SOCIETY OF ACTUARIES

The Predictive Analytics & Futurism Section Presents

Practical Predictive Analytics Seminar

May 10, 2017 | Sheraton Seattle Hotel | Seattle, WA

Presenters:
Eileen Sheila Burns, FSA, MAAA
Talex Diede, MS
Jean-Marc Fix, FSA, MAAA
Brian D. Holland, FSA, MAAA
Matthias Kullowatz, MS
Ricardo Trachtman, FSA, MAAA

Practical Predictive Analytics Seminar

Jean-Marc Fix, FSA, MAAA

Intro to R

10 May 2017

SOCIETY OF ACTUARIES

SOCIETY OF ACTUARIES
Antitrust Notice for Meetings

Active participation in the Society of Actuaries is an important aspect of membership. However, any Society activity that arguably could be perceived as a restraint of trade exposes the SOA and its members to antitrust risk. Accordingly, meeting participants should refrain from any discussion which may provide the basis for an inference that they agreed to take any action relating to prices, services, production, allocation of markets or any other matter having a market effect. These discussions should be avoided both at official SOA meetings and informal gatherings and activities. In addition, meeting participants should be sensitive to other matters that may raise particular antitrust concern: membership restrictions, codes of ethics or other forms of self-regulation, product standardization or certification. The following are guidelines that should be followed at all SOA meetings, informal gatherings and activities:

- DON'T discuss your own, your firm's, or others' prices or fees for service, or anything that might affect prices or fees, such as costs, discounts, terms of sale, or profit margins.
- DON'T stay at a meeting where any such price talk occurs.
- DON'T make public announcements or statements about your own or your firm's prices or fees, or those of competitors, at any SOA meeting or activity.
- DON'T talk about what other entities or their members or employees plan to do in particular geographic or product markets or with particular customers.
- DON'T speak or act on behalf of the SOA or any of its committees unless specifically authorized to do so.
- DO alert SOA staff or legal counsel about any concerns regarding proposed statements to be made by the association on behalf of a committee or section.
- DO consult with your own legal counsel or the SOA before raising any matter or making any statement that you think may involve competitively sensitive information.
- DO be alert to improper activities, and don't participate if you think something is improper.

- If you have specific questions, seek guidance from your own legal counsel or from the SOA's Executive Director or legal counsel.

Presentation Disclaimer

Presentations are intended for educational purposes only and do not replace independent professional judgment. Statements of fact and opinions expressed are those of the participants individually and, unless expressly stated to the contrary, are not the opinion or position of the Society of Actuaries, its cosponsors or its committees. The Society of Actuaries does not endorse or approve, and assumes no responsibility for, the content, accuracy or completeness of the information presented. Attendees should note that the sessions are audio-recorded and may be published in various media, including print, audio and video formats without further notice.

A is for Actuary

B is for Big

C is for Complex

D is for Data

What R you afraid of?

Basic R: A Programming Language!

R Studio: be a star on your own computer

A script without a movie

Ggraphhing withh ggplot2

Playing with dplyr

The black box

Pygmalion

Pygmalion
by Etienne Falconet

Be a learn-R

Practical Predictive Analytics Seminar

Eileen S. Burns
Matthias Kullowatz

Session 2: Predictive Models in Life and Annuities
May 10, 2017

SOCIETY OF ACTUARIES

Theory

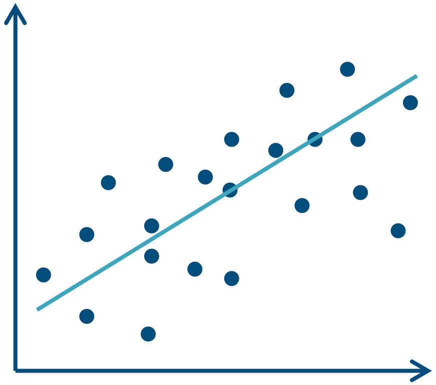
Theory

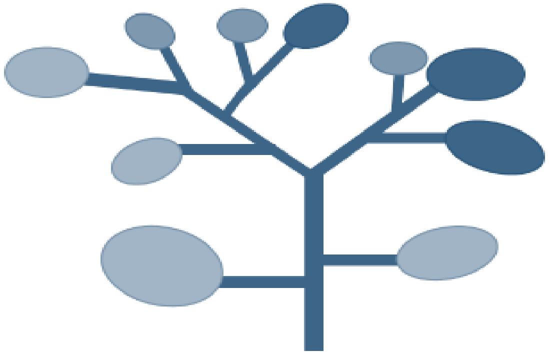
- Questions of interest for life and annuity products
- Predictive model forms that are best suited to investigating them
- Associated theoretical concerns that may arise in the modeling process.

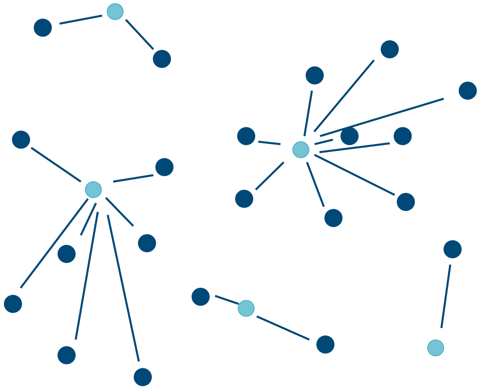
Questions of interest

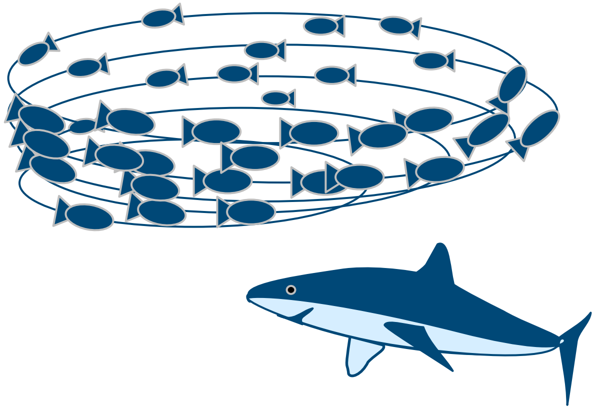
- When will a policyholder…
  - Lapse?
  - Partially withdraw?
  - Die?
- How will a policyholder utilize the policy?
- What drives these “behaviors” and why?

Predictive model forms



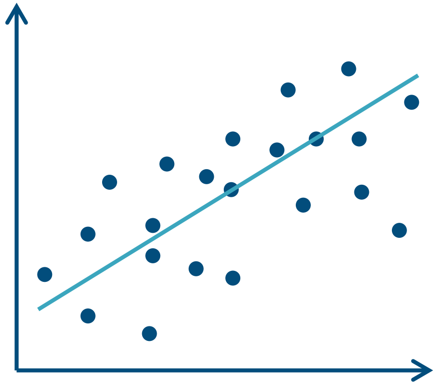






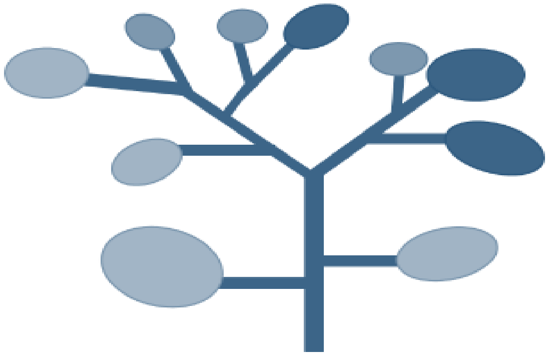
Regression

- OLS, GLM, ridge, lasso, elastic net
- Pros
  - Quick fitters
  - Interpretable coefficients and output
  - Harder to overfit
- Cons
  - Constrained by functional form
  - Multicollinearity issues



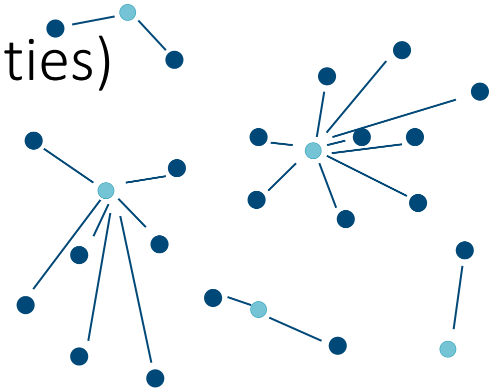
Tree-based models

- Decision trees, bagging, boosting
- Pros
  - Non-parametric
  - Intuitive output
- Cons
  - Black-box formula
  - Constrained to rectangular regions



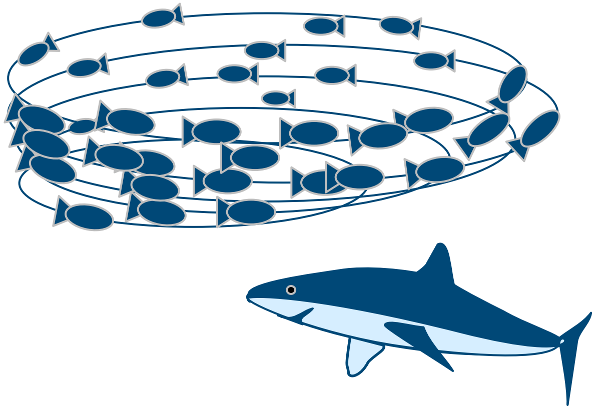
Clustering

- K-means, hierarchical, k-nearest neighbors
- Pros
  - Non-rectangular regions
  - Can be unsupervised
- Cons
  - Sensitive to outliers
  - Lacks proximity to other clusters (probabilities)



Agent-based modeling

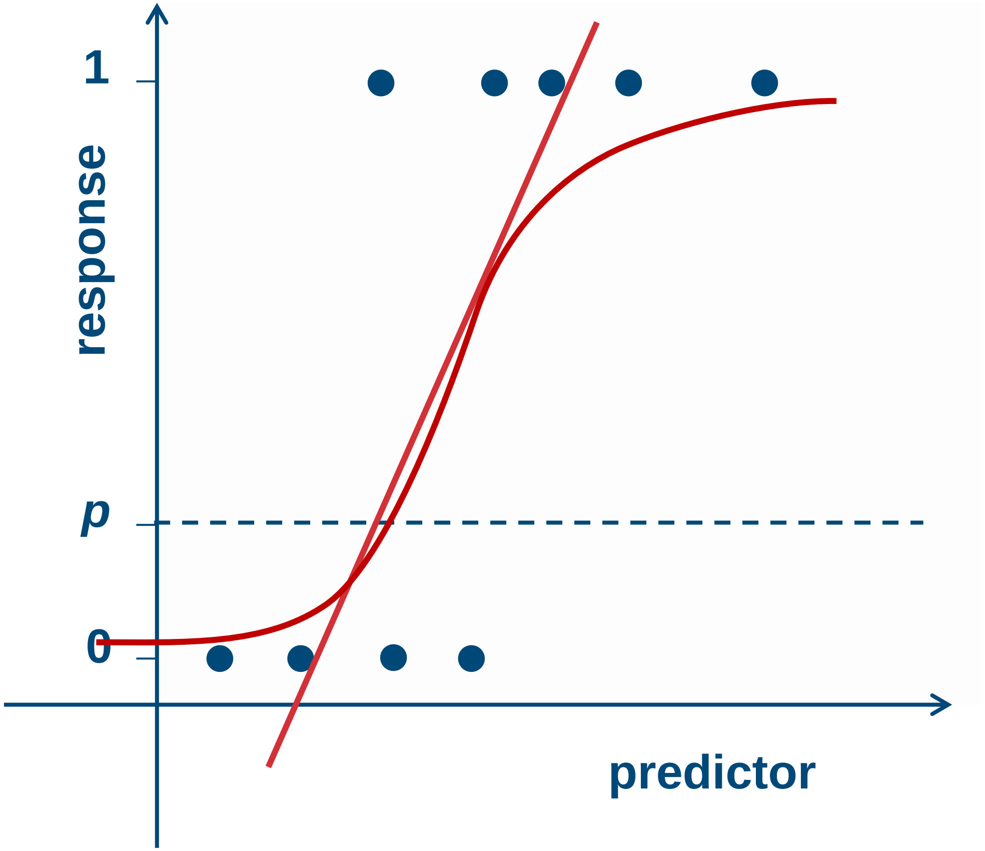
- Pros
  - Causation is implied in the model
  - Outputs easily interpretable
- Cons
  - At high risk of modeler bias
  - Difficult to validate



Logistic GLM

- For predicting probabilities of binary outcomes
- Link function provides much needed flexibility
- Predictor variables can be quantitative or qualitative

Why a link function?

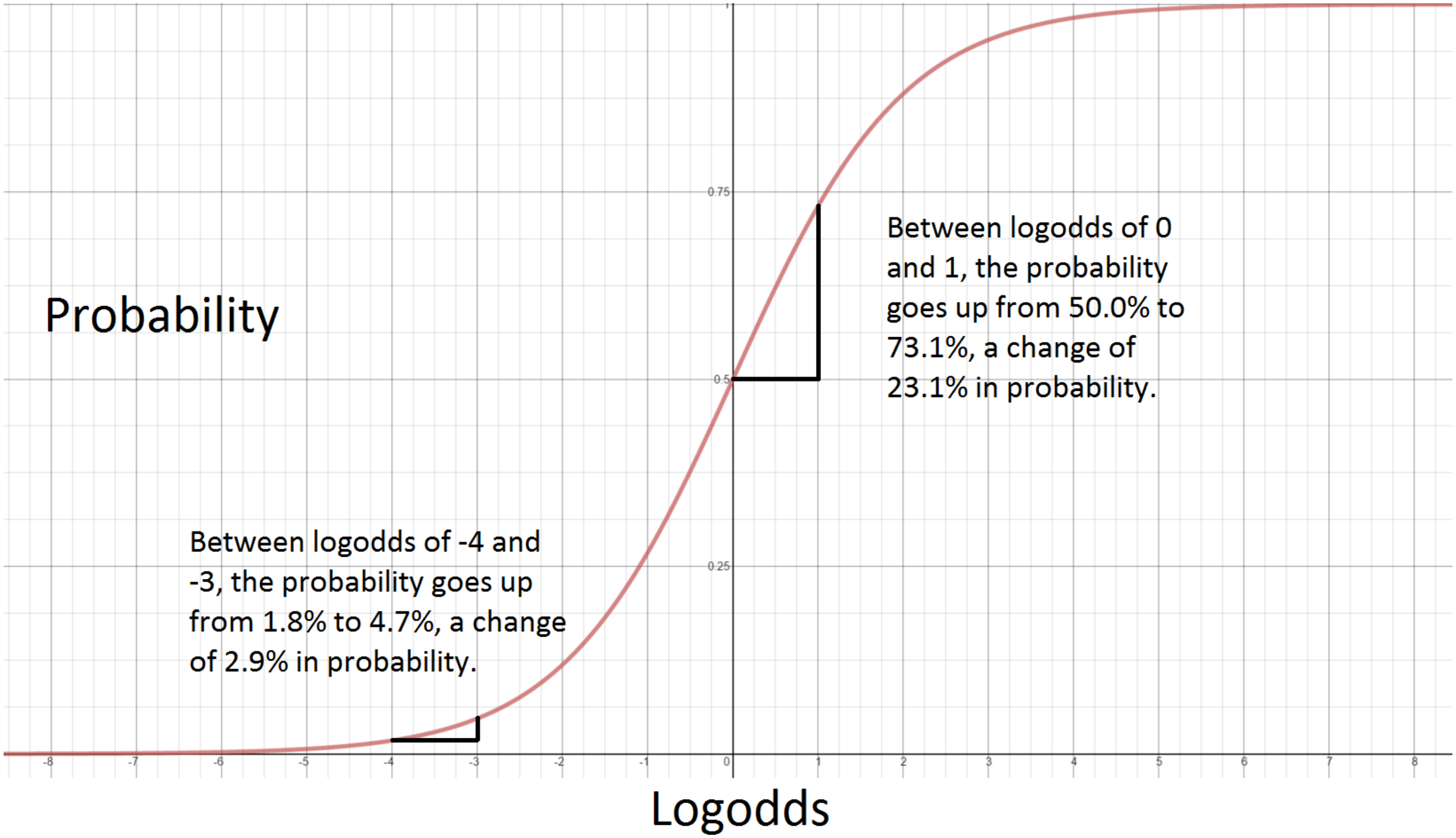


The logistic function

- $\hat{y} = g(L) = \frac{e^L}{1+e^L}$
  - $L = \widehat{\beta_0} + \widehat{\beta_1} x_1 + \widehat{\beta_2} x_2 + \cdots + \widehat{\beta_p} x_p$
  - $\lim_{L \to \infty} g(L) = 1 \; and \lim_{L \to -\infty} g(L) = 0$

- $g^{-1}(\hat{y}) = \ln\left(\frac{\hat{y}}{1-\hat{y}}\right) = L$
  - Logit function (“logodds”)

Consequences of logit link



Between logodds of 0 and 1, the probability goes up from 50.0% to 73.1%, a change of 23.1% in probability.

Between logodds of -4 and -3, the probability goes up from 1.8% to 4.7%, a change of 2.9% in probability.

Generalized linear models

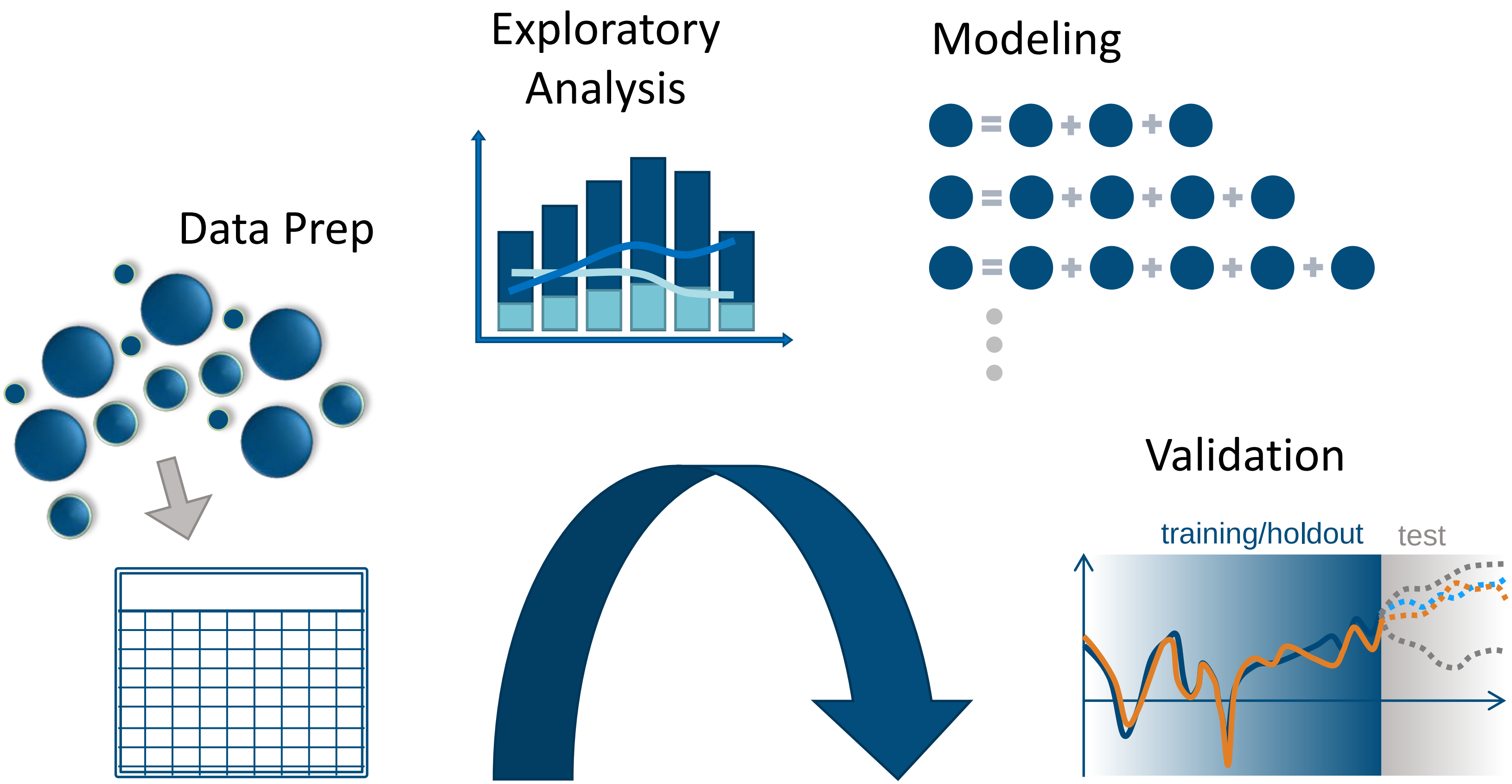
- Theoretical extras
  - Independent observations
  - The model is fit by maximizing the following:

$$loglikelihood = \sum [Y_i \ln(\hat{y}_i) + (1 - Y_i)\ln(1 - \hat{y}_i)]$$

  - $AIC = -2 \times loglikelihood + 2 \times parameters$
  - $BIC = -2 \times loglikelihood + \ln(N) \times parameters$

Practical concerns

Predictive analytics process



Exploratory Analysis

Modeling

Data Prep

Validation

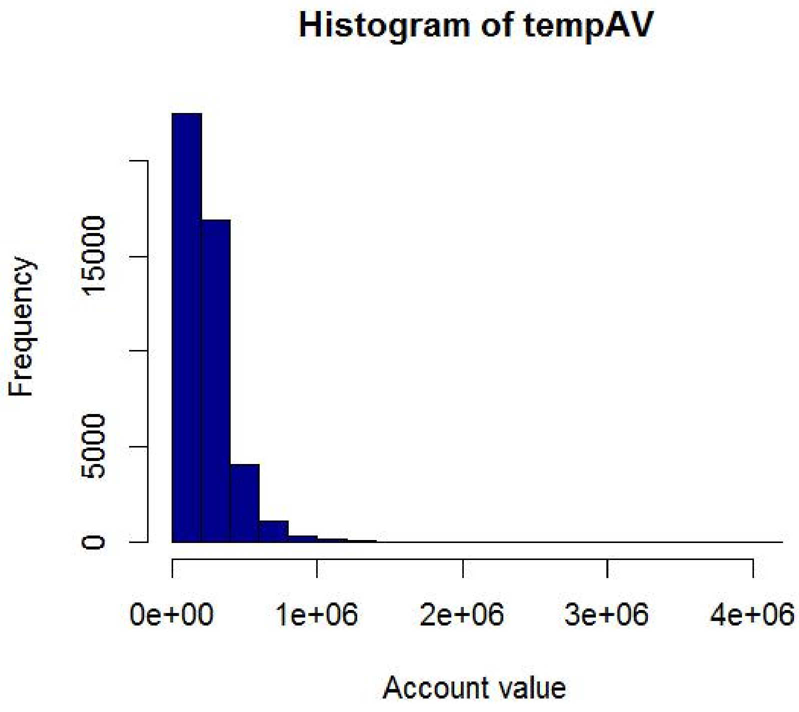
Practical concerns: Data

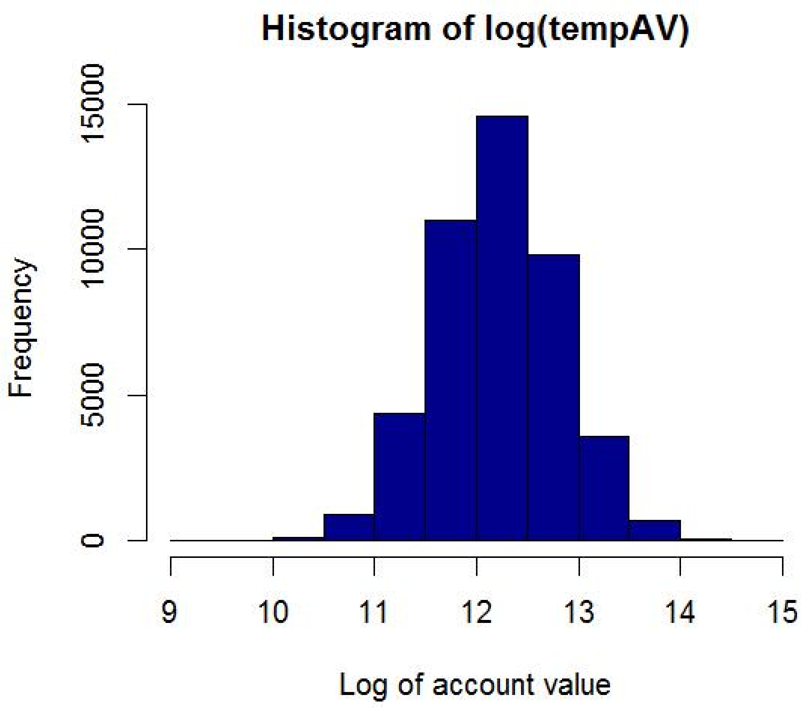
- Formatting variables (1)
- Identifying and dealing with outlier data values (2)
- Accounting for missing data (2)
- Derive new variables for modeling (3)
- Compile dataset into appropriate format (4)

Practical concerns: Modeling

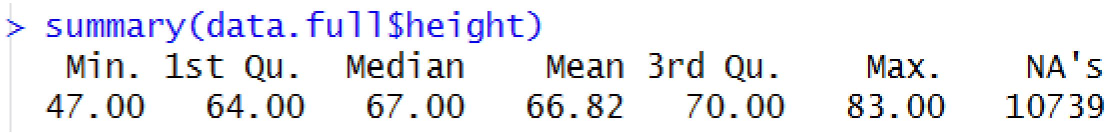
- Holdout dataset (2A)
- Fitting a model (2C)
- Using the step function for variable selection (2D)
- Multicollinearity concerns (2E)
- Setting reference levels for factors (DataPrep 2)
- Piecewise terms (2F)
- Undersampling (3)

Data outliers

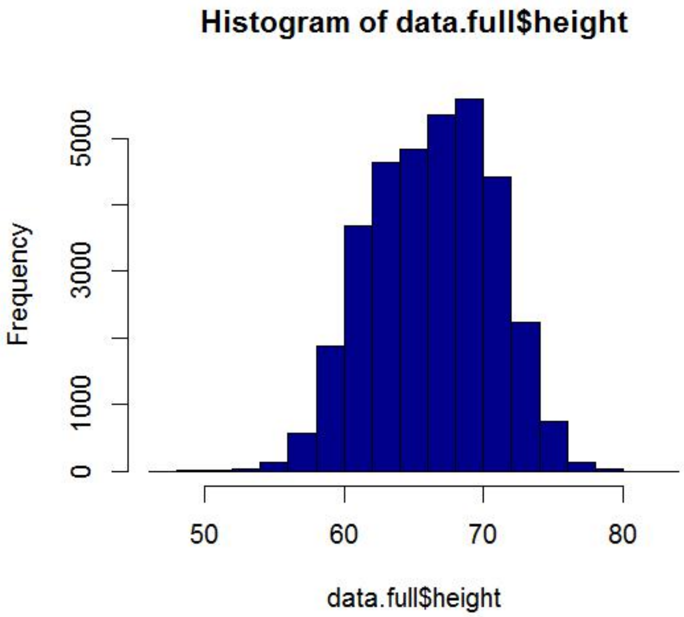


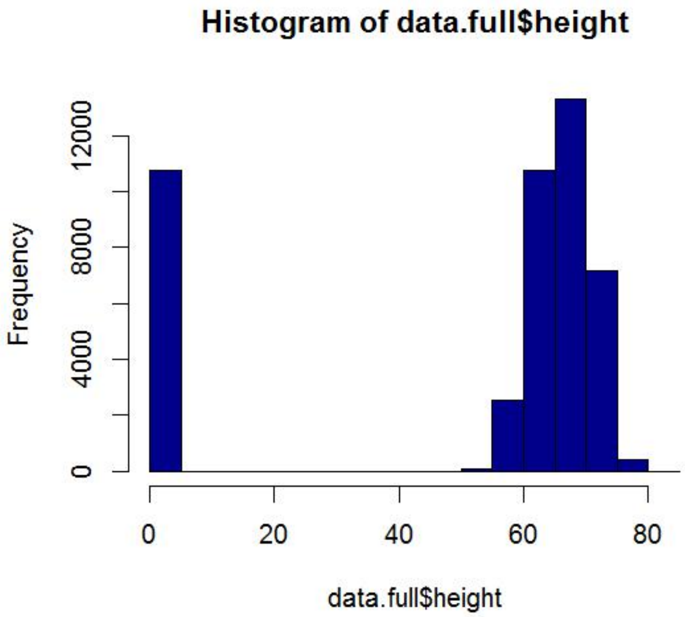


Missing values



```
> summary(data.full$height)
  Min. 1st Qu.  Median    Mean 3rd Qu.    Max.    NA's 
 47.00   64.00   67.00   66.82   70.00   83.00   10739 
```



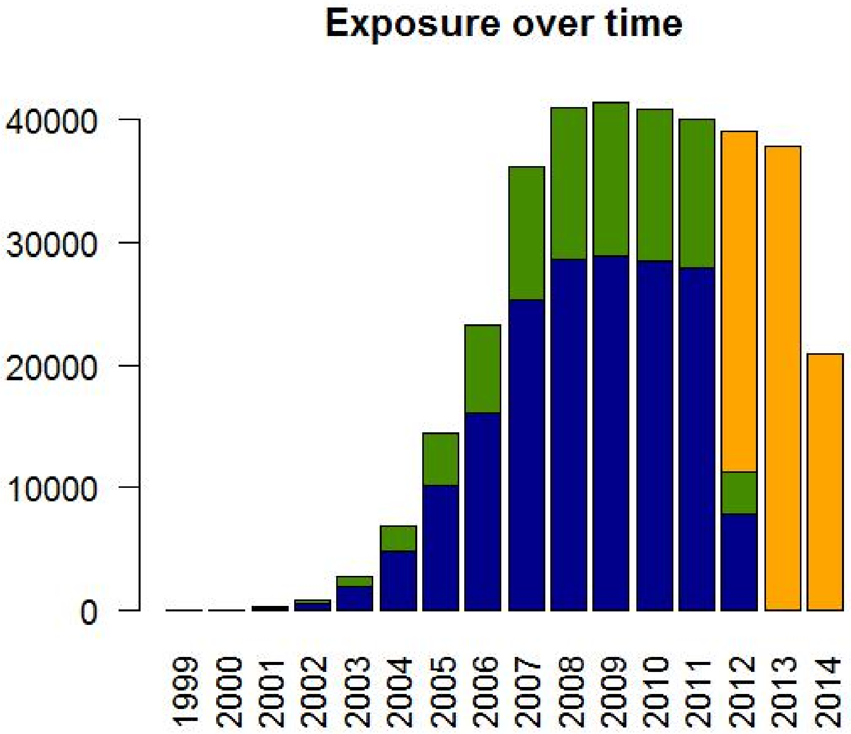


Missing values

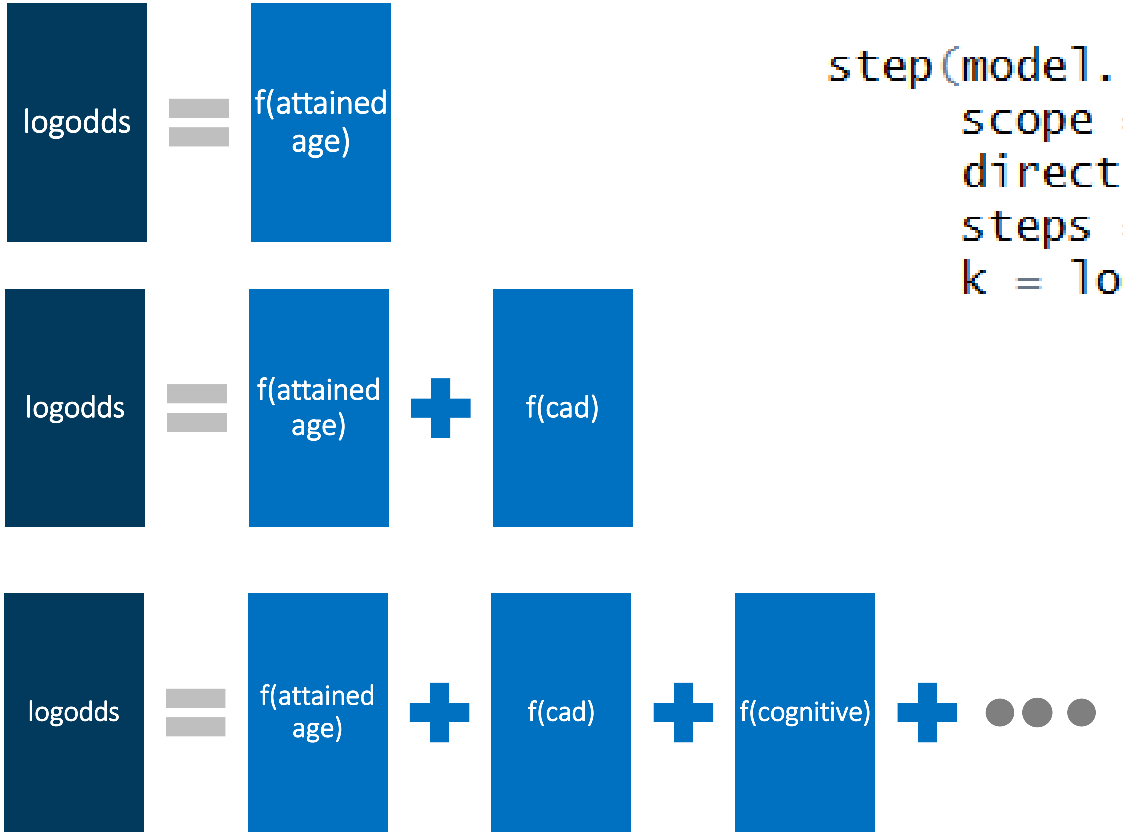
Model	NA treatment	Intercept	Height coefficient	Flag coefficient
Death ~ height	Removed	-4.418	0.0100	N/A
Death ~ height + Ind	Set to 0	-3.580	0.0100	-0.838
Death ~ height + Ind	Set to mean	-4.245	0.0100	-0.173
Death ~ height	Set to 0	-3.589	-0.0024	N/A
Death ~ height	Set to mean	-4.343	0.0095	N/A

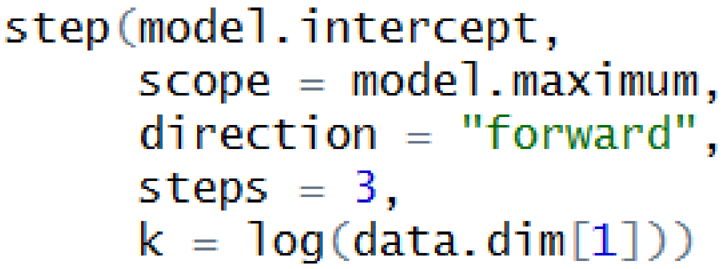
- The first three models are mathematically equivalent
- The second two are biased

Training versus holdout data



Stepwise model building

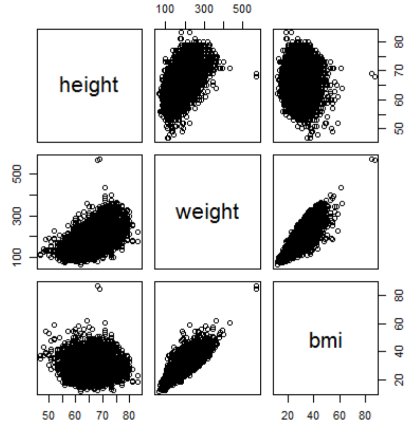




```
step(model.intercept,
     scope = model.maximum,
     direction = "forward",
     steps = 3,
     k = log(data.dim[1]))
```

Multicollinearity

- pairs()

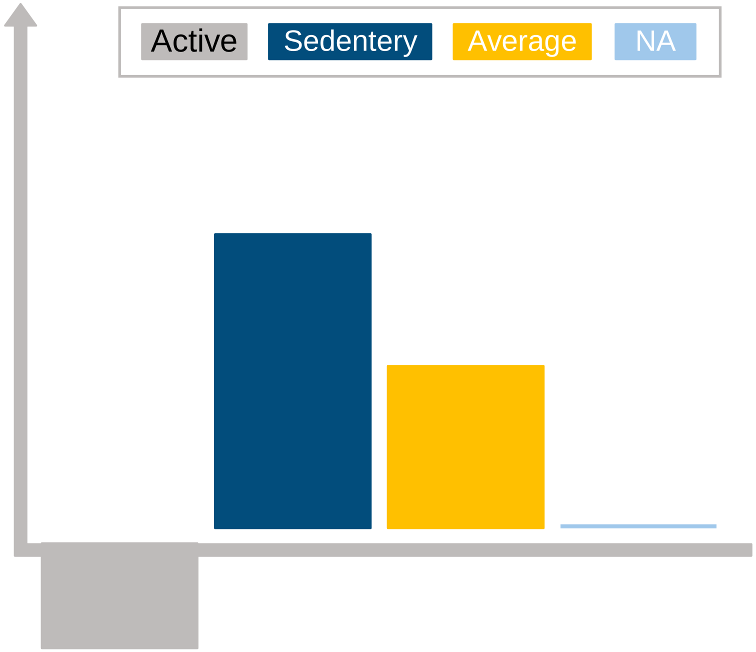


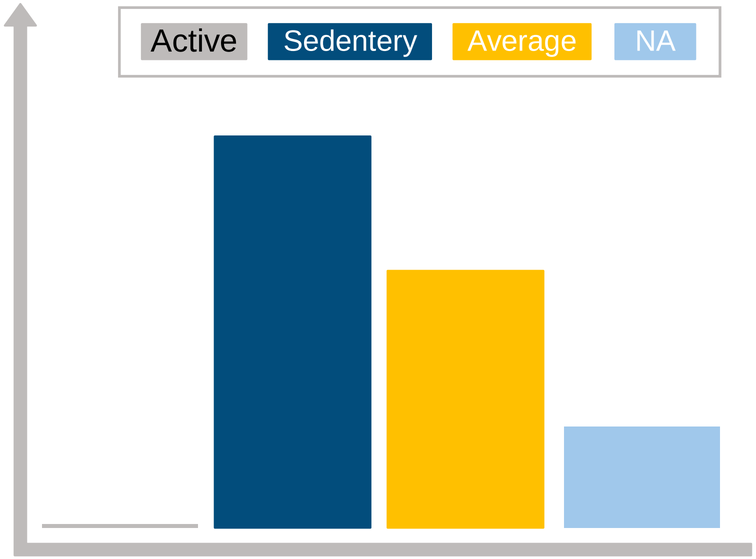
- cor()

```
        height     weight     bmi
 height  1.000000 0.637640 0.052578
 weight  0.637640 1.000000 0.795710
 bmi     0.052578 0.795710 1.000000
```

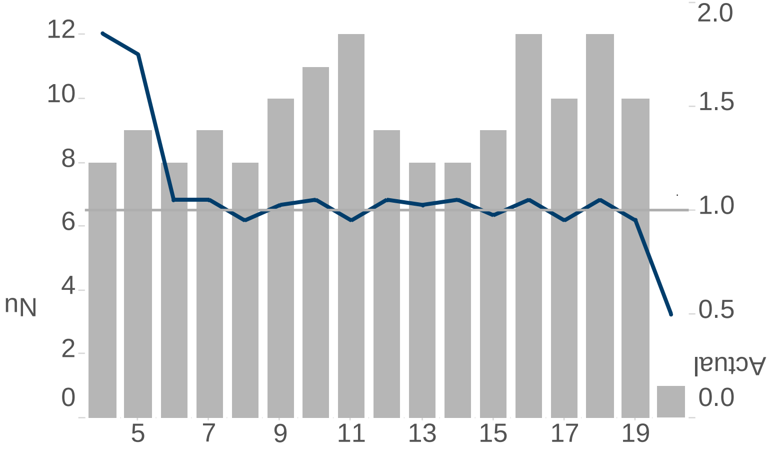
- vif()

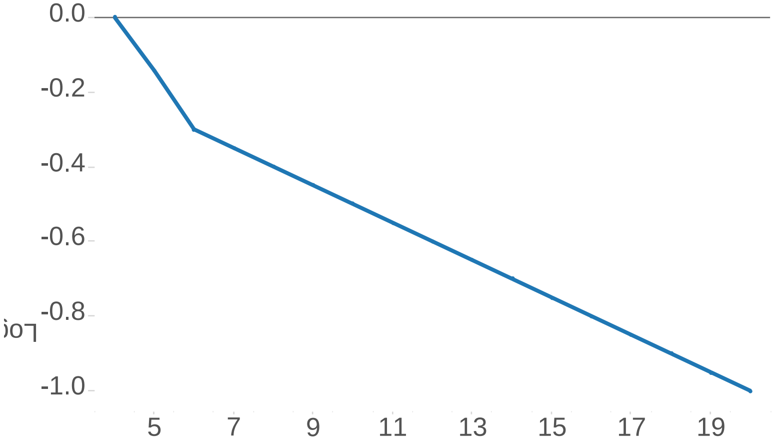
Reference levels





Piecewise linear effects


A/E by predictor before piecewise split


Piecewise impact of example predictor

Undersampling

- For logistic regression, undersampling can help improve runtimes:
  - All deaths (n) +
  - Randomly selected non-deaths (3n)
- Fitting the model Death ~ AttAge

Dataset	Records	Runtime	Intercept	AttAge coefficient
Full	259,284	2.15	-14.13	0.129
Undersampled	25,152	0.12	-10.99	0.123

Validation

SOCIETY OF
ACTUARIES

Validation and comparison

- Overall model fit (4A)
  - Bias-variance tradeoff
- Comparison between two candidate models (4B)

Model fit

- R^2
- AIC/BIC
- Actual-to-expected plots (4A-i)
- Confusion matrix (4A-ii)
- AUC (4A-iii)

Confusion matrix

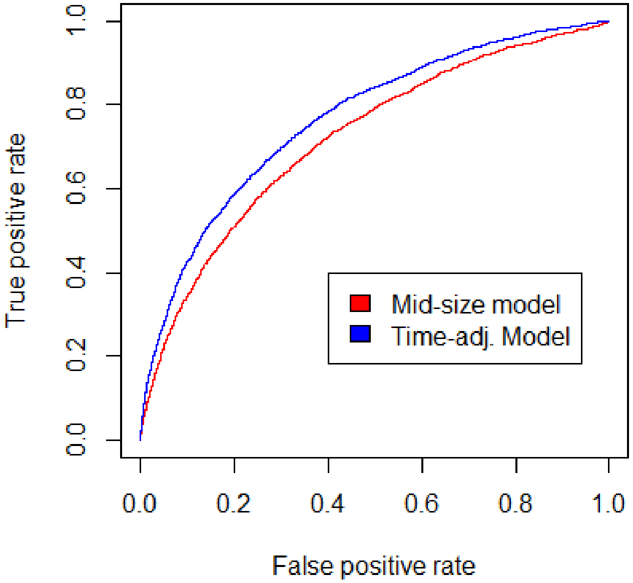
- Select a threshold for predicting the outcome
- Build a 2x2 contingency table

Prediction	Death		
	0	1	Total
0	65,815	835	66,650
1	18,500	1,313	19,813
Total	84,315	2,148	86,463

True positive rate = 1,313/2,148 = 0.658
False positive rate = 18,500/84,315 = 0.301

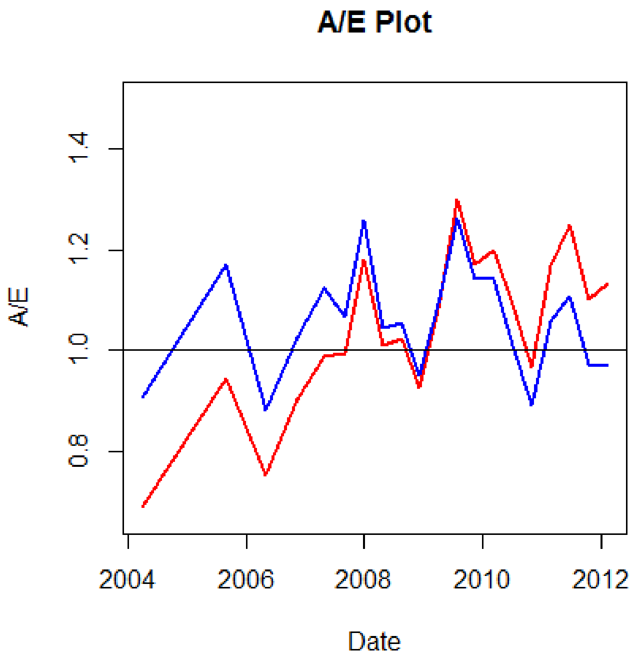
Area under the curve (AUC)

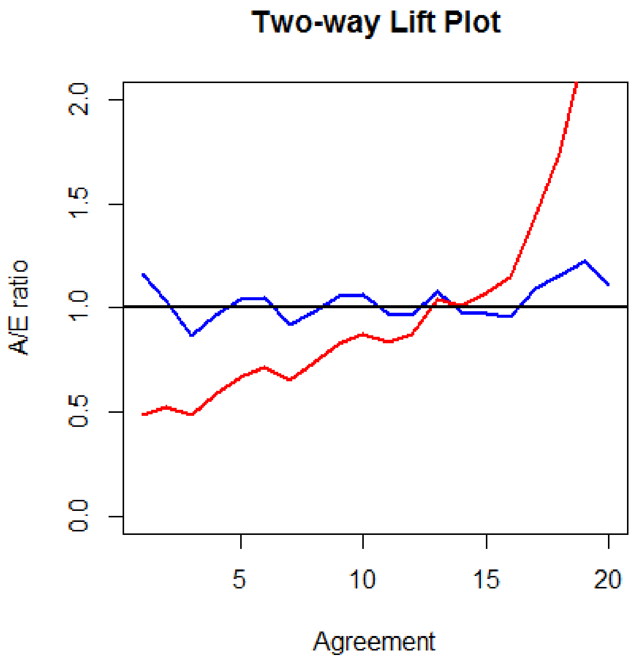
- The curve here is the relationship of the true positive rate and false positive rate as the threshold moves from 0 to 1



Model comparison: Lift charts

- Actual to expected (4B)
- Two-way lift (4B)





PPAS data preparation guide

Society of Actuaries
Life and Annuity Symposium

Eileen S. Burns and Matthias Kullowatz

May 10, 2017

Contents

Intro

Throughout the day, 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 afternoon 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.

Note: we have included only select output to show in this document, but we encourage the reader to run these on his/her own machine to see all output.

Set up R's environment

Set the working directory and load packages that we'll be using.

```r
setwd("C:/work/SOASeminar") # the directory where you're working
library(dplyr)
library(car)
library(zoo)
library(lubridate)
```

Note: in our Intro to R document, we discuss using R Projects for organizing and documenting your work, as well as pre-loading packages and other options each time you open the project.

Data assessment/Formatting variables

Import data sets and bind together. The as.is input preserves categorical variables in the character class, rather than the factor class.

```r
dsData <- read.csv("SampleStudy.rpt", as.is = TRUE)
dsData2 <- read.csv("SampleStudy2.rpt", as.is = TRUE)
dsData3 <- read.csv("SampleStudy3.rpt", as.is = TRUE)
data.full <- rbind(dsData, dsData2, dsData3)
```

Take a quick peak at the field names:

```r
names(data.full)
```

Fix first column name. Notice that for data frames, names() and colnames() are equivalent functions.

```r
colnames(data.full)[1] <- "timeinstudy"
```

Check the field names for potential date fields; often these are loaded incorrectly as character or factor vectors. Let's format the date fields called enteredstudydate, studyenddate, dob, and dod.

```r
date.form <- "%Y-%m-%d"
data.full$dob <- as.Date(as.character(data.full$dob), date.form)
data.full$dod <- as.Date(as.character(data.full$dod), date.form)
data.full$studyenddate <- as.Date(as.character(data.full$studyenddate),
    date.form)
data.full$enteredstudydate <- as.Date(as.character(data.full$enteredstudydate),
    date.form)
```

Check automatically for character fields, and change to factor vectors. This is more a user preference than a requirement, but sometimes it's helpful when R treats these types of categorical variables as factors rather than characters. Other times, dealing with factors can be a pain! You'll learn.

```
classes <- sapply(data.full, class)
for (i in 1:dim(data.full)[2]) {
    if (classes[i] %in% c("character")) {
        data.full[, i] <- as.factor(data.full[, i])
    }
}
```

Identify outliers and missing data

With date and factor fields now edited, using the summary() function can shed some light on outliers and missing data:

```
summary(data.full)
```

A quick review of the output reveals a few interesting things…

There are 36,600 NA values in dod (date of death), and further analysis reveals there are also exactly 36,600 timetodeath values of zero. Here we use a contingency table to show that those are the same 36,600 people.

```
table(is.na(data.full$dod), data.full$timetodeath == 0)
```

```
         FALSE  TRUE
  FALSE  8400     0
  TRUE      0 36600
```

Here we show that there are 4,700 deaths indicated in in the "died" column.

```
sum(data.full$died)  # Number of recorded deaths
```

```
[1] 4700
```

This is concerning because the first bit of code implies that 8,400 people (45,000 - 36,600) died during the study (19%), but only 4,700 are marked as having died in the second bit of code (10%). Let's contruct a second time to death variable to check the first, as well as a death indicator–both based on the "dod" field. We'll see that the timetodeath2 variable matches the original, based on 100% of the observations being within one day of each other. So we'll proceed with our new, correct mortality indicator, death_ind, which is based on the "dod" field.

```
data.full <- data.full %>% mutate(death_ind = ifelse(!is.na(dod),
    1, 0), timetodeath2 = ifelse(!is.na(dod), (dod - enteredstudydate)/365.25,
    0))
mean(abs(data.full$timetodeath - data.full$timetodeath2) < 1/365)
```

```
[1] 1
```

```
# ~ 1-day error tolerance
```

The summary from above (output not shown) also revealed that 10,739 people do not have height or weight, and it is the same people that are missing both. Additionally, those two fields are factors! Let's make them numeric, impute the missing records with a value of zero, and create a missing value indicator field to mark them for later. To convert factors to numerics, you have to first convert to characters as we do below.

```
data.full$height <- as.numeric(as.character(data.full$height))
data.full$weight <- as.numeric(as.character(data.full$weight))
table(is.na(data.full$height), is.na(data.full$weight))
```

```
        
            FALSE  TRUE
      FALSE 34261     0
      TRUE      0 10739
```

```
data.full <- mutate(data.full, ht.wt.flag = ifelse(is.na(height),
    1, 0), height = ifelse(is.na(height), 0, height), weight = ifelse(is.na(weight),
    0, weight))
```

Note that this is just one of many ways to impute missing values for modeling. This method of using a missing value indicator may not be the most appropriate since there is a slight correlation between this indicator and mortality. In other words, the missing values are *informative*. We conducted a two-sample independent proportions test below, and we found that there is a statistically significant difference between mortality proportions for those with and without height and weight values (p = 3.09 x 10^{-15}). The actual magnitude of the difference is only about 3.5 percentage points.

```
prop.test(x = tapply(data.full$death_ind, data.full$ht.wt.flag,
    sum), n = table(data.full$ht.wt.flag), conf.level = 0.95)
```

The consquences of using this missing value indicator are that it might dim some of the effects of height, weight, and BMI on mortality. This method of imputation is still statistically sound, but our interpretation of the model's coefficients will need to reflect our understanding of how missing values were handled. *Note:* Regression Modeling Strategies *by Frank E. Harrell, Jr. has an entire chapter devoted to the handling of missing values for regression analysis.*

Moving on, many of the disease fields have blanks. It's probably fair to assume that this disease was never reported for those people, and we'll want to supply an "unreported" factor level in place of the blank. Here's a quick, automated way to do that. Note that we also "relevel" the new factor variables so that "unreported" becomes the baseline/reference level. After that, we take another look at the data summary to admire our work.

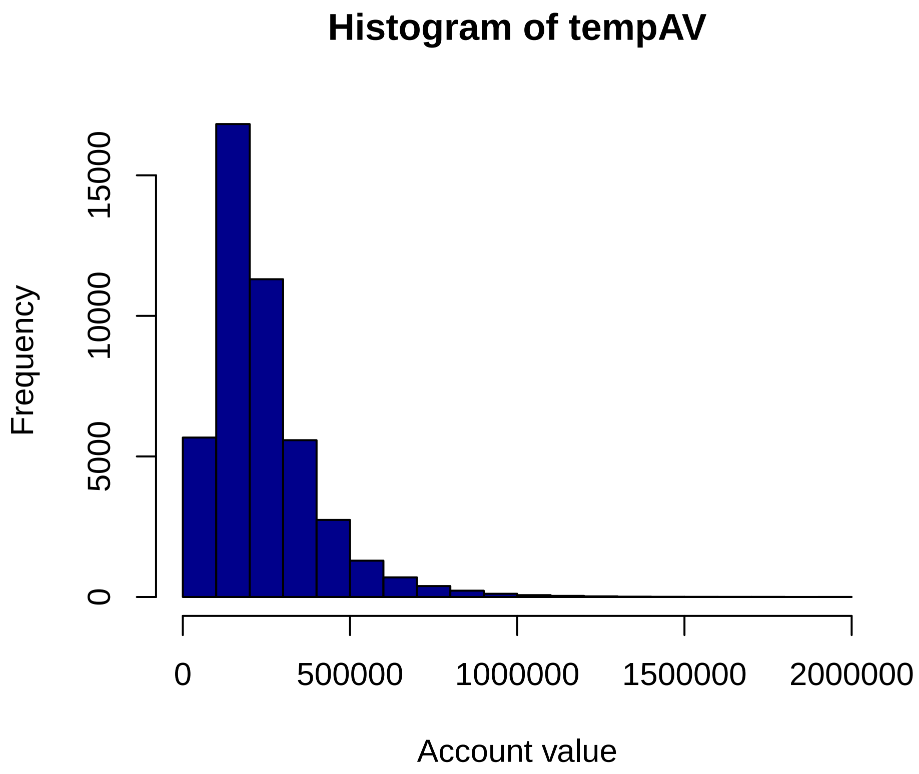
```
(blank.replace <- which(sapply(data.full, function(x) {
    ifelse(class(x) == "factor", sum(x == ""), 0)
}) > 0))
for (i in blank.replace) {
    new.labels <- c("unreported", levels(data.full[, i])[-1])
    data.full[, i] <- relevel(factor(data.full[, i], labels = new.labels),
        ref = "unreported")
}
summary(data.full)
```

Because of the NA values in the cancer_liver field, we need a special fix for that one. Here we'll assume NA cases are "unreported".

```
data.full$cancer_liver[is.na(data.full$cancer_liver)] <- ""
data.full$cancer_liver <- relevel(factor(data.full$cancer_liver,
    labels = c("unreported", "reported")), ref = "unreported")
```

This dataset contains no account information as you might find with life/annuity data, so for the purpose of showing one more technique we'll create a random set of face amounts/account values distributed similarly to how we typically see them in our datasets.

```
set.seed(1)
tempAV <- exp(rnorm(45000, sd = 0.6)) * 2e+05
hist(tempAV, xlab = "Account value", col = "blue4")
```

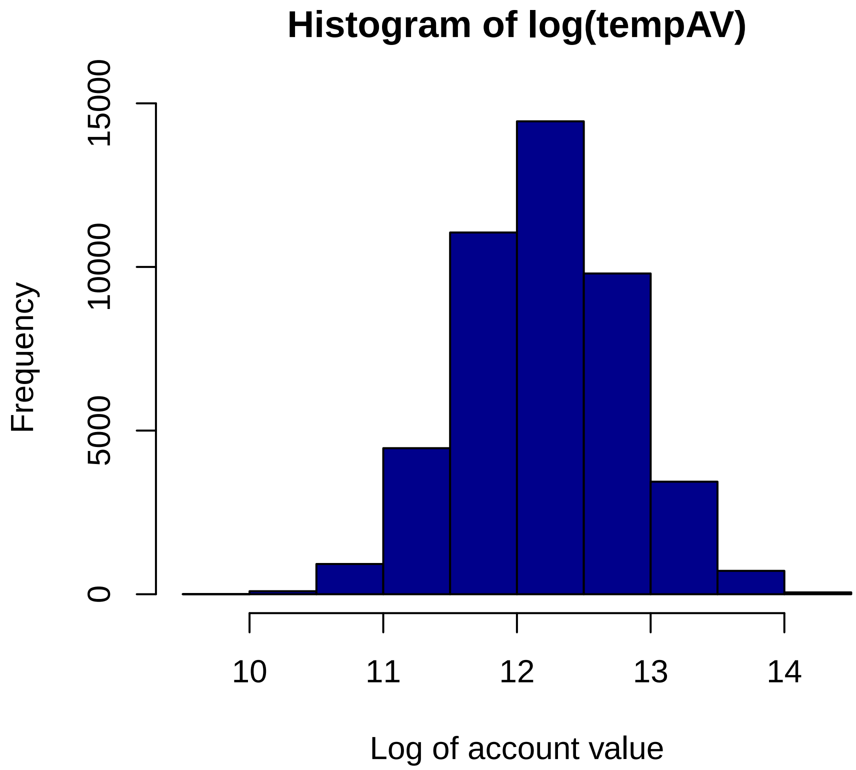


Note that the gap between the maximum value and the 99th percentile value is really wide–more than a million dollars–even though the average is only about $250,000. For skewed distributions this is often the case, and those outliers can have too much influence on linear models. What we often use in modeling is the logarithm of these right-skewed data. Take a look at the log's distribution. It has a recognizable bell shape!

```
unname(max(tempAV) - quantile(tempAV, 0.99))
```

```
    [1] 1147934
```

```
hist(log(tempAV), xlab = "Log of account value", col = "blue4")
```



Create new variables

Mutate a new field for BMI, and for squared BMI distance from the mean (which is 27.2). Also, let's create a grouped variable for all cancers since some are quite rare, and a healthy indicator for those without any reported disease.

```
data.full <- data.full %>% mutate(bmi = ifelse(ht.wt.flag ==
    1, 0, (weight/height^2) * 703), bmisqrerr = ifelse(ht.wt.flag ==
    1, 0, (bmi - mean(bmi[ht.wt.flag == 0]))^2))
(cancer_columns <- grep("cancer", names(data.full)))
```

```
     [1] 24 25 26 27 28 29 30 31 32 33
```

```
cancer_ind <- apply(data.full[, cancer_columns], 1, function(x) {
    min(sum(x == "reported"), 1)
})
healthy_ind <- apply(data.full[, 15:33], 1, function(x) {
    sum(x == "unreported" | x == 0) == 19
})
data.full <- data.frame(data.full, cancer_ind, healthy_ind)
```

Year-by-year dataset

We create a year-by-year dataset so that we can observe policy holders one year at a time. First, we'll determine each observation's number of years in the dataset so that we know how many rows to make. We'll also record the fraction of the last year that each person was under observation.

```
data.full <- data.full %>% mutate(years = ceiling(timeinstudy),
    finalyearfrac = 1 - (years - timeinstudy))
```

```
data.full[["PolNum"]] <- 1:45000
data.large <- data.frame(PolNum = rep(data.full$PolNum, data.full$years))
data.large <- data.large %>% group_by(PolNum) %>% mutate(PolYear = 1:n()) %>%
    ungroup()  # Always ungroup in case you want to group again by something else.
```

Join our original static policy information onto our expanded dataset.

```
data.large <- data.large %>% left_join(data.full, by = "PolNum")
```

Calculate attained age from age at entry and policy year; set YearFrac to be the exposure in each observed year; and indicate when deaths occur, which will be our response (or "target") variable in modeling.

```
data.large <- data.large %>% mutate(AttAge = age + PolYear -
    1, YearFrac = ifelse(PolYear == years, finalyearfrac, 1),
    Death = ifelse(PolYear == years & death_ind == 1, 1, 0))
```

Mutate on a field for current date so that we can later segment an out-of-time sample.

```
data.large <- data.large %>% mutate(current.date = enteredstudydate %m+%
    months(12 * (PolYear - 1)))
```

Save dataset for modeling and validation steps:

```
saveRDS(data.large, file = "PPASExpandedData_20170320.rds")
```

PPAS modeling and validation guide

Society of Actuaries

Life and Annuity Symposium

Eileen S. Burns and Matthias Kullowatz

May 10, 2017

Contents

Intro

In the previous sessions we were introduced to the dataset, we saw how to do some intitial data exploration and cleaning. We also looked at fitting a logistic GLM to the dataset, use stepwise regression to identify significant variables, and assess overall model fit. We also discussed how to compare candidate models. In this session we will explore the world beyond linear models, moving into machine learning methods.

Load packages

```
library(dplyr)
library(lubridate)
library(car)
library(glmnet)
```

Import cleaned data:

```
data.large <- readRDS("PPASExpandedData_20170320.rds")
```

1. Data setup

A. Data exploration

Here we'll take another quick look at the variables in the dataset and their respective summaries. This was already demonstrated in the previous session but will helpful information to have around for this analysis as well.

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

B. Training, holdout, and testing datasets

Now we'll split the dataset into the same subsets it was split into for the previous analysis. A 50% training sample, a 25% in-time holdout sample, and a 25% out-of-time holdout sample. If you were continuing on from the previous analysis you wouln't need to repeat this process.

```
(test.cut.date <- quantile(data.large$current.date, 0.75, type = 1))
```

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

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

2. Machine learning methods

A. Regularization

Regularization is a technique that is mostly used to avoid the problem of overfitting. The idea is to add a complexity term to the loss function to penalize more complex models. The package glmnet

Let's dive into our first regularization method, ridge regression. Ridge regression, sometimes also called weight decay, adds an L2-norm penalty term to the loss function.

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

```
    Call:
    glm(formula = form.mid, family = binomial(link = "logit"), data = filter(data.large,
        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. The effect sizes are given by the coefficient "Estimate" column, and you can assess statistical significance in the P(>|z|) column, which gives the Wald p-values.

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.

It should be noted that stepwise selection can lead to an overconfident model, one that things more highly of its effects (its coefficient estimates) than is actually true. One should always perform some method of model validation.

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, I have limited the algorithm to just one step here because even that took nearly a full 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 + AttAge)
```

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

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 I've chosen variables for this stepwise selection that were among the top variables to be added from the last stepwise function. This method takes about half as long to arrive at the one-step model.

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

```r
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. In *Regression Modeling Strategies*, author Frank E. Harrell, Jr. notes that this top-down approach is preferrable a bottom-up approach (next section) for a few reasons, not the least of which is that it "usually performs better…especially when collinearity is present." We discuss (multi)collinearity in a later section.

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 I will include the code for a full, one-directional backward stepwise method.

```r
system.time(step(glm(form.medium, family = binomial, data = filter(data.large,
    Sample == "training")), direction = "backward", k = 2))  # This is the default value, which leads t
```

iii) Bottom-up approach

Generally, a bottom-up approach–or "forward" stepwise selection–starts with an empty model, and then 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 self explanatory, 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 (yes, we realized that's like saying ATM machine).

```
(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), and "cognitive", "cad", and "tia" are distant runner-ups. This suggests that our original four-variable model may not have started with the best-fitting team of variables, but more steps would be required to confirm that.

Let's make our final model from this section the result of a BIC bi-directional model selection process, starting with the full model.

```
(data.dim <- dim(filter(data.large, Sample == "training")))
system.time(stepmodel.final <- step(glm(formula = form.medium,
    family = binomial(link = "logit"), data = filter(data.large,
        Sample == "training")), scope = form.medium, direction = "both",
    k = log(data.dim[1])))
```

After a few minutes, we have arrived at a model that we'll evaluate going forward. One thing to note is that the flag for missing height/weight values was removed from the model. The stepwise algorithm may not have understand its importance, and we should definitely consider it moving forward.

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

E. Multicollinearity

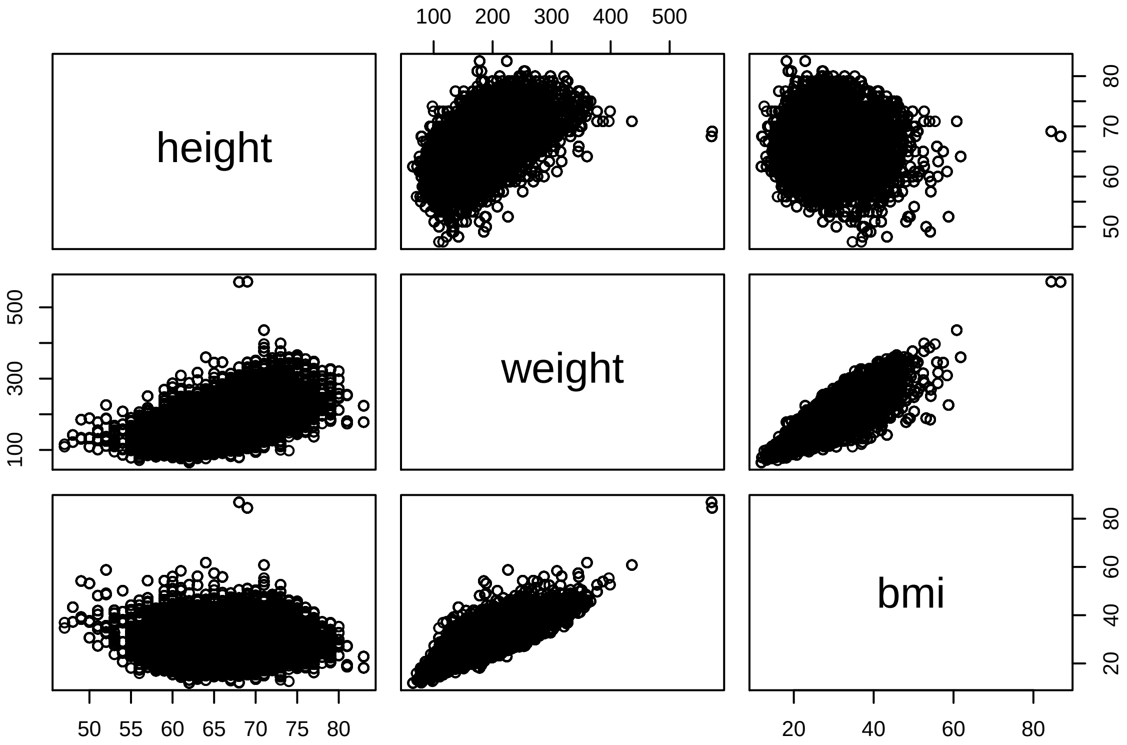
We recommended that you use bi-directional stepwise methods before, and that is true regardless of your criteria and whether you are going with a top-down or bottom-up approach. The correlation between variables in the dataset is referred to as multicollinearity, and its consequences require that the 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 == 0) %>% 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 == 0) %>% 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 = filter(data.large,
    Sample == "training")))

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

summary(glm(Death ~ ht.wt.flag + weight + height + bmi, family = binomial,
    data = filter(data.large, 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 at the 5% level. This is a good example of why it's important to monitor the linear correlation between candidate predictor variables. Not only were the coefficients highly sensitive to the addition of new variables, but the standard errors of the coefficients grew with the addition of new variables. As an example, the standard error on the weight coefficient grew from 0.0005, to 0.0006, to 0.0050.

Now, back to our model. Let's assess the correlation between our model's variables in terms of Variance Inflation Factors (VIF's).

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

```
                     GVIF Df GVIF^(1/(2*Df))
AttAge           1.137976  1        1.066760
cancer_ind       1.015396  1        1.007668
smoker           1.016890  1        1.008410
cad              2.025490  3        1.124834
cognitive        1.267572  3        1.040308
tia              2.110898  3        1.132603
diabetes         1.100237  3        1.016048
adls             1.394092  1        1.180717
activitylevel    1.206108  3        1.031726
height        2150.881620  1       46.377598
weight         609.197311  1       24.681923
bmi            557.970545  1       23.621400
ht.wt.flag    2080.970546  1       45.617656
```

Yikes! Using this metric (the right-most column from the output), 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 practically break a linear model. Recall that a minium VIF value is 1, which means that particular variable is completely uncorrelated with all the other predictor variables in the model. 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. In simple terms, GVIF's allow us to measure multicollinearity for models that include categorical variables.

We certainly want to do something about the multicollinearity. This is an instance where human intuition is preferable to throwing more algorithms at the problem. We know that BMI makes an attempt to score one's weight relative to height, so let's use that one.

```
form.final <- as.formula(Death ~ AttAge + cancer_ind + smoker +
    cad + cognitive + tia + diabetes + adls + activitylevel +
    bmi + ht.wt.flag)
model.final <- glm(form.final, family = binomial, data = filter(data.large,
    Sample == "training"))
```

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 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 values. Here's an example of such analysis across attained age and BMI.

First, we append the predictions to the whole dataset.

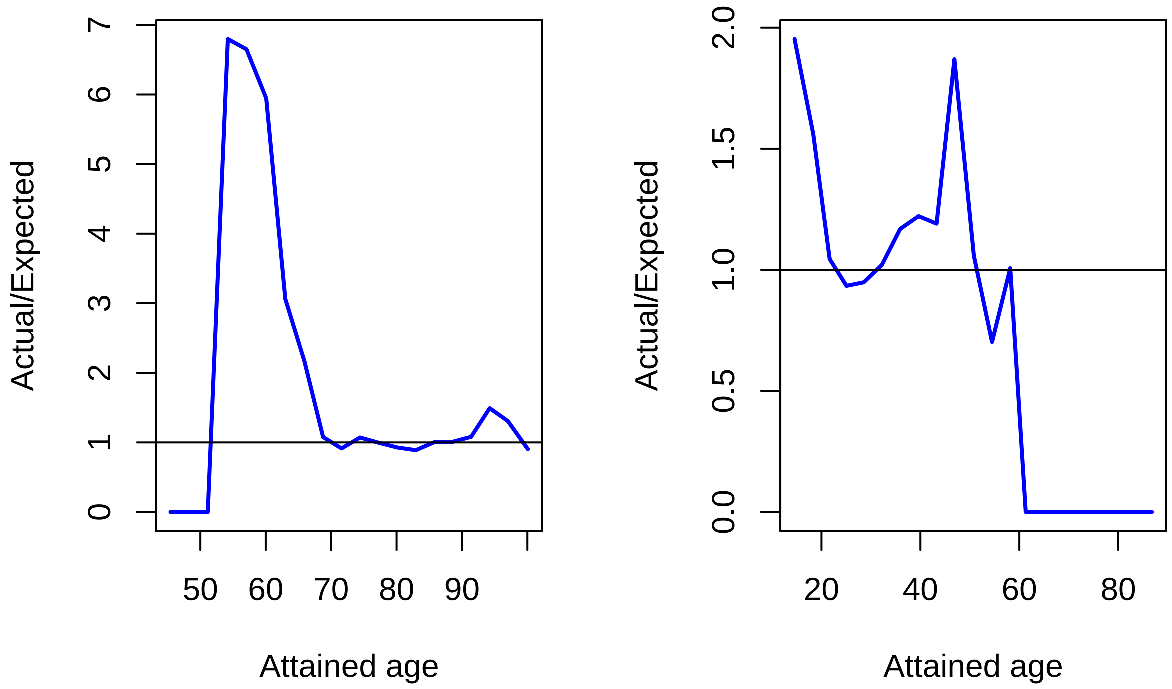
```
data.large <- data.frame(data.large, Preds.large = predict(model.final,
    newdata = data.large, type = "response"), Preds.middle = predict(model.mid,
    newdata = data.large, type = "response"))
```

Then, we break up observations into buckets by attained age and BMI values using the cut() function. We group observations into their buckets and calculate 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())
BMI.plotdata <- data.large %>% filter(Sample == "training", ht.wt.flag ==
    0) %>% group_by(BMI.cut = cut(bmi, breaks = 20)) %>% summarize(BMI.avg = mean(bmi),
    Deaths = mean(Death), PredDeaths = mean(Preds.large), AE.bmi = mean(Death)/mean(Preds.large),
    N = n())
```

Now let's plot the A/E ratio over attained age. I use the type = "l" input to create a line graph, rather than a scatter plot.

```
par(mfrow = c(1, 2))
plot(x = AttAge.plotdata$AttAge.avg, y = AttAge.plotdata$AE.attage,
    type = "l", lwd = 2, col = "blue", xlab = "Attained age",
    ylab = "Actual/Expected")
abline(h = 1)
plot(x = BMI.plotdata$BMI.avg, y = BMI.plotdata$AE.bmi, type = "l",
    lwd = 2, col = "blue", xlab = "Attained age", ylab = "Actual/Expected")
abline(h = 1)
```



Those plots strongly suggest non-linearity across both variables. Combining the attained age plot with its AttAge.plotdata chart, we see that the high A/E values from ages 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. As for BMI, most of the data exist below a value of 45, and the quadratic appearance in that range of BMI suggests that we could improve fit with a squared version of BMI (fortunately, we already made one!).

Note that our human intuition is useful here as a final filter. For BMI especially, we should expect mortality to increase for both very low and very high values, and a quadratic effect makes a lot of sense. For attained age, the reason for the non-linearity is more ambiguous, implying that it may be spurious. Here, we'll include adjustments for both variables for the sake of demonstrating strategies for dealing with non-linear relationships and their corresponding code.

For attained age, 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." We'll use "bmisqrerr," the centered-squared version of "bmi."

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

Now let's see how our model likes those "new" variables.

```
form.nonlin <- as.formula(Death ~ AttAge + AttAge2 + cancer_ind +
    smoker + cad + cognitive + tia + diabetes + adls + activitylevel +
    bmi + bmisqrerr + ht.wt.flag)
model.nonlin <- glm(form.nonlin, family = binomial, data = filter(data.large,
    Sample == "training"))
```

```
summary(model.nonlin)
```

Our additions appear to be 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. As mentioned before, 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 entire dataset:

```
data.large <- data.frame(data.large, Preds.nonlin = predict(model.nonlin,
    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 6,288 deaths in our training dataset, so randomly selecting 6,288 non-deaths would be a symmetric way to create a new dataset. However, that symmetry is not statistically necessary, largely due to the fact that the coefficients of a logistic GLM are odds ratios in disguise, and odds ratios are immune to undersampling.

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, but all the model's coefficients except the intercept 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.

```
ratio <- 3  # Thrice as many non-deaths as deaths
data.deaths <- data.large %>% filter(Sample == "training" & Death ==
    1)
data.sample <- sample_n(tbl = data.large %>% filter(Sample ==
    "training" & Death == 0), size = ratio * dim(data.deaths)[1])
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 make final model edits and to show others how well our model could do when applied to new data. We'll stick to the in-time holdout portion here 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

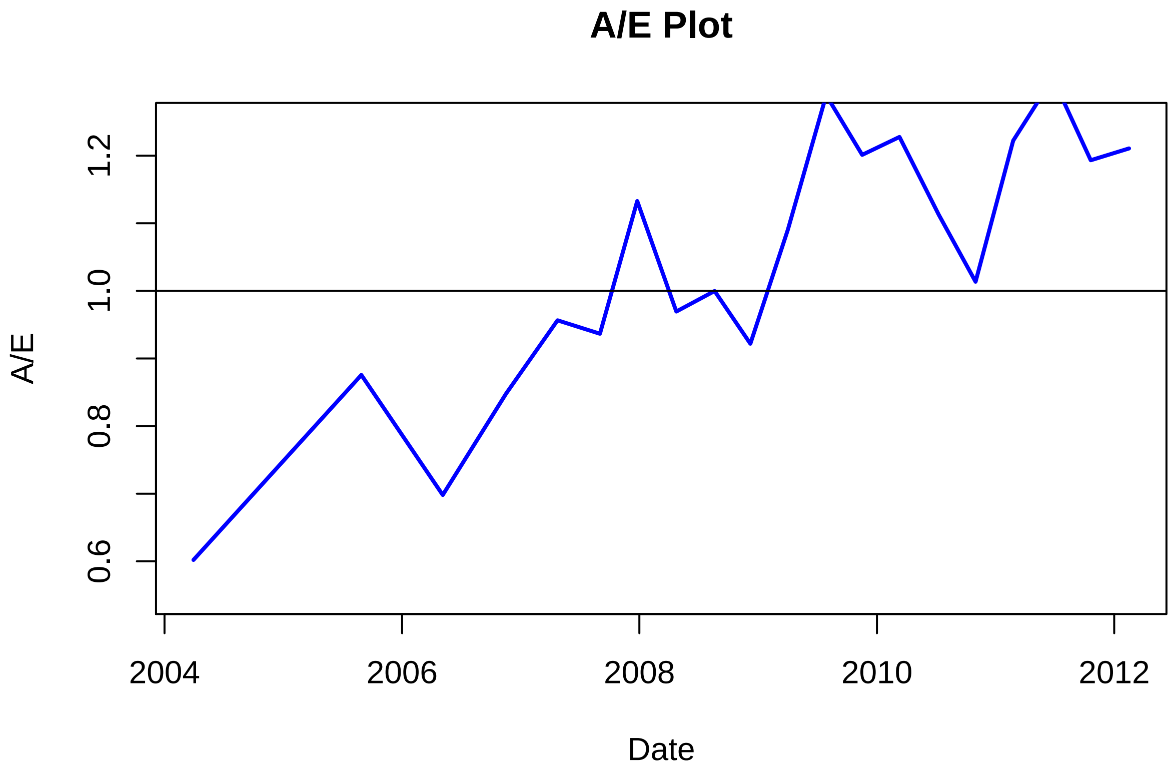
There are many ways to test a logistic model's fit, including A/E ratio plots, confusion matrices, and calculation of the Area under the ROC curve (AUC).

i) Actual to expected plots

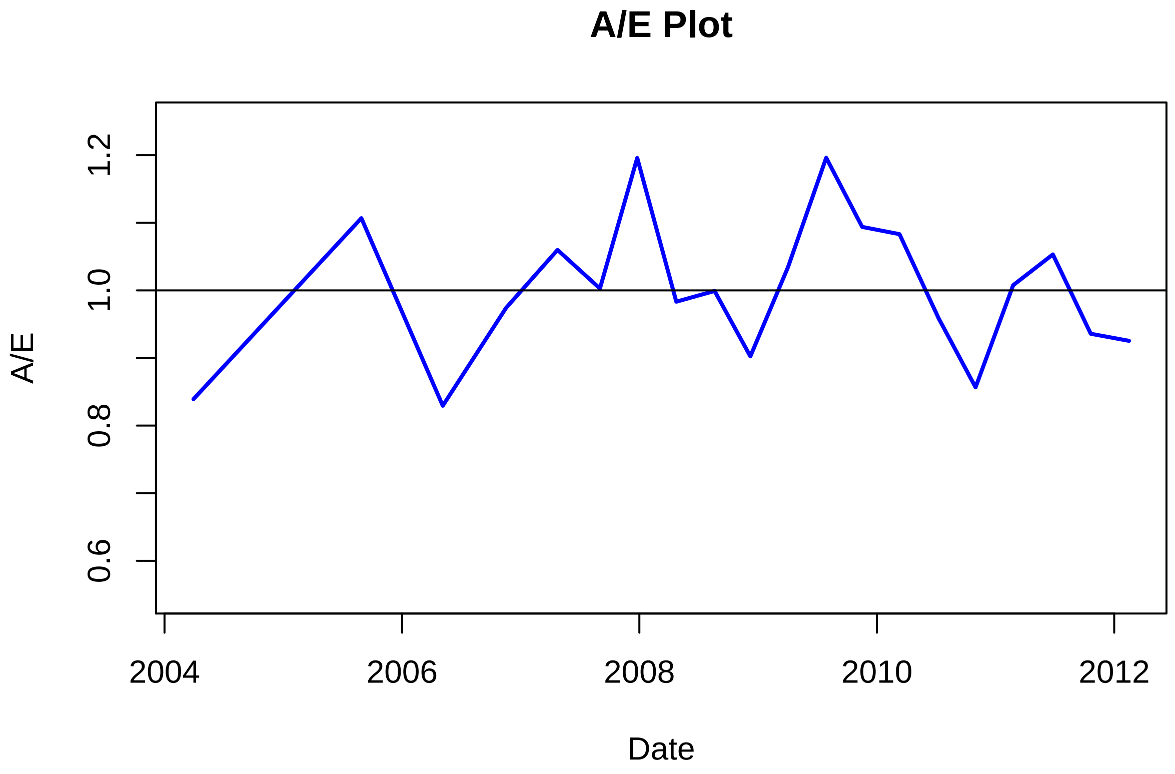
Typically, we first create A/E plots across each of the variables in our model using training data, and also across time-related variables like policy year or valuation date. Then we make tweaks as we did with attained age and BMI. Later, we review these A/E plots using the in-time holdout dataset.

As an example, here we'll make an A/E ratio plot across time. Our holdout 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 110 deaths and 4,500 total observations per bucket.

```
date.plotdata <- data.large %>% filter(Sample == "holdout") %>%
    group_by(Date.bin = ntile(current.date, 20)) %>% summarize(Date = mean(current.date),
    AE.date = sum(Death)/sum(Preds.nonlin))
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))
lines(date.plotdata$Date, date.plotdata$AE.date, lwd = 2, col = "blue")
abline(h = 1)
```



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.



There, errors now look more random across the time dimension. The in-time "holdout" dataset can be used to make further tweaks to the model. The out-of-time "testing" dataset is what you'd use to convince someone that the model you've created can be used to make future predictions.

Note: If we had dug into this data deeper a little earlier, we would have seen that the mortality rates plummet in the final year or so of exposure, which represents a large portion of the testing data subset. In a full study, we'd want to go figure out what happened with the data before getting into modeling.

ii) Confusion matrices

A confusion matrix, also known as a 2x2 contingency table, cross tabulates observations over predictions and actual outcomes. If you're not familiar with these tables, we have provided an example from base R below. The "caret" package, among others, has functions for developing contingency tables and calculating classification metrics.

```
data.large <- data.large %>% filter(Sample == "holdout") %>%
    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 60400   700
           1 23915  1448
```

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

```
[1] 0.6741155
```

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

```
[1] 0.7163613
```

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 assigned the one that died a higher probability of doing so? A 50% AUC would basically be a model that randomly assigned death probabilities, so that becomes our baseline.

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

Note: typically we'd do this on an unused holdout dataset, like our testing subset. However, as mentioned before, this particular dataset has some oddities in the testing subset.

```
library(ROCR)
```

```
Loading required package: gplots


Attaching package: 'gplots'

The following object is masked from 'package:stats':

    lowess
```

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

Now let's compare the AUC percentages. The following output shows that the larger model outperformed the smaller one by AUC, 76.0% to 71.7%.

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

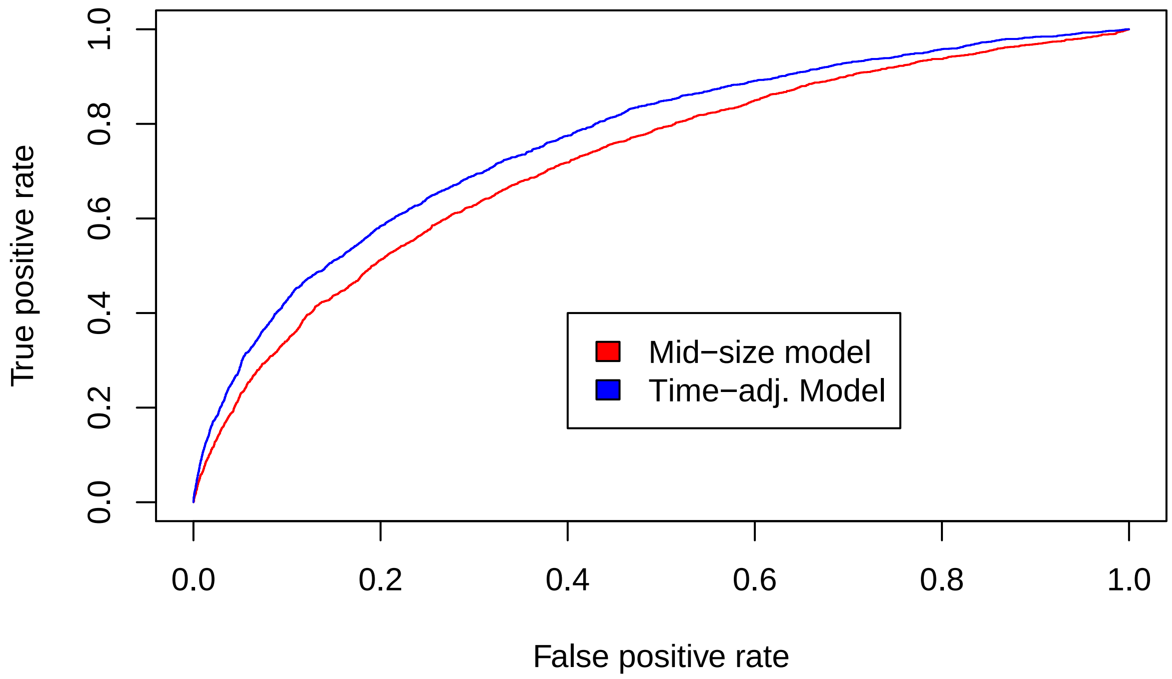
```
[1] 0.7203179
```

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

```
[1] 0.7651487
```

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"))
```



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 (x-axis) would be considered the best model.

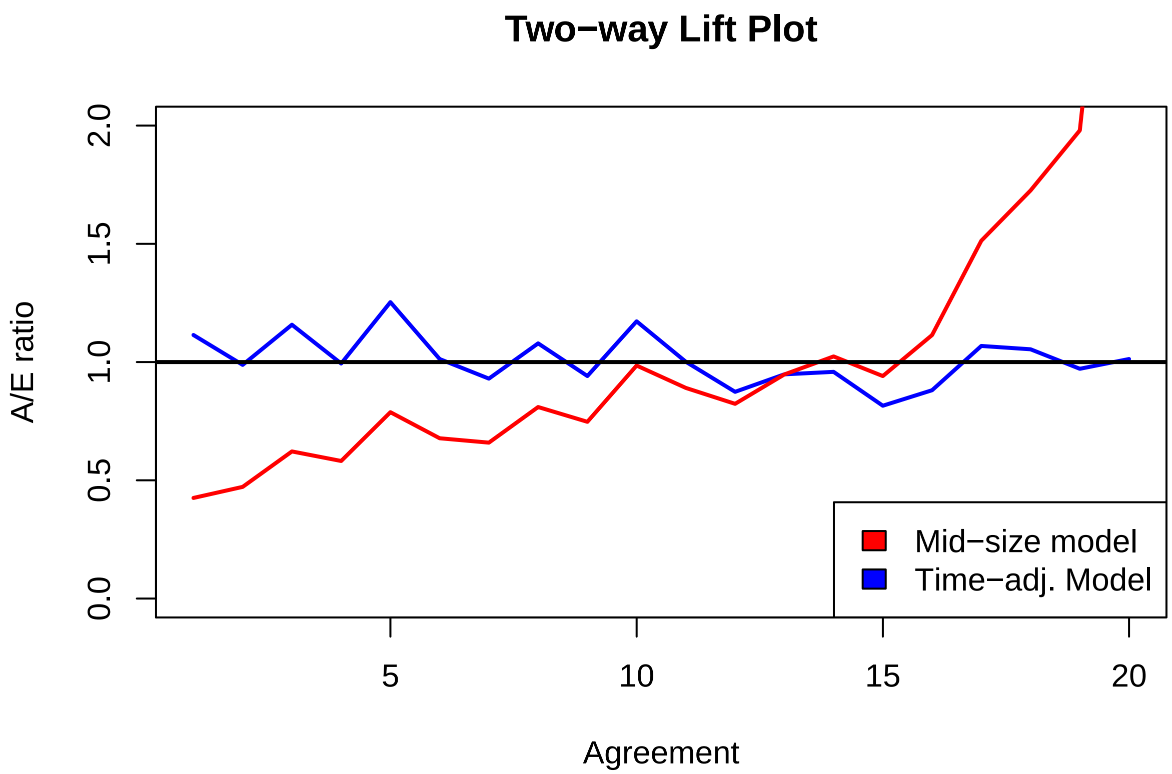
First, the plot data:

```
data.twoway <- filter(data.large, Sample == "holdout") %>% group_by(Agreement = ntile(Preds.time/Preds.
    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, cex = 0, ylim = c(0,
    2), xlab = "Agreement", ylab = "A/E ratio", main = "Two-way Lift Plot")
legend("bottomright", legend = c("Mid-size model", "Time-adj. Model"),
    fill = c("red", "blue"))
lines(data.twoway$Agreement, data.twoway$AE.large, lwd = 2, col = "blue")
lines(data.twoway$Agreement, data.twoway$AE.middle, lwd = 2,
    col = "red")
```

```r
abline(h = 1, lwd = 2)
```



The plot shows that the fuller model did better across nearly the entire range of Agreement (blue line closer to 1).

Practical Predictive Analytics Seminar

Talex Diede, MS

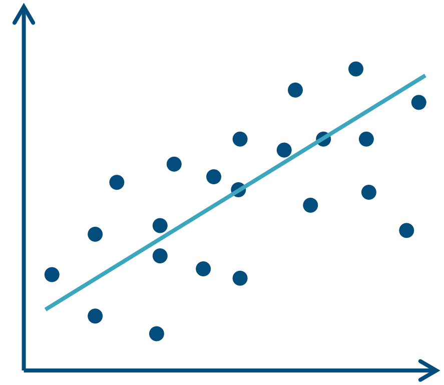
Session 3: Machine Learning Topics

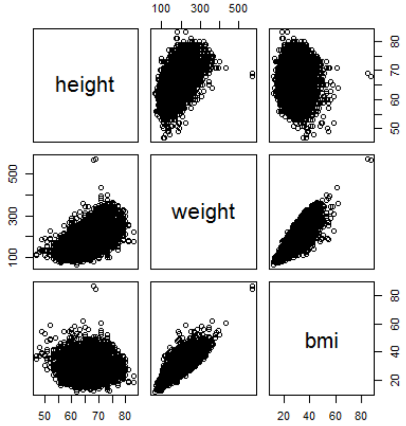
May 10, 2017

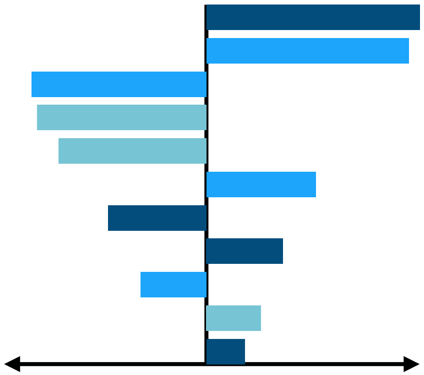
SOCIETY OF ACTUARIES

GLM review

- Linear model
- Interpretable
- Issues:
  - Multicollinearity
  - Variable selection
  - Variable importance
  - Interactions

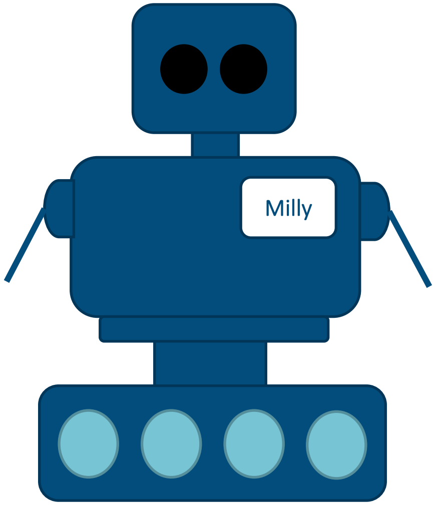






Machine learning techniques

- Regularization methods
- Classification and regression trees
- Ensemble models
- Others:
  - Clustering
  - Bayesian
  - Neural network
  - Deep learning

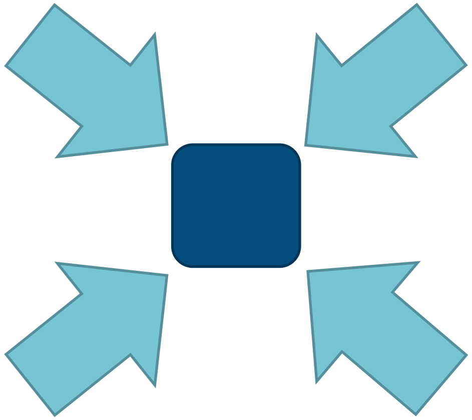


Regularization Methods

SOCIETY OF ACTUARIES

What is “regularization”?

- Regularization is a technique used to avoid the problem of overfitting. The idea is to add a complexity term to the loss function to penalize more complex models.



Regularization methods

- Ridge regression
- LASSO
- ElasticNet

- In R:
  - Packages: **glmnet**, MASS, ridge, lars, elasticnet, …

Ridge regression

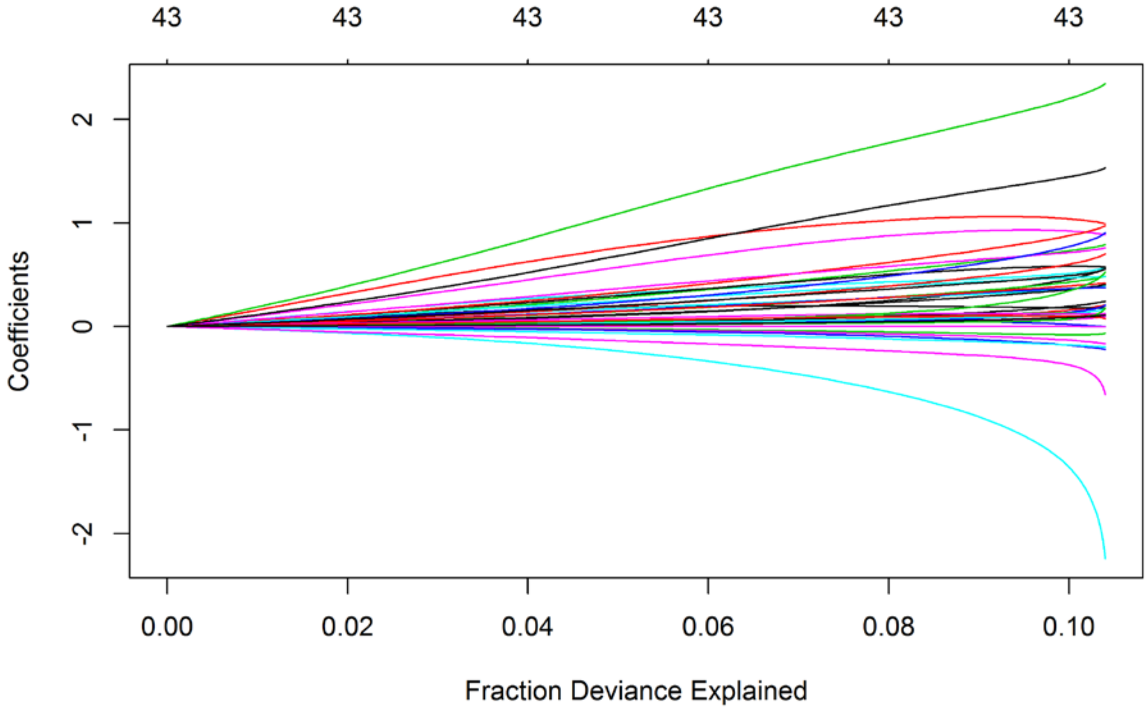
- weight decay
- L2-norm penalty
- $loglikelihood =$

$$-\sum[Y_i \ln(\hat{y}_i) + (1 - Y_i)\ln(1 - \hat{y}_i)] + \boldsymbol{\lambda}\sum \boldsymbol{\beta}^{\mathbf{2}}$$

Example (2A)

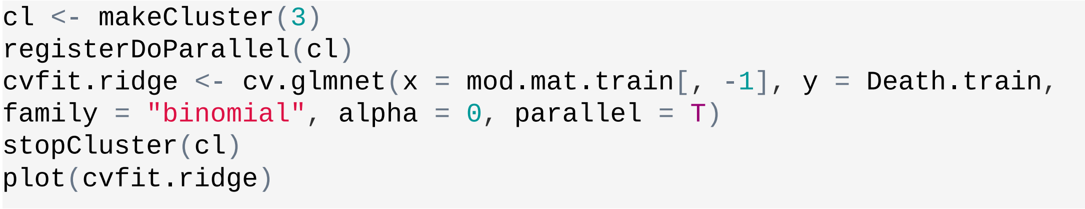
i. Ridge regression

```r
fit <- glmnet(x = mod.mat.train[, -1], y = Death.train, family = "binomial", alpha = 0)
plot(fit, xvar = "dev")
```

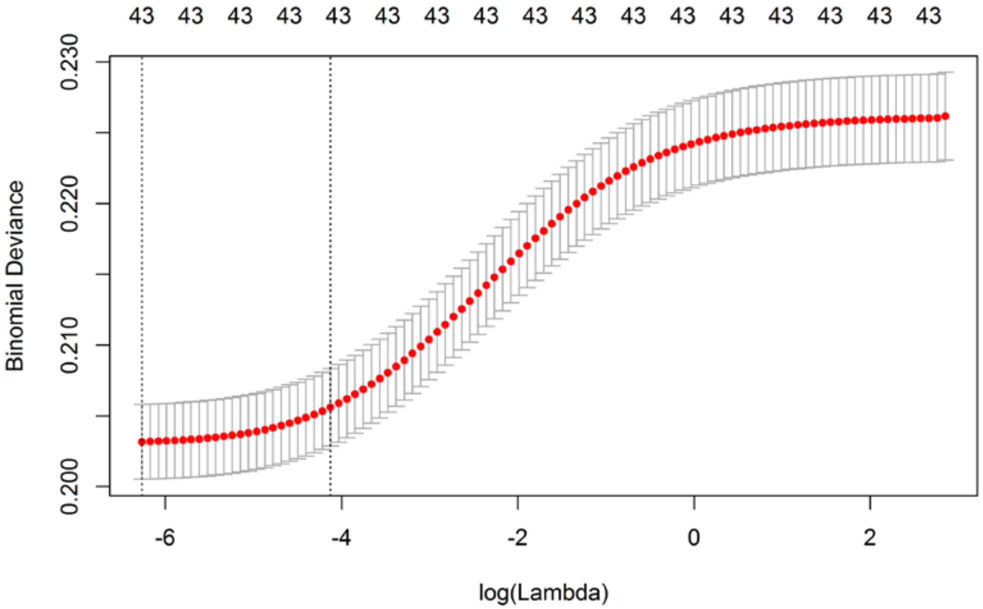


Example (2A)

i. Ridge regression

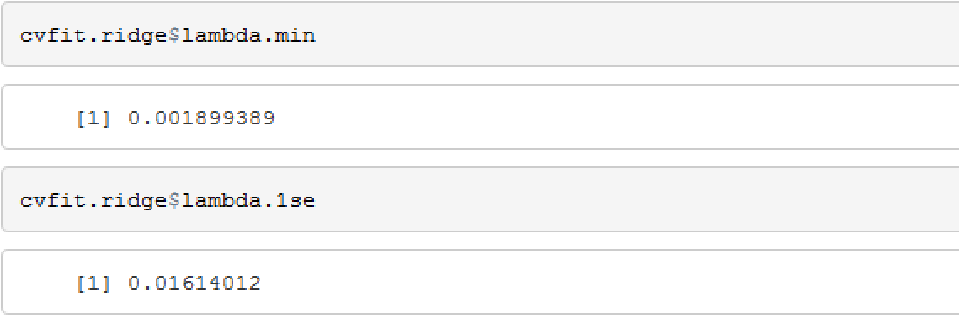


```
cl <- makeCluster(3)
registerDoParallel(cl)
cvfit.ridge <- cv.glmnet(x = mod.mat.train[, -1], y = Death.train,
family = "binomial", alpha = 0, parallel = T)
stopCluster(cl)
plot(cvfit.ridge)
```



Example (2A)

i. Ridge regression

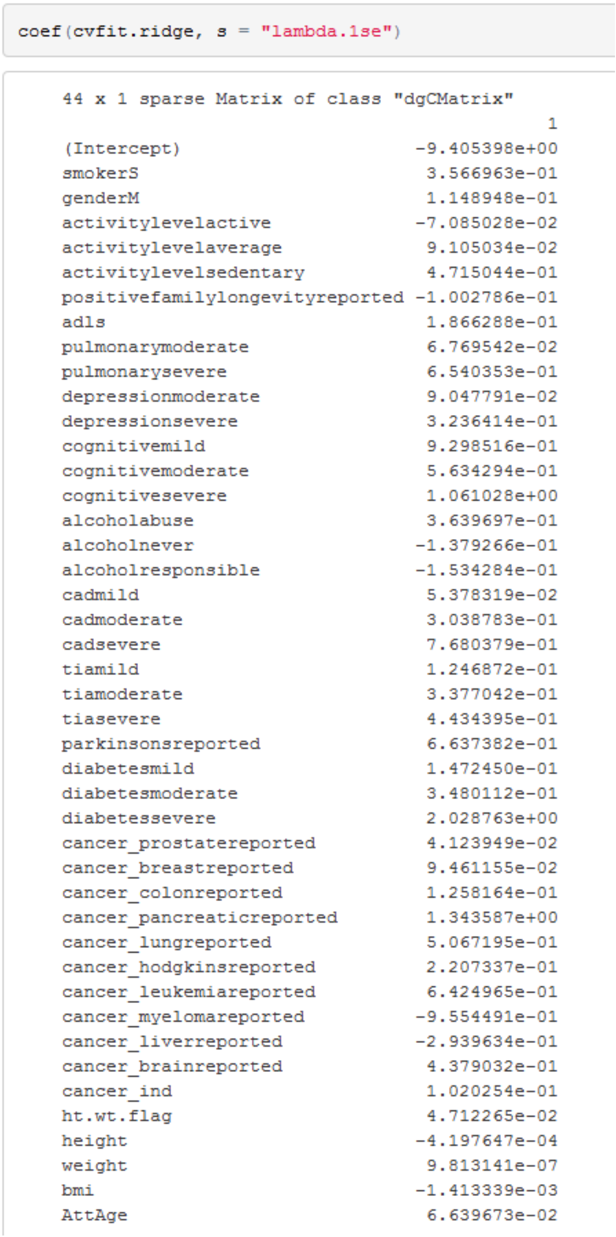


```
cvfit.ridge$lambda.min
```

```
[1] 0.001899389
```

```
cvfit.ridge$lambda.1se
```

```
[1] 0.01614012
```

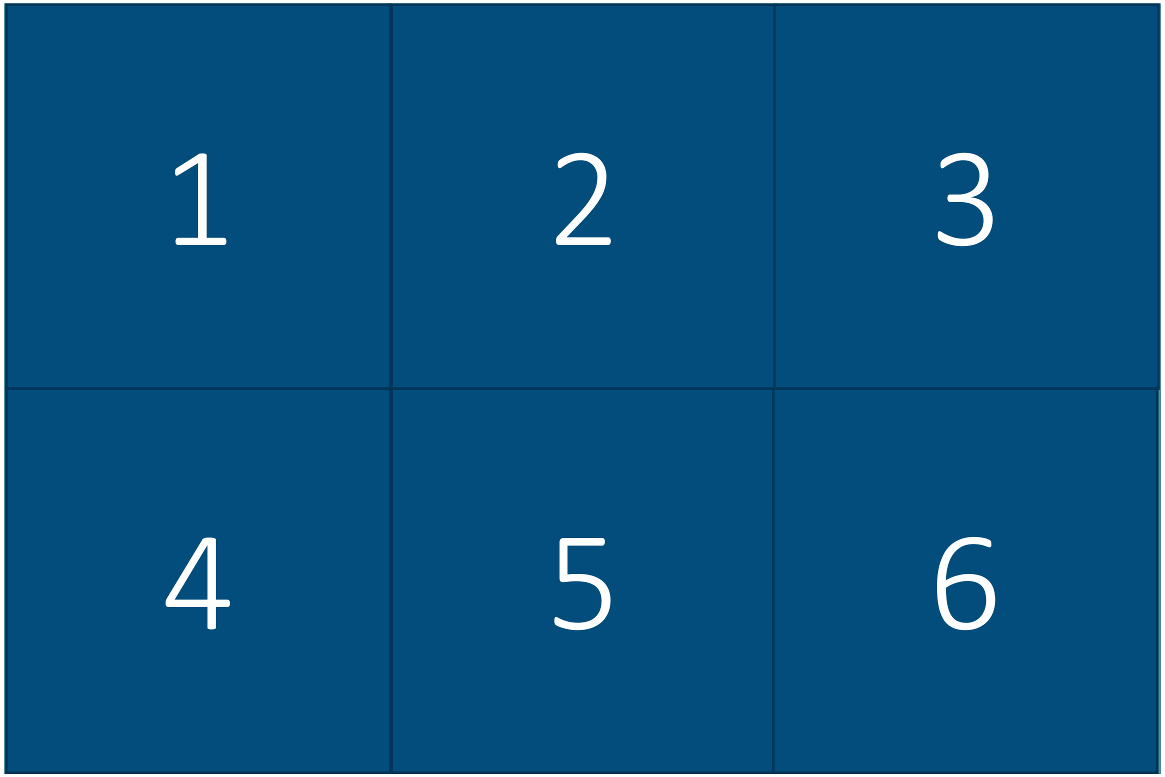


```
coef(cvfit.ridge, s = "lambda.1se")
```

```
44 x 1 sparse Matrix of class "dgCMatrix"
                                            1
(Intercept)                     -9.405398e+00
smokerS                          3.566963e-01
genderM                          1.148948e-01
activitylevelactive             -7.085028e-02
activitylevelaverage             9.105034e-02
activitylevelsedentary           4.715044e-01
positivefamilylongevityreported -1.002786e-01
adls                             1.866288e-01
pulmonarymoderate                6.769542e-02
pulmonarysevere                  6.540353e-01
depressionmoderate               9.047791e-02
depressionsevere                 3.236414e-01
cognitivemild                    9.298516e-01
cognitivemoderate                5.634294e-01
cognitivesevere                  1.061028e+00
alcoholabuse                     3.639697e-01
alcoholnever                    -1.379266e-01
alcoholresponsible              -1.534284e-01
cadmild                          5.378319e-02
cadmoderate                      3.038783e-01
cadsevere                        7.680379e-01
tiamild                          1.246872e-01
tiamoderate                      3.377042e-01
tiasevere                        4.434395e-01
parkinsonsreported               6.637382e-01
diabetesmild                     1.472450e-01
diabetesmoderate                 3.480112e-01
diabetessevere                   2.028763e+00
cancer_prostatereported          4.123949e-02
cancer_breastreported            9.461155e-02
cancer_colonreported             1.258164e-01
cancer_pancreaticreported        1.343587e+00
cancer_lungreported              5.067195e-01
cancer_hodgkinsreported          2.207337e-01
cancer_leukemiareported          6.424965e-01
cancer_myelomareported          -9.554491e-01
cancer_liverreported            -2.939634e-01
cancer_brainreported             4.379032e-01
cancer_ind                       1.020254e-01
ht.wt.flag                       4.712265e-02
height                          -4.197647e-04
weight                           9.813141e-07
bmi                             -1.413339e-03
AttAge                           6.639673e-02
```

Aside: Cross-Validation

- Useful for smaller datasets



1 2 3

4 5 6

LASSO

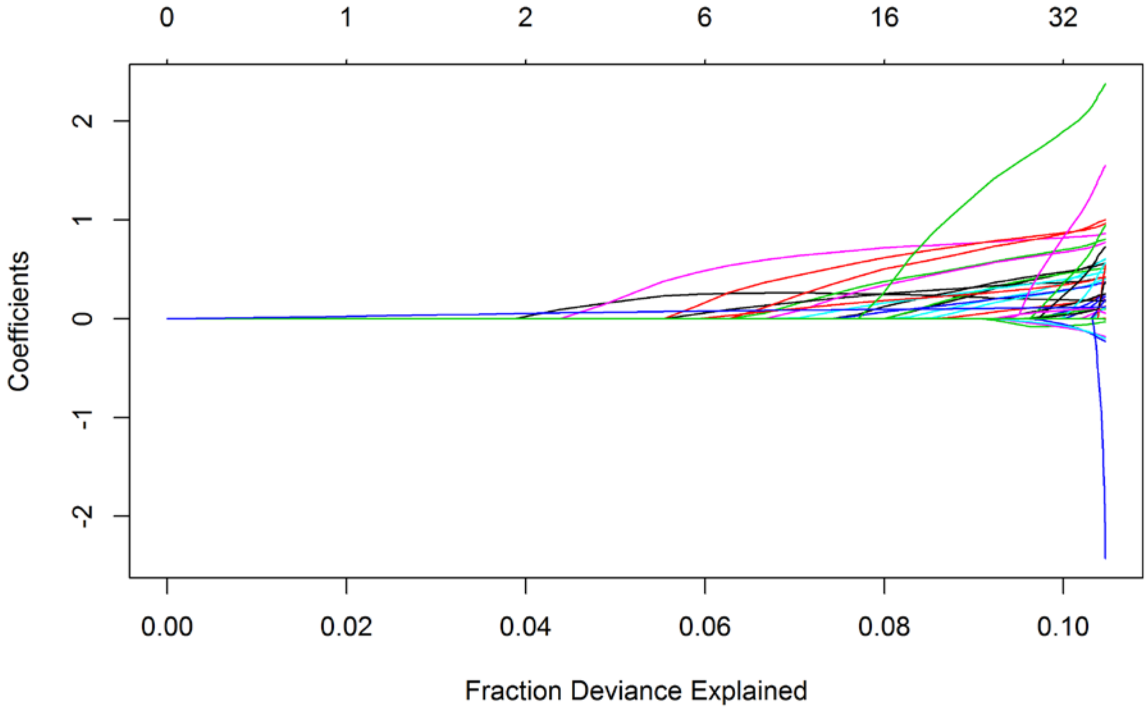
- Least absolute shrinkage and selection operator
- L1-norm penalty
- $loglikelihood =$

$$-\sum[Y_i \ln(\hat{y}_i) + (1 - Y_i)\ln(1 - \hat{y}_i)] + \boldsymbol{\lambda}\sum|\boldsymbol{\beta}|$$

Example (2A)

ii. LASSO

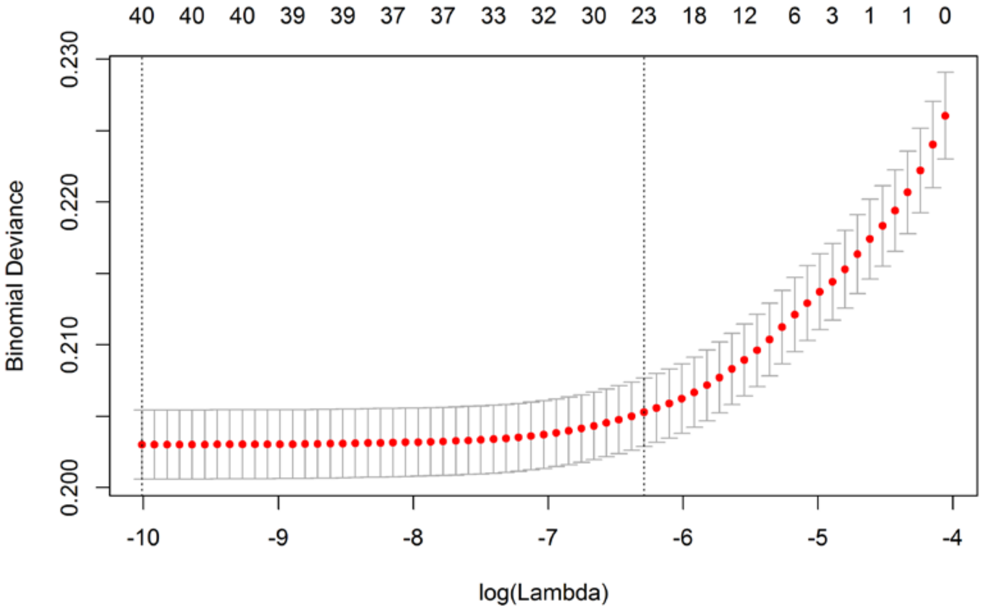
```r
fit <- glmnet(x = mod.mat.train[, -1], y = Death.train, family = "binomial", alpha = 1)
plot(fit, xvar = "dev")
```



Example (2A)

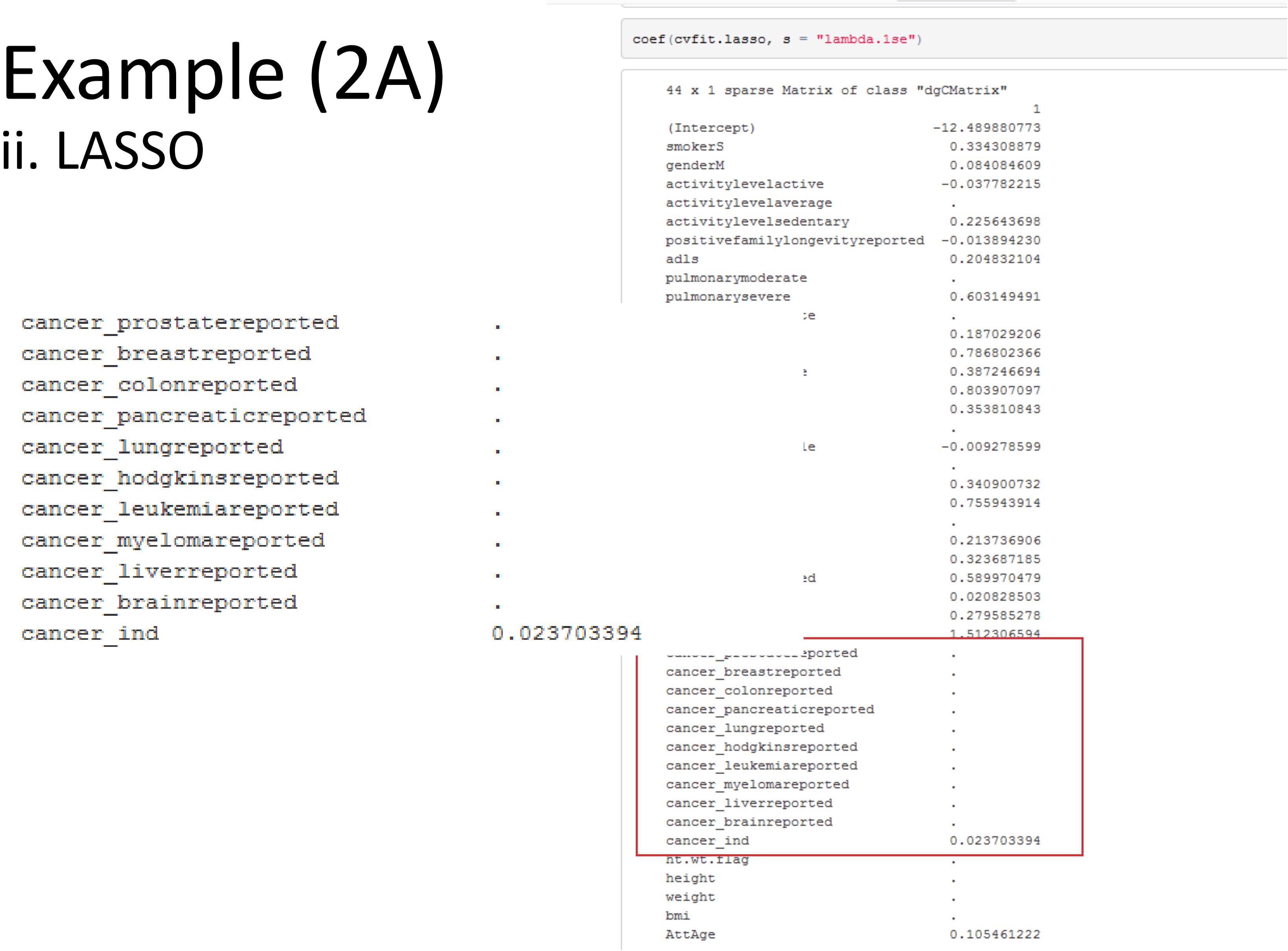
ii. LASSO

```
cl <- makeCluster(3)
registerDoParallel(cl)
cvfit.lasso <- cv.glmnet(x = mod.mat.train[, -1], y = Death.train,
family = "binomial", alpha = 1, parallel = T)
stopCluster(cl)
plot(cvfit.lasso)
```



Example (2A)

ii. LASSO



```
cancer_prostatereported            .
cancer_breastreported              .
cancer_colonreported               .
cancer_pancreaticreported          .
cancer_lungreported                .
cancer_hodgkinsreported            .
cancer_leukemiareported            .
cancer_myelomareported             .
cancer_liverreported               .
cancer_brainreported               .
cancer_ind                         0.023703394
```

```
coef(cvfit.lasso, s = "lambda.1se")
```

```
44 x 1 sparse Matrix of class "dgCMatrix"
                                             1
(Intercept)                       -12.489880773
smokerS                             0.334308879
genderM                             0.084084609
activitylevelactive                -0.037782215
activitylevelaverage                .
activitylevelsedentary              0.225643698
positivefamilylongevityreported    -0.013894230
adls                                0.204832104
pulmonarymoderate                   .
pulmonarysevere                     0.603149491
                 :e                 .
                                    0.187029206
                                    0.786802366
                 :                  0.387246694
                                    0.803907097
                                    0.353810843
                                    .
                 le                -0.009278599
                                    .
                                    0.340900732
                                    0.755943914
                                    .
                                    0.213736906
                                    0.323687185
                 :d                 0.589970479
                                    0.020828503
                                    0.279585278
                                    1.512306594
```

```
cancer_prostatereported             .
cancer_breastreported               .
cancer_colonreported                .
cancer_pancreaticreported           .
cancer_lungreported                 .
cancer_hodgkinsreported             .
cancer_leukemiareported             .
cancer_myelomareported              .
cancer_liverreported                .
cancer_brainreported                .
cancer_ind                          0.023703394
```

```
ht.wt.flag                          .
height                              .
weight                              .
bmi                                 .
AttAge                              0.105461222
```

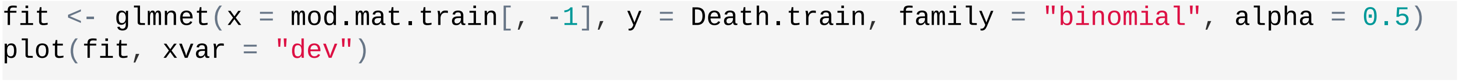
ElasticNet

- Convex combination of ridge and LASSO
- L2 & L1-norm penalties
- $loglikelihood =$

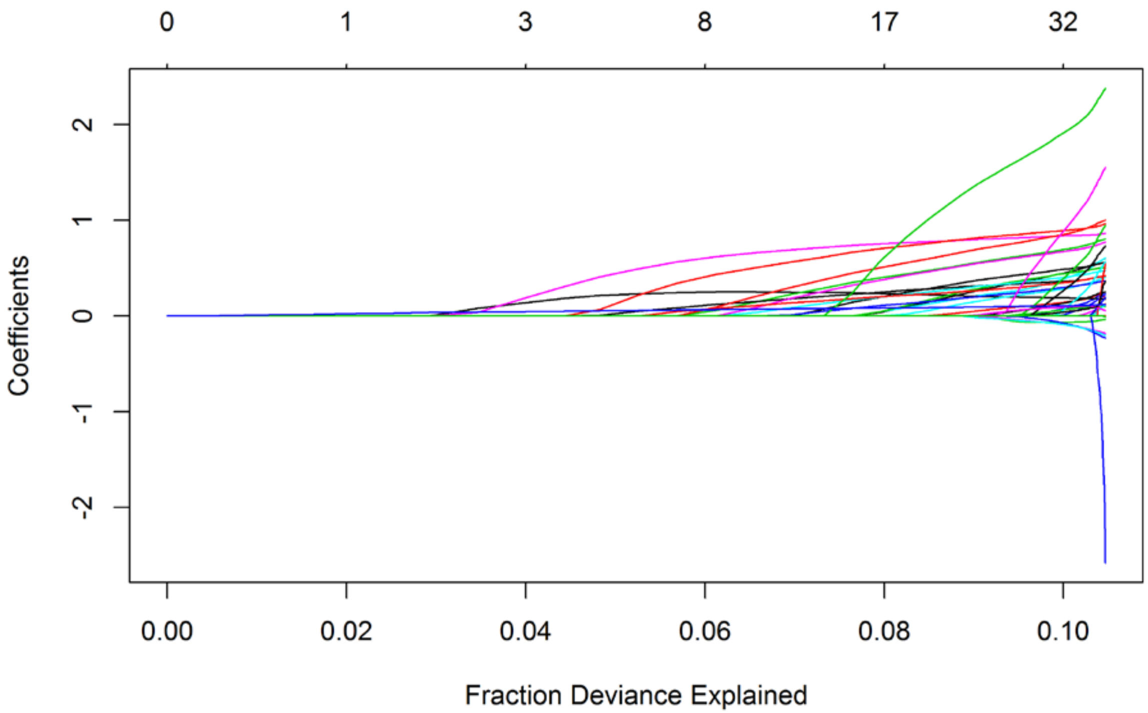
$$-\sum[Y_i \ln(\hat{y}_i) + (1 - Y_i)\ln(1 - \hat{y}_i)] + \boldsymbol{\lambda}\left((\mathbf{1} - \boldsymbol{\alpha})\sum \boldsymbol{\beta}^{\mathbf{2}} + \boldsymbol{\alpha}\sum |\boldsymbol{\beta}|\right)$$

Example (2A)

iii. ElasticNet



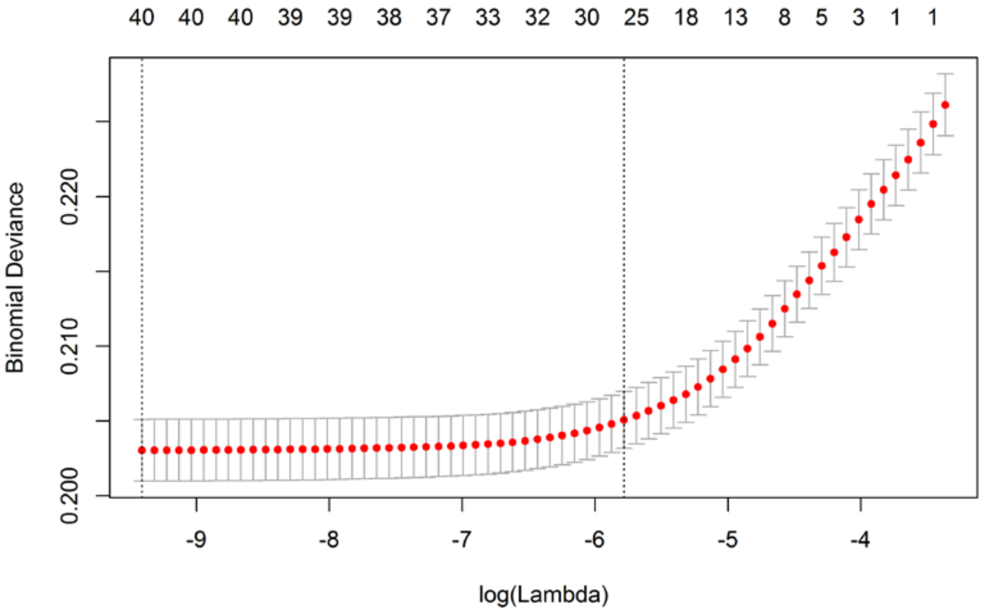
```
fit <- glmnet(x = mod.mat.train[, -1], y = Death.train, family = "binomial", alpha = 0.5)
plot(fit, xvar = "dev")
```



Example (2A)

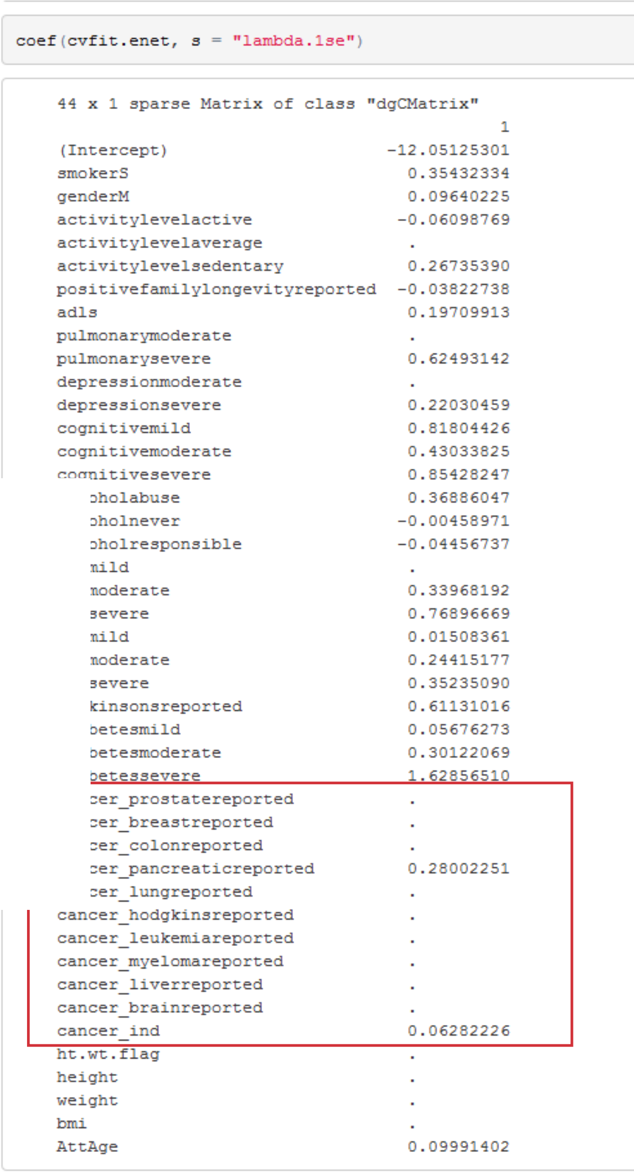
iii. ElasticNet

```
cl <- makeCluster(3)
registerDoParallel(cl)
cvfit.enet <- cv.glmnet(x = mod.mat.train[, -1], y = Death.train,
family = "binomial", alpha = 0.5, parallel = T)
stopCluster(cl)
plot(cvfit.enet)
```



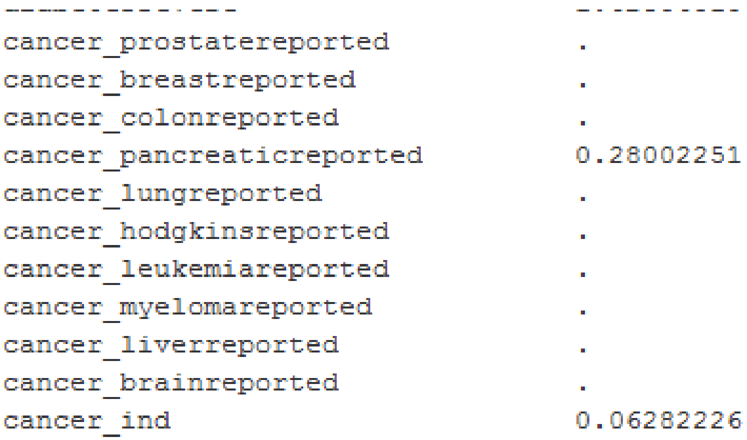
Example (2A)

iii. ElasticNet



```
coef(cvfit.enet, s = "lambda.1se")
```

```
44 x 1 sparse Matrix of class "dgCMatrix"
                                          1
(Intercept)                    -12.05125301
smokerS                          0.35432334
genderM                          0.09640225
activitylevelactive             -0.06098769
activitylevelaverage             .
activitylevelsedentary           0.26735390
positivefamilylongevityreported -0.03822738
adls                             0.19709913
pulmonarymoderate                .
pulmonarysevere                  0.62493142
depressionmoderate               .
depressionsevere                 0.22030459
cognitivemild                    0.81804426
cognitivemoderate                0.43033825
cognitivesevere                  0.85428247
   holabuse                      0.36886047
   holnever                     -0.00458971
   holresponsible               -0.04456737
   ild                           .
   oderate                       0.33968192
   evere                         0.76896669
   ild                           0.01508361
   oderate                       0.24415177
   evere                         0.35235090
   kinsonsreported               0.61131016
   etesmild                      0.05676273
   etesmoderate                  0.30122069
   etessevere                    1.62856510
   er_prostatereported           .
   er_breastreported             .
   er_colonreported              .
   er_pancreaticreported         0.28002251
   er_lungreported               .
cancer_hodgkinsreported          .
cancer_leukemiareported          .
cancer_myelomareported           .
cancer_liverreported             .
cancer_brainreported             .
cancer_ind                       0.06282226
ht.wt.flag                       .
height                           .
weight                           .
bmi                              .
AttAge                           0.09991402
```



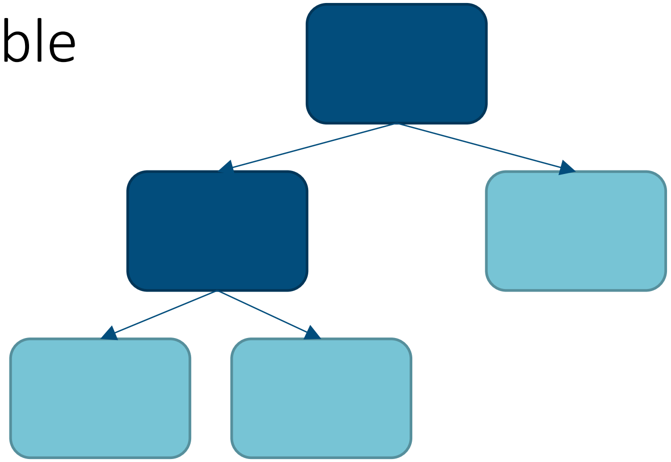
```
cancer_prostatereported          .
cancer_breastreported            .
cancer_colonreported             .
cancer_pancreaticreported        0.28002251
cancer_lungreported              .
cancer_hodgkinsreported          .
cancer_leukemiareported          .
cancer_myelomareported           .
cancer_liverreported             .
cancer_brainreported             .
cancer_ind                       0.06282226
```

Classification and Regression Trees (CART)

Trees

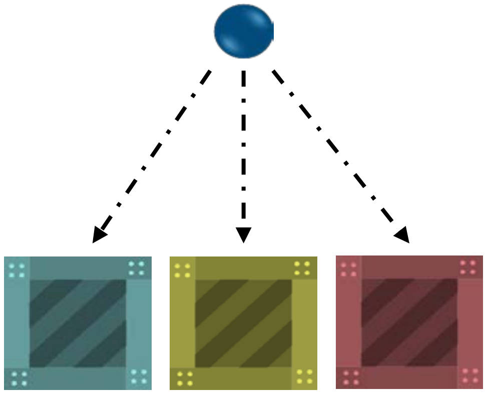
- Sequence of questions/rules for splitting the data
- Elements of CART algorithms
  - Rules for splitting data at each node
  - Stopping criteria
  - Prediction for the target variable

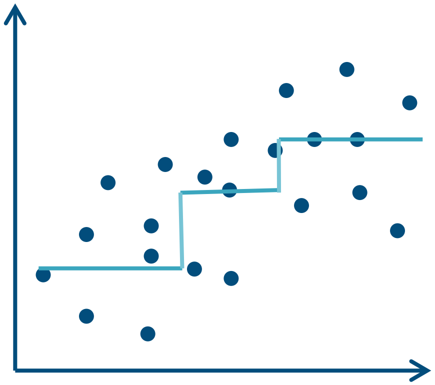
N = 350
0 = 200/350
1 = 150/350



Classification vs regression

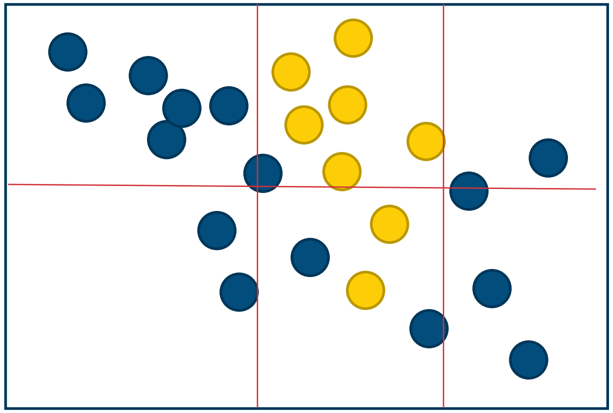
- Classification trees: used for categorical or binary target variables
  - Predict the category a policy will fall into
- Regression trees: continuous target variable
  - Predict the value of the continuous target





Splitting nodes

- Goal: choose the split that results in nodes with maximum homogeneity
- Classification: “Impurity” function
  - Entropy
  - Missclassification rate
  - Gini index
  - Twoing



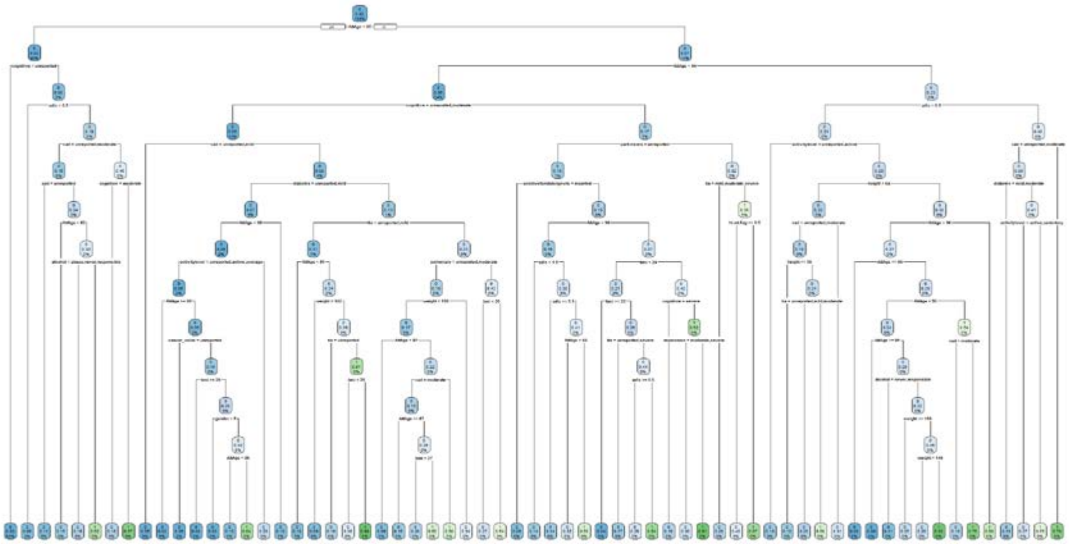
- Regression: Squared residuals minimization

Stopping rules

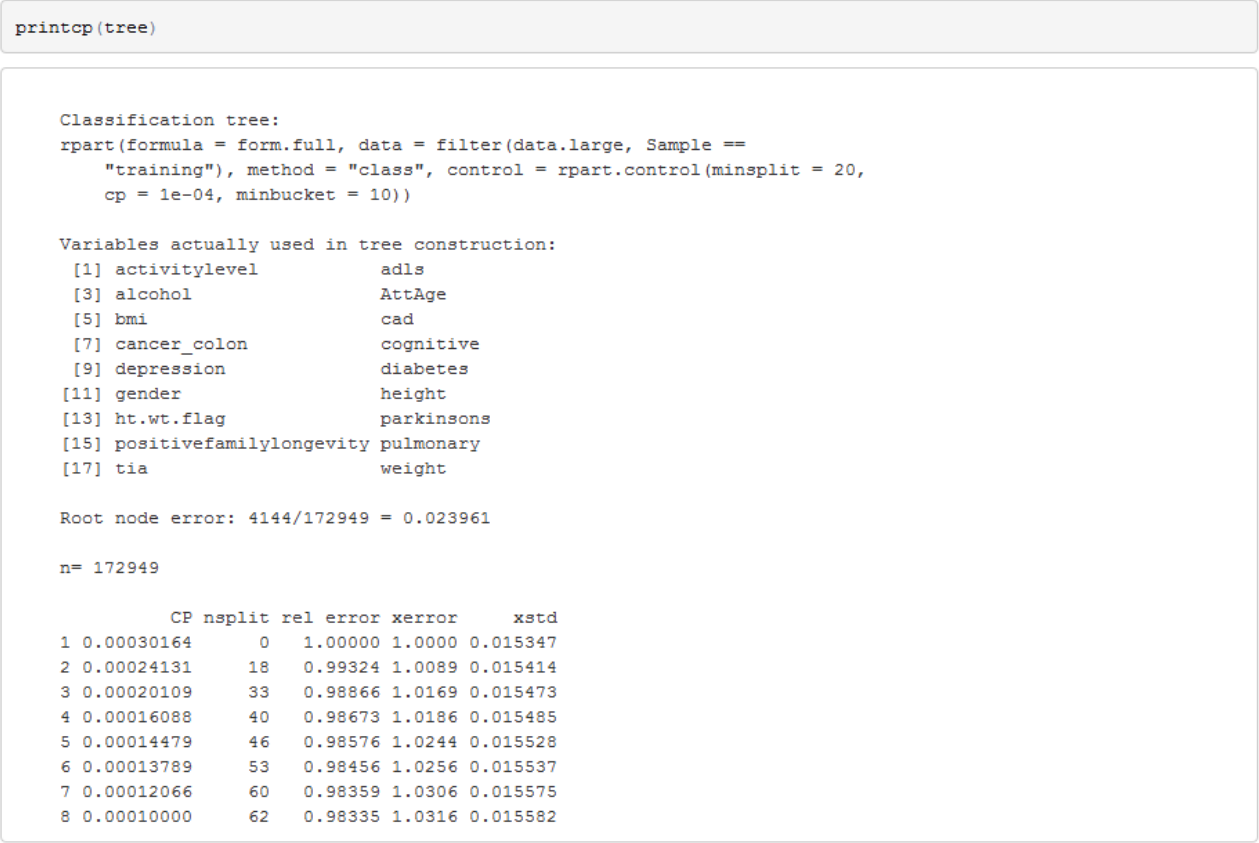
- Depth
- Size
- Number of nodes
- Complexity parameter

Example (2B)

```
tree <- rpart(formula = form.full, data = filter(data.large, Sample == "training"),
method = "class", control = rpart.control(minsplit = 20, cp = 1e-04, minbucket = 10))

rpart.plot(tree)
```



Example (2B)

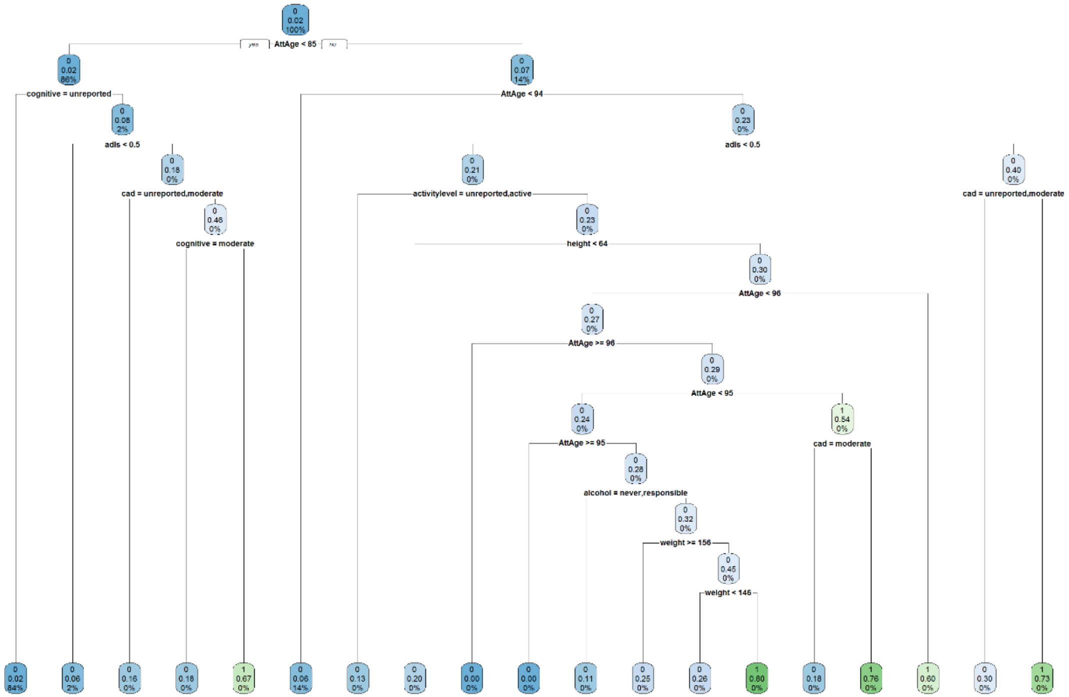


```
printcp(tree)
```

```
Classification tree:
rpart(formula = form.full, data = filter(data.large, Sample ==
    "training"), method = "class", control = rpart.control(minsplit = 20,
    cp = 1e-04, minbucket = 10))

Variables actually used in tree construction:
 [1] activitylevel             adls
 [3] alcohol                   AttAge
 [5] bmi                       cad
 [7] cancer_colon              cognitive
 [9] depression                diabetes
[11] gender                    height
[13] ht.wt.flag                parkinsons
[15] positivefamilylongevity pulmonary
[17] tia                       weight

Root node error: 4144/172949 = 0.023961

n= 172949

          CP nsplit rel error xerror     xstd
1 0.00030164      0   1.00000 1.0000 0.015347
2 0.00024131     18   0.99324 1.0089 0.015414
3 0.00020109     33   0.98866 1.0169 0.015473
4 0.00016088     40   0.98673 1.0186 0.015485
5 0.00014479     46   0.98576 1.0244 0.015528
6 0.00013789     53   0.98456 1.0256 0.015537
7 0.00012066     60   0.98359 1.0306 0.015575
8 0.00010000     62   0.98335 1.0316 0.015582
```

Example (2B)

```
fit <- prune(tree, cp = 0.00025)
rpart.plot(fit)
```



Ensemble Models

SOCIETY OF
ACTUARIES

Overview

- What:
  - An ensemble model is the aggregation of two or more related but different models, averaged into a single prediction.

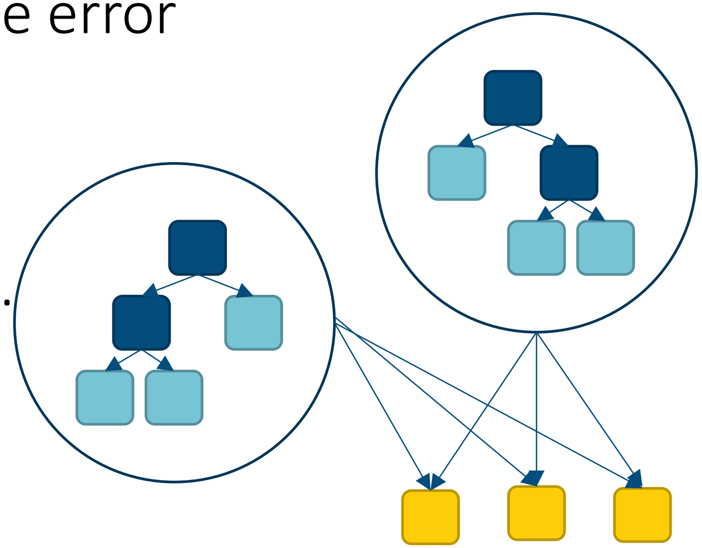
- Why:
  - Improve accuracy of predictions
  - Improve stability of the model

Ensemble methods

- Bagging
- Boosting
- Stacking

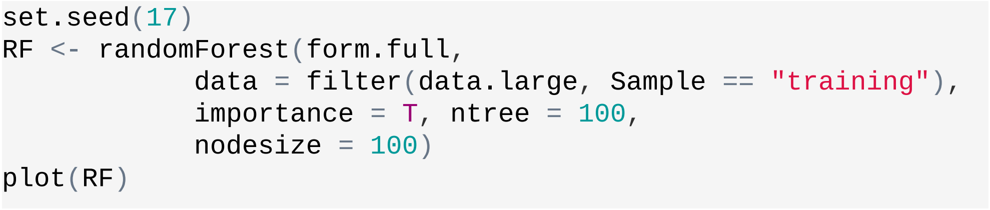
Bagging

- What is it:
  - Building multiple models from different subsamples of the training dataset, results are then combined for the final prediction.
  - Helps to reduce the variance error
- Example:
  - Random Forest
  - R package: **randomForest**, …

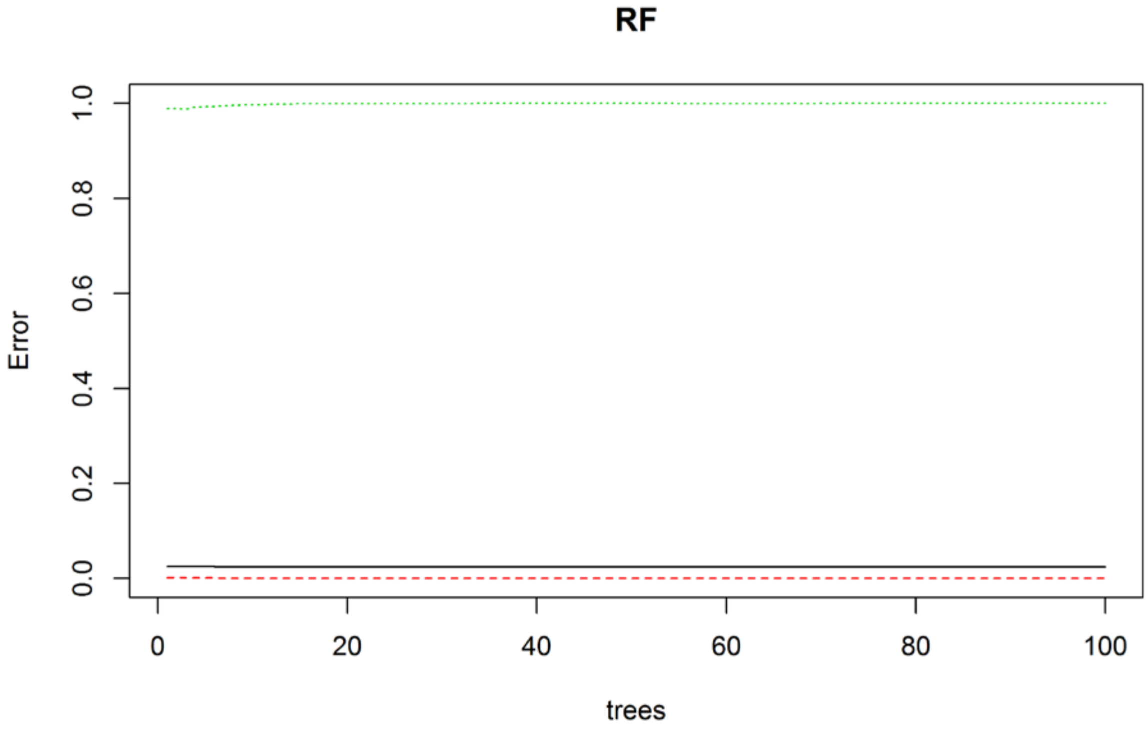


Example (2C)

i. Random forest



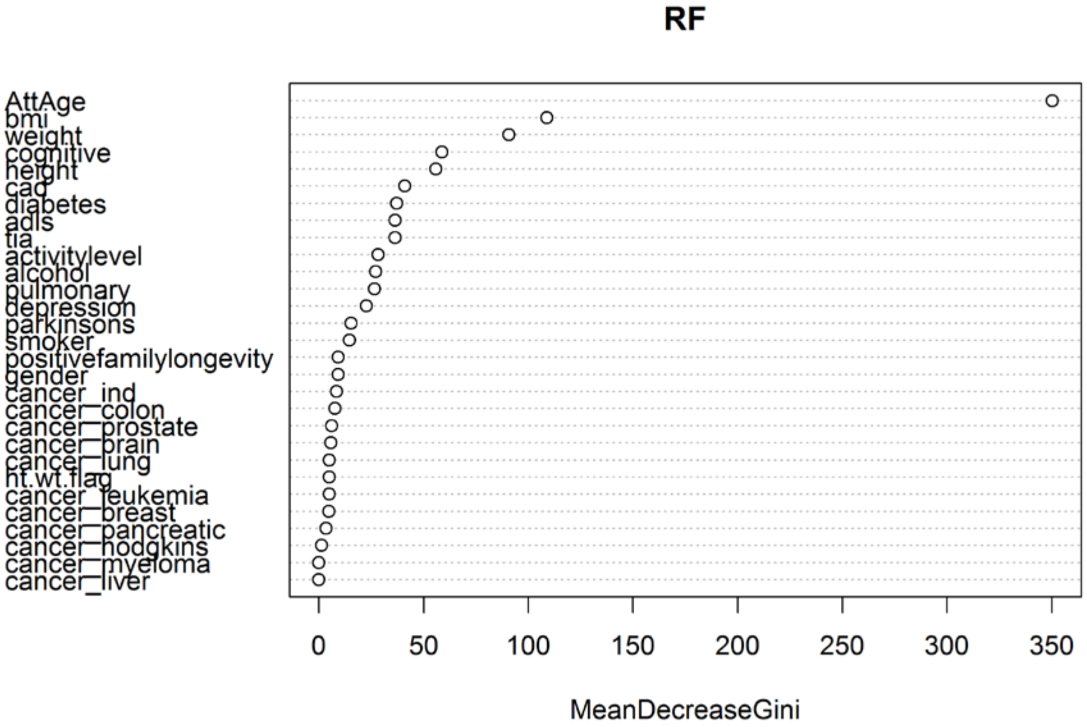
```r
set.seed(17)
RF <- randomForest(form.full,
            data = filter(data.large, Sample == "training"),
            importance = T, ntree = 100,
            nodesize = 100)
plot(RF)
```



Example (2C)

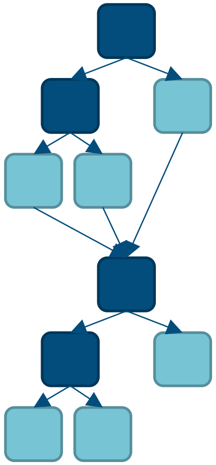
i. Random forest

```
varImpPlot(RF, type = 2)
```



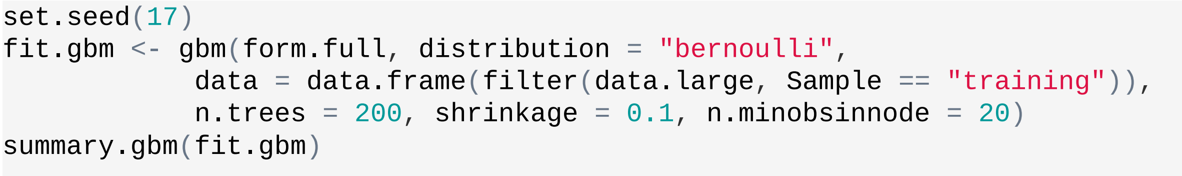
Boosting

- What is it:
  - Building multiple models, each of which is built to improve the prediction errors of a prior model
  - Has shown better predictive accuracy than bagging, but more likely to overfit
- Example:
  - Gradient Boosted Machines (GBM)
  - R packages: **gbm**, xgboost, …

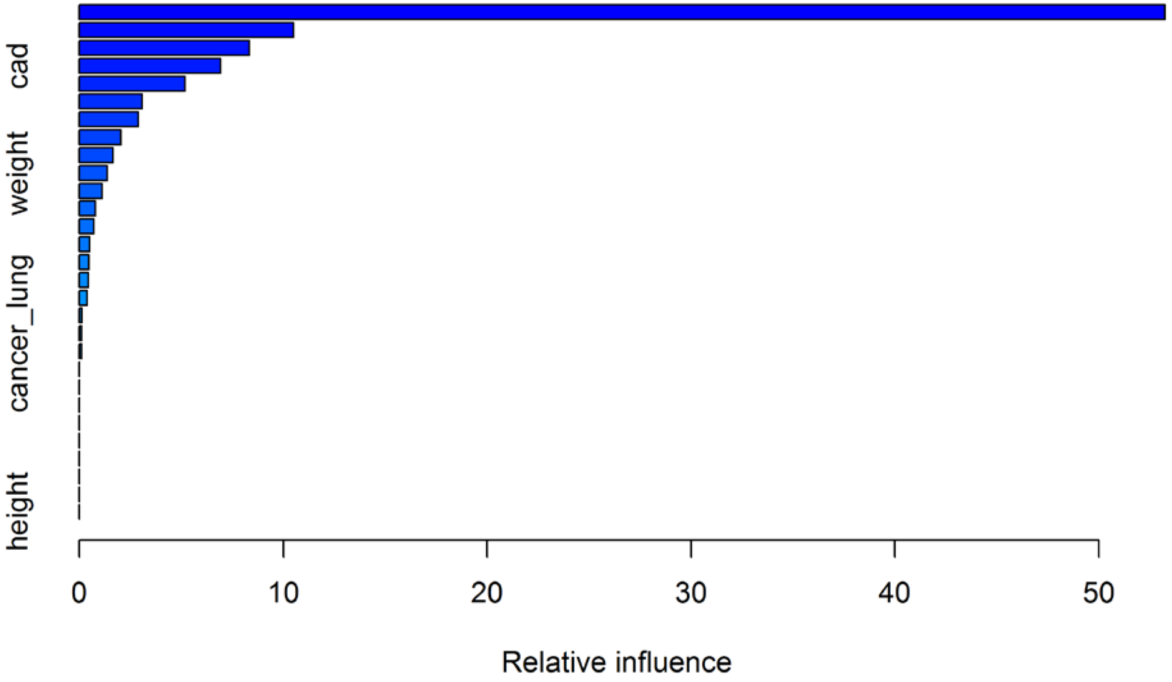


Example (2C)

ii. GBM

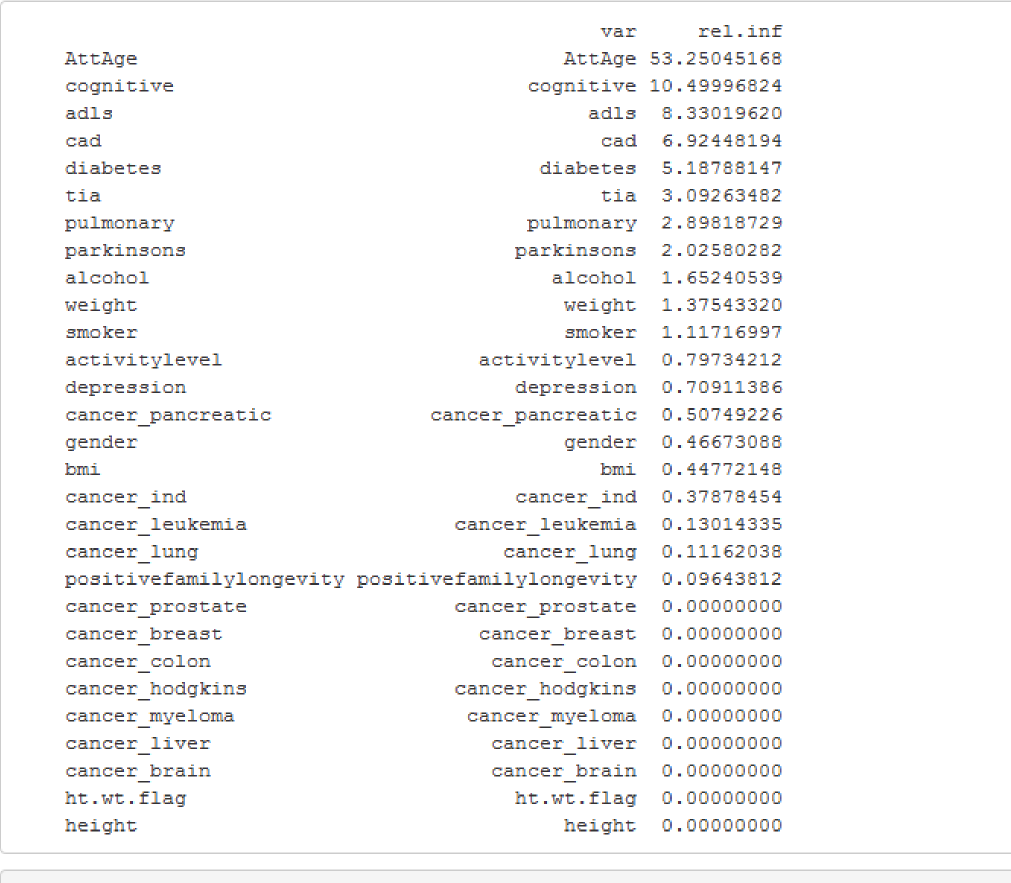


```
set.seed(17)
fit.gbm <- gbm(form.full, distribution = "bernoulli",
            data = data.frame(filter(data.large, Sample == "training")),
            n.trees = 200, shrinkage = 0.1, n.minobsinnode = 20)
summary.gbm(fit.gbm)
```



Example (2C)

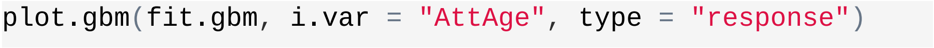
ii. GBM



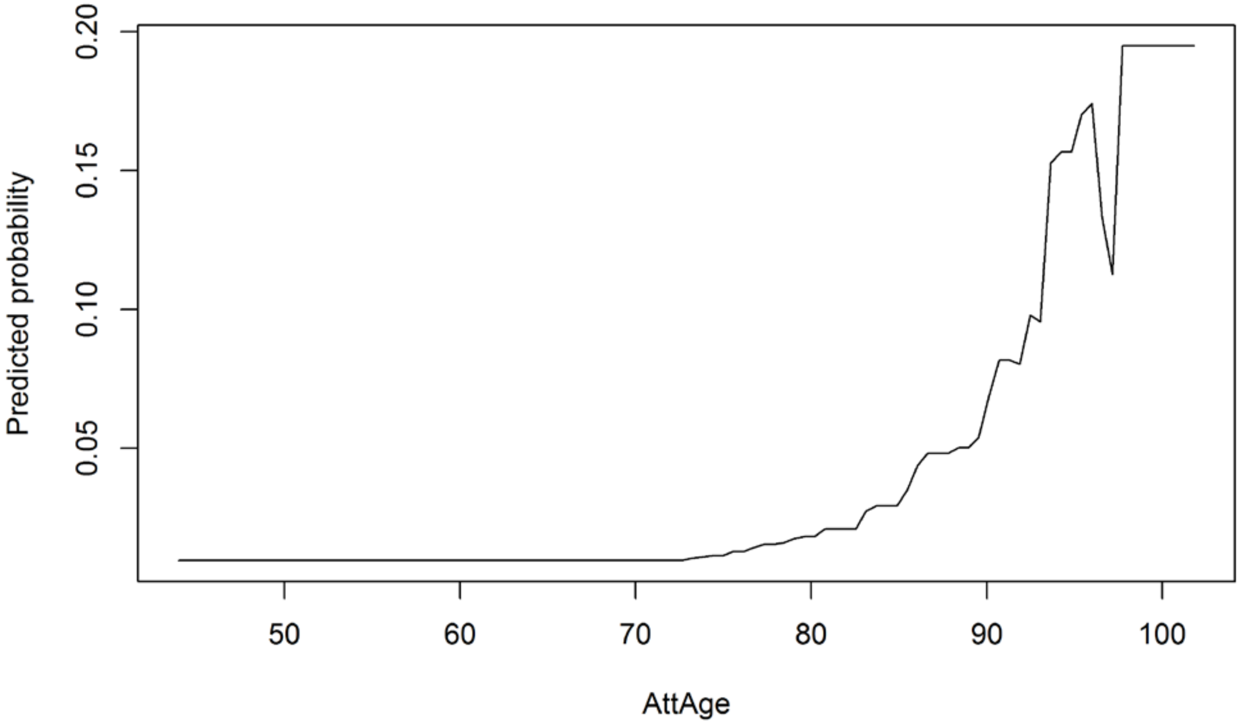
```
                                            var     rel.inf
AttAge                                   AttAge 53.25045168
cognitive                             cognitive 10.49996824
adls                                       adls  8.33019620
cad                                         cad  6.92448194
diabetes                               diabetes  5.18788147
tia                                         tia  3.09263482
pulmonary                             pulmonary  2.89818729
parkinsons                           parkinsons  2.02580282
alcohol                                 alcohol  1.65240539
weight                                   weight  1.37543320
smoker                                   smoker  1.11716997
activitylevel                     activitylevel  0.79734212
depression                           depression  0.70911386
cancer_pancreatic             cancer_pancreatic  0.50749226
gender                                   gender  0.46673088
bmi                                         bmi  0.44772148
cancer_ind                           cancer_ind  0.37878454
cancer_leukemia                 cancer_leukemia  0.13014335
cancer_lung                         cancer_lung  0.11162038
positivefamilylongevity positivefamilylongevity  0.09643812
cancer_prostate                 cancer_prostate  0.00000000
cancer_breast                     cancer_breast  0.00000000
cancer_colon                       cancer_colon  0.00000000
cancer_hodgkins                 cancer_hodgkins  0.00000000
cancer_myeloma                   cancer_myeloma  0.00000000
cancer_liver                       cancer_liver  0.00000000
cancer_brain                       cancer_brain  0.00000000
ht.wt.flag                           ht.wt.flag  0.00000000
height                                   height  0.00000000
```

Example (2C)

ii. GBM



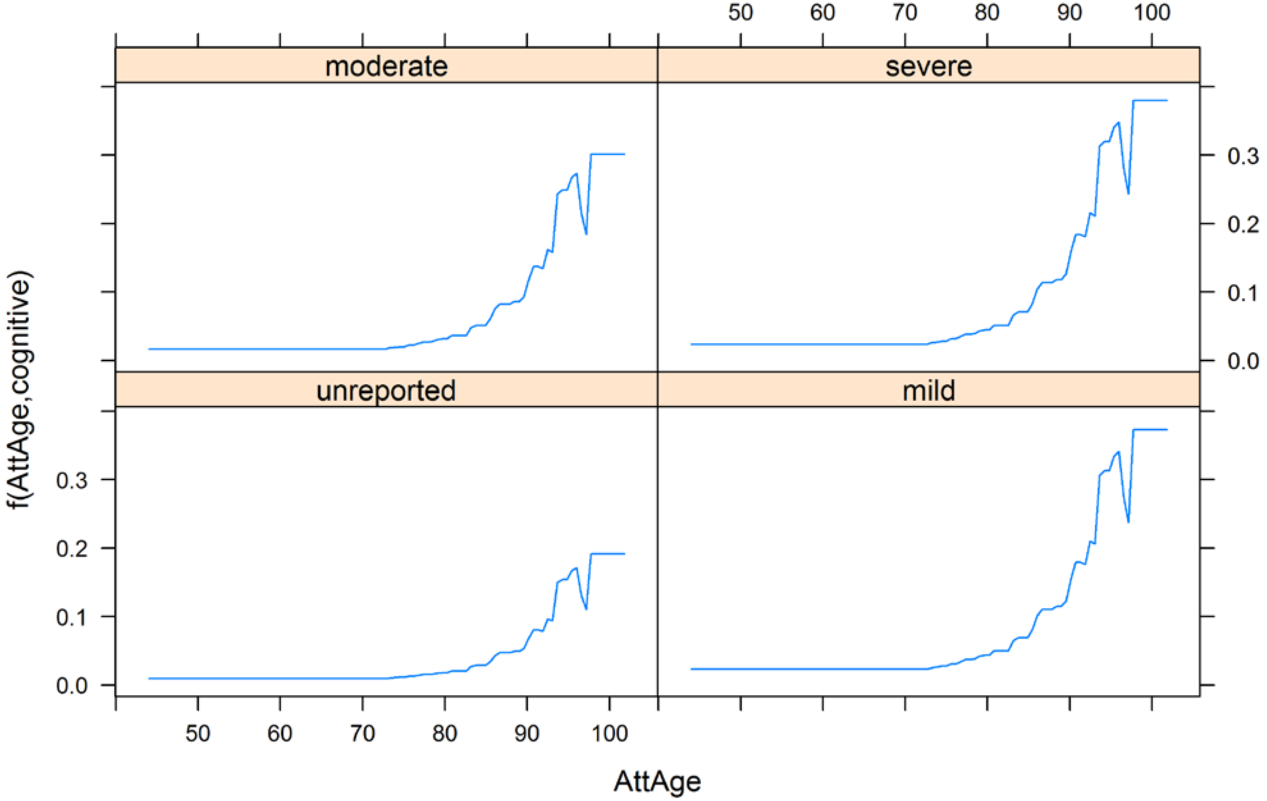
```
plot.gbm(fit.gbm, i.var = "AttAge", type = "response")
```



Example (2C)

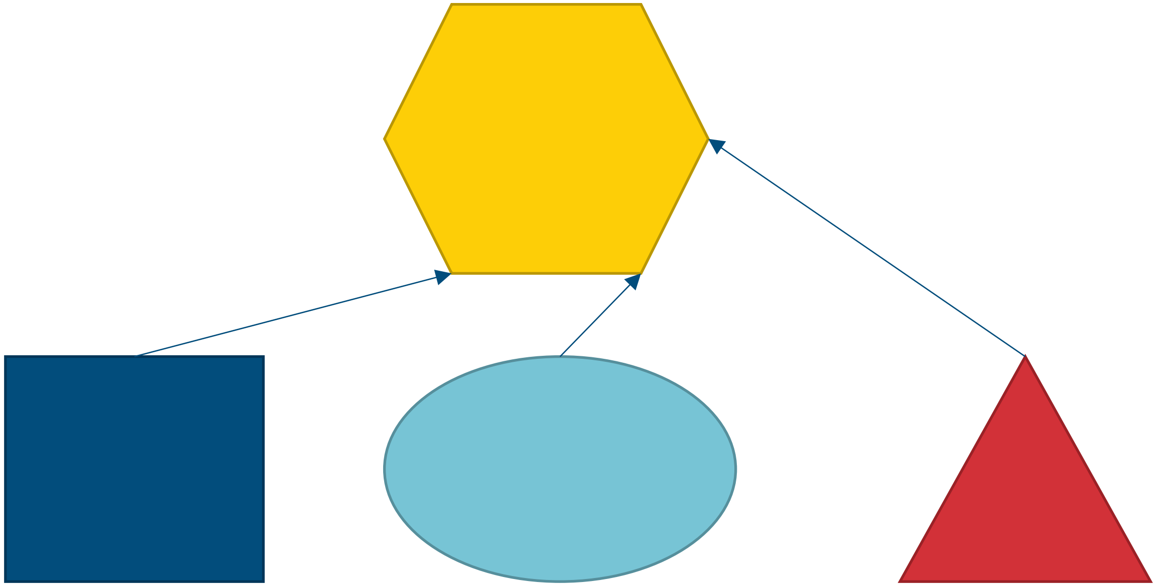
ii. GBM

```
plot.gbm(fit.gbm, i.var = c("AttAge", "cognitive"), type = "response")
```



Stacking

- What is it:
  - Building multiple models, typically different types of models, then having a supervisor model that determines how to best combine those results

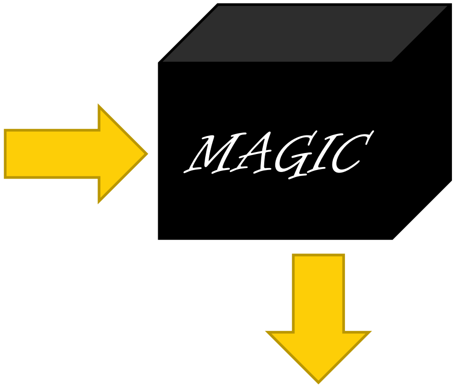


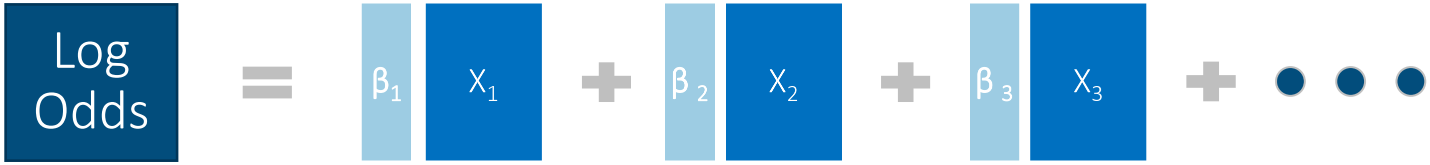
Final Thoughts

SOCIETY OF ACTUARIES

Weighing your options

- Implementation
- Explanation
- Cost





$$\text{Log Odds} = \beta_1 X_1 + \beta_2 X_2 + \beta_3 X_3 + \cdots$$

Other considerations

- Actuarial judgement
- Model selection
- Data issues
- Hardware/Software

MALE
FEMALE
NA

Now you know everything…

…that I could fit into 45 minutes!

Tools it’s a pity to miss

Git and similar bits

Brian D. Holland, FSA, MAAA
2017.05.10, Seattle, WA

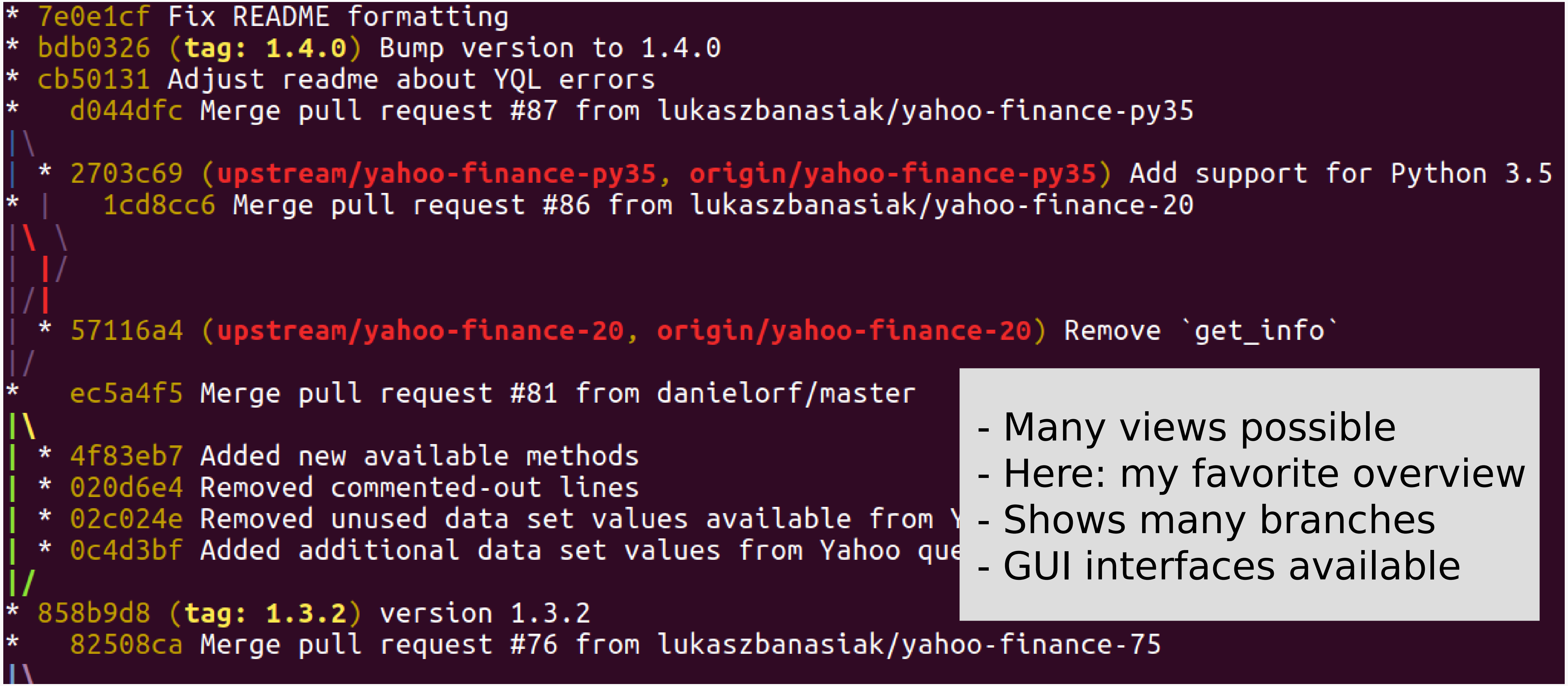
SVC

- *Source and Version Control*
- Why use an SVC system: **answer the W's**

  Who did **what**, **when**, and **why**

  You document this anyway (right?): the tool helps
- Software packages: discussion and list at... https://en.wikipedia.org/wiki/Version_control
- **Here: focus on *git***

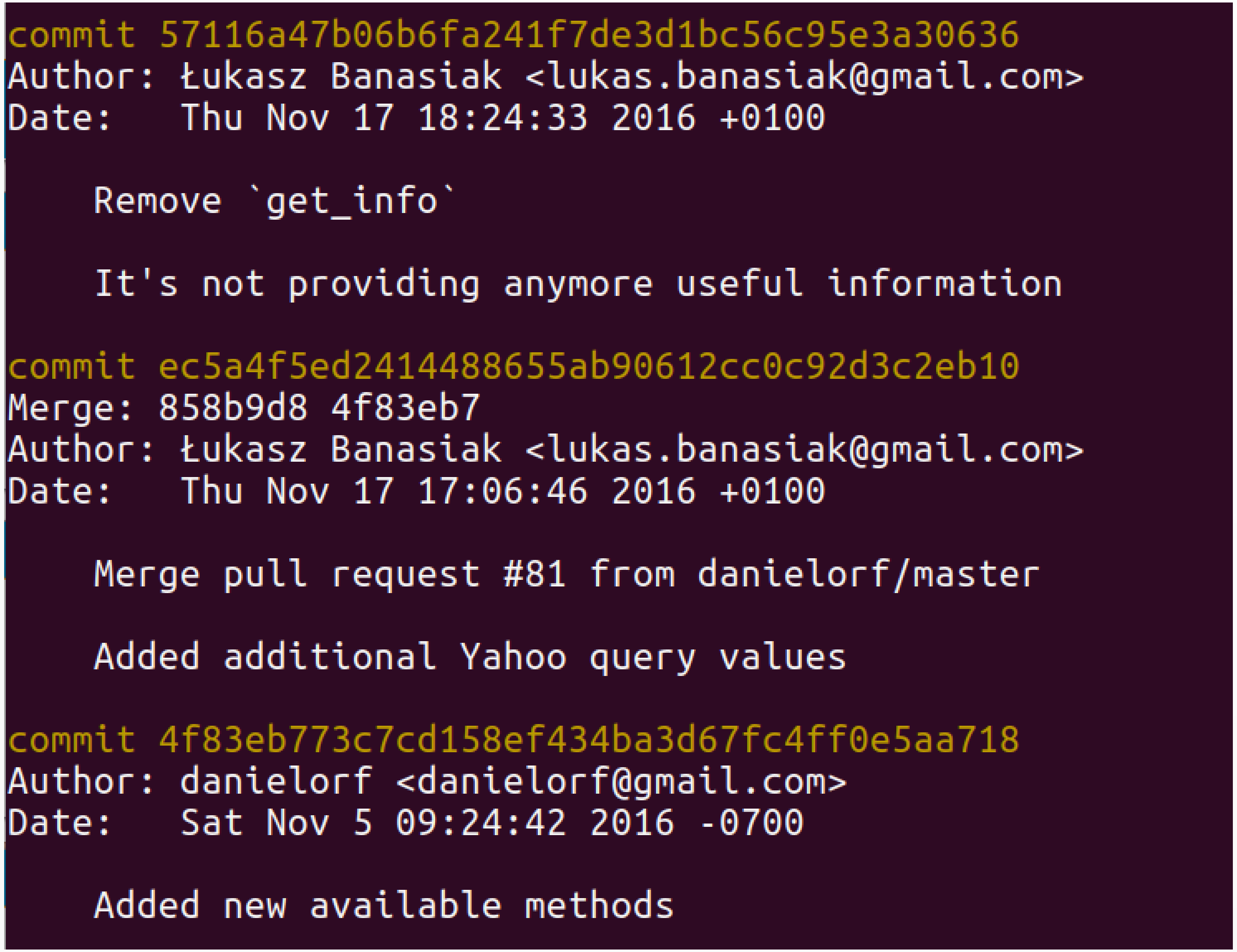
How does this look?



```
* 7e0e1cf Fix README formatting
* bdb0326 (tag: 1.4.0) Bump version to 1.4.0
* cb50131 Adjust readme about YQL errors
*   d044dfc Merge pull request #87 from lukaszbanasiak/yahoo-finance-py35
|\
| * 2703c69 (upstream/yahoo-finance-py35, origin/yahoo-finance-py35) Add support for Python 3.5
* |   1cd8cc6 Merge pull request #86 from lukaszbanasiak/yahoo-finance-20
|\ \
| |/
|/|
| * 57116a4 (upstream/yahoo-finance-20, origin/yahoo-finance-20) Remove `get_info`
|/
*   ec5a4f5 Merge pull request #81 from danielorf/master
|\
| * 4f83eb7 Added new available methods
| * 020d6e4 Removed commented-out lines
| * 02c024e Removed unused data set values available from Y
| * 0c4d3bf Added additional data set values from Yahoo que
|/
* 858b9d8 (tag: 1.3.2) version 1.3.2
*   82508ca Merge pull request #76 from lukaszbanasiak/yahoo-finance-75
|\
```

- Many views possible
- Here: my favorite overview
- Shows many branches
- GUI interfaces available

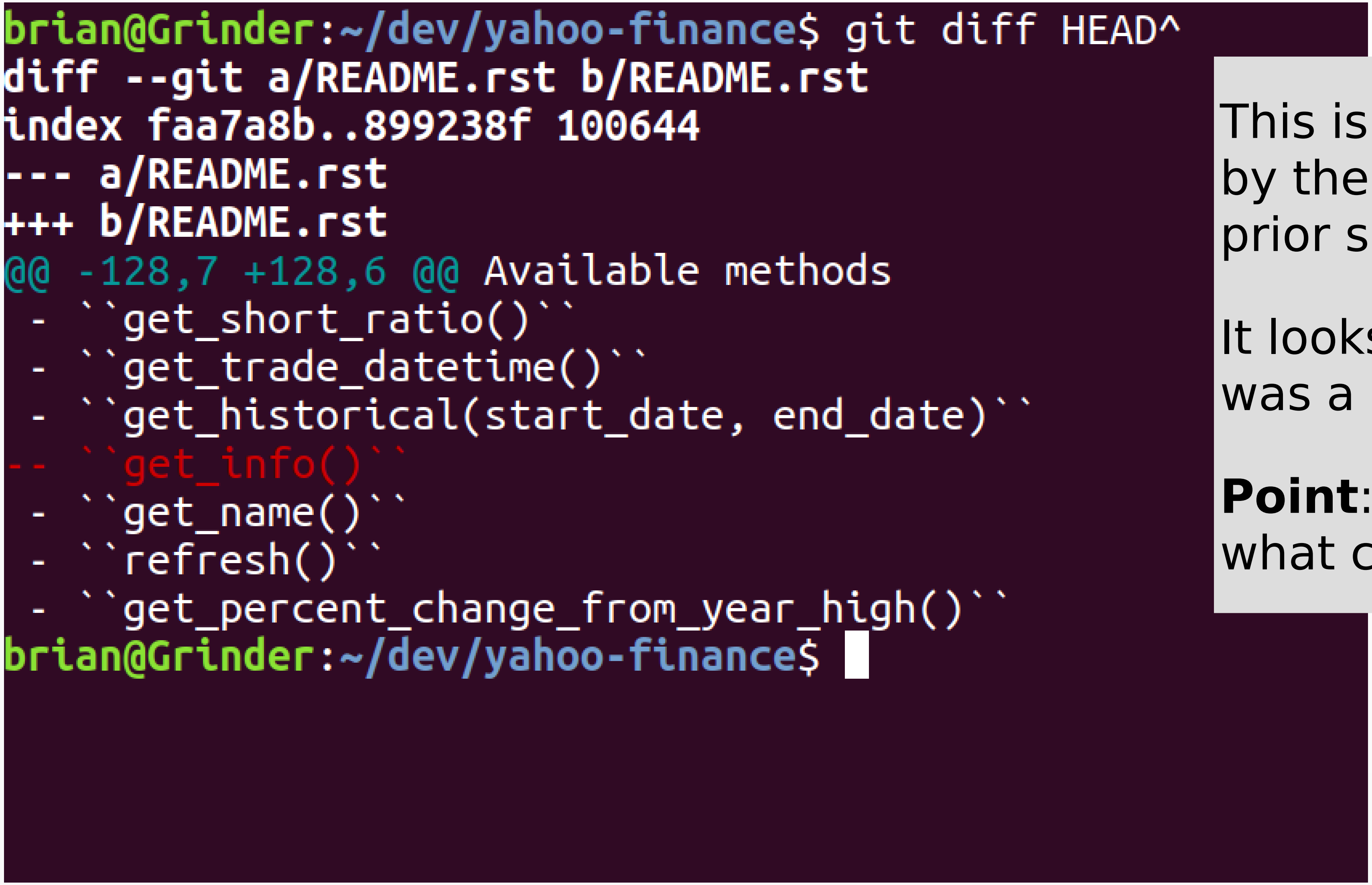
Log: more details



```
commit 57116a47b06b6fa241f7de3d1bc56c95e3a30636
Author: Łukasz Banasiak <lukas.banasiak@gmail.com>
Date:   Thu Nov 17 18:24:33 2016 +0100

    Remove `get_info`

    It's not providing anymore useful information

commit ec5a4f5ed2414488655ab90612cc0c92d3c2eb10
Merge: 858b9d8 4f83eb7
Author: Łukasz Banasiak <lukas.banasiak@gmail.com>
Date:   Thu Nov 17 17:06:46 2016 +0100

    Merge pull request #81 from danielorf/master

    Added additional Yahoo query values

commit 4f83eb773c7cd158ef434ba3d67fc4ff0e5aa718
Author: danielorf <danielorf@gmail.com>
Date:   Sat Nov 5 09:24:42 2016 -0700

    Added new available methods
```

- ID of the change
- Who did it
- When
- Why (comment)
- What: … next slide

What changed?



```
brian@Grinder:~/dev/yahoo-finance$ git diff HEAD^
diff --git a/README.rst b/README.rst
index faa7a8b..899238f 100644
--- a/README.rst
+++ b/README.rst
@@ -128,7 +128,6 @@ Available methods
 - ``get_short_ratio()``
 - ``get_trade_datetime()``
 - ``get_historical(start_date, end_date)``
-- ``get_info()``
 - ``get_name()``
 - ``refresh()``
 - ``get_percent_change_from_year_high()``
brian@Grinder:~/dev/yahoo-finance$ 
```

This is the change made by the top commit on the prior slide.

It looks like the comment was a good description.

Point: it’s easy to check what changed.

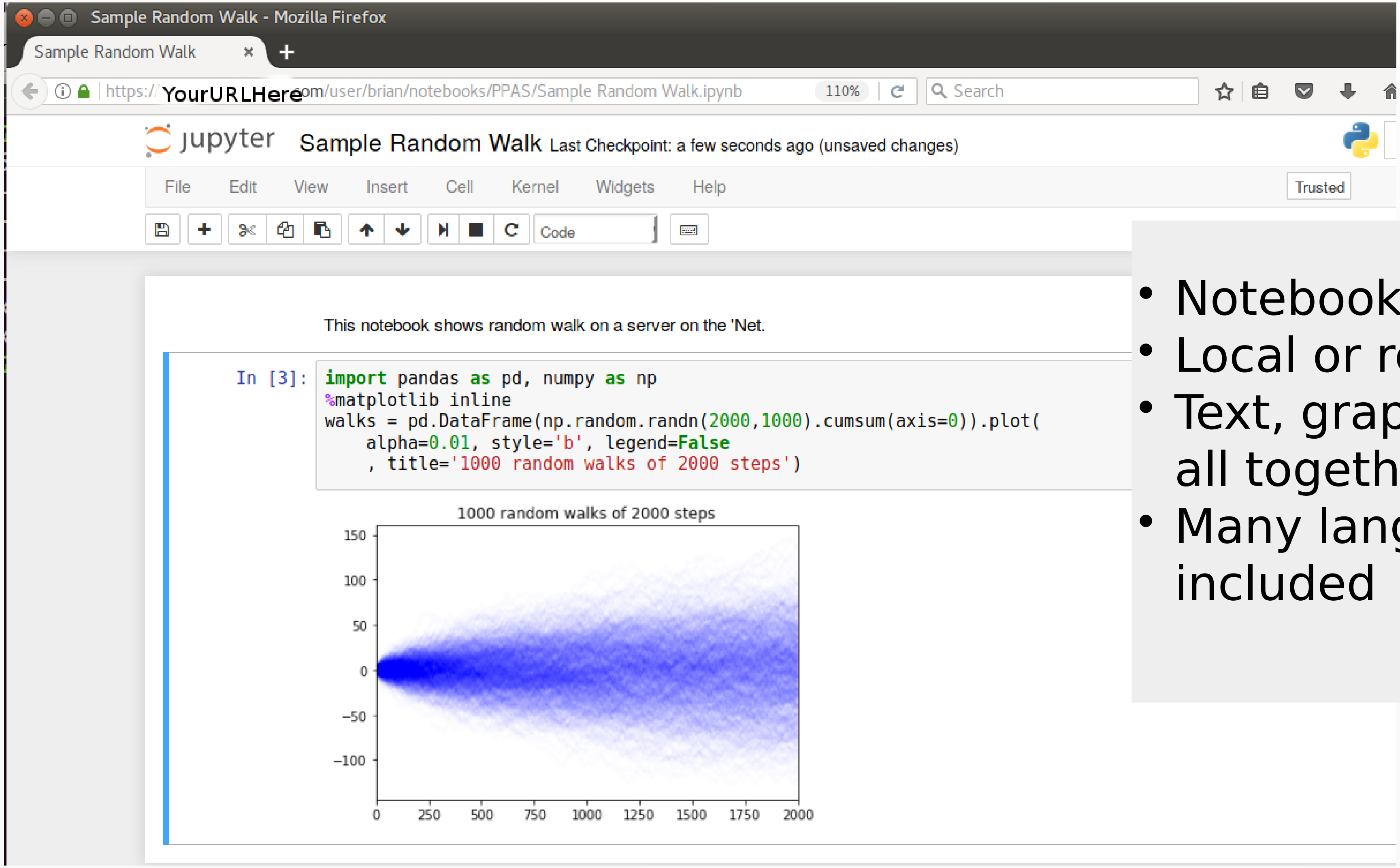
Social networking: github, gitlab, CI

- **Additional features**
- Pull requests
  - to ask others to use your work
- Issue flagging
  - including assigning issues with a due date
- Workflow in general
  - Marking when issues are complete

What do you do with this?

1. **Write** your work - text files for source code
2. **Commit** your changes – grouped as you like
   - You can "undo" to this point later.
   - You have to comment on the "commit point."
3. **Branch** off in a new direction and work on that
4. **Check out** a different branch, like “undoing” to another point
5. **Merge** other changes into yours.
6. **Push** the changes to a place that is shared with others.
7. **Pull** others' changes to your local work to synchronize.

Jupyter project


Sample Random Walk - Mozilla Firefox
Sample Random Walk
https://YourURLHere.com/user/brian/notebooks/PPAS/Sample Random Walk.ipynb
110%
Search
jupyter
Sample Random Walk Last Checkpoint: a few seconds ago (unsaved changes)
File
Edit
View
Insert
Cell
Kernel
Widgets
Help
Trusted
Code
1000 random walks of 2000 steps
150
100
50
0
−50
−100
0
250
500
750
1000
1250
1500
1750
2000

This notebook shows random walk on a server on the 'Net.

```
In [3]: import pandas as pd, numpy as np
        %matplotlib inline
        walks = pd.DataFrame(np.random.randn(2000,1000).cumsum(axis=0)).plot(
            alpha=0.01, style='b', legend=False
            , title='1000 random walks of 2000 steps')
```

- Notebook workspace
- Local or remote
- Text, graphs, computation all together
- Many languages: Python, R included

Cloud computing - topics

- **Which vendor? Some names:**

  AWS = Amazon Web Services

  Microsoft Azure

  Google, IBM also have offereings

  Rackspace, DigitalOcean

- **For what**

  Using a big server for a few hours

  Having a common workspace with others

  … many possible reasons: off-site backups, …

Where to learn more

- Git:

  https://git-scm.com/documentation

- Github:

  https://guides.github.com/activities/hello-world/

- Jupyter

  https://jupyter-notebook-beginner-guide.readthedocs.io/en/latest/

- AWS:

  http://docs.aws.amazon.com/gettingstarted/latest/awsgsg-intro/gsg-aws-intro.html

Have fun!

Thanks

Brian Holland, FSA, MAAA