Simple Regression Examples

Doug Perez

7/6/2019

Introduction

This document is intended as a brief introduction to regression models in R. Specifically, we will cover logistic, multinomial, and Poisson regression types.

Logistic Regression

To begin, we will load a data set of graduate school admissions.

```
# Graduate school admissions data from UCLA
mydata <- read.csv("https://stats.idre.ucla.edu/stat/data/binary.csv")
# Saved at: \Consulting\Public Code Samples\DataScience\Data
# Alternate URL:  https://1drv.ms/u/s!AgXALnk1NWLck4JA_01l6gpD6pZxpQ


## 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
```

The data set includes a binary response variable called admit. This variable will be the focus of our attention, and we will attempt to use the gre, gpa, and rank variables as predictors. Rank is an integer from 1-4 reflecting the prestige of the admitting institution.

Let's really dive in and take a look at the data set:

```
summary(mydata)
```

```
##      admit             gre             gpa             rank      
##  Min.   :0.0000   Min.   :220.0   Min.   :2.260   Min.   :1.000  
##  1st Qu.:0.0000   1st Qu.:520.0   1st Qu.:3.130   1st Qu.:2.000  
##  Median :0.0000   Median :580.0   Median :3.395   Median :2.000  
##  Mean   :0.3175   Mean   :587.7   Mean   :3.390   Mean   :2.485  
##  3rd Qu.:1.0000   3rd Qu.:660.0   3rd Qu.:3.670   3rd Qu.:3.000  
##  Max.   :1.0000   Max.   :800.0   Max.   :4.000   Max.   :4.000
```

```r
sapply(mydata, sd)
```

```
##       admit         gre         gpa        rank
##   0.4660867 115.5165364   0.3805668   0.9444602
```

```r
xtabs(~admit + rank, data = mydata)
```

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

Model the data

For this first regression example, we will attempt to fit a generalized linear model (glm) to predict the admit (response) variable from the other predictors.

```r
# Factor the rank variable and create a logistic regretion using glm() - Generalized Linear Model
mydata$rank <- factor(mydata$rank)
mylogit <- glm(admit ~ gre + gpa + rank, data = mydata, family = "binomial")

# See regression results using summary
summary(mylogit)
```

```
##
## Call:
## glm(formula = admit ~ gre + gpa + rank, family = "binomial",
##     data = mydata)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -1.6268  -0.8662  -0.6388   1.1490   2.0790
##
## Coefficients:
##              Estimate Std. Error z value Pr(>|z|)
## (Intercept) -3.989979   1.139951  -3.500 0.000465 ***
## gre          0.002264   0.001094   2.070 0.038465 *
## gpa          0.804038   0.331819   2.423 0.015388 *
## rank2       -0.675443   0.316490  -2.134 0.032829 *
## rank3       -1.340204   0.345306  -3.881 0.000104 ***
## rank4       -1.551464   0.417832  -3.713 0.000205 ***
## ---
## 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: 470.52
##
## Number of Fisher Scoring iterations: 4
```

Model output explained

The output begins with 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

Model fitness tests

```
# We can create confidence levels by log likelihood
print("CI by log likelihood")
```

```
## [1] "CI by log likelihood"
```

```
confint(mylogit)
```

```
## Waiting for profiling to be done...
```

```
##                     2.5 %       97.5 %
## (Intercept) -6.2716202334 -1.792547080
## gre          0.0001375921  0.004435874
## gpa          0.1602959439  1.464142727
## rank2       -1.3008888002 -0.056745722
## rank3       -2.0276713127 -0.670372346
## rank4       -2.4000265384 -0.753542605
```

```
# or based on the standard errors:
print("CI by standard error")
```

```
## [1] "CI by standard error"
```

```
confint.default(mylogit)
```

```
##                     2.5 %       97.5 %
## (Intercept) -6.2242418514 -1.755716295
## gre          0.0001202298  0.004408622
## gpa          0.1536836760  1.454391423
## rank2       -1.2957512650 -0.055134591
## rank3       -2.0169920597 -0.663415773
## rank4       -2.3703986294 -0.732528724
```

```
# The wald test will give us the overall effect of the rank variable on admit:
print("Wald test of the three rank terms")
```

```
## [1] "Wald test of the three rank terms"
```

```
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 Wald test gives us an overall measure of significance of the rank variable. The Chi-squared test statistic of 20.9 on 3 degrees of freedom with a p-value of 0.00011 indicates that the rank variable *does* have an overall effect on the admit (response) variable.

To test variances within the rank term, we can perform another Wald test, isolating the ranks of 2 and 3, the fourth and fifth terms from above, by multiplying them by 1 and -1, respectively, and canceling the rest of the terms by multiplying by zero. This technique is demonstrated below:

```
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
```

Here we see a result of the chi-squared test statistic of 5.5 with 1 degree of freedom, associated with a p-value of 0.019. This result tells us that the difference between the coefficient for rank=2 and the coefficient for rank=3 is statistically significant.

Predictions using the model

In order to create predicted probabilities we first need to create a new data frame with the values we want the independent variables to take on to create our predictions.

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

## view data frame
newdata1
```

```
##     gre    gpa rank
## 1 587.7 3.3899    1
## 2 587.7 3.3899    2
## 3 587.7 3.3899    3
## 4 587.7 3.3899    4
```

These objects must have the same names as the variables in your logistic regression above (e.g. in this example the mean for gre must be named gre).

Now that we have the data frame we want to use to calculate the predicted probabilities, we can tell R to create the predicted probabilities. We pass the dataframe "newdata1" to the mylogit model using the predict() function. The result will be a predicted rank in the rankP column of the data frame:

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

```
##     gre    gpa rank     rankP
## 1 587.7 3.3899    1 0.5166016
## 2 587.7 3.3899    2 0.3522846
## 3 587.7 3.3899    3 0.2186120
## 4 587.7 3.3899    4 0.1846684
```

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))))
```

The code to generate the predicted probabilities (the first line below) is the same as before, except we are also going to ask for standard errors so we can plot a confidence interval. We get the estimates on the link scale and back transform both the predicted values and confidence limits into probabilities.

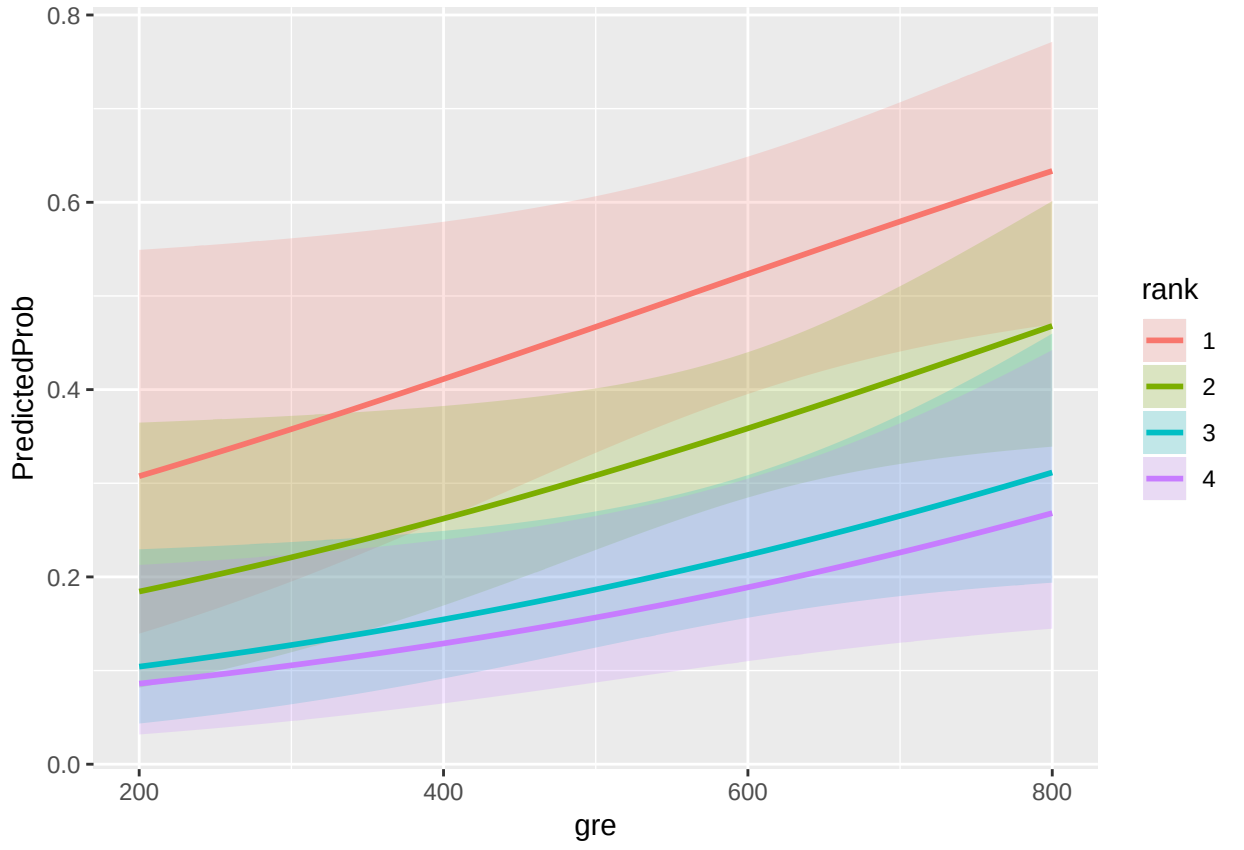
```
newdata3 <- cbind(newdata2, predict(mylogit, newdata = newdata2, type = "link",
    se = TRUE))
newdata3 <- within(newdata3, {
    PredictedProb <- plogis(fit)
    LL <- plogis(fit - (1.96 * se.fit))
    UL <- plogis(fit + (1.96 * se.fit))
})

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

```
##        gre    gpa rank        fit    se.fit residual.scale        UL
## 1 200.0000 3.3899    1 -0.8114870 0.5147714              1 0.5492064
## 2 206.0606 3.3899    1 -0.7977632 0.5090986              1 0.5498513
## 3 212.1212 3.3899    1 -0.7840394 0.5034491              1 0.5505074
## 4 218.1818 3.3899    1 -0.7703156 0.4978239              1 0.5511750
## 5 224.2424 3.3899    1 -0.7565919 0.4922237              1 0.5518545
## 6 230.3030 3.3899    1 -0.7428681 0.4866494              1 0.5525464
##          LL PredictedProb
## 1 0.1393812     0.3075737
## 2 0.1423880     0.3105042
## 3 0.1454429     0.3134499
## 4 0.1485460     0.3164108
## 5 0.1516973     0.3193867
## 6 0.1548966     0.3223773
```

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)
```



Fit tests and log likelihood

Concluding, we run some fit tests and take the log likelihood of the model:

```
# Some methods to test fit below
with(mylogit, null.deviance - deviance)
```

```
## [1] 41.45903
```

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

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

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

```
## [1] 7.578194e-08
```

```
#To see the model's log likelihood
logLik(mylogit)
```

```
## 'log Lik.' -229.2587 (df=6)
```

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.

Multinomial Regression

In this portion of the discussion, we will look at multinomial regression, where we examine the influence of more than one predictor variable on the response. For example, we might attempt to predict a person's weight using predictor variables of height and gender.

Data Import and Review

The data we will use to help us explore this regression is a vocational survey of high school students, tracking occupational outcomes along varied academic paths.

```
# A data set showing occupational outcomes for vocational or academic educational paths
ml <- read.dta("https://stats.idre.ucla.edu/stat/data/hsbdemo.dta")
# Saved at: \Consulting\Public Code Samples\DataScience\Data\UCLAHighSchoolVocationalSurvey.dta
# Alternate URL:  https://1drv.ms/u/s!AgXALnk1NWLck4JDxYy1aZKct65nOw?e=7JBQzi

# Have a look at the data:
with(ml, table(ses, prog))
```

```
##         prog
## ses      general academic vocation
##   low         16       19       12
##   middle      20       44       31
##   high         9       42        7
```

```
with(ml, do.call(rbind, tapply(write, prog, function(x) c(M = mean(x), SD = sd(x)))))
```

```
##                 M       SD
## general  51.33333 9.397775
## academic 56.25714 7.943343
## vocation 46.76000 9.318754
```

ses is a 3-level categorical variable representing Socio-Economic Status (SES). In the first table above, we see the outcomes split by ses; the second table gives us the mean and standard deviation of the three outcome possibilities.

Construct an initial model

To properly model the multiple levels in our outcome, we must first specify one of the levels to use as our baseline. Using relevel() below, we choose the "academic" program as our baseline; "general" and "vocation" will be modeled in comparison to "academic".

```
# Let's use relevel to create a new variable weighing the "academic" prog variable response
ml$prog2 <- relevel(ml$prog, ref = "academic")
test <- multinom(prog2 ~ ses + write, data = ml) # multinom() does not require reshaping like mlogit()
```

```
## # weights:  15 (8 variable)
## initial  value 219.722458
## iter  10 value 179.982880
## final  value 179.981726
## converged
```

```
summary(test)
```

```
## Call:
## multinom(formula = prog2 ~ ses + write, data = ml)
##
## Coefficients:
##          (Intercept)  sesmiddle    seshigh      write
## general     2.852198 -0.5332810 -1.1628226 -0.0579287
## vocation    5.218260  0.2913859 -0.9826649 -0.1136037
##
## Std. Errors:
##          (Intercept) sesmiddle   seshigh      write
## general     1.166441 0.4437323 0.5142196 0.02141097
## vocation    1.163552 0.4763739 0.5955665 0.02221996
##
## Residual Deviance: 359.9635
## AIC: 375.9635
```

```
# Test is a multinom() output of our model using the releveled program
# We run a couple of tests on the model:
z <- summary(test)$coefficients/summary(test)$standard.errors
z
```

```
##          (Intercept)  sesmiddle   seshigh     write
## general     2.445214 -1.2018081 -2.261334 -2.705562
## vocation    4.484769  0.6116747 -1.649967 -5.112689
```

```
# 2-tailed z test
p <- (1 - pnorm(abs(z), 0, 1)) * 2
p
```

```
##           (Intercept) sesmiddle    seshigh        write
## general  0.0144766100 0.2294379 0.02373856 6.818902e-03
## vocation 0.0000072993 0.5407530 0.09894976 3.176045e-07
```

Model results interpretation

A one-unit increase in the variable write is associated with the decrease in the log odds of being in general program vs. academic program in the amount of .058.

A one-unit increase in the variable write is associated with the decrease in the log odds of being in vocation program vs. academic program. in the amount of .1136 .

The log odds of being in general program vs. in academic program will decrease by 1.163 if moving from ses="low" to ses="high".

The log odds of being in general program vs. in academic program will decrease by 0.533 if moving from ses="low"to ses="middle", although this coefficient is not significant.

The log odds of being in vocation program vs. in academic program will decrease by 0.983 if moving from ses="low" to ses="high".

The log odds of being in vocation program vs. in academic program will increase by 0.291 if moving from ses="low" to ses="middle", although this coefficient is not significant.

Model fit tests and review

```
# We can start by generating the predicted probabilities for the observations in our dataset and viewin

head(pp <- fitted(test))
```

```
##    academic   general  vocation
## 1 0.1482764 0.3382454 0.5134781
## 2 0.1202017 0.1806283 0.6991700
## 3 0.4186747 0.2368082 0.3445171
## 4 0.1726885 0.3508384 0.4764731
## 5 0.1001231 0.1689374 0.7309395
## 6 0.3533566 0.2377976 0.4088458
```

Using the fitted() function, we produce predicted probabilities of the different possible outcomes, based on the supplied variables in the observation.

Another way to understand the model using the predicted probabilities is to look at the averaged predicted probabilities for different values of the continuous predictor variable write within each level of ses.

```
dwrite <- data.frame(ses = rep(c("low", "middle", "high"), each = 41), write = rep(c(30:70),
    3))

## store the predicted probabilities for each value of ses and write
pp.write <- cbind(dwrite, predict(test, newdata = dwrite, type = "probs", se = TRUE))

## calculate the mean probabilities within each level of ses
by(pp.write[, 3:5], pp.write$ses, colMeans)
```

```
## pp.write$ses: high
##  academic   general  vocation
## 0.6164315 0.1808037 0.2027648
## ------------------------------------------------------------
## pp.write$ses: low
##  academic   general  vocation
## 0.3972977 0.3278174 0.2748849
## ------------------------------------------------------------
## pp.write$ses: middle
##  academic   general  vocation
## 0.4256198 0.2010864 0.3732938
```

Model Visualized

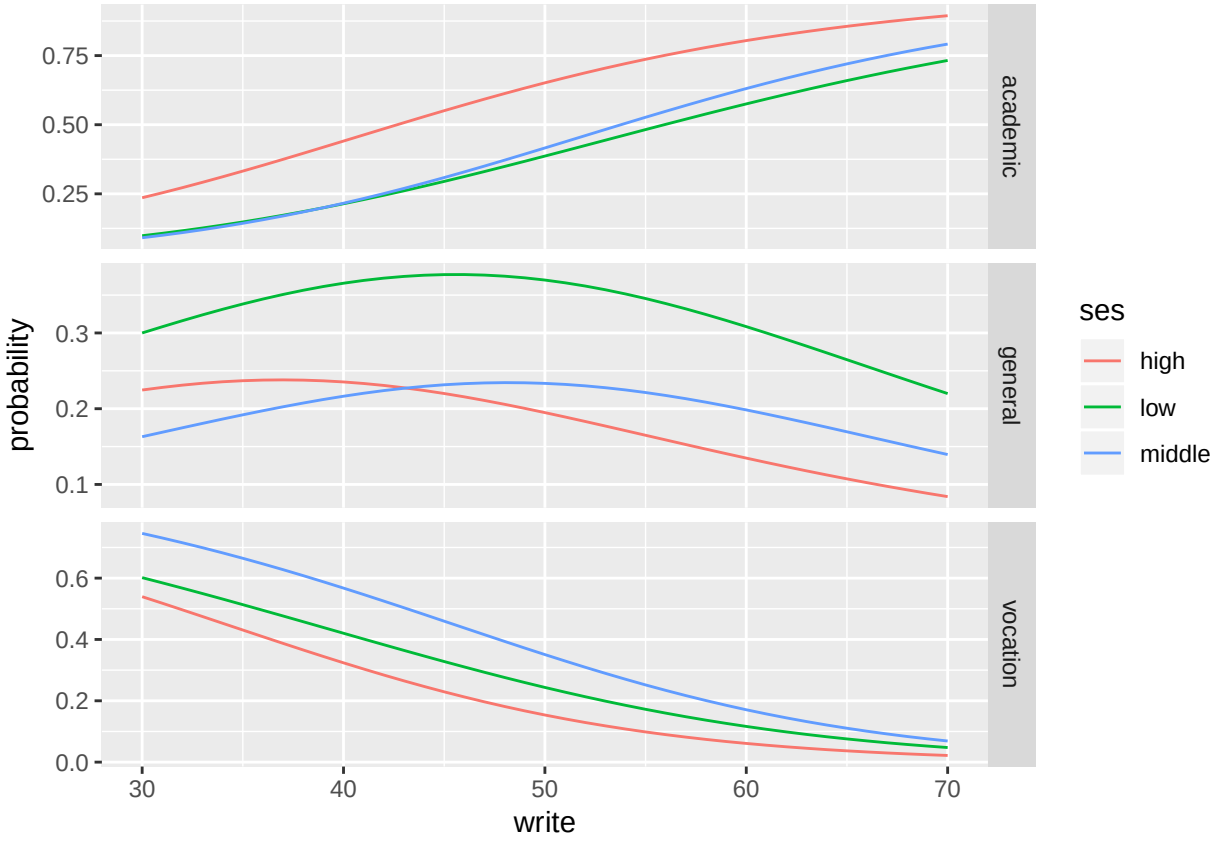
We close this portion of our analysis plotting some of our predicted results to help identify anything significant lurking inside of the data.

Using the melt() function, we spread our predicted values across the sample range. The result is what the spread looks like across a wider data set, visually represented showing areas of overlap, convergence, and divergence.

```
## melt data set to long for ggplot2
lpp <- melt(pp.write, id.vars = c("ses", "write"), value.name = "probability")
head(lpp)  # view first few rows
```

```
##   ses write variable probability
## 1 low    30 academic  0.09843588
## 2 low    31 academic  0.10716868
## 3 low    32 academic  0.11650390
## 4 low    33 academic  0.12645834
## 5 low    34 academic  0.13704576
## 6 low    35 academic  0.14827643
```

```
## plot predicted probabilities across write values for each level of ses
## facetted by program type
ggplot(lpp, aes(x = write, y = probability, colour = ses)) + geom_line() + facet_grid(variable ~
    ., scales = "free")
```



Poisson Regression (Zero-truncated)

A Poisson regression is the choice for counting variables, such as cars arriving as a parking lot over a period of time. In this example, we will use a Poisson Regression where any zero values are removed from the list to make predictions from our data.

The data set we will use is from hospitals, tracking patients' length of stay, HMO participation, and the variable we will be examining, whether or not the patient died.

```
# Import data from hospital tracking patients who have died
dat <- read.dta("https://stats.idre.ucla.edu/stat/data/ztp.dta")
# Saved at: \Consulting\Public Code Samples\DataScience\Data\HospitalDeathTracking.dta
# Alternate URL:  https://1drv.ms/u/s!AgXALnk1NWLck4JBKcMG9lGWFR8TEA?e=jZf1H7

# Convert a couple of the variables we'll be working with to factors
dat <- within(dat, {
  hmo <- factor(hmo)
  died <- factor(died)
})

summary(dat)
```

```
##       stay             age        hmo      died
##  Min.   : 1.000   Min.   :1.000   0:1254   0:981
```

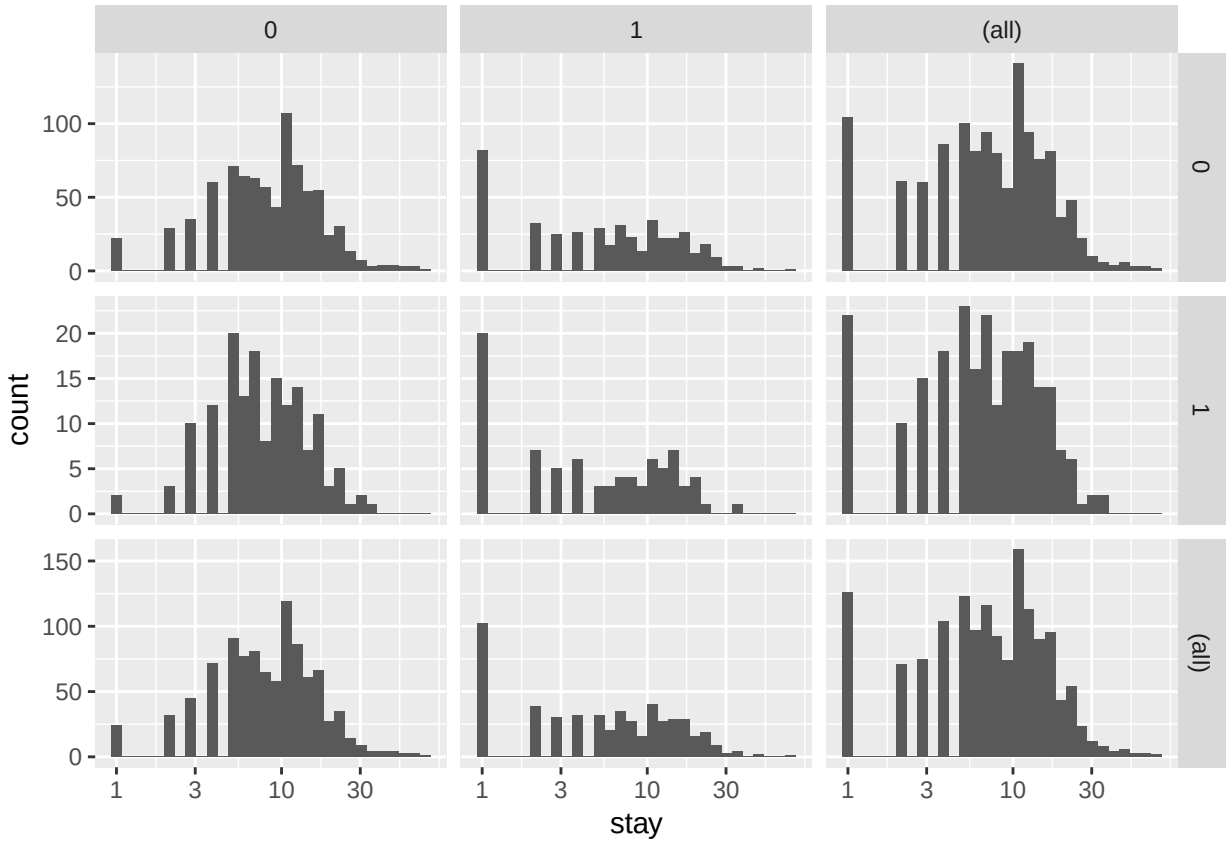
```
##  1st Qu.: 4.000   1st Qu.:4.000   1: 239   1:512
##  Median : 8.000   Median :5.000
##  Mean   : 9.729   Mean   :5.234
##  3rd Qu.:13.000   3rd Qu.:6.000
##  Max.   :74.000   Max.   :9.000
```

Inspecting the data visually

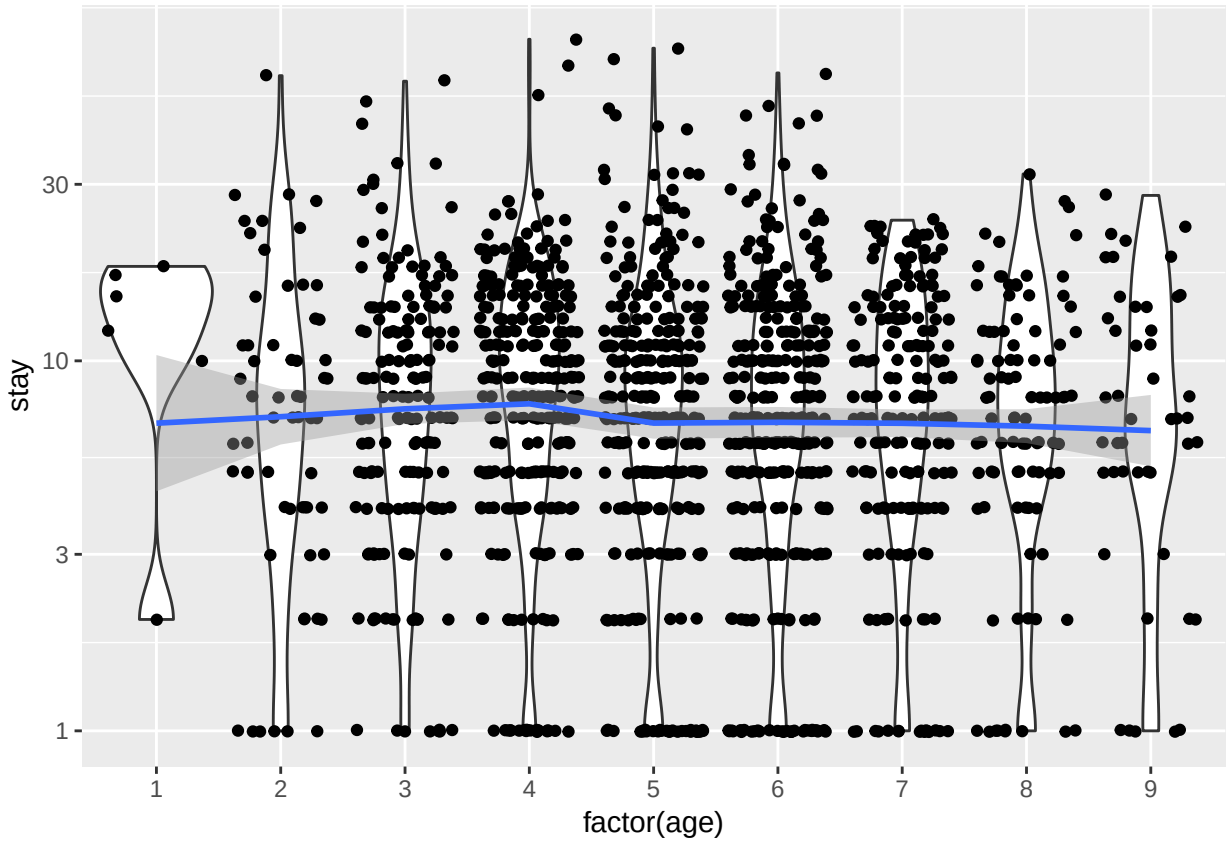
Let's dive right into this set and start visualizing what's here.

```
# To really get a sense for the data, let's plot some graphs:
# Length of stay with died in facets
ggplot(dat, aes(stay)) +
  geom_histogram() +
  scale_x_log10() +
  facet_grid(hmo ~ died, margins=TRUE, scales="free_y")
```

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

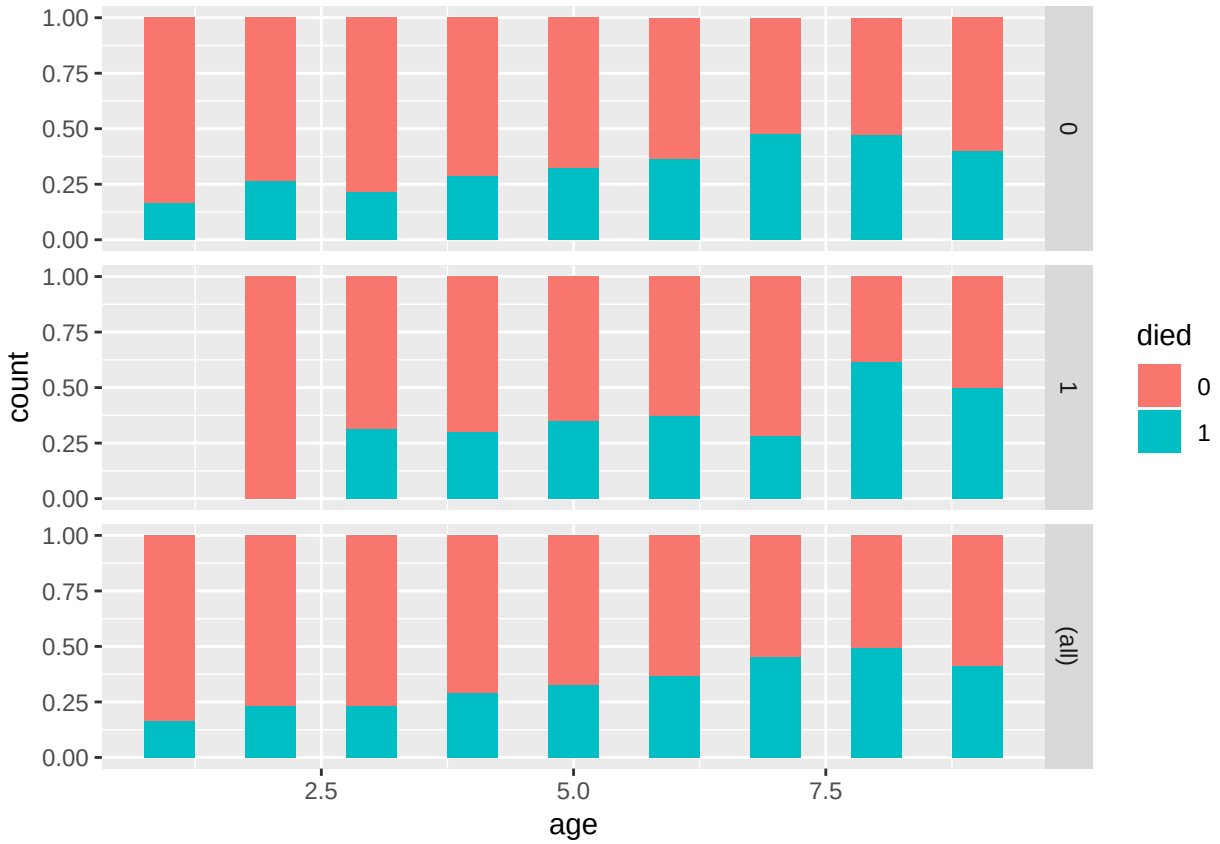



```
# Examine how stay looks across age groups:
ggplot(dat, aes(factor(age), stay)) +
  geom_violin() + # Violin plot is a great choice for this data spread
  geom_jitter(size=1.5) +
  scale_y_log10() +
  stat_smooth(aes(x = age, y = stay, group=1), method="loess")
```




```
# Look at live/die *by hmo*
ggplot(dat, aes(age, fill=died)) +
  geom_histogram(binwidth=.5, position="fill") +
  facet_grid(hmo ~ ., margins=TRUE)
```

```
## Warning: Removed 50 rows containing missing values (geom_bar).
```



The first plot, the histogram, reveals a correlation between shorter stays and HMO participation and the died variable. This may be a relationship worth examining.

The violin plot doesn't show a significant variance across age groups, and the bar plot shows us how the probability of died = 1 increases with age, which is entirely expected and not terribly enlightening to examine in greater depth.

Poisson Model Fitting

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.

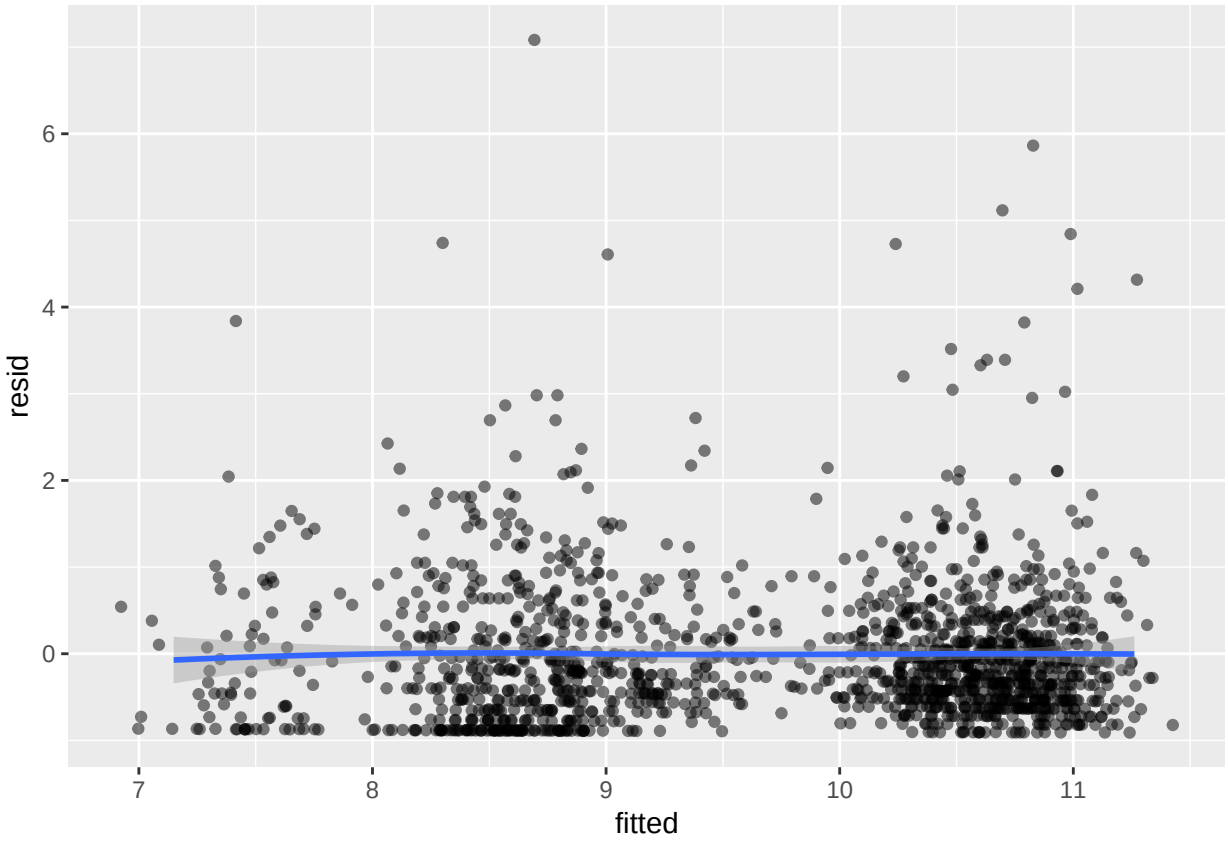
```
m1 <- vglm(stay ~ age + hmo + died, family = pospoisson(), data = dat)
summary(m1)
```

```
##
## Call:
## vglm(formula = stay ~ age + hmo + died, family = pospoisson(),
##     data = dat)
##
## Pearson residuals:
##                       Min     1Q  Median     3Q   Max
## loglink(lambda) -3.032 -1.727 -0.5868 0.9794 20.84
```

```
##
## Coefficients:
##              Estimate Std. Error z value Pr(>|z|)
## (Intercept)  2.435808   0.027332  89.119  < 2e-16 ***
## age         -0.014442   0.005035  -2.869  0.00412 **
## hmo1        -0.135903   0.023741  -5.724 1.04e-08 ***
## died1       -0.203771   0.018372 -11.091  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Name of linear predictor: loglink(lambda)
##
## Log-likelihood: -6908.799 on 1489 degrees of freedom
##
## Number of Fisher scoring iterations: 3
##
## No Hauck-Donner effect found in any of the estimates
```

Let's visualize the residuals and fitted values to get an idea of how our model fits:

```
# plot of residuals versus fitted values
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")
```



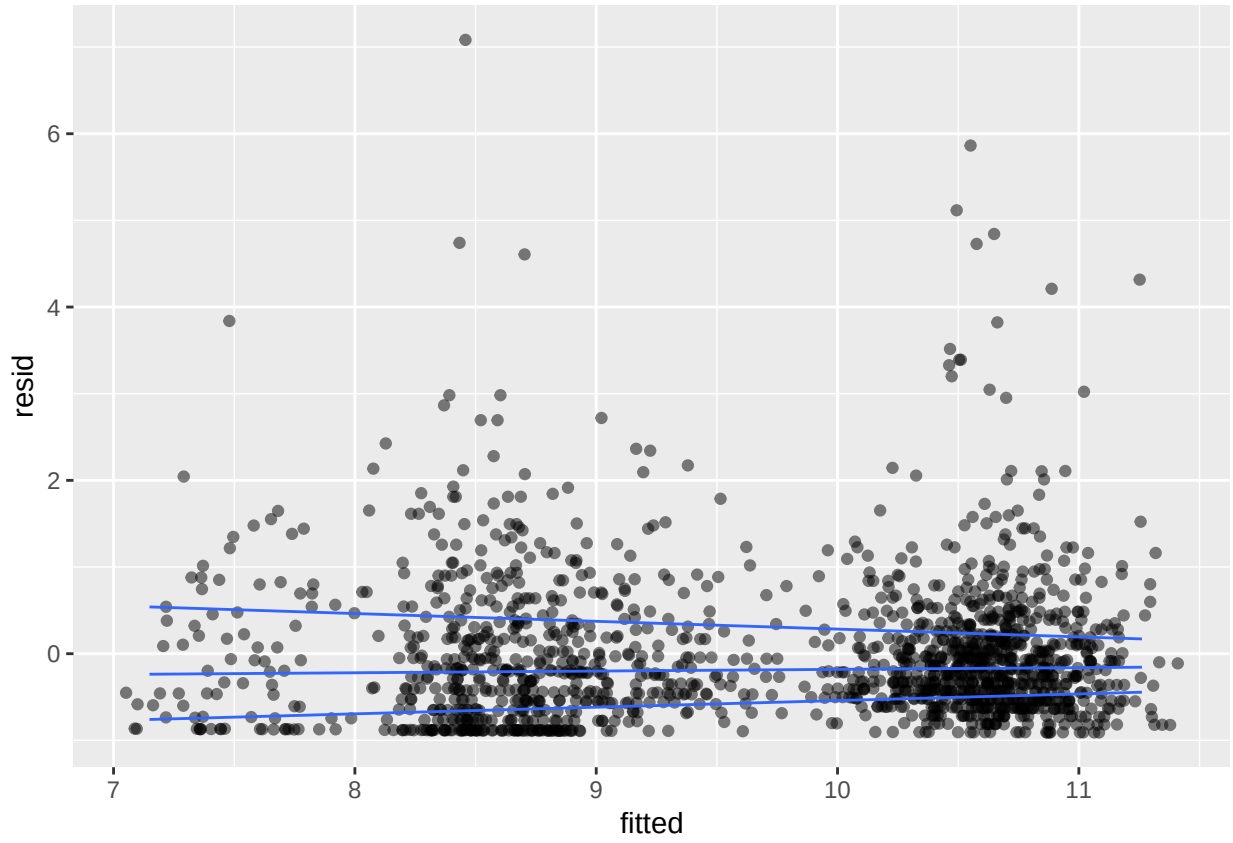
```r
# Fit lines using quantile regression:
ggplot(output, aes(fitted, resid)) +
  geom_jitter(position=position_jitter(width=.25), alpha=.5) +
  stat_quantile(method="rq")
```

```
## Loading required package: SparseM
```

```
##
## Attaching package: 'SparseM'
```

```
## The following object is masked from 'package:base':
##
##     backsolve
```

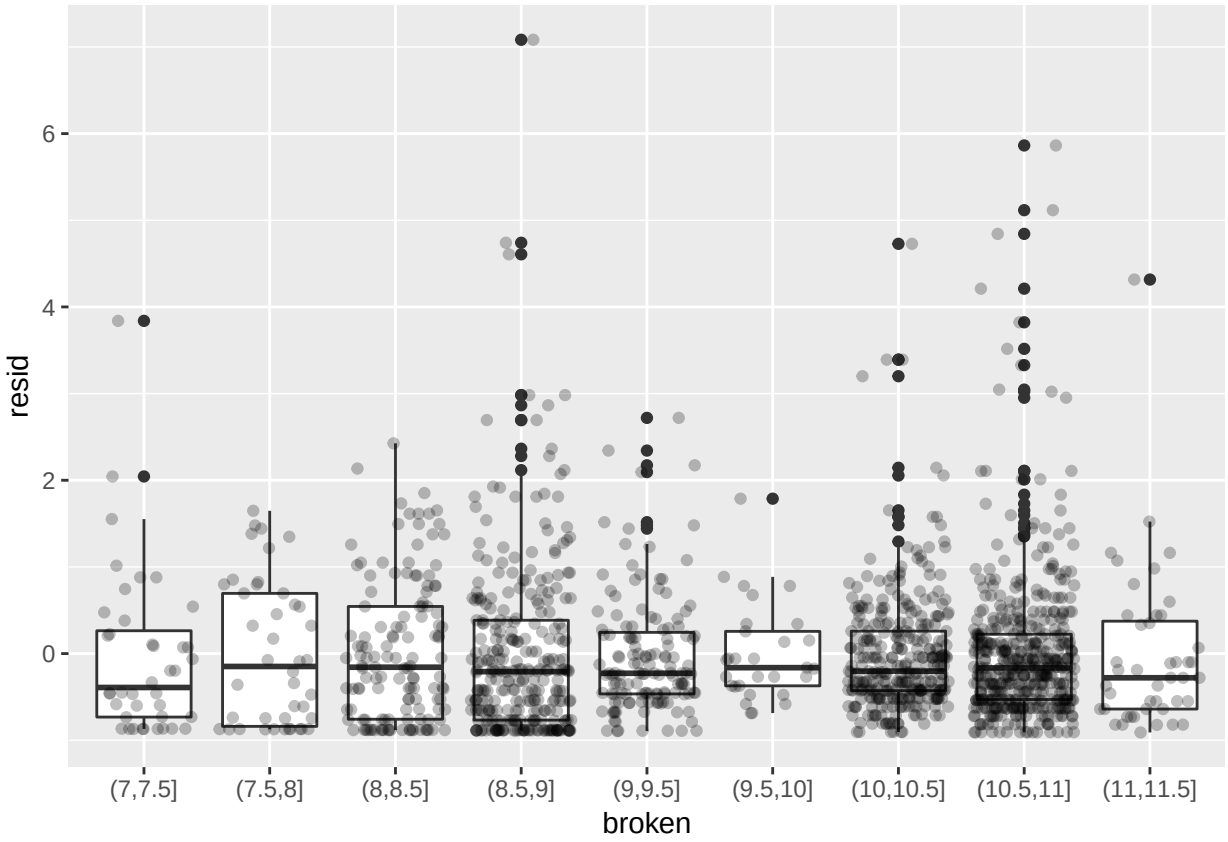
```
## Smoothing formula not specified. Using: y ~ x
```



The first plot shows a mean concentrated around zero. We then calculate the quartiles and add fit lines for the 25th, Median, and 75th percentiles show in the second plot. As the fitted value increases, we do see the quartiles converging, so let's get a different look at the data for more information:

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

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



Poisson Model Fitting

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

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

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

```
## 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*  2.43580824  2.123876e-04 6.962189e-02
## t2*  0.02733232  8.530996e-06 5.621009e-04
## t3* -0.01444203 -5.728615e-05 1.211602e-02
## t4*  0.00503471  2.530820e-06 9.896101e-05
## t5* -0.13590330  1.161636e-03 5.104657e-02
## t6*  0.02374183  1.343953e-05 7.496660e-04
## t7* -0.20377089 -1.486727e-03 4.984121e-02
## t8*  0.01837281  3.871559e-05 3.556392e-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)  2.43580824  2.2993325  2.57780532  2.29381116  2.57228397
## age         -0.01444203 -0.0400288  0.00979822 -0.03868227  0.01114475
## hmo1        -0.13590330 -0.2387489 -0.03807353 -0.23373306 -0.03305769
## died1       -0.20377089 -0.3053032 -0.10627717 -0.30126462 -0.10223856
```

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) 11.4250491 9.9675271 13.1682091 9.6818891 12.8825712
## age          0.9856618 0.9607618  1.0098464 0.9614771  1.0105617
## hmo1         0.8729270 0.7876126  0.9626422 0.7832119  0.9582415
## died1        0.8156492 0.7368999  0.8991754 0.7321230  0.8943985
```

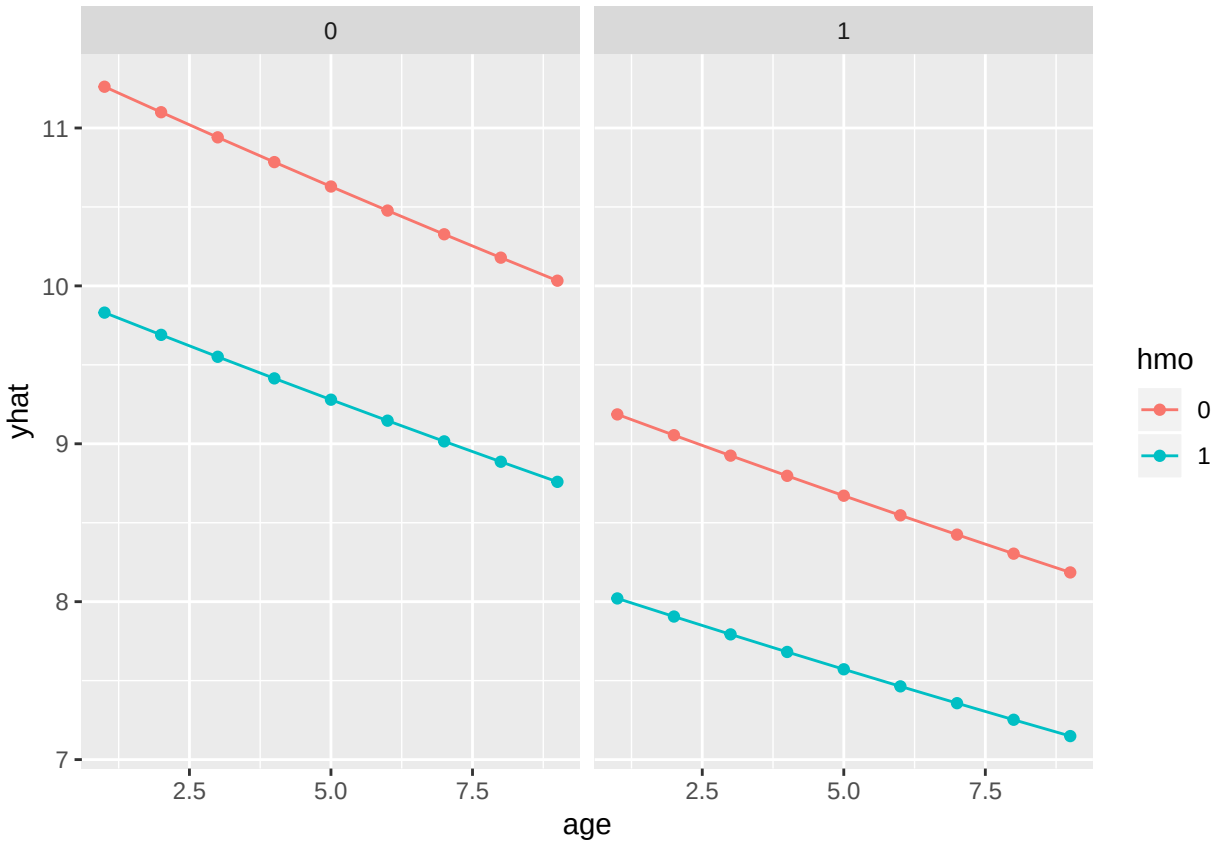
The results are consistent with what we initially viewed graphically, age does not have a significant effect, *but hmo and died both do.*

Poisson Prediction Plots

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 is the mean predicted value (i.e., for the estimate, not a new individual).

```
## 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)
```