Lab 8: Binomiaal Models in R

TA: Eric Parajon

Today's lab has been adapted from code by Chelsea Estancona, Isabel Laterzo, and Simon Hoellerbauer.

Binomial Models

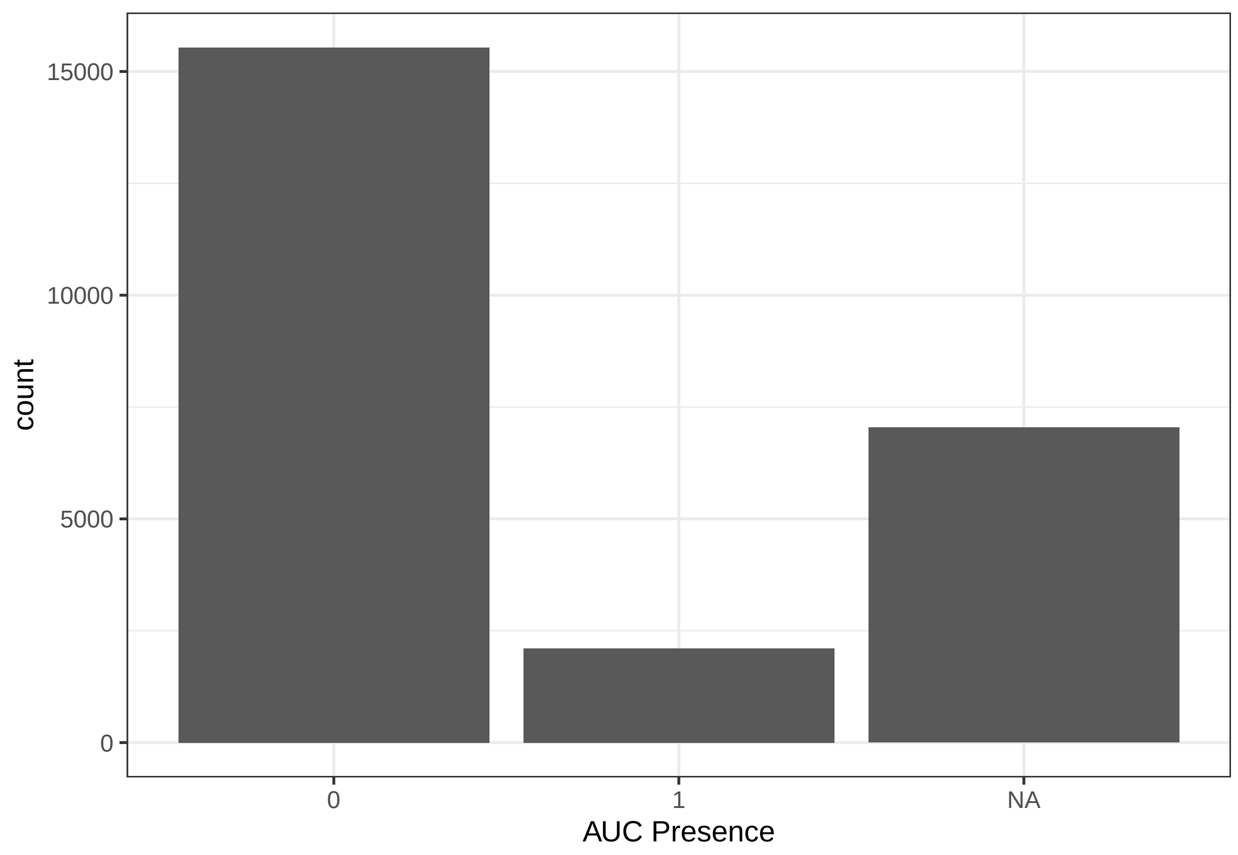
Today we'll work through a fairly standard problem: you have a dichotomous dependent variable and a set of independent variables, and you want to estimate a model. Let's walk through a full example.

Load the `Lab_8_data.dta` file. These are data from a previous project (Chelsea Estancona ('18)'s) about paramilitary groups in Colombia. Our DV is a dichotomous variable coding paramilitary presence in certain municipalities.

```
colombia <- rio::import("Lab_8_data.dta")
```

Let's plot our DV. This is always a good practice for *all* variables (independent or dependent) in your work.

```
ggplot(colombia, aes(x = as.factor(AUC))) +
  geom_bar() +
  labs(x = 'AUC Presence') +
  theme_bw()
```



Note that plotting in this way can also give us information about missingness (which there's a lot of on this variable). Remember - what does R do with our missing data if we leave it to handle it on its own?

Now we're ready to put together a simple model.

```
m1 <- glm(AUC ~ tribut_2_log + tribut_2_log_sq + pop_attacks + disbogota +
            drugs + pop_log, data=colombia, family = binomial(link = 'logit'))

summary(m1)
```

```
##
## Call:
## glm(formula = AUC ~ tribut_2_log + tribut_2_log_sq + pop_attacks +
##     disbogota + drugs + pop_log, family = binomial(link = "logit"),
##     data = colombia)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -3.3055  -0.5430  -0.4036  -0.2646   3.3173
##
## Coefficients:
##                     Estimate Std. Error z value Pr(>|z|)
## (Intercept)       -4.6124385  0.4593284 -10.042  < 2e-16 ***
## tribut_2_log       0.8385542  0.0774072  10.833  < 2e-16 ***
## tribut_2_log_sq   -0.0317784  0.0059871  -5.308 1.11e-07 ***
## pop_attacks        0.3111794  0.0180785  17.213  < 2e-16 ***
## disbogota          0.0017522  0.0001601  10.943  < 2e-16 ***
## drugs            0.0902158  0.0581690   1.551    0.121
## pop_log         -0.1930674  0.0454457  -4.248 2.15e-05 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 9910.7  on 13210  degrees of freedom
## Residual deviance: 8758.6  on 13204  degrees of freedom
##   (11473 observations deleted due to missingness)
## AIC: 8772.6
##
## Number of Fisher Scoring iterations: 6
```

If we wanted to use a probit link:

```
m2 <- glm(AUC ~ ., data = colombia, family = binomial(link = 'probit'))
#what is the . doing above?
summary(m2)
```

```
##
## Call:
## glm(formula = AUC ~ ., family = binomial(link = "probit"), data = colombia)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -3.6899  -0.5438  -0.4035  -0.2562   3.5075
##
## Coefficients:
##                   Estimate Std. Error z value Pr(>|z|)
## (Intercept)     -2.3374214  0.2389584  -9.782  < 2e-16 ***
## tribut_2_log     0.3893985  0.0391568   9.945  < 2e-16 ***
## tribut_2_log_sq -0.0125022  0.0031486  -3.971 7.17e-05 ***
## pop_attacks      0.1773512  0.0104044  17.046  < 2e-16 ***
## disbogota        0.0009930  0.0000876  11.336  < 2e-16 ***
## drugs            0.0542321  0.0314773   1.723   0.0849 .
## pop_log         -0.1102537  0.0243219  -4.533 5.81e-06 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 9910.7  on 13210  degrees of freedom
## Residual deviance: 8742.9  on 13204  degrees of freedom
##   (11473 observations deleted due to missingness)
## AIC: 8756.9
##
## Number of Fisher Scoring iterations: 6
```

The sign (and generally, significance) of our coefficient estimates remain the same. The values differ due to these link functions.

This is all very well and good, but how useful are the coefficient estimates? How do we interpret them?

We might want to use our model to estimate predicted probabilities, as this is a readily-understood measure of how changes in our independent variable affect the probability of observing a certain outcome.

In doing so, we'll want to think about a 'hypothetical case' for which we want to predict: for example, here, we'll calculate the predicted probability of the United Self-Defenses of Colombia or Autodefensas Unidas de Colombia (AUC) presence in a municipality with the mean tax income, mean attacks on population, mean distance from the capitol, drug crops, and the average population.

Note: This is the so-called average case approach.

```
## calculate value of the linear predictor for this variable profile
linpred <- as.numeric(with(colombia, coef(m1)[1] +
                             coef(m1)[2] * mean(tribut_2_log, na.rm = T) +
                             coef(m1)[3] * mean(tribut_2_log, na.rm = T)^2 +
                             coef(m1)[4] * mean(pop_attacks, na.rm = T) +
                             coef(m1)[5] * mean(disbogota, na.rm = T) +
                             coef(m1)[6] * (1) + #could set this to its mode
                             coef(m1)[7] * mean(pop_log, na.rm = T)))

#let's take a look!
arm::invlogit #no parentheses from the arm package
```

```
## function (x)
## {
##     1/(1 + exp(-x))
## }
## <bytecode: 0x000001b109182398>
## <environment: namespace:arm>
```

```
## invlogit lets us transform our log-odds to the range (0,1) - now interpretable
## as a standard probability
invlogit(linpred)
```

```
## [1] 0.1337655
```

We can also do this using the 'predict' function:

```
## create a 'new data' frame: we want to predict the probability for a hypothetical situation.
new_data <- with(colombia, data.frame(tribut_2_log = mean(tribut_2_log, na.rm = T),
                                      tribut_2_log_sq = mean(tribut_2_log, na.rm = T)^2,
                                      pop_attacks = mean(pop_attacks, na.rm = T),
                                      disbogota = mean(disbogota, na.rm = T),
                                      drugs = c(0, 1),
                                      pop_log = mean(pop_log, na.rm = T)))
## note above that I include both possible values for drugs. the 'data.frame()'
## command is flexible and will intelligently duplicate the other values

predict(m1, newdata = new_data, type = 'response')
```

```
##         1         2
## 0.1236530 0.1337655
```

What if we were interested in a *range* of values' predicted probability? We could modify our newdata frame and our predict() function to allow for this.

```
min_tax <- min(colombia$tribut_2_log, na.rm = T)
max_tax <- max(colombia$tribut_2_log, na.rm = T)
min_tax_2 <- min(colombia$tribut_2_log_sq, na.rm = T)
max_tax_2 <- max(colombia$tribut_2_log_sq, na.rm = T)

new_data_2 <- with(colombia,
```

```r
                 data.frame(tribut_2_log = seq(from = min_tax, to = max_tax,
                                               length.out = 100),
                            tribut_2_log_sq = seq(from = min_tax_2, to = max_tax_2,
                                                  length.out = 100),
                            pop_attacks = mean(pop_attacks, na.rm = T),
                            disbogota = mean(disbogota, na.rm = T),
                            drugs = 1, pop_log = mean(pop_log, na.rm = T)))

new_data_3 <- with(colombia,
                 data.frame(tribut_2_log = seq(from = min_tax, to = max_tax,
                                               length.out = 100),
                            tribut_2_log_sq = seq(from = min_tax_2, to = max_tax_2,
                                                  length.out = 100),
                            pop_attacks = mean(pop_attacks, na.rm = T),
                            disbogota = mean(disbogota, na.rm = T),
                            drugs = 0, pop_log = mean(pop_log, na.rm = T)))


predict(m1, newdata = new_data_2, type = 'response') # what is the length of this output?
```

```
##           1           2           3           4           5           6
## 0.003455595 0.003646847 0.003848644 0.004061561 0.004286207 0.004523222
##           7           8           9          10          11          12
## 0.004773280 0.005037092 0.005315406 0.005609012 0.005918739 0.006245462
##          13          14          15          16          17          18
## 0.006590100 0.006953624 0.007337052 0.007741457 0.008167970 0.008617777
##          19          20          21          22          23          24
## 0.009092127 0.009592335 0.010119781 0.010675916 0.011262266 0.011880434
##          25          26          27          28          29          30
## 0.012532101 0.013219036 0.013943093 0.014706218 0.015510453 0.016357939
##          31          32          33          34          35          36
## 0.017250919 0.018191746 0.019182883 0.020226906 0.021326515 0.022484530
##          37          38          39          40          41          42
## 0.023703902 0.024987712 0.026339178 0.027761657 0.029258650 0.030833805
##          43          44          45          46          47          48
## 0.032490922 0.034233951 0.036067003 0.037994344 0.040020402 0.042149766
##          49          50          51          52          53          54
## 0.044387188 0.046737584 0.049206029 0.051797762 0.054518176 0.057372821
##          55          56          57          58          59          60
## 0.060367396 0.063507742 0.066799836 0.070249784 0.073863806 0.077648224
##          61          62          63          64          65          66
## 0.081609451 0.085753973 0.090088326 0.094619083 0.099352822 0.104296105
##          67          68          69          70          71          72
## 0.109455450 0.114837298 0.120447976 0.126293666 0.132380365 0.138713838
##          73          74          75          76          77          78
## 0.145299577 0.152142757 0.159248180 0.166620231 0.174262818 0.182179327
##          79          80          81          82          83          84
## 0.190372557 0.198844674 0.207597148 0.216630705 0.225945269 0.235539914
##          85          86          87          88          89          90
## 0.245412816 0.255561204 0.265981328 0.276668417 0.287616655 0.298819154
##          91          92          93          94          95          96
## 0.310267947 0.321953975 0.333867096 0.345996091 0.358328692 0.370851609
##          97          98          99         100
## 0.383550579 0.396410409 0.409415048 0.422547650
```

```
predict(m2, newdata = new_data_3, type = 'response')
```

```
##           1           2           3           4           5           6 
## 0.001380744 0.001525641 0.001684280 0.001857802 0.002047427 0.002254454 
##           7           8           9          10          11          12 
## 0.002480269 0.002726347 0.002994256 0.003285659 0.003602320 0.003946107 
##          13          14          15          16          17          18 
## 0.004318995 0.004723068 0.005160526 0.005633683 0.006144974 0.006696957 
##          19          20          21          22          23          24 
## 0.007292312 0.007933846 0.008624497 0.009367327 0.010165534 0.011022442 
##          25          26          27          28          29          30 
## 0.011941509 0.012926322 0.013980597 0.015108178 0.016313034 0.017599258 
##          31          32          33          34          35          36 
## 0.018971061 0.020432768 0.021988815 0.023643742 0.025402186 0.027268876 
##          37          38          39          40          41          42 
## 0.029248621 0.031346306 0.033566878 0.035915338 0.038396729 0.041016123 
##          43          44          45          46          47          48 
## 0.043778610 0.046689283 0.049753223 0.052975485 0.056361080 0.059914961 
##          49          50          51          52          53          54 
## 0.063642005 0.067546994 0.071634599 0.075909358 0.080375659 0.085037723 
##          55          56          57          58          59          60 
## 0.089899578 0.094965046 0.100237718 0.105720939 0.111417783 0.117331040 
##          61          62          63          64          65          66 
## 0.123463189 0.129816389 0.136392451 0.143192830 0.150218600 0.157470444 
##          67          68          69          70          71          72 
## 0.164948638 0.172653036 0.180583057 0.188737677 0.197115414 0.205714326 
##          73          74          75          76          77          78 
## 0.214531999 0.223565543 0.232811587 0.242266283 0.251925298 0.261783820 
##          79          80          81          82          83          84 
## 0.271836560 0.282077760 0.292501196 0.303100191 0.313867624 0.324795942 
##          85          86          87          88          89          90 
## 0.335877179 0.347102965 0.358464555 0.369952836 0.381558360 0.393271359 
##          91          92          93          94          95          96 
## 0.405081776 0.416979284 0.428953318 0.440993102 0.453087676 0.465225929 
##          97          98          99         100 
## 0.477396626 0.489588444 0.501789999 0.513989879
```

```
#we can calculate the logit predicted probs by hand by doing 1/(1 + exp(-linpred))
#How could we calculate this for probit?

#We can use pnorm! (this is, after all, the difference between the logit and probit)
all(predict(m2, newdata = new_data_3, type = "response") ==
      pnorm(as.matrix(cbind(1, new_data_3)) %*% coef(m2)))
```

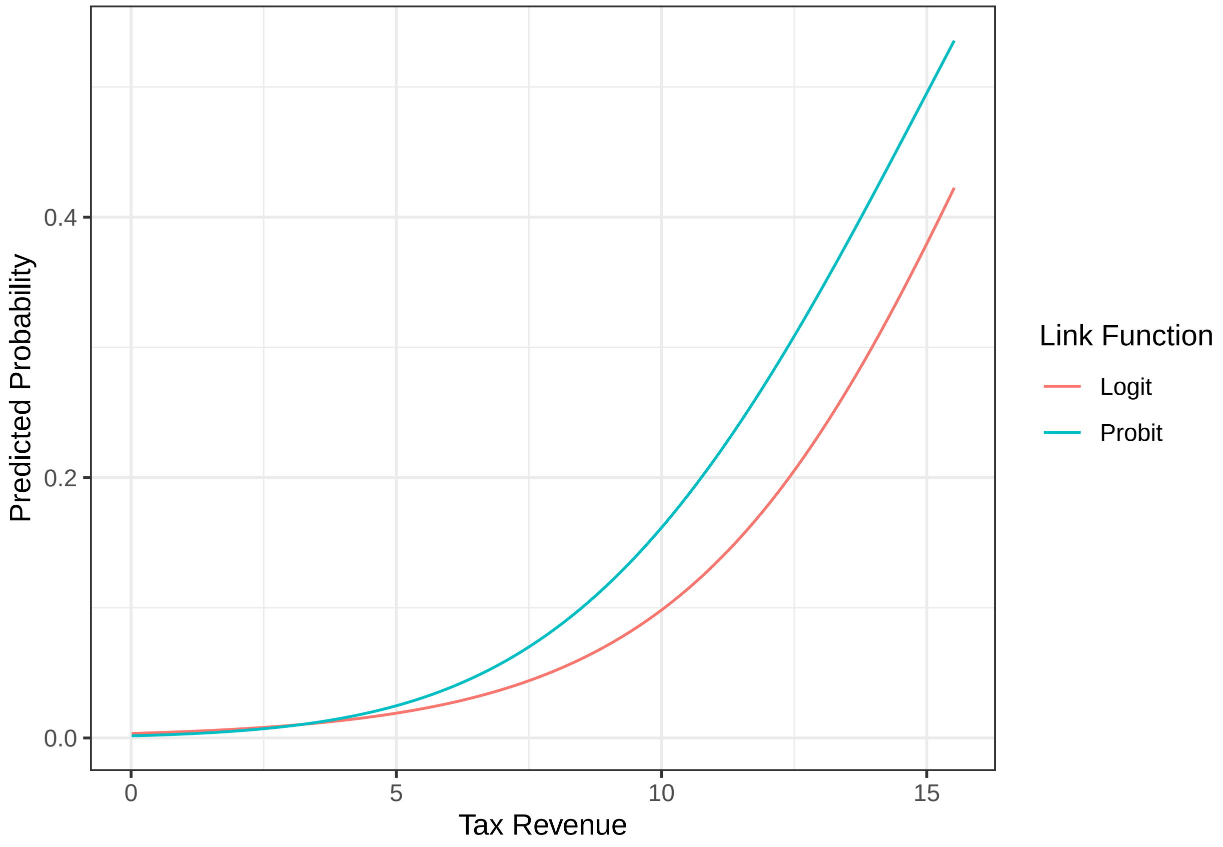
```
## [1] TRUE
```

Note that in this case, it is perfectly ok to vary the tax variable and the the squared tax value from its min to its max, as the square roots of the minimum and maximum squared are equal to the minimum and maximum (unless for some reason you have missingness in one variable at the max and min, which should be unlikely).

We know from the numeric output that logit and probit links produce different coefficient estimates, but how do these differences translate into predicted probabilities? We can plot the predicted probabilities from each model to quickly inspect these differences.

```
## generate predicted probabilities for municipality with drugs and all other
## variables at their means
```

```r
pred_output <- data.frame(Logit = predict(m1, newdata = new_data_2, type = 'response'),
                          Probit = predict(m2, newdata = new_data_2, type = 'response'),
                          tax = new_data_2$tribut_2_log)

plot_pred1 <- gather(pred_output, key = "link", value = "Pred_Probs", -tax)

## plot predicted probabilities
ggplot(data = plot_pred1, aes(x = tax, y = Pred_Probs, color = link)) +
  geom_line() +
  scale_color_discrete(name = 'Link Function') +
  labs(x = 'Tax Revenue', y = 'Predicted Probability') +
  theme_bw()
```



Simulation

We talked about simulations in Lab 4, when we talked about OLS. This time around, we will do similar things, but with non-Normal outcomes, where simulations are even more helpful!

Today, we're going to simulate predicted probabilities for a model with an interaction term, use the distribution of these simulated probabilities to define confidence intervals, and plot our results, using both the average case and the observed-case approaches.

Using Average-Case Approach

The following should all look familiar:

```
## number of simulations for predicted probabilities
N_sim <- 1000

## extract beta hats and construct vcv from our first model
betas <- m1$coef
vcv <- vcov(m1)

## use the mvrnorm function with the betas, vcv, and # of simulations to simulate
## parameter estimates. will have 1,000 estimates of each beta
beta_sims <- mvrnorm(N_sim, betas, vcv)

## create a vector that is a sequence of 100 points, evenly spaced between the
## lowest and highest observed values of tax income in the data set. we have to
## do this for the (tax^2) variable, too!
tax_min <- with(colombia, min(tribut_2_log, na.rm = T))
tax_max <- with(colombia, max(tribut_2_log, na.rm = T))
tax_vec <- seq(from = tax_min, to = tax_max, length = 100)
trans_vec <- (tax_vec^2)
## these will be the x values over which we can plot the predicted probability.
## we're interested in how the predicted probability of AUC presence varies over
## the municipalities' values of tax revenue

## create a hypothetical independent variable as before: mean attacks on population,
## mean distance from the capitol, drug crops set to 1, and the average population.
hyp <- with(colombia, data.frame(intercept = 1,
                                 tribut_2_log = tax_vec,
                                 tribut_2_log_sq = trans_vec,
                                 pop_attacks = mean(pop_attacks, na.rm = T),
                                 disbogota = mean(disbogota, na.rm = T),
                                 drugs = 1,
                                 pop_log = mean(pop_log, na.rm = T)))

#We could also have done this using model.matrix()

## create an empty matrix of values to fill with simulated predicted probabilities
pp_sims <- matrix(NA, nrow = N_sim, ncol = length(tax_vec))
#Using a for() loop to fill wth simulated pred. probs
for(i in 1:N_sim) {

  pp_sims[i, ] <- invlogit(as.matrix(hyp) %*% beta_sims[i, ])

}

#we could also do this with less code:
pp_sims2 <- invlogit(as.matrix(hyp) %*% t(beta_sims))

dim(pp_sims)
```

```
## [1] 1000  100
```

```
dim(pp_sims2)
```

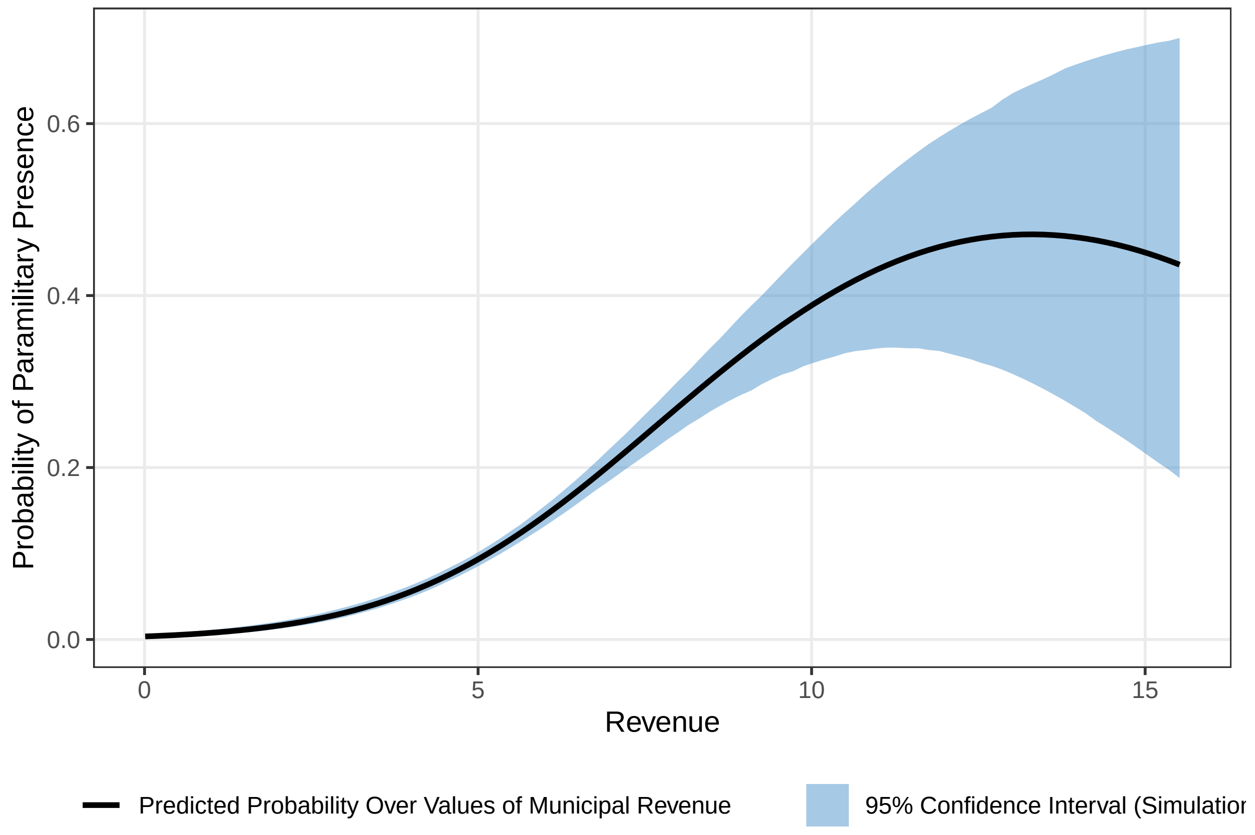
```
## [1]  100 1000
```

```r
all(t(pp_sims) == pp_sims2)
```

```
## [1] TRUE
```

We're not going to worry about confidence intervals today, but below is the code for them as well.

```r
## CIs
pe <- apply(pp_sims, 2, mean)
lo <- apply(pp_sims, 2, quantile, prob = .025)
hi <- apply(pp_sims, 2, quantile, prob = .975)

sim_output <- data.frame(tax = hyp[, 2], pe, lo, hi)

ggplot(sim_output, aes(x = tax, y = pe, ymin = lo, ymax = hi)) +
  geom_ribbon(aes(fill = '95% Confidence Interval (Simulation)'), alpha = .5) +
  geom_line(aes(color = 'Predicted Probability Over Values of Municipal Revenue'),
            size = 1) +
  labs(x = 'Revenue', y = 'Probability of Paramilitary Presence') +
  theme_bw() +
  theme(legend.title = element_text(), legend.position = 'bottom',
        panel.grid.minor.x = element_blank(),
        panel.grid.minor.y = element_blank()) +
  scale_fill_manual(values = 'steelblue3', name = '') +
  scale_color_manual(values = 'black', name = '')
```



Using Observed-Case Approach - Average Predicted Effects

The code above created a hypothetical set of cases in which we might be interested, but what if we wanted more of an average effect? To this end, we can code and plot an 'average predictive effect' that covers the values of our key variable (we still generate predicted probabilities 'along' these values) and all possible values of the other variables in our dataset.

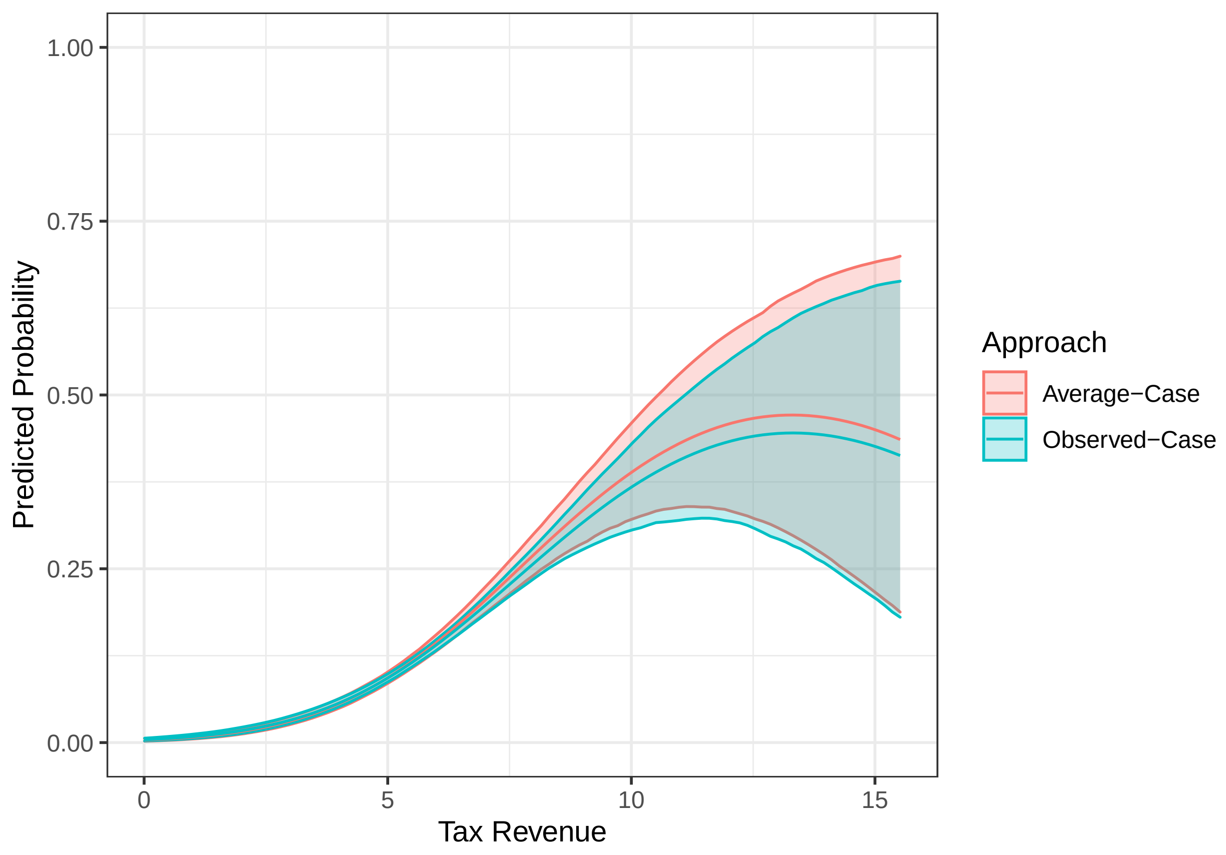
```r
## create a 'temporary' dataset - bind a 1 for the intercept and omit missingness
temporary <- model.matrix( ~ tribut_2_log + tribut_2_log_sq + pop_attacks +
                             disbogota + drugs + pop_log, data = colombia)

## create an object and store predicted probabilities
## look into the ?map_dfc function from the purrr package (part of the tidyverse)

#note this will take a few minutes
pp <- purrr::map_dfc(tax_vec, function(j) {
    temporary[, "tribut_2_log"] <- j
    temporary[, "tribut_2_log_sq"] <- j^2

    pp <- invlogit(temporary %*% t(beta_sims))

    #getting mean for each set of betas
    pp <- apply(pp, 2, mean)

    return(pp)
    })
```

```
## New names:
## * `` -> `...1`
## * `` -> `...2`
## * `` -> `...3`
## * `` -> `...4`
## * `` -> `...5`
## * `` -> `...6`
## * `` -> `...7`
## * `` -> `...8`
## * `` -> `...9`
## * `` -> `...10`
## * `` -> `...11`
## * `` -> `...12`
## * `` -> `...13`
## * `` -> `...14`
## * `` -> `...15`
## * `` -> `...16`
## * `` -> `...17`
## * `` -> `...18`
## * `` -> `...19`
## * `` -> `...20`
## * `` -> `...21`
## * `` -> `...22`
## * `` -> `...23`
## * `` -> `...24`
## * `` -> `...25`
## * `` -> `...26`
## * `` -> `...27`
## * `` -> `...28`
## * `` -> `...29`
## * `` -> `...30`
## * `` -> `...31`
## * `` -> `...32`
## * `` -> `...33`
## * `` -> `...34`
## * `` -> `...35`
## * `` -> `...36`
## * `` -> `...37`
## * `` -> `...38`
## * `` -> `...39`
## * `` -> `...40`
## * `` -> `...41`
## * `` -> `...42`
## * `` -> `...43`
## * `` -> `...44`
## * `` -> `...45`
## * `` -> `...46`
## * `` -> `...47`
## * `` -> `...48`
## * `` -> `...49`
## * `` -> `...50`
## * `` -> `...51`
## * `` -> `...52`
## * `` -> `...53`
## * `` -> `...54`
## * `` -> `...55`
## * `` -> `...56`
## * `` -> `...57`
## * `` -> `...58`
## * `` -> `...59`
## * `` -> `...60`
## * `` -> `...61`
## * `` -> `...62`
## * `` -> `...63`
## * `` -> `...64`
## * `` -> `...65`
## * `` -> `...66`
## * `` -> `...67`
## * `` -> `...68`
## * `` -> `...69`
## * `` -> `...70`
## * `` -> `...71`
## * `` -> `...72`
## * `` -> `...73`
## * `` -> `...74`
## * `` -> `...75`
## * `` -> `...76`
## * `` -> `...77`
## * `` -> `...78`
## * `` -> `...79`
## * `` -> `...80`
## * `` -> `...81`
## * `` -> `...82`
## * `` -> `...83`
## * `` -> `...84`
## * `` -> `...85`
## * `` -> `...86`
## * `` -> `...87`
## * `` -> `...88`
## * `` -> `...89`
## * `` -> `...90`
## * `` -> `...91`
## * `` -> `...92`
## * `` -> `...93`
## * `` -> `...94`
## * `` -> `...95`
## * `` -> `...96`
## * `` -> `...97`
## * `` -> `...98`
## * `` -> `...99`
## * `` -> `...100`
```

```
## find the mean (average predictive effect) for each observation, and bind with
## confidence measures (interval)
plotdat <- t(apply(pp, 2, function(x) { # transposing w/ t() puts data into a column
    c(M = mean(x), quantile(x, c(0.025, 0.975)))
}))

# add x vector and convert to data frame
plotdat <- data.frame(tax_vec, plotdat)

# better names and show the first few rows (to check)
colnames(plotdat) <- c('tax', 'pe', 'lo', 'hi')
head(plotdat)
```

```
##               tax          pe          lo          hi
## ...1 0.009950331 0.004089794 0.002376621 0.006330280
## ...2 0.166588192 0.004614804 0.002753678 0.007041995
## ...3 0.323226054 0.005200760 0.003197065 0.007828497
## ...4 0.479863916 0.005853703 0.003703885 0.008693533
## ...5 0.636501778 0.006580128 0.004253209 0.009608072
## ...6 0.793139640 0.007386993 0.004878889 0.010602127
```

```
#now let's rowbind in the average-case predicted probabilities, so that we can
#compare them
plotdat <- rbind(plotdat, sim_output) %>%
  mutate(Approach = c(rep("Observed-Case",length(tax_vec)),
                      rep("Average-Case", length(tax_vec))))

## plot of predicted probabilities
ggplot(plotdat, aes(x = tax, y = pe, ymin = lo, ymax = hi, color = Approach,
                    fill = Approach)) +
  geom_ribbon(alpha = .25) +
  geom_line() +
  labs(x = 'Tax Revenue', y = 'Predicted Probability') +
  ylim(c(0, 1)) +
  theme_bw()
```



Assignment

Read in the data 'turnout.csv.' Fit a logit model that predicts voting based on race, education and income.

Now, fit a second model that predicts voting based on race, education, income, and age. Assume that you've hypothesized that age has a curvilinear relationship with voting turnout, such that lower ages and higher ages are less likely to vote. Include this transformed variable.

Compare the model fit of these two options using lrtest, AIC, and BIC. Which do you choose as the better fitting model?

Simulate and plot predicted probabilities over the range of age, holding all other variables at their mean, for the second model. Plot this.

Generate an average predicted effect over the observed values of income for the first model. Plot this.

Required packages and data:

```r
#Load in packages
pacman::p_load(lmtest)

turnout<-import("turnout.csv")
```

First, we fit the models:

```r
m1 <- glm(vote ~ race + educate + income, turnout, family = binomial(link = 'logit'))
m2 <- glm(vote ~ race + educate + income + poly(age, 2, raw = T), turnout,
          family = binomial(link = 'logit'))
```

```
lrtest(m1, m2)
```

```
## Likelihood ratio test
##
## Model 1: vote ~ race + educate + income
## Model 2: vote ~ race + educate + income + poly(age, 2, raw = T)
##   #Df  LogLik Df  Chisq Pr(>Chisq)
## 1   4 -1048.3
## 2   6 -1004.8  2 87.023  < 2.2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```
AIC(m1)
```

```
## [1] 2104.598
```

```
AIC(m2)
```

```
## [1] 2021.575
```

```
BIC(m1)
```

```
## [1] 2127.001
```

```
BIC(m2)
```

```
## [1] 2055.18
```

Average-Case Approach

When we combine the average-case approach with simulation, we create a hypothetical design matrix, where we hold all continuous variables at their mean or another justifiable value, and then let the predictor of interest vary along some range.

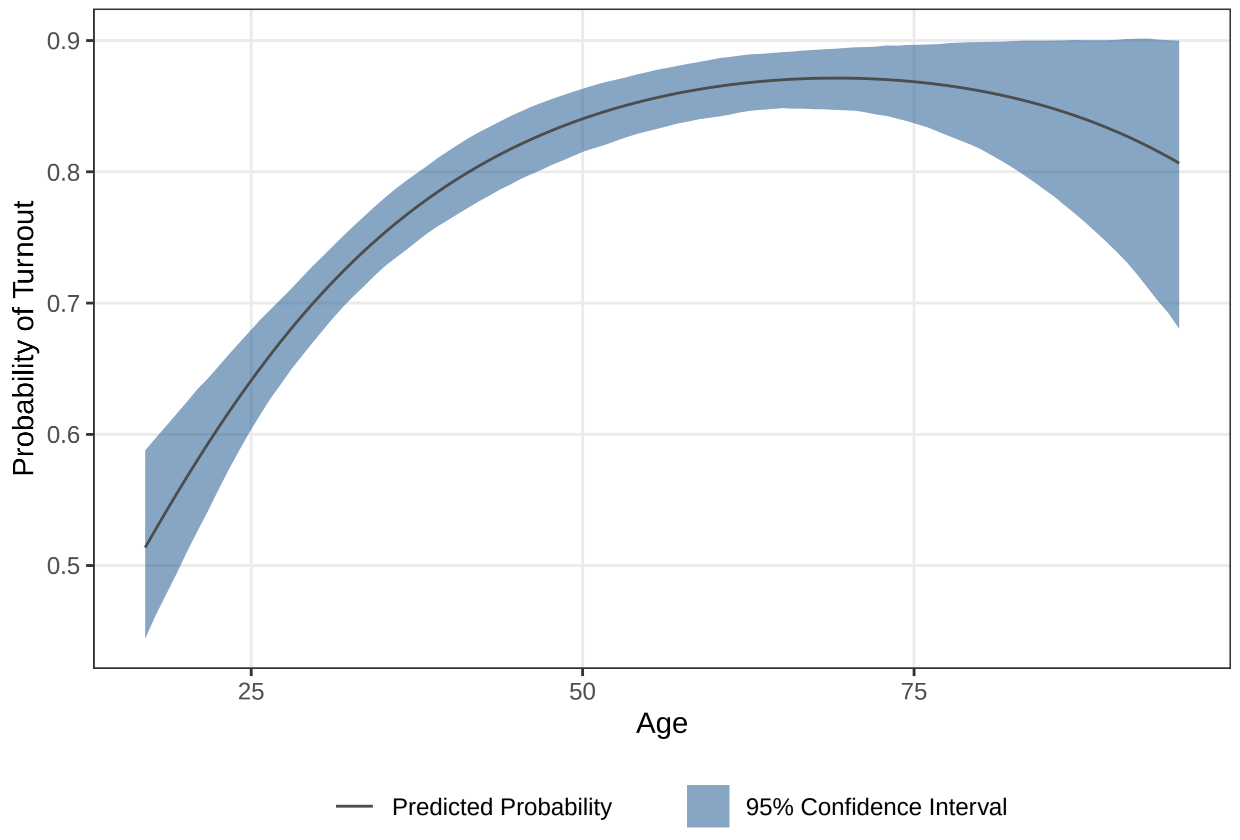
```
sim_dat <- with(turnout, data.frame(intercept = 1, race = 1,
                                    educate = mean(educate), income = mean(income),
                                    age = seq(min(age), max(age), length.out = 100),
                                    age2 = (seq(min(age), max(age), length.out = 100))^2))
```

In order to simulate different betas, we take a predetermined number of draws (1000 in this case here) from a multi-variate normal distribution centered around the coefficient estimates from our model, with a variance-covariance matrix also from our model. We then matrix multiply our hypothetical design matrix with our simulated betas. This means that we will have 1000 different predicted probabilities for each row in our hypothetical design matrix (in other words, we have 1000 different predicted probabilities for each element of the vector of values for our predictor of interest). We then have to make sure that we take the mean, and the quantiles we are interested in, along whichever dimension represents the elements of the hypothetical range vector. We can do this using `apply`. Note that in the example below, we are using `apply` along the rows of `sim_pp` because of how we constructed `sim_pp` (also using `apply`.) If we had not used `apply` and had instead simply done `as.matrix(sim_dat) %*% sim_betas`, the dimensions of `sim_pp` would have been reversed, and we would have to `apply` the mean and quantile functions along the columns, not the rows.

```
sim_betas <- mvrnorm(1e3, coef(m2), vcov(m2))

sim_pp <- apply(sim_betas, 1, function(x) invlogit(as.matrix(sim_dat) %*% x))

sim_gg <- data.frame(age = sim_dat$age, pe = apply(sim_pp, 1, mean),
                     lo = apply(sim_pp, 1, quantile, probs = .025),
                    hi = apply(sim_pp, 1, quantile, probs = .975))
```

We can plot the predicted probabilities and our confidence bands.

```
ggplot(sim_gg, aes(x = age, y = pe, ymin = lo, ymax = hi)) +
  geom_ribbon(aes(fill = '95% Confidence Interval'), alpha = .5) +
  geom_line(aes(color = 'Predicted Probability')) +
  labs(x = 'Age', y = 'Probability of Turnout') +
  theme_bw() +
  theme(legend.title = element_text(), legend.position = 'bottom',
        panel.grid.minor.x = element_blank(),
        panel.grid.minor.y = element_blank()) +
  scale_fill_manual('', values = 'dodgerblue4') +
  scale_color_manual('', values = 'grey30')
```



Observed-Case Approach

When taking the observed-case approach (see Hanmer and Kalkan (2013) for why this is the preferable approach) and using simulation to get confidence intervals, we have to do things slightly differently. Specifically, we will have to take an addition mean. This is because whereas in the average-case approach, we had one row for each hypothetical value of our predictor of interest (however long we decided to make it), in the observed-case approach, we will have a whole matrix or dataframe for each hypothetical value of our predictor of interest.

We start by creating a temporary design matrix. This while start out as being the same exact design matrix used to fit the model, which is why we can just use `model.matrix()` on our model object.

```
temporary <- model.matrix(m1)
```

We then have to make sure to once again simulate our betas, using our model object.

```
#getting sim betas for m1
sim_betas_m1 <- mvrnorm(1e3, coef(m1), vcov(m1))
```

Just like with the average-case approach, we have to create the values we want our predictor of interest (*income* in this case) to take on. Just like in the average-case approach, these are really hypothetical values. Here, we will let income vary from its min to its max.

```
#getting range of income
income_vec <- seq(from = min(turnout$income), to = max(turnout$income),
                  length.out = 100)
```

In the next step, we do several important things at once. We will do these things using the `map_dfc` function from `purrr` (part of the tidyverse). This works a lot like the `apply` family of functions, but is somewhat faster. It maps a function to a vector and then, because of the `_dfc` suffix, column binds the results of the function together. In other words, we apply the function to each element in the vector, and then (because the function returns a vector) we get a dataframe in as output. Note that we could do something similar with `lapply`, but then we would get a list of vectors.

```
pp <- map_dfc(income_vec, function(j) {
    temporary[, "income"] <- j

    pp <- invlogit(temporary %*% t(sim_betas_m1))

    pp <- apply(pp, 2, mean)

    return(pp)
    })
```

```
## New names:
## * `` -> `...1`
## * `` -> `...2`
## * `` -> `...3`
## * `` -> `...4`
## * `` -> `...5`
## * `` -> `...6`
## * `` -> `...7`
## * `` -> `...8`
## * `` -> `...9`
## * `` -> `...10`
## * `` -> `...11`
## * `` -> `...12`
## * `` -> `...13`
## * `` -> `...14`
## * `` -> `...15`
## * `` -> `...16`
## * `` -> `...17`
## * `` -> `...18`
## * `` -> `...19`
## * `` -> `...20`
## * `` -> `...21`
## * `` -> `...22`
## * `` -> `...23`
## * `` -> `...24`
## * `` -> `...25`
## * `` -> `...26`
## * `` -> `...27`
## * `` -> `...28`
## * `` -> `...29`
## * `` -> `...30`
## * `` -> `...31`
## * `` -> `...32`
## * `` -> `...33`
## * `` -> `...34`
## * `` -> `...35`
## * `` -> `...36`
## * `` -> `...37`
## * `` -> `...38`
## * `` -> `...39`
## * `` -> `...40`
## * `` -> `...41`
## * `` -> `...42`
## * `` -> `...43`
## * `` -> `...44`
## * `` -> `...45`
## * `` -> `...46`
## * `` -> `...47`
## * `` -> `...48`
## * `` -> `...49`
## * `` -> `...50`
## * `` -> `...51`
## * `` -> `...52`
## * `` -> `...53`
## * `` -> `...54`
## * `` -> `...55`
## * `` -> `...56`
## * `` -> `...57`
## * `` -> `...58`
## * `` -> `...59`
## * `` -> `...60`
## * `` -> `...61`
## * `` -> `...62`
## * `` -> `...63`
## * `` -> `...64`
## * `` -> `...65`
## * `` -> `...66`
## * `` -> `...67`
## * `` -> `...68`
## * `` -> `...69`
## * `` -> `...70`
## * `` -> `...71`
## * `` -> `...72`
## * `` -> `...73`
## * `` -> `...74`
## * `` -> `...75`
## * `` -> `...76`
## * `` -> `...77`
## * `` -> `...78`
## * `` -> `...79`
## * `` -> `...80`
## * `` -> `...81`
## * `` -> `...82`
## * `` -> `...83`
## * `` -> `...84`
## * `` -> `...85`
## * `` -> `...86`
## * `` -> `...87`
## * `` -> `...88`
## * `` -> `...89`
## * `` -> `...90`
## * `` -> `...91`
## * `` -> `...92`
## * `` -> `...93`
## * `` -> `...94`
## * `` -> `...95`
## * `` -> `...96`
## * `` -> `...97`
## * `` -> `...98`
## * `` -> `...99`
## * `` -> `...100`
```

The function we apply to each element of our hypothetical predictor value vector does a few things: 1. It takes the temporary design matrix we created above, and then replaces *all* of the values of our predictor of interest with one value (taken from the hypothetical value vector). This looks like the following:

```
head(income_vec)
```

```
## [1] 0.0000000 0.1507545 0.3015091 0.4522636 0.6030182 0.7537727
```

```
temporary[, "income"] <- income_vec[1]

head(temporary)
```

```
##   (Intercept) racewhite educate income
## 1           1         1      14      0
## 2           1         1      10      0
## 3           1         1      12      0
## 4           1         1       8      0
## 5           1         1      12      0
## 6           1         1      12      0
```

2. We then matrix multiply this new design matrix, with all actual values of our predictor of interest replaced by one hypothetical value of our predictor interest and with all the other values held at their observed values, by our simulated betas. This creates a matrix, which has as many rows as the rows of our data, and as many columns as there are samples of our betas.

```
dim(invlogit(temporary %*% t(sim_betas_m1)))
```

```
## [1] 2000 1000
```

3. We then take the mean across the columns (`margins = 2`) because different columns represent different predicted probabilities, for each of the observations in our temporary design matrix, when we use different betas and when our predictor of interest has been replaced with one value. This means that we then have a mean predicted probability for each different set of betas when our predictor of interest is at a certain value.

The call to map_dfc then returns a dataframe that has 1000 rows and 100 columns (1 column for each value of our hypothetical value vector). Because we are interested in how the predicted probability changes according to the hypothetical values of our predictor of interest and because we currently have 1000 predicted probabilities for each of these hypothetical values (1 for each of our simulated betas), we once again use apply with margins = 2. We get the mean of predicted values for each value of our hypothetical value vector, and then use the quantile with a probs argument equal to the bounds of our desired confidence interval (for example, if we want a 95% confidence interval, we will do quantile(x, probs = c(0.025, 0.975))).

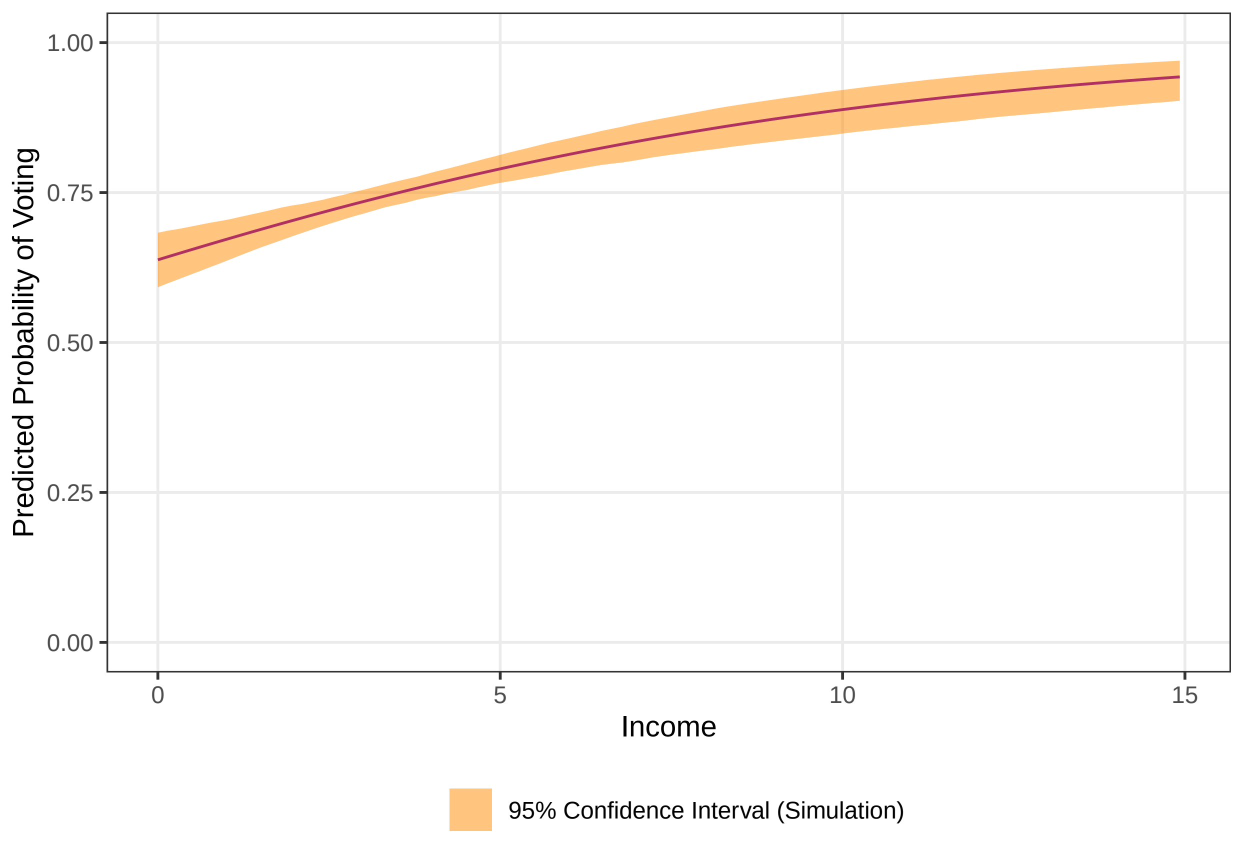
```
plotdat <- t(apply(pp, 2, function(x) { # transposing w/ t() puts data into a column
    c(M = mean(x), quantile(x, c(0.025, 0.975)))
}))

plotdat <- data.frame(plotdat, income_vec)
```

We can then use these data to plot the mean predicted probability and our confidence bands.

```
colnames(plotdat) <- c('pe', 'lo', 'hi', 'income')
head(plotdat)
```

```
##              pe        lo        hi    income
## ...1 0.6378566 0.5921072 0.6830007 0.0000000
## ...2 0.6430982 0.5989037 0.6865457 0.1507545
## ...3 0.6483069 0.6054548 0.6893075 0.3015091
## ...4 0.6534816 0.6120481 0.6926099 0.4522636
## ...5 0.6586213 0.6184941 0.6960571 0.6030182
## ...6 0.6637249 0.6252134 0.6994702 0.7537727
```

```
ggplot(plotdat, aes(x = income, y = pe, ymin = lo, ymax = hi)) +
  geom_ribbon(aes(fill = '95% Confidence Interval (Simulation)'),
              alpha = .5) +
  geom_line(col = 'maroon') +
  labs(x = 'Income', y = 'Predicted Probability of Voting') +
  ylim(c(0, 1)) +
  scale_fill_manual('', values = 'darkorange') +
  theme_bw() +
  theme(legend.title = element_text(), legend.position = 'bottom',
        panel.grid.minor.x = element_blank(),
        panel.grid.minor.y = element_blank())
```


Predicted Probability of Voting
1.00
0.75
0.50
0.25
0.00
0
5
10
15
Income
95% Confidence Interval (Simulation)