Math 463 HW7

Ian McConachie

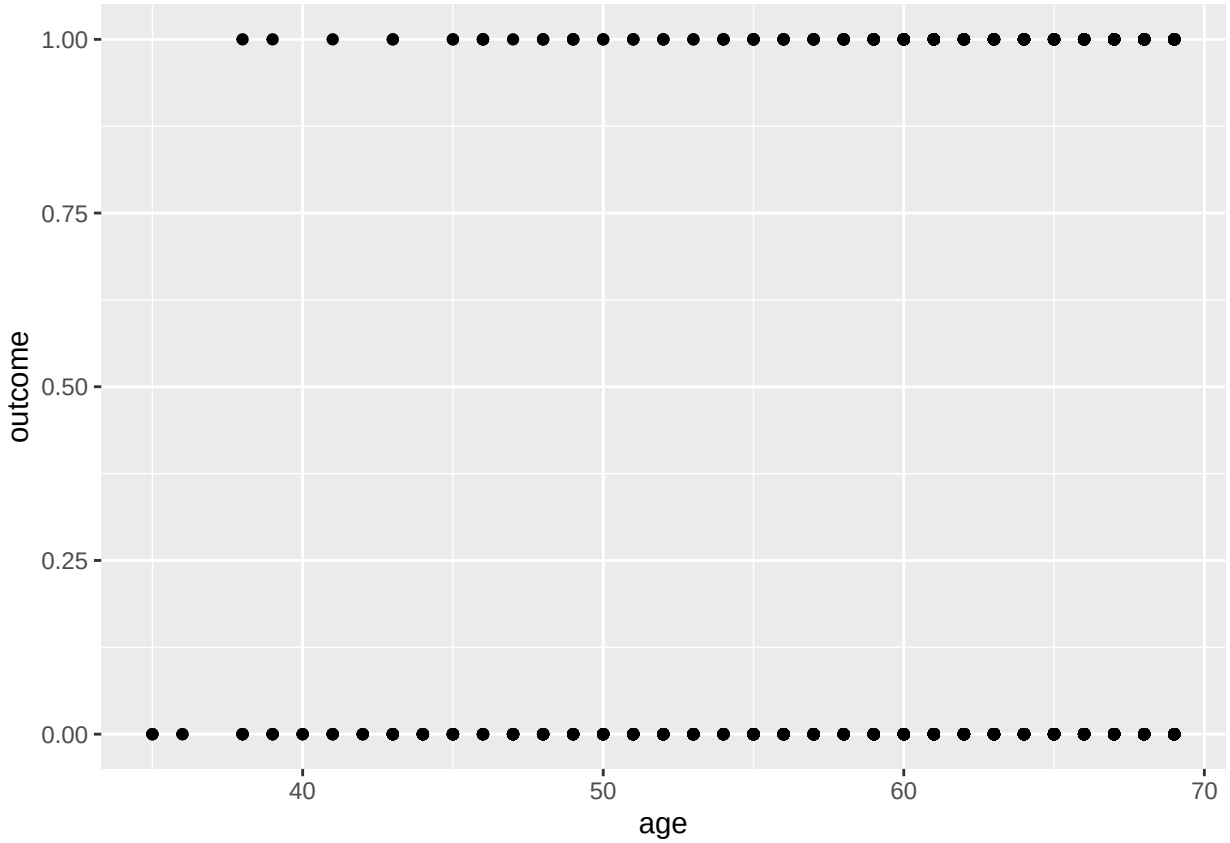
Heart Disease Data

We can start by importing the data into our workspace from the University's servers.

```
myfile <- "https://pages.uoregon.edu/dlevin/DATA/heart.csv"
heart <- read.csv(myfile, header=T)
heart$outcome <- ifelse(heart$outcome=='dead',1,0)
```

The first thing we can do is graph the data so we can get a general idea of the shape of the data. For this data, the response variable we are interested in is the "outcome" variable which says whether the patient with heart disease is currently either alive or dead (1 being dead and 0 being alive). The primary predictor we'll be looking at for this response is age. So, in the below graph we will display age on the x axis and outcome on the y (where outcome has been transformed so that 1 indicates the patient is dead and 0 indicates they are alive).

```
ggplot(data=heart, aes(x=age,y=outcome)) + geom_point()
```



This graph suggests that a logistic regression fit is probably the best course to take for building a model to fit to the data—we can see that a logistic curve may be a good fit because the response is a binary indicator and the probability of different binary outcome is different at different values of the age variable.

Given this, we can start by generating a logistic regression model that has only age as a predictor for the patient's outcome. The R code below creates such a logistic regression model using a Bayesian approach.

```
heartfit_1 = stan_glm(outcome ~ age, data=heart, family=binomial(link = "logit"), refresh=0)
print(heartfit_1)
```

```
## stan_glm
##  family:       binomial [logit]
##  formula:      outcome ~ age
##  observations: 1295
##  predictors:   2
## ------
##             Median MAD_SD
## (Intercept) -4.0    0.7
## age          0.0    0.0
##
## ------
## * For help interpreting the printed output see ?print.stanreg
## * For info on the priors used see ?prior_summary.stanreg
```

```
print(coef(heartfit_1))
```

```
## (Intercept)         age
## -4.00658245  0.04707458
```

The coefficients that make up this model have been printed above–there is one intercept coefficient and one that corresponds with the age predictor. What these values tell us is that applying an inverse logistic transformation to the linear combination of the intercept value and the product of the age variable and the age coefficient gives us an estimated probability of the patient being dead according to this logistic model we have fit to the data.

We can now increase the complexity of this model by including high blood pressure (the highbp variable) and high cholesterol (the hichol variable) as predictors for the outcome of a patient. The R code below generates a model with these 3 predictors.

```
heartfit_2 = stan_glm(outcome ~ age+highbp+hichol, data=heart, family=binomial(link = "logit"), refresh=
print(heartfit_2)
```

```
## stan_glm
##  family:       binomial [logit]
##  formula:      outcome ~ age + highbp + hichol
##  observations: 1295
##  predictors:   6
## ------
##             Median MAD_SD
## (Intercept) -4.4    0.7
## age          0.0    0.0
## highbpnk     3.3    0.4
## highbpy      0.1    0.2
```

```
## hicholnk     0.4    0.2
## hicholy     -0.5    0.2
##
## ------
## * For help interpreting the printed output see ?print.stanreg
## * For info on the priors used see ?prior_summary.stanreg
```

```
print(coef(heartfit_2))
```

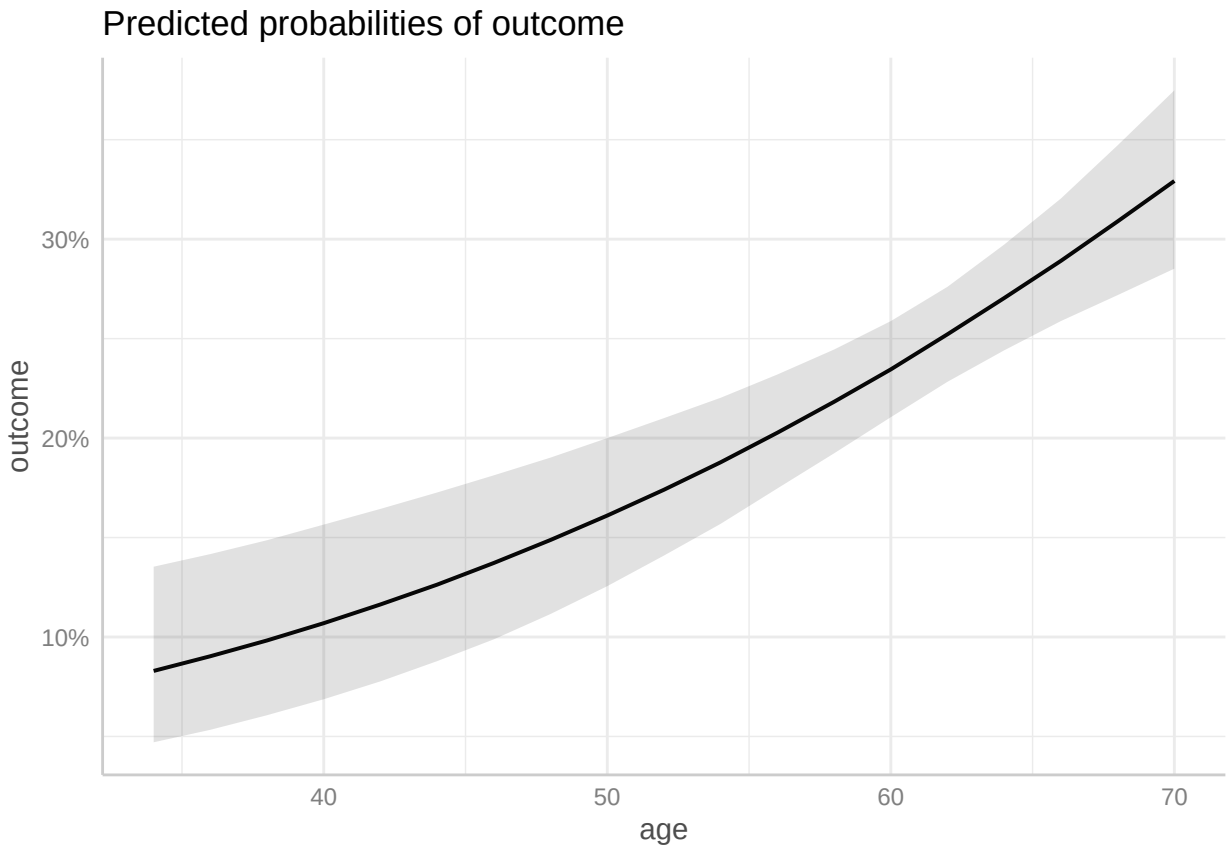
```
## (Intercept)         age    highbpnk     highbpy    hicholnk     hicholy
## -4.35610879  0.04918833  3.25909823  0.11545828  0.38758358 -0.48707952
```

In the above model, the coefficient for age and the intercept value can be interpreted in much the same way they were with heartfit_1. However, the high blood pressure and high cholesterol variables are categorical, so their coefficients have a slightly different meaning. Note that for these categorical variables, there must be a "default" category which is "no" for both variables. This means that if the patient has no high blood pressure or high cholesterol, we would calculate the linear combination (eta) the same way we did in our interpretation of heartfit_1. If the patient has high BP, then we would add the "highbpy" coefficient to the linear combination—this also applies for patients with high cholesterol (i.e. adding hicholy), and patients where either variable is "not known" or "nk" (i.e. adding highbpnk and/or hicholnk).

Just like above, eta in this regression is put through an inverse logistic transformation to end up with a probability p that is the predicted expected probability that a patient will be alive.

For the first model, we can display the predicted probability as a function of the one variable we used as a predictor: age. The R code below generates a graph showing this relation according to the first model.

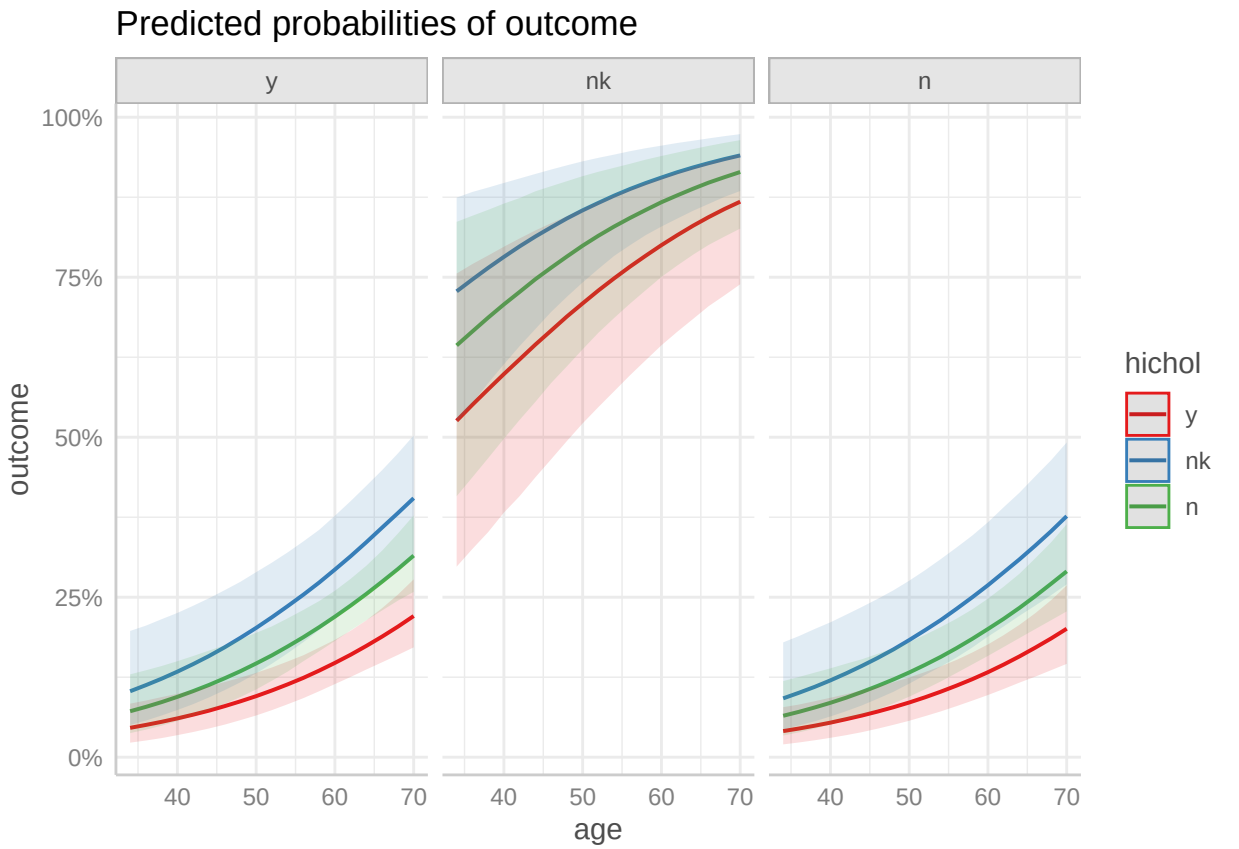
```
library(ggeffects)
plot(ggpredict(heartfit_1,terms=c("age")))
```



The graph above clearly shows an positive correlation between age and the predicted probability of the patient being dead. This makes sense intuitively because as a person gets older, we would expect that they are more likely to be die due to the biological effects of aging.

Now we can create a similar visualization for the second model, but this time we will include more variables as the second model has 3 predictors instead of just 1. We can visualize the effect of all three variables on the predicted response probability with the following R code.

```
library(ggeffects)
plot(ggpredict(heartfit_2,terms=c("age", "hichol", "highbp")))
```



This graph once again shows the relationship between the age of patients and the predicted probability that they are dead. Like above, all of these graphs show a positive correlation between age and the response probability–which makes sense because the intuitive reasoning given above still holds.

There are 3 different graphs for the different categories of high bloood pressure. Among these graphs, we see that the patient having an unknown blood pressure status increases their chance of being dead across the board. This could possibly have something to do with how this data was collected or be related to some unseen variable like frequency of doctor visits or socioeconomic status that is strongly correlated with the researchers not knowing the blood pressure status of the patient.

The high cholesterol categories are displayed visually by the three different lines in each of the graphs. We can note that the shaded regions around each line represent some confidence/uncertainty interval associated with the regression fit. This is of particular importance here because we can see that there is quite a bit of overlap in these shaded regions and most lines have another line in their shaded region. This suggests that the differences in response between cholesterol categories could be caused by random variation rather than cholesterol having a true impact on the probability that a patient is dead.

Now we can compare these two models with cross validation. Specifically, we'll use the standard cross validation technique of splitting the data into two sets: one for training the model and one for validating the model. The R code below splits up our data set and trains each model we generated above on the data that is meant for creating the model.

```
ind <- sample(1:1295,295,replace=FALSE)
hearttest <- heart[ind,]
hearttrain <- heart[-ind,]
modtrain1 <- stan_glm(outcome~age, family=binomial(link="logit"), data=hearttrain, refresh=0)
modtrain2 <- stan_glm(outcome~age+highbp+hichol, family=binomial(link="logit"), data=hearttrain, refresh
```

Now that we have both our models trained, we can generate predicted binary response values for each datapoint and then compare that to the observed binary outcome responses that we have from the data provided. We can make this comparison in the form of two confusion tables (one for each model) which show the predicted binary categories in the columns of the table and the observed binary value in the rows.

```
p1 <- predict(modtrain1, newdata=hearttest, type="response")
po1 <- as.numeric(p1>0.5)
table(hearttest$outcome,po1)
```

```
##    po1
##       0
##   0 210
##   1  85
```

```
p2 <- predict(modtrain2, newdata=hearttest, type="response")
po2 <- as.numeric(p2>0.5)
table(hearttest$outcome,po2)
```

```
##    po2
##       0   1
##   0 209   1
##   1  66  19
```

The above confusion tables will be different every time the code is run because of the pseduo-random processes behind the sample and predict functions. However, what I have noticed after running the code a few times is that these tables generally indicate that the second, more complex, model is better at predicting the binary response value. We can see this in the proportion of the sum of the top left and bottom right cells to the sum of the top right and bottom left cells; the former counts the number of correct predictions and the latter counts the number of incorrect predictions.

Every time I have ran this code, I get that the smaller model predicts 0 for all response values—meaning it is a very biased predictor. Adding the additional variables does improve predictive performance, but I would argue that this is not because of the difference between people with or without high blood pressure and/or high cholesterol, rather the effect of being in the "not known" category in these variables.

We can see from the effect plots above that there is not much of difference between the "y" and "n" categories in either highbp or hichol, but the "nk" category has a significantly higher probability of being dead across the board. This is likely caused by some quirk in how the data was collected (e.g. people in the "nk" category were already dead) or by some correlation to another unseen variable like access to health care and frequency of doctor visits. Essentially, our findings suggest we need to take another look at our data and examine our research methods before making any interpretations of these models.

13.1: Fitting logistic regression to data

First, we can write a little R code to import the data into our workspace and format it in a usable way.

```
myfile <- "https://raw.githubusercontent.com/avehtari/ROS-Examples/master/NES/data/nes.txt"
nes <- read.table(myfile, header=T)
# Making sure categorical variables are classified as categorical
nes$gender <- as.factor(nes$gender)
nes$educ3 <- as.factor(nes$educ3)
nes$ideo7 <- as.factor(nes$ideo7)
nes$partyid7 <- as.factor(nes$partyid7)
nes$race <- as.factor(nes$race)
```

We can then write some code that gets rid of the datapoints with NA values in the variables we are interested in, namely the variables: rvote, gender, ideo7, educ3, partyid7, and race.

```
nes_subset <- nes[ , c("rvote", "gender", "ideo7", "educ3", "partyid7", "race")]
nes2 <- nes[complete.cases(nes_subset), ]
```

Note that the throw out of data we did above got rid of a lot of our data (27,527 of the 34,908 data points). This is a trade off between making conclusions off of incomplete data and making conclusions based on less data–different situations call for different choices. Here, I decided to throw out a lot of the data to get rid of the problem of NA values in important predictors; this also fixes some problems later on when we compare the fit between different models.

The problem asks us to fit a logistic regression to the data with the binary indicator of a vote for Bush as the response and sex (all the data has is a gender variable), education, political ideology, ethnicity (all they had is a race variable), and party identification. For many of these predictors, there were several possible variables that could have selected from to be in this model. When possible, the variables that were on a scale of 1 to 7 were selected–although for the gender variable, the only option was on a scale from 1 to 2.

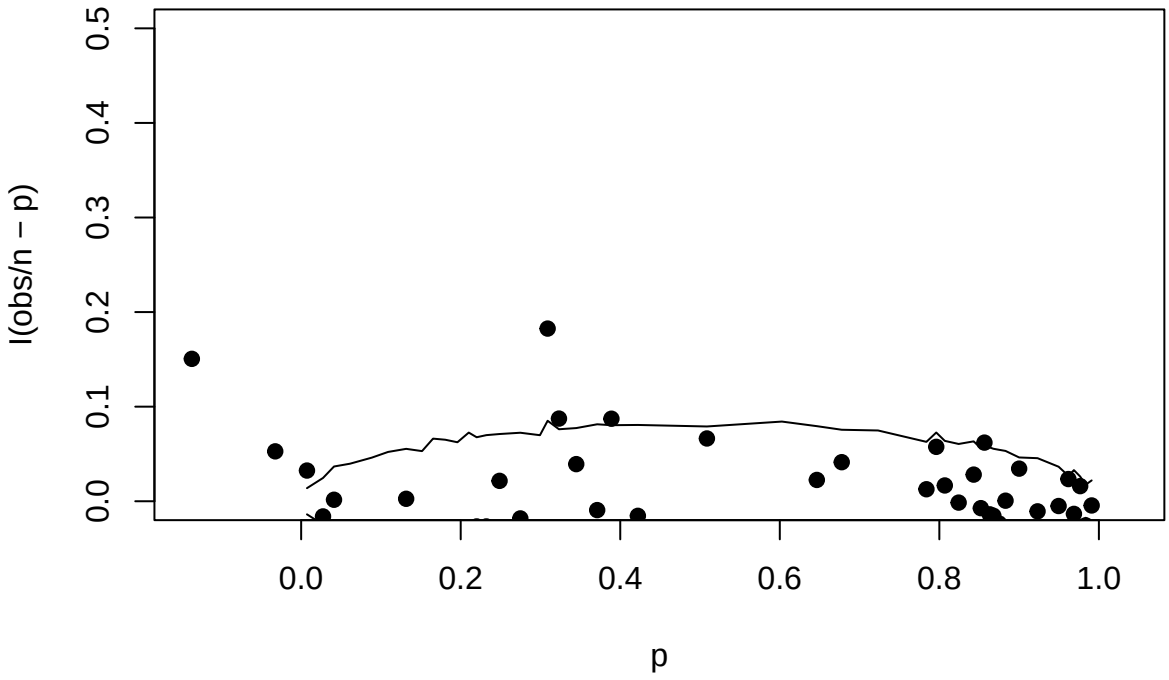
```
votefit_1 <- lm(rvote~gender+ideo7+educ3+partyid7+race, data=nes, family=binomial(link = "logit"))
```

Now that we have that model created, we can generate larger models that have interaction terms as well as main effect variables. We can start by generating a logistic model that has an interaction variable between ideology and party identification–two variables that likely have a lot to do with each other as political parties are supposed to be defined by ideologies. For our next bigger model, we can also include interactions between gender and political party as well as education and political party. These two interaction variables don't interact in as clear of a way as ideology and political party, but there is still some intuitive logic to back up their presence. The R code below generates both of these larger models.

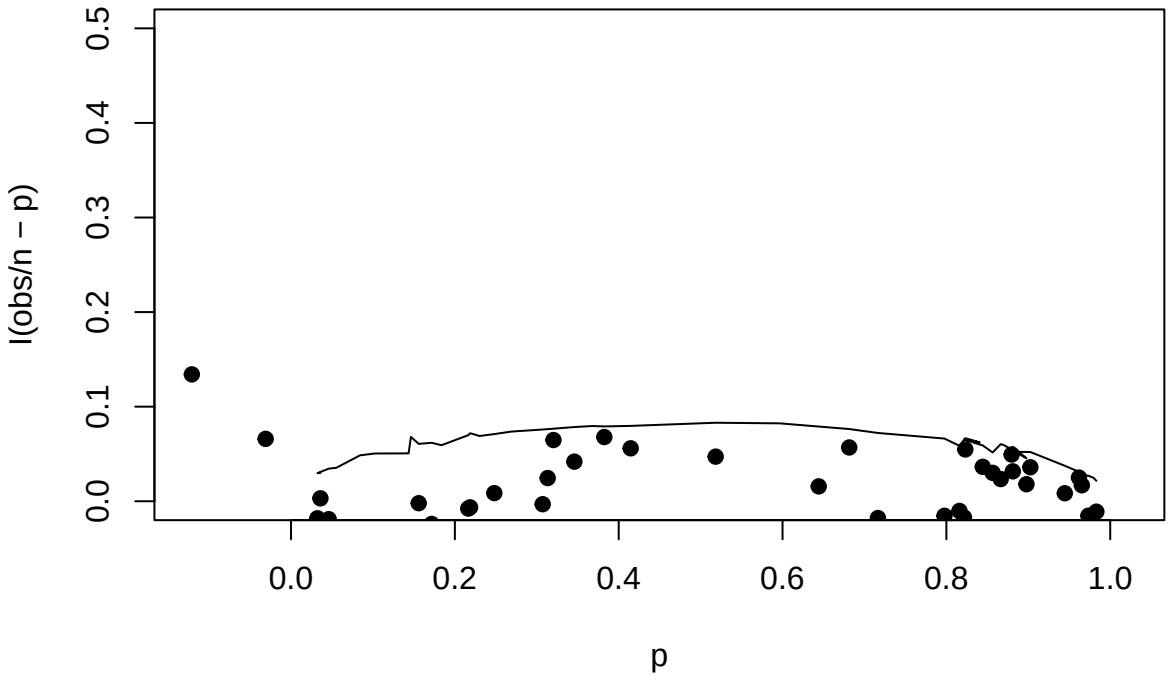
```
votefit_2 <- lm(rvote~gender+ideo7+educ3+partyid7+race+ ideo7:partyid7, data=nes2, family=binomial(link
votefit_3 <- lm(rvote~gender+ideo7+educ3+partyid7+race+ ideo7:partyid7 + gender:partyid7 + educ3:partyi
```

Now, with 3 models to choose from, we can compare the fits and predictive performances of all three to select which one is the best at describing the trends seen in the data provided. The first thing we can do is generate a kind of residual plot for the logistic regression models which splits the data into 50 sections and compares the proportion of observed datapoints that are 1 to the proportion of predicted data points that are 1. We can then plot the difference on the y axis against the fitted value on the x axis. To better analyze these plots, we can also draw a line representing two standard deviations on either side of the expected mean of 0—if more than the acceptable number of residual points fall outside of these lines, this is some indication that the model's fit may not be up to par.

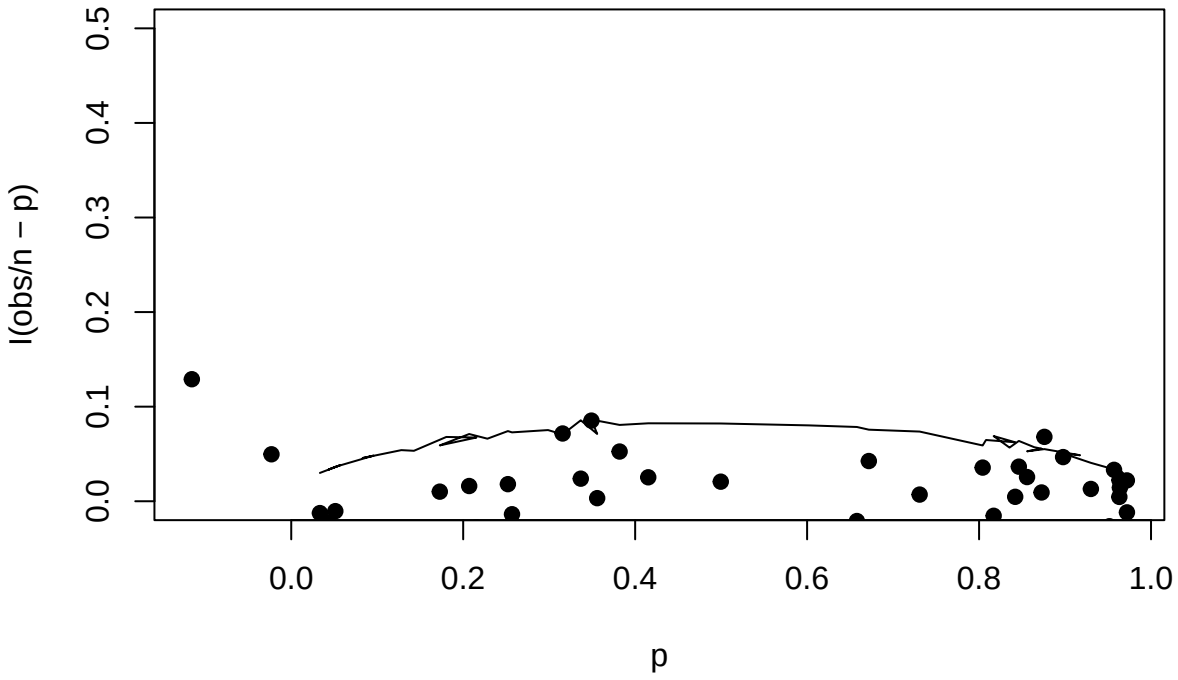
```
fvg <- cut_number(fitted.values(votefit_1),50)
obs <- tapply(nes2$rvote,fvg,sum)
n <- tapply(fitted.values(votefit_1),fvg,length)
p <- tapply(fitted.values(votefit_1),fvg,mean)
plot(p,I(obs/n-p), pch=19, ylim=c(0,0.5))
lines(p,2*sqrt(p*(1-p)/n))
lines(p,-2*sqrt(p*(1-p)/n))
```




```
fvg <- cut_number(fitted.values(votefit_2),50)
obs <- tapply(nes2$rvote,fvg,sum)
n <- tapply(fitted.values(votefit_1),fvg,length)
p <- tapply(fitted.values(votefit_1),fvg,mean)
plot(p,I(obs/n-p), pch=19, ylim=c(0,0.5))
p1 <- seq(0,1,0.02)
lines(p,2*sqrt(p*(1-p)/n))
lines(p,-2*sqrt(p*(1-p)/n))
```




```
fvg <- cut_number(fitted.values(votefit_3),50)
obs <- tapply(nes2$rvote,fvg,sum)
n <- tapply(fitted.values(votefit_1),fvg,length)
p <- tapply(fitted.values(votefit_1),fvg,mean)
plot(p,I(obs/n-p), pch=19, ylim=c(0,0.5))
p1 <- seq(0,1,0.02)
lines(p,2*sqrt(p*(1-p)/n))
lines(p,-2*sqrt(p*(1-p)/n))
```



Note that because of the pseudo-random processes involved in the sample function, the results of this analysis will be different each time. Analyzing these graphs means that we have to count the discrete number of data points that fall outside our acceptable range. To do this, I just described the plots from the first time I ran them, so any specific information below refers to those plots.

From these 3 plots, we can see that pretty much all three models have acceptable fits to some degree. However, the first model has 7 points outside the acceptable range of two standard deviations from zero. This is more than the 2.5 (rounded up to 3) that we would expect to be outside the range of two standard deviations on either side of the mean. The third model also exceeds this expected limit because it has 4 points outside of our line-defined range—but it is also important to note that the data points to not fall as far outside the range as they do with the first model.

The only model that meets our expectations for the expected number of points to fall in the range of two standard deviations from the mean is votefit_2 which has only 2 data points outside the range. Therefore, this initial residual analysis suggests that the second model we generated has the best fit.

We can continue our analysis of these 3 models by taking a look at their predictive performance by splitting the data and examining confusion tables as a form of cross validation.

```
# First we split up the data into a training set and a validation set
ind <- sample(1:7381,1500,replace=FALSE)
nes_test <- nes2[ind,]
nes_train <- nes2[-ind,]

# Then we fit our 3 models to the training set
tr_fit1 <- lm(rvote~gender+ideo7+educ3+partyid7+race, data=nes_train, family=binomial(link = "logit"))
tr_fit2 <- lm(rvote~gender+ideo7+educ3+partyid7+race+ ideo7:partyid7, data=nes_train, family=binomial(l
tr_fit3 <- lm(rvote~gender+ideo7+educ3+partyid7+race+ ideo7:partyid7 + gender:partyid7 + educ3:partyid7
```

```
# Then we compare the predicted responses to the observed and generate 3 confusion tables
p1 <- predict(tr_fit1, newdata=nes_test, type="response")
po1 <- as.numeric(p1>0.5)
table(nes_test$rvote,po1)
```

```
##    po1
##       0   1
##   0 608 110
##   1 145 637
```

```
p2 <- predict(tr_fit2, newdata=nes_test, type="response")
po2 <- as.numeric(p2>0.5)
table(nes_test$rvote,po2)
```

```
##    po2
##       0   1
##   0 603 115
##   1 138 644
```

```
p3 <- predict(tr_fit3, newdata=nes_test, type="response")
po3 <- as.numeric(p3>0.5)
table(nes_test$rvote,po3)
```

```
##    po3
##       0   1
##   0 603 115
##   1 140 642
```

The above confusion tables will have different results every time because of the pseudo-randomness in the sample function, but every time I have ran this code, I end up with three confusion tables that do not look all that different. This tells us that the predictive performance may not be notably different between the three models. Now we only have the "residual" analysis we did earlier meaning we should go with the model we favored in that analysis: votefit_2.

We now have our chosen model and can look at the coefficients to get an idea of which variables have the most importance in the model.

```
print(summary(votefit_2))
```

```
##
## Call:
## lm(formula = rvote ~ gender + ideo7 + educ3 + partyid7 + race +
##     ideo7:partyid7, data = nes2, family = binomial(link = "logit"))
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -1.04930 -0.17109  0.02714  0.15343  1.10751
##
## Coefficients:
##                   Estimate Std. Error t value Pr(>|t|)
## (Intercept)       0.141879   0.047131   3.010 0.002619 **
```

```
## gender2          -0.003467   0.008354  -0.415 0.678179
## ideo72           -0.044105   0.047505  -0.928 0.353216
## ideo73           -0.054444   0.049857  -1.092 0.274867
## ideo74            0.035398   0.047520   0.745 0.456347
## ideo75            0.113863   0.055137   2.065 0.038949 *
## ideo76            0.090210   0.050343   1.792 0.073191 .
## ideo77            0.105840   0.086966   1.217 0.223633
## educ32            0.012622   0.022615   0.558 0.576766
## educ33           -0.046282   0.019183  -2.413 0.015862 *
## educ34            0.051416   0.021782   2.360 0.018278 *
## educ35           -0.024555   0.019116  -1.285 0.198999
## educ36           -0.033667   0.019827  -1.698 0.089543 .
## educ37           -0.054887   0.022133  -2.480 0.013166 *
## partyid72        -0.078622   0.080679  -0.975 0.329839
## partyid73         0.008942   0.085482   0.105 0.916692
## partyid74        -0.020168   0.139903  -0.144 0.885381
## partyid75         0.427029   0.252427   1.692 0.090747 .
## partyid76         0.886142   0.353917   2.504 0.012308 *
## partyid77         0.648864   0.181008   3.585 0.000340 ***
## race2            -0.177263   0.015085 -11.751  < 2e-16 ***
## race3            -0.005741   0.041357  -0.139 0.889599
## race4            -0.021115   0.029013  -0.728 0.466767
## race5            -0.066978   0.022691  -2.952 0.003170 **
## ideo72:partyid72  0.144371   0.086207   1.675 0.094035 .
## ideo73:partyid72  0.199415   0.086877   2.295 0.021740 *
## ideo74:partyid72  0.277107   0.084046   3.297 0.000982 ***
## ideo75:partyid72  0.262836   0.090332   2.910 0.003629 **
## ideo76:partyid72  0.295069   0.087947   3.355 0.000797 ***
## ideo77:partyid72  0.247271   0.147082   1.681 0.092769 .
## ideo72:partyid73  0.005210   0.091541   0.057 0.954616
## ideo73:partyid73  0.069623   0.092368   0.754 0.451019
## ideo74:partyid73  0.075375   0.089636   0.841 0.400427
## ideo75:partyid73  0.123727   0.097922   1.264 0.206441
## ideo76:partyid73  0.102233   0.096558   1.059 0.289740
## ideo77:partyid73  0.089375   0.158878   0.563 0.573769
## ideo72:partyid74  0.304944   0.151157   2.017 0.043690 *
## ideo73:partyid74  0.541243   0.149489   3.621 0.000296 ***
## ideo74:partyid74  0.472099   0.143062   3.300 0.000972 ***
## ideo75:partyid74  0.472723   0.148588   3.181 0.001472 **
## ideo76:partyid74  0.489645   0.147801   3.313 0.000928 ***
## ideo77:partyid74  0.482003   0.197444   2.441 0.014662 *
## ideo72:partyid75  0.089604   0.259687   0.345 0.730069
## ideo73:partyid75  0.191131   0.257580   0.742 0.458095
## ideo74:partyid75  0.259675   0.253978   1.022 0.306610
## ideo75:partyid75  0.160206   0.255472   0.627 0.530613
## ideo76:partyid75  0.302973   0.254603   1.190 0.234093
## ideo77:partyid75  0.272575   0.270570   1.007 0.313772
## ideo72:partyid76 -0.225047   0.360298  -0.625 0.532244
## ideo73:partyid76 -0.263886   0.356873  -0.739 0.459664
## ideo74:partyid76 -0.233842   0.354837  -0.659 0.509910
## ideo75:partyid76 -0.246088   0.355918  -0.691 0.489326
## ideo76:partyid76 -0.221913   0.355271  -0.625 0.532232
## ideo77:partyid76 -0.230160   0.367203  -0.627 0.530814
## ideo72:partyid77  0.208396   0.194001   1.074 0.282767
```

```
## ideo73:partyid77  0.236317   0.192531   1.227 0.219702
## ideo74:partyid77  0.130226   0.183799   0.709 0.478641
## ideo75:partyid77  0.096747   0.185531   0.521 0.602062
## ideo76:partyid77  0.119926   0.183375   0.654 0.513135
## ideo77:partyid77  0.097830   0.199724   0.490 0.624271
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.351 on 7321 degrees of freedom
## Multiple R-squared:  0.5108, Adjusted R-squared:  0.5068
## F-statistic: 129.6 on 59 and 7321 DF,  p-value: < 2.2e-16
```

As you can see from the above printout, there are a lot of coefficients we have to deal with when we have 5 predictors and an interaction variable. We can get some indication of which variables are the "most important" in the model from the Pr(>|t|) column in the far right of the summary which tells us the probability under the null hypothesis (H0 is the coefficient is 0) that we observe an estimated coefficient value that is as or more extreme than the one we observed. This is a good statistic to look at because it doesn't only take into account the magnitude of the coefficient, but also the standard error–so we aren't disproportionately placing importance on variables with large ranges of random variation.

In the above summary, we see that most of the predictors have at least one indicator variable with a somewhat significant effect on the probability of the person voting for Bush. However, among these variables we see that gender likely has little impact in the model because it has a low-magnitude coefficient with a comparatively large standard error. The education and race variables have a similar scenario to a lesser degree indicating that they may also be less crucial to the model.

The predictors we see having the biggest impact on the linear combination, and therefore the predicted response generated by transforming the linear combination, are the party ID and the interaction variable between party and ideology. The indicator coefficients for these predictors often have large values with comparatively small standard errors telling us that they have a greater proportional impact on eta.

13.3: Understanding logistic regression coefficients

For this problem, we have been given the following linear model that uses the economic growth rate in a given year i (g_i) as a predictor for the response variable F_i which represents the percent vote going to the incumbent party in year i. Note that in below equation, epsilon represents the error in our observed values which is assumed to be normally distributed with a mean of 0 and a standard deviation of 3.8.

$$F_i = 46.2 + 3.1 \times g_i + \epsilon_i$$

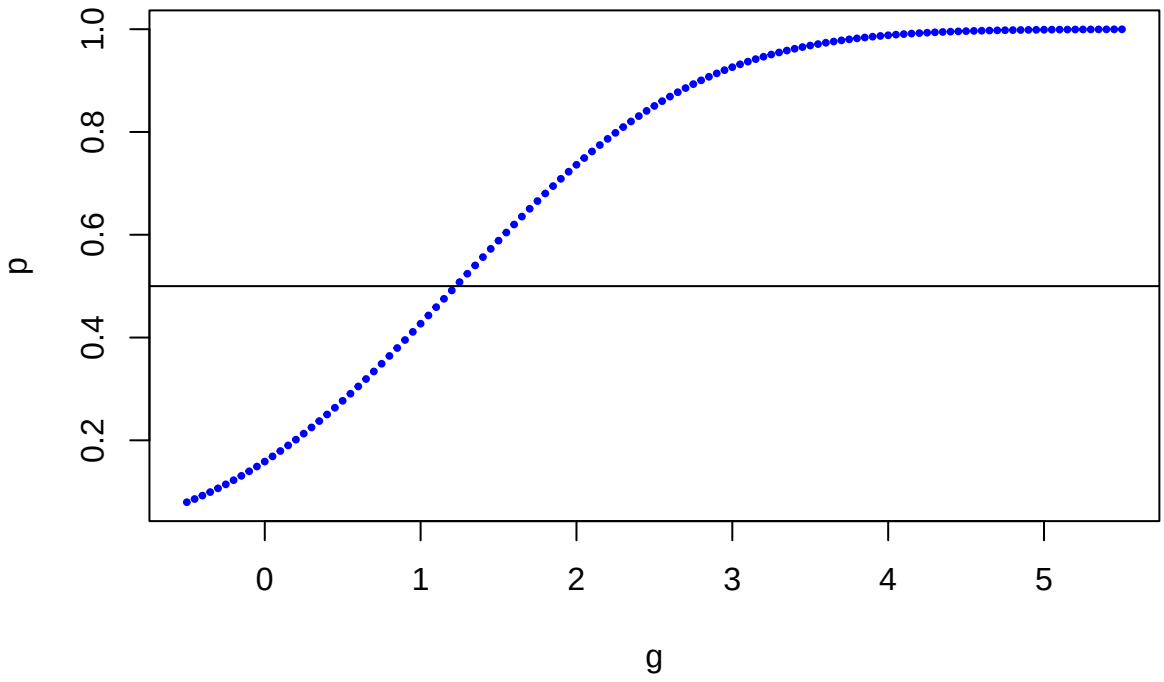
From this model, we want to generate a logistic model where our binary indicator is whether the incumbent won more than 50% of the vote (and therefore won the election). In other words, we want to generate a model with the following form:

$$Pr(vote > 50) = logit^{-1}(a + b * (growth))$$

To estimate the coefficients for such a model, we can start by estimating the probability that the incumbent party gets more than 50% of the vote for different values of growth between -0.5 and 5.5 [going by increments of 0.05]. The R code below, makes such estimates and stores them in the vector p.

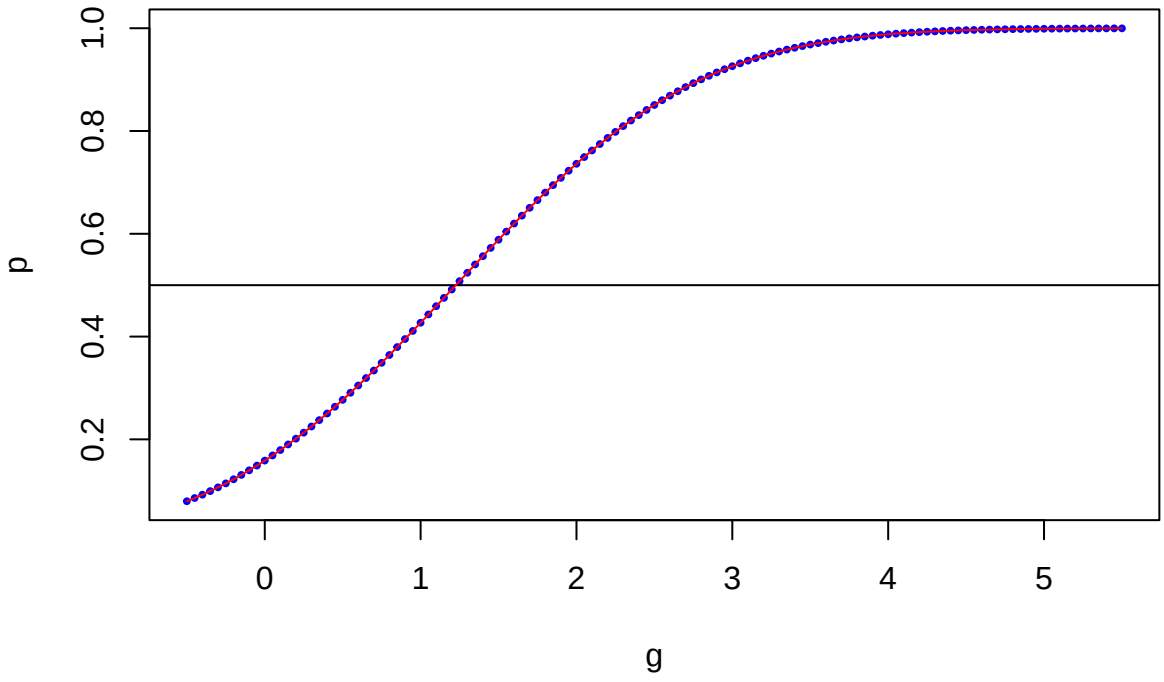
Now we can plot the estimates and draw a horizontal line at probability 0.5—which will come into play later when we get into the divide-by-4 rule. The code below generates such a graph and we can see visually that it roughly follows a logistic curve.

```
plot(g,p, pch=19, cex=0.4, type="p", col="blue")
abline(h=0.5)
```



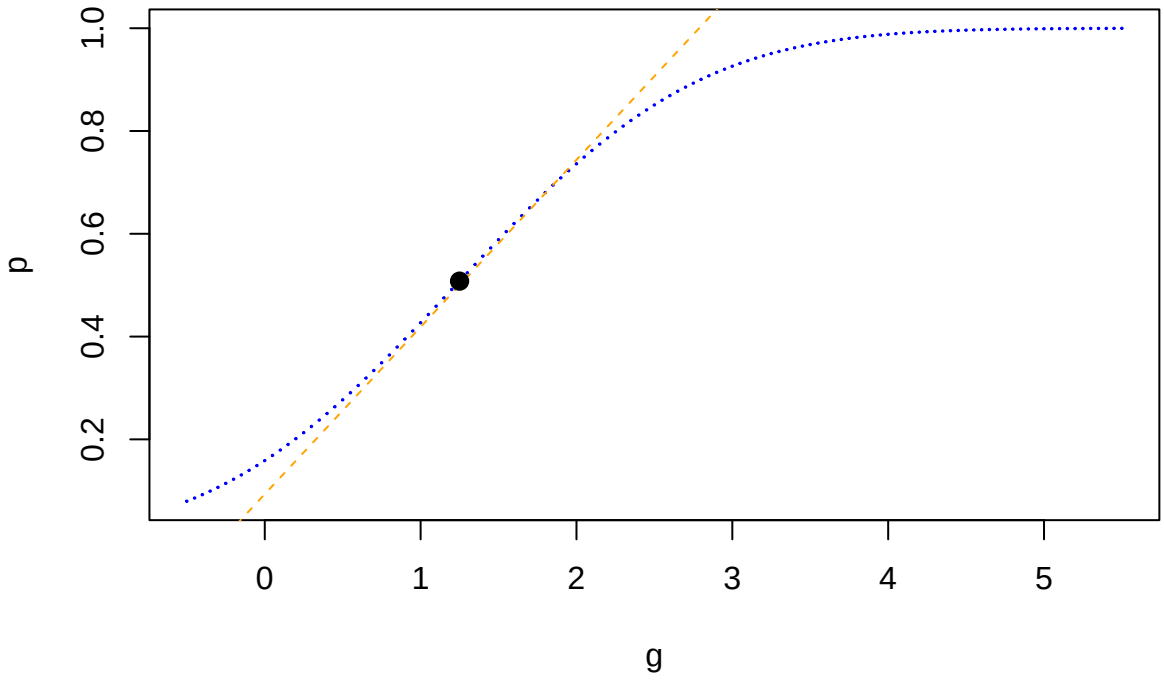
The above visualization strongly suggests that this relationship in the data follows a logistic trend, but we can further see this by connecting the data points in this graph and observing that the resulting line looks very similar to a logistic curve. The R code below generates a plot with the same data points as above (in blue), but this time with a line connecting them (in red).

```
plot(g,p, pch=19, cex=0.4, type="p", col="blue")
abline(h=0.5)
lines(g[order(g)], p[order(g)], xlim=range(g), ylim=range(p), pch=19, col="red")
```



The line drawn over the data above confirms for us that our estimates along the growth axis roughly follow a logistic trend which allows us to use the divide-by-four rule to generate an estimate for the coefficients in a logistic model fitted to this data. To do the divide-by-four rule, we will pick the estimate data point that has a probability closest to 0.5 and then generate an estimate for the tangent line at this point using the two closest adjacent points. This tangent line will have a slope m which acts as an estimate of the b coefficient divided by four. The R code below does this tangent line approximation and graphs it along with the estimate data points and drawn logistic curve that we have in the plot above.

```
mi <- which.min(abs(p-0.5))
sl <- (p[mi+1]-p[mi-1])/(g[mi+1]-g[mi-1])
b = 4*sl
gmir <- round(g[mi], digits=2)
br <- round(b,digits=2)
ic <- (0.5-sl*g[mi])
plot(g,p, pch=19,cex=0.1, col="blue")
abline(ic,sl,col="orange",lty=2)
points(g[mi],p[mi], pch=19, cex=1.2)
```

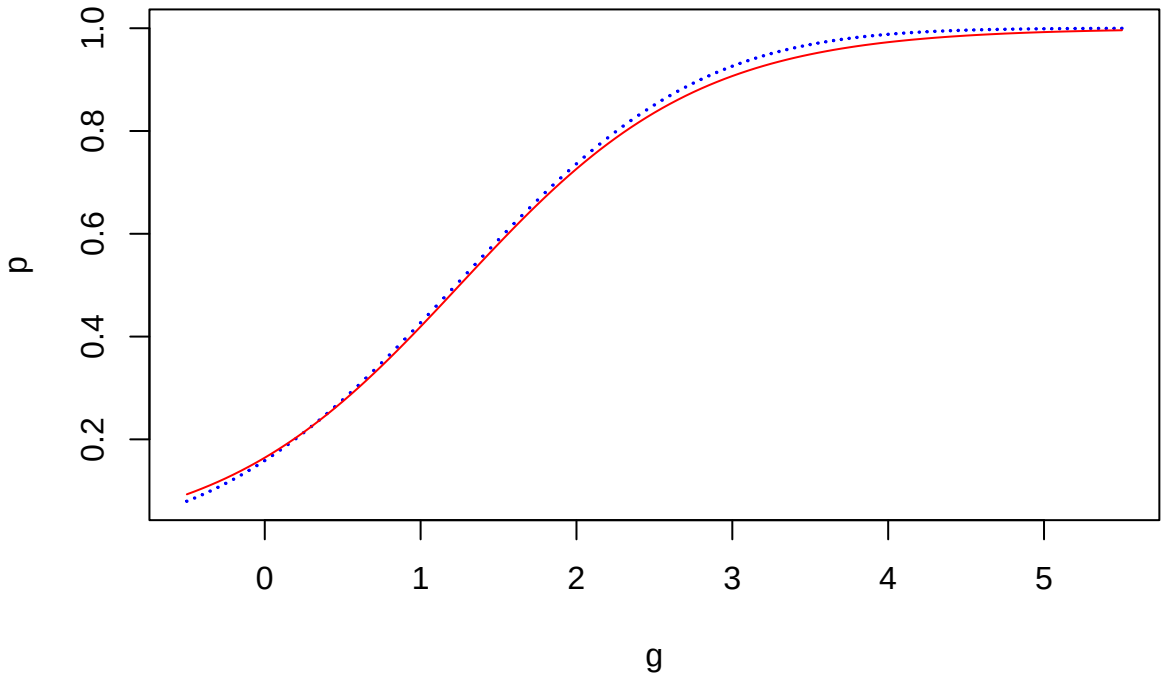


Now that we have this tangent slope value, we can use it to get an estimate for our b coefficient in our logistic regression model. All that is left to do is to produce an estimate for the a coefficient and then we will have a reasonable estimate for a model. To estimate the a coefficient, we can simply take an estimated data point from the set we have already generated and put it in the inverse logistic function along with our estimate for b. Doing this leaves us with an equation where a is the only unknown, which we can solve using algebra. The below R code does this setup and solving for us, producing an estimate for the a coefficient—one that is specifically based on the 0.5 point we used to find an estimate for b above.

```
a <- -b*g[mi]
```

Now we have an estimate for both the a and b coefficient in our logistic regression model, so we can produce the model in full. Now all that remains to do is to graph the estimated logistic trend alongside our estimated data points we generated for our very first plot.

```
fl <- invlogit(a+b*g)
plot(g,p, pch=19,cex=0.1, col="blue")
lines(g,fl,col="red",lty=1)
```



We can see in our plot above that the logistic regression model generated from our estimated coefficients is a good fit to the estimated data points that we are basing the data trends on. This tells us that our estimates are fairly accurate, at least in regards to the information we were given for this problem.

13.11: Building a logistic regression model

First we can write some R code to import the New York apartment data into our work space.

```
myfile <- "https://raw.githubusercontent.com/avehtari/ROS-Examples/master/Rodents/rodents.dat"
rodents <- read.table(myfile, header=T)
# Make race and rodent2 categorical variables
rodents$race <- as.factor(rodents$race)
rodents$rodent2 <- as.factor(rodents$rodent2)
rodents$borough <- as.factor(rodents$borough)
```

Then, we can fit the initial logistic model with race as the sole predictor (represented as enumerated categories) for the response variable rodent2, which is a binary indicator for the presence of rodents.

```
rodentfit_1 = stan_glm(rodent2 ~ race, data=rodents, family=binomial(link = "logit"), refresh=0)
print(rodentfit_1)
```

```
## stan_glm
##  family:       binomial [logit]
##  formula:      rodent2 ~ race
##  observations: 1551
```

```
##  predictors:   7
## ------
##             Median MAD_SD
## (Intercept) -2.2    0.1
## race2        1.4    0.2
## race3        1.6    0.2
## race4        2.0    0.2
## race5        0.8    0.3
## race6        0.3    1.2
## race7        0.1    1.2
##
## ------
## * For help interpreting the printed output see ?print.stanreg
## * For info on the priors used see ?prior_summary.stanreg
```

The information printed above summarizes the logistic model we have fit to this data with race category as a predictor and the presence of rodents as a response. The values shown are estimated coefficients of the linear combination formula. The linear combination has a value eta_i for each data point i that is transformed using the inverse of the link function phi (which here is the inverse logistic function) to give us a predicted probability of the response value rodent2 being 1 (1 meaning rodents are present).

In this model, we are given enumerated race categories (without knowing what race they correspond to) and we have selected race 1 as our "default" category (this could be a problematic concept, but it is a necessary evil for doing this kind of regression on categorical variables). This means that when a data point has race 1, we do not add any additional value to the intercept. For all other races, if the apartment fits into the race category of x, then the racex coefficient is added to the linear combination.

We can build upon our model predicting presence of rodents from race by including the race variables that describe the surrounding community in addition to the variables that describe the particular apartment. These variables include black_Mean and hispanic_Mean which describe the percent of the community that belongs to each of the respective ethnic groups.

We can combine these variables to make a hisbl_Mean variable that describes the percent of the community that is either Black or Hispanic. Note that the way we are making this variable assumes that individuals in the survey cannot be both Black and Hispanic—this is a problematic assumption, but aligns with the way racial groups have been classified in the race variable (i.e. non-overlapping discrete groups). Therefore, this assumption seems to be built into the data collected and cannot be avoided. The R code below makes the hisbl_Mean variable and puts it into the data frame, then fits a Bayesian logistic regression model to the appended data that includes hisbl_Mean as a predictor.

```
rodents$hisbl_Mean <- (rodents$black_Mean + rodents$hispanic_Mean)
rodentfit_2 = stan_glm(rodent2 ~ race+hisbl_Mean, data=rodents, family=binomial(link = "logit"), refresh
print(rodentfit_2)
```

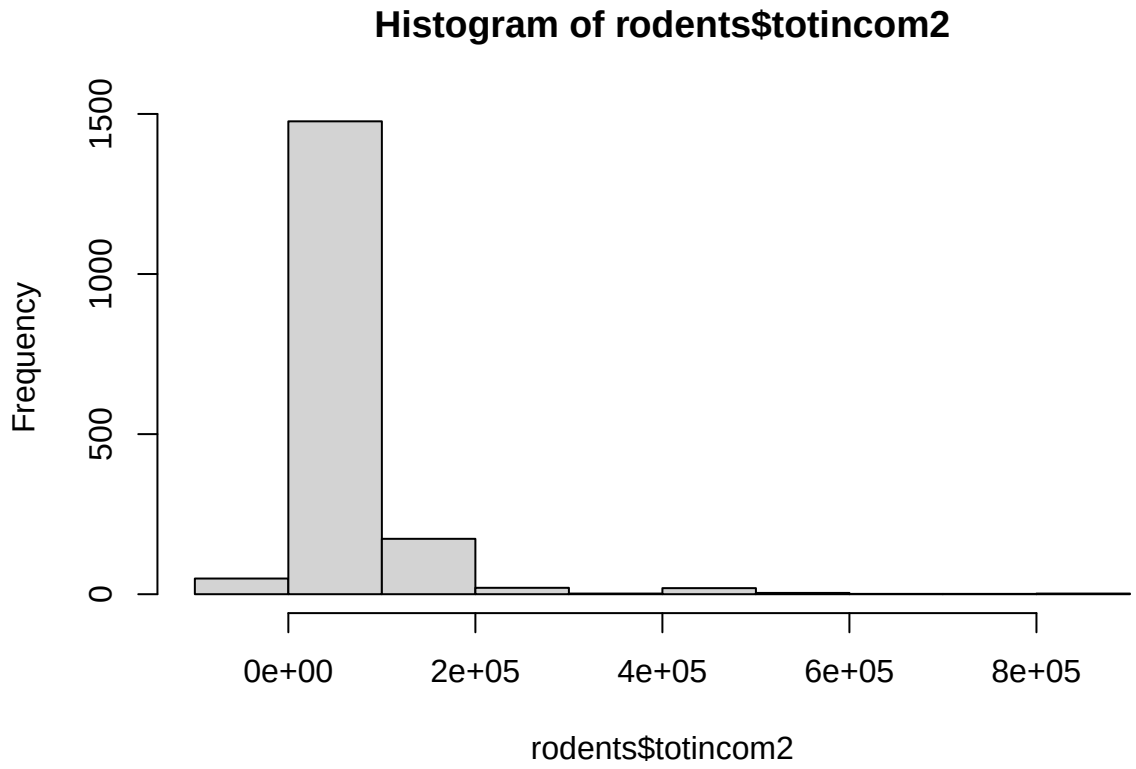
```
## stan_glm
##  family:       binomial [logit]
##  formula:      rodent2 ~ race + hisbl_Mean
##  observations: 1551
##  predictors:   8
## ------
##             Median MAD_SD
## (Intercept) -2.6    0.2
## race2        0.8    0.2
## race3        1.2    0.2
## race4        1.6    0.2
```

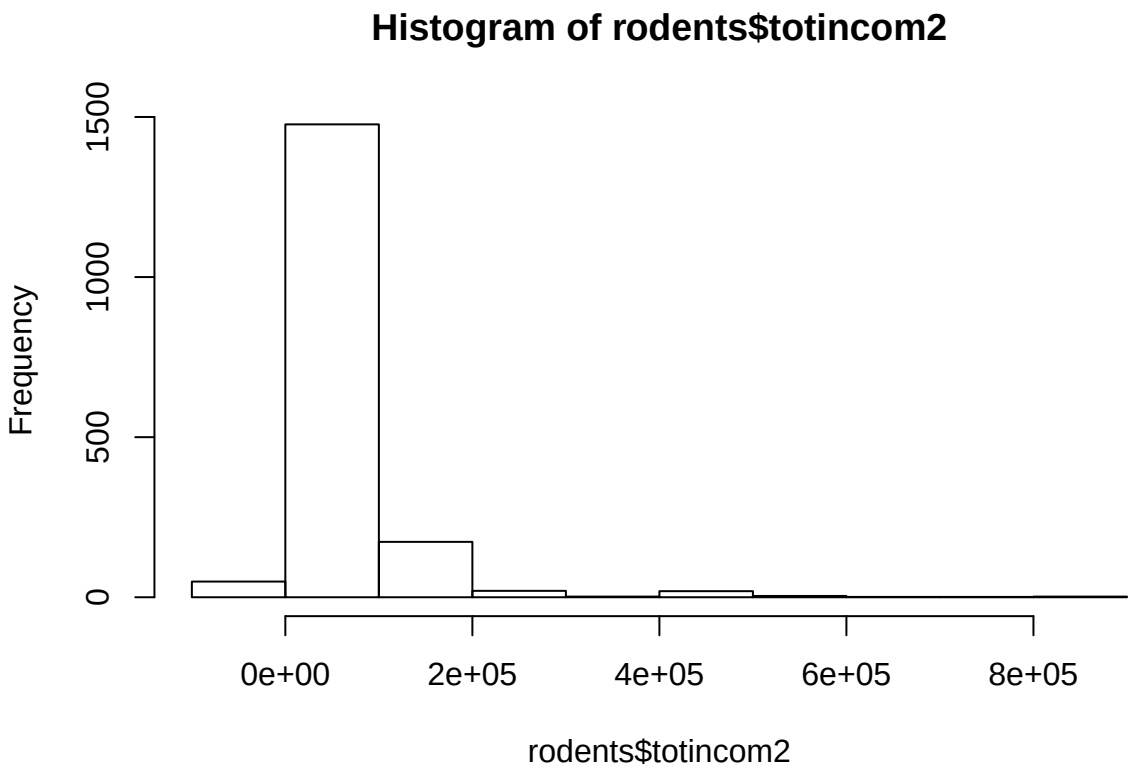
```
## race5        0.7    0.3
## race6       -0.4    1.3
## race7       -0.5    1.2
## hisbl_Mean   1.2    0.3
##
## ------
## * For help interpreting the printed output see ?print.stanreg
## * For info on the priors used see ?prior_summary.stanreg
```

The interpretation of the racex coefficients and the intercept is the same as what we described above for rodentfit_1. However, now we have the additional hisbl_Mean variable which is numeric rather than categorical. This means that hisbl_Mean has no "default' value, rather the estimated coefficient tells us the change in the linear combination for every increase of 1 in the variable hisbl_Mean with all other variables held constant (this change of 1 does not correspond to any real value, rather it exists only in this theoretical statistics space). We still end up with a linear combination eta_i for every datapoint i and do the same inverse-link transformation to get the response value of probability that rodents are present.

Now we can increase the complexity of our model even further by adding more predictors that may give us some indication of whether rodents will be present or not. The following model includes the previous predictors as well as income (from the totincom2 variable), the borough the apartment is in, and percent of the surrounding community that is public housing (from the pubhous_Mean variable). However, before we generate this model, let's take a look at the continuous numeric variable totincom2 in the form of a histogram:

```
plot(hist(rodents$totincom2))
```





You can note from the histogram above, that the total income of these apartments has much of its density close to 0 and then a long tail that extends into the higher ranges of income. This pattern suggests that a log transformation may be useful to make the regression have a better fit and to not have income have a disproportionate effect on the ultimate value of the linear combination due to its scale.

We can also note that some of the apartments have a negative or zero income value, and non-positive values have undefined behavior when taken the log of. This means we need to transform our data to all positive values—we can do this by simply removing the two negative values (which don't really make sense in terms of income) and adding 1 to all other values to avoid the case where income is 0. The R code below does this transformation for us and then generates the aforementioned logistic regression model.

```
rodents <- rodents[-c(9880,450),]
rodents$logincom <- log(rodents$totincom2 + 1)
rodentfit_3 = stan_glm(rodent2 ~ race+hisbl_Mean+logincom+pubhous_Mean+borough, data=rodents, family=bi
print(rodentfit_3)
```

```
## stan_glm
##  family:       binomial [logit]
##  formula:      rodent2 ~ race + hisbl_Mean + logincom + pubhous_Mean + borough
##  observations: 1548
##  predictors:   14
## ------
##              Median MAD_SD
## (Intercept)  -2.1    0.4
## race2         0.9    0.2
## race3         1.2    0.3
```

```
## race4         1.7    0.2
## race5         0.9    0.3
## race6        -0.1    1.3
## race7        -0.7    1.2
## hisbl_Mean    1.1    0.3
## logincom      0.0    0.0
## pubhous_Mean -3.3    1.1
## borough2      0.1    0.2
## borough3     -0.1    0.2
## borough4     -1.0    0.2
## borough5     -1.9    0.6
##
## ------
## * For help interpreting the printed output see ?print.stanreg
## * For info on the priors used see ?prior_summary.stanreg
```

Now we have our largest model with 5 predictors and 14 coefficients (because two predictor variables are categorical). There are really too many coefficients to interpret each of them one by one, but one thing we can pay attention to with these is the MAD_SD which tells us the standard deviation of coefficient estimate. For certain coefficients like the one for race6, the one for logincom, and the one for borough2 have 0 within 2 or even 1 standard deviations of their estimate. This suggests that these coefficients may in actuality be 0 and any observed effect on the response could have come about from random variation rather than true correlation.

This model is made harder to interpret because of its complexity. We have 5 main effects, but interaction terms would make sense for many of these including interactions between hisbl_Mean and the race category and pubhous_Mean and the income variable. In the end, a simpler model may be better because the amount of predictors in this model may make it more difficult to decide the effect of particular variables rather than easier. We could do cross validation to see if these additional predictors even improve the predictive performance of the model, but the problem did not ask for this and I want to go to bed now.