnorm(abs(coef(m1)/std.err), lower.tail=FALSE),
LL = coef(m1) - 1.96 * std.err,
UL = coef(m1) + 1.96 * std.err)


r.est
```

```
##                Estimate Robust SE  Pr(>|z|)      LL       UL
## (Intercept)    -5.24712   0.64600 4.567e-16 -6.5133 -3.98097
## progAcademic    1.08386   0.32105 7.355e-04  0.4546  1.71311
## progVocational  0.36981   0.40042 3.557e-01 -0.4150  1.15463
## math            0.07015   0.01044 1.784e-11  0.0497  0.09061
with(m1, cbind(res.deviance = deviance, df = df.residual,
  p = pchisq(deviance, df.residual, lower.tail=FALSE)))
```

Now let's look at the output of function glm more closely.

- The output begins with echoing the function call. The information on deviance residuals is displayed next. Deviance residuals are approximately normally distributed if the model is specified correctly.In our example, it shows a little bit of skeweness since median is not quite zero.
- Next come the Poisson regression coefficients for each of the variables along with the standard errors, z-scores, p-values and 95% confidence intervals for the coefficients. The coefficient for math is .07. This means that the expected log count for a one-unit increase in math is .07. The indicator variable progAcademic compares between prog = "Academic" and prog = "General", the expected log count for prog = "Academic" increases by about 1.1. The indicator variable prog.Vocational is the expected difference in log count ((approx .37)) between prog = "Vocational" and the reference group (prog = "General").
- The information on deviance is also provided. We can use the residual deviance to perform a goodness of fit test for the overall model. The residual deviance is the difference between the deviance of the current model and the maximum deviance of the ideal model where the predicted values are identical to the observed. Therefore, if the residual difference is small enough, the goodness of fit test will not be significant, indicating that the model fits the data. We conclude that the model fits reasonably well because the goodness-of-fit chi-squared test is not statistically significant. If the test had been statistically significant, it would indicate that the data do not fit the model well. In that situation, we may try to determine if there are omitted predictor variables, if our linearity assumption holds and/or if there is an issue of over-dispersion.

```
with(m1, cbind(res.deviance = deviance, df = df.residual,
  p = pchisq(deviance, df.residual, lower.tail=FALSE)))
```

```
##     res.deviance  df     p
## [1,]       189.4 196 0.6182
```

We can also test the overall effect of prog by comparing the deviance of the full model with the deviance of the model excluding prog. The two degree-of-freedom chi-square test indicates that prog, taken together, is a statistically significant predictor of num_awards.

```
## update m1 model dropping prog
m2 <- update(m1, . ~ . - prog)
## test model differences  with chi square test
anova(m2, m1, test="Chisq")
```

```
## Analysis of Deviance Table
##
## Model 1: num_awards ~ math
## Model 2: num_awards ~ prog + math
##   Resid. Df Resid. Dev Df Deviance Pr(>Chi)
## 1       198       204
## 2       196       189  2     14.6  0.00069 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Sometimes, we might want to present the regression results as incident rate ratios and their standard errors, together with the confidence interval. To compute the standard error for the incident rate ratios, we will use the Delta method. To this end, we make use the function deltamethod implemented in R package msm.

```
s <- deltamethod(list(~  exp(x1), ~ exp(x2), ~ exp(x3), ~ exp(x4)),
                                    coef(m1), cov.m1)


## exponentiate old estimates dropping the p values
rexp.est <- exp(r.est[, -3])
## replace SEs with estimates for exponentiated coefficients
rexp.est[, "Robust SE"] <- s


rexp.est
```

```
##                Estimate Robust SE       LL      UL
## (Intercept)    0.005263   0.00340 0.001484 0.01867
## progAcademic   2.956065   0.94904 1.575551 5.54620
## progVocational 1.447458   0.57959 0.660335 3.17284
## math           1.072672   0.01119 1.050955 1.09484
```

The output above indicates that the incident rate for prog = “Academic” is 2.96 times the incident rate for the reference group (prog = “General”). Likewise, the incident rate for prog = “Vocational” is 1.45 times the incident rate for the reference group holding the other variables at constant. The percent change in the incident rate of num_awards is by 7% for every unit increase in math.

Sometimes, we might want to look at the expected marginal means. For example, what are the expected counts for each program type holding math score at its overall mean? To answer this question, we can make use of the predict function. First off, we will make a small data set to apply the predict function to it.

```
(s1 <- data.frame(math = mean(p$math),
  prog = factor(1:3, levels = 1:3, labels = levels(p$prog))))
```

```
##    math       prog
## 1 52.65    General
## 2 52.65   Academic
## 3 52.65 Vocational
```

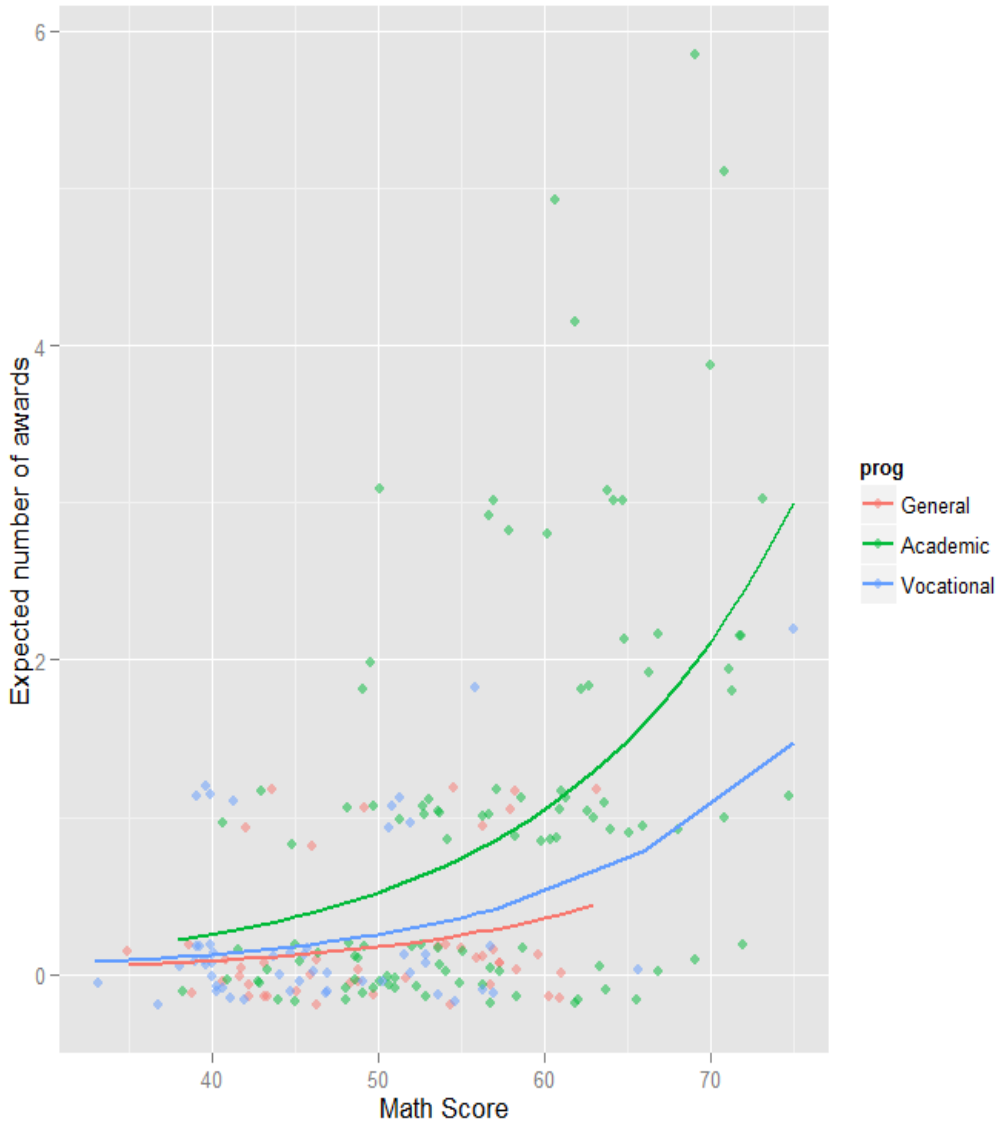
```
predict(m1, s1, type="response", se.fit=TRUE)
```

```
## $fit
##      1      2      3
## 0.2114 0.6249 0.3060
##
## $se.fit
##       1       2       3
## 0.07050 0.08628 0.08834
##
## $residual.scale
## [1] 1
```

In the output above, we see that the predicted number of events for level 1 of prog is about .21, holding math at its mean. The predicted number of events for level 2 of prog is higher at .62, and the predicted number of events for level 3 of prog is about .31. The ratios of these predicted counts ((frac{.625}{.211} = 2.96), (frac{.306}{.211} = 1.45)) match what we saw looking at the IRR.

```
## calculate and store predicted values
p$phat <- predict(m1, type="response")


## order by program and then by math
```

```
p <- p[with(p, order(prog, math)), ]


## create the plot
ggplot(p, aes(x = math, y = phat, colour = prog)) +
  geom_point(aes(y  = num_awards), alpha=.5, position=position_jitter(h=.2))  +
  geom_line(size  = 1) +
  labs(x = "Math Score", y = "Expected number of awards")
```



Things to consider:

- When there seems to be an issue of dispersion, we should first check if our model is appropriately specified, such as omitted variables and functional forms. For example, if we omitted the predictor variable prog in the example above, our model would seem to have a problem with over-dispersion. In other

words, a misspecified model could present a symptom like an over-dispersion problem.

- Assuming that the model is correctly specified, the assumption that the conditional variance is equal to the conditional mean should be checked. There are several tests including the likelihood ratio test of over-dispersion parameter alpha by running the same model using negative binomial distribution. R package pscl (Political Science Computational Laboratory, Stanford University) provides many functions for binomial and count data including odTest for testing over-dispersion.
- One common cause of over-dispersion is excess zeros, which in turn are generated by an additional data generating process. In this situation, zero-inflated model should be considered.
- If the data generating process does not allow for any 0s (such as the number of days spent in the hospital), then a zero-truncated model may be more appropriate.
- Count data often have an exposure variable, which indicates the number of times the event could have happened. This variable should be incorporated into a Poisson model with the use of the offset option.
- The outcome variable in a Poisson regression cannot have negative numbers, and the exposure cannot have 0s.
- Many different measures of pseudo-R-squared exist. They all attempt to provide information similar to that provided by R-squared in OLS regression, even though none of them can be interpreted exactly as R-squared in OLS regression is interpreted. For a discussion of various pseudo-R-squares, see Long and Freese (2006) or our FAQ page What are pseudo R-squareds?.
- Poisson regression is estimated via maximum likelihood estimation. It usually requires a large sample size.

Question-03:

School administrators study the attendance behavior of high school juniors at two schools. Predictors of the number of days of absence include the type of program in which the student is enrolled and a standardized test in math. The data set is taken from **("https://stats.idre.ucla.edu/stat/stata/dae/nb_data.dta")** and fit the negative binomial generalized linear models (GLMs) to identify the factors associated with number of awards earned by students at one high school. Interpret the result.

Answer-03:

Let's look at the data. It is always a good idea to start with descriptive statistics and plots.

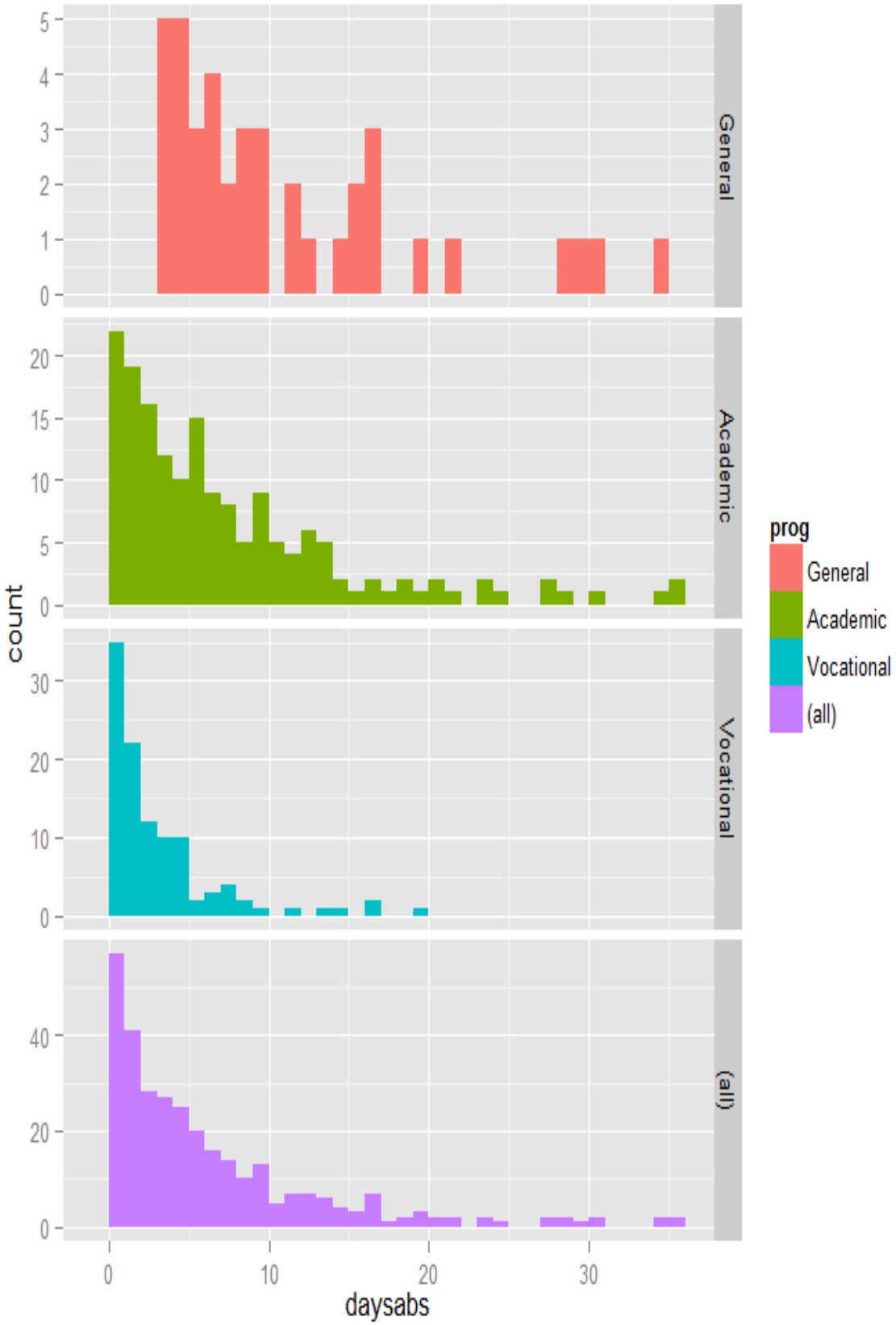
```
install.packages("foreign")
install.packages("MASS")
require(foreign)
require(ggplot2)
```

```
require(MASS)
```

```
dat <- read.dta("https://stats.idre.ucla.edu/stat/stata/dae/nb_data.dta")
dat <- within(dat, {
   prog <- factor(prog, levels = 1:3, labels = c("General",  "Academic", "Vocational"))
   id <- factor(id)
})


summary(dat)
```

```
##        id         gender        math         daysabs
##  1001   :  1   female:160   Min.   : 1.0   Min.   : 0.00
##  1002   :  1   male  :154   1st Qu.:28.0   1st Qu.: 1.00
##  1003   :  1                Median :48.0   Median : 4.00
##  1004   :  1                Mean   :48.3   Mean   : 5.96
##  1005   :  1                3rd Qu.:70.0   3rd Qu.: 8.00
##  1006   :  1                Max.   :99.0   Max.   :35.00
##  (Other):308
##          prog
##  General   : 40
##  Academic  :167
##  Vocational:107
```

```
ggplot(dat, aes(daysabs, fill = prog)) + geom_histogram(binwidth = 1) + facet_grid(
prog ~
   ., margins = TRUE, scales = "free")
```



Each variable has 314 valid observations and their distributions seem quite reasonable. The unconditional mean of our outcome variable is much lower than its variance.

Let's continue with our description of the variables in this dataset. The table below shows the average numbers of days absent by program type and seems to suggest that program type is a good candidate for predicting the number of days absent, our outcome variable, because the mean value of the outcome appears to vary by prog. The variances within each level of prog are higher than the means within each level. These are the conditional means and variances. These differences suggest that over-dispersion is present and that a Negative Binomial model would be appropriate.

```
with(dat, tapply(daysabs, prog, function(x) {
  sprintf("M (SD) = %1.2f (%1.2f)", mean(x), sd(x))
}))
```

```
##              General             Academic             Vocational
## "M (SD) = 10.65 (8.20)"  "M (SD) = 6.93 (7.45)"  "M (SD) = 2.67 (3.73)"
```

Analysis methods you might consider:

Below is a list of some analysis methods you may have encountered. Some of the methods listed are quite reasonable, while others have either fallen out of favor or have limitations.

- Negative binomial regression -Negative binomial regression can be used for over-dispersed count data, that is when the conditional variance exceeds the conditional mean. It can be considered as a generalization of Poisson regression since it has the same mean structure as Poisson regression and it has an extra parameter to model the over-dispersion. If the conditional distribution of the outcome variable is over-dispersed, the confidence intervals for the Negative binomial regression are likely to be wider as compared to those from a Poisson regression model.
- Poisson regression – Poisson regression is often used for modeling count data. Poisson regression has a number of extensions useful for count models.
- Zero-inflated regression model – Zero-inflated models attempt to account for excess zeros. In other words, two kinds of zeros are thought to exist in the data, "true zeros" and "excess zeros". Zero-inflated models estimate two equations simultaneously, one for the count model and one for the excess zeros.
- OLS regression – Count outcome variables are sometimes log-transformed and analyzed using OLS regression. Many issues arise with this approach, including loss of data due to undefined values generated by taking the log of zero (which is undefined), as well as the lack of capacity to model the dispersion.

Negative binomial regression analysis:

Below we use the glm.nb function from the MASS package to estimate a negative binomial regression.

```
summary(m1 <- glm.nb(daysabs ~ math + prog, data = dat))
```

```
## Call:
## glm.nb(formula = daysabs ~ math + prog, data = dat, init.theta = 1.032713156,
##     link = log)
##
## Deviance Residuals:
##    Min      1Q  Median      3Q     Max
## -2.155  -1.019  -0.369   0.229   2.527
##
## Coefficients:
##                 Estimate Std. Error z value Pr(>|z|)
## (Intercept)     2.61527    0.19746   13.24  < 2e-16 ***
## math           -0.00599    0.00251   -2.39    0.017 *
## progAcademic   -0.44076    0.18261   -2.41    0.016 *
## progVocational -1.27865    0.20072   -6.37  1.9e-10 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for Negative Binomial(1.033) family taken to be 1)
##
##     Null deviance: 427.54  on 313  degrees of freedom
## Residual deviance: 358.52  on 310  degrees of freedom
## AIC: 1741
##
## Number of Fisher Scoring iterations: 1
##
##
##               Theta:  1.033
##           Std. Err.:  0.106
##
##  2 x log-likelihood:  -1731.258
```

R first displays the call and the deviance residuals. Next, we see the regression coefficients for each of the variables, along with standard errors, z-scores, and p-values. The variable math has a coefficient of -0.006, which is statistically significant. This means that for each one-unit increase in math, the expected log count of the number of days absent decreases by 0.006. The indicator variable shown as progAcademic is the expected difference in log count between group 2 and the reference group (prog=1). The expected log count for level 2 of prog is 0.44 lower than the expected log count for level 1. The indicator variable for progVocational is the expected difference in log count between group 3 and the reference group.The expected log count for level 3 of prog is 1.28 lower than the expected log count for level 1. To determine if prog itself, overall, is statistically significant, we can compare a model with and without prog. The reason it is important to fit separate models, is that unless we do, the overdispersion parameter is held constant.

```
m2 <- update(m1, . ~ . - prog)
anova(m1, m2)
```

```
## Likelihood ratio tests of Negative Binomial Models
##
## Response: daysabs
##         Model  theta Resid. df    2 x log-lik.   Test    df LR stat.
## 1        math 0.8559       312          -1776
## 2 math + prog 1.0327       310          -1731 1 vs 2     2    45.05
##     Pr(Chi)
## 1
## 2 1.652e-10
```

```
m3 <- glm(daysabs ~ math + prog, family = "poisson", data = dat)
pchisq(2 * (logLik(m1) - logLik(m3)), df = 1, lower.tail = FALSE)
```

```
## 'log Lik.' 2.157e-203 (df=5)
```

The associated chi-squared value estimated from 2*(logLik(m1) – logLik(m3)) is 926.03 with one degree of freedom. This strongly suggests the negative binomial model, estimating the dispersion parameter, is more appropriate than the Poisson model.

We can get the confidence intervals for the coefficients by profiling the likelihood function.

```
(est <- cbind(Estimate = coef(m1), confint(m1)))
```

```
##                 Estimate   2.5 %    97.5 %
## (Intercept)     2.615265  2.2421  3.012936
## math           -0.005993 -0.0109 -0.001067
## progAcademic   -0.440760 -0.8101 -0.092643
## progVocational -1.278651 -1.6835 -0.890078
```

We might be interested in looking at incident rate ratios rather than coefficients. To do this, we can exponentiate our model coefficients. The same applies to the confidence intervals.

```
exp(est)
```

```
##                 Estimate  2.5 %  97.5 %
## (Intercept)     13.6708 9.4127 20.3470
## math             0.9940 0.9892  0.9989
## progAcademic     0.6435 0.4448  0.9115
## progVocational   0.2784 0.1857  0.4106
```

The output above indicates that the incident rate for prog = 2 is 0.64 times the incident rate for the reference group (prog = 1). Likewise, the incident rate for prog = 3 is 0.28 times the incident rate for the reference group holding the other variables constant. The percent change in the incident rate of daysabs is a 1% decrease for every unit increase in math.

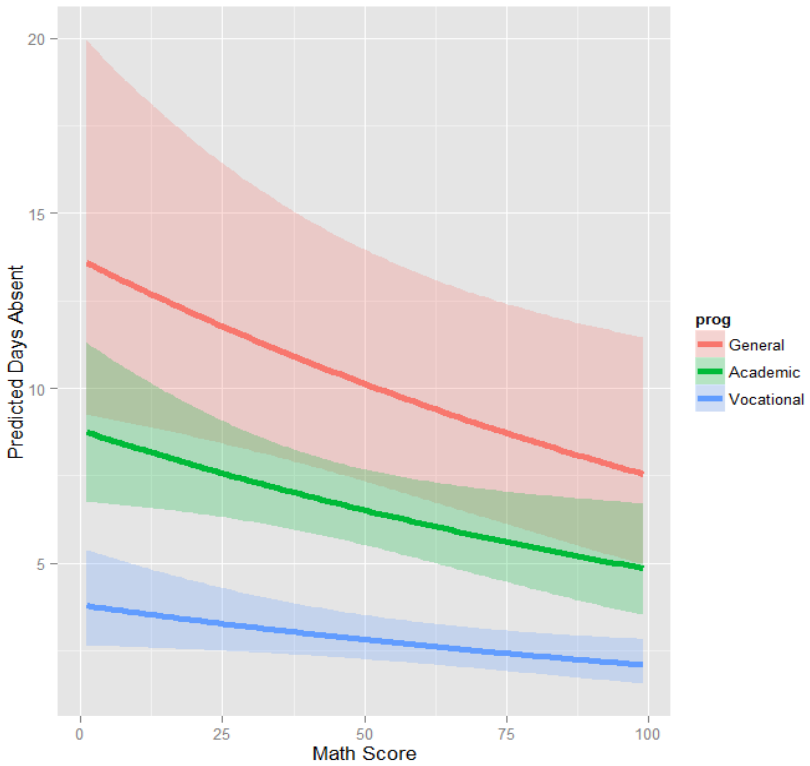
```
newdata1 <- data.frame(math = mean(dat$math), prog = factor(1:3, levels = 1:3,
   labels = levels(dat$prog)))
newdata1$phat <- predict(m1, newdata1, type = "response")
newdata1
```

```
##    math       prog   phat
## 1 48.27    General 10.237
## 2 48.27   Academic  6.588
## 3 48.27 Vocational  2.850
```

In the output above, we see that the predicted number of events (e.g., days absent) for a general program is about 10.24, holding math at its mean. The predicted number of

events for an academic program is lower at 6.59, and the predicted number of events for a vocational program is about 2.85.

```
newdata2 <- data.frame(
  math = rep(seq(from = min(dat$math), to = max(dat$math), length.out = 100), 3),
  prog = factor(rep(1:3, each = 100), levels = 1:3, labels =
  levels(dat$prog)))

newdata2 <- cbind(newdata2, predict(m1, newdata2, type = "link", se.fit=TRUE))
newdata2 <- within(newdata2, {
  DaysAbsent <- exp(fit)
  LL <- exp(fit - 1.96 * se.fit)
  UL <- exp(fit + 1.96 * se.fit)
})

ggplot(newdata2, aes(math, DaysAbsent)) +
  geom_ribbon(aes(ymin = LL, ymax = UL, fill = prog), alpha = .25) +
  geom_line(aes(colour = prog), size = 2) +
  labs(x = "Math Score", y = "Predicted Days Absent")
```



The graph shows the expected count across the range of math scores, for each type of program along with 95 percent confidence intervals. Note that the lines are not straight because this is a log linear model, and what is plotted are the expected values, not the log of the expected values.

Things to consider:

- It is not recommended that negative binomial models be applied to small samples.
- One common cause of over-dispersion is excess zeros by an additional data generating process. In this situation, zero-inflated model should be considered.
- If the data generating process does not allow for any 0s (such as the number of days spent in the hospital), then a zero-truncated model may be more appropriate.
- Count data often have an exposure variable, which indicates the number of times the event could have happened. This variable should be incorporated into your negative binomial regression model with the use of the offset option. See the glm documentation for details.
- The outcome variable in a negative binomial regression cannot have negative numbers.
- You will need to use the m1$resid command to obtain the residuals from our model to check other assumptions of the negative binomial model (see Cameron and Trivedi (1998) and Dupont (2002) for more information).

Question-04:

The state wildlife biologists want to model how many fish are being caught by fishermen at a state park. Visitors are asked how long they stayed, how many people were in the group, were there children in the group and how many fish were caught. Some visitors do not fish, but there is no data on whether a person fished or not. Some visitors who did fish did not catch any fish so there are excess zeros in the data because of the people that did not fish. The data set is taken from **("https://stats.idre.ucla.edu/stat/data/fish.csv")** and fit the Zero-Inflated Poisson regression generalized linear models (GLMs) to identify the factors associated with number of awards earned by students at one high school. Interpret the result.

Answer-04:

Let's look at the data.

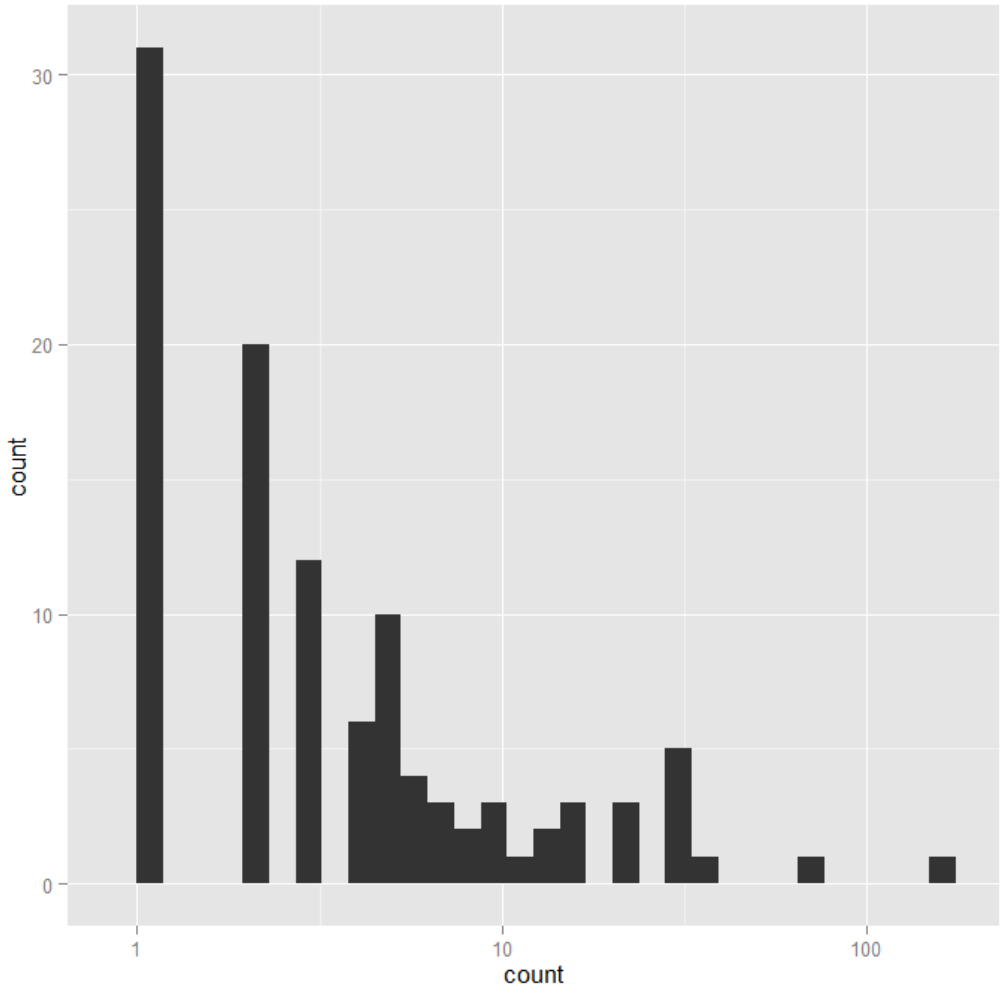
```
install.packages("pscl")
install.packages("boot")
update.packages("ggplot2")
require(ggplot2)
require(pscl)
```

```
require(boot)
```

```
zinb <- read.csv("https://stats.idre.ucla.edu/stat/data/fish.csv")
zinb <- within(zinb, {
   nofish <- factor(nofish)
   livebait <- factor(livebait)
   camper <- factor(camper)
})

summary(zinb)
```

```
##  nofish  livebait camper     persons         child             xb
##  0:176   0: 34    0:103   Min.   :1.00   Min.   :0.000   Min.   :-3.275
##  1: 74   1:216    1:147   1st Qu.:2.00   1st Qu.:0.000   1st Qu.: 0.008
##                           Median :2.00   Median :0.000   Median : 0.955
##                           Mean   :2.53   Mean   :0.684   Mean   : 0.974
##                           3rd Qu.:4.00   3rd Qu.:1.000   3rd Qu.: 1.964
##                           Max.   :4.00   Max.   :3.000   Max.   : 5.353
##       zg              count
##  Min.   :-5.626   Min.   :  0.0
##  1st Qu.:-1.253   1st Qu.:  0.0
##  Median : 0.605   Median :  0.0
##  Mean   : 0.252   Mean   :  3.3
##  3rd Qu.: 1.993   3rd Qu.:  2.0
##  Max.   : 4.263   Max.   :149.0
```

```
## histogram with x axis in log10 scale
ggplot(zinb, aes(count)) + geom_histogram() + scale_x_log10()
```



Analysis methods you might consider

Below is a list of some analysis methods you may have encountered. Some of the methods listed are quite reasonable while others have either fallen out of favor or have limitations.

- Zero-inflated Poisson Regression – The focus of this web page.
- Zero-inflated Negative Binomial Regression – Negative binomial regression does better with over dispersed data, i.e. variance much larger than the mean.
- Ordinary Count Models – Poisson or negative binomial models might be more appropriate if there are no excess zeros.
- OLS Regression – You could try to analyze these data using OLS regression. However, count data are highly non-normal and are not well estimated by OLS regression.

Zero-inflated Poisson regression:

Though we can run a Poisson regression in R using the glm function in one of the core packages, we need another package to run the zero-inflated Poisson model. We use the pscl package.

```
summary(m1 <- zeroinfl(count ~ child + camper | persons, data = zinb))
```

```
## Call:
## zeroinfl(formula = count ~ child + camper | persons, data = zinb)
##
## Pearson residuals:
##     Min      1Q  Median      3Q     Max
## -1.2369 -0.7540 -0.6080 -0.1921 24.0847
##
## Count model coefficients (poisson with log link):
##             Estimate Std. Error z value Pr(>|z|)
## (Intercept)  1.59789    0.08554  18.680   <2e-16 ***
## child       -1.04284    0.09999 -10.430   <2e-16 ***
## camper1      0.83402    0.09363   8.908   <2e-16 ***
##
## Zero-inflation model coefficients (binomial with logit link):
##             Estimate Std. Error z value Pr(>|z|)
## (Intercept)   1.2974     0.3739   3.470 0.000520 ***
## persons      -0.5643     0.1630  -3.463 0.000534 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Number of iterations in BFGS optimization: 12
## Log-likelihood: -1032 on 5 Df
```

The output looks very much like the output from two OLS regressions in R. Below the model call, you will find a block of output containing Poisson regression coefficients for each of the variables along with standard errors, z-scores, and p-values for the coefficients. A second block follows that corresponds to the inflation model. This includes logit coefficients for predicting excess zeros along with their standard errors, z-scores, and p-values.

All of the predictors in both the count and inflation portions of the model are statistically significant. This model fits the data significantly better than the null model, i.e., the intercept-only model. To show that this is the case, we can compare with the current model to a null model without predictors using chi-squared test on the difference of log likelihoods.

```
mnull <- update(m1, . ~ 1)


pchisq(2 * (logLik(m1) - logLik(mnull)), df = 3, lower.tail = FALSE)
```

```
## 'log Lik.' 4.041e-41 (df=5)
```

Since we have three predictor variables in the full model, the degrees of freedom for the chi-squared test is 3. This yields a high significant p-value; thus, our overall model is statistically significant.

```
dput(coef(m1, "count"))
```

```
## structure(c(1.59788828690411, -1.04283909332231, 0.834023618148891
## ), .Names = c("(Intercept)", "child", "camper1"))
```

```
dput(coef(m1, "zero"))
```

```
## structure(c(1.29744027908309, -0.564347365357873), .Names = c("(Intercept)",
## "persons"))
```

```
f <- function(data, i) {
  require(pscl)
  m <- zeroinfl(count ~ child + camper | persons, data = data[i, ],
    start = list(count = c(1.598, -1.0428, 0.834), zero = c(1.297, -0.564)))
  as.vector(t(do.call(rbind, coef(summary(m)))[, 1:2]))
}


set.seed(10)
res <- boot(zinb, f, R = 1200, parallel = "snow", ncpus = 4)


## print results
res
```

```
##
## ORDINARY NONPARAMETRIC BOOTSTRAP
##
##
## Call:
## boot(data = zinb, statistic = f, R = 1200, parallel = "snow",
```

```
##     ncpus = 4)
##
##
## Bootstrap Statistics :
##      original    bias    std. error
## t1*   1.59789 -0.056661     0.30307
## t2*   0.08554  0.004257     0.01670
## t3*  -1.04284 -0.002510     0.40557
## t4*   0.09999  0.004395     0.01539
## t5*   0.83402  0.017178     0.40465
## t6*   0.09363  0.004581     0.01536
## t7*   1.29744  0.020810     0.48058
## t8*   0.37385  0.008224     0.03662
## t9*  -0.56435 -0.030103     0.26673
## t10*  0.16296  0.005272     0.02981
```

The results are alternating parameter estimates and standard errors. That is, the first row has the first parameter estimate from our model. The second has the standard error for the first parameter. The third column contains the bootstrapped standard errors, which are considerably larger than those estimated by zeroinfl.

Now we can get the confidence intervals for all the parameters. We start on the original scale with percentile and bias adjusted CIs. We also compare these results with the regular confidence intervals based on the standard errors.

```
## basic parameter estimates with percentile and bias adjusted CIs
parms <- t(sapply(c(1, 3, 5, 7, 9), function(i) {
  out <- boot.ci(res, index = c(i, i + 1), type = c("perc", "bca"))
  with(out, c(Est = t0, pLL = percent[4], pUL = percent[5],
    bcaLL = bca[4], bcaLL = bca[5]))
}))


## add row names
row.names(parms) <- names(coef(m1))
## print results
parms
```

```
##                    Est     pLL      pUL     bcaLL    bcaLL
## count_(Intercept)  1.5979  0.8793  2.07810  1.087354  2.22614
## count_child       -1.0428 -1.7509 -0.17531 -1.618509 -0.02203
## count_camper1      0.8340  0.0596  1.62653  0.001571  1.59995
## zero_(Intercept)   1.2974  0.3503  2.21984  0.293577  2.12070
## zero_persons      -0.5643 -1.1087 -0.07847 -1.008526  0.00633
```

```
## compare with normal based approximation
confint(m1)
```

```
##                    2.5 %  97.5 %
## count_(Intercept)  1.4302  1.7655
## count_child       -1.2388 -0.8469
## count_camper1      0.6505  1.0175
## zero_(Intercept)   0.5647  2.0302
## zero_persons      -0.8838 -0.2449
```

The bootstrapped confidence intervals are considerably wider than the normal based approximation. The bootstrapped CIs are more consistent with the CIs from Stata when using robust standard errors.

Now we can estimate the incident risk ratio (IRR) for the Poisson model and odds ratio (OR) for the logistic (zero inflation) model. This is done using almost identical code as before, but passing a transformation function to the h argument of boot.ci, in this case, exp to exponentiate.

```
## exponentiated parameter estimates with percentile and bias adjusted CIs
expparms <- t(sapply(c(1, 3, 5, 7, 9), function(i) {
  out <- boot.ci(res, index = c(i, i + 1), type = c("perc", "bca"), h = exp)
  with(out, c(Est = t0, pLL = percent[4], pUL = percent[5],
    bcaLL = bca[4], bcaLL = bca[5]))
}))


## add row names
row.names(expparms) <- names(coef(m1))
```

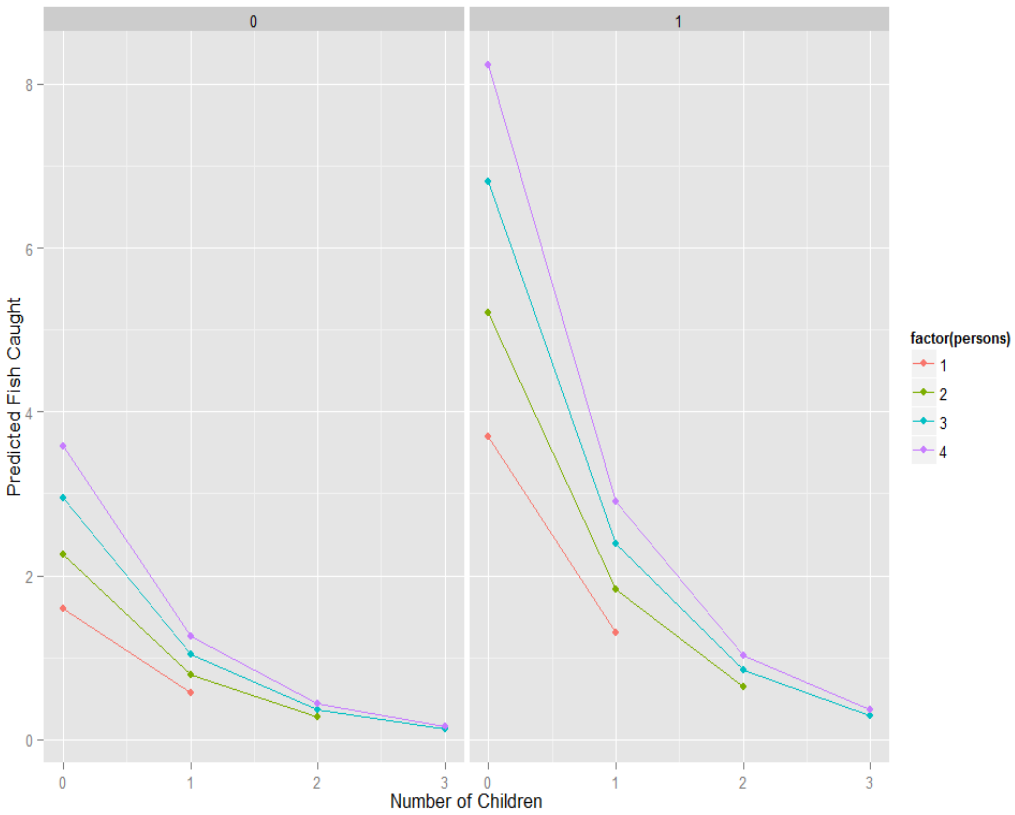
```
## print results
expparms
```

```
##                  Est    pLL    pUL  bcaLL  bcaLL
## count_(Intercept) 4.9426 2.4091 7.9892 2.9664 9.2641
## count_child       0.3525 0.1736 0.8392 0.1982 0.9782
## count_camper1     2.3026 1.0614 5.0862 1.0016 4.9528
## zero_(Intercept)  3.6599 1.4195 9.2058 1.3412 8.3370
## zero_persons      0.5687 0.3300 0.9245 0.3648 1.0063
```

To better understand our model, we can compute the expected number of fish caught for different combinations of our predictors. In fact, since we are working with essentially categorical predictors, we can compute the expected values for all combinations using the expand.grid function to create all combinations and then the predict function to do it. We also remove any rows where the number of children exceeds the number of persons, which does not make sense logically, using the subset function.

Finally we create a graph.

```
newdata1 <- expand.grid(0:3, factor(0:1), 1:4)
colnames(newdata1) <- c("child", "camper", "persons")
newdata1 <- subset(newdata1, subset=(child<=persons))
newdata1$phat <- predict(m1, newdata1)


ggplot(newdata1, aes(x = child, y = phat, colour = factor(persons))) +
  geom_point() +
  geom_line() +
  facet_wrap(~camper) +
  labs(x = "Number of Children", y = "Predicted Fish Caught")
```



Things to consider:

- Since zip has both a count model and a logit model, each of the two models should have good predictors. The two models do not necessarily need to use the same predictors.
- Problems of perfect prediction, separation or partial separation can occur in the logistic part of the zero-inflated model.
- Count data often use exposure variables to indicate the number of times the event could have happened. You can incorporate a logged version of the exposure variable into your model by using the offset() option.
- It is not recommended that zero-inflated Poisson models be applied to small samples. What constitutes a small sample does not seem to be clearly defined in the literature.
- Pseudo-R-squared values differ from OLS R-squareds, please see FAQ: What are pseudo R-squareds? for a discussion on this issue.
- In times past, the Vuong test had been used to test whether a zero-inflated Poisson model or a Poisson model (without the zero-inflation) was a better fit for the data. However, this test is no longer considered valid. Please see The Misuse of The Vuong Test For Non-Nested Models to Test for Zero-Inflation by Paul Wilson for further information.

Question-05:

The state wildlife biologists want to model how many fish are being caught by fishermen at a state park. Visitors are asked how long they stayed, how many people

were in the group, were there children in the group and how many fish were caught. Some visitors do not fish, but there is no data on whether a person fished or not. Some visitors who did fish did not catch any fish so there are excess zeros in the data because of the people that did not fish. The data set is taken from ("https://stats.idre.ucla.edu/stat/data/fish.csv") and fit the Zero-Inflated Binomial regression generalized linear models (GLMs) to identify the factors associated with number of awards earned by students at one high school. Interpret the result.

Answer-05:

Let's look at the data.

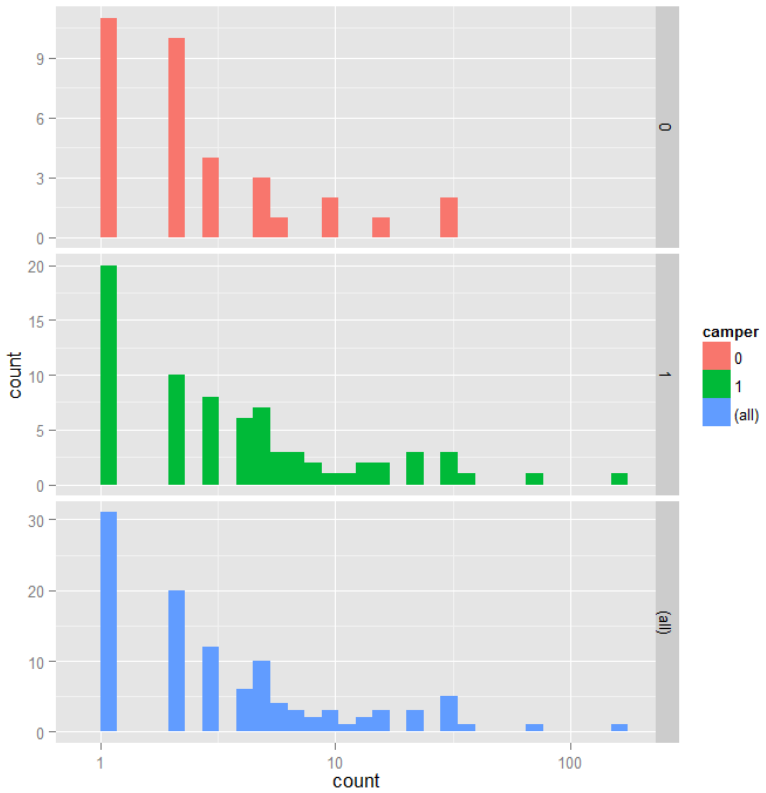
```
require(ggplot2)
require(pscl)
require(MASS)
require(boot)
```

```
zinb <- read.csv("https://stats.idre.ucla.edu/stat/data/fish.csv")
zinb <- within(zinb, {
  nofish <- factor(nofish)
  livebait <- factor(livebait)
  camper <- factor(camper)
})

summary(zinb)
##  nofish  livebait camper    persons        child            xb
##  0:176   0: 34    0:103   Min.   :1.00   Min.   :0.000   Min.   :-3.275
##  1: 74   1:216    1:147   1st Qu.:2.00   1st Qu.:0.000   1st Qu.: 0.008
##                           Median :2.00   Median :0.000   Median : 0.955
##                           Mean   :2.53   Mean   :0.684   Mean   : 0.974
##                           3rd Qu.:4.00   3rd Qu.:1.000   3rd Qu.: 1.964
##                           Max.   :4.00   Max.   :3.000   Max.   : 5.353
##       zg              count
##  Min.   :-5.626   Min.   :  0.0
##  1st Qu.:-1.253   1st Qu.:  0.0
##  Median : 0.605   Median :  0.0
##  Mean   : 0.252   Mean   :  3.3
```

```
##  3rd Qu.: 1.993   3rd Qu.:  2.0
##  Max.   : 4.263   Max.   :149.0
```

```
## histogram with x axis in log10 scale
ggplot(zinb, aes(count, fill = camper)) +
  geom_histogram() +
  scale_x_log10() +
  facet_grid(camper ~ ., margins=TRUE, scales="free_y")
```

```
## stat_bin: binwidth defaulted to range/30. Use 'binwidth = x' to adjust this.
## stat_bin: binwidth defaulted to range/30. Use 'binwidth = x' to adjust this.
## stat_bin: binwidth defaulted to range/30. Use 'binwidth = x' to adjust this.
```



Analysis methods you might consider:

Before we show how you can analyze this with a zero-inflated negative binomial analysis, let's consider some other methods that you might use.

- OLS Regression – You could try to analyze these data using OLS regression. However, count data are highly non-normal and are not well estimated by OLS regression.
- Zero-inflated Poisson Regression – Zero-inflated Poisson regression does better when the data are not over-dispersed, i.e. when variance is not much larger than the mean.
- Ordinary Count Models – Poisson or negative binomial models might be more appropriate if there are not excess zeros.

Zero-inflated negative binomial regression:

A zero-inflated model assumes that zero outcome is due to two different processes.

```
m1 <- zeroinfl(count ~ child + camper | persons,
  data = zinb, dist = "negbin")
summary(m1)
```

```
##
## Call:
## zeroinfl(formula = count ~ child + camper | persons, data = zinb,
##     dist = "negbin", EM = TRUE)
##
## Pearson residuals:
##    Min     1Q Median     3Q    Max
## -0.586 -0.462 -0.389 -0.197 18.013
##
## Count model coefficients (negbin with log link):
##             Estimate Std. Error z value Pr(>|z|)
## (Intercept)    1.371      0.256    5.35  8.6e-08 ***
## child         -1.515      0.196   -7.75  9.4e-15 ***
## camper1        0.879      0.269    3.26   0.0011 **
## Log(theta)    -0.985      0.176   -5.60  2.1e-08 ***
##
## Zero-inflation model coefficients (binomial with logit link):
##             Estimate Std. Error z value Pr(>|z|)
## (Intercept)    1.603      0.836    1.92    0.055 .
## persons       -1.666      0.679   -2.45    0.014 *
```

```
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Theta = 0.373
## Number of iterations in BFGS optimization: 2
## Log-likelihood: -433 on 6 Df
```

The output looks very much like the output from two OLS regressions in R.

- Below the model call, you will find a block of output containing negative binomial regression coefficients for each of the variables along with standard errors, z-scores, and p-values for the coefficients. A second block follows that corresponds to the inflation model. This includes logit coefficients for predicting excess zeros along with their standard errors, z-scores, and p-values.
- All of the predictors in both the count and inflation portions of the model are statistically significant. This model fits the data significantly better than the null model, i.e., the intercept-only model. To show that this is the case, we can compare with the current model to a null model without predictors using chi-squared test on the difference of log likelihoods.

```
m0 <- update(m1, . ~ 1)


pchisq(2 * (logLik(m1) - logLik(m0)), df = 3, lower.tail=FALSE)
```

```
## 'log Lik.' 1.28e-13 (df=6)
```

From the output above, we can see that our overall model is statistically significant.

- The predictors child and camper in the part of the negative binomial regression model predicting number of fish caught (count) are both significant predictors.
- The predictor person in the part of the logit model predicting excessive zeros is statistically significant.
- For these data, the expected change in log(count) for a one-unit increase in child is -1.515255 holding other variables constant.
- A camper (camper = 1) has an expected log(count) of 0.879051 higher than that of a non-camper (camper = 0) holding other variables constant.
- The log odds of being an excessive zero would decrease by 1.67 for every additional person in the group. In other words, the more people in the group the less likely that the zero would be due to not gone fishing. Put plainly, the larger the group the person was in, the more likely that the person went fishing.

```
dput(round(coef(m1, "count"), 4))
```

```
## structure(c(1.3711, -1.5152, 0.879), .Names = c("(Intercept)",
## "child", "camper1"))
```

```
dput(round(coef(m1, "zero"), 4))
```

```
## structure(c(1.6028, -1.6663), .Names = c("(Intercept)", "persons"
## ))
```

```
f <- function(data, i) {
  require(pscl)
  m <- zeroinfl(count ~ child + camper | persons,
    data = data[i, ], dist = "negbin",
    start = list(count = c(1.3711, -1.5152, 0.879), zero = c(1.6028, -1.6663)))
  as.vector(t(do.call(rbind,  coef(summary(m)))[,  1:2]))
}

set.seed(10)
(res <- boot(zinb, f, R = 1200, parallel = "snow", ncpus = 4))
```

```
##
## ORDINARY NONPARAMETRIC BOOTSTRAP
##
##
## Call:
## boot(data = zinb, statistic = f, R = 1200, parallel = "snow",
##     ncpus = 4)
##
##
## Bootstrap Statistics :
##      original     bias    std. error
## t1*    1.3711 -0.083023     0.39403
```

```
## t2*    0.2561 -0.002622     0.03191
## t3*   -1.5153 -0.061487     0.26892
## t4*    0.1956  0.006034     0.02027
## t5*    0.8791  0.091431     0.47124
## t6*    0.2693  0.001873     0.01998
## t7*   -0.9854  0.080120     0.21896
## t8*    0.1760  0.002577     0.01689
## t9*    1.6031  0.473597     1.59331
## t10*   0.8365  3.767327    15.65780
## t11*  -1.6666 -0.462364     1.56789
## t12*   0.6793  3.771994    15.69675
```

The results are alternating parameter estimates and standard errors. That is, the first row has the first parameter estimate from our model. The second has the standard error for the first parameter. The third column contains the bootstrapped standard errors, which are considerably larger than those estimated by zeroinfl.

Now we can get the confidence intervals for all the parameters. We start on the original scale with percentile and bias adjusted CIs. We also compare these results with the regular confidence intervals based on the standard errors.

```
## basic parameter estimates with percentile and bias adjusted CIs
parms <- t(sapply(c(1, 3, 5, 9, 11), function(i) {
  out <- boot.ci(res, index = c(i, i + 1), type = c("perc", "bca"))
  with(out, c(Est = t0, pLL = percent[4], pUL = percent[5],
    bcaLL = bca[4], bcaUL = bca[5]))
}))


## add row names
row.names(parms) <- names(coef(m1))
## print results
parms
```

```
##                  Est     pLL     pUL   bcaLL   bcaUL
## count_(Intercept)  1.3711  0.5676  2.0620  0.7226  2.2923
## count_child       -1.5153 -2.1382 -1.0887 -2.0175 -0.9593
```

```
## count_camper1      0.8791  0.0431  1.8331 -0.2016  1.6669
## zero_(Intercept)   1.6031  0.4344  8.2380  0.0282  3.5197
## zero_persons      -1.6666 -8.5436 -1.1002 -7.8329 -1.0781
```

```
## compare with normal based approximation
confint(m1)
```

```
##                    2.5 %  97.5 %
## count_(Intercept)  0.86911  1.8731
## count_child       -1.89860 -1.1319
## count_camper1      0.35127  1.4068
## zero_(Intercept)  -0.03636  3.2419
## zero_persons      -2.99701 -0.3355
```

The bootstrapped confidence intervals are considerably wider than the normal based approximation. The bootstrapped CIs are more consistent with the CIs from Stata when using robust standard errors.

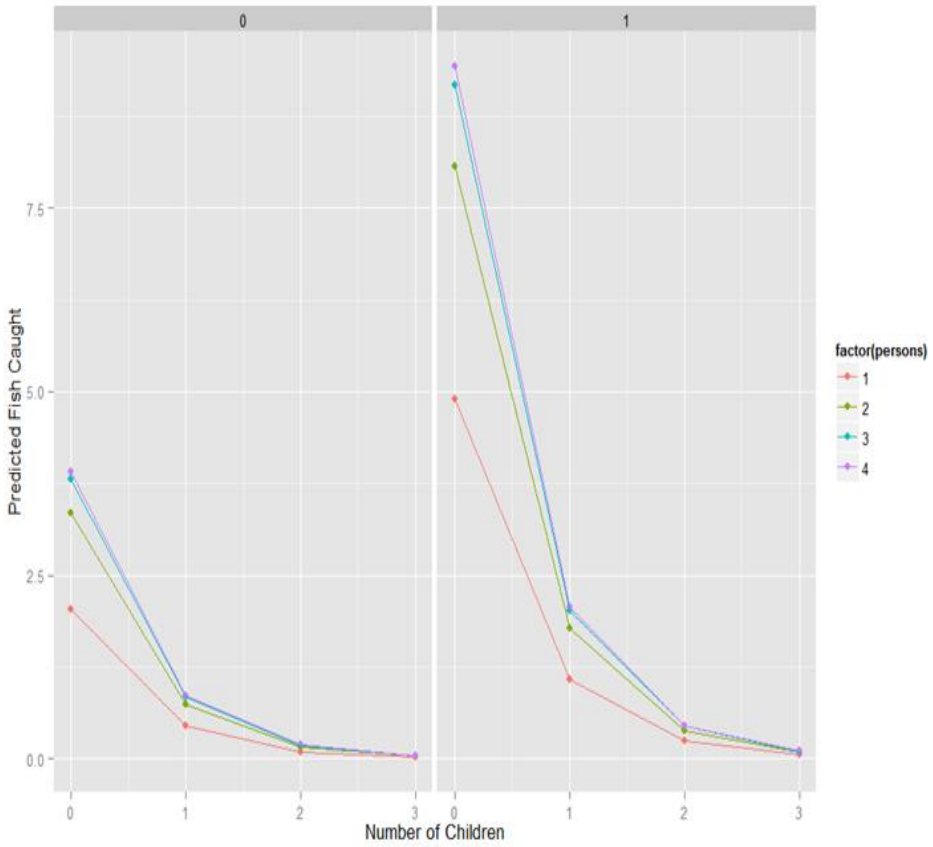
```
## exponentiated parameter estimates with percentile and bias adjusted CIs
expparms <- t(sapply(c(1, 3, 5, 7, 9), function(i) {
  out <- boot.ci(res, index = c(i, i + 1), type = c("perc", "bca"), h = exp)
  with(out, c(Est = t0, pLL = percent[4], pUL = percent[5],
    bcaLL = bca[4], bcaUL = bca[5]))
}))


## add row names
row.names(expparms) <- names(coef(m1))
## print results
expparms
```

```
##                    Est    pLL      pUL  bcaLL   bcaUL
## count_(Intercept) 3.9395 1.7641   7.8615 2.0599  9.8981
## count_child       0.2198 0.1179   0.3367 0.1330  0.3832
```

```
## count_camper1     2.4086 1.0441    6.2534 0.8175  5.2958
## zero_(Intercept)  4.9686 1.5441 3781.9642 1.0286 33.7757
## zero_persons      0.1889 0.0002    0.3328 0.0004  0.3402
```

To better understand our model, we can compute the expected number of fish caught for different combinations of our predictors. In fact, since we are working with essentially categorical predictors, we can compute the expected values for all combinations using the expand.grid function to create all combinations and then the predict function to do it. Finally we create a graph.

```
newdata1 <- expand.grid(0:3, factor(0:1), 1:4)
colnames(newdata1) <- c("child", "camper", "persons")
newdata1$phat <- predict(m1, newdata1)


ggplot(newdata1, aes(x = child, y = phat, colour = factor(persons))) +
 geom_point() +
 geom_line() +
 facet_wrap(~camper) +
 labs(x = "Number of Children", y = "Predicted Fish Caught")
```



Things to consider:

- Here are some issues that you may want to consider in the course of your research analysis.

- Question about the over-dispersion parameter is in general a tricky one. A large over-dispersion parameter could be due to a miss-specified model or could be due to a real process with over-dispersion. Adding an over-dispersion problem does not necessarily improve a miss-specified model.
- The zero inflated negative binomial model has two parts, a negative binomial count model and the logit model for predicting excess zeros, so you might want to review these Data Analysis Example pages, Negative Binomial Regression and Logit Regression.
- Since zero inflated negative binomial has both a count model and a logit model, each of the two models should have good predictors. The two models do not necessarily need to use the same predictors.
- Problems of perfect prediction, separation or partial separation can occur in the logistic part of the zero-inflated model.
- Count data often use exposure variable to indicate the number of times the event could have happened. You can incorporate exposure (also called an offset) into your model by using the offset() function.
- It is not recommended that zero-inflated negative binomial models be applied to small samples. What constitutes a small sample does not seem to be clearly defined in the literature.
- Pseudo-R-squared values differ from OLS R-squareds, please see FAQ: What are pseudo R-squareds? for a discussion on this issue.
- In times past, the Vuong test had been used to test whether a zero-inflated negative binomial model or a negative binomial model (without the zero-inflation) was a better fit for the data. However, this test is no longer considered valid. Please see The Misuse of The Vuong Test For Non-Nested Models to Test for Zero-Inflation by Paul Wilson for further information.

Question-06:

A study of length of hospital stay, in days, as a function of age, kind of health insurance and whether or not the patient died while in the hospital. Length of hospital stay is recorded as a minimum of at least one day.. The data set is taken from ("**https://stats.idre.ucla.edu/stat/data/ztp.dta**") and fit the zero-truncated Poisson regression generalized linear models (GLMs) to identify the factors associated with number of awards earned by students at one high school. Interpret the result.

Answer-06:

```
require(foreign)
require(ggplot2)
```

```
require(VGAM)
require(boot)
```

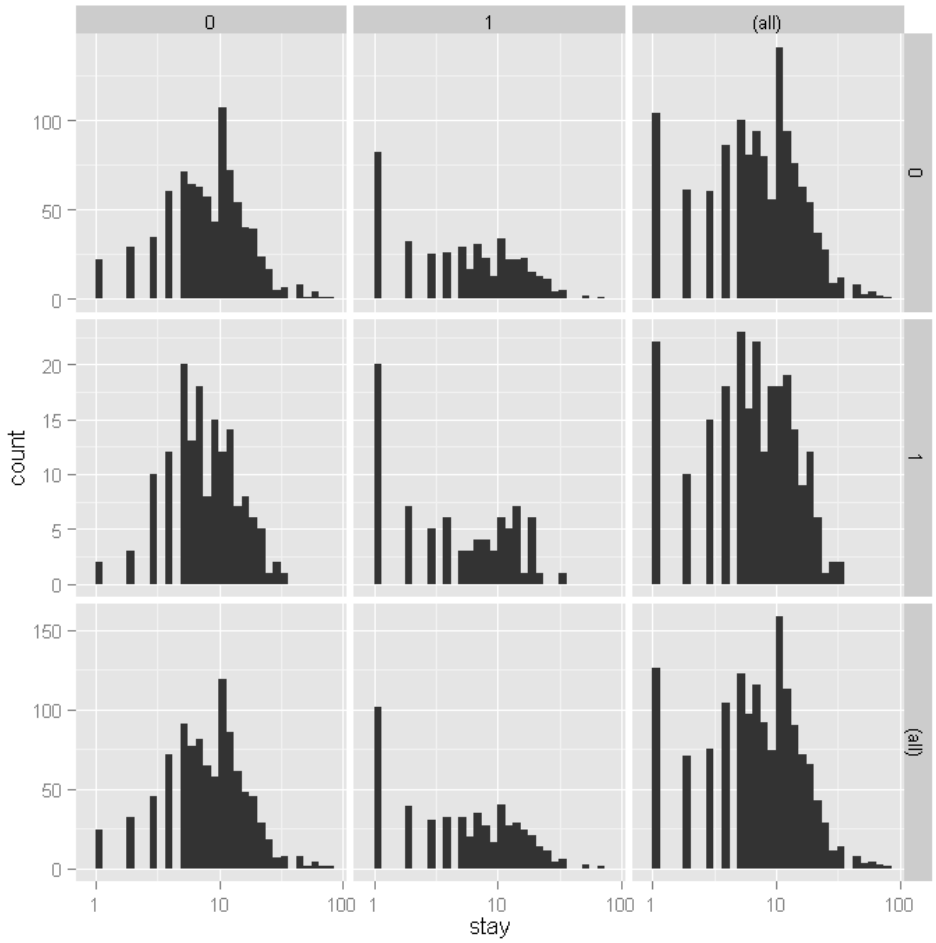
Let's look at the data. We import the Stata dataset using the foreign package.

```
dat <- read.dta("https://stats.idre.ucla.edu/stat/data/ztp.dta")


dat <- within(dat, {
   hmo <- factor(hmo)
   died <- factor(died)
})


summary(dat)
```

```
##      stay           age       hmo       died
## Min.   : 1.00   Min.   :1.00   0:1254   0:981
## 1st Qu.: 4.00   1st Qu.:4.00   1: 239   1:512
## Median : 8.00   Median :5.00
## Mean   : 9.73   Mean   :5.23
## 3rd Qu.:13.00   3rd Qu.:6.00
## Max.   :74.00   Max.   :9.00
```

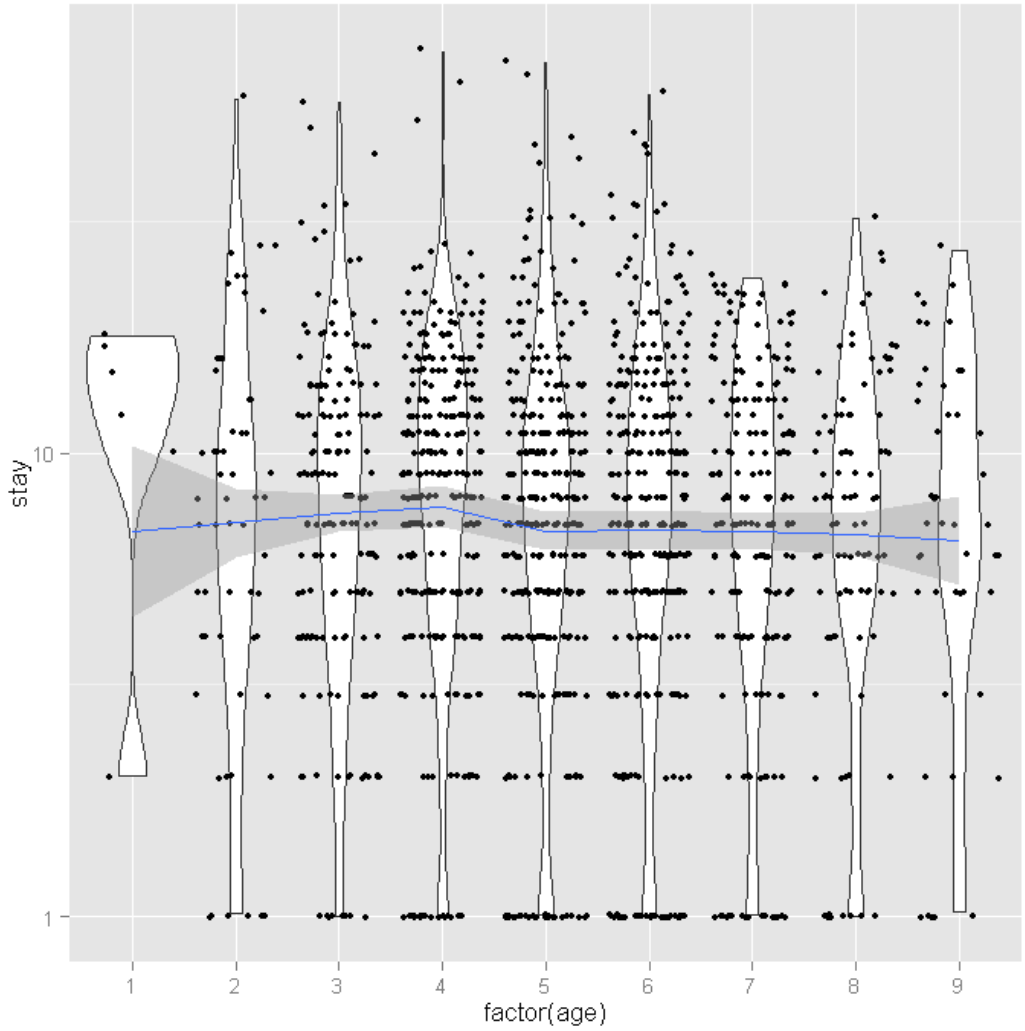
We will use the ggplot2 package. First we can look at histograms of stay broken down by hmo on the rows and died on the columns. We also include the marginal distributions, thus the lower right corner represents the overall histogram. We use a log base 10 scale to approximate the canonical link function of the poisson distribution (natural logarithm).

```
ggplot(dat, aes(stay)) +
 geom_histogram() +
 scale_x_log10() +
 facet_grid(hmo ~ died, margins=TRUE, scales="free_y")
```



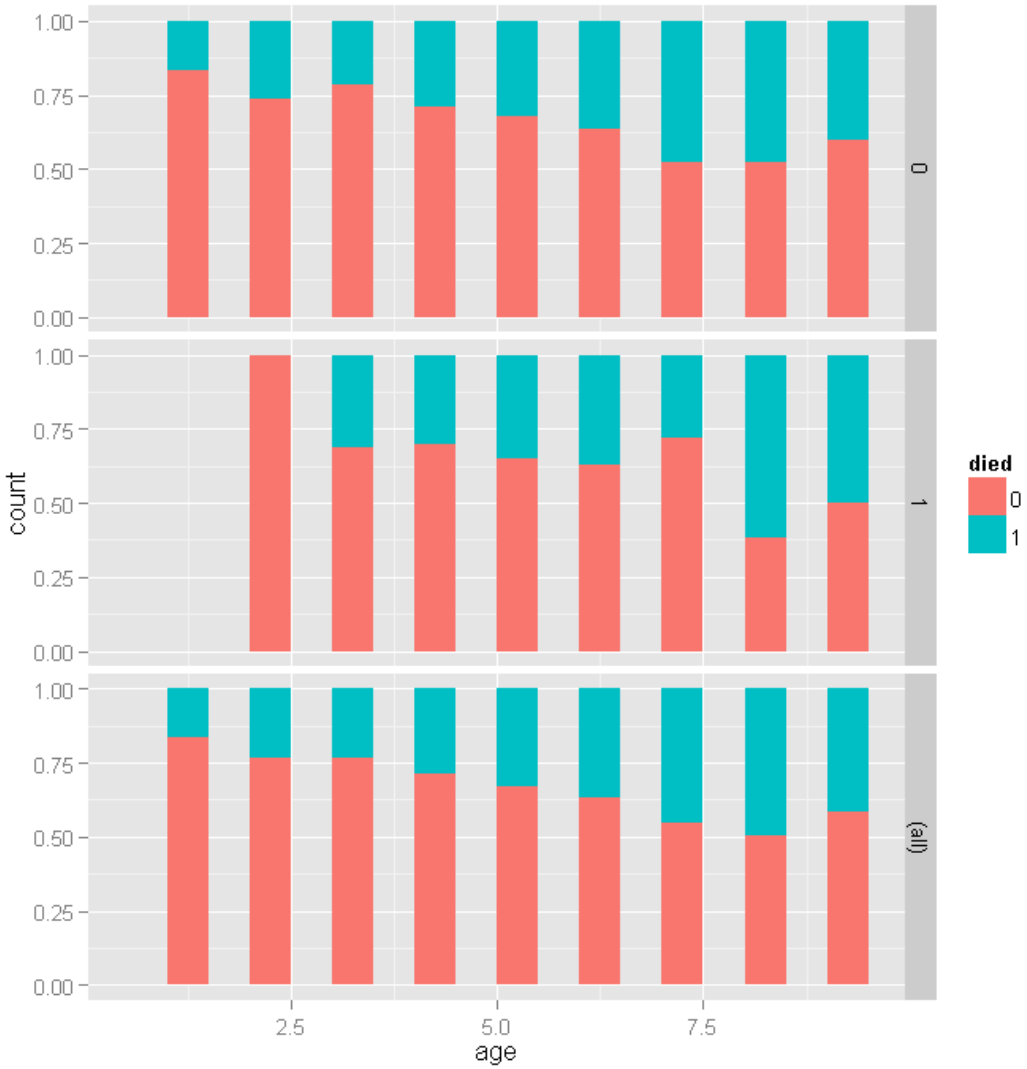
From the histograms, it looks like the density of the distribution, does vary across levels of hmo and died, with shorter stays for those in HMOs (1) and shorter for those who did die, including what seems to be an inflated number of 1 day stays. To examine how stay varies across age groups, we can use conditional violin plots which show a kernel density estimate of the distribution of stay mirrored (hence the violin) and conditional on each age group. To further understand the raw data going into each density estimate, we add raw data on top of the violin plots with a small amount of random noise (jitter) to alleviate over plotting. Finally, to get a sense of the overall trend, we add a locally weighted regression line.

```
ggplot(dat, aes(factor(age), stay)) +
  geom_violin() +
  geom_jitter(size=1.5)  +
  scale_y_log10() +
  stat_smooth(aes(x  = age, y = stay, group=1), method="loess")
```



The distribution of length of stay does not seem to vary much across age groups. This observation from the raw data is corroborated by the relatively flat loess line. Finally let's look at the proportion of people who lived or died across age groups by whether or not they are in HMOs.

```
ggplot(dat, aes(age, fill=died)) +
  geom_histogram(binwidth=.5, position="fill") +
  facet_grid(hmo ~ ., margins=TRUE)
```



For the lowest ages, a smaller proportion of people in HMOs died, but for higher ages, there does not seem to be a huge difference, with a slightly higher proportion in HMOs dying if anything. Overall, as age group increases, the proportion of those dying increases, as expected.

Analysis methods you might consider:

Below is a list of some analysis methods you may have encountered. Some of the methods listed are quite reasonable while others have either fallen out of favor or have limitations.

- Zero-truncated Poisson Regression – The focus of this web page.
- Zero-truncated Negative Binomial Regression – If you have overdispersion in addition to zero truncation. See the Data Analysis Example for ztnb.
- Poisson Regression – Ordinary Poisson regression will have difficulty with zero-truncated data. It will try to predict zero counts even though there are no zero values.

- Negative Binomial Regression – Ordinary Negative Binomial regression will have difficulty with zero-truncated data. It will try to predict zero counts even though there are no zero values.
- OLS Regression – You could try to analyze these data using OLS regression. However, count data are highly non-normal and are not well estimated by OLS regression.

Zero-truncated Poisson regression:

To fit the zero-truncated poisson model, we use the vglm function in the VGAM package. This function fits a very flexible class of models called vector generalized linear models to a wide range of assumed distributions. In our case, we believe the data are poisson, but without zeros. Thus the values are strictly positive poisson, for which we use the positive poisson family via the pospoisson function passed to vglm.

```
m1 <- vglm(stay ~ age + hmo + died, family = pospoisson(), data = dat)
summary(m1)
```

```
##
## Call:
## vglm(formula = stay ~ age + hmo + died, family = pospoisson(),
##     data = dat)
##
## Pearson Residuals:
##                 Min  1Q Median  3Q Max
## log(lambda)  -3 -1.7  -0.59 0.98  21
##
## Coefficients:
##             Estimate Std. Error z value
## (Intercept)    2.436      0.027    89.1
## age           -0.014      0.005    -2.9
## hmo1          -0.136      0.024    -5.7
## died1         -0.204      0.018   -11.1
##
## Number of linear predictors:  1
##
## Name of linear predictor: log(lambda)
```

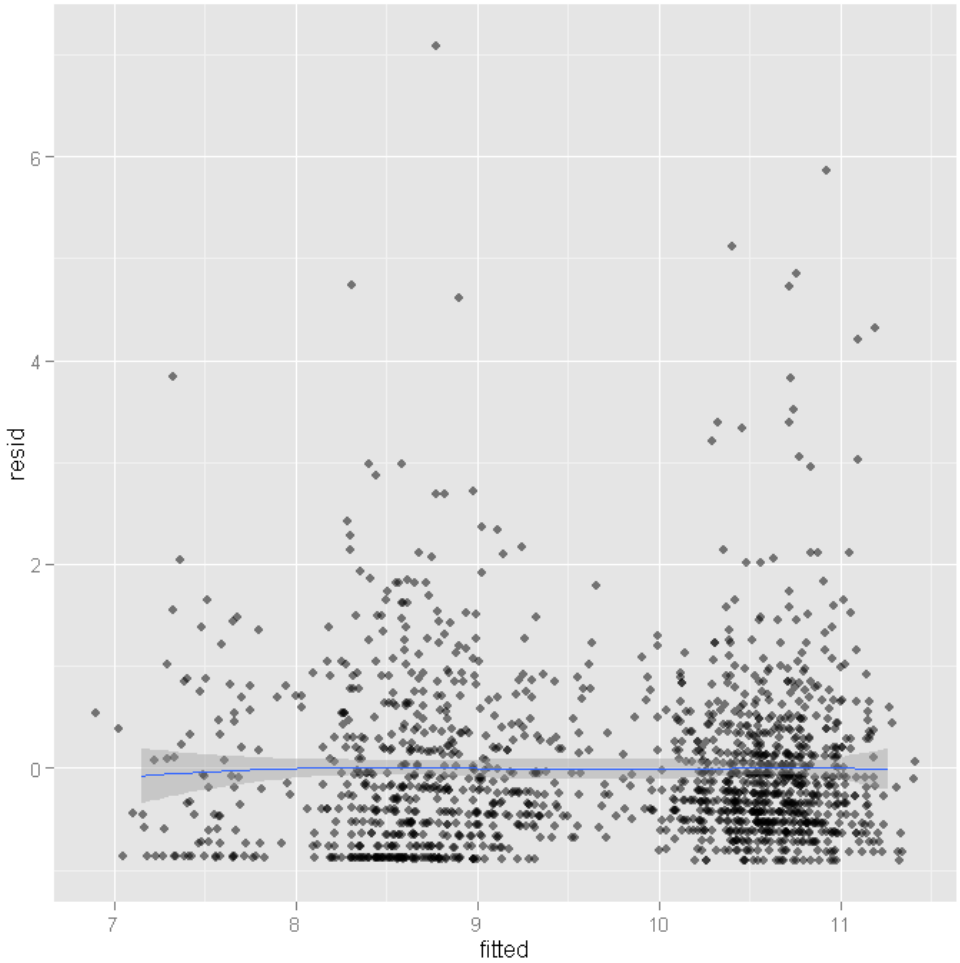
```
##
## Dispersion Parameter for pospoisson family:   1
##
## Log-likelihood: -6909 on 1489 degrees of freedom
##
## Number of iterations: 4
```

The output looks very much like the output from an OLS regression:

- It begins by echoing the function call showing us what we modeled.
- Next comes the spread of the residuals, there is at least one high residual because the max is much higher than the min.
- Following the residuals are the parameter estimates and standard errors. The z values (frac{Estimate}{SE}) are also printed. No p values are given, although if one wished to assume that the estimates followed the normal distribution, one could easily compute the probability of obtaining that z value.
- The number of linear predictors is printed because vglm can fit far more complex models, for this page, this will always be one, the expected mean, (lambda).
- Next the dispersion parameter is printed, for the Poisson, this is assumed to be 1 and is not estimated.
- Finally the log likelihood -6908.7991 is printed along with the number of iterations needed to reach convergence.

Now let's look at a plot of the residuals versus fitted values. We add random horizontal noise as well as 50 percent transparency to alleviate over plotting and better see where most residuals fall.

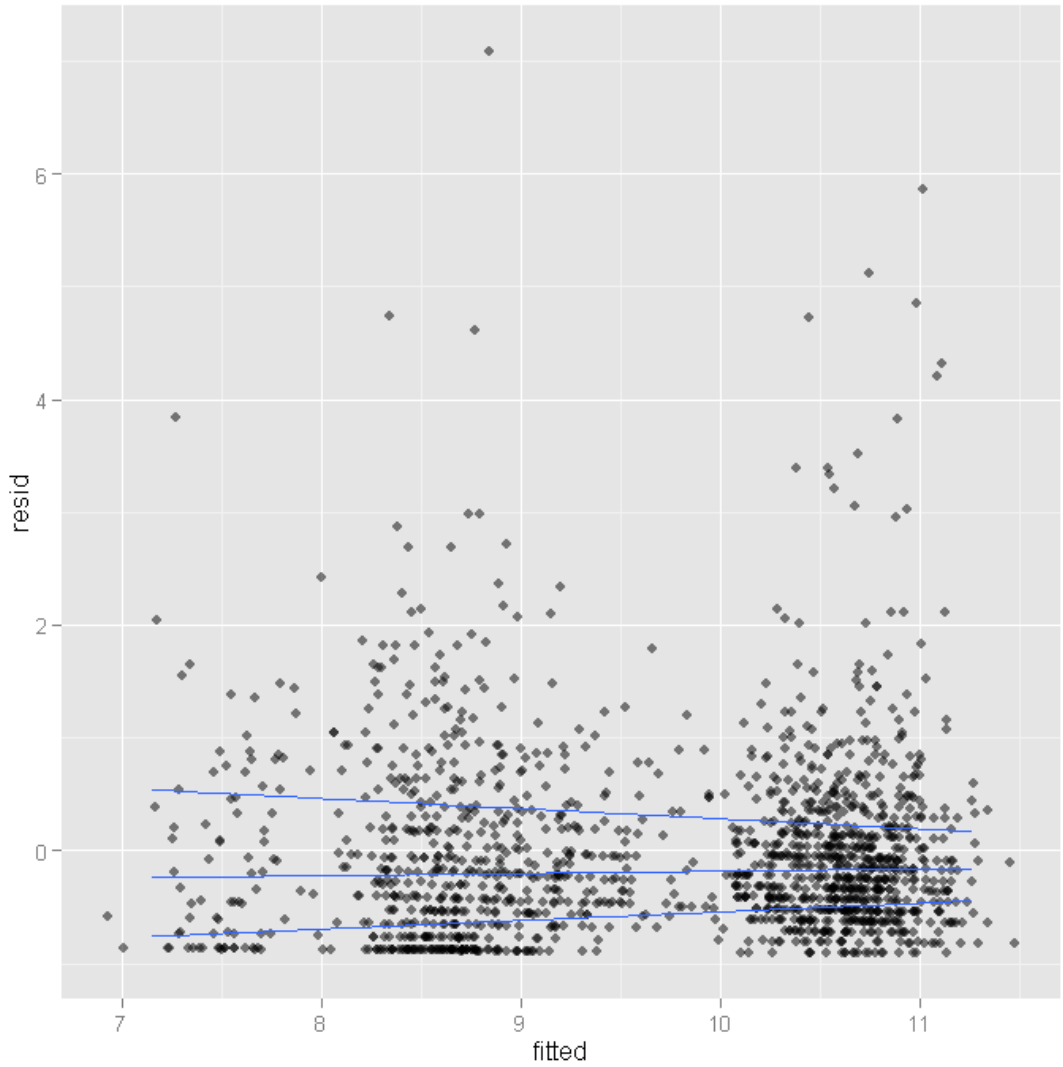
```
output <- data.frame(resid = resid(m1), fitted = fitted(m1))
ggplot(output, aes(fitted, resid)) +
  geom_jitter(position=position_jitter(width=.25), alpha=.5) +
  stat_smooth(method="loess")
```



The mean is around zero across all the fitted levels it looks like. However, there are some values that look rather extreme. To see if these have much influence, we can fit lines using quantile regression, these lines represent the 75th, 50th, and 25th percentiles.

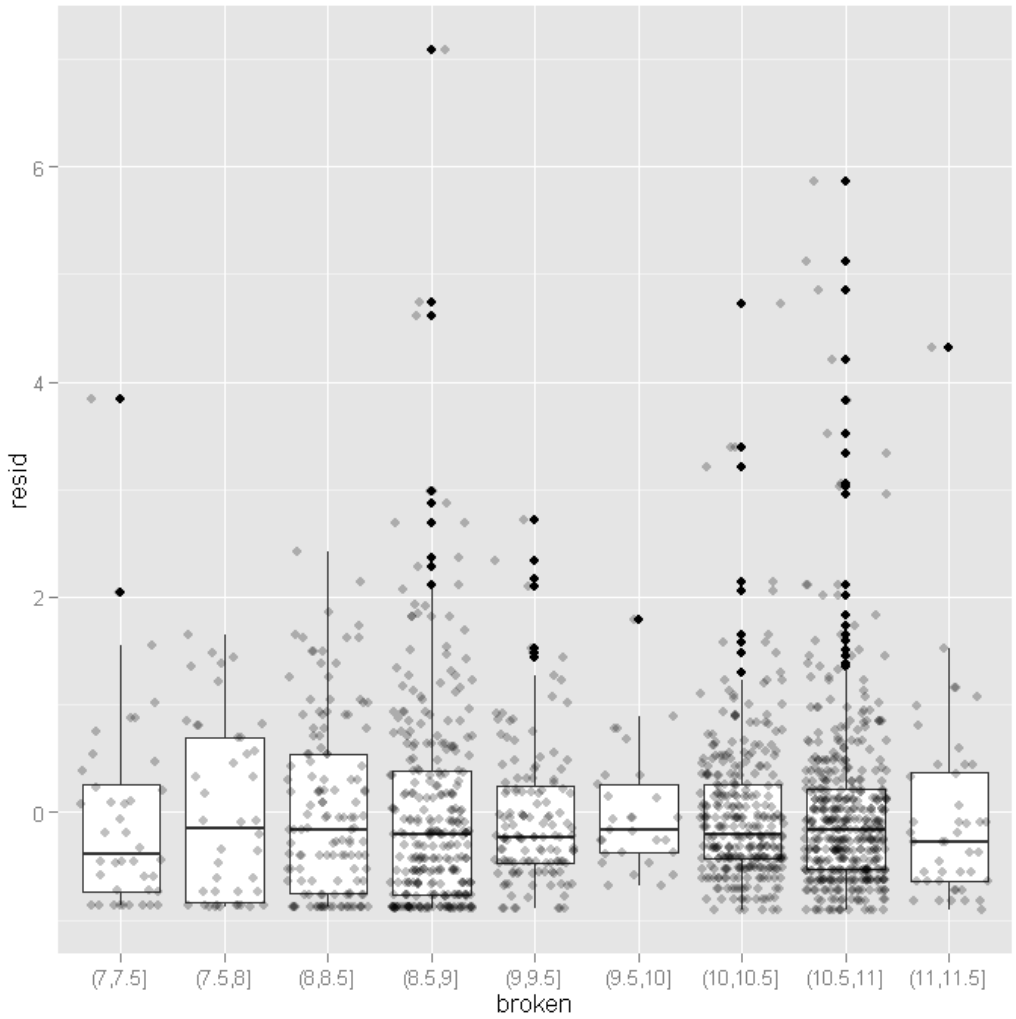
```
ggplot(output, aes(fitted, resid)) +
  geom_jitter(position=position_jitter(width=.25),  alpha=.5) +
  stat_quantile(method="rq")
```

```
## Smoothing formula not specified. Using: y ~ x
```



Here we see the spread narrowing at higher levels. Let's cut the data into intervals and check box plots for each. We will get the breaks from the algorithm for a histogram.

```
output <- within(output, {
  broken <- cut(fitted, hist(fitted, plot=FALSE)$breaks)
})


ggplot(output, aes(broken, resid)) +
 geom_boxplot() +
 geom_jitter(alpha=.25)
```



Now that feel a little more confident the model fits okay, let's look at the coefficients.

- The value of the coefficient for age, -0.0144 suggests that the log count of stay decreases by 0.0144 for each year increase in age.
- The coefficient for hmo, -0.1359 indicates that the log count of stay for HMO patient is 0.1359 less than for non-HMO patients.
- The log count of stay for patients who died while in the hospital was 0.2038 less than those patients who did not die.
- Finally, the value of the constant 2.4358 is the log count of the stay when all of the predictors equal zero.

We can get confidence intervals for the parameters and the exponentiated parameters using bootstrapping. For the Poisson model, these would be incident risk ratios. We use the boot package. First, we get the coefficients from our original model to use as start values for the model to speed up the time it takes to estimate. Then we write a short function that takes data and indices as input and returns the parameters we are interested in. Finally, we pass that to the boot function and do 1200 replicates, using snow to

distribute across four cores. Note that you should adjust the number of cores to whatever your machine has. Also, for final results, one may wish to increase the number of replications to help ensure stable results.

```
dput(round(coef(m1),3))
```

```
## structure(c(2.436, -0.014, -0.136, -0.204), .Names = c("(Intercept)",
## "age", "hmo1", "died1"))
```

```
f <- function(data, i) {
  require(VGAM)
  m <- vglm(formula = stay ~ age + hmo + died, family = pospoisson(),
    data = data[i, ], coefstart = c(2.436, -0.014, -0.136, -0.204))
  as.vector(t(coef(summary(m))[, 1:2]))
}

set.seed(10)
res <- boot(dat, f, R = 1200, parallel = "snow", ncpus = 4)

## print results
res
##
## ORDINARY NONPARAMETRIC BOOTSTRAP
##
##
## Call:
## boot(data = dat, statistic = f, R = 1200, parallel = "snow",
##     ncpus = 4)
##
##
## Bootstrap Statistics :
##     original     bias    std. error
## t1* 2.435808  2.124e-04   6.962e-02
```

```
## t2*  0.027332  8.324e-06   5.621e-04
## t3* -0.014442 -5.729e-05   1.212e-02
## t4*  0.005035  2.490e-06   9.896e-05
## t5* -0.135903  1.162e-03   5.105e-02
## t6*  0.023742  1.313e-05   7.497e-04
## t7* -0.203771 -1.487e-03   4.984e-02
## t8*  0.018373  3.854e-05   3.556e-04
```

The results are alternating parameter estimates and standard errors. That is, the first row has the first parameter estimate from our model. The second has the standard error for the first parameter. The third column contains the bootstrapped standard errors.

Now we can get the confidence intervals for all the parameters. We start on the original scale with percentile and basic bootstrap CIs.

```
## basic parameter estimates with percentile and bias adjusted CIs
parms <- t(sapply(c(1, 3, 5, 7), function(i) {
  out <- boot.ci(res, index = c(i, i + 1), type = c("perc", "basic"))
  with(out, c(Est = t0, pLL = percent[4], pUL = percent[5],
    basicLL = basic[4], basicLL = basic[5]))
}))

## add row names
row.names(parms) <- names(coef(m1))
## print results
parms
```

```
##                Est      pLL       pUL  basicLL  basicLL
## (Intercept)  2.43581  2.29933  2.577805  2.29381  2.57228
## age         -0.01444 -0.04003  0.009798 -0.03868  0.01114
## hmo1        -0.13590 -0.23875 -0.038074 -0.23373 -0.03306
## died1       -0.20377 -0.30530 -0.106277 -0.30126 -0.10224
```

The bootstrapped confidence intervals are wider than would be expected using a normal based approximation. The bootstrapped CIs are more consistent with the CIs from Stata when using robust standard errors.

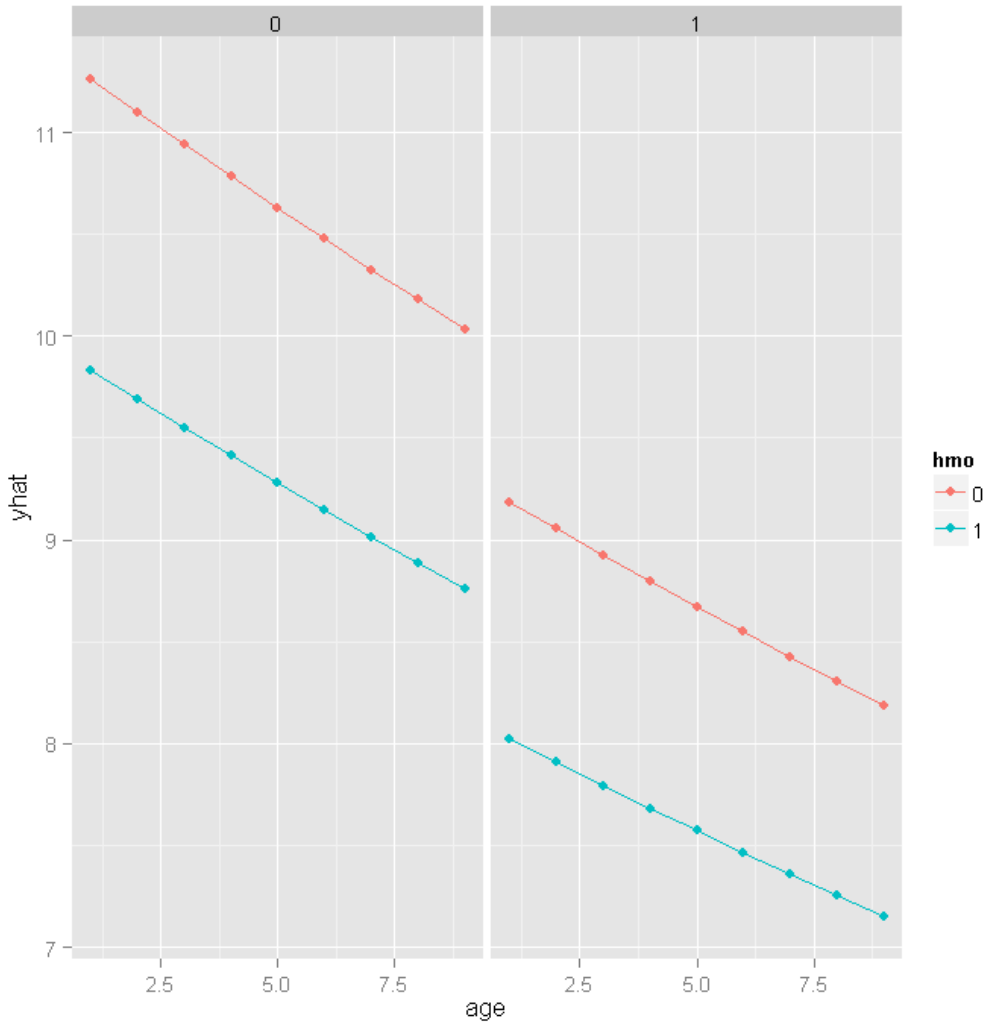
Now we can estimate the incident risk ratio (IRR) for the Poisson model. This is done using almost identical code as before, but passing a transformation function to the h argument of boot.ci, in this case, exp to exponentiate.

```
## exponentiated parameter estimates with percentile and bias adjusted CIs
expparms <- t(sapply(c(1, 3, 5, 7), function(i) {
  out <- boot.ci(res, index = c(i, i + 1), type = c("perc", "basic"), h = exp)
  with(out, c(Est = t0, pLL = percent[4], pUL = percent[5],
    basicLL = basic[4], basicLL = basic[5]))
}))

## add row names
row.names(expparms) <- names(coef(m1))
## print results
expparms
```

```
##                Est    pLL     pUL basicLL basicLL
## (Intercept) 11.4250 9.9675 13.1682  9.6819 12.8826
## age          0.9857 0.9608  1.0098  0.9615  1.0106
## hmo1         0.8729 0.7876  0.9626  0.7832  0.9582
## died1        0.8156 0.7369  0.8992  0.7321  0.8944
```

The results are consistent with what we initially viewed graphically, age does not have a significant effect, but hmo and died both do. In order to better understand our results and model, let's plot some predicted values. Because all of our predictors were categorical (hmo and died) or had a small number of unique values (age) we will get predicted values for all possible combinations. First we create a new data set using the expand.grid function, then estimate the predicted values using the predict function, and finally plot them.

```
newdata <- expand.grid(age = 1:9, hmo = factor(0:1), died = factor(0:1))
newdata$yhat <- predict(m1, newdata, type = "response")


ggplot(newdata, aes(x = age, y = yhat, colour = hmo)) +
  geom_point() +
  geom_line() +
  facet_wrap(~ died)
```

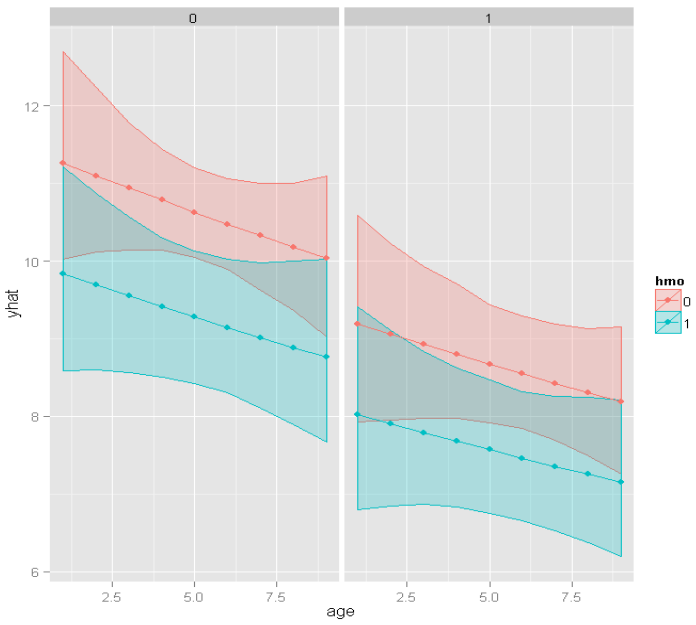


If we really wanted to compare the predicted values, we could bootstrap confidence intervals around the predicted estimates. These confidence intervals are not for the predicted value themselves, but that that is the mean predicted value (i.e., for the estimate, not a new individual). If we wanted to be efficient, we could have done this with our prior bootstrap so we only fit the models once. However, it is fast enough we just rerun the bootstrap rather than combine them.

```
## function to return predicted values
fpred <- function(data, i, newdata) {
  require(VGAM)
  m <- vglm(formula = stay ~ age + hmo + died, family = pospoisson(),
    data = data[i, ], coefstart = c(2.436, -0.014, -0.136, -0.204))
  predict(m, newdata, type = "response")
}
```

```
## set seed and run bootstrap with 1,200 draws
set.seed(10)
respred <- boot(dat, fpred, R = 1200, newdata = newdata,
  parallel = "snow", ncpus = 4)

## get the bootstrapped percentile CIs
yhat <- t(sapply(1:nrow(newdata), function(i) {
  out <- boot.ci(respred, index = i, type = c("perc"))
  with(out, c(Est = t0, pLL = percent[4], pUL = percent[5]))
}))

## merge CIs with predicted values
newdata <- cbind(newdata, yhat)
```

```
## graph with CIs
ggplot(newdata, aes(x = age, y = yhat, colour = hmo, fill = hmo))  +
  geom_ribbon(aes(ymin  = pLL, ymax = pUL), alpha = .25) +
  geom_point() +
  geom_line() +
  facet_wrap(~ died)
```



Things to consider:

- Count data often use exposure variable to indicate the number of times the event could have happened. You can incorporate exposure into your model by using the exposure() option.
- It is not recommended that zero-truncated poisson models be applied to small samples. What constitutes a small sample does not seem to be clearly defined in the literature.
- Pseudo-R-squared values differ from OLS R-squareds, please see FAQ: What are pseudo R-squareds? for a discussion on this issue.

Question-07:

A study of length of hospital stay, in days, as a function of age, kind of health insurance and whether or not the patient died while in the hospital. Length of hospital stay is recorded as a minimum of at least one day. The data set is taken from ("**https://stats.idre.ucla.edu/stat/data/ztp.dta**") and fit the zero-truncated negative binomial regression generalized linear models (GLMs) to identify the factors associated with number of awards earned by students at one high school. Interpret the result.

Answer-07:

```
require(foreign)
require(ggplot2)
require(VGAM)
require(boot)
```

Let's look at the data. These are the same data as were used in the ztp example. We import the Stata dataset using the foreign package.

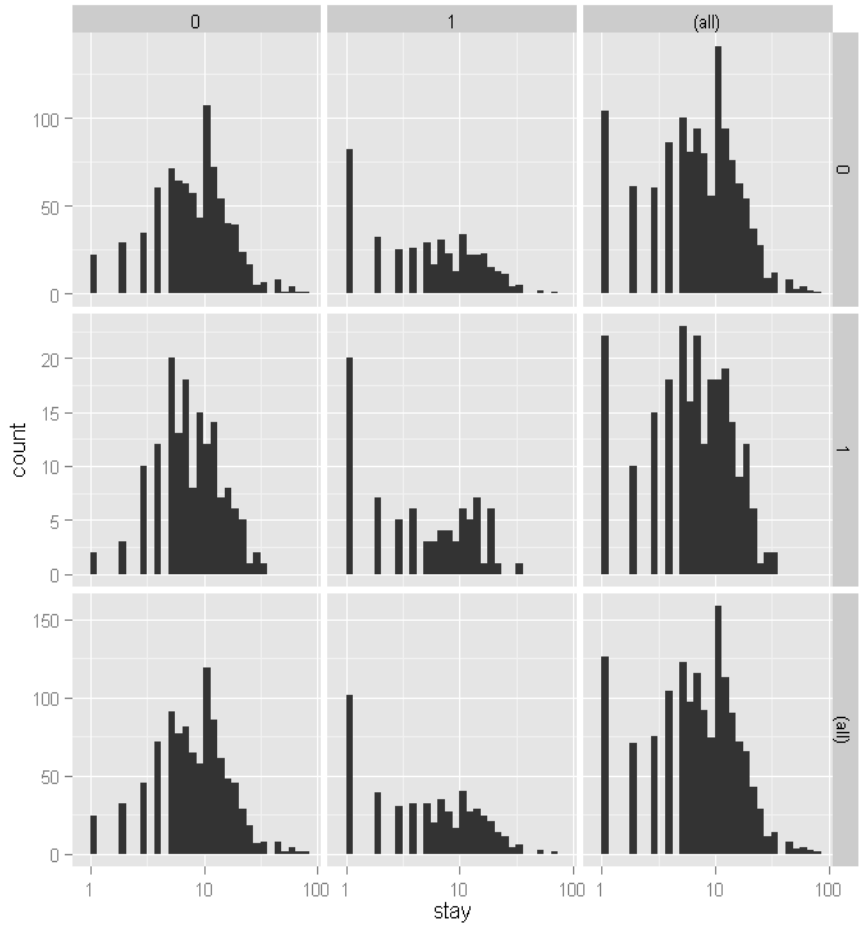
```
dat <- read.dta("https://stats.idre.ucla.edu/stat/data/ztp.dta")


dat <- within(dat, {
   hmo <- factor(hmo)
   died <- factor(died)
})
```

```
##      stay         age      hmo      died
```

```
##  Min.   : 1.00   Min.   :1.00   0:1254   0:981
##  1st Qu.: 4.00   1st Qu.:4.00   1: 239   1:512
##  Median : 8.00   Median :5.00
##  Mean   : 9.73   Mean   :5.23
##  3rd Qu.:13.00   3rd Qu.:6.00
##  Max.   :74.00   Max.   :9.00
```

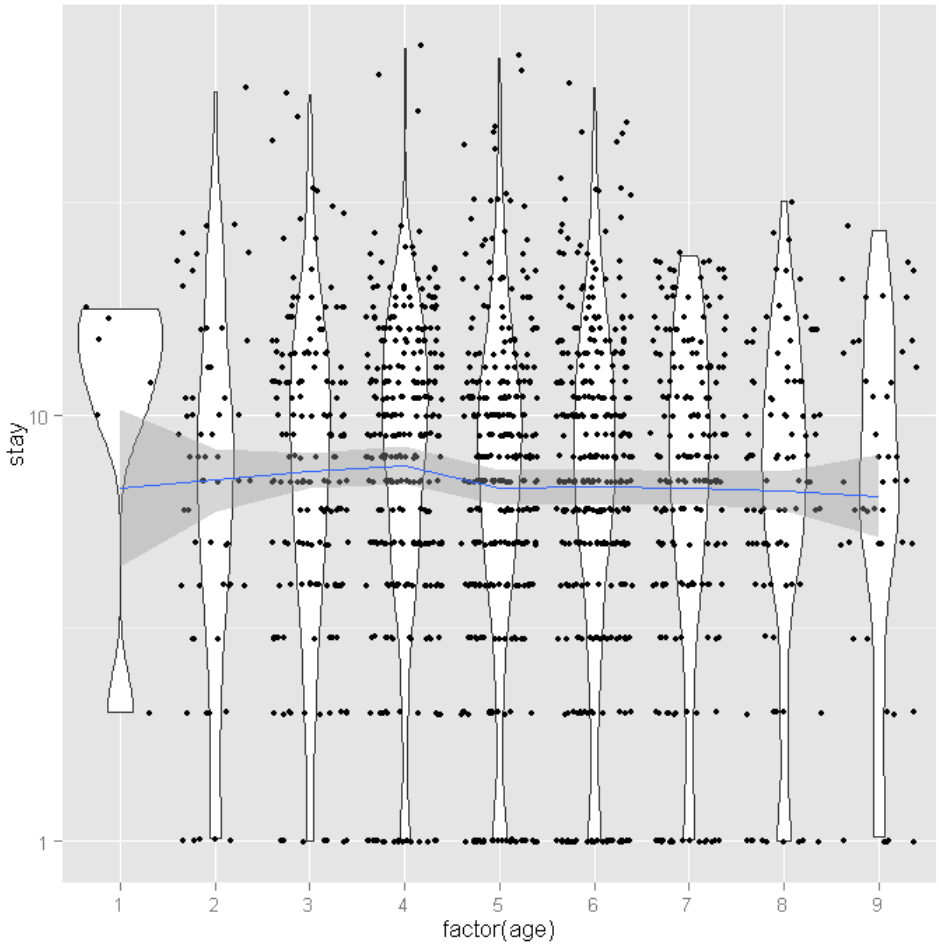
Now let's look at some graphs of the data conditional on various combinations of the variables to get a sense of how the variables work together. We will use the ggplot2 package. First we can look at histograms of stay broken down by hmo on the rows and died on the columns. We also include the marginal distributions, thus the lower right corner represents the overall histogram. We use a log base 10 scale to approximate the canonical link function of the negative binomial distribution (natural logarithm).

```
ggplot(dat, aes(stay)) + geom_histogram() + scale_x_log10() + facet_grid(hmo ~
    died, margins = TRUE, scales = "free_y")
```



From the histograms, it looks like the density of the distribution, does vary across levels of hmo and died, with shorter stays for those in HMOs (1) and shorter for those who did die, including what seems to be an inflated number of 1 day stays. To examine how stay varies across age groups, we can use conditional violin plots which show a kernel density estimate of the distribution of stay mirrored (hence the violin) and conditional on each age group. To further understand the raw data going into each density estimate, we add raw data on top of the violin plots with a small amount of random noise (jitter) to alleviate over plotting. Finally, to get a sense of the overall trend, we add a locally weighted regression line.

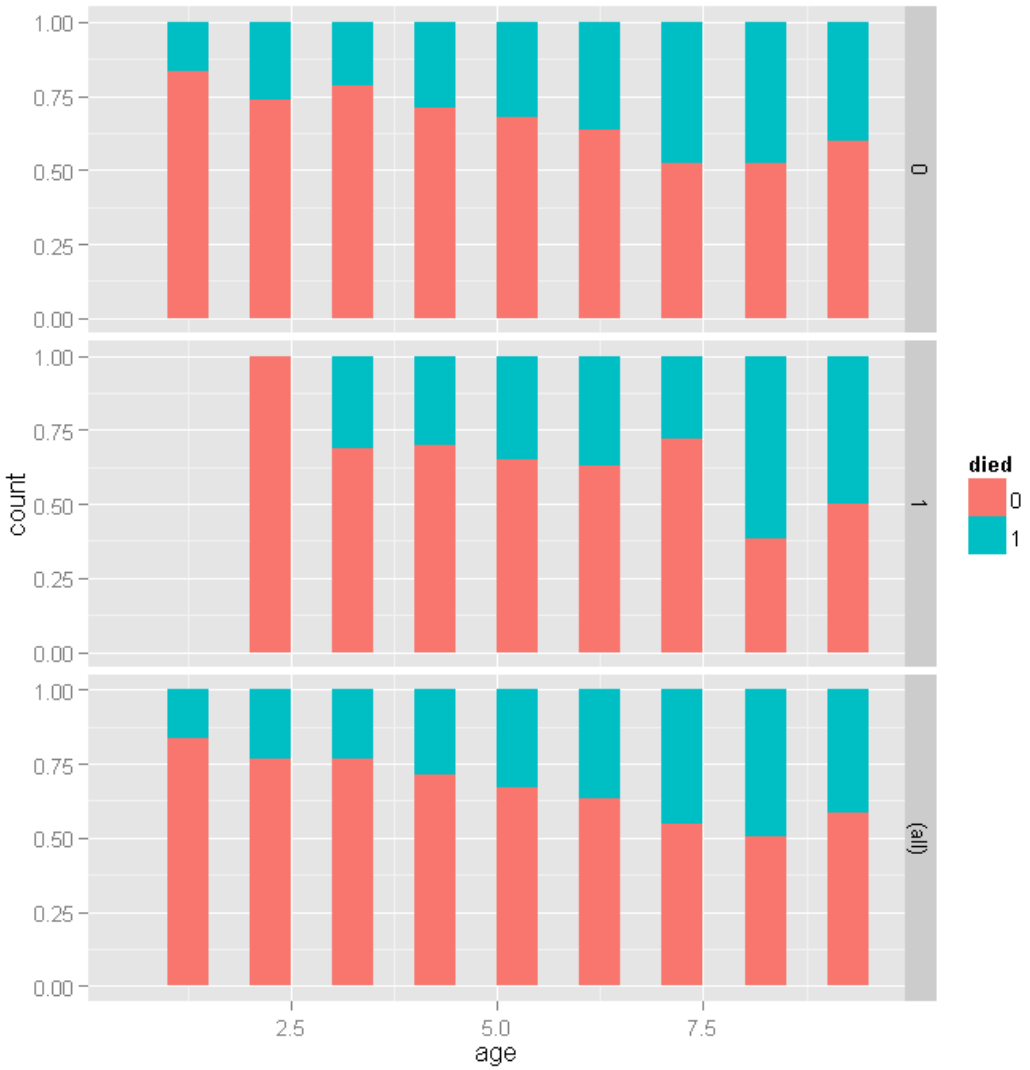
```
ggplot(dat, aes(factor(age), stay)) +
  geom_violin() +
  geom_jitter(size=1.5)  +
  scale_y_log10() +
  stat_smooth(aes(x  = age, y = stay, group=1), method="loess")
```



The distribution of length of stay does not seem to vary much across age groups. This observation from the raw data is corroborated by the relatively flat loess line. Finally

let's look at the proportion of people who lived or died across age groups by whether or not they are in HMOs.

```
ggplot(dat, aes(age, fill=died)) +
 geom_histogram(binwidth=.5, position="fill") +
 facet_grid(hmo ~ ., margins=TRUE)
```



For the lowest ages, a smaller proportion of people in HMOs died, but for higher ages, there does not seem to be a huge difference, with a slightly higher proportion in HMOs dying if anything. Overall, as age group increases, the proportion of those dying increases, as expected.

Analysis methods you might consider:

Below is a list of some analysis methods you may have encountered. Some of the methods listed are quite reasonable while others have either fallen out of favor or have limitations.

- Zero-truncated Negative Binomial Regression – The focus of this web page.
- Zero-truncated Poisson Regression – Useful if you have no overdispersion in See the Data Analysis Example for ztp.
- Poisson Regression – Ordinary Poisson regression will have difficulty with zero-truncated data. It will try to predict zero counts even though there are no zero values.
- Negative Binomial Regression – Ordinary Negative Binomial regression will have difficulty with zero-truncated data. It will try to predict zero counts even though there are no zero values.
- OLS Regression – You could try to analyze these data using OLS regression. However, count data are highly non-normal and are not well estimated by OLS regression.

Zero-truncated negative binomial regression:

To fit the zero-truncated negative binomial model, we use the vglm function in the VGAM package. This function fits a very flexible class of models called vector generalized linear models to a wide range of assumed distributions. In our case, we believe the data come from the negative binomial distribution, but without zeros. Thus the values are strictly positive poisson, for which we use the positive negative binomial family via the posnegbinomial function passed to vglm.

```
m1 <- vglm(stay ~ age + hmo + died, family = posnegbinomial(), data = dat)
```

```
## [1] "head(w)"
##      [,1]
## [1,]    1
## [2,]    1
## [3,]    1
## [4,]    1
## [5,]    1
## [6,]    1
## [1] "head(y)"
##   [,1]
## 1    4
## 2    9
```

```
## 3    3
## 4    9
## 5    1
## 6    4
```

```
summary(m1)
```

```
##
## Call:
## vglm(formula = stay ~ age + hmo + died, family = posnegbinomial(),
##     data = dat)
##
## Pearson Residuals:
##                Min     1Q Median    3Q Max
## log(munb)   -1.4 -0.70  -0.23 0.45 9.8
## log(size) -14.1 -0.27   0.45 0.76 1.0
##
## Coefficients:
##               Estimate Std. Error z value
## (Intercept):1    2.408      0.072    33.6
## (Intercept):2    0.569      0.055    10.4
## age             -0.016      0.013    -1.2
## hmo1             -0.147      0.059    -2.5
## died1            -0.218      0.046    -4.7
##
## Number of linear predictors:  2
##
## Names of linear predictors: log(munb), log(size)
##
## Dispersion Parameter for posnegbinomial family:   1
##
## Log-likelihood: -4755 on 2981 degrees of freedom
```

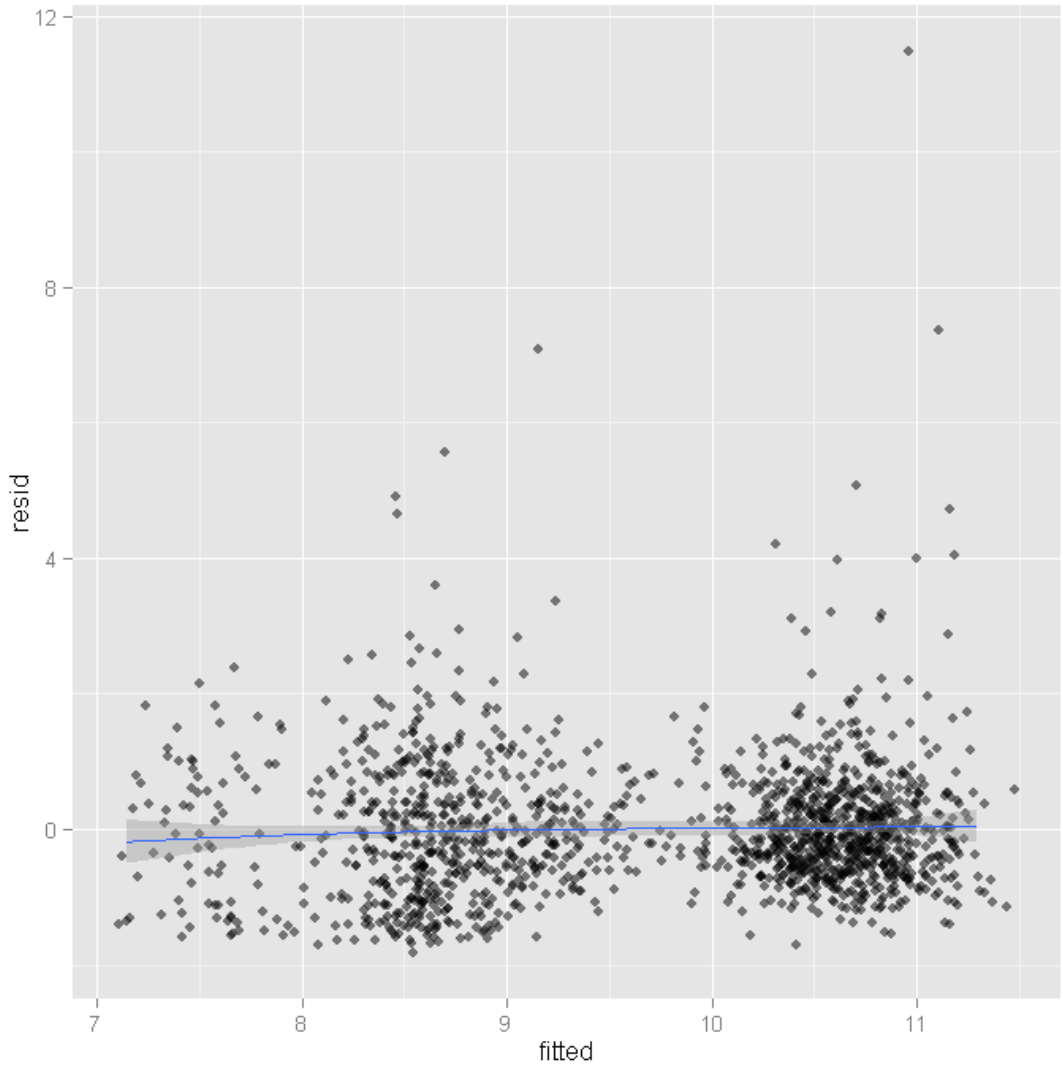
```
##
## Number of iterations: 4
```

The output looks very much like the output from an OLS regression:

- It begins by echoing the function call showing us what we modeled.
- Next comes the spread of the residuals, there is at least one high residual because the max is much higher than the min.
- Following the residuals are the parameter estimates and standard errors. The z values (frac{Estimate}{SE}) are also printed. No p values are given, although if one wished to assume that the estimates followed the normal distribution, one could easily compute the probability of obtaining that z value. The first intercept is what we know as the typical intercept. The second is the over dispersion parameter, (alpha).
- The number of linear predictors is two, one for the expected mean (lambda) and one for the over dispersion.
- Next the dispersion parameter is printed, assumed to be one after accounting for overdispersion.
- Finally the log likelihood -4755.2796 is printed along with the number of iterations needed to reach convergence.

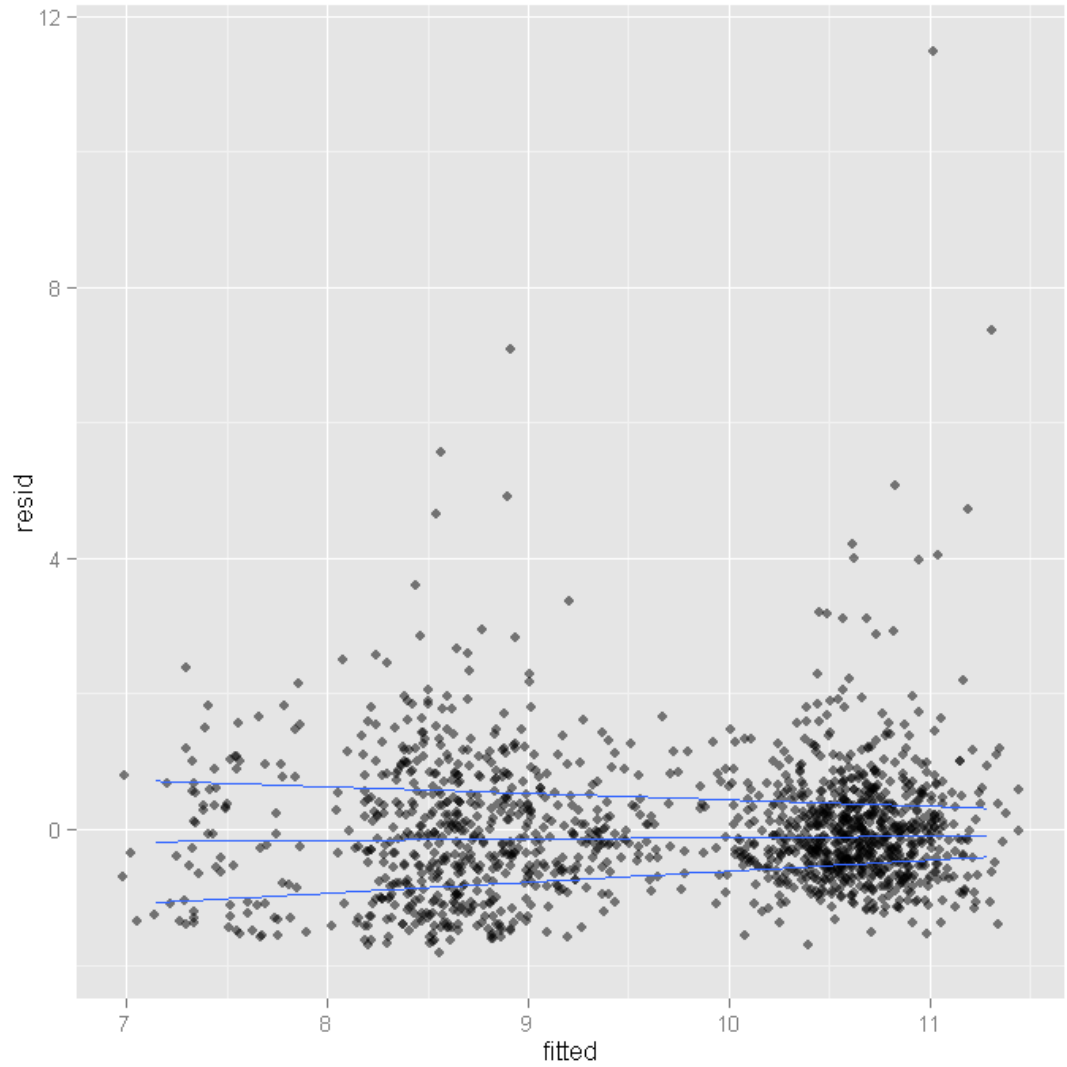
Now let's look at a plot of the residuals versus fitted values. We add random horizontal noise as well as 50 percent transparency to alleviate over plotting and better see where most residuals fall. Note that these residuals are for the mean prediction.

```
output <- data.frame(resid = resid(m1)[, 1], fitted = fitted(m1))

ggplot(output, aes(fitted, resid)) + geom_jitter(position = position_jitter(width = 0.25),
    alpha = 0.5) + stat_smooth(method = "loess")
```



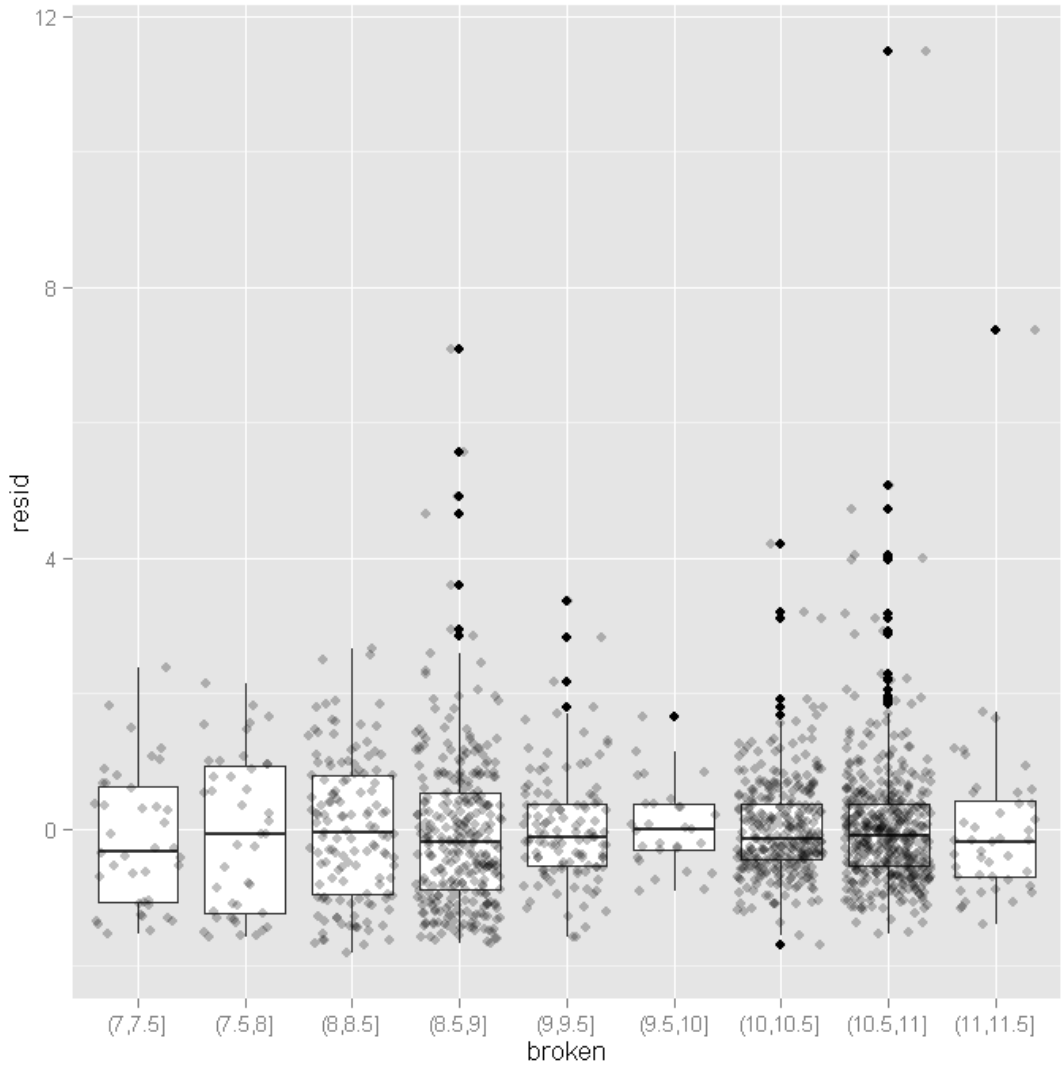
The mean is around zero across all the fitted levels it looks like. However, there are some values that look rather extreme. To see if these have much influence, we can fit lines using quantile regression, these lines represent the 75th, 50th, and 25th percentiles.

```
ggplot(output, aes(fitted, resid)) +
  geom_jitter(position=position_jitter(width=.25),  alpha=.5) +
  stat_quantile(method="rq")
## Smoothing formula not specified.  Using: y ~ x
```



Here we see the spread narrowing at higher levels. Let's cut the data into intervals and check box plots for each. We will get the breaks from the algorithm for a histogram.

```
output <- within(output, {
  broken <- cut(fitted, hist(fitted, plot=FALSE)$breaks)
})


ggplot(output, aes(broken, resid)) +
 geom_boxplot() +
 geom_jitter(alpha=.25)
```



The variance seems to decrease slightly at higher fitted values, except for the very last category (this shown by the hinges of the boxplots).

- The value of the coefficient for age, -0.0157 suggests that the log count of stay decreases by 0.0157 for each year increase in age.
- The coefficient for hmo, -0.1471 indicates that the log count of stay for HMO patient is 0.1471 less than for non-HMO patients.
- The log count of stay for patients who died while in the hospital was 0.2178 less than those patients who did not die.
- The value of the constant 2.4083 is the log count of the stay when all of the predictors equal zero.
- The value of the second intercept, the over dispersion parameter, (alpha) is 0.5686.

To test whether we need to estimate over dispersion, we could fit a zero-truncated Poisson model and compare the two.

```
m2 <- vglm(formula = stay ~ age + hmo + died, family = pospoisson(), data = dat)
```

```
## change in deviance
(dLL <- 2 * (logLik(m1) - logLik(m2)))
```

```
## [1] 4307
```

```
## p-value, 1 df---the overdispersion parameter
pchisq(dLL, df = 1, lower.tail = FALSE)
```

```
## [1] 0
```

Based on this, we would conclude that the negative binomial model is a better fit to the data.

```
dput(round(coef(m1),3))
```

```
## structure(c(2.408, 0.569, -0.016, -0.147, -0.218), .Names = c("(Intercept):1",
## "(Intercept):2", "age", "hmo1", "died1"))
```

```
f <- function(data, i, newdata) {
  require(VGAM)
  m <- vglm(formula = stay ~ age + hmo + died, family = posnegbinomial(),
    data = data[i, ], coefstart = c(2.408, 0.569, -0.016, -0.147, -0.218))
  mparams <- as.vector(t(coef(summary(m))[, 1:2]))
  yhat <- predict(m, newdata, type = "response")
  return(c(mparams, yhat))
}

## newdata for prediction
newdata <- expand.grid(age = 1:9, hmo = factor(0:1), died = factor(0:1))
newdata$yhat <- predict(m1, newdata, type = "response")

set.seed(10)
```

```
res <- boot(dat, f, R = 1200, newdata = newdata, parallel = "snow", ncpus = 4)
```

```
## [1] "head(w)"
##      [,1]
## [1,]    1
## [2,]    1
## [3,]    1
## [4,]    1
## [5,]    1
## [6,]    1
## [1] "head(y)"
##   [,1]
## 1    4
## 2    9
## 3    3
## 4    9
## 5    1
## 6    4
```

The results are alternating parameter estimates and standard errors for the parameters (the first 10). That is, the first row has the first parameter estimate from our model. The second has the standard error for the first parameter. The third column contains the bootstrapped standard errors.

Now we can get the confidence intervals for all the parameters. We start on the original scale with percentile and basic bootstrap CIs.

```
## basic parameter estimates with percentile and bias adjusted CIs
parms <- t(sapply(c(1, 3, 5, 7, 9), function(i) {
  out <- boot.ci(res, index = c(i, i + 1), type = c("perc", "basic"))
  with(out, c(Est = t0, pLL = percent[4], pUL = percent[5],
    basicLL = basic[4], basicLL = basic[5]))
}))


## add row names
row.names(parms) <- names(coef(m1))
```

```
## print results
parms
```

```
##                  Est      pLL      pUL  basicLL  basicLL
## (Intercept):1  2.40833  2.26288  2.55285  2.26381  2.55377
## (Intercept):2  0.56864  0.43812  0.70414  0.43314  0.69916
## age           -0.01569 -0.04233  0.01089 -0.04228  0.01095
## hmo1          -0.14706 -0.26276 -0.03931 -0.25481 -0.03135
## died1         -0.21777 -0.32846 -0.11476 -0.32078 -0.10708
```

The bootstrapped confidence intervals are wider than would be expected using a normal based approximation. The bootstrapped CIs are more consistent with the CIs from Stata when using robust standard errors.

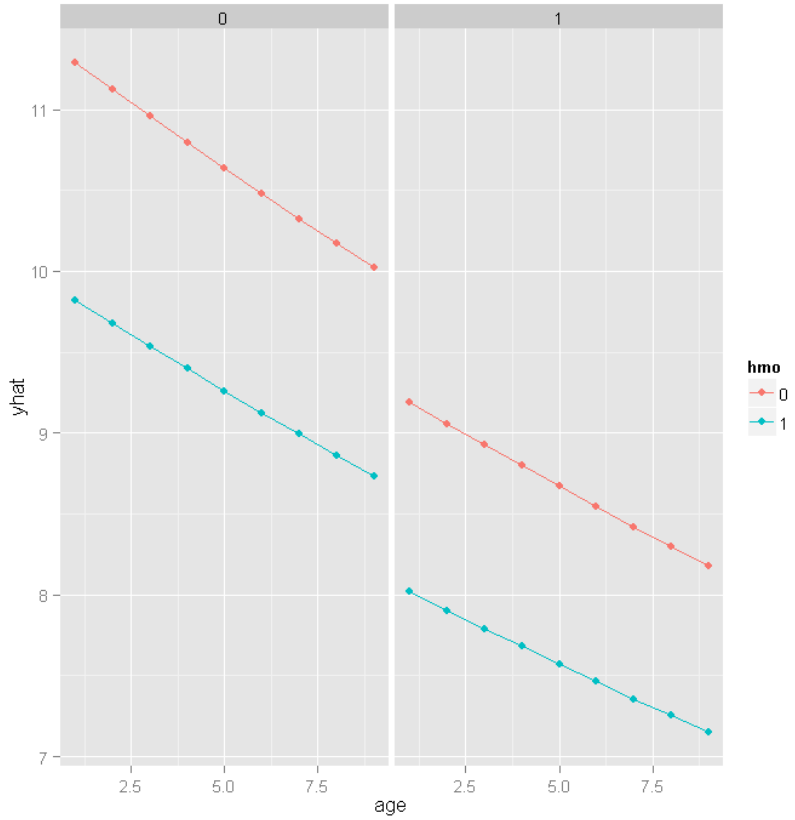
Now we can estimate the incident risk ratio (IRR) for the negative binomial model.

```
## exponentiated parameter estimates with percentile and bias adjusted CIs
expparms <- t(sapply(c(1, 3, 5, 7, 9), function(i) {
  out <- boot.ci(res, index = c(i, i + 1), type = c("perc", "basic"), h = exp)
  with(out, c(Est = t0, pLL = percent[4], pUL = percent[5],
    basicLL = basic[4], basicLL = basic[5]))
}))

## add row names
row.names(expparms) <- names(coef(m1))
## print results
expparms
```

```
##                  Est    pLL     pUL basicLL basicLL
## (Intercept):1 11.1154 9.6107 12.8436  9.3871 12.6200
## (Intercept):2  1.7659 1.5498  2.0221  1.5096  1.9819
## age            0.9844 0.9585  1.0110  0.9579  1.0103
## hmo1           0.8632 0.7689  0.9615  0.7650  0.9576
## died1          0.8043 0.7200  0.8916  0.7170  0.8886
```

The results are consistent with what we initially viewed graphically, age does not have a significant effect, but hmo and died both do. In order to better understand our results and model, let's plot some predicted values. Because all of our predictors were categorical (hmo and died) or had a small number of unique values (age) we will get predicted values for all possible combinations. This was actually done earlier when we bootstrapped the parameter estimates by creating a new data set using the expand.grid function, then estimating the predicted values using the predict function. Now we can plot that data.

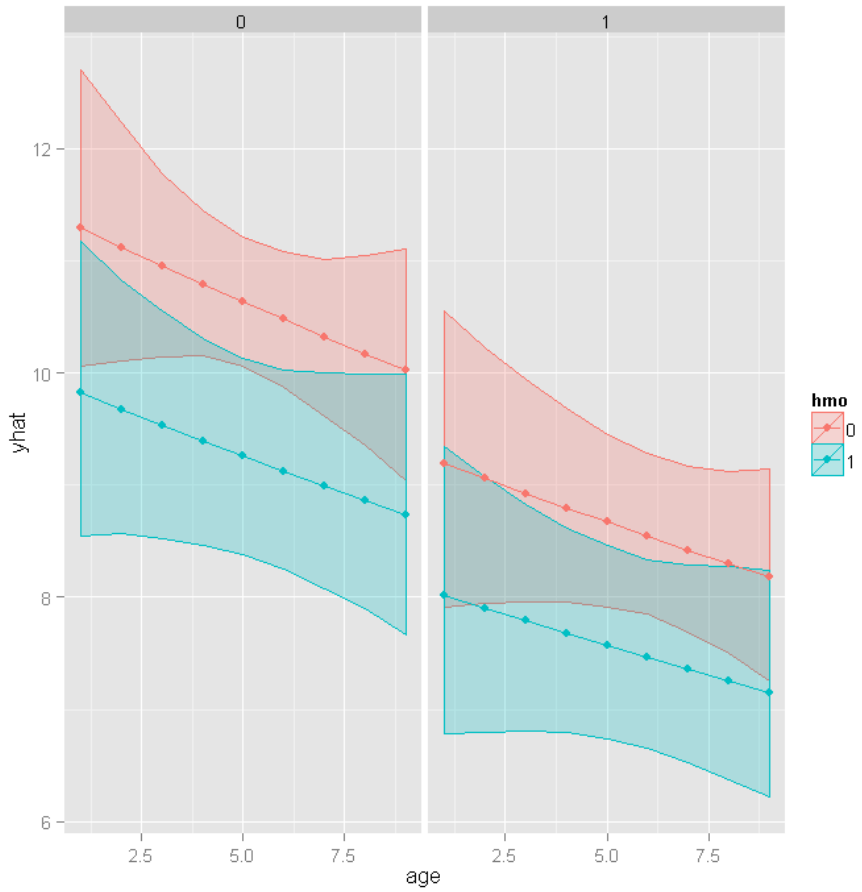
```
ggplot(newdata, aes(x = age, y = yhat, colour = hmo))  +
  geom_point() +
  geom_line() +
  facet_wrap(~ died)
```



If we really wanted to compare the predicted values, we could bootstrap confidence intervals around the predicted estimates. These confidence intervals are not for the predicted value themselves, but that that is the mean predicted value (i.e., for the estimate, not a new individual). Because fitting these models is slow, we included the predicted values earlier when we bootstrapped the model parameters. We will go back to the bootstrap output now and get the confidence intervals for the predicted values.

```
## get the bootstrapped percentile CIs
yhat <- t(sapply(10 + (1:nrow(newdata)), function(i) {
```

```
  out <- boot.ci(res,  index = i, type = c("perc"))
  with(out, c(Est = t0, pLL = percent[4], pUL = percent[5]))
}))

## merge CIs with predicted values
newdata <- cbind(newdata, yhat)

## graph with CIs
ggplot(newdata, aes(x = age, y = yhat, colour = hmo, fill = hmo))  +
  geom_ribbon(aes(ymin  = pLL, ymax = pUL), alpha = .25) +
  geom_point() +
  geom_line() +
  facet_wrap(~ died)
```



Things to consider:

- Count data often use exposure variable to indicate the number of times the event could have happened. You can incorporate exposure into your model by using the offset() option.
- It is not recommended that zero-truncated negative models be applied to small samples. What constitutes a small sample does not seem to be clearly defined in the literature.
- Pseudo-R-squared values differ from OLS R-squareds, please see FAQ: What are pseudo R-squareds? for a discussion on this issue.

References:

UCLA. (n.d.). *Resources*. Retrieved from UCLA: https://stats.oarc.ucla.edu/other/dae/