Now, we'll take a longer look at using logistic regression in R.

We now define the **logistic regression** model.

$$\log\left(\frac{p(\mathbf{x})}{1-p(\mathbf{x})}\right) = \beta_0 + \beta_1 x_1 + \ldots + \beta_{p-1} x_{p-1}$$

$$p(\mathbf{x}) = P[Y = 1 \mid \mathbf{X} = \mathbf{x}] = \frac{e^{\beta_0+\beta_1 x_1+\cdots+\beta_{p-1}x_{(p-1)}}}{1+e^{\beta_0+\beta_1 x_1+\cdots+\beta_{p-1}x_{(p-1)}}}$$

we're dealing with a binary response. The thing that we'll be interested in is the probability that that response takes the value 1 given values of the predictors. So the model has on the right-hand side this linear combination of the predictors and then through the use of the logit link function, we'll relate those predictors to that probability. So, we can undo that link function and directly obtain an expression for that probability as a function of the model parameters and the values of the predictors.

```r
sim_logistic_data = function(sample_size = 25, beta_0 = -2, beta_1 = 3) {
  x = rnorm(n = sample_size)
  eta = beta_0 + beta_1 * x
  p = 1 / (1 + exp(-eta))
  y = rbinom(n = sample_size, size = 1, prob = p)
  data.frame(y, x)
}
```

$$\log\left(\frac{p(\mathbf{x})}{1-p(\mathbf{x})}\right) = -2 + 3x$$

better matches the function we're using t

$$Y_i \mid \mathbf{X_i} = \mathbf{x_i} \sim \text{Bern}(p_i)$$

$$p_i = p(\mathbf{x_i}) = \frac{1}{1+e^{-\eta(\mathbf{x_i})}}$$

$$\eta(\mathbf{x_i}) = -2 + 3x_i$$

y is technically a numeric predictor but it just happens to only take two possible values, zero and one. So, because this y is technically numeric, we can use ordinary linear regression.

```r
set.seed(1)
example_data = sim_logistic_data()
head(example_data)
```

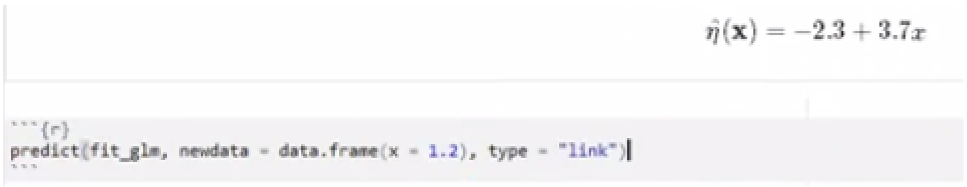
```r
# ordinary linear regression
fit_lm  = lm(y ~ x, data = example_data)
# logistic regression
fit_glm = glm(y ~ x, data = example_data, family = binomial)
```

```r
# more detailed call to glm for logistic regression
fit_glm = glm(y ~ x, data = example_data, family = binomial(link = "logit"))
```

Making predictions with an object of type `glm` is slightly different than making predictions after fitting with `lm()`. In the case of logistic regression, with `family = binomial`, we have:

`type`	Returned
`"link"` [default]	$\hat{\eta}(\mathbf{x}) = \log\left(\frac{\hat{p}(\mathbf{x})}{1-\hat{p}(\mathbf{x})}\right)$
`"response"`	$\hat{p}(\mathbf{x}) = \frac{e^{\hat{\eta}(\mathbf{x})}}{1+e^{\hat{\eta}(\mathbf{x})}} = \frac{1}{1+e^{-\hat{\eta}(\mathbf{x})}}$

We'll look at the coefficients from the logistic regression and we see that they're not all that far from the truth.



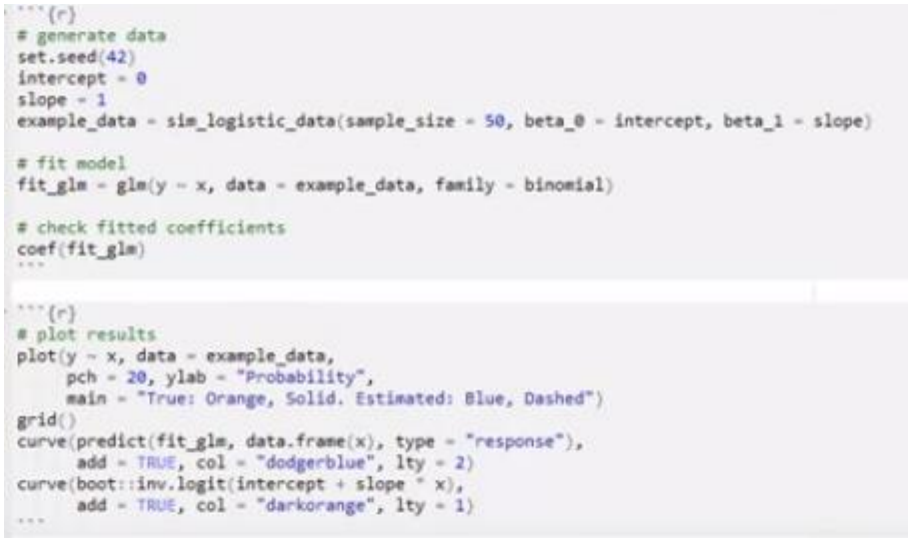
$$\hat{\eta}(\mathbf{x}) = -2.3 + 3.7x$$

~~~
```{r}
predict(fit_glm, newdata = data.frame(x = 1.2), type = "link")
```
~~~

we can use predict function in logistic regression but it operates a little bit differently. So, when we specify type equals <u>link</u>, we get estimated linear combination of the predictors or an estimate of the <u>log odds ratio</u> but that might not be a super intuitive. With type=<u>response</u> we get an estimate of the probability that y=1 given x is 1.2. We're using type equals response but it's not returning responses zero or ones, it's returning probabilities of y=1.

```
plot(y ~ x, data = example_data,
     pch = 20, ylab = "Estimated Probability",
     main = "Ordinary vs Logistic Regression")
grid()
abline(fit_lm, col = "darkorange")
curve(predict(fit_glm, data.frame(x), type = "response"),
      add = TRUE, col = "dodgerblue", lty = 2)
legend("topleft", c("Ordinary", "Logistic", "Data"), lty = c(1, 2, 0),
       pch = c(NA, NA, 20), lwd = 2, col = c("darkorange", "dodgerblue", "black"))
```

Okay, so we'll plot what we just did there. So, this dashed blue line is the estimated probability that y is one given different values of x and then this orange solid line is the ordinary linear regression fitted values for different values of x.



~~~
```{r}
# generate data
set.seed(42)
intercept = 0
slope = 1
example_data = sim_logistic_data(sample_size = 50, beta_0 = intercept, beta_1 = slope)

# fit model
fit_glm = glm(y ~ x, data = example_data, family = binomial)

# check fitted coefficients
coef(fit_glm)
```

```{r}
# plot results
plot(y ~ x, data = example_data,
     pch = 20, ylab = "Probability",
     main = "True: Orange, Solid. Estimated: Blue, Dashed")
grid()
curve(predict(fit_glm, data.frame(x), type = "response"),
      add = TRUE, col = "dodgerblue", lty = 2)
curve(boot::inv.logit(intercept + slope * x),
      add = TRUE, col = "darkorange", lty = 1)
```
~~~

So, what if we change the true values of the coefficients and plot. Start with beta_zero (intercept) = 0 and a beta_one (slope)= 1 , the estimated betas are fairly close. So I'm plotting two things. I'm plotting the <u>true</u> probability that y =1 given X and that is the orange solid curve (using the known values). Then

I'm also plotting the <u>estimated</u> probability that y is one given the values of the predictors and again that's that dash blue line here. notice that there's roughly the same amount of zeros for the response and one's in the response.

So now I'm going to do is change intercept=2 and regenerate this plot. now we have a lot more ones in the data for the response and far fewer zeros. If I change intercept again like -2, we see far more zeros in the data than ones in the data. So that tells us that this parameter here, it seems to have some control over the proportion of zeros and ones in the data.

So lets changes intercept back to zero and now we're going to look at the effect of slope parameter here. For beta_one =1 we see this increase in probability. So let's increase slope=2, there's a sharper increase in probability along the way. If we continue to increase this, this relationship will increase even faster. So, this beta one influences how sharp this increase is, sort of the curvature of the curve. If we instead made it negative, we'll see a decrease in probability that y=1 as x increases. If we make it say even more negative, it will be a steeper descent.

Let's make intercept=10 and slope = -10, we see some things there. warning that says "algorithms did not converge" and "fitted probabilities numerically zero or one occurred" and then our estimates for the intercept are 273 instead of 10 and our estimate for beta one instead of - 10 is - 211. So the algorithm used to fit this model is an iterative algorithm and for the number of iterations it was allowed to run, it didn't quite converge. I would stress that this is a warning not an error. So our models will still fit, it just didn't converge. we also see this other issue, fitted probabilities, numerical is zero or one occurred. So, we mentioned that to fit this model, we don't have an analytic solution, we have to use numerical methods, and those methods have issues when things like this happen, which is probability is very very very very very close to one. But again, I would stress that this is a warning not an error, so our model was fit, it's just, we have to be careful about how far off these coefficients are from the truth.

So, this model does okay in some regions, and terrible on other regions and these values are wildly off. So, we certainly understand that when we are doing estimation, there's some variability. but this is sort of a whole different issue. It's not just variability in estimation, it's literally an issue with the estimation. We can actually take a look at changing the seed value and see what happens. So, essentially when you see this warning, you should be very very very skeptical of your estimated coefficients. we can potentially still use the results of the logistic regression for the sake of classification, because we see that this dashed blue curves still gives us a sense of, should something be zero or one, it's just really bad with these in-between cases.

we'll move on to looking at data about males that live in a high-risk region and high risk of heart disease. this variable CHD, for whether or not an individual has coronary heart disease. I would think that it would be reasonable to try to model whether or not someone has coronary heart disease as a function of LDL. I will get the coefficients, this coefficient being positive, tells us that the higher the

LDL, the higher the log-odds ratio, therefore, the higher the probability that someone has coronary heart disease.

```r
# install.packages("ElemStatLearn")
library(ElemStatLearn)
data("SAheart")
```

```r
chd_mod_ldl = glm(chd ~ ldl, data = SAheart, family = binomial)
plot(jitter(chd, factor = 0.1) ~ ldl, data = SAheart, pch = 20,
     ylab = "Probability of CHD", xlab = "Low Density Lipoprotein Cholesterol")
grid()
curve(predict(chd_mod_ldl, data.frame(ldl = x), type = "response"),
      add = TRUE, col = "dodgerblue", lty = 2)
```

```r
coef(summary(chd_mod_ldl))
```

```
##               Estimate Std. Error  z value     Pr(>|z|)
## (Intercept) -1.9686681 0.27307908 -7.209150 5.630207e-13
## ldl          0.2746613 0.05163983  5.318787 1.044615e-07
```

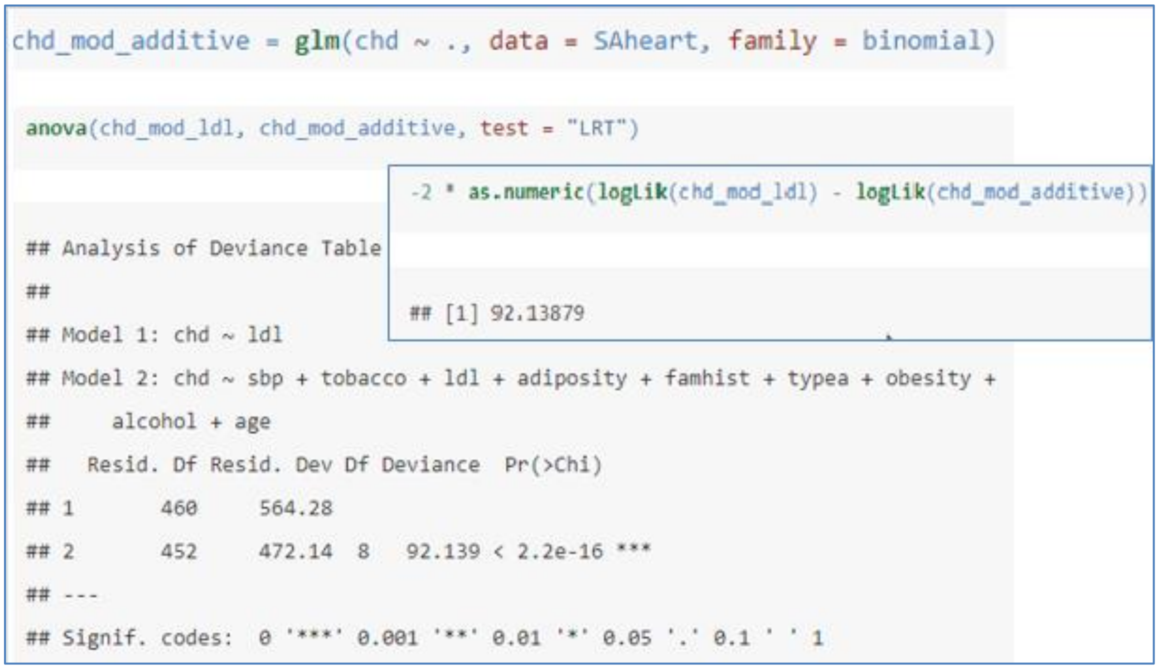
$$\log\left(\frac{P[\text{chd}=1]}{1-P[\text{chd}=1]}\right) = \beta_0 + \beta_{\text{ldl}} x_{\text{ldl}}$$

We've jittered the data so we can see it, there's a lot of it this time. We see that, maybe not a sharp increase, but certainly an increase we estimate in probability of CHD as LDL cholesterol increases.

But the question is, "is that significant?" The tests that we'd like to do, is test whether or not that coefficient in front of LDL is actually significant. Is this relationship just due to chance, or are we actually seeing a relationship here? run the summary function - The difference now is we have a z-value instead of a t-value, and a probability according to this z, since standard normal distribution for the p-value. The p-value for the test that we're interested is extremely low, we will reject this null hypothesis, we would actually say this relationship is significant.

some of these variables should be fairly useful for understanding CHD. What we'll do is compare different models using the likelihood ratio tests. Two models, one is nested inside of the other, we have this test statistic here, which approximately follows a chi-square distribution, with the degrees of freedom of the difference of the size of the models. We'll go ahead and fit just model with all of these variables here, and we'll call that the additive model. So, now we have two models, one just with LDL one with all possible predictors. compare these two models the difference is setting a bunch of the coefficients to be equal to zero. So, I could manually calculate this test statistic, it would look something like this and then I could manually calculate the p-value, but I don't want to do that.

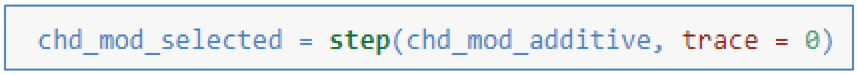
perform a specific test that is the likelihood ratio test and when we do that we have what's called analysis of deviance table. here we see the value of the test statistic that we had already calculated ourselves, but then it also automatically calculates the p-value for us which in this case is extremely small so we would reject the null hypothesis. So, we prefer the larger model because it seems that LDL does not explain the relationship, as well as, using additional variables.



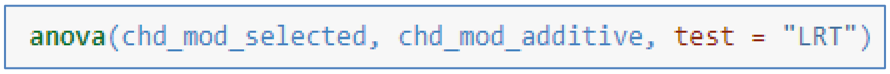
```
chd_mod_additive = glm(chd ~ ., data = SAheart, family = binomial)

anova(chd_mod_ldl, chd_mod_additive, test = "LRT")
```

```
-2 * as.numeric(logLik(chd_mod_ldl) - logLik(chd_mod_additive))

## [1] 92.13879
```

```
## Analysis of Deviance Table
##
## Model 1: chd ~ ldl
## Model 2: chd ~ sbp + tobacco + ldl + adiposity + famhist + typea + obesity +
##     alcohol + age
##   Resid. Df Resid. Dev Df Deviance  Pr(>Chi)
## 1       460     564.28
## 2       452     472.14  8   92.139 < 2.2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```



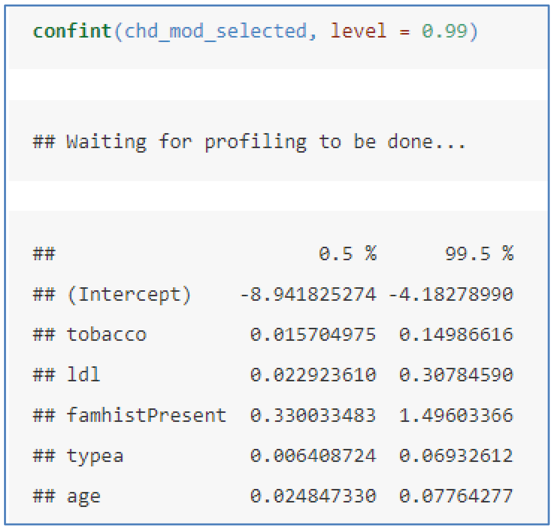
```
chd_mod_selected = step(chd_mod_additive, trace = 0)
```



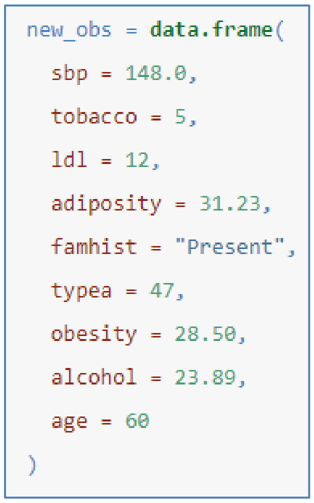
```
anova(chd_mod_selected, chd_mod_additive, test = "LRT")
```

but perhaps there's a model with some subset of the variables that is preferred. So, we'd like to do model selection. what this is going to do is work backwards and check AICs to arrive at a selected model. So, we could compare this model we selected to the additive model that would be another instance of the ratio test as we have two nested models and we see that we still prefer the smaller of those who model so we like this model that we had selected. we could call the summary function on it, so each individual line here is performing a test about a single parameter given that the other parameters in the model, much like we had seen in ordinary linear regression.

coef(chd_mod_selected) - So, here we have the estimates of the model parameters, but again, we're performing estimation here, so there's certainly some variability there. So, perhaps we would like to instead of just use point estimates, we'd like to create interval estimates.



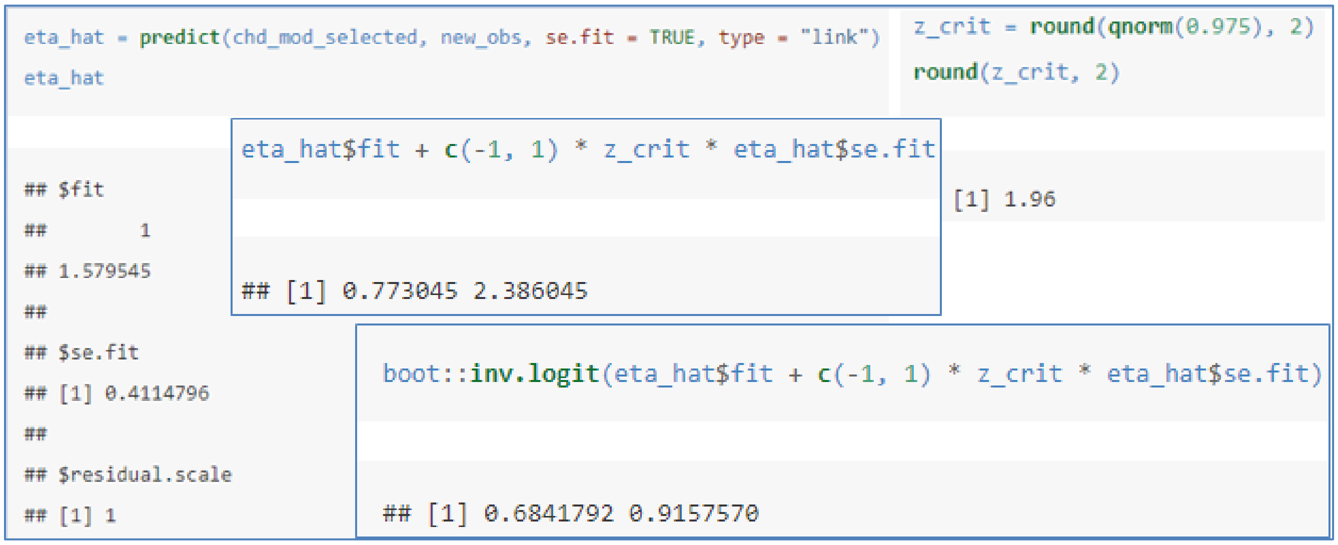
```
confint(chd_mod_selected, level = 0.99)

## Waiting for profiling to be done...

##                       0.5 %      99.5 %
## (Intercept)    -8.941825274 -4.18278990
## tobacco         0.015704975  0.14986616
## ldl             0.022923610  0.30784590
## famhistPresent  0.330033483  1.49603366
## typea           0.006408724  0.06932612
## age             0.024847330  0.07764277
```



```
new_obs = data.frame(
  sbp = 148.0,
  tobacco = 5,
  ldl = 12,
  adiposity = 31.23,
  famhist = "Present",
  typea = 47,
  obesity = 28.50,
  alcohol = 23.89,
  age = 60
)
```

```r
predict(chd_mod_selected, new_obs, type = "response")
```

So, here we had an estimate of 1.6 for LDL's coefficient or we could instead use this 99 percent confidence interval here. What about confidence interval for the mean response for this new person. I can predict this person's probability of CHD, so there's roughly 83% chance that this person has CHD. But maybe I want an interval estimate here.



```r
eta_hat = predict(chd_mod_selected, new_obs, se.fit = TRUE, type = "link")
eta_hat
```

```
## $fit
##        1
## 1.579545
##
## $se.fit
## [1] 0.4114796
##
## $residual.scale
## [1] 1
```

```r
z_crit = round(qnorm(0.975), 2)
round(z_crit, 2)
```

```
[1] 1.96
```

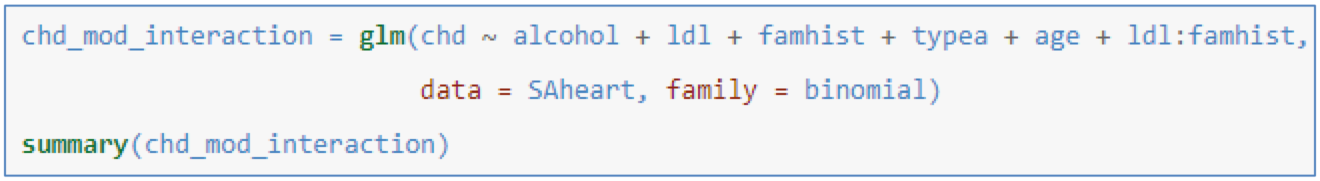
```r
eta_hat$fit + c(-1, 1) * z_crit * eta_hat$se.fit
```

```
## [1] 0.773045 2.386045
```

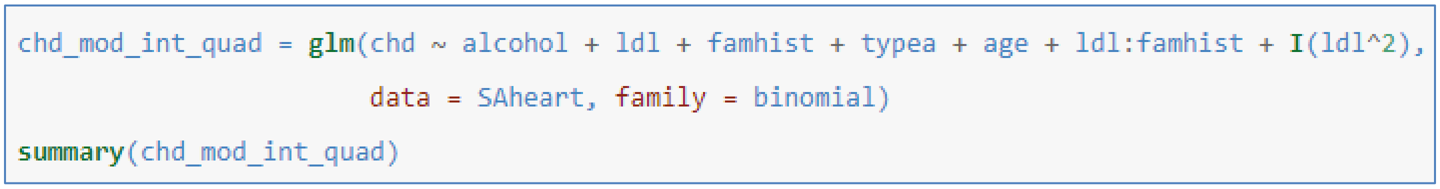
```r
boot::inv.logit(eta_hat$fit + c(-1, 1) * z_crit * eta_hat$se.fit)
```

```
## [1] 0.6841792 0.9157570
```

So, what we saw as we could first make an interval estimates for the log odds. Get eta_hat and by using this se.fit equals true, I can also get the standard error for that. Then if I acquire the correct critical value I could get a confidence interval then for the log odds. So this is a confidence interval for the log odds, but again that might not be as uninterpretable if you're not super familiar with log odds. So instead, we'll use this inverse logit function from the boot library instead of writing our own which will then convert this interval for the log odds to the probability of Y equals one which in this case is presence of CHD. So, we get an interval estimate instead of a point estimate which is often useful.



```r
chd_mod_interaction = glm(chd ~ alcohol + ldl + famhist + typea + age + ldl:famhist,
                          data = SAheart, family = binomial)
summary(chd_mod_interaction)
```



```r
chd_mod_int_quad = glm(chd ~ alcohol + ldl + famhist + typea + age + ldl:famhist + I(ldl^2),
                       data = SAheart, family = binomial)
summary(chd_mod_int_quad)
```

just want to specifically say that logistic regression and really any GLMs, is specifying this linear combination of the predictors and also specifying this is the binary response to them. simply think of deviance as the analog to residual sum of squares. So, it's some measurement of how much error there is in the model. So, those are three models that are nested inside of each other growing in number of parameters, and the thing you might notice is that deviance decreases. That is something similar to what we saw with training root mean squared error or residual sum of squares in ordinary linear regression but we'll come back to this.

The Null deviance is the deviance for the null model, that is, a model with no predictors. The Residual deviance is the deviance for the mode that was fit. Deviance compares the model to a saturated model. (Without repeated observations, a saturated model is a model that fits perfectly, using a parameter for each observation.) Essentially, deviance is a generalized residual sum of squared for GLMs. Like RSS, deviance decreased as the model complexity increases. we see that deviance does decrease as the model size becomes larger. So while a lower deviance is better, if the model becomes too big, it may be overfitting. Note that R also outputs AIC in the summary, which will penalize according to model size, to prevent overfitting.

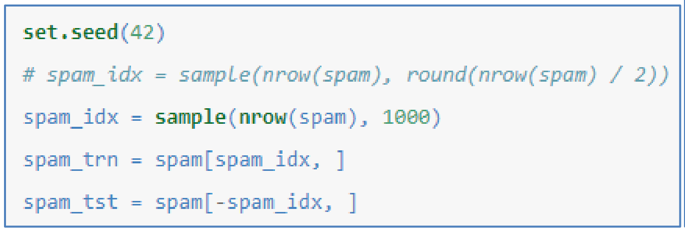
So far we've mostly used logistic regression to estimate class probabilities. The obvious next step is to use these probabilities to make "predictions," which in this context, we would call classifications. Based on the values of the predictors, should an observation be classified as Y=1 or as Y=0 ? So now we'll go ahead and take a look at a very quick introduction to classification in R.

Simply put, the Bayes classifier (not to be confused with the Naive Bayes Classifier) minimizes the probability of misclassification by classifying each observation to the class with the highest probability. Unfortunately, in practice, we won't know the necessary probabilities to directly use the Bayes classifier. Instead we'll have to use estimated probabilities.

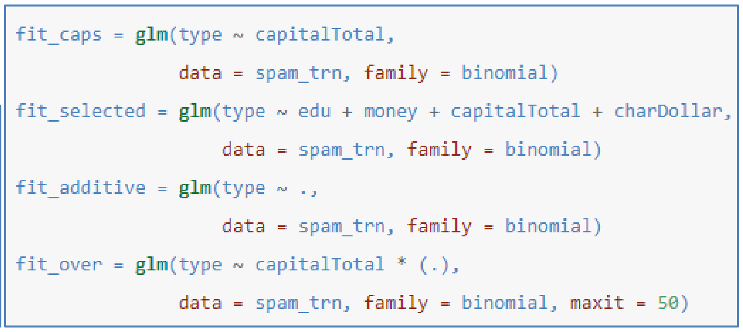
$$\boxed{\hat{C}(\mathbf{x}) = \begin{cases} 1 & \hat{p}(\mathbf{x}) > 0.5 \\ 0 & \hat{p}(\mathbf{x}) \leq 0.5 \end{cases}}$$

To use logistic regression for classification, we first use logistic regression to obtain estimated probabilities, then use these in conjunction with the above classification rule. Logistic regression is just one of many ways that these probabilities could be estimated. In a course completely focused on machine learning, you'll learn many additional ways to do this, as well as methods to directly make classifications without needing to first estimate probabilities. So, we're going to continue to restrict ourselves to the binary case. You could do classification with more categories but for our purposes we'll consider just the two class case. we will create a Bayes classifier that classifies to one if the probability is greater than 0.5 in the binary case or if this probability was less than 0.5 classified as zero.

we're going to look at this spam data set from the Kernlab library. we essentially want to predict whether or not an email is spam or non-spam. so we see a difference in this data set than the previous data sets that this variable is a factor variable with two levels non-spam and spam. the model fitting this variable will be recognized as zero one, the first level will correspond to zero, so non-spam will be zero and the second level spam that will be one. Okay, so because we're dealing with prediction here we're going to split up the data into a training set and a test set.



```r
set.seed(42)
# spam_idx = sample(nrow(spam), round(nrow(spam) / 2))
spam_idx = sample(nrow(spam), 1000)
spam_trn = spam[spam_idx, ]
spam_tst = spam[-spam_idx, ]
```



```r
fit_caps = glm(type ~ capitalTotal,
               data = spam_trn, family = binomial)
fit_selected = glm(type ~ edu + money + capitalTotal + charDollar,
                   data = spam_trn, family = binomial)
fit_additive = glm(type ~ .,
                   data = spam_trn, family = binomial)
fit_over = glm(type ~ capitalTotal * (.),
               data = spam_trn, family = binomial, maxit = 50)
```

so we're going to look at four different models, over fitting model. Which uses all the predictors and the interaction between this total variable and all the other predictors. So, one thing to note is that these models are all nested. so this warning suggests we should be suspicious of these estimates of the model parameters but it's still reasonable to evaluate using this model for a classifier and we're not actually interested in the estimates of the coefficients, we're not interested in which variables are significant. So we wouldn't be doing tests that would be unreliable because these coefficients are highly suspicious. We are not doing inference we are interested in making predictions. So, we're going to evaluate how good we are at making predictions. centrally maximum iterations - because the default number of iterations was not enough to converge.

we would like to use each of the models as a classifier. We would use their predictive probabilities, make classifications and evaluate the misclassification rate. So, essentially we want to know how often are we making incorrect classifications? If i just call predict without giving it additional data, it will use whatever data was used to fit this model. So, these are predictions for each of the thousand observations that were used to fit this model. They are not the estimated probability, they are the log odds. So, a predictive probability of 0.5 corresponds to an estimated log odds of zero. So, essentially what I want to know is if we have a high log odds here that is greater than zero, we would want to predict spam. Whereas, if we have a low log odds here we want to predict non-spam. We can say - type equals response. and now we see these are actually predicted probabilities p(x). if these are greater than 0.5, I would want to return a particular value.

```r
mean(ifelse(predict(fit_caps) > 0, "spam", "nonspam") != spam_trn$type)
mean(ifelse(predict(fit_caps, type=”response”) > 0, "spam", "nonspam") !=
spam_trn$type)
```

So, this is essentially making classifications on the training data. Recall that, this probability which are being returned here is a probability of one which in this case is spam due to the levels of that factor. an equivalent way of doing this would be to not compare to the response, but instead to log odds and those being greater than zero. we're going to see how many of them are not equal to the true value and take the average of those and we'll do that for all four models.

So running this, so this first model the one and only had a single predictor has a miss classification rate and the training of 0.34, the selected model has 0.21, the additive model 0.064 and the huge model with all the interaction terms, 0.063. So the model that has the lowest misclassification rate in the training data is the biggest model. This should sort of be no surprise. This is because the bigger the model is the fewer errors it makes. But that doesn't indicate to us which of these will make predictions on unseen data well. Essentially, this gives us no indication of if any of these models are overfitting. This is the same phenomenon we saw in regression where the bigger the model the smaller the residual sum of squares or training root mean squared error.

We will cross validate these models. So cross-validation with logistic regression, we'll want to use the cv.glm function from the boot library. So back in ordinary linear regression we were able to automatically and quickly do leave-one-out cross-validation due to a nice old trick, but that trick doesn't exist in logistic regression. So we'll want to use either 5-fold or 10-fold cross-validation, and essentially this will require fitting five times so it will be a little bit slower, but it will give us an average of a misclassification rate over five different held out sets. We're skipping a lot of the details here but essentially, this will be a useful metric whereas this training misclassification rate not so much.

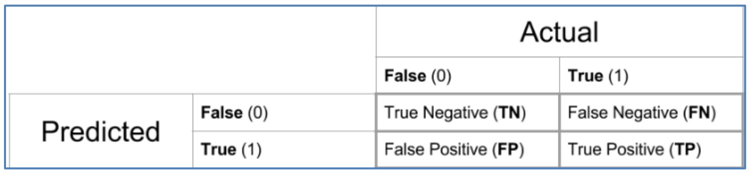
```
library(boot)
set.seed(1)
cv.glm(spam_trn, fit_caps, K = 5)$delta[1] = 0.2166961
cv.glm(spam_trn, fit_selected, K = 5)$delta[1] = 0.1587043
cv.glm(spam_trn, fit_additive, K = 5)$delta[1] = 0.08684467
cv.glm(spam_trn, fit_over, K = 5)$delta[1] = 0.135
```

Based on these results, `fit_caps` and `fit_selected` are underfitting relative to `fit_additive`. Similarly, `fit_over` is overfitting relative to `fit_additive`. Thus, based on these results, we prefer the classifier created based on the logistic regression fit and stored in `fit_additive.` So, the worst cross-validated misclassification rate was for the smallest model, and then it gets better as model gets bigger and then it eventually gets worse. That's why we're saying this model is overfitting relative to the best model which was the additive model, and these two models are underfitting relative to this model. So between these four models, add them all right now, we like the additive models.

So misclassification rate is certainly one metric we should look at. We want to be making as few errors as possible especially on unseen data, so either via cross-validation or just using a single held out test set. So it turns out that not all errors are created equal. So we have here is what's called a confusion matrix, so we can compare the actual values (we use false and zero interchangeably, and true and one interchangeably) versus the predicted values. So if the true value is one and the predicted value is one, that's what we call a true positive, we like that or if the actual value is zero and the predicted value zero, that's true negatively like that, but there's two types of errors. If the actual value is one and we predict a zero that's a false negative whereas if the actual value is false and we return or predict a true, that will be false positive. So, whether or not a false positive or a false negative is worse depends on the situation.

So in this case, the positives, the trues, the ones are spam and the falses, the zeros are non-spam. So, consider the two errors here. First consider a false negative, an email that truly is spam, but we label/predict/classify it to be non-spam. That's an error, but this is an error that's not the worst thing. So in the email situation, I think we would be a little bit more worried about false positives instead about false negatives.

Confusion Matrix - It further breaks down the classification errors into false positives and false negatives.



		Actual	
		False (0)	**True** (1)
Predicted	**False** (0)	True Negative (**TN**)	False Negative (**FN**)
	True (1)	False Positive (**FP**)	True Positive (**TP**)

```
make_conf_mat = function(predicted, actual) {
  table(predicted = predicted, actual = actual)   }

# spam_tst_pred = ifelse(predict(fit_additive, spam_tst) > 0, "spam", "nonspam")
spam_tst_pred = ifelse(predict(fit_additive, spam_tst, type = "response") > 0.5,
                "spam",  "nonspam")
```

a log odds ratio of zero is equivalent to a probability of 0.5 so these two bits of code would both create the same classifier OR $\eta(x)=0 \quad \Leftrightarrow \quad p(x)=0.5$

Now we'll use these predictions to create a confusion matrix. look at metrics on the test data set. we want to use unseen data to evaluate false positives and false negatives. About this diagonal we have the errors, so these are true spams that are marked as non-spam that we're not too worried about, and here we have non-spam emails are marked as spam, we don't like that.

```
(conf_mat_50 = make_conf_mat(predicted = spam_tst_pred, actual = spam_tst$type))
##            actual
## predicted  nonspam spam
##   nonspam     2057  157
##   spam         127 1260
```

So we want to introduce three new ideas now, the prevalence, the sensitivity and specificity. So, the prevalence is just the prevalence of the class labeled as one. So, the total number of positives in the dataset divided by the total number of observations. So, the prevalence in this dataset is about 40 percent. So, we have about 40 percent spam in this dataset. A really obvious classifier is to classify to the majority class, non-spam (60%). This would be our base error rate (40%). So, basically, if we don't get an error rate better than this or models pride pretty bad. We get an error rate of about eight percent, which sometimes instead of misclassification rate, people will report the accuracy, which is just one minus that so about 92 percent. So, obviously, this classifier using that additive logistic regression is doing much better than simply guessing based on the majority class.

$$\boxed{\text{Prev} = \frac{\text{P}}{\text{Total Obs}} = \frac{\text{TP} + \text{FN}}{\text{Total Obs}}}$$

```
table(spam_tst$type) / nrow(spam_tst)
##
```

```
##   nonspam      spam
## 0.6064982 0.3935018
mean(spam_tst_pred == spam_tst$type)   0.921133  # Test accuracy
mean(spam_tst_pred != spam_tst$type)   0.07886   # Error/Misclassification rate
```

$$\boxed{\text{Sens} = \text{True Positive Rate} = \frac{\text{TP}}{\text{P}} = \frac{\text{TP}}{\text{TP} + \text{FN}}} \quad \boxed{\text{Spec} = \text{True Negative Rate} = \frac{\text{TN}}{\text{N}} = \frac{\text{TN}}{\text{TN} + \text{FP}}}$$

consider a new metric called sensitivity, which is basically the true positive rate. number of true positives divided by the number of positives in the data set. So, high sensitivity is low false negatives. specificity is the true negative rate. So high specificity is high is low false positives. So, we have, a sensitivity of about 88 and a specificity of about 93. So, in this case, because we said we want to guard against false positives, because that would be labeling non-spam as spam, so we want this number to be high. there's a trade off between these two quantities. You can sacrifice a little bit of specificity to get more sensitivity or you can sacrifice a little bit of sensitivity to get more specificity.

```
get_sens = function(conf_mat) {
  conf_mat[2, 2] / sum(conf_mat[, 2])   }

get_spec =  function(conf_mat) {
  conf_mat[1, 1] / sum(conf_mat[, 1])   }

get_sens(conf_mat_50)   # 0.8892025
get_spec(conf_mat_50)   # 0.9418498
```

So, the natural thing to do is to use this 0.5 as a cutoff. If the probability that something is spam is greater than 0.5, label it as spam. 0.5 is the number if we want to essentially make the misclassification rate as low as possible. But if we're willing to sacrifice this overall rate for one of the two specific rates, we can make this potentially better. instead of using 0.5 here, we could say perhaps lower the threshold. Maybe we label for something is spam if there's a 10 percent chance or better, which is what we could do here.

```
spam_tst_pred_10 = ifelse(predict(fit_additive, spam_tst, type = "response") > 0.1,
                          "spam",
                          "nonspam")

(conf_mat_10 = make_conf_mat(predicted = spam_tst_pred_10, actual = spam_tst$type))
##          actual
## predicted nonspam spam
##   nonspam    1583   29
##   spam        601 1388
```

So, this is creating a slightly different classifier using the same logistic regression. So, we're getting the same predicted probabilities, but now, anything with a predicted probabilities spam greater than 0.1, we will label spam, everything else non-spam. We can create a confusion matrix with that. So, we've greatly reduced the false negatives (29), but with a lot more false positives (601). this is essentially just decreasing threshold to label something or to predict something as spam. get a lot more false positives. So, we see that our specificity has gone down a lot at an increase sensitivity, but we want to do actually the exact opposite of that. So, what we'll instead do is we'll increase the threshold for something being labeled as spam. So, instead of 0.5, we'll try 0.9. So, essentially, we have to be very certain of something being spam to classify it to spam. So, now, we have a lot more false negatives, but far fewer false positives partially because we're simply predicting less things to be labeled as spam. We'll see that we get much lower sensitivity at the trade-off of much greater specificity.