Limited Dependent Variable Models

Dr. Sarah Hunter

3/26/2020

Not all outcomes that we want to study are continuous. Some dependent variables are are "dummy variables", also called "indicator variables". These are the variables that have two possible values, usually 0 or 1 where 1 indicates the presence of the variable and 0 indicating otherwise. Examples of dummy variable outcomes from political science include:

- Support for a policy (1= yes, 0=no)
- Vote choice (1=Vote for X, 0=did not vote for X)
- War (1=war happened, 0= war did not happen)
- Sanctions (1= sanctions imposed, 0=no sanctions)
- Election wins (1=win, 0=not)

Dependent variables with only 2 possible values make modelling decisions somewhat more complicated. Linear Regression is only for continuous dependent variables. This lesson today is about modeling those dependent variables that are limited to two values. We will cover 3 strategies: The Linear Probability Model, Logit Models, and Probit Models.

Linear Probability Model

The Linear Probability Model (LPM) is essentially an OLS model where the dependent variable is a dummy variable. It works because the predicted values ($\hat{Y}$) are `predicted probabilities`. Predicted probabilities are the estimated probability that $Y = 1$, in other words, the probability that the observed value of the dummy variable was 1 (indicating that the event happened). To do this, we take the usual linear regression model:

$$Y_i = \hat{\alpha} + \hat{\beta} * X_i + \hat{u}_i$$

and simple change the Y_i to the Probability that $Y_1 = 1$ written as $\hat{P}(Y_i = 1)$:

$$\hat{P}(Y_i = 1) = \hat{\alpha} + \hat{\beta} * X_i + \hat{u}_i$$

Estimating this model in R is very simple: you estimate it with the same command that you would for a normal linear regression model: `lm`. Example code of a Linear Probability Model is below.

For this model, I use the `states` dataset from the "poliscidata" library in R. This is a dataset focused on President Obama's 2012 and 2008 election results in each state as well as various stat characteristics and demographics.

```
library(poliscidata)
```

```
## Registered S3 method overwritten by 'gdata':
##   method         from
##   reorder.factor gplots
```

```
#Loading data from an R package
statedata<-states
```

This next code shows how to estimate the LPM. The dependent variable here is whether or not President Obama won that state in 2012. Currently, the way this variable is coded is "yes" for winning the state and "no" for not winning it. We can tell this by using the `summary` command:

```
summary(statedata$obama_win12)
```

```
##  No Yes
##  24  26
```

We want this variable to be a dummy variable where 1 indicates the state voted for President Obama, and 0 indicates the state did not. We can convert this variable to a dummy variable by creating a new variable using the `ifelse` command, which can create a variable with conditions. The syntax is: `ifelse(T/F condition, what to put if true, what to put if false)`. Below is the code used to create this dummy variable:

```
statedata$new2012win<-ifelse(statedata$obama_win12=="Yes", 1, 0)

table(statedata$new2012win)
```

```
##
##  0  1
## 24 26
```

Now we can estimate our LPM. The independent variables in this model are percentage of the population with a bachelor's degree or more ("ba_or_more") and the percent of the population living in urban areas in that state ("urban").

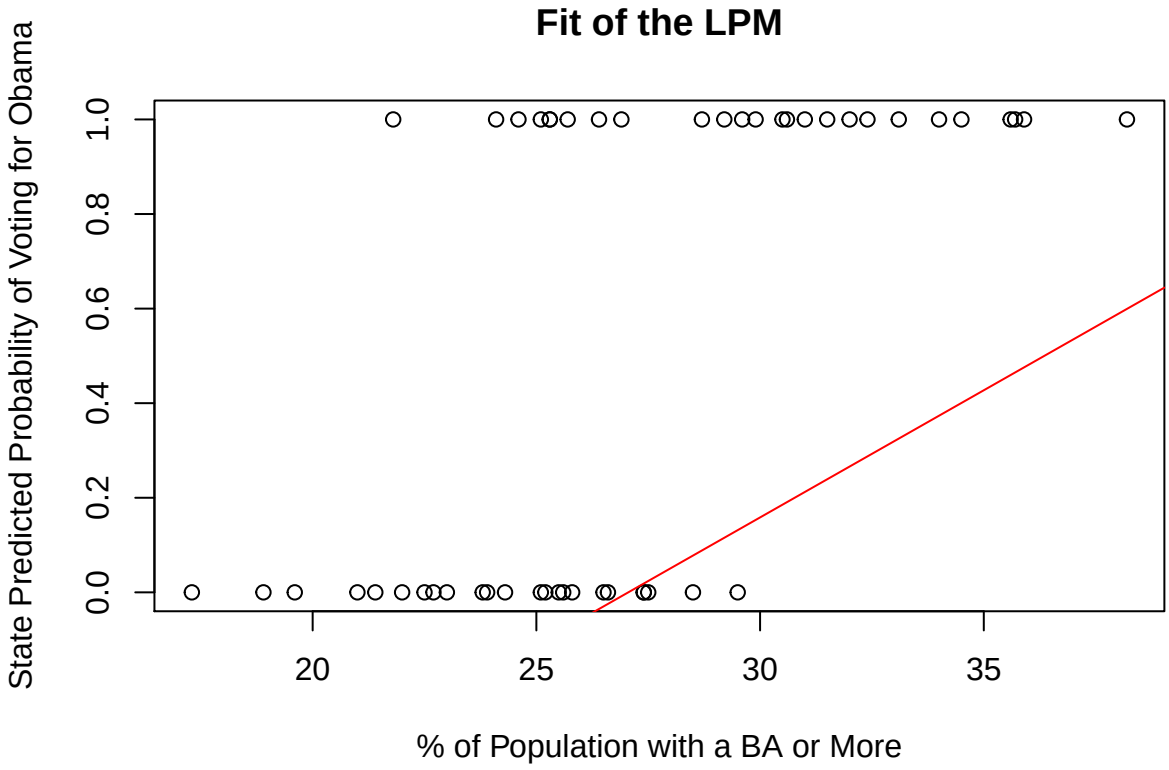
```
LPM<-lm(new2012win ~ ba_or_more + urban, data=statedata)

summary(LPM)
```

```
##
## Call:
## lm(formula = new2012win ~ ba_or_more + urban, data = statedata)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -0.70979 -0.28984 -0.07551  0.29516  0.72049
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept) -1.455472   0.356030  -4.088 0.000169 ***
## ba_or_more   0.053780   0.013775   3.904 0.000301 ***
## urban        0.007171   0.004374   1.640 0.107774
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.3979 on 47 degrees of freedom
## Multiple R-squared:  0.4036, Adjusted R-squared:  0.3782
## F-statistic:  15.9 on 2 and 47 DF,  p-value: 5.31e-06
```

The results for the LPM look identical to the OLS model. The interpretation, however, does change slightly. With an LPM, the regression coefficients are interpreted as: "A one unit increase in x leads to a β change in the **predicted probability** of y". Therefore, we can say of the above model that "for every one unit increase in the respondent's education there is a .025373 decrease in the predicted probability of voting for Bush".

While the LPM is an easy model to estimate, it does present some issues. Mainly, the LPM violates OLS assumptions of correct functional form. Because OLS was designed for continuous dependent variables, the LPM can estimate predicted probabilities that are not possible (negative probabilities or those greater than

one). Additionally, the LPM does not fit the data very well. The figure below shows the predicted probability of voting for Obama in 2012 as a function of the percent of the state population with a BA or more with the regression line in red:



The next two models (logits and probits) we will discuss solve this problem by forcing the predicted probabilities to remain between 0 and 1. These models do this with a **link function**. Link functions link the linear component of a logit or probit model, to the quantity in which we are interested, the predicted probability that the dummy dependent variable equals one. Each model is defined by which link function it uses.

Logit Model

A Logit model uses a link function that is represented by Λ, or the upper case Greek letter lambda. We write the regression equation as:

$$\hat{P}(Y_i = 1) = \Lambda(\hat{\alpha} + \hat{\beta} * X_i + \hat{u}_i)$$

Which can be shortened to:

$$\hat{P}(Y_i = 1) = \Lambda(\hat{\beta}_i X_i)$$

where $\hat{\beta}_i X_i$ is a matrix of all independent variables and their coefficients. This is a shorthand notation to stand in for all the terms in the model, rather than writing out all of them individually.

We can substitute Λ for the actual mathematical function, called the logistic function, and write the above expression as:

$$\hat{P}(Y_i = 1) = \frac{exp(\hat{\beta}_i X_i)}{exp(\hat{\beta}_i X_i)+1}$$

where $exp(\hat{\beta}_i X_i) = e^{(\hat{\beta}_i X_i)}$.

Recall that e is an irrational, mathematical constant approximately equal to 2.718281828459. It is a number like π that continues infinitely, never repeating. e is the base number of the natural log (ln) and is used in

logistic regression.

While this is a difficult computation, R implements this function easily with the command `glm`. The `glm` command functions much like the `lm` command in that it uses the same formula structure. However, with `glm`, you must also tell R which link function you want to use with the `family=binomial(link=)` option. The following shows to estimate a logit model with the same variables used in the LPM:

```
logit<-glm(new2012win ~ ba_or_more + urban,
           family=binomial(link="logit"), data=statedata)

summary(logit)
```

```
##
## Call:
## glm(formula = new2012win ~ ba_or_more + urban, family = binomial(link = "logit"),
##     data = statedata)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -1.8847  -0.6640   0.1235   0.5968   1.7989
##
## Coefficients:
##              Estimate Std. Error z value Pr(>|z|)
## (Intercept) -13.94910    3.99720  -3.490 0.000484 ***
## ba_or_more    0.39319    0.12792   3.074 0.002114 **
## urban         0.04914    0.02864   1.716 0.086183 .
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 69.235  on 49  degrees of freedom
## Residual deviance: 43.055  on 47  degrees of freedom
## AIC: 49.055
##
## Number of Fisher Scoring iterations: 5
```

The interpretation of a Logit model is quite different than for the linear model, due to the use of link functions. This is not a linear model, so a one unit increase in x leads to a change in y that depends on the value of x. Therefore, the correct interpretation of the coefficients for this model (in a generic form) would be: a one unit increase in x leads to a $\hat{\beta}$ change in **the log odds of y**. The best way to interpret and present your results in a logit model is to plot the predicted probabilities, discussed later.

Probit Model

The probit model, on the other hand, uses the cumulative distribution function (CDF) of the standard normal distribution as a link function. We represent the probit link function with Φ, the uppercase Greek letter phi. We write the regression equation as:

$$\hat{P}(Y_i = 1) = \Phi(\hat{\alpha} + \hat{\beta} * X_i + \hat{u}_i)$$

Which can be shortened to:

$$\hat{P}(Y_i = 1) = \Phi(\hat{\beta}_i X_i)$$

With the Probit model,

$\Phi(x) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{x} e^{-\frac{y^2}{2}} dy.$

As you can see, this is a very nasty function. However, R does take care of this for us in a simple way, using the same `glm` function as with the logit. The only difference is the link function specified:

```
probit<-glm(new2012win ~ ba_or_more + urban,
            family=binomial(link="probit"), data=statedata)

summary(probit)
```

```
##
## Call:
## glm(formula = new2012win ~ ba_or_more + urban, family = binomial(link = "probit"),
##     data = statedata)
##
## Deviance Residuals:
##      Min        1Q    Median        3Q       Max
## -1.87747  -0.66240   0.05882   0.58245   1.77642
##
## Coefficients:
##             Estimate Std. Error z value Pr(>|z|)
## (Intercept) -8.45766    2.20900  -3.829 0.000129 ***
## ba_or_more   0.24097    0.07226   3.335 0.000853 ***
## urban        0.02877    0.01660   1.733 0.083034 .
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 69.235  on 49  degrees of freedom
## Residual deviance: 42.565  on 47  degrees of freedom
## AIC: 48.565
##
## Number of Fisher Scoring iterations: 7
```

Notice that these coefficients are very different than the logit model, despite using the exact same data. This is due to the vastly different scales produced by the link functions. While the logit coefficients are on the log odds scale, the probit coefficients are on the z-scale (the standard normal distribution, similar to Student's t distribution). Follow this link for a more in depth explanation: link.

The proper interpretation of a probit coefficient is (in generic terms): "a unit unit increase in x leads to a $\hat{\beta}$ change in the associated Z (or t) score". Again, this is not an intuitive interpretation of a model. The best way to show the effect of an x variable on the probability of y is to plot the predicted probabilities, just like in the case of logit models.

Predicted Probability Plots

I have mentioned a few times that the best way to display logit and probit results is to plot the predicted probabilities. We can make predicted probability plots easily with the effects package from the linear regression lesson. The code is simple and the one modification that is very important is to add the `rescale.axis=F`, otherwise R will assume you want a line. The following code does the predicted probability plots from both the logit and probit models:

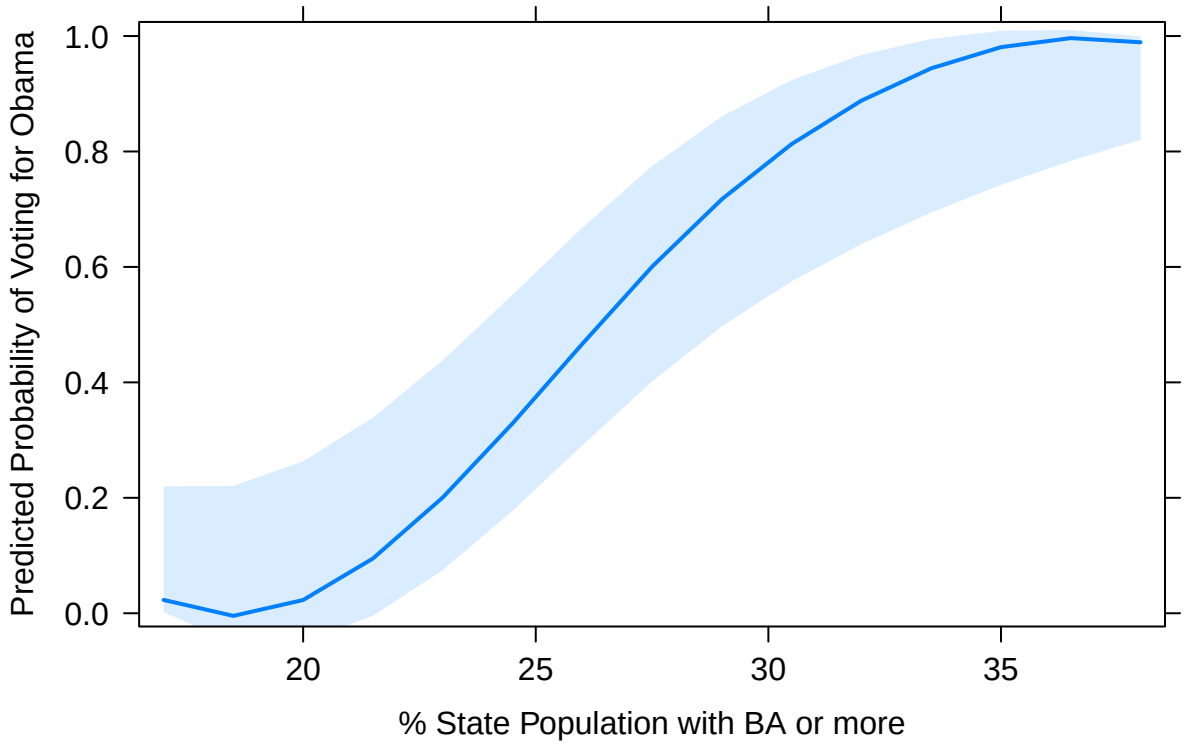
```
library(effects)
```

```
## Loading required package: carData
```

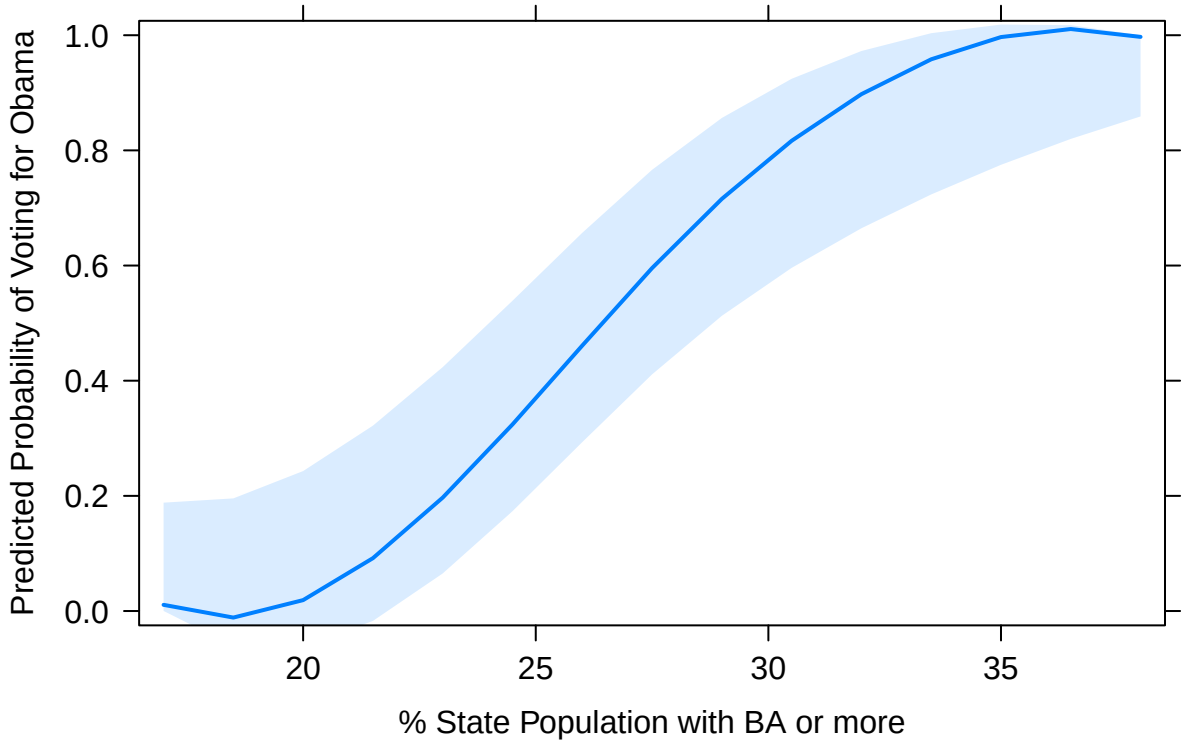
```
## Registered S3 methods overwritten by 'lme4':
##   method                          from
##   cooks.distance.influence.merMod car
##   influence.merMod                car
##   dfbeta.influence.merMod         car
##   dfbetas.influence.merMod        car
```

```
## lattice theme set by effectsTheme()
## See ?effectsTheme for details.
```

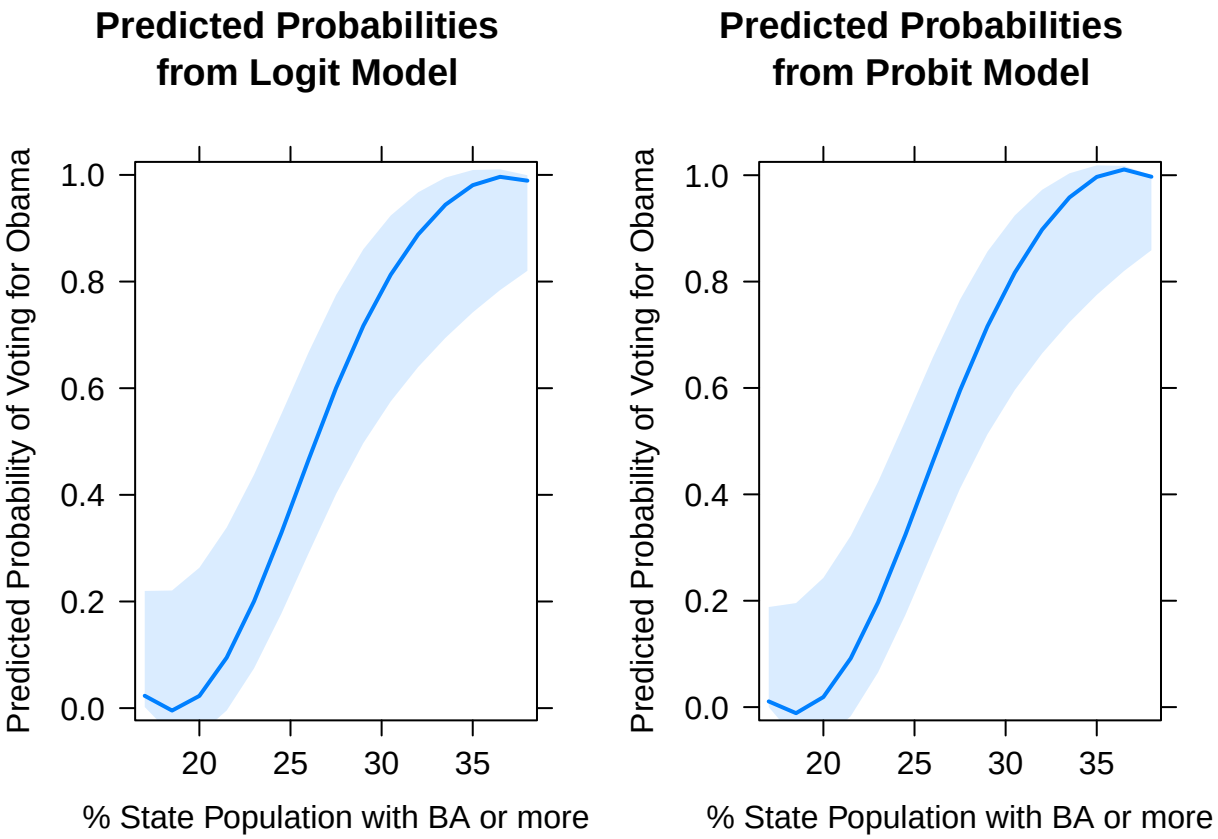
```
plot(Effect("ba_or_more", mod=logit), rescale.axis=F,
     ylab="Predicted Probability of Voting for Obama",
     xlab="% State Population with BA or more",
     main="Predicted Probabilities from Logit Model", rug=F)
```




```
plot(Effect("ba_or_more", mod=probit), rescale.axis=F,
     ylab="Predicted Probability of Voting for Obama",
     xlab="% State Population with BA or more",
     main="Predicted Probabilities from Probit Model", rug=F)
```




```
a<-plot(Effect("ba_or_more", mod=logit), rescale.axis=F,
        ylab="Predicted Probability of Voting for Obama",
        xlab="% State Population with BA or more",
        main="Predicted Probabilities \n from Logit Model", rug=F)

b<-plot(Effect("ba_or_more", mod=probit), rescale.axis=F,
        ylab="Predicted Probability of Voting for Obama",
        xlab="% State Population with BA or more",
        main="Predicted Probabilities \n from Probit Model", rug=F)

#install.packages("gridExtra")

library(gridExtra)

grid.arrange(a, b, nrow=1)
```



Looking at them side-by-side, they predicted probabilities look very similar considering the very different coefficients. This is the point: even from different scales, logit and probit models should give nearly identical predicted probability results. On that point, it really does not matter which type you use. Industry standard in political science is the logit model. However, academic fields such as economics tend to use the probit model more.

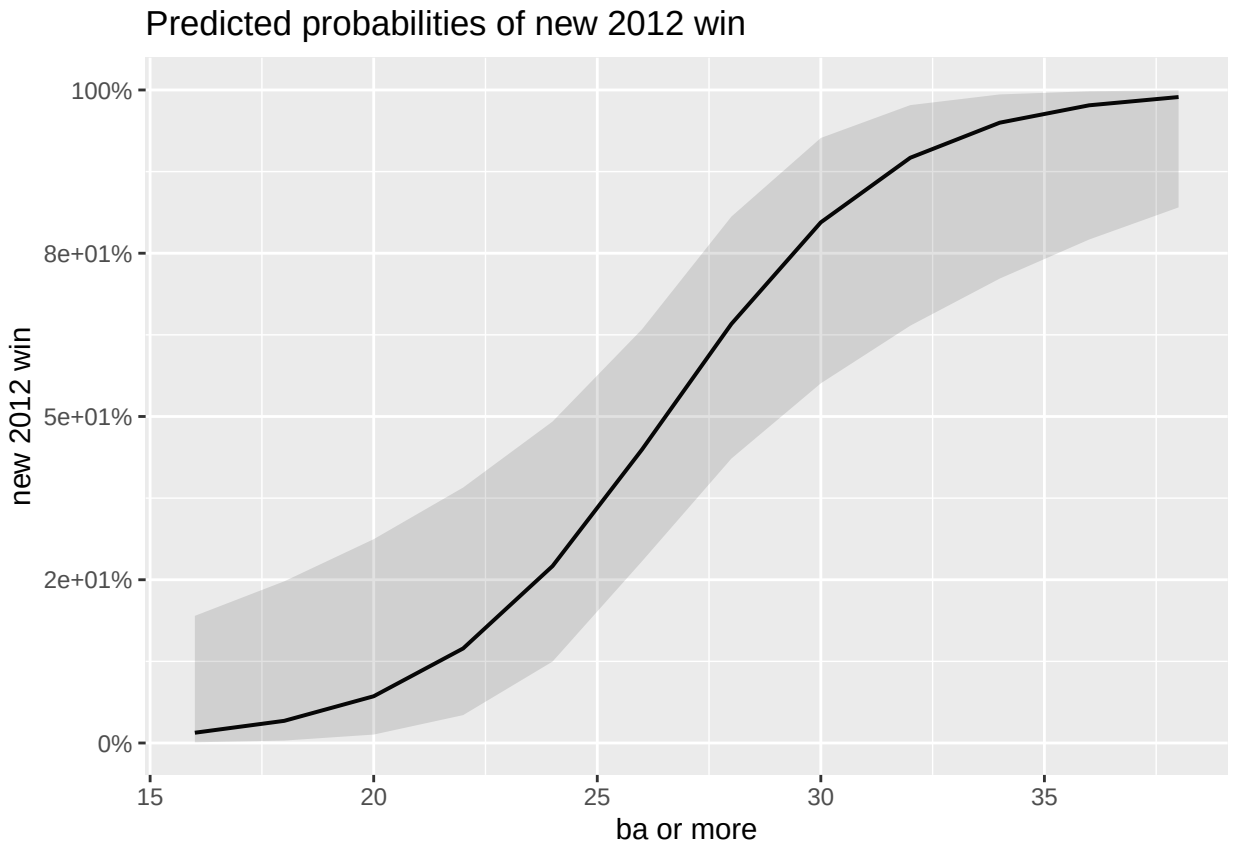
You can also use another package called "sjPlot" to plot predicted probabilities. An example is shown below:

```
#install.packages("sjPlot")
#install.packages("sjmisc")
#install.packages("ggplot2")

library(sjPlot)
```

```
## Learn more about sjPlot with 'browseVignettes("sjPlot")'.
```

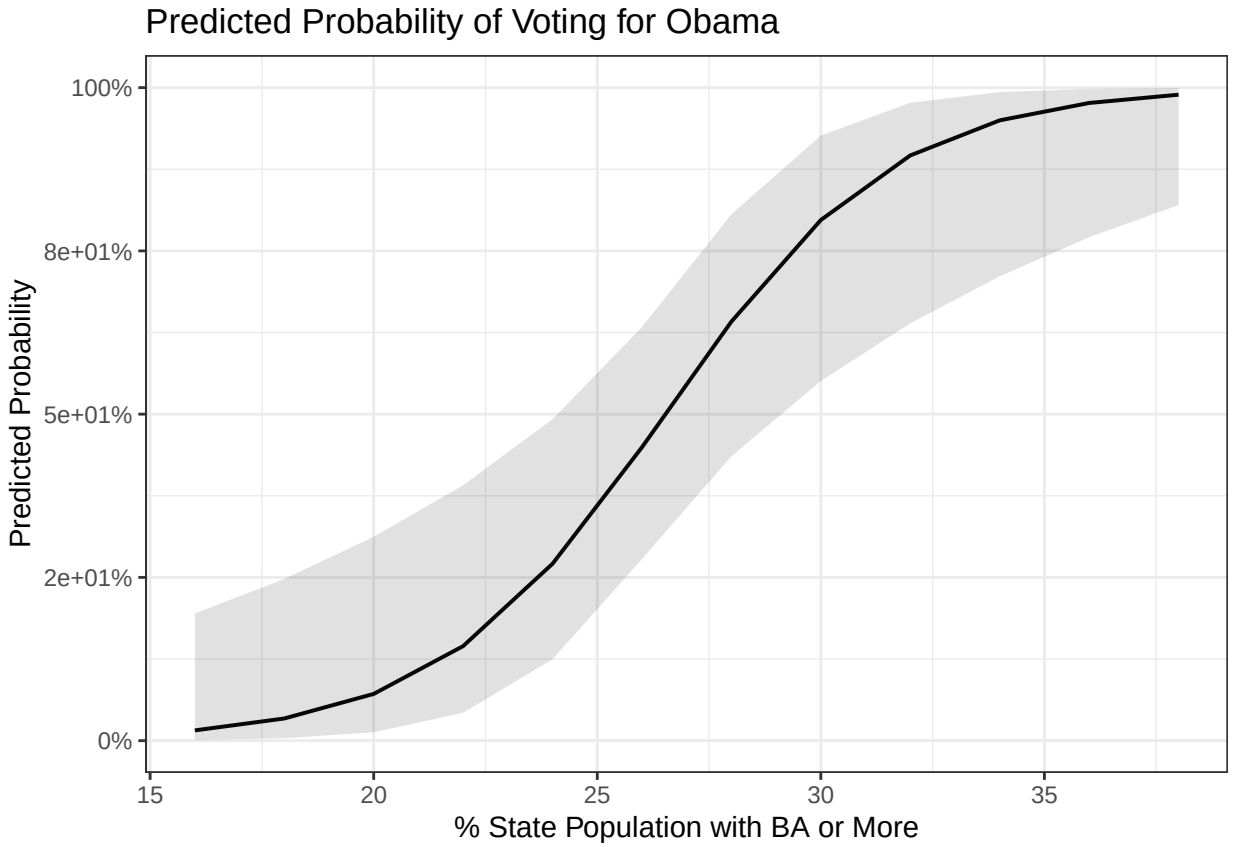
```
library(sjmisc)
library(ggplot2)

plot_model(logit, type="pred", terms="ba_or_more")
```

```
## Data were 'prettified'. Consider using `terms="ba_or_more [all]"` to get smooth plots.
```




```
plot_model(logit, type="pred", terms="ba_or_more")+theme_bw()+
  xlab("% State Population with BA or More")+
  ylab("Predicted Probability")+
  ggtitle("Predicted Probability of Voting for Obama")
```

```
## Data were 'prettified'. Consider using `terms="ba_or_more [all]"` to get smooth plots.
```



This plot uses the language of a package called "ggplot2" that makes some very nice-looking plots, but uses a very different coding syntax than base R. However, it is worth learning if you want to make some professional plots.

Calculating Goodness of Fit (PRE and Percent Correctly Classified)

One thing that you do not get when estimated a logit or probit model is an R-squared. Because of the non-linear nature of the model and the fact that the dependent variable can only take on two values, R-squared is not a useful measure of model fit. However, we do have a few options, of which, we will discuss two: proportional reduction in error (PRE) and percent correctly classified. The PRE essentially tells us how much more of the dependent variable we correctly predicted than the null model (aka if we just guessed the most frequent category). The percent correctly classified is just as it sounds: how many times does our model correctly predict the value of the dependent variable.

Luckily, there is a package in R that does both of these calculations for us! The following code shows how to calculate the PRE and the percent correctly classified:

```
#install.packages("DAMisc")

library(DAMisc)
```

```
## fANCOVA 0.5-1 loaded
```

```
#PRE

pre(logit)
```

```
## mod1:  new2012win ~ ba_or_more + urban
```

```
## mod2:  new2012win ~ 1
##
## Analytical Results
##  PMC =  0.520
##  PCP =  0.740
##  PRE =  0.458
## ePMC =  0.501
## ePCP =  0.712
## ePRE =  0.424
```

Within the `pre` command, you get both the PRE and the percent correctly classified/predicted. The PRE number is the proportional reduction in error. Our logit model from earlier has a PRE of 0.458. This means that our model correctly guesses 45.8% more than simply guessing 1 each time. Overall, the model correctly guesses 74% of the observations (see the PCP line in the results form our `pre` command).