Practical Predictive Analytics Seminar: Modeling and validation

Contents

Intro

We will discuss questions of interest for life and annuity products, as well as the predictive model forms that are best suited to investigating them. There will be some focus on the corresponding theoretical concerns that may arise in the modeling process. We will set the stage for the later session to address more practical concerns by introducing several concepts such as identifying and dealing with outlier data values, accounting for missing values, using the step() function for variable selection, identifying correlated variables, setting reference levels for factor variables, and testing and improving the model fit across the range of each covariate. We will also explore a technique to improve computational efficiency for logistic GLMs. Finally, we will discuss assessing overall model fit and comparison between two candidate models.

Load packages

```
library(dplyr)
library(lubridate)
library(car)
library(ggplot2)
library(tidyr)
```

Import cleaned data. We saved it in two formats (.Rdata and .rds) for demonstration. You can read .rds files into the environment with any object name you want, but you can only save one R object per .rds file. With .Rdata files, you can save many R objects into a single file, but you're stuck with the object names you gave them originally.

```
load("PPASExpandedData.Rdata")
data.large <- readRDS("PPASExpandedData.rds")
```

1. Data exploration

Take a quick look at the variables in the dataset and their respective summaries.

```
summary(data.large)
```

2. Modeling

A. Training, holdout, and testing datasets

Now we'll split the dataset into a 50% training sample, a 25% in-time holdout sample, and a 25% out-of-time holdout sample using a column called "Sample". We set what's called a seed for the random number generator so that we all get the same data subsets.

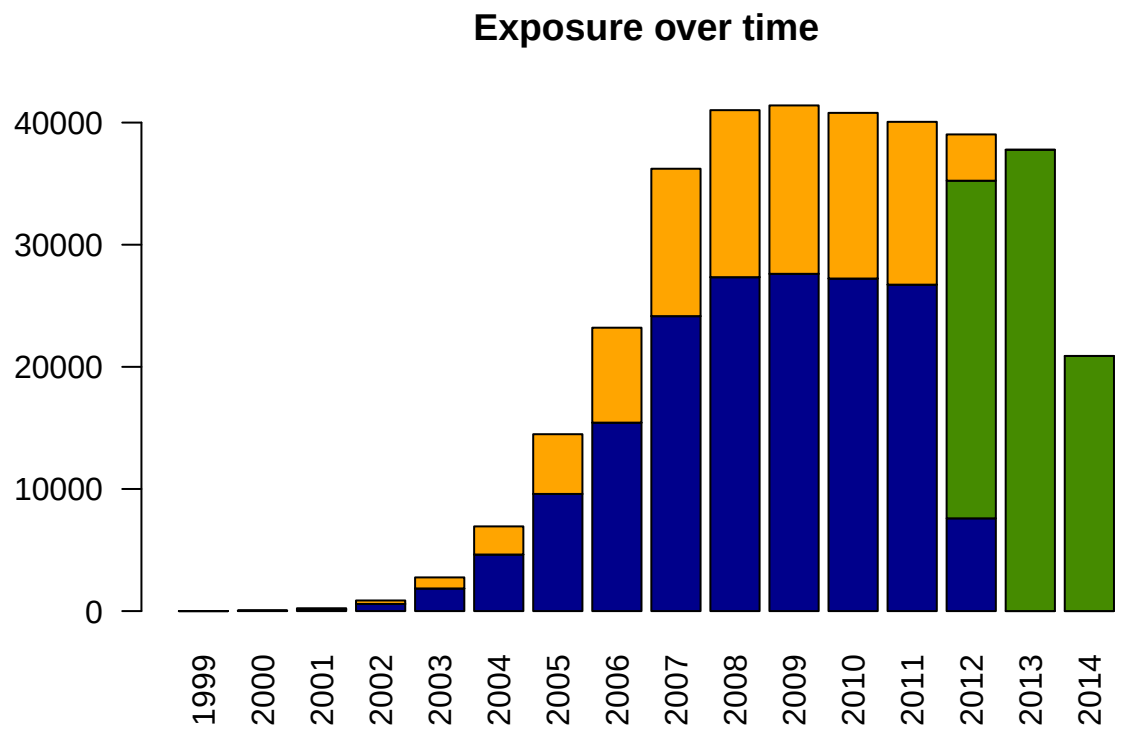
```
(test.cut.date <- quantile(data.large$current.date, .75, type = 1))
```

```
           75%
   "2012-04-18"
```

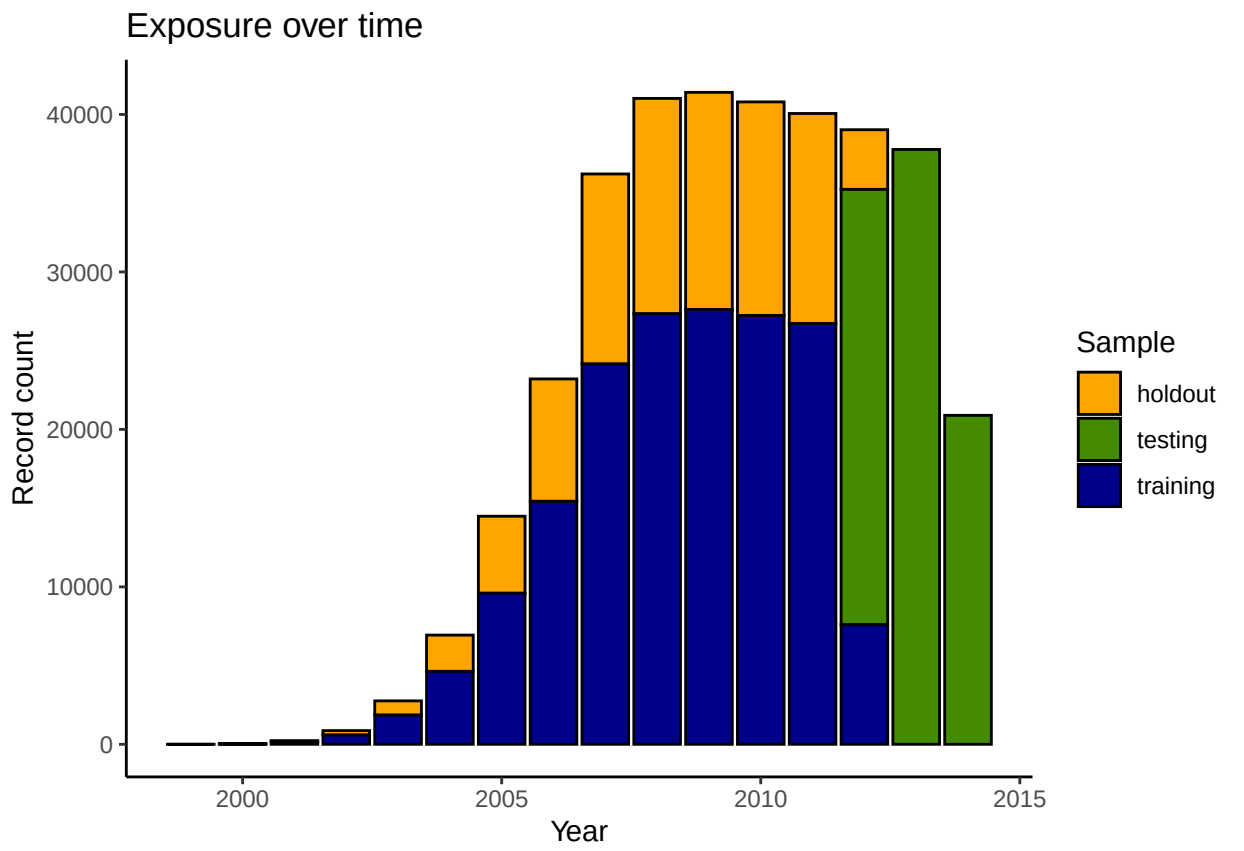
```
set.seed(1)
sample.rand <- data.frame(PolNum = 1:45000,
                          RandNum = runif(45000, 0, 1))
data.large <- data.large %>%
  left_join(sample.rand,
            by = "PolNum")
data.large <- data.large %>%
  mutate(Sample = ifelse(current.date > test.cut.date, "testing",
                         ifelse(RandNum > 2/3, "holdout", "training")))
```

Here we'll recreate the colored exposure plot from the presentation for those curious about how that's done. First using base R plotting, and then using ggplot.

```
data.large <- data.large %>%
  mutate(SampleColor = ifelse(Sample == "holdout",
                              "orange", ifelse(Sample == "training",
                                               "blue4", "chartreuse4")))
date.plottable <- table(data.large$SampleColor, year(data.large$current.date))
barplot(date.plottable,
        col = rownames(date.plottable),
        las = 2,
        main = "Exposure over time")
```




```
data.large %>%
  ggplot(aes(x = year(current.date), fill = Sample)) +
  geom_bar(color = "black") +
  scale_fill_manual(values = c("holdout" = "orange", "training" = "blue4", "testing" = "chartreuse4")) +
  labs(x = "Year", y = "Record count", title = "Exposure over time") +
  theme_classic()
```



B. Pivot Tables

You may find that you want to look at some tables before diving right into modeling. Here is a reminder on how to make a pivot table, using mortality versus activity level as an example:

```
data.large %>%
  group_by(activitylevel) %>%
  summarize(Obs = n(),
            Deaths = sum(Death),
            MortRate = Deaths/Obs)
```

```
  # A tibble: 4 x 4
    activitylevel    Obs Deaths MortRate
    <fct>          <int>  <dbl>    <dbl>
  1 unreported     17793    351   0.0197
  2 active         37825    839   0.0222
  3 average       287747   7019   0.0244
  4 sedentary       2358    191   0.0810
```

C. Fitting a model

Let's begin with a smaller model. Here we'll suppose that deaths can be predicted by age, the cancer indicator, the smoker indicator, and gender (indicator for male). Note that we subset the data to only use the training portion.

```
form.mid <- as.formula(Death ~ AttAge + cancer_ind + smoker + gender)
model.mid <- glm(formula = form.mid,
                 family = binomial(link = "logit"),
```

```
                data = data.large %>%
                  filter(Sample == "training"))
summary(model.mid)
```

```
Call:
glm(formula = form.mid, family = binomial(link = "logit"), data = data.large %>%
    filter(Sample == "training"))

Deviance Residuals:
    Min       1Q   Median       3Q      Max
-0.9237  -0.2452  -0.1848  -0.1397   3.7397

Coefficients:
              Estimate Std. Error z value Pr(>|z|)
(Intercept) -14.614857   0.233202 -62.670  < 2e-16 ***
AttAge        0.132273   0.002783  47.533  < 2e-16 ***
cancer_ind    0.197248   0.047340   4.167 3.09e-05 ***
smokerS       0.756202   0.067335  11.230  < 2e-16 ***
genderM       0.338423   0.033558  10.085  < 2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

(Dispersion parameter for binomial family taken to be 1)

    Null deviance: 39113  on 172948  degrees of freedom
Residual deviance: 36578  on 172944  degrees of freedom
AIC: 36588

Number of Fisher Scoring iterations: 7
```

As expected, each of the variables we chose corresponds to increased probability of death in the coming year, which we can see from the positive coefficients. The statistical significance of the relationships are shown in the P(>|z|) column, which gives the "Wald"" p-values testing whether each coefficient is different from 0.

D. Stepwise Variable Selection

Stepwise selection algorithms are convenient methods for paring down a lot of variables into a smaller model. It is best to allow your algorithm to both add and subtract variables at any given step, as it is common for a variable's importance to be contingent on the combination of other variables in the model. We will refer to this as bi-directional stepwise selection.

Stepwise approaches are not a replacement for thinking! Be sure to understand the variables being tested and the direction of the effect you should expect. Because stepwise approaches are effectively testing hundreds or even thousands of potential models, p-values are no longer a meaningful way to interpret the statistical significance of the coefficients. After using stepwise variable selection, robust validation procedures using holdout datasets, as described later in this document, must be used to confirm that the model is not overfit. Also, look out for correlated predictor variables here. We discuss in the Multicollinearity section.

i) Middle-out approach Now let's take a look at the code and output for R's stepwise function. First we'll need to input a biggest possible model formula object. For example's sake, we have limited the algorithm to just one step here because even that took nearly a minute on my computer. Notice that you can wrap any function with the system.time() function to time it.

```r
form.big <- as.formula(Death ~ smoker + gender + activitylevel +
                         positivefamilylongevity + adls + pulmonary + depression +
                         cognitive + alcohol + cad + tia + parkinsons + diabetes +
                         cancer_prostate + cancer_breast + cancer_colon + cancer_pancreatic +
                         cancer_lung + cancer_hodgkins + cancer_leukemia + cancer_myeloma +
                         cancer_liver + cancer_brain + cancer_ind + ht.wt.flag +
                         height + weight + bmi + bmisqrerr + AttAge)
```

```r
system.time(stepmodel.mid <- step(object = model.mid,
                                  scope = form.big,
                                  direction = "both",
                                  steps = 1))
```

```
Start:  AIC=36587.7
Death ~ AttAge + cancer_ind + smoker + gender

                           Df Deviance   AIC
+ cad                       3    36163 36179
+ cognitive                 3    36173 36189
+ tia                       3    36263 36279
+ diabetes                  3    36310 36326
+ adls                      1    36332 36344
+ activitylevel             3    36414 36430
+ pulmonary                 2    36434 36448
+ depression                2    36446 36460
+ parkinsons                1    36448 36460
+ alcohol                   3    36465 36481
+ positivefamilylongevity   1    36534 36546
+ bmisqrerr                 1    36551 36563
+ cancer_prostate           1    36562 36574
+ cancer_lung               1    36566 36578
+ cancer_leukemia           1    36568 36580
+ ht.wt.flag                1    36569 36581
+ cancer_brain              1    36571 36583
+ height                    1    36571 36583
+ cancer_pancreatic         1    36571 36583
+ bmi                       1    36575 36587
<none>                            36578 36588
+ cancer_myeloma            1    36576 36588
+ cancer_hodgkins           1    36577 36589
+ cancer_colon              1    36577 36589
+ weight                    1    36577 36589
+ cancer_breast             1    36578 36590
+ cancer_liver              1    36578 36590
- cancer_ind                1    36594 36602
- gender                    1    36682 36690
- smoker                    1    36682 36690
- AttAge                    1    38979 38987

Step:  AIC=36179.1
Death ~ AttAge + cancer_ind + smoker + gender + cad
```

```
   user  system elapsed
  24.86    4.03   28.89
```

This function aims to optimize the model based on Akaike's Information Criterion (AIC) by default. Even a single step provides some interesting output. The coronary artery disease (cad) variable would lower the AIC by the most; in other words, of all three- and five-variable models you could get to in one step from this one, this is AIC's best model. Additionally, the output ranks each possible step in order of how much it would improve the model. A variable's predictive ability is tied to which other variables are currently in the model, so we shouldn't necessarily expect that the next steps will be to add "cognitive", "tia" and "diabetes" in that order.

To make stepwise a little faster, we could reduce the number of models it has to look at by inputing a different biggest model formula. Note that we've chosen variables for this stepwise selection that were among the top variables from the last stepwise function. This method took about half as long to arrive at the one-step model.

```
form.medium <- as.formula(Death ~ AttAge + cancer_ind +
                            smoker + gender + cad + cognitive +
                            tia + diabetes + adls +
                            activitylevel + ht.wt.flag +
                            height + weight + bmi)
```

```
system.time(step(object = model.mid,
                 scope = form.medium,
                 direction = "both",
                 steps = 1))
```

ii) Top-down approach Generally, a top-down approach begins with considering every possible predictor variable that is available. Variables are then removed from the model based on the criteria of choice. These stepwise methods can be very computationally intensive. Taking advantage of parallel computing and other, more efficient, machine learning algorithms is often necessary. We won't cover those here, but for completeness we will include the code for a full, one-directional backward stepwise method.

```
system.time(step(glm(form.big,
                     family = binomial,
                     data = data.large %>%
                       filter(Sample == "training")),
                 direction = "backward",
                 k = 2)) # This is the default "k" value, which leads to AIC criterion.
```

iii) Bottom-up approach Generally, a bottom-up approach starts with an empty model and variables are added based on some criteria. Even an algorithmic stepwise approach has some options for you to consider. The direction of the algorithm, as discussed previously, indicates whether you will be adding variables, removing variables, or both. The maximum number of steps to make is fairly straight forward, and another option is which criteria is used to make these additions and subractions. The AIC and the Bayesian Information Criterion (BIC) are two common options. Here we'll take a look at a one-step, one-directional stepwise algorithm using BIC as the criterion for selection. BIC penalizes the addition of variables to the model more so than AIC, and this makes it attractive for a researcher looking to avoid overfitting the model.

Set up the empty, "intercept" model:

```
model.int <- glm(Death ~ 1,
                 family = binomial(link = "logit"),
                 data = filter(data.large, Sample == "training"))
```

Run "forward selection" with the BIC criterion.

```
(data.dim <- dim(filter(data.large, Sample == "training")))
system.time(step(model.int,
```

```
                  scope = form.medium,
                  direction = "forward",
                  steps = 1,
                  k = log(data.dim[1]))) ## Here is how we input the BIC criterion
```

The output suggests that the most important variable out of the gate is one's attained age (AttAge).

Let's make our final model from this section the result of a BIC, bottom-up model selection process. Note: this line of code takes up to 30 minutes to run. You can skip to the results in the next code chunk.

```
system.time(stepmodel.final <- step(model.int,
                                    scope = form.big,
                                    direction = "both",
                                    k = log(data.dim[1])))
```

After about 30 minutes, we have arrived at a model that we'll use going forward.

```
form.final <- as.formula(Death ~ AttAge + cad + cognitive + diabetes +
                           alcohol + pulmonary + adls + smoker +
                           parkinsons + tia + gender + activitylevel +
                           positivefamilylongevity + depression +
                           cancer_ind + ht.wt.flag + height + bmisqrerr)
model.final <- glm(form.final,
                   family = binomial,
                   data = data.large %>%
                     filter(Sample == "training"))
```

E. Multicollinearity

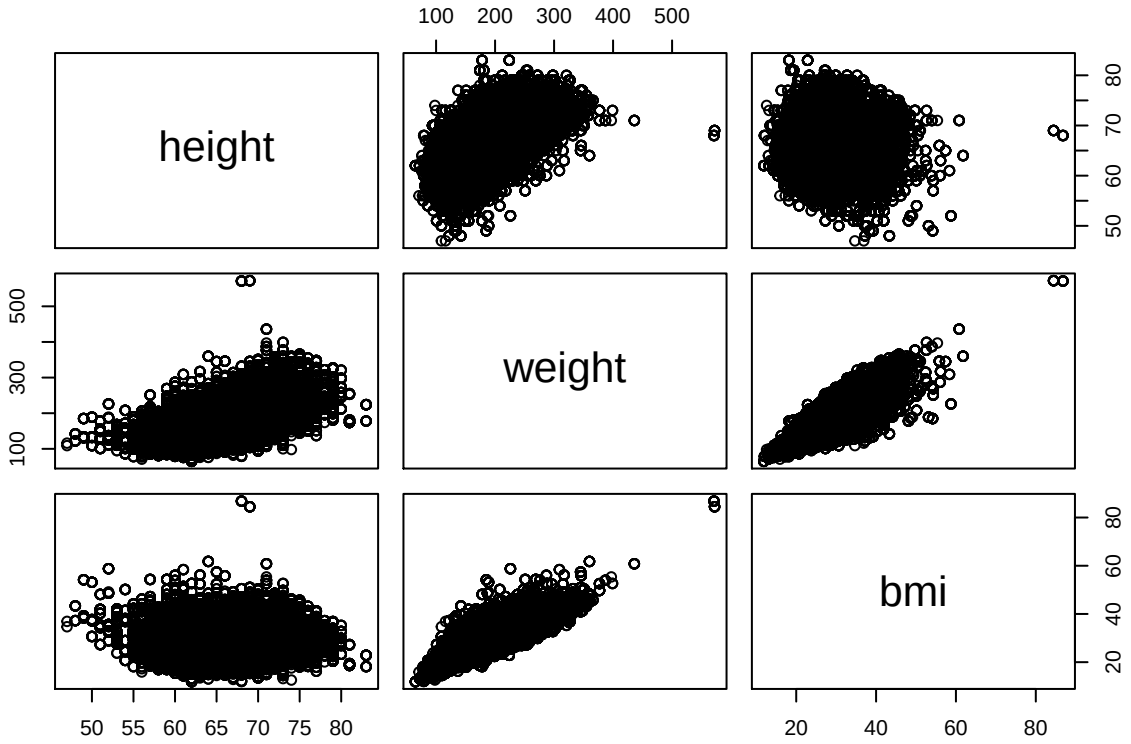
We recommended that you use bi-directional stepwise methods before, and that is true regardless of your criteria and whether you are going with a top-down or bottom-up approach. The correlation between variables in the dataset is referred to as multicollinearity, and its consequences require that the algorithm be able to "change its mind" about a variable at a later step. On that note, let's take a quick detour to assess the correlation between height, weight and BMI, and how that affects model fits.

If we check out the correlation between those three variables, we see some high linear correlations. That means the coefficients fit to each will depend very heavily on which of the others has also been included.

```
cor(data.large %>%
      filter(ht.wt.flag == 1) %>%
      select(height, weight, bmi))
```

```
              height    weight        bmi
height 1.00000000 0.6376396 0.05257831
weight 0.63763964 1.0000000 0.79571002
bmi    0.05257831 0.7957100 1.00000000
```

```
pairs(data.large %>%
        filter(ht.wt.flag == 1) %>%
        select(height, weight, bmi))
```



We'll fit a model with only height first, then add the other variables one at a time.

```
summary(glm(Death ~ ht.wt.flag + weight,
            family = binomial,
            data = data.large %>%
              filter(Sample == "training")))

summary(glm(Death ~ ht.wt.flag +
              weight + height,
            family = binomial,
            data = data.large %>%
              filter(Sample == "training")))

summary(glm(Death ~ ht.wt.flag +
              weight + height + bmi,
            family = binomial,
            data = data.large %>%
              filter(Sample == "training")))
```

If you're following along, you probably noticed that the role of the weight variable really yo-yoed. The variables in a linear model are like the players on a basketball team. Some players will play better with others, and when substitutions are made, the roles of those still on the court will probably change. Too many ball hogs–i.e. too many correlated variables–can lead to no one getting a good shot.

Following the outputs from the models above, here's a summary of what happened. Weight began as an insignificant predictor of death when it was alone in the model. Once we added height, weight became a significant predictor with a negative slope. But then, when BMI was added to the model, all variables became

statistically insignificant. This shows why it's important to monitor the linear correlation between variables in your model–and with variables that were excluded from the model. It can help you to avoid confusing models with unintuitive coefficients.

Now for our model. Let's assess the correlation between our model's variables in terms of Variance Inflation Factors (VIF's). The car package's vif() function utilizes a generalized formula for variance inflation factors to account for categorical predictor variables, but the concept is the same. The greater the VIF, the more correlated the predictor variable is to other predictor variables in the model.

```
vif(model.final)
```

```
                             GVIF Df GVIF^(1/(2*Df))
AttAge                   1.135080  1        1.065401
cad                      2.112910  3        1.132783
cognitive                1.316748  3        1.046929
diabetes                 1.108611  3        1.017333
alcohol                  1.285585  3        1.042758
pulmonary                1.024293  2        1.006019
adls                     1.422945  1        1.192873
smoker                   1.033750  1        1.016735
parkinsons               1.068499  1        1.033682
tia                      2.135209  3        1.134767
gender                   1.656793  1        1.287165
activitylevel            1.258939  3        1.039124
positivefamilylongevity  1.046588  1        1.023029
depression               1.213464  2        1.049559
cancer_ind               1.022717  1        1.011295
ht.wt.flag             106.239580  1       10.307259
height                 106.405379  1       10.315298
bmisqrerr                1.033724  1        1.016722
```

Recall that a minium VIF value is 1, which means that a particular variable is completely uncorrelated with all the other predictor variables in the model. Based on our experience, values as high as 3 or 4 should encourage you to reconsider the need for these variables; their information may be largely redundant. Note here that most of our variables are uncorrelated with the rest, as shown by the scaled, generalized VIF's in the final column. Using this metric, it becomes obvious why weight, height, and BMI can't all exist in the model as is. They are so linearly correlated to each other, that they render the linear model incapable of fitting intuitive coefficients.

F. Finding non-linear relationships

Linear models are more flexible than you might think, and you can manipulate the variables in the model to test for non-linear relationships. To do so, we typically look at how well the model predicted actual mortality (in this case) across the range of each variable. By looking at the discrepancy between actual and predicted death rates, specifically by ratio, we will assess whether or not there are patterns in those errors. The ratio of actual to predicted death rates is often referred to as "Actual-to-expected ratios" or "A/E ratios." This is not unlike some residual analyses for gaussian linear models, but because of the binary nature of the response, we will need to group observations into buckets. The buckets will be determined by each variable's distribution. Here's an example of such analysis across attained age.

First, we append the predictions to the whole dataset.

```
data.large <- data.frame(data.large,
                         Preds.large = predict(model.final,
                                               newdata = data.large,
                                               type = "response"),
                         Preds.middle = predict(model.mid,
```

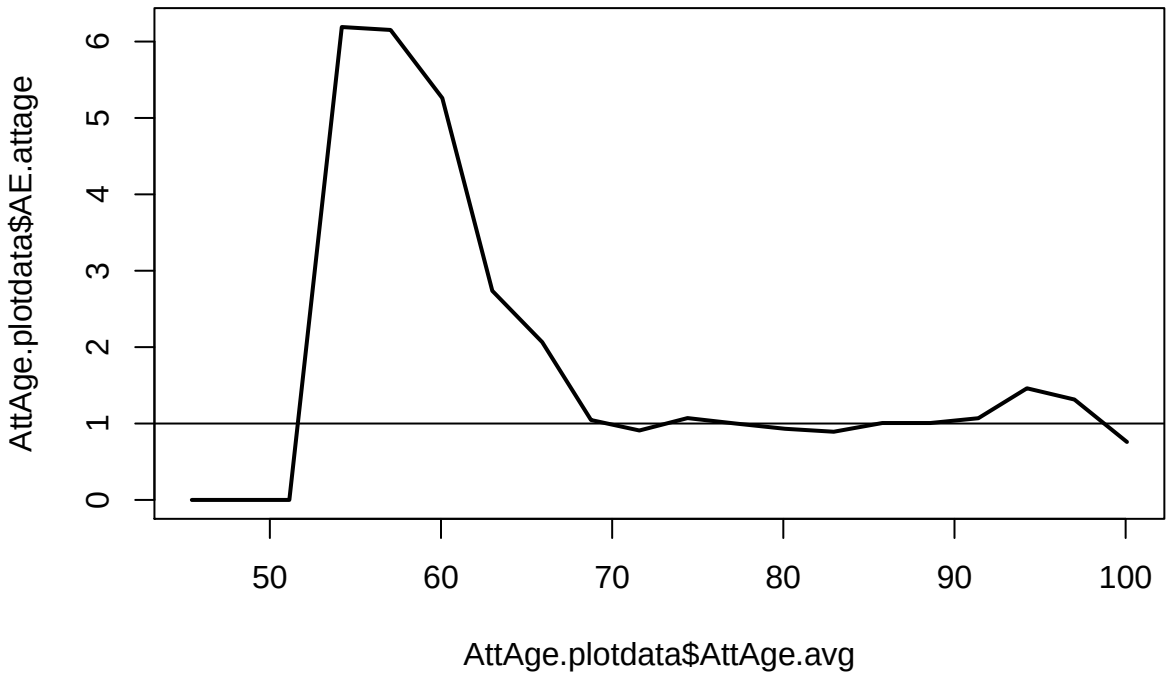
```
                                                    newdata = data.large,
                                                    type = "response"))
```

Then, we break up observations into buckets by attained age value using the cut() function. We group observations into their buckets and calculated A/E ratios. We've also included the number of observations in each bucket with the n() function so that we don't get caught making model changes based on small subsets of our training data.

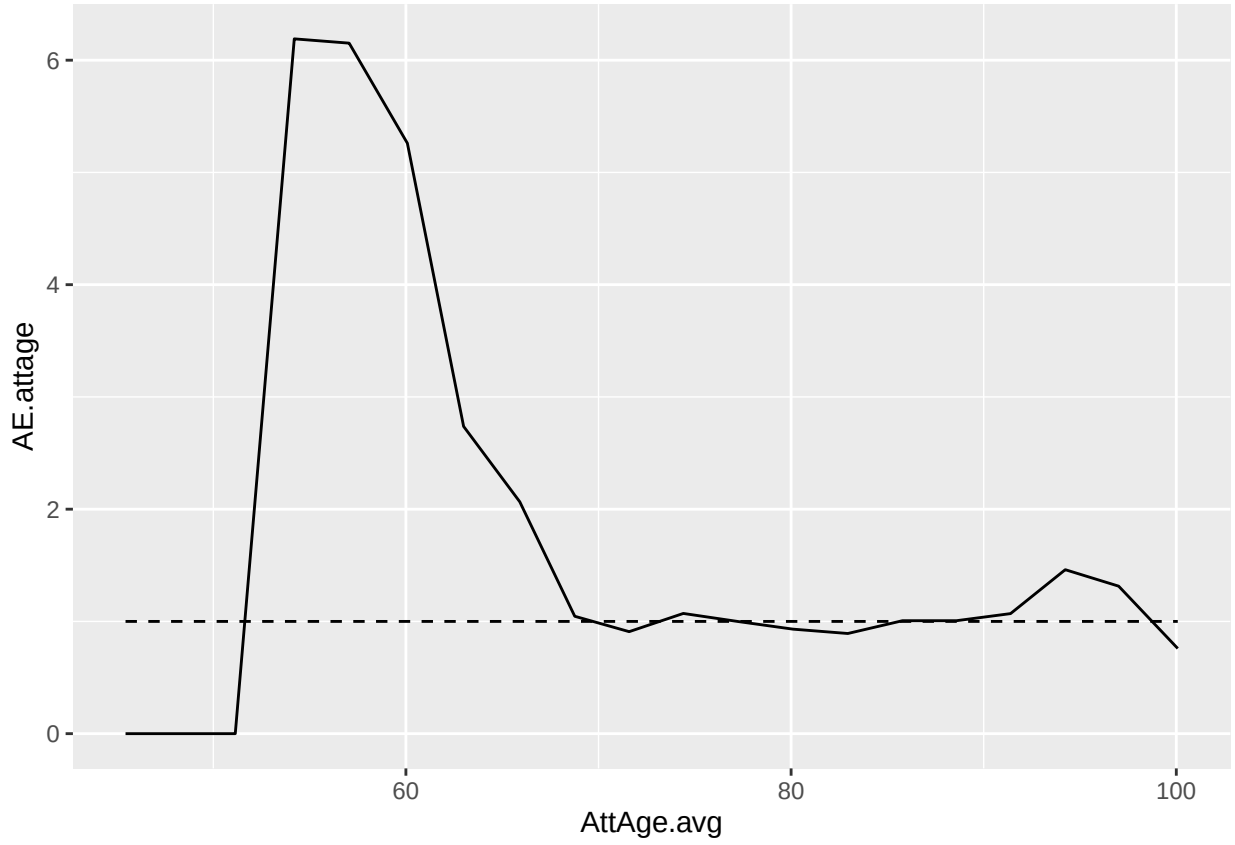
```
(AttAge.plotdata <- data.large %>%
   filter(Sample == "training") %>%
   group_by(AttAge.cut = cut(AttAge, breaks = 20)) %>%
   summarize(AttAge.avg = mean(AttAge),
             Deaths = mean(Death),
             PredDeaths = mean(Preds.large),
             AE.attage = mean(Death)/mean(Preds.large),
             N = n()))
```

Now let's plot the A/E ratio over attained age. We produce a line graph by inputting type = "l". There are many plot options in R, so this is by no means the only way to achieve such a plot. (Might we suggest the loess() function?)

```
plot(x = AttAge.plotdata$AttAge.avg,
     y = AttAge.plotdata$AE.attage,
     lwd = 2,
     type = "l")
abline(h = 1)
```



```
#ggplot alternative
AttAge.plotdata %>%
  ggplot(aes(x = AttAge.avg, y = AE.attage)) +
  geom_line() +
  geom_line(aes(y = 1), linetype = 2)
```



That plot strongly suggests non-linearity. Combining information from the plot with its AttAge.plotdata chart, we see that the high A/E values from age 54.2 to 65.9 are based on a combined sample size of more than 5,000 observations. There are algorithmic methods for finding where exactly the inflection points might be, but for now let's just split the attained age variable into two pieces at 68.8.

To do so, we'll create a variable that is equal to zero for all attained ages below 68.8, and the difference between attained age and 68.8 for those observations that are older. In addition to the orginal AttAge variable, this provides us a simple way to let the model fit two linear pieces over one variable. The distinct linear terms over one variable are often called "piecewise terms."

```
data.large <- mutate(data.large,
                     AttAge2 = pmax(0, AttAge - 68.8))
```

Now let's see how our model likes that "new" variable. (Additionally, we'll remove BMI from the model to avoid the multicollinearity issue discussed above.)

```
form.agepiece <- as.formula(Death ~ AttAge + AttAge2 + cad + cognitive + diabetes +
                              alcohol + pulmonary + adls + smoker +
                              parkinsons + tia + gender + activitylevel +
                              positivefamilylongevity + depression +
                              cancer_ind + ht.wt.flag + height + bmisqrerr)
model.agepiece <- glm(form.agepiece,
```

```
                        family = binomial,
                        data = filter(data.large, Sample == "training"))
summary(model.agepiece)
```

Both pieces of the attained age variable are statistically significant, so we'll proceed with this model. Note that it is important to be conservative with the addition of piecewise terms. Look for trends that continue over multiple buckets, and be sure the split makes sense. In this case, it doesn't seem like a negative initial slope coefficient on attained age makes sense, but it gives us something to check on our holdout dataset later.

Let's make sure to add predictions from that model to the dataset:

```
data.large <- data.frame(data.large,
                         Preds.agepiece = predict(model.agepiece,
                                                  newdata = data.large,
                                                  type = "response"))
```

3. Undersampling

Often we work with many more observations and many more variables than there are in this dataset. If there is a way to move through the modeling stages more quickly, then we'll do it. To that end, undersampling is extremely helpful. Undersampling involves removing a random sample of the majority outcome. For our dataset, we would remove many yearly observations of non-deaths. We have 4144 deaths in our training dataset, so randomly selecting 4144 non-deaths would be a symmetric way to create a new dataset. However, that symmetry is not statistically necessary due to the fact that the coefficients of a logistic GLM are odds ratios in disguise, and odds ratios remain statistically unchanged after undersampling. That is, they only vary randomly, not systematically.

Here, we will simply show some code for creating this undersampled training dataset. All of the same analysis we have done to this point could be done on that training set. Obviously, mortality rates will appear higher, increasing the value of the intercept coefficient. But all the model's other coefficients will yield the same values to within a margin of error. Once you've selected your best model on an undersampled dataset, all you have to do is fit that model once on the full dataset, saving a lot of time over the whole process.

```
ratio <- 3 # Thrice as many non-deaths as deaths
data.deaths <- data.large %>%
  filter(Sample == "training",
         Death == 1)
set.seed(1)
data.sample <- sample_n(data.large %>%
                          filter(Sample == "training",
                                 Death == 0),
                        size = ratio*nrow(data.deaths))
data.small <- rbind(data.deaths, data.sample)
```

4. Validation and model comparison

This is where our holdout observations are required. At no point in the modeling process did we ever "check our answers" on the holdout observations, referred to as "holdout" and "testing," so we can use them now to show others how well our model could do when applied to new data. We'll stick to the in-time holdout portion since our data did not allow our model to be particularly dynamic, and that does not bode well for long-term future predictions.

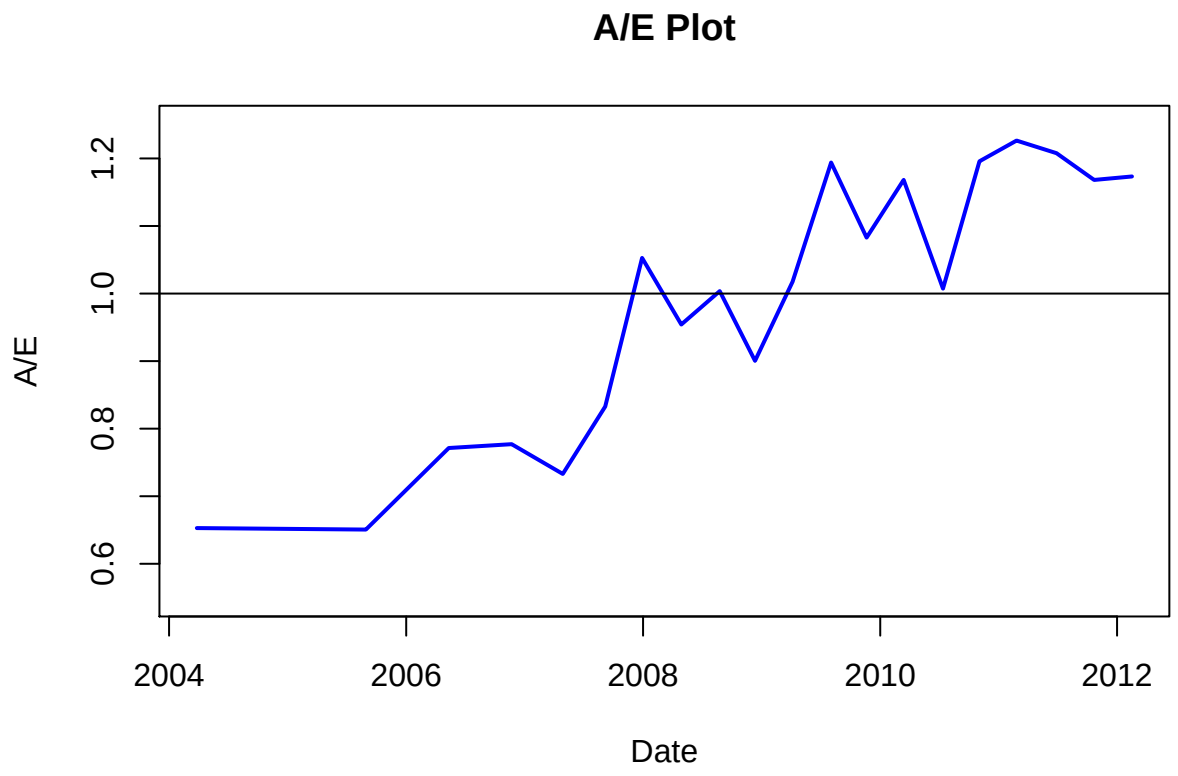
A. Overall model fit

There are many ways to test a logistic model's fit, including A/E ratio plots, confusion matrices, and calculation of the Area under the ROC curve (AUC).

i) Actual to expected plots Before testing our model on our holdout datasets, we want to make sure this is the best model we could fit to the training dataset. Typically, we create A/E plots across each of the variables in our model, and also across time-related variables like policy year or valuation date. As an example, here we'll make an A/E ratio plot across time. Our training dataset stretches from April of 1999 to April of 2012 with about 86,500 observations. That gives us enough data to break the observations up into 20 buckets across the current date variable, with an average of about 107 deaths and 4,323 total observations per bucket.

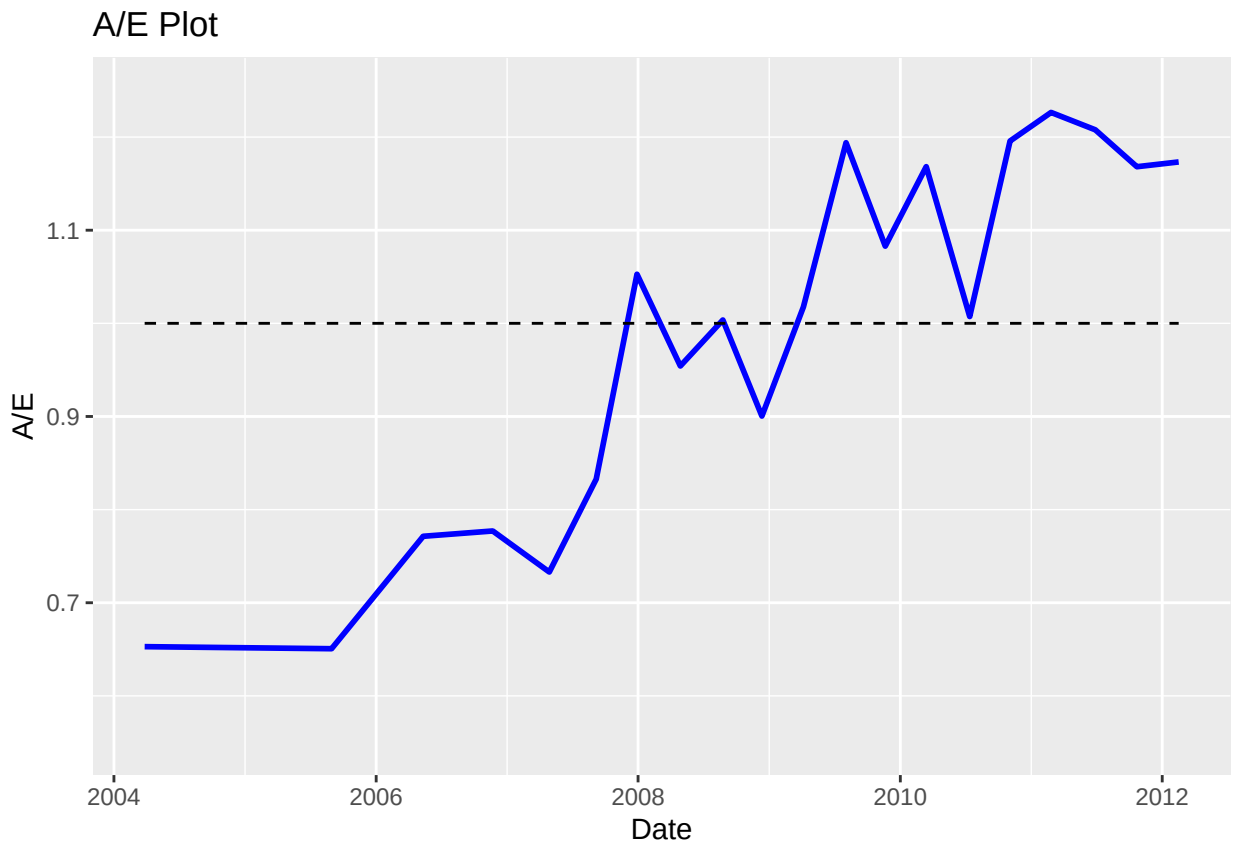
```
date.plotdata <- filter(data.large, Sample == "training") %>%
  group_by(Date.bin = ntile(current.date, 20)) %>%
  summarize(Date = mean(current.date),
            AE.date = sum(Death)/sum(Preds.agepiece))

plot(date.plotdata$Date, date.plotdata$AE.date,
     pch = ".", cex = 0,
     main = "A/E Plot",
     xlab = "Date",
     ylab = "A/E",
     ylim = c(0.55, 1.25),
     type = "l",
     lwd = 2,
     col = "blue")
abline(h = 1)
```



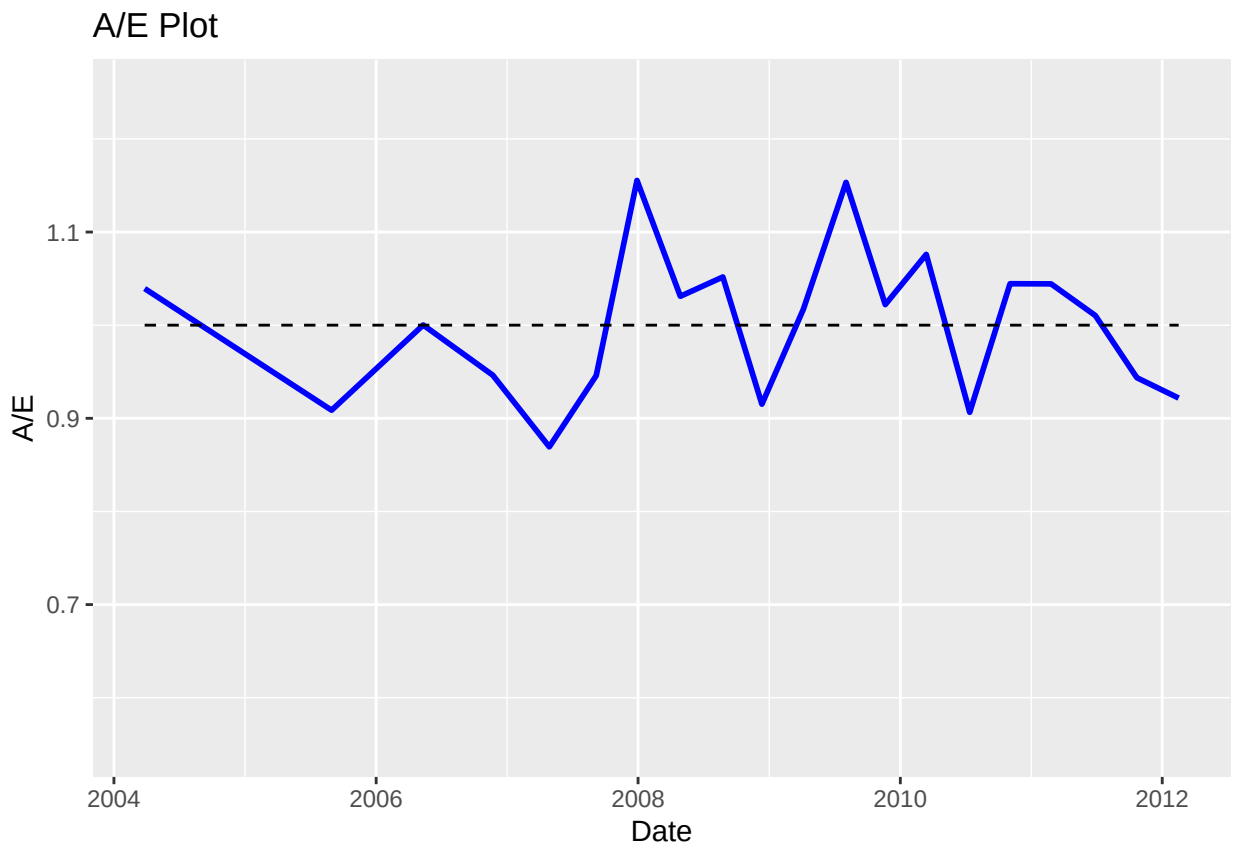

```
#ggplot alternative
date.plotdata %>%
  ggplot(aes(x = Date, y = AE.date)) +
```

```
  geom_line(size = 1, color = "blue") +
  labs(title = "A/E Plot", x = "Date", y = "A/E")+
  ylim(c(0.55, 1.25)) +
  geom_line(aes(y = 1), linetype = 2)
```



Ouch, we must have missed something! The plot suggests that we overpredict deaths earlier in the exposure period and underpredict later. Because we had few dynamic variables at our disposal, it made it hard for the model to keep up with changes to the subjects. To correct, let's quickly add a time predictor variable. However, it's important to note that time trends can be due to many root causes. As an actuary/data scientist, it's important to consider why the time trend exists before blindly extrapolating into the future. In this case, it could be due to some sort of duration "select-and-ultimate" effect. If these policies were underwritten, then the selection effect could be wearing off as time goes on.


A/E Plot
A/E
0.6
0.8
1.0
1.2
2004
2006
2008
2010
2012
Date



There! Errors now look more random across the time dimension. Of course, we still want to identify the true source of the time trend, but the approach can be applied to a wide range of variables, whether they are in the current model or not. This process can be applied in an identical manner to the holdout dataset, so long as you have appended the predictions from the training model to the entire dataset (including holdouts). The following validation techniques are all performed on the holdout dataset.

ii) Confusion matrices A confusion matrix, also known as a 2x2 contingency table, cross tabulates observations by predicted outcomes and actual outcomes. If you're not familiar with these tables, here's an example. We use the average prediction *probability* as a threshold for predicting 1s and 0s.

```
data.large <- data.large %>%
  mutate(Pred.bin = ifelse(Preds.time > mean(Preds.time), 1, 0))

(conf.mat <- xtabs(~ Pred.bin + Death,
                   data = filter(data.large, Sample == "holdout")))
```

```
          Death
Pred.bin      0     1
       0  65815   835
       1  18500  1313
```

```
(sensitivity <- conf.mat[2,2]/sum(conf.mat[,2]))
```

```
[1] 0.6112663
```

```
(specificity <- conf.mat[1,1]/sum(conf.mat[,1]))
```

```
[1] 0.7805847
```

iii) Area under the (ROC) curve Here's how to check AUC. The technical definition of AUC is more complicated than what it implies. AUC is basically this: given you select two holdout observations at random, one that died and one that did not, what are chances that the model scored the one that died with a higher probability of doing so? A 50% AUC would basically be a model that randomly assigned death probabilities, so 50% becomes our baseline.

First we'll load the ROCR package, and create "prediction" objects:

```
library(ROCR)
preds.mid <- prediction((data.large %>%
                           filter(Sample == "holdout"))$Preds.mid,
                        (data.large %>%
                           filter(Sample == "holdout"))$Death)
preds.time <- prediction((data.large %>%
                            filter(Sample == "holdout"))$Preds.time,
                         (data.large %>%
                            filter(Sample == "holdout"))$Death)
```

Now let's compare the AUC percentages. The following output shows that the larger model outperformed the smaller one by AUC, 76.7% to 72%.

```
(auc.mid <- performance(preds.mid, "auc")@y.values[[1]])
```

```
## [1] 0.7203179
```

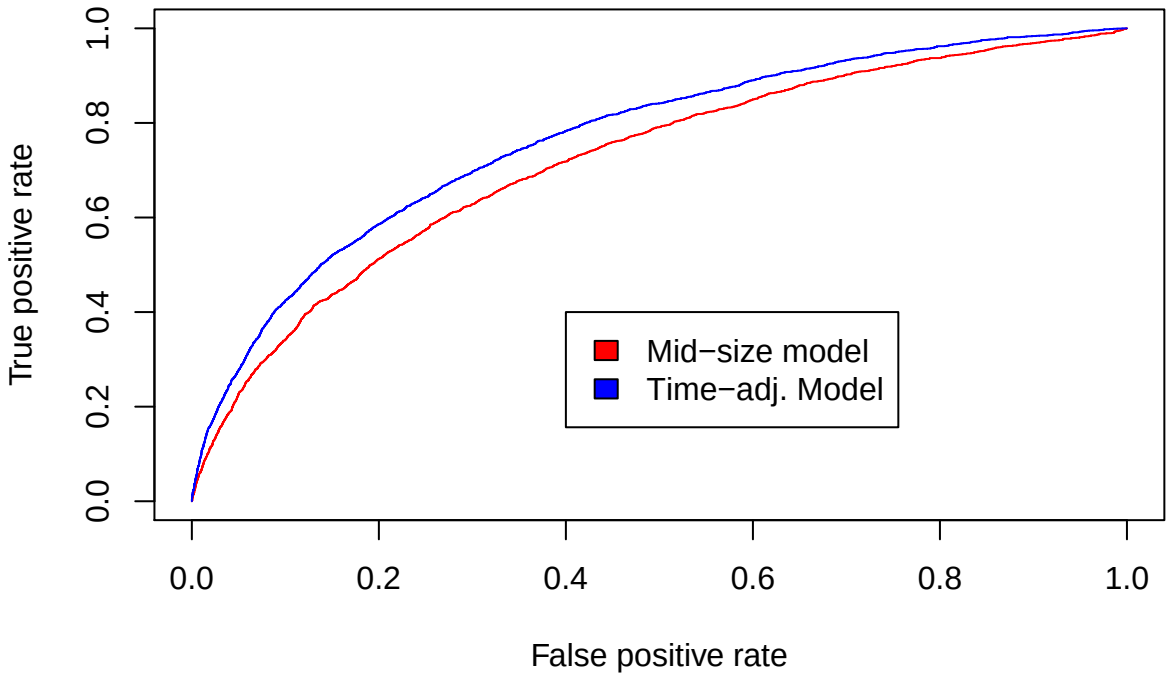
```
(auc.time <- performance(preds.time, "auc")@y.values[[1]])
```

```
## [1] 0.7667362
```

For a more visual approach, here's how to plot the ROC curves:

```
auc.plot.mid <- performance(preds.mid, "tpr", "fpr")
auc.plot.time <- performance(preds.time, "tpr", "fpr")

plot(auc.plot.mid,
     col = "red")
plot(auc.plot.time,
     add = T, col = "blue")
legend(0.4, 0.4,
       legend = c("Mid-size model", "Time-adj. Model"),
       fill = c("red", "blue"))
```

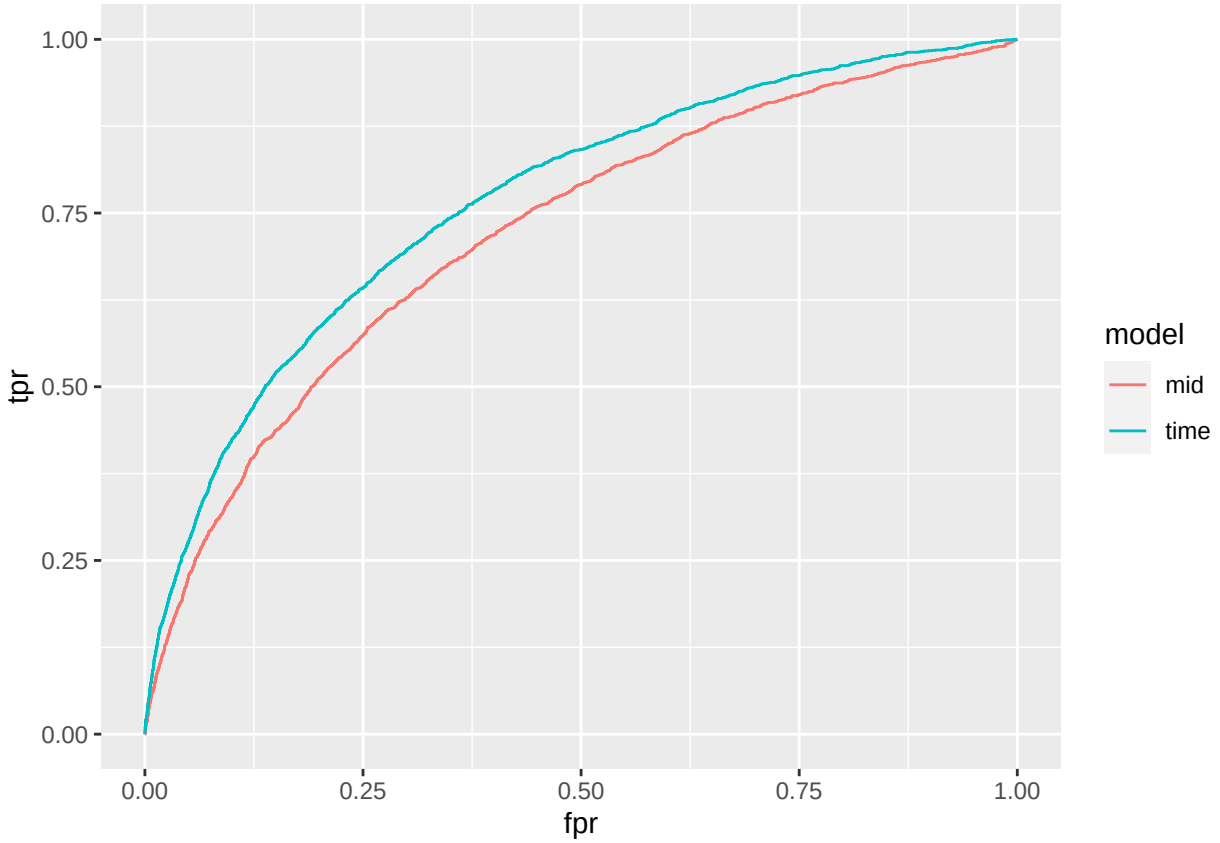



```
#ggplot alternative
data.frame(fpr = auc.plot.mid@x.values[[1]],
           tpr = auc.plot.mid@y.values[[1]],
           model = "mid") %>%
  bind_rows(data.frame(fpr = auc.plot.time@x.values[[1]],
                       tpr = auc.plot.time@y.values[[1]],
                       model = "time")) %>%
  ggplot(aes(x = fpr, y = tpr, color = model)) +
  geom_line()
```

```
Warning in bind_rows_(x, .id): Unequal factor levels: coercing to character

Warning in bind_rows_(x, .id): binding character and factor vector, coercing
into character vector

Warning in bind_rows_(x, .id): binding character and factor vector, coercing
into character vector
```



B. Comparison between two candidate models

We'll compare our "middle model" and our "full model" with the time variable as an example. Two-way lift charts are a bit complicated at first, but they do a very good job of comparing models visually.

At its core, a two-way lift chart is an A/E ratio plot, but the trick is in how we arrange the buckets across the x-axis. We'll order the *ratios* of predictions between the two models from least to greatest. So the extreme right and left sides of the graph will consist of buckets representing where the two models disagreed the most, while the middle of the x-axis will consist of buckets where the two models tended to agree. The model with an A/E ratio line closest to one across this entire range of model agreement would be considered the best model.

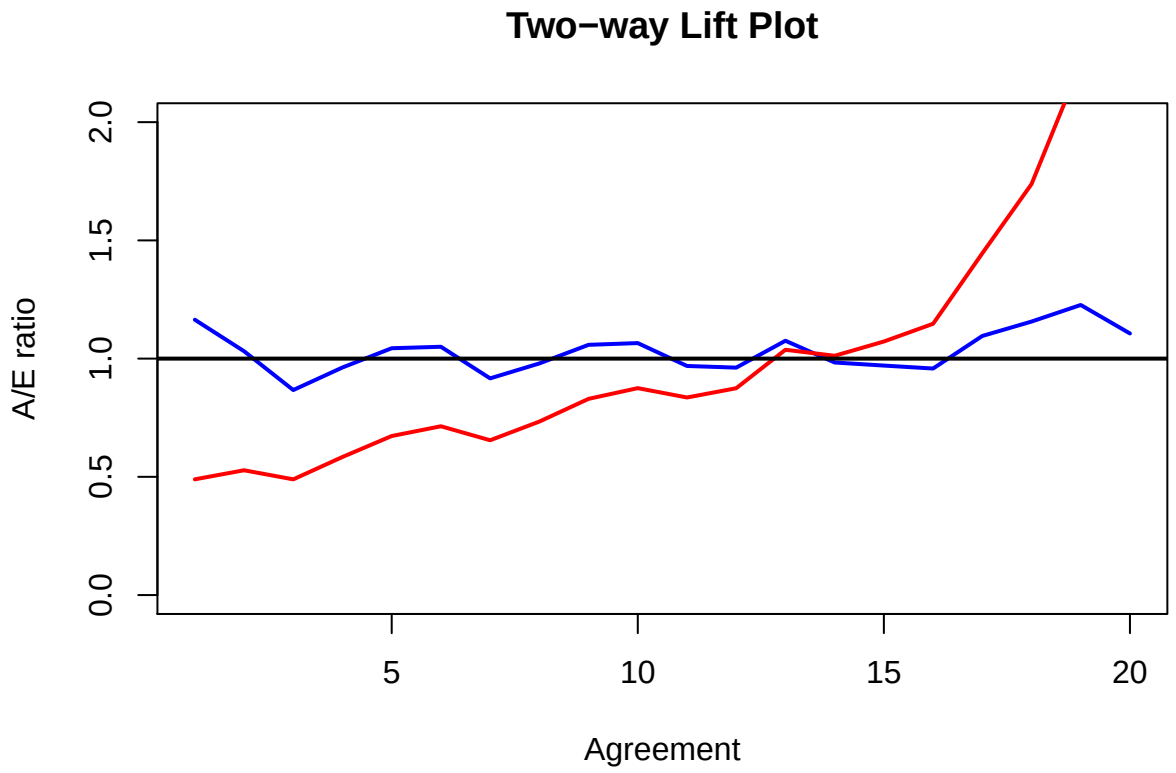
First, the plot data:

```
data.twoway <- data.large %>%
  filter(Sample == "holdout") %>%
  group_by(Agreement = ntile(Preds.time/Preds.middle, 20)) %>%
  summarize(Pred.ratio = sum(Preds.time)/sum(Preds.middle),
            AE.large = sum(Death)/sum(Preds.time),
            AE.middle = sum(Death)/sum(Preds.middle))
```

Now let's plot it:

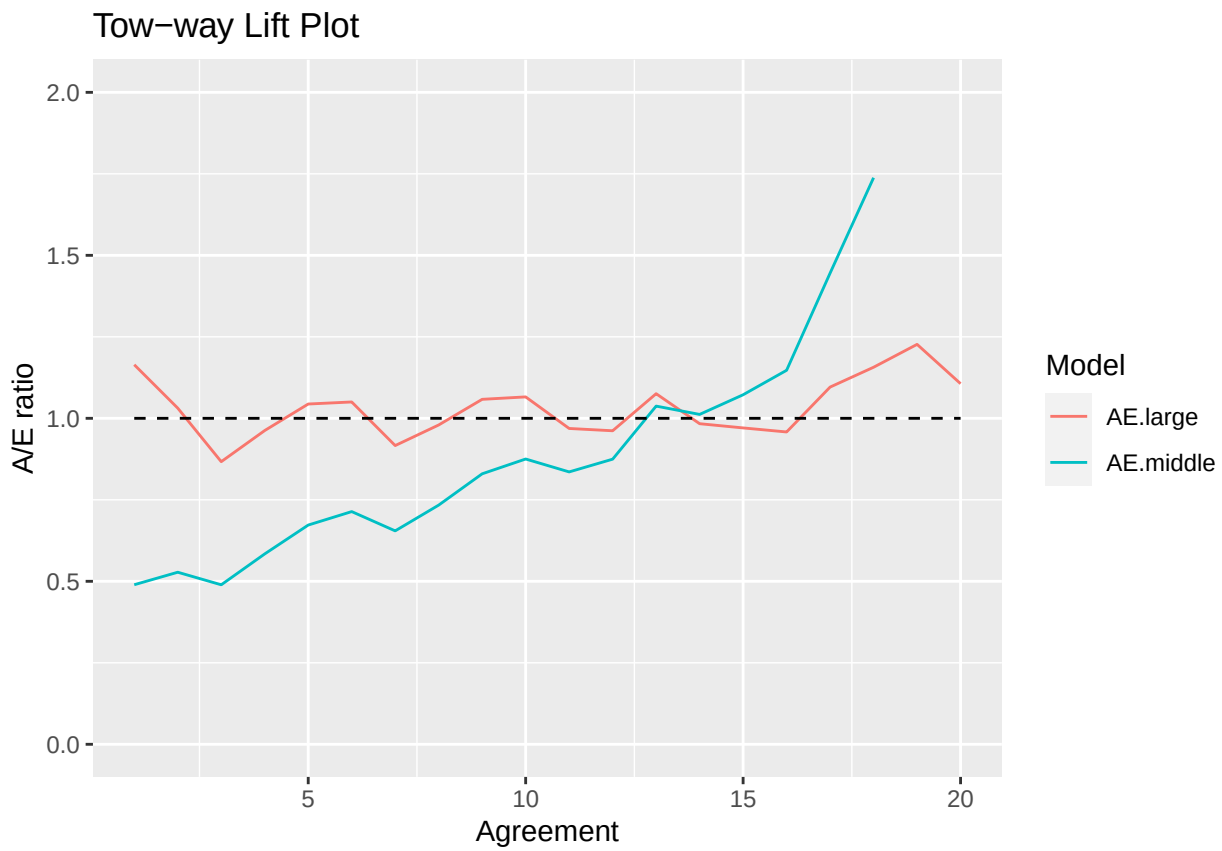
```
plot(data.twoway$Agreement,
     data.twoway$AE.large,
     ylim = c(0,2),
     xlab = "Agreement",
     ylab = "A/E ratio",
```

```
     main = "Two-way Lift Plot",
     type = "l",
     lwd = 2,
     col = "blue")
lines(data.twoway$Agreement,
      data.twoway$AE.middle,
      lwd = 2,
      col = "red")
abline(h = 1, lwd = 2)
```




```
#ggplot alternative
data.twoway %>%
  gather("Model", "A/E ratio", AE.large, AE.middle) %>% # gather() is useful function from tidyr packag
  ggplot(aes(x = Agreement, y = `A/E ratio`, color = Model)) +
  geom_line() +
  labs(title = "Tow-way Lift Plot", x = "Agreement", y = "A/E ratio") +
  ylim(c(0,2)) +
  geom_line(aes(y = 1), linetype = 2, color = "black")
```

```
## Warning: Removed 2 row(s) containing missing values (geom_path).
```



The plot shows that the fuller "large" model did better across nearly the entire range of Agreement (blue line closer to 1). Recall that the two-way plot, in addition to the last few validation techniques, was constructed on the holdout dataset, whose data did nothing to influence our model building.