Vignette for Recommended Modeling Strategies

Abraham Apfel

Thursday, April 25, 2024

Contents

1 Purpose

The purpose of this vignette is to be used as a guide for colleagues performing analyses involving regression modeling. Much of these strategies in this guide currently focus on the situation where the primary endpoint is binary, but can be generalized to continuous and survival outcomes as well. They are largely based off of Frank Harrell's book (Harrell Jr, 2015).

In this vignette we cover methods for imputation of missing data, flexible modeling techniques (e.g. cubic splines), proper visualizations, etc.

2 Dataset

We make use of data consisting of 1638 subjects from 9 randomized clinical trials for demonstration purposes. The goal of our analysis is to predict a binary outcome ("Outcome") from 19 clinical variables.

3 Multiple Imputation

3.1 Assessing the Extent of Missingness in your Data

The first step in any analysis is to assess the extent of missing data in your cohort. If there are any variables for which there is a very high proportion missing (e.g. $> 50\%$), it may be preferable to leave that variable out of the model since it will be very difficult for the imputation models to predict that variable well. One useful tool to get a high level overview of your data is the `describe` function from the `rms` package.

It may also be helpful to look at a correlation plot of the variables in your dataset. If some of your variables are highly correlated it will help advise you to include them in your imputation model even if they are not expected to be included in your actual analysis. This can be implemented via the `corrplot` function. We recommend for this step to include any columns of data you have access to even though you don't plan on using them in your model.

A tool to help visualize the patterns and extent of missingness in your data is the `naclus` and `naplot` functions. `naclus` creates a similarity matrix of the columns in your data according to the fraction of subjects with missing data. `naplot` offers useful visualizations.

```
# Get overview of dataset
describe(mod_dat)
```

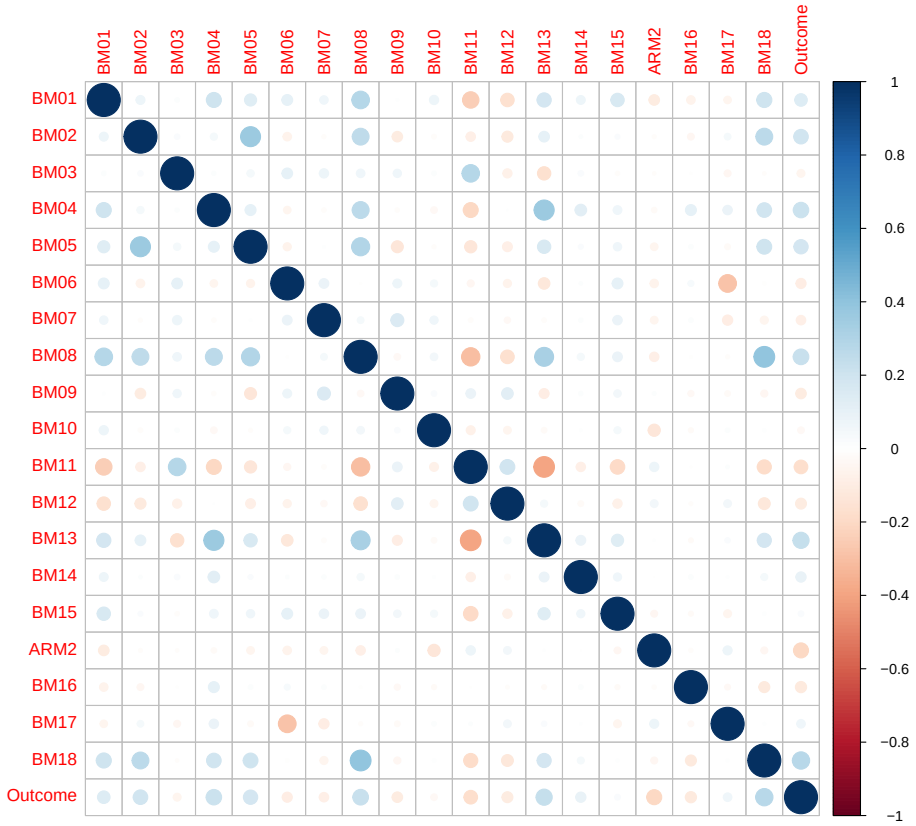
```
## mod_dat
##
##  20  Variables      1638  Observations
## ---------------------------------------------------------------------------
## BM01
##        n  missing distinct
##     1635        3        2
##
## Value          0   >=1
## Frequency   1166   469
## Proportion 0.713 0.287
## ---------------------------------------------------------------------------
## BM02
##        n  missing distinct     Info      Sum     Mean      Gmd
##     1638        0        2    0.665      544   0.3321   0.4439
##
## ---------------------------------------------------------------------------
## BM03
##        n  missing distinct
##     1638        0        2
##
## Value         NO   YES
## Frequency    565  1073
## Proportion 0.345 0.655
## ---------------------------------------------------------------------------
## BM04
##        n  missing distinct     Info     Mean      Gmd      .05      .10
##     1635        3     1051        1    4.959    2.571    2.304    2.700
##      .25      .50      .75      .90      .95
##    3.355    4.292    5.643    7.600    9.357
##
## lowest : 1.178  1.22   1.427  1.589  1.596 , highest: 30.031 33.079 34.167 36.774 51.504
## ---------------------------------------------------------------------------
```

```
## BM05
##        n  missing distinct
##     1551       87        4
##
## Value      Type A Type B Type C Type D
## Frequency      69    214    314    954
## Proportion  0.044  0.138  0.202  0.615
## ---------------------------------------------------------------------------
## BM06
##        n  missing distinct     Info     Mean      Gmd      .05      .10
##     1638        0       72    0.999    58.76    15.42       33       40
##      .25      .50      .75      .90      .95
##       50       60       69       75       79
##
## lowest : 18 19 21 22 23, highest: 86 87 88 89 90
## ---------------------------------------------------------------------------
## BM07
##        n  missing distinct
##     1638        0        3
##
## Value      Type A Type B Type C
## Frequency     841    617    180
## Proportion  0.513  0.377  0.110
## ---------------------------------------------------------------------------
## BM08
##        n  missing distinct     Info     Mean      Gmd      .05      .10
##     1638        0      267        1     74.7       62     13.0     17.0
##      .25      .50      .75      .90      .95
##     31.0     56.0    101.0    155.3    197.1
##
## lowest : 9     10    10.07 10.1  10.76, highest: 332   339   343   372   374
## ---------------------------------------------------------------------------
## BM09
##        n  missing distinct     Info     Mean      Gmd      .05      .10
##     1602       36      305        1    27.68    5.819     20.3     21.8
##      .25      .50      .75      .90      .95
##     23.9     27.0     30.6     34.6     37.0
##
## lowest : 12.6 15   16.5 17   17.1, highest: 51.1 53.4 54   54.4 56.5
## ---------------------------------------------------------------------------
## BM10
##        n  missing distinct     Info     Mean      Gmd      .05      .10
##     1441      197      394        1    163.1    211.3       19       25
##      .25      .50      .75      .90      .95
##       38       61      124      366      643
##
## lowest :    1    2    3    4    5, highest: 2747 3142 3243 3957 4364
## ---------------------------------------------------------------------------
## BM11
##        n  missing distinct     Info     Mean      Gmd      .05      .10
##     1636        2       97        1    133.8     19.7      101      109
##      .25      .50      .75      .90      .95
##      123      136      146      155      159
##
```

```
## lowest :  77  78  79  80  81, highest: 171 172 175 180 187
## ---------------------------------------------------------------------------
## BM12
##        n  missing distinct     Info     Mean      Gmd      .05      .10
##     1635        3      784        1    1.538   0.6978    0.670    0.830
##      .25      .50      .75      .90      .95
##    1.100    1.450    1.880    2.320    2.685
##
## lowest : 0.16  0.19  0.272 0.275 0.3  , highest: 4.77  4.896 5.612 5.98  7.3
## ---------------------------------------------------------------------------
## BM13
##        n  missing distinct     Info     Mean      Gmd      .05      .10
##     1635        3      388        1    269.3    102.7    150.7    171.0
##      .25      .50      .75      .90      .95
##    203.0    250.0    313.0    386.6    462.9
##
## lowest :   77   90  100  105  106, highest:  708  765  776  831 1056
## ---------------------------------------------------------------------------
## BM14
##        n  missing distinct     Info      Sum     Mean      Gmd
##     1638        0        2     0.18      105   0.0641   0.1201
##
## ---------------------------------------------------------------------------
## BM15
##        n  missing distinct     Info      Sum     Mean      Gmd
##     1638        0        2    0.498      344     0.21    0.332
##
## ---------------------------------------------------------------------------
## ARM2
##        n  missing distinct
##     1638        0        4
##
## Value      ARM A ARM B ARM C ARM D
## Frequency    292   206   521   619
## Proportion 0.178 0.126 0.318 0.378
## ---------------------------------------------------------------------------
## BM16
##        n  missing distinct     Info     Mean      Gmd      .05      .10
##     1381      257       35    0.944    10.27    16.26        0        0
##      .25      .50      .75      .90      .95
##        0        1        7       35       60
##
## lowest :   0   1   2   3   4, highest:  90  95  98  99 100
## ---------------------------------------------------------------------------
## BM17
##        n  missing distinct
##     1521      117        2
##
## Value         NO   YES
## Frequency   1006   515
## Proportion 0.661 0.339
## ---------------------------------------------------------------------------
## BM18
##        n  missing distinct     Info     Mean      Gmd      .05      .10
```

```
##     1621       17      541        1    2.425   0.2775    2.140    2.176
##      .25      .50      .75      .90      .95
##    2.241    2.346    2.559    2.762    2.962
##
## lowest : 1.88081 1.90309 1.9345  1.93952 1.97313
## highest: 3.46285 3.50365 3.51388 3.71332 3.76856
## ---------------------------------------------------------------------------
## Outcome
##        n  missing distinct
##     1638        0        2
##
## Value          N     Y
## Frequency   1053   585
## Proportion 0.643 0.357
## ---------------------------------------------------------------------------
```

```r
# We first turn all variables into numeric just for the purpose of this correlation plot
mod_num <- sapply(mod_dat, as.numeric)


# Make correlation matrix to inform on imputation model
cor_mat <- rcorr(mod_num)

# Plot the correlation matrix
corrplot(cor_mat$r)
```



3.2 Prepare for Multiple Imputation

In general, we recommend using multiple imputation to impute the missing data rather than the more commonly (and simpler) applied approach of performing a complete case analysis. One reason for this is that in a complete case analysis any subject with any missing data will be thrown out of the model. Thus it is a very inefficient approach to modeling. Additionally, a complete case analysis assumes the data is Missing Completely at Random (MCAR), a very strong assumption (assumes that a data point which is missing is completely unrelated to any characteristic of the subject) whereas an imputation model only assumes the data was Missing at Random (MAR), a much weaker assumption (assumes that a data point which is missing may depend on the values of variables which are measured).

We also don't recommend imputing the mean/median of a given variable because that will artificially deflate the relationship between that variable and response and will thus yield a very conservative estimate of the true relationship.

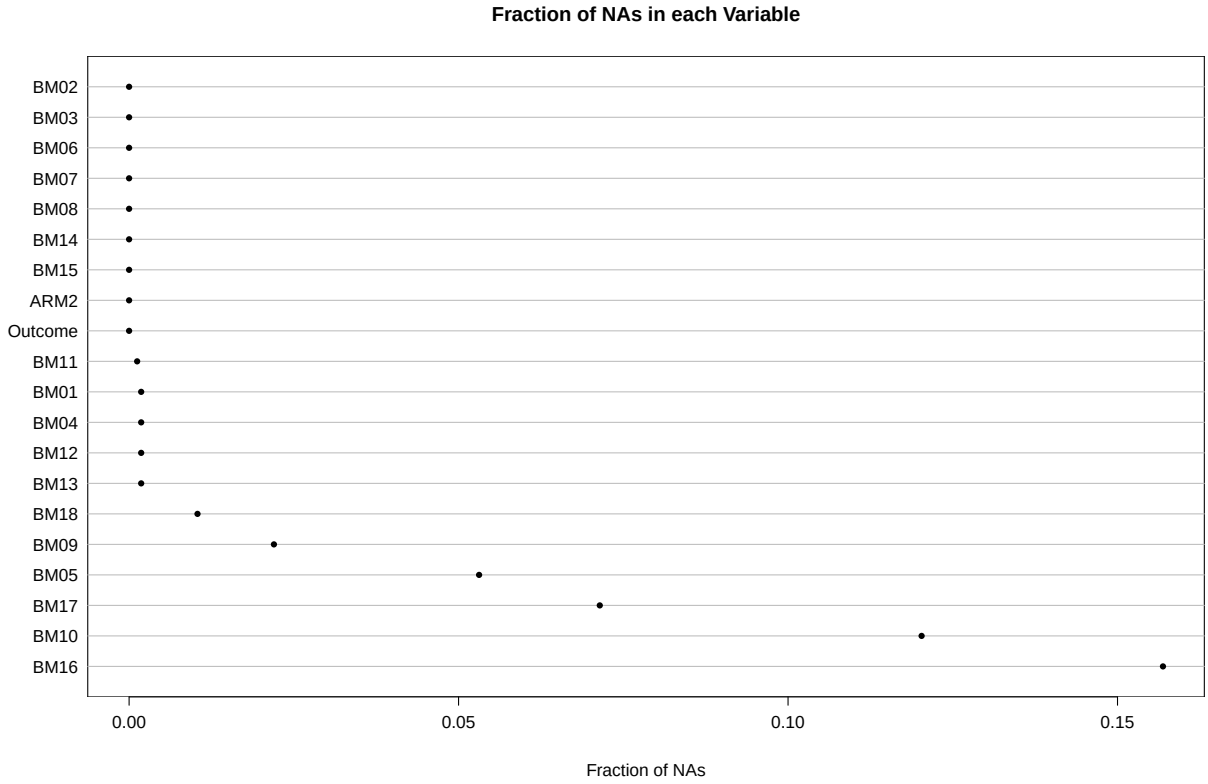
A much preferred approach is to impute the missing values by building a model predicting the missing value based on all of the other variables. In particular, we recommend the Predictive Mean Matching method via Chained equations (see Chapter 3 in (Harrell Jr, 2015)) for more details. This is the default method applied by the `aregimpute` function we demonstrate below (this is as opposed to the popular `MICE` package which by default uses regular regression which is more prone to extreme imputations and less robust).

We also recommend strongly to use multiple imputations instead of relying on a single imputation of the missing value. The reason for this is because a single imputation will not capture the uncertainty that exists in the imputation of the missing value itself, which will yield underestimated standard errors in the model outputs and overly optimistic associations between the imputed variable and the outcome (biased coefficients). By applying multiple imputations we can capture that uncertainty and yield unbiased estimates of the coefficients as well.

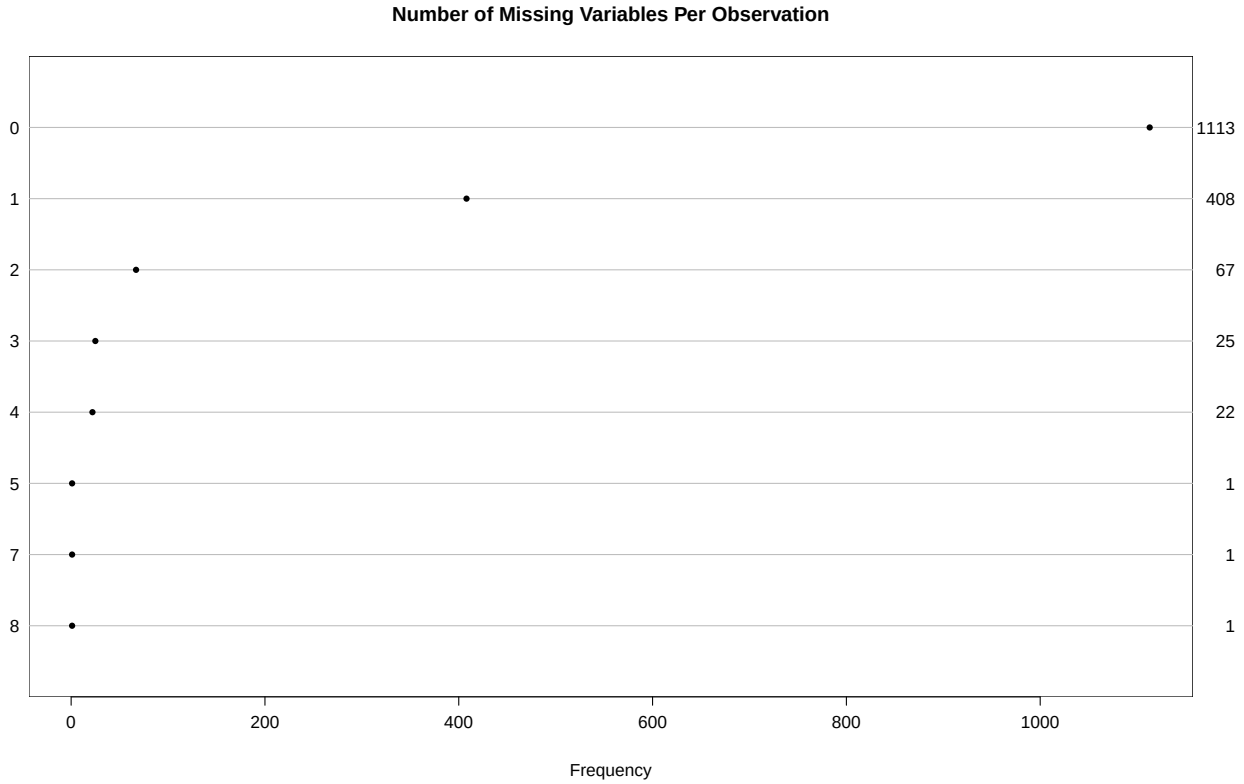
The question then arises, how many imputations are appropriate? As a general rule of thumb, we recommend one imputation for every % of the data that has any missing values. An easy way to visualize this can be accomplished via the `naplot` functions. We present some useful visualizations below.

It is important to remember for the purpose of this visualization, to first remove from your initial dataset any columns that you do not plan on including in your model. This is because our estimate of the number of imputations is based on the fraction of subjects with missing data with which will be included in the model.

```
# Create dataset containing only variables which will be used in modeling

clin_cov <- c("BM01", "BM02", "BM03", "BM04", "BM05", "BM06", "BM07", "BM08", "BM09",
              "BM10", "BM11", "BM12", "BM13", "BM14", "BM15", "ARM2")
BM <- c("BM16", "BM17", "BM18")

# View the % missing for each variable in your data
na.patterns <- naclus(mod_dat)
naplot(na.patterns, 'na per var')
```



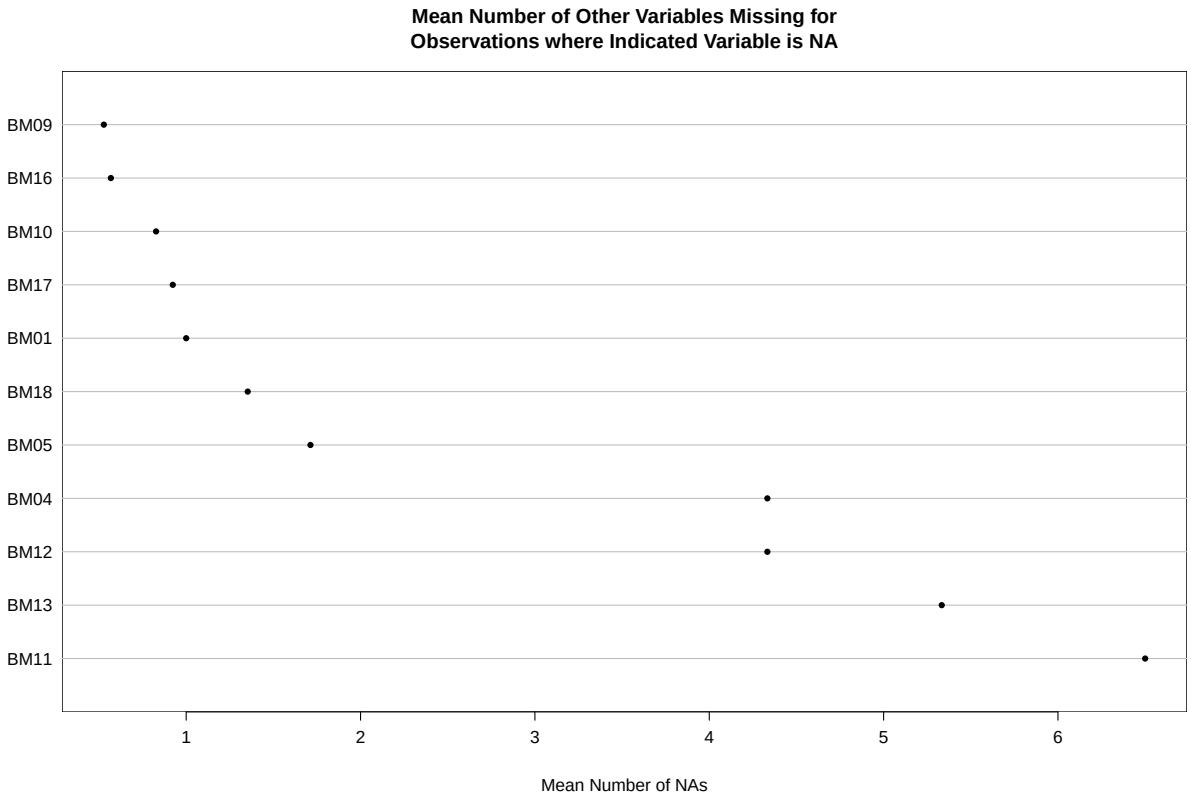

```
# Estimate number of imputations is necessary - 32% of data has at least one column missing.
naplot(na.patterns, 'na per obs')
```




```
# Calculate the fraction of subjects with at least 1 missing column ~ 32%
(1638 - 1113)/1638
```

```
## [1] 0.3205128
```

```
# Another, sometimes useful, visualization
naplot(na.patterns, "mean na")
```



3.3 Imputation Model

The next step is to run the multiple imputation model itself. This can be implemented via the `aregimpute` function from the `rms` package.

One must include in the imputation model every covariate which one plans to include in the actual analysis, in addition to any other variables you think can be helpful for predicting any covariate with missing data.

To make the function allow for nonlinearity when trying to predict missing variables, one should set `tlinear = F`. This is one advantage of the `aregimpute` function over the popular `MICE` package, which does not have this capability and forces you to assume linear relationships amongst the variables. Another advantage is that the default method applied for the imputations is Predictive Mean Matching as opposed to the `MICE` package which by default uses regular regression which is more prone to extreme imputations and less robust.

One VERY important detail (although perhaps counterintuitive) is that it is important to include the outcome variable in the imputation model. Often the outcome has the strongest ability amongst the covariates in predicting the missing variables. This can only be done when performing multiple imputation (as opposed to single imputation) because the imputation process will account for the uncertainty that exists in the assumed relationship between the predictor and outcome such that there will be no deflation of the estimated standard errors as a result of using this relationship in imputing the missing covariates which will then in turn be used to estimate the relationship between the predictors and outcome. The details justifying this claim can be found in Chapter 3 in (Moons et al., 2006).

For illustration purposes we perform only 5 imputations, although according to the above rule of thumb we should really be making 32 imputations. It is important to remember to set a seed as each imputation is based on bootstrapping sample of your data. It can also be somewhat time consuming so we recommend saving your imputation object as a .rds object to save time in the future.

The imputation object will consist of 5 imputed datasets, one for each imputation. We will use this object for our models in the analysis. The coefficients from our models will reflect the average coefficient across each of the 5 imputed datasets. The standard errors are the sum of the standard error within each dataset and across the datasets.

We can get a sense of the predictive accuracy of the imputations by looking at the R2 in the summary of the results from our imputation below.

```
set.seed(4234)

if(!file.exists(file.path(results, "MI5.rds"))) {
MI5 <- aregImpute( ~ BM16 + BM17  + Outcome + BM09 + BM05 + BM18 + BM06 + BM07 + BM14 +
                     BM08 + BM01 + BM11 + BM13 + BM12 + BM04 + BM02 + ARM2 + BM03 + BM10 +
                     BM15,
                   data = mod_dat, n.impute = 5, tlinear = F)

saveRDS(MI5, file.path(results, "MI5.rds"))
} else {
  MI5 <- readRDS(file.path(results, "MI5.rds"))
}

print(MI5)
```

```
##
## Multiple Imputation using Bootstrap and PMM
##
## aregImpute(formula = ~BM16 + BM17 + Outcome + BM09 + BM05 + BM18 +
##     BM06 + BM07 + BM14 + BM08 + BM01 + BM11 + BM13 + BM12 + BM04 +
##     BM02 + ARM2 + BM03 + BM10 + BM15, data = mod_dat, n.impute = 5,
##     tlinear = F)
##
## n: 1638  p: 20   Imputations: 5      nk: 3
##
## Number of NAs:
##    BM16    BM17 Outcome    BM09    BM05    BM18    BM06    BM07    BM14    BM08
##     257     117       0      36      87      17       0       0       0       0
##    BM01    BM11    BM13    BM12    BM04    BM02    ARM2    BM03    BM10    BM15
##       3       2       3       3       3       0       0       0     197       0
##
##         type d.f.
## BM16       s    2
## BM17       c    1
## Outcome    c    1
## BM09       s    2
## BM05       c    3
## BM18       s    2
## BM06       s    2
## BM07       c    2
## BM14       l    1
## BM08       s    2
## BM01       c    1
## BM11       s    2
## BM13       s    2
## BM12       s    2
## BM04       s    2
```

```
## BM02       l    1
## ARM2       c    3
## BM03       c    1
## BM10       s    2
## BM15       l    1
##
## R-squares for Predicting Non-Missing Values for Each Variable
## Using Last Imputations of Predictors
##  BM16  BM17  BM09  BM05  BM18  BM01  BM11  BM13  BM12  BM04  BM10
## 0.110 0.155 0.138 0.298 0.299 0.234 0.337 0.395 0.223 0.294 0.212
```

4 Redundancy Analysis

We now shift our focus to explore if there is a way to filter out predictors which can be easily predicted from the other predictors. It is important to bear in mind that although looking at an apparent relationship between the outcome and predictors prior to running your model can lead to overfitting and should generally (with few exceptions: one of which we have already discussed in the context of multiple imputation, another of which we will discuss below) not be done, there is no concern of overfitting by looking at the relationships amongst the predictors prior to modeling.

We suggest the following algorithm to remove these "redundant" predictors:

1. Predict each predictor from all other predictors
2. Remove the predictor that can be predicted with the highest R2
3. Predict each remaining predictor from their complement
4. Continue until no other predictor can be predicted with R2 greater than a specified threshold OR until dropping the variable with highest R2 would cause a variable dropped earlier to have R2 less than threshold

This can all be implemented via the `redun` function from the `rms` package. Read the documentation for 2 approaches how to handle categorical predictors. We illustrate below. In our example, we include all covariates we wish to consider in our modeling but NOT the outcome. We set our R2 threshold to 0.6.

```
# Redundancy Analysis - No redundancies
redun(formula = ~ BM16 +  BM18 + ARM2 + BM07 + BM15 +
        BM08 + BM09 + BM10 + BM11 + BM12 + BM13 +
        BM02 + BM17 + BM03 + BM04 + BM01 + BM05 + BM06, r2 = 0.6, data = mod_dat)
```

```
##
## Redundancy Analysis
##
## ~BM16 + BM18 + ARM2 + BM07 + BM15 + BM08 + BM09 + BM10 + BM11 +
##     BM12 + BM13 + BM02 + BM17 + BM03 + BM04 + BM01 + BM05 + BM06
##
## n: 1113  p: 18   nk: 3
##
## Number of NAs:    525
## Frequencies of Missing Values Due to Each Variable
## BM16 BM18 ARM2 BM07 BM15 BM08 BM09 BM10 BM11 BM12 BM13 BM02 BM17 BM03 BM04 BM01
##  257   17    0    0    0    0   36  197    2    3    3    0  117    0    3    3
## BM05 BM06
##   87    0
##
```

```
##
## Transformation of target variables forced to be linear
##
## R-squared cutoff: 0.6    Type: ordinary
##
## R^2 with which each variable can be predicted from all other variables:
##
##  BM16  BM18  ARM2  BM07  BM15  BM08  BM09  BM10  BM11  BM12  BM13  BM02  BM17
## 0.071 0.270 0.254 0.128 0.073 0.374 0.118 0.138 0.364 0.171 0.365 0.214 0.161
##  BM03  BM04  BM01  BM05  BM06
## 0.214 0.232 0.194 0.277 0.212
##
## No redundant variables
```

In our example none of our predictors met the requisite threshold, so we will include them all in our modeling.

5 Estimating How Many Degrees of Freedom (df) You Can Afford to Include in Your Model

5.1 Introduction

Before you begin modeling, it is important to be aware how many terms, a.ka.a. df (where each column in your model matrix takes up 1 df), you can afford to include in your model and it to remain robust. The concern is that if you include too many terms, your model may be overfit and will not generalize well to new data. Depending on how many df you can afford to include in your model, you may be able to make less assumptions in your modeling. Some standard assumptions which you may benefit by not making include linearity and the additivity assumption (namely that there are no interactions in your model). However, to break these assumptions will require additional df. Additionally, sometimes you may have too many predictors for your sample size to produce robust estimates and depending on how many df you can afford, you may have to form clusters from some of these predictors to reduce the number of df in your model.

One common mistake, is to try to perform feature selection in order to reduce the number of terms in your model. This is sometimes attempted in the form of stepwise regression. Alternatively, the analyst may first do univariable screening to filter out "unimportant" predictors. Both of these methods are not recommended. By filtering out features from the same cohort you then train your model on, you will have underestimtated standard errors and p-values, overestimated coefficients, confidence intervals that are too narrow, and overestimated performance metrics for the model (e.g. R2).

Instead we recommend trying to filter out unimportant terms apriori via literature review and attempting a redundancy analysis. Beyond this, you should include all terms in your model, and as mentioned previously, if you can not afford the df, try dimension reduction via clustering. Other methods are beyond the scope of this vignette.

5.2 Effective Sample Size

A simple, rough estimate for how many df you can afford to include in your model is that you can afford 1 df for roughly 10-15 **effective** samples in your data. I emphasize the term "effective" samples because it is only equal to the actual sample size of your data in the case of a continuous outcome. However, if there is a binary outcome, the effective sample size is equivalent to the min(n_0, n_1), where n_i is number of subjects for which Y = i. For example, if you have a cohort of 200 patients and you are predicting BOR where Y = {CR/PR, SD/PD}, and there is a 30% response rate, your effective sample size is 60. Therefore

you can afford to include between 4-6 df in your model. For a survival outcome, the effective sample size is the number of events in your cohort.

I summarize this information in the table below:

Response Variable	Effective Sample Size
Continuous	n (total sample size)
Binary	$\min(n_0, n_1)$
Survival Time	Number of events

When multiple imputation is used for missing data, the effective sample size will in fact be little less than the estimates above. Additionally, the above rule of thumb does not take into account the df necessary to estimate the intercept or residual variance. Therefore typically, it is recommended to have ~ 15 effective samples for every df in your model. Shrinkage (e.g. elastice net) can help allow for a smaller ratio.

In our dataset, we have 585 subjects who were primary resistant, thus we can allow for roughly `585/15 = 39` df in our analysis. However, as we will soon see, after accounting for multiple imputation, the true number of df we can afford to include will be much less.

5.3 Shrinkage Factor

A more precise estimate of the number of terms you can afford is done via van Houwelingen's shrinkage factor, $\hat{\gamma}$ (Van Houwelingen and Le Cessie, 1990). Where $\hat{\gamma} = (model\ \chi^2 - p)/model\ \chi^2$ where p is the total df in your model. The model χ^2 is equal to the log Likelihood Ratio for the model. If γ falls below 0.9, for example, we may be concerned with the lack of calibration our model will experience with new data. We can try to estimate the number of df, q, needed to yield a $\gamma \geq 0.9$ via the following algorithm:

1. Run full "saturated" model including all terms you'd ideally like to include (including splines, interactions). Let p denote the number of df in this model.
2. Calculate $\hat{\gamma} = (LR - p)/LR$
3. If $\hat{\gamma} > p + 9$, then reducing the number of terms can yield a better predictive model.
4. Set $q = (LR - p)/9$

We demonstrate below.

```
# There are a few standard overhead steps prior to modeling with the rms package.
options(contrasts=c("contr.treatment", "contr.treatment"))
ddist<- datadist(mod_dat, adjto.cat = "first")
options(datadist='ddist')

# First we run saturated model - Details of this code will be discussed later
sat_mod <- fit.mult.impute(Outcome ~ rcs(BM16, 4) +  rcs(BM18, 4) + ARM2 + BM07 + BM15 +
                           rcs(BM08, 4) + rcs(BM09, 4) + rcs(BM10, 4) + rcs(BM11, 4) + rcs (BM12, 4) +
                           rcs(BM13, 4) + BM02 + BM17 + BM03 + rcs(BM04, 4) + BM01 + BM05 +
                           rcs(BM06, 4) + BM16 %ia% ARM2 + BM18 %ia% ARM2,
                           lrm, MI5, data = mod_dat)
```

```
##
## Wald Statistic Information
##
## Variance Inflation Factors Due to Imputation:
##
##         Intercept              BM16             BM16'              BM18
```

```
##             1.01             1.39             1.46             1.00
##            BM18'           BM18''       ARM2=ARM B       ARM2=ARM C
##             1.00             1.00             1.02             1.01
##       ARM2=ARM D      BM07=Type B      BM07=Type C             BM15
##             1.01             1.01             1.00             1.00
##             BM08            BM08'           BM08''             BM09
##             1.02             1.01             1.01             1.01
##            BM09'           BM09''             BM10            BM10'
##             1.03             1.03             1.28             1.31
##           BM10''             BM11            BM11'           BM11''
##             1.31             1.01             1.01             1.01
##             BM12            BM12'           BM12''             BM13
##             1.01             1.00             1.00             1.01
##            BM13'           BM13''             BM02         BM17=YES
##             1.01             1.01             1.02             1.05
##         BM03=YES             BM04            BM04'           BM04''
##             1.02             1.01             1.01             1.01
##         BM01=>=1      BM05=Type B      BM05=Type C      BM05=Type D
##             1.01             1.01             1.01             1.05
##             BM06            BM06'           BM06'' BM16 * ARM2=ARM B
##             1.01             1.00             1.00             1.10
## BM16 * ARM2=ARM C BM16 * ARM2=ARM D BM18 * ARM2=ARM B BM18 * ARM2=ARM C
##             1.05             1.04             1.02             1.01
## BM18 * ARM2=ARM D
##             1.01
##
## Fraction of Missing Information:
##
##        Intercept             BM16            BM16'             BM18
##             0.01             0.28             0.31             0.00
##            BM18'           BM18''       ARM2=ARM B       ARM2=ARM C
##             0.00             0.00             0.02             0.01
##       ARM2=ARM D      BM07=Type B      BM07=Type C             BM15
##             0.01             0.01             0.00             0.00
##             BM08            BM08'           BM08''             BM09
##             0.02             0.01             0.01             0.01
##            BM09'           BM09''             BM10            BM10'
##             0.03             0.03             0.22             0.24
##           BM10''             BM11            BM11'           BM11''
##             0.24             0.01             0.01             0.01
##             BM12            BM12'           BM12''             BM13
##             0.01             0.00             0.00             0.01
##            BM13'           BM13''             BM02         BM17=YES
##             0.01             0.01             0.02             0.05
##         BM03=YES             BM04            BM04'           BM04''
##             0.02             0.01             0.01             0.01
##         BM01=>=1      BM05=Type B      BM05=Type C      BM05=Type D
##             0.01             0.01             0.01             0.05
##             BM06            BM06'           BM06'' BM16 * ARM2=ARM B
##             0.01             0.00             0.00             0.09
## BM16 * ARM2=ARM C BM16 * ARM2=ARM D BM18 * ARM2=ARM B BM18 * ARM2=ARM C
##             0.05             0.04             0.02             0.01
## BM18 * ARM2=ARM D
##             0.01
```

```
##
## d.f. for t-distribution for Tests of Single Coefficients:
##
##         Intercept              BM16             BM16'              BM18
##          98682.84             50.75             40.42         491088.20
##             BM18'            BM18''        ARM2=ARM B        ARM2=ARM C
##         507805.60         339441.44          14532.62          42084.62
##        ARM2=ARM D       BM07=Type B       BM07=Type C              BM15
##          41970.08         103973.61        1067072.19         208756.46
##              BM08             BM08'            BM08''              BM09
##          15580.42          41806.45          52717.42          18463.04
##             BM09'            BM09''              BM10             BM10'
##           4954.45           4045.00             85.51             70.73
##            BM10''              BM11             BM11'            BM11''
##             70.21          86121.71          49775.77          24081.69
##              BM12             BM12'            BM12''              BM13
##          82485.72         278270.49         312538.46          18629.32
##             BM13'            BM13''              BM02          BM17=YES
##          29781.63          40519.28           9552.06           1636.17
##          BM03=YES              BM04             BM04'            BM04''
##          15760.25          79824.79          66111.69          70425.33
##          BM01=>=1       BM05=Type B       BM05=Type C       BM05=Type D
##         119913.76          48886.27          28142.45           1754.19
##              BM06             BM06'            BM06'' BM16 * ARM2=ARM B
##          95835.27         201006.18         249787.19            465.22
## BM16 * ARM2=ARM C BM16 * ARM2=ARM D BM18 * ARM2=ARM B BM18 * ARM2=ARM C
##           1575.30           3244.42          10912.08          27568.83
## BM18 * ARM2=ARM D
##          42705.57
##
## The following fit components were averaged over the 5 model fits:
##
##   stats linear.predictors
```

```
print(sat_mod)
```

```
## Logistic Regression Model
##
## fit.mult.impute(formula = Outcome ~ rcs(BM16, 4) + rcs(BM18,
##     4) + ARM2 + BM07 + BM15 + rcs(BM08, 4) + rcs(BM09, 4) + rcs(BM10,
##     4) + rcs(BM11, 4) + rcs(BM12, 4) + rcs(BM13, 4) + BM02 +
##     BM17 + BM03 + rcs(BM04, 4) + BM01 + BM05 + rcs(BM06, 4) +
##     BM16 %ia% ARM2 + BM18 %ia% ARM2, fitter = lrm, xtrans = MI5,
##     data = mod_dat)
##
##                        Model Likelihood        Discrimination    Rank Discrim.
##                              Ratio Test               Indexes          Indexes
## Obs          1638    LR chi2     469.81        R2       0.342    C       0.801
##  N           1053    d.f.            48      R2(48,1638)0.227    Dxy     0.603
##  Y            585    Pr(> chi2) <0.0001    R2(48,1128.2)0.312    gamma   0.603
## max |deriv| 9e-08                            Brier    0.168    tau-a   0.277
##
##                  Coef    S.E.    Wald Z Pr(>|Z|)
## Intercept        -0.3002  4.1508 -0.07  0.9424
## BM16             -0.0873  0.0193 -4.53  <0.0001
```

```
## BM16'              0.5848  0.1270  4.60  <0.0001
## BM18               2.6228  1.6510  1.59  0.1121
## BM18'             -6.6122 12.9026 -0.51  0.6083
## BM18''            11.9351 25.3981  0.47  0.6384
## ARM2=ARM B         2.4593  2.2018  1.12  0.2640
## ARM2=ARM C         0.1437  1.8502  0.08  0.9381
## ARM2=ARM D        -0.9694  1.8123 -0.53  0.5927
## BM07=Type B       -0.3812  0.1408 -2.71  0.0068
## BM07=Type C       -0.4140  0.2167 -1.91  0.0561
## BM15              -0.1141  0.1583 -0.72  0.4713
## BM08              -0.0043  0.0087 -0.49  0.6213
## BM08'              0.0600  0.0790  0.76  0.4479
## BM08''            -0.1113  0.1424 -0.78  0.4343
## BM09              -0.0368  0.0566 -0.65  0.5156
## BM09'              0.1831  0.2241  0.82  0.4139
## BM09''            -0.5973  0.6150 -0.97  0.3314
## BM10              -0.0185  0.0072 -2.56  0.0104
## BM10'              1.4900  0.6353  2.35  0.0190
## BM10''            -2.4051  1.0290 -2.34  0.0194
## BM11              -0.0063  0.0114 -0.55  0.5815
## BM11'              0.0038  0.0245  0.16  0.8765
## BM11''             0.0036  0.1494  0.02  0.9806
## BM12              -0.9088  0.4573 -1.99  0.0469
## BM12'              0.6721  1.8059  0.37  0.7098
## BM12''             0.0411  5.2333  0.01  0.9937
## BM13              -0.0049  0.0036 -1.35  0.1773
## BM13'              0.0428  0.0196  2.18  0.0291
## BM13''            -0.1051  0.0491 -2.14  0.0322
## BM02               0.4003  0.1411  2.84  0.0046
## BM17=YES           0.2982  0.1412  2.11  0.0346
## BM03=YES          -0.2364  0.1458 -1.62  0.1049
## BM04               0.0012  0.1880  0.01  0.9949
## BM04'              1.2788  1.1997  1.07  0.2864
## BM04''            -3.2321  2.7550 -1.17  0.2407
## BM01=>=1           0.0858  0.1472  0.58  0.5599
## BM05=Type B       -0.8338  0.3470 -2.40  0.0163
## BM05=Type C       -0.2665  0.3256 -0.82  0.4130
## BM05=Type D       -0.0711  0.3193 -0.22  0.8237
## BM06              -0.0211  0.0153 -1.38  0.1672
## BM06'              0.0174  0.0337  0.52  0.6055
## BM06''            -0.0690  0.1731 -0.40  0.6900
## BM16 * ARM2=ARM B -0.0157  0.0111 -1.42  0.1567
## BM16 * ARM2=ARM C -0.0105  0.0093 -1.13  0.2573
## BM16 * ARM2=ARM D -0.0098  0.0096 -1.02  0.3074
## BM18 * ARM2=ARM B -1.2055  0.8957 -1.35  0.1783
## BM18 * ARM2=ARM C -0.3622  0.7629 -0.47  0.6350
## BM18 * ARM2=ARM D -0.2439  0.7457 -0.33  0.7436
```

```
# We have a LR chi2 of ~ 468 and p = 48 df. Thus, Gamma = (468 - 48)/468 = 0.9.
# Since Gamma >= 0.9, this implies we can keep our saturated model!
# Had we used the heuristic rule of thumb, we would've allowed for 585/15 ~ 39 df.
```

With our current data we have enough samples to allow for the degree of flexibility we would like even without dimension reduction. We demonstrate how to estimate q, the desired number of df on a reduced sample below. FOR ILLUSTRATION PURPOSES ONLY, we will remove all observations with missing data in

order to reduce the sample size.

```
# We will retain only observations with complete data, reducing our sample size from 1638 to 1113
com_dat <- mod_dat[complete.cases(mod_dat),]


# We repeat the above algorithm:
# First we run saturated model - This time modeling directly from the data instead
# of multiple imputation object. Details of this code will be discussed later
sat_mod2 <- lrm(Outcome ~ rcs(BM16, 4) +  rcs(BM18, 4) + ARM2 + BM07 + BM15 +
                          rcs(BM08, 4) + rcs(BM09, 4) + rcs(BM10, 4) + rcs(BM11, 4) +
                rcs (BM12, 4) + rcs(BM13, 4) + BM02 + BM17 + BM03 + rcs(BM04, 4) +
                BM01 + BM05 + rcs(BM06, 4) + BM16 %ia% ARM2 + BM18 %ia% ARM2,
              data = com_dat)

print(sat_mod2)
```

```
## Logistic Regression Model
##
## lrm(formula = Outcome ~ rcs(BM16, 4) + rcs(BM18, 4) + ARM2 +
##     BM07 + BM15 + rcs(BM08, 4) + rcs(BM09, 4) + rcs(BM10, 4) +
##     rcs(BM11, 4) + rcs(BM12, 4) + rcs(BM13, 4) + BM02 + BM17 +
##     BM03 + rcs(BM04, 4) + BM01 + BM05 + rcs(BM06, 4) + BM16 %ia%
##     ARM2 + BM18 %ia% ARM2, data = com_dat)
##
##                        Model Likelihood       Discrimination    Rank Discrim.
##                              Ratio Test              Indexes          Indexes
## Obs          1113    LR chi2     330.22       R2       0.352    C       0.807
##  N            713    d.f.            48     R2(48,1113)0.224    Dxy     0.615
##  Y            400    Pr(> chi2) <0.0001    R2(48,768.7)0.307    gamma   0.615
## max |deriv| 3e-08                            Brier    0.167    tau-a   0.283
##
##                   Coef    S.E.    Wald Z Pr(>|Z|)
## Intercept          3.3233  5.1076  0.65  0.5153
## BM16              -0.0869  0.0195 -4.45  <0.0001
## BM16'              0.5898  0.1279  4.61  <0.0001
## BM18               1.3388  2.0402  0.66  0.5117
## BM18'             -2.9217 15.6972 -0.19  0.8523
## BM18''             6.4412 32.3405  0.20  0.8421
## ARM2=ARM B         1.1881  2.5428  0.47  0.6403
## ARM2=ARM C        -3.0944  2.3918 -1.29  0.1958
## ARM2=ARM D        -2.0137  2.0289 -0.99  0.3209
## BM07=Type B       -0.3495  0.1755 -1.99  0.0464
## BM07=Type C       -0.2528  0.2551 -0.99  0.3216
## BM15              -0.1117  0.1872 -0.60  0.5508
## BM08              -0.0078  0.0106 -0.74  0.4619
## BM08'              0.0900  0.0983  0.92  0.3600
## BM08''            -0.1702  0.1798 -0.95  0.3438
## BM09              -0.0772  0.0698 -1.11  0.2685
## BM09'              0.2678  0.2723  0.98  0.3254
## BM09''            -0.7889  0.7453 -1.06  0.2898
## BM10              -0.0215  0.0081 -2.66  0.0078
## BM10'              1.4291  0.5691  2.51  0.0120
## BM10''            -2.3800  0.9502 -2.50  0.0123
## BM11              -0.0088  0.0139 -0.63  0.5274
```

```
## BM11'              0.0102  0.0278  0.37  0.7134
## BM11''            -0.0682  0.1969 -0.35  0.7292
## BM12              -0.7602  0.5686 -1.34  0.1812
## BM12'              0.0341  2.1980  0.02  0.9876
## BM12''             1.7982  6.5600  0.27  0.7840
## BM13              -0.0049  0.0044 -1.11  0.2650
## BM13'              0.0428  0.0249  1.72  0.0853
## BM13''            -0.1060  0.0618 -1.72  0.0863
## BM02               0.5862  0.1710  3.43  0.0006
## BM17=YES           0.2938  0.1654  1.78  0.0756
## BM03=YES          -0.1386  0.1766 -0.78  0.4326
## BM04               0.1298  0.2255  0.58  0.5648
## BM04'              0.3745  1.4383  0.26  0.7945
## BM04''            -1.2160  3.3564 -0.36  0.7171
## BM01=>=1           0.1945  0.1824  1.07  0.2863
## BM05=Type B       -1.2728  0.4119 -3.09  0.0020
## BM05=Type C       -0.4565  0.3700 -1.23  0.2173
## BM05=Type D       -0.3278  0.3566 -0.92  0.3580
## BM06              -0.0076  0.0188 -0.40  0.6868
## BM06'             -0.0072  0.0423 -0.17  0.8641
## BM06''            -0.0278  0.2174 -0.13  0.8983
## BM16 * ARM2=ARM B -0.0146  0.0123 -1.19  0.2354
## BM16 * ARM2=ARM C -0.0048  0.0123 -0.39  0.6995
## BM16 * ARM2=ARM D -0.0093  0.0106 -0.88  0.3779
## BM18 * ARM2=ARM B -0.7001  1.0306 -0.68  0.4969
## BM18 * ARM2=ARM C  0.9456  0.9843  0.96  0.3367
## BM18 * ARM2=ARM D  0.1906  0.8372  0.23  0.8199
```

```
# This time we have a LR chi2 of ~ 330 and p = 48 df.
# Thus, Gamma = (330 - 48)/330 = 0.85.
# Since Gamma < 0.9, this implies we need to improve our model by reducing the number
# of df it includes.
# We solve for q: (LR - p)/9 = (330 - 48)/9 = 31.3.
# Thus we can safely include 31-32 df in our model.
```

6 Spending Your Degrees of Freedom

Based on the information above, we can only include 31-32 df in our model instead of our ideal 48. Therefore we will have to make some difficult decisions. Suppose we decide to do away with any interactions (2*3 df = 6 df). This would reduce our ideal model to 48 - 6 = 42 df. We still have to reduce our model further. The next step would be to then prioritize which variables we allow to be non-linear. For simplicity, assuming we decide to keep the number of knots in each cubic spline at 4 (3 df), this would allow us to have 4 continuous predictors with cubic splines, the rest constrained to be linear.

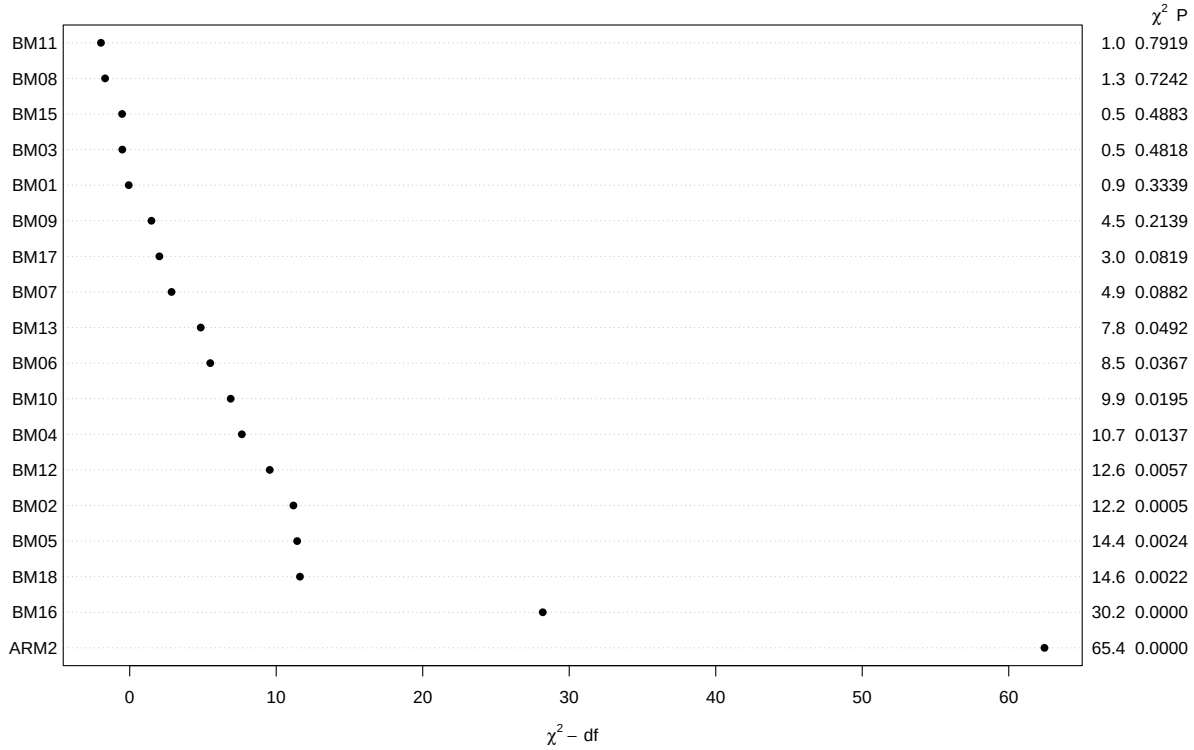
In order to choose which predictors to allow flexibility for, we suggest prioritizing the most important ones, where important terms are the ones most strongly associated with the response variable `Outcome`. This may sound counterintuitive considering our strong warning against using the outcome to perform feature selection above (section 5.1). However, Harrell Jr (2015) asserts that since there is no reason to assume the degree of non-linearity should be associated with the strength of association with y, this approach has the potential to hurt your model just as it can improve it and therefore does not lead to overfitting. However, this approach serves to prevent big mistakes and only allow for small ones.

In order to rank the predictors, we will first run a saturated (and overfit) model allowing every continuous predictor to be a cubic spline with 4 knots. We will then choose the 4 predictors with the strongest associations

to retain cubic splines, the rest will be constrained to be linear.

We demonstrate below.

```
# First we run a saturated model
sat_mod3 <- lrm(Outcome ~ rcs(BM16, 4) +  rcs(BM18, 4) + ARM2 + BM07 + BM15 +
                rcs(BM08, 4) + rcs(BM09, 4) + rcs(BM10, 4) + rcs(BM11, 4) + rcs (BM12, 4) + rcs(BM13, 4
                BM02 + BM17 + BM03 + rcs(BM04, 4) + BM01 + BM05 + rcs(BM06, 4),
                data = com_dat)

# Then we visualize the ranks of the predictors - NOT LOOKING at tests for non-linearity
plot(anova(sat_mod3))
```




```
# The 4 continuous predictors with the greatest strengths of association are BM18, BM04, BM12, and BM16
# The rest we will constrain to be linear.
# We now fit our final (reduced) model

red_mod <- lrm(Outcome ~ rcs(BM16, 4) +  rcs(BM18, 4) + ARM2 + BM07 + BM15 +
                            BM08 + BM09 + BM10 + BM11 + rcs(BM12, 4) + BM13 +
                            BM02 + BM17 + BM03 + rcs(BM04, 4) + BM01 + BM05 + BM06,
               data = com_dat)
```

7 Dimension Reduction

Suppose you do not have enough df to fit all of the predictors from your data into the model. In this case it is necessary to first perform some sort of dimension reduction prior to modeling. One way to accomplish this

is via clustering.

We illustrate below.

```
# For illustration purposes, we will remove half of our observations from our original cohort
set.seed(123)
index <- sample(nrow(mod_dat), size = nrow(mod_dat)/2, replace = F)
half <- mod_dat[index,]

# We use rule of thumb to estimate df we can afford. Here we have effective sample
# size of 309 therefore 309/15 = 21 df. We definitely need major dimension reduction.
describe(half$Outcome)
```

```
## half$Outcome
##        n  missing distinct
##      819        0        2
##
## Value          N     Y
## Frequency    510   309
## Proportion 0.623 0.377
```

```
# Here we perform multiple imputation via the transcan function
# It has a different algorithm than aregimpute. Generally aregimpute is preferred
# However, transcan also transforms variables to optimize correlation between variables
# This is useful for dimension reduction via clustering
nimp <- 20
ptrans <- transcan(~ BM16 +  BM18 + ARM2 + BM07 + BM15 +
                             BM08 + BM09 + BM10 + BM11 + BM12 + BM13 +
                             BM02 + BM17 + BM03 + BM04 + BM01 + BM05 + BM06,
                imputed = T, transformed = T, trantab = T, pl = F, show.na = T, data = half,
                n.impute = nimp, pr = F)
```

```
## Warning in transcan(~BM16 + BM18 + ARM2 + BM07 + BM15 + BM08 + BM09 + BM10 + : transcan provides only
## A better approximation is provided by the aregImpute function.
## The MICE and other S libraries provide imputations from Bayesian posterior distributions.
```

```
summary(ptrans, digits = 4)
```

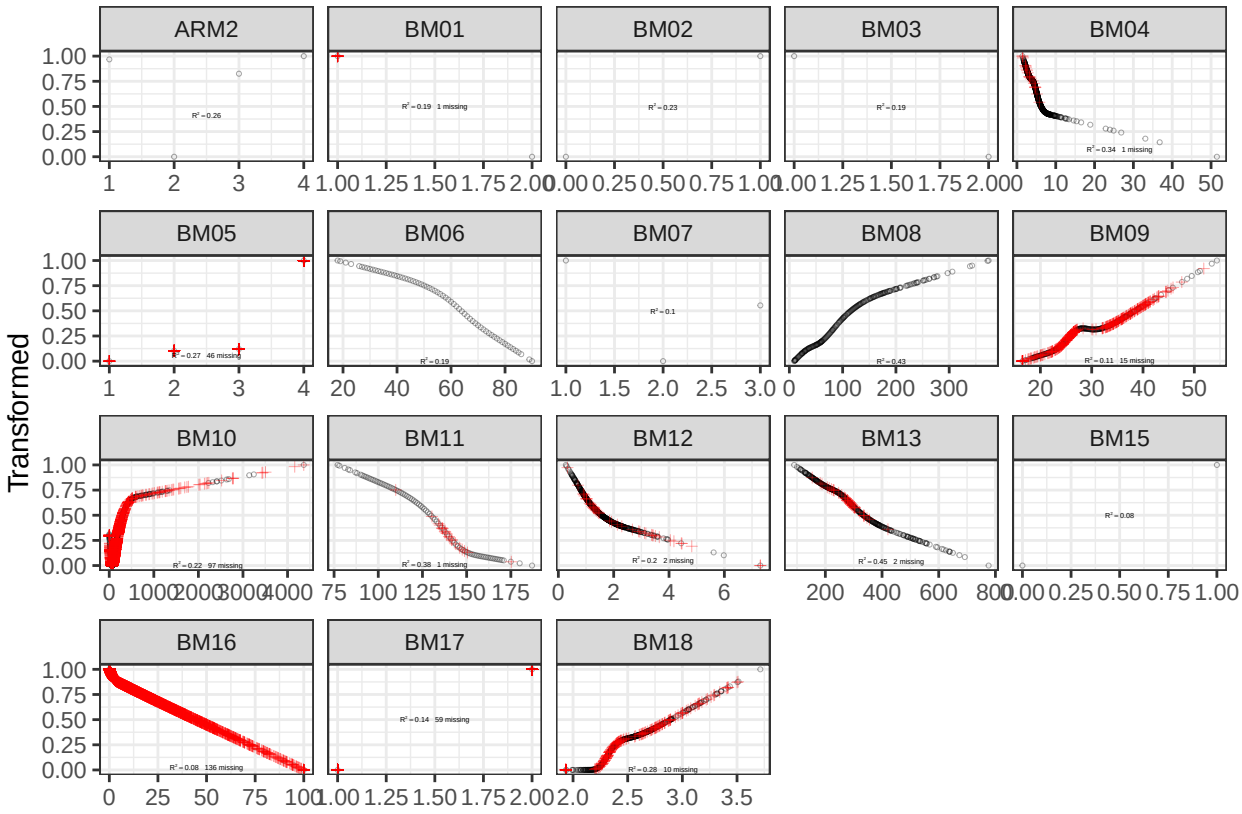
```
## transcan(x = ~BM16 + BM18 + ARM2 + BM07 + BM15 + BM08 + BM09 +
##     BM10 + BM11 + BM12 + BM13 + BM02 + BM17 + BM03 + BM04 + BM01 +
##     BM05 + BM06, imputed = T, n.impute = nimp, trantab = T, transformed = T,
##     pr = F, pl = F, show.na = T, data = half)
##
## Iterations: 6
##
## R-squared achieved in predicting each variable:
##
##  BM16  BM18  ARM2  BM07  BM15  BM08  BM09  BM10  BM11  BM12  BM13  BM02  BM17
## 0.078 0.280 0.257 0.104 0.083 0.431 0.106 0.225 0.383 0.197 0.447 0.230 0.142
##  BM03  BM04  BM01  BM05  BM06
## 0.187 0.336 0.190 0.270 0.188
##
## Adjusted R-squared:
##
##  BM16  BM18  ARM2  BM07  BM15  BM08  BM09  BM10  BM11  BM12  BM13  BM02  BM17
## 0.049 0.260 0.238 0.081 0.061 0.415 0.081 0.201 0.366 0.175 0.431 0.212 0.120
##  BM03  BM04  BM01  BM05  BM06
```

```
## 0.168 0.317 0.170 0.250 0.166
##
## Coefficients of canonical variates for predicting each (row) variable
##
##      BM16  BM18  ARM2  BM07  BM15  BM08  BM09  BM10  BM11  BM12  BM13  BM02
## BM16        0.83  0.15  0.16  0.77 -0.18  0.26 -0.01 -0.01  0.04 -0.01  0.41
## BM18  0.41       -0.05  0.15 -0.37  0.60  0.04  0.06  0.05  0.06  0.08  0.50
## ARM2  0.10 -0.05        0.07 -0.21 -0.26  0.12 -0.80 -0.24  0.10 -0.16  0.11
## BM07  0.19  0.30  0.13        0.00 -0.28 -0.52 -0.40 -0.15 -0.29 -0.03 -0.57
## BM15  0.04 -0.04 -0.02  0.00        0.01  0.02 -0.02  0.04  0.02 -0.03  0.01
## BM08 -0.07  0.39 -0.16 -0.09  0.06        0.04 -0.02  0.19  0.18 -0.34  0.33
## BM09  0.24  0.08  0.20 -0.49  0.44  0.11        0.19  0.05 -0.38  0.46 -0.19
## BM10  0.02  0.07 -0.91 -0.21 -0.19 -0.04  0.12        0.08  0.13 -0.06 -0.03
## BM11  0.00  0.04 -0.16 -0.05  0.29  0.22  0.02  0.06        0.34 -0.47 -0.13
## BM12  0.04  0.07  0.13 -0.19  0.21  0.37 -0.25  0.17  0.61        0.54  0.20
## BM13  0.01  0.05 -0.09 -0.01 -0.14 -0.33  0.14 -0.03 -0.39  0.25        0.15
## BM02  0.02  0.06  0.01 -0.03  0.01  0.06 -0.01  0.00 -0.02  0.02  0.03
## BM17  0.15  0.00  0.50  0.05 -0.30  0.11 -0.25  0.32  0.18 -0.04  0.38 -0.22
## BM03 -0.06  0.15  0.17  0.15  0.18 -0.45  0.00  0.03  0.91 -0.20 -0.27  0.13
## BM04  0.19 -0.23  0.08 -0.02 -0.15 -0.18 -0.16  0.16  0.14  0.04  0.79  0.05
## BM01 -0.26 -0.01  0.02  0.01 -0.25 -0.27  0.00 -0.05 -0.39 -0.23  0.14  0.29
## BM05  0.00  0.11 -0.04  0.07  0.09  0.34 -0.21 -0.12  0.04  0.09 -0.16  1.36
## BM06  0.17 -0.13 -0.25  0.01 -0.52  0.03 -0.10 -0.14 -0.37  0.05 -0.63  0.16
##      BM17  BM03  BM04  BM01  BM05  BM06
## BM16  0.18 -0.05  0.48 -0.39  0.00  0.28
## BM18 -0.01  0.11 -0.27 -0.01  0.11 -0.10
## ARM2  0.34  0.13  0.10  0.02 -0.05 -0.20
## BM07  0.07  0.22 -0.04  0.02  0.20  0.02
## BM15 -0.02  0.01 -0.02 -0.02  0.01 -0.04
## BM08  0.04 -0.21 -0.13 -0.13  0.21  0.01
## BM09 -0.33  0.00 -0.39  0.00 -0.41 -0.14
## BM10  0.23  0.04  0.22 -0.04 -0.13 -0.11
## BM11  0.08  0.48  0.12 -0.21  0.02 -0.20
## BM12 -0.03 -0.19  0.07 -0.23  0.12  0.05
## BM13  0.15 -0.12  0.57  0.06 -0.09 -0.28
## BM02 -0.02  0.01  0.01  0.03  0.16  0.01
## BM17       -0.01 -0.46  0.18  0.11  0.76
## BM03 -0.02        0.15  0.03 -0.18  0.20
## BM04 -0.24  0.09        0.16 -0.01  0.09
## BM01  0.13  0.03  0.26       -0.17  0.23
## BM05  0.08 -0.13  0.00 -0.12       -0.01
## BM06  0.66  0.20  0.14  0.24 -0.02
##
## Summary of imputed values
##
## BM16
##        n  missing distinct     Info     Mean      Gmd      .05      .10
##     2720        0     1701    0.971    18.46       24    0.000    0.000
##      .25      .50      .75      .90      .95
##    0.000    7.854   30.782   53.310   67.250
##
## lowest : 0         0.0005688 0.002858  0.01112   0.01362
## highest: 98.38     98.64     98.76     99.87     100
## BM18
```

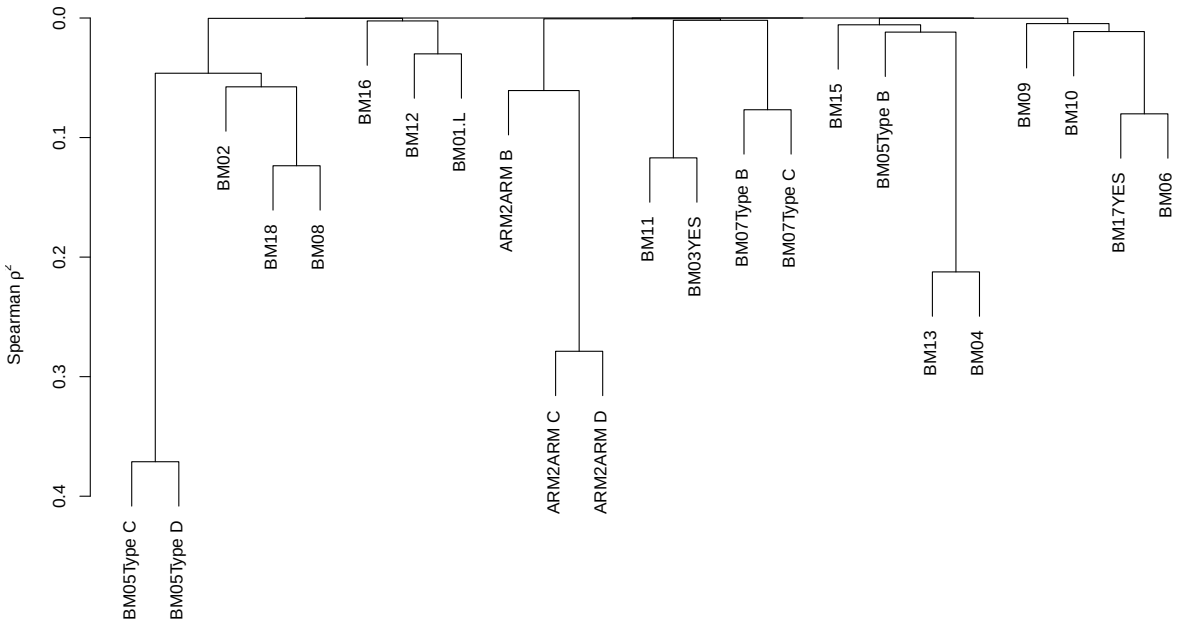
```
##        n  missing distinct     Info     Mean      Gmd      .05      .10
##      200        0      146    0.993    2.471   0.4487    1.934    1.934
##      .25      .50      .75      .90      .95
##    2.263    2.386    2.755    3.039    3.232
##
## lowest : 1.934 2.2   2.208 2.217 2.235, highest: 3.411 3.417 3.419 3.501 3.51
## BM09
##        n  missing distinct     Info     Mean      Gmd      .05      .10
##      300        0      247    0.999     29.2    9.391    16.50    17.05
##      .25      .50      .75      .90      .95
##    23.67    26.48    35.99    40.39    41.96
##
## lowest : 16.5  16.53 16.67 17.09 17.54, highest: 44.85 45.46 45.85 47.56 51.84
## BM10
##        n  missing distinct     Info     Mean      Gmd      .05      .10
##     1940        0     1253    0.989    155.4    159.3    24.12    44.79
##      .25      .50      .75      .90      .95
##    64.00    87.12   160.20   255.81   340.86
##
## lowest : 3.01  3.021 3.135 3.29  3.386, highest: 3426  3444  3506  4161  4364
## BM11
##        n  missing distinct     Info     Mean      Gmd      .05      .10
##       20        0       20        1    140.4    11.63    129.4    133.7
##      .25      .50      .75      .90      .95
##    136.2    139.3    144.2    149.0    151.7
##
## Value      109.9 130.4 134.1 134.8 136.1 136.2 136.5 137.6 138.0 138.6 140.0
## Frequency      1     1     1     1     1     1     1     1     1     1     1
## Proportion  0.05  0.05  0.05  0.05  0.05  0.05  0.05  0.05  0.05  0.05  0.05
##
## Value      140.5 141.6 141.7 143.6 146.0 147.5 148.8 150.5 175.2
## Frequency      1     1     1     1     1     1     1     1     1
## Proportion  0.05  0.05  0.05  0.05  0.05  0.05  0.05  0.05  0.05
##
## For the frequency table, variable is rounded to the nearest 0
## BM12
##        n  missing distinct     Info     Mean      Gmd      .05      .10
##       40        0       39        1    2.455    1.788   0.8895   0.9456
##      .25      .50      .75      .90      .95
##   1.2117   1.8850   3.4162   4.4463   4.9459
##
## lowest : 0.3298 0.7098 0.899  0.9182 0.9486, highest: 4.177  4.444  4.467  4.822  7.3
## BM13
##        n  missing distinct     Info     Mean      Gmd      .05      .10
##       40        0       39        1    278.4     55.8    188.0    213.0
##      .25      .50      .75      .90      .95
##    255.1    284.3    301.4    335.8    347.4
##
## lowest : 153.6 181.9 188.3 207   213.7, highest: 335.7 336.8 347   355.4 420.8
## BM17
##        n  missing distinct     Info     Mean      Gmd
##     1180        0        2    0.672    1.339   0.4485
##
## Value          1     2
```

```
## Frequency    780   400
## Proportion 0.661 0.339
## BM04
##        n  missing distinct     Info     Mean      Gmd      .05      .10
##       20        0       19    0.999    3.297    1.527    1.427    1.553
##      .25      .50      .75      .90      .95
##    2.303    3.200    4.472    4.774    5.188
##
## Value      1.427 1.567 2.179 2.233 2.326 2.488 2.609 2.793 3.195 3.206 3.316
## Frequency      2     1     1     1     1     1     1     1     1     1     1
## Proportion  0.10  0.05  0.05  0.05  0.05  0.05  0.05  0.05  0.05  0.05  0.05
##
## Value      3.609 4.099 4.401 4.684 4.711 4.731 5.157 5.785
## Frequency      1     1     1     1     1     1     1     1
## Proportion  0.05  0.05  0.05  0.05  0.05  0.05  0.05  0.05
##
## For the frequency table, variable is rounded to the nearest 0
## BM01
##        n  missing distinct     Info     Mean      Gmd
##       20        0        1        0        1        0
##
## Value       1
## Frequency  20
## Proportion  1
## BM05
##        n  missing distinct     Info     Mean      Gmd
##      920        0        4    0.795     3.37   0.6614
##
## Value          1     2     3     4
## Frequency     20    40   440   420
## Proportion 0.022 0.043 0.478 0.457
##
## For the frequency table, variable is rounded to the nearest 0
##
## Starting estimates for imputed values:
##
##    BM16    BM18    ARM2    BM07    BM15    BM08    BM09    BM10    BM11    BM12
##   1.000   2.356   4.000   1.000   0.000  57.000  27.100  64.000 136.000   1.450
##    BM13    BM02    BM17    BM03    BM04    BM01    BM05    BM06
## 252.000   0.000   1.000   2.000   4.226   1.000   4.000  60.000
```

```
# Setting scale = T scales transformed values to [0,1] before plotting
ggplot(ptrans, scale = T)
```




```
# Cluster variables
vc <- varclus(~ BM16 +  BM18 + ARM2 + BM07 + BM15 +
                        BM08 + BM09 + BM10 + BM11 + BM12 + BM13 +
                        BM02 + BM17 + BM03 + BM04 + BM01 + BM05 + BM06,
              data = half)

plot(vc)
```




```
# Make list of each imputed dataset (without transformations)
half_tr <- list()
for(i in 1:nimp){
half_tr[[i]] <- impute.transcan(ptrans, imputation = i,  data = half, list.out = T)
half_tr[[i]] <- as.data.frame(half_tr[[i]])
}
```

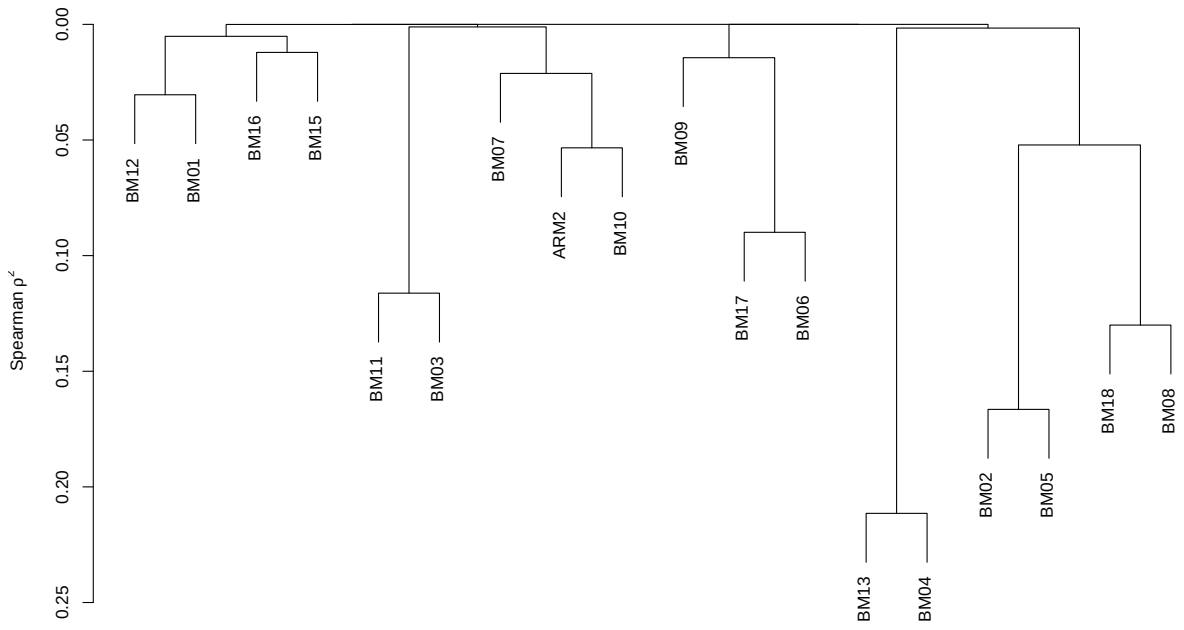
```
##
##
## Imputed missing values with the following frequencies
##  and stored them in variables with their original names:
##
## BM16 BM18 BM09 BM10 BM11 BM12 BM13 BM17 BM04 BM01 BM05
##  136   10   15   97    1    2    2   59    1    1   46
##
##
## Imputed missing values with the following frequencies
##  and stored them in variables with their original names:
##
## BM16 BM18 BM09 BM10 BM11 BM12 BM13 BM17 BM04 BM01 BM05
##  136   10   15   97    1    2    2   59    1    1   46
##
##
## Imputed missing values with the following frequencies
##  and stored them in variables with their original names:
##
## BM16 BM18 BM09 BM10 BM11 BM12 BM13 BM17 BM04 BM01 BM05
```

```
##  136   10   15   97    1    2    2   59    1    1   46
##
##
## Imputed missing values with the following frequencies
##  and stored them in variables with their original names:
##
## BM16 BM18 BM09 BM10 BM11 BM12 BM13 BM17 BM04 BM01 BM05
##  136   10   15   97    1    2    2   59    1    1   46
##
##
## Imputed missing values with the following frequencies
##  and stored them in variables with their original names:
##
## BM16 BM18 BM09 BM10 BM11 BM12 BM13 BM17 BM04 BM01 BM05
##  136   10   15   97    1    2    2   59    1    1   46
##
##
## Imputed missing values with the following frequencies
##  and stored them in variables with their original names:
##
## BM16 BM18 BM09 BM10 BM11 BM12 BM13 BM17 BM04 BM01 BM05
##  136   10   15   97    1    2    2   59    1    1   46
##
##
## Imputed missing values with the following frequencies
##  and stored them in variables with their original names:
##
## BM16 BM18 BM09 BM10 BM11 BM12 BM13 BM17 BM04 BM01 BM05
##  136   10   15   97    1    2    2   59    1    1   46
##
##
## Imputed missing values with the following frequencies
##  and stored them in variables with their original names:
##
## BM16 BM18 BM09 BM10 BM11 BM12 BM13 BM17 BM04 BM01 BM05
##  136   10   15   97    1    2    2   59    1    1   46
##
##
## Imputed missing values with the following frequencies
##  and stored them in variables with their original names:
##
## BM16 BM18 BM09 BM10 BM11 BM12 BM13 BM17 BM04 BM01 BM05
##  136   10   15   97    1    2    2   59    1    1   46
##
##
## Imputed missing values with the following frequencies
##  and stored them in variables with their original names:
##
## BM16 BM18 BM09 BM10 BM11 BM12 BM13 BM17 BM04 BM01 BM05
##  136   10   15   97    1    2    2   59    1    1   46
##
##
## Imputed missing values with the following frequencies
##  and stored them in variables with their original names:
```

```
##
## BM16 BM18 BM09 BM10 BM11 BM12 BM13 BM17 BM04 BM01 BM05
##  136   10   15   97    1    2    2   59    1    1   46
##
##
## Imputed missing values with the following frequencies
##  and stored them in variables with their original names:
##
## BM16 BM18 BM09 BM10 BM11 BM12 BM13 BM17 BM04 BM01 BM05
##  136   10   15   97    1    2    2   59    1    1   46
##
##
## Imputed missing values with the following frequencies
##  and stored them in variables with their original names:
##
## BM16 BM18 BM09 BM10 BM11 BM12 BM13 BM17 BM04 BM01 BM05
##  136   10   15   97    1    2    2   59    1    1   46
##
##
## Imputed missing values with the following frequencies
##  and stored them in variables with their original names:
##
## BM16 BM18 BM09 BM10 BM11 BM12 BM13 BM17 BM04 BM01 BM05
##  136   10   15   97    1    2    2   59    1    1   46
##
##
## Imputed missing values with the following frequencies
##  and stored them in variables with their original names:
##
## BM16 BM18 BM09 BM10 BM11 BM12 BM13 BM17 BM04 BM01 BM05
##  136   10   15   97    1    2    2   59    1    1   46
##
##
## Imputed missing values with the following frequencies
##  and stored them in variables with their original names:
##
## BM16 BM18 BM09 BM10 BM11 BM12 BM13 BM17 BM04 BM01 BM05
##  136   10   15   97    1    2    2   59    1    1   46
##
##
## Imputed missing values with the following frequencies
##  and stored them in variables with their original names:
##
## BM16 BM18 BM09 BM10 BM11 BM12 BM13 BM17 BM04 BM01 BM05
##  136   10   15   97    1    2    2   59    1    1   46
##
##
## Imputed missing values with the following frequencies
##  and stored them in variables with their original names:
##
## BM16 BM18 BM09 BM10 BM11 BM12 BM13 BM17 BM04 BM01 BM05
##  136   10   15   97    1    2    2   59    1    1   46
##
##
```

```
## Imputed missing values with the following frequencies
##  and stored them in variables with their original names:
##
## BM16 BM18 BM09 BM10 BM11 BM12 BM13 BM17 BM04 BM01 BM05
##  136   10   15   97    1    2    2   59    1    1   46
##
##
## Imputed missing values with the following frequencies
##  and stored them in variables with their original names:
##
## BM16 BM18 BM09 BM10 BM11 BM12 BM13 BM17 BM04 BM01 BM05
##  136   10   15   97    1    2    2   59    1    1   46
```

```
# Sample raw imputed dataset
# We will perform Principal Component Analysis on this dataset
imp1 <- half_tr[[1]]

# Clustering of imputed, transformed data
# This can be helpful to manually assign clusters to variables
vc_tr <- varclus(ptrans$transformed,
                 data = imp1)

plot(vc_tr)
```




```
# Based on diagram, lets make the following clusters:
# 1) BM13, BM04
# 2) BM02, BM05, BM18, BM08
```

```r
# 3) BM11, BM03
# 4) BM17, BM06
# 5) Other - don't cluster

# We will then take 1st principal component from each cluster and model it
# Make function to compute first princ comp from given cluster
pco <- function(v, data_tr) {
f <-  princomp(data_tr[,v], cor=TRUE)
vars  <-  f$sdev^2
cat( 'Fraction of variance explained by PC1:',
round(vars[1]/sum(vars ),2), ' \n ')
f$scores[,1]
}

cluster1 <-  pco(v = c( 'BM13' , 'BM04'), data_tr = ptrans$transformed)
```

```
## Fraction of variance explained by PC1: 0.75
##
```

```r
cluster2 <- pco(v =c("BM02", "BM05", "BM18", "BM08"), data_tr = ptrans$transformed)
```

```
## Fraction of variance explained by PC1: 0.5
##
```

```r
cluster3 <- pco(v =c("BM11", "BM03"), data_tr = ptrans$transformed)
```

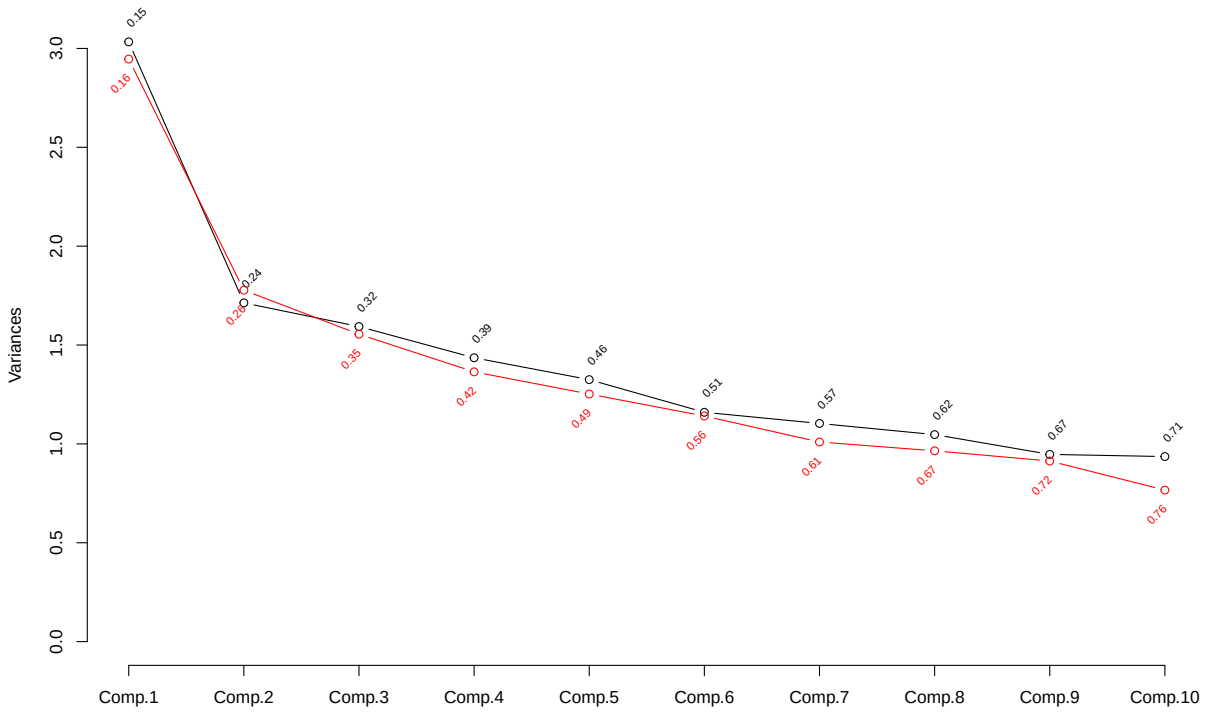
```
## Fraction of variance explained by PC1: 0.66
##
```

```r
cluster4 <- pco(v =c("BM17", "BM06"), data_tr = ptrans$transformed)
```

```
## Fraction of variance explained by PC1: 0.65
##
```

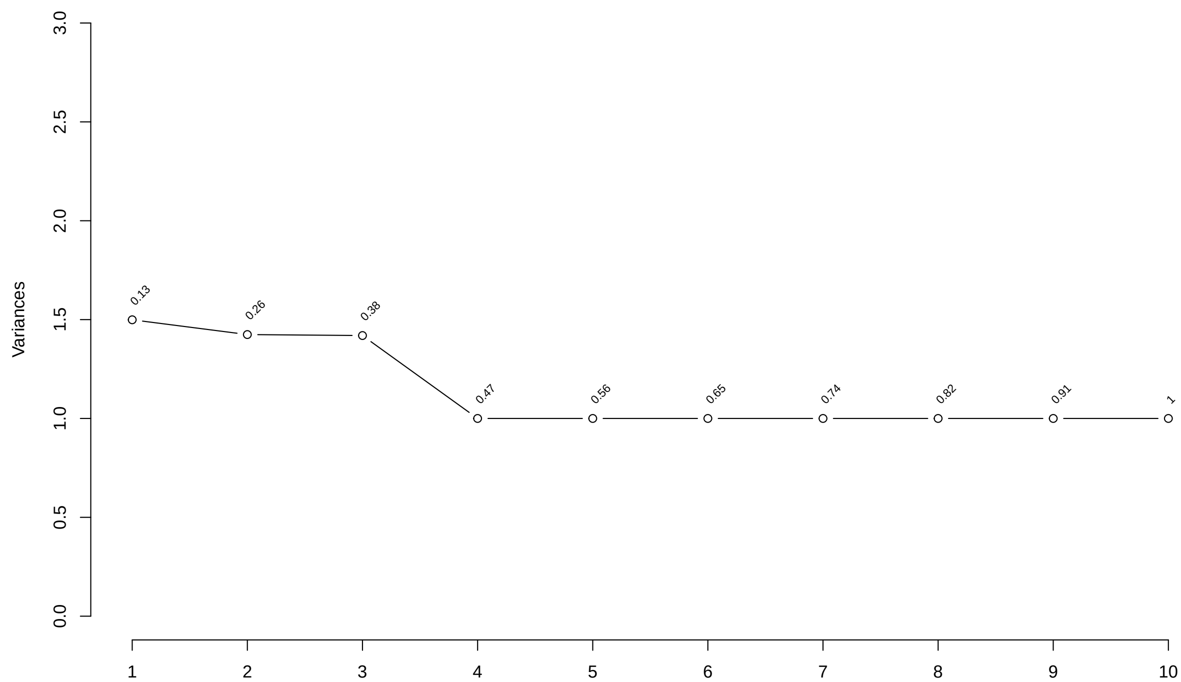
```r
cluster5 <- ptrans$transformed[, c("BM12", "BM01", "BM16", "BM15", "BM07", "ARM2", "BM10", "BM09")]

# Run model
Y <- half$Outcome
f_clust <- lrm(Y ~ cluster1 + cluster2 + cluster3 + cluster4 + cluster5, x = T, y = T)
AIC_clust <- AIC(f_clust)

# Principal Component Analysis (PCA)
# We take out treatment since that will be added separately to model

# First we transform categorical variables to numeric
BM05_num <- model.matrix(~ BM05, data = imp1)[, -1]
BM07_num <- model.matrix(~ BM07, data = imp1)[, -1]
BM17_num <- model.matrix(~ BM17, data = imp1)[, -1]
BM03_num <- model.matrix(~ BM03, data = imp1)[, -1]
BM01_num <- model.matrix(~ BM01, data = imp1)[, -1]

# Calculate principal components
prin.raw <- princomp(~ BM16 +  BM18 + BM07_num + BM15 +
                            BM08 + BM09 + BM10 + BM11 + BM12 + BM13 +
                            BM02 + BM17_num + BM03_num + BM04 + BM01_num + BM05_num + BM06,
                     cor = T, data = imp1)
```

```r
# Make scree plot of PCA on transformed variables
plot (prin.raw , type = 'lines' , main = '', ylim = c(0, 3))

# Make function to add cumulative variance explained to scree plot
addscree <- function (x, npcs=min(10, length (x$sdev)),
plotv =FALSE, col=1, offset =.8 , adj=0, pr=FALSE) {
vars <- x$sdev^2
cumv <-  cumsum (vars)/sum(vars)
if(pr) print (cumv)
text(1:npcs , vars[1:npcs] + offset*par('cxy')[2],
as.character(round(cumv [1:npcs], 2)),
srt=45, adj=adj , cex=.65 , xpd=NA , col=col)
if(plotv) lines(1:npcs, vars[1: npcs], type = 'b' , col=col)
}

addscree (prin.raw)

# Calculate PCA for transformed variables
prin.trans <- princomp(ptrans$transformed, cor = T)
addscree(prin.trans, npcs = 10, plotv = T, col = "red",
         offset = -0.8, adj = 1)
```




```r
# Assess how many PCs to include

pcs_raw <- prin.raw$scores
aic_raw <- rep(NA, ncol(prin.raw$scores))
```

```r
for (i in seq_len(ncol(prin.raw$scores))) {
  ps <- pcs_raw[,seq_len(i)]
  aic_raw[i] <- AIC(lrm(half$Outcome ~ ps + ptrans$transformed[, "ARM2"]))
}


# plot (seq_len(ncol(prin.raw$scores)), aic , xlab= 'Number of Components Used' ,
# ylab = 'AIC' , type= 'l', ylim = c(900, 1000))


pcs_tr <- prin.trans$scores
aic_tr <- rep(NA, length(aic_raw))
for (i in seq_len(ncol(prin.trans$scores))) {
  ps <- pcs_tr[,seq_len(i)]
  aic_tr[i] <- AIC(lrm(half$Outcome ~ ps))
}
# lines(seq_len(ncol(prin.trans$scores)), aic_tr, col = "red")

# aicpl <- data.frame(x=seq_len(length(aic)), aic = aic)
# ggplot(aicpl, aes(x = x, y = aic)) +
#   geom_line() +
#   xlab("Number of Components Used") +
#   ylab("AIC") +
#   scale_x_continuous(breaks = seq_len(nrow(aicpl))) +
#   ylim(950, 1050)


f_full <- lrm(Y ~ BM16 +  BM18 + BM07_num + BM15 +
                            BM08 + BM09 + BM10 + BM11 + BM12 + BM13 +
                            BM02 + BM17_num + BM03_num + BM04 + BM01_num + BM05_num +
                BM06 + ARM2, data = imp1, x = T, y = T)

f_spline <- lrm(Y ~ BM16 +  rcs(BM18, 4) + BM07_num + BM15 +
                  rcs(BM08, 4) + rcs(BM09, 4) + rcs(BM10, 4) + rcs(BM04, 4) + BM01_num +
                  rcs(BM11, 4) + rcs(BM12, 4) + rcs(BM13, 4) + BM02 + BM17_num + BM03_num +
                  BM05_num + rcs(BM06, 4) + ARM2, data = imp1, x = T, y = T)

AIC_full <- AIC(f_full)
AIC_spline <- AIC(f_spline)
# abline (h=AIC(f), col= 'blue')


# Compare to Sparse PCA
s <- sPCAgrid(ptrans$transformed, k = 10, method = "sd",
              center = mean, scale = sd, scores = T)

plot(s, type = 'lines' , main= '' , ylim =c(0,3))
addscree(s)
```



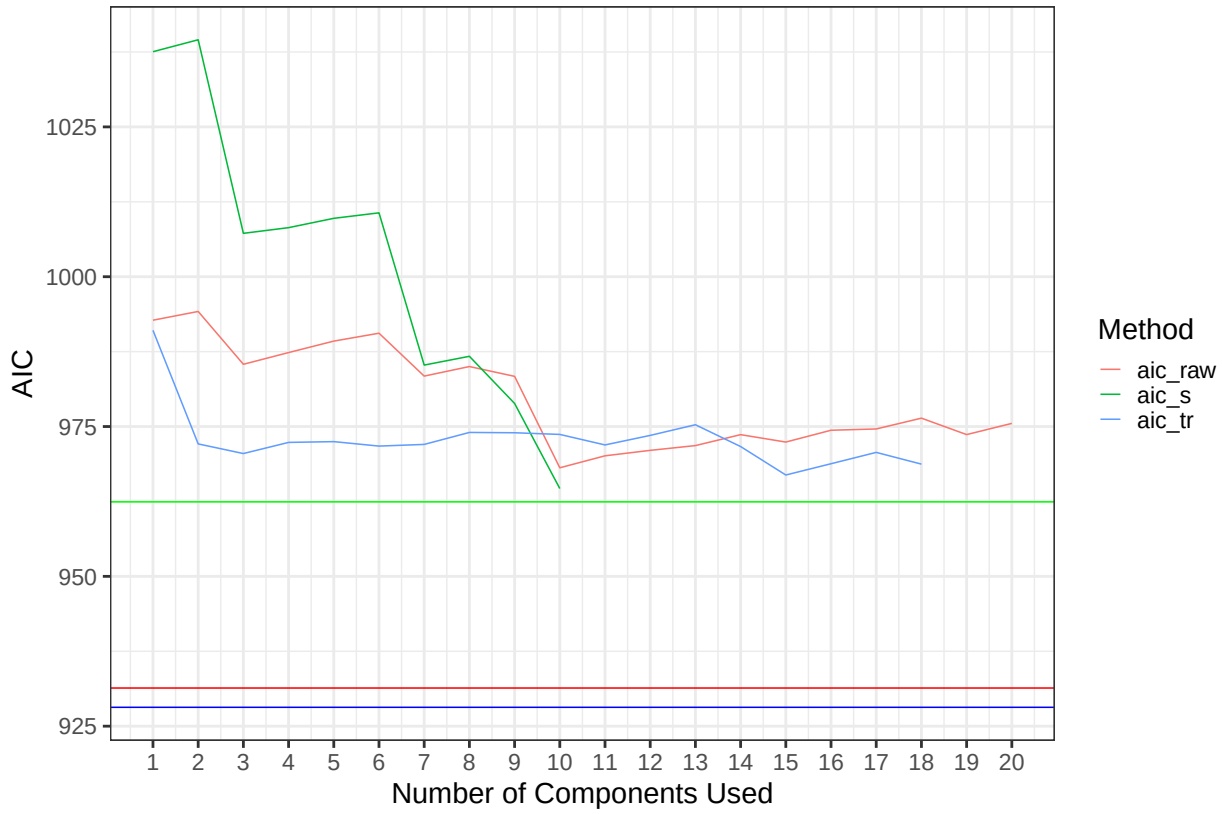

```r
s$loadings
```

```
##
## Loadings:
##      Comp.1 Comp.2 Comp.3 Comp.4 Comp.5 Comp.6 Comp.7 Comp.8 Comp.9 Comp.10
## BM16                                                                1.000
## BM18                                           1.000
## ARM2        -0.707
## BM07                       1.000
## BM15
## BM08
## BM09                                    1.000
## BM10         0.707
## BM11                                                  1.000
## BM12                                                         1.000
## BM13  0.707
## BM02                0.707
## BM17                              1.000
## BM03
## BM04  0.707
## BM01
## BM05                0.707
## BM06
##
##                Comp.1 Comp.2 Comp.3 Comp.4 Comp.5 Comp.6 Comp.7 Comp.8 Comp.9
## SS loadings     1.000  1.000  1.000  1.000  1.000  1.000  1.000  1.000  1.000
## Proportion Var  0.056  0.056  0.056  0.056  0.056  0.056  0.056  0.056  0.056
```

```
## Cumulative Var  0.056  0.111  0.167  0.222  0.278  0.333  0.389  0.444  0.500
##                Comp.10
## SS loadings      1.000
## Proportion Var   0.056
## Cumulative Var   0.556
```

```r
pcs_s <- s$scores
aic_s <- rep(NA, length(aic_raw))
for (i in seq_len(ncol(s$scores))) {
  ps <- pcs_s[, seq_len(i)]
  aic_s[i] <- AIC(lrm(Y ~ ps))
}

# Choose optimal number of components for each method
# Minimal risk of overfitting since selection is performed in prespecified, rigid order
# (see RMS book pg 170)
# We include reference lines for full linear (blue) and spline (red) models
aicpl <- data.frame(x=seq_len(length(aic_raw)), aic_raw = aic_raw, aic_tr = aic_tr, aic_s = aic_s)
aic_long <- pivot_longer(aicpl, starts_with("aic"), names_to = "Method", values_to = "AIC")

ggplot(aic_long, aes(x = x, y = AIC, color = Method)) +
  geom_line() +
  xlab("Number of Components Used") +
  ylab("AIC") +
  scale_x_continuous(breaks = seq_len(nrow(aicpl))) +
  geom_hline(yintercept = AIC_full, color = "blue") +
  geom_hline(yintercept = AIC_spline, color = "red") +
  geom_hline(yintercept = AIC_clust, color = "green")
```

```
## Warning: Removed 12 rows containing missing values (`geom_line()`).
```




```
  # ylim(950, 1050)


# Validate each method to compare performance
# Running pca with 10 components for raw data
ps_raw <- pcs_raw[,seq_len(10)]
f_raw <- lrm(half$Outcome ~ ps_raw, x = T, y = T)

# Running pca with 10 components for transformed data
ps_tr <- pcs_raw[,seq_len(15)]
f_tr <- lrm(half$Outcome ~ ps_tr, x = T, y = T)

# Running pca with 6 components for sparse PCA
ps_s <- pcs_s[, seq_len(6)]
f_s <- lrm(half$Outcome ~ ps_s + ptrans$transformed[, "ARM2"], x = T, y = T)

# Compare performance across methods
validate(f_raw, B = 100)
```

```
##           index.orig training    test optimism index.corrected   n
## Dxy           0.4621   0.4728  0.4489   0.0240          0.4381 100
## R2            0.2143   0.2244  0.2020   0.0224          0.1919 100
## Intercept     0.0000   0.0000 -0.0288   0.0288         -0.0288 100
## Slope         1.0000   1.0000  0.9337   0.0663          0.9337 100
## Emax          0.0000   0.0000  0.0200   0.0200          0.0200 100
## D             0.1700   0.1790  0.1594   0.0196          0.1504 100
## U            -0.0024  -0.0024  0.0009  -0.0033          0.0009 100
```

```
## Q             0.1725   0.1815  0.1585   0.0230          0.1495 100
## B             0.1963   0.1940  0.1990  -0.0049          0.2013 100
## g             1.0760   1.1181  1.0390   0.0791          0.9969 100
## gp            0.2204   0.2245  0.2135   0.0110          0.2094 100
```

```
validate(f_full, B = 100)
```

```
##           index.orig training    test optimism index.corrected   n
## Dxy           0.5635   0.5898  0.5369   0.0528          0.5107 100
## R2            0.3021   0.3302  0.2729   0.0573          0.2448 100
## Intercept     0.0000   0.0000 -0.0558   0.0558         -0.0558 100
## Slope         1.0000   1.0000  0.8576   0.1424          0.8576 100
## Emax          0.0000   0.0000  0.0437   0.0437          0.0437 100
## D             0.2496   0.2769  0.2225   0.0545          0.1951 100
## U            -0.0024  -0.0024  0.0054  -0.0078          0.0054 100
## Q             0.2520   0.2794  0.2171   0.0623          0.1897 100
## B             0.1782   0.1728  0.1842  -0.0114          0.1896 100
## g             1.3953   1.5189  1.2989   0.2199          1.1754 100
## gp            0.2650   0.2769  0.2510   0.0259          0.2391 100
```

```
validate(f_spline, B = 100)
```

```
##           index.orig training    test optimism index.corrected   n
## Dxy           0.6042   0.6556  0.5610   0.0946          0.5096 100
## R2            0.3436   0.4011  0.2959   0.1053          0.2384 100
## Intercept     0.0000   0.0000 -0.1114   0.1114         -0.1114 100
## Slope         1.0000   1.0000  0.7590   0.2410          0.7590 100
## Emax          0.0000   0.0000  0.0812   0.0812          0.0812 100
## D             0.2896   0.3486  0.2438   0.1048          0.1848 100
## U            -0.0024  -0.0024  0.0169  -0.0194          0.0169 100
## Q             0.2920   0.3510  0.2268   0.1242          0.1679 100
## B             0.1716   0.1599  0.1821  -0.0223          0.1939 100
## g             1.5647   1.8309  1.3891   0.4418          1.1229 100
## gp            0.2833   0.3073  0.2612   0.0461          0.2373 100
```

```
validate(f_tr, B = 100)
```

```
##           index.orig training    test optimism index.corrected   n
## Dxy           0.4746   0.4997  0.4556   0.0441          0.4305 100
## R2            0.2216   0.2455  0.2033   0.0423          0.1794 100
## Intercept     0.0000   0.0000 -0.0545   0.0545         -0.0545 100
## Slope         1.0000   1.0000  0.8830   0.1170          0.8830 100
## Emax          0.0000   0.0000  0.0369   0.0369          0.0369 100
## D             0.1764   0.1978  0.1604   0.0374          0.1391 100
## U            -0.0024  -0.0024  0.0026  -0.0051          0.0026 100
## Q             0.1789   0.2002  0.1578   0.0425          0.1364 100
## B             0.1945   0.1900  0.1984  -0.0084          0.2029 100
## g             1.1117   1.2023  1.0506   0.1516          0.9600 100
## gp            0.2253   0.2363  0.2148   0.0215          0.2038 100
```

```
validate(f_s, B = 100)
```

```
##           index.orig training    test optimism index.corrected   n
## Dxy           0.3925   0.4033  0.3791   0.0241          0.3684 100
## R2            0.1446   0.1553  0.1355   0.0198          0.1247 100
## Intercept     0.0000   0.0000 -0.0326   0.0326         -0.0326 100
## Slope         1.0000   1.0000  0.9346   0.0654          0.9346 100
```

```
## Emax          0.0000   0.0000  0.0205   0.0205          0.0205 100
## D             0.1110   0.1201  0.1036   0.0165          0.0945 100
## U            -0.0024  -0.0024  0.0007  -0.0032          0.0007 100
## Q             0.1135   0.1225  0.1029   0.0197          0.0938 100
## B             0.2084   0.2062  0.2106  -0.0044          0.2128 100
## g             0.8440   0.8799  0.8115   0.0684          0.7756 100
## gp            0.1827   0.1880  0.1764   0.0117          0.1711 100
```

```
validate(f_clust, B = 100)
```

```
##           index.orig training    test optimism index.corrected   n
## Dxy           0.4828   0.4981  0.4669   0.0312          0.4516 100
## R2            0.2267   0.2425  0.2119   0.0306          0.1961 100
## Intercept     0.0000   0.0000 -0.0309   0.0309         -0.0309 100
## Slope         1.0000   1.0000  0.9157   0.0843          0.9157 100
## Emax          0.0000   0.0000  0.0247   0.0247          0.0247 100
## D             0.1808   0.1950  0.1679   0.0271          0.1538 100
## U            -0.0024  -0.0024  0.0013  -0.0038          0.0013 100
## Q             0.1833   0.1974  0.1666   0.0309          0.1524 100
## B             0.1929   0.1896  0.1962  -0.0065          0.1995 100
## g             1.1263   1.1853  1.0783   0.1069          1.0194 100
## gp            0.2282   0.2352  0.2203   0.0149          0.2132 100
```

8 Ordinal Regression Case Study with Dimension Reduction

As eluded to earlier, when working with a binary outcome your effective sample size is at best 50% of the effective sample size had the outcome been continuous. It certainly stands to reason that if one has a multinomial outcome it would be inefficient to convert the multinomial outcome into binary. A common example where this is done is with BOR, where analysts will typically take a 4-level outcome and split it into 2 levels (CR/PR vs SD/PD). We provide an example below of an analysis where we keep the outcome in its original 4-level form, thus preserving the efficiency of our analysis.

```
# Turn BOR into ordinal factor
dat_ord$BOR_CONF <- factor(dat_ord$BOR_CONF, ordered = T, levels = c("PD", "SD", "PR", "CR"))



# Describe dataset
# options(contrasts=c("contr.treatment", "contr.treatment"))
ddist<- datadist(dat_ord, adjto.cat = "first")
options(datadist='ddist')

describe(dat_ord)
```

```
## dat_ord
##
##  19  Variables      930  Observations
## ---------------------------------------------------------------------------
## BM07
##        n  missing distinct
##      930        0        3
##
## Value      Type A Type B Type C
## Frequency     490    308    132
## Proportion  0.527  0.331  0.142
```

```
## ---------------------------------------------------------------------------
## BM06
##        n  missing distinct     Info     Mean      Gmd      .05      .10
##      930        0       70    0.999    58.91    15.38       34       40
##      .25      .50      .75      .90      .95
##       50       60       69       75       79
##
## lowest : 18 19 21 23 24, highest: 85 86 87 88 89
## ---------------------------------------------------------------------------
## BM05
##        n  missing distinct
##      910       20        4
##
## Value      Type A Type B Type C Type D
## Frequency      37    141    196    536
## Proportion  0.041  0.155  0.215  0.589
## ---------------------------------------------------------------------------
## BM08
##        n  missing distinct     Info     Mean      Gmd      .05      .10
##      930        0      196        1    68.25    55.38    12.00    16.00
##      .25      .50      .75      .90      .95
##    30.25    52.00    89.75   140.00   181.10
##
## lowest :  10  11  12  13  14, highest: 278 283 321 339 372
## ---------------------------------------------------------------------------
## BM09
##        n  missing distinct     Info     Mean      Gmd      .05      .10
##      910       20      212        1    28.02    5.694    20.90    22.29
##      .25      .50      .75      .90      .95
##    24.30    27.40    31.10    34.71    37.00
##
## lowest : 12.6 17   17.2 17.3 17.8, highest: 45.3 50.7 51.1 53.4 54
## ---------------------------------------------------------------------------
## BM17
##        n  missing distinct
##      893       37        2
##
## Value         NO   YES
## Frequency    606   287
## Proportion 0.679 0.321
## ---------------------------------------------------------------------------
## BM11
##        n  missing distinct     Info     Mean      Gmd      .05      .10
##      929        1       89        1    136.2    18.48      105      113
##      .25      .50      .75      .90      .95
##      126      138      148      155      160
##
## lowest :  77  78  84  87  90, highest: 170 171 175 180 187
## ---------------------------------------------------------------------------
## BM13
##        n  missing distinct     Info     Mean      Gmd      .05      .10
##      929        1      307        1    258.1    89.86    154.0    171.0
##      .25      .50      .75      .90      .95
##    201.0    241.0    298.0    367.2    409.6
```

```
##
## lowest :  90 106 107 113 117, highest: 626 639 654 673 708
## ---------------------------------------------------------------------------
## BM12
##        n  missing distinct     Info     Mean      Gmd      .05      .10
##      928        2      536        1    1.558   0.6948   0.6900   0.8494
##      .25      .50      .75      .90      .95
##   1.1180   1.4805   1.8885   2.3378   2.6598
##
## lowest : 0.16  0.272 0.32  0.373 0.377, highest: 4.1   4.77  4.896 5.612 5.98
## ---------------------------------------------------------------------------
## BM04
##        n  missing distinct     Info     Mean      Gmd      .05      .10
##      929        1      686        1    4.646    2.135    2.270    2.667
##      .25      .50      .75      .90      .95
##    3.310    4.188    5.330    6.860    8.236
##
## lowest : 1.427  1.596  1.652  1.7    1.741 , highest: 17     17.1   18.9   26.877 51.504
## ---------------------------------------------------------------------------
## BM02
##        n  missing distinct     Info      Sum     Mean      Gmd
##      930        0        2    0.614      267   0.2871   0.4098
##
## ---------------------------------------------------------------------------
## ARM2
##        n  missing distinct
##      930        0        4
##
## Value      ARM A ARM B ARM C ARM D
## Frequency    234    22   423   251
## Proportion 0.252 0.024 0.455 0.270
## ---------------------------------------------------------------------------
## BM03
##        n  missing distinct
##      930        0        2
##
## Value         NO   YES
## Frequency    315   615
## Proportion 0.339 0.661
## ---------------------------------------------------------------------------
## BM10
##        n  missing distinct     Info     Mean      Gmd      .05      .10
##      803      127      253        1    140.1    166.9     22.0     29.0
##      .25      .50      .75      .90      .95
##     44.0     65.0    114.0    259.6    392.6
##
## lowest :    2    4    6    7    8, highest: 2480 2681 2702 2747 3957
## ---------------------------------------------------------------------------
## BM18
##        n  missing distinct     Info     Mean      Gmd      .05      .10
##      918       12      386        1    2.406   0.2563    2.134    2.170
##      .25      .50      .75      .90      .95
##    2.236    2.334    2.535    2.720    2.886
##
```

```
## lowest : 1.9345  1.93952 1.99123 2.02531 2.04139
## highest: 3.35295 3.35526 3.35622 3.36624 3.39811
## ---------------------------------------------------------------------------
## BM15
##        n  missing distinct     Info      Sum     Mean      Gmd
##      930        0        2     0.47      181   0.1946   0.3138
##
## ---------------------------------------------------------------------------
## BM01
##        n  missing distinct
##      927        3        2
##
## Value          0   >=1
## Frequency    693   234
## Proportion 0.748 0.252
## ---------------------------------------------------------------------------
## BOR_CONF
##        n  missing distinct
##      930        0        4
##
## Value         PD    SD    PR    CR
## Frequency    297   143   312   178
## Proportion 0.319 0.154 0.335 0.191
## ---------------------------------------------------------------------------
## ARM
##        n  missing distinct
##      930        0        3
##
## Value      ARM A ARM C ARM D
## Frequency    234   445   251
## Proportion 0.252 0.478 0.270
## ---------------------------------------------------------------------------
```

```
# Here we apply single imputation due to minimal missing
ord_scan <- transcan(~ BM18 + ARM + BM07 + BM15 +
                            BM08 + BM09 + BM10 + BM11 + BM12 + BM13 +
                            BM02 + BM03 + BM04 + BM01 + BM05 + BM06,
                imputed = T, transformed = T, trantab = T, pl = F, show.na = T, data = dat_ord,
                n.impute = 1, pr = F)
```

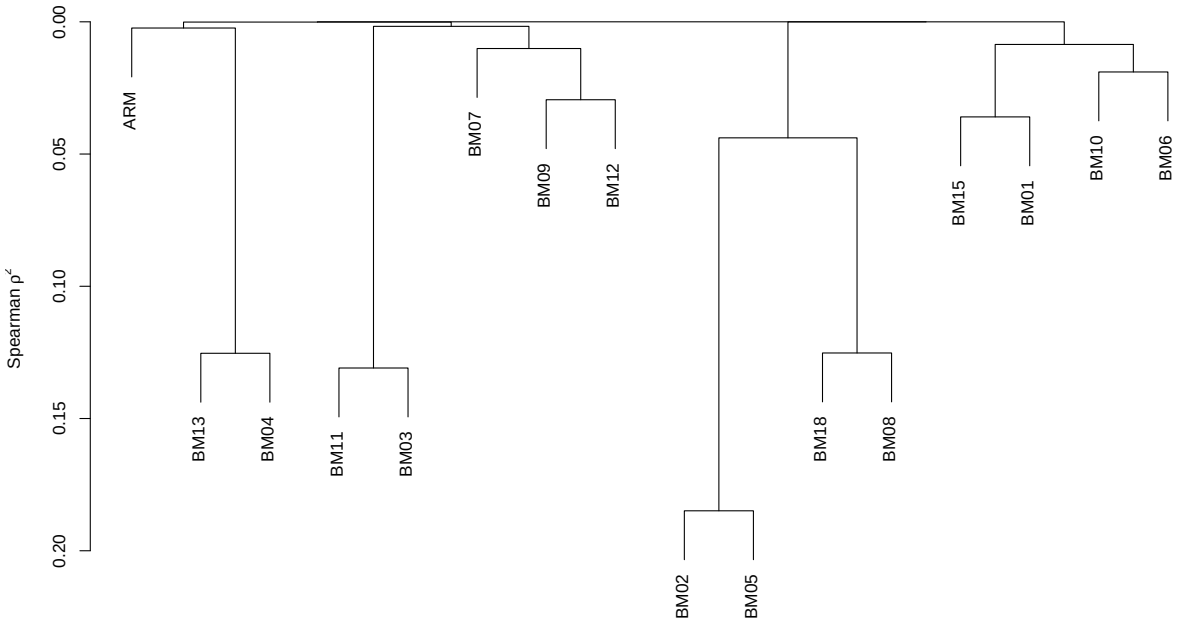
```
## Warning in transcan(~BM18 + ARM + BM07 + BM15 + BM08 + BM09 + BM10 + BM11 + : transcan provides only
## A better approximation is provided by the aregImpute function.
## The MICE and other S libraries provide imputations from Bayesian posterior distributions.
```

```
ord_imp <- data.frame(impute.transcan(ord_scan, imputation = 1,  data = dat_ord, list.out = T))
```

```
##
##
## Imputed missing values with the following frequencies
##  and stored them in variables with their original names:
##
## BM18 BM09 BM10 BM11 BM12 BM13 BM04 BM01 BM05
##   12   20  127    1    2    1    1    3   20
```

```
BOR <- dat_ord$BOR_CONF
```

```
ord_imp <- data.frame(ord_imp, BOR = BOR)

# Plot clusters
vclust <- varclus(ord_scan$transformed, data = ord_imp)

plot(vclust)
```




```
# Make clusters
# 1) BM02, BM05
# 2) BM18, BM08
# 3) BM13, BM04
# 4) BM11, BM03
# 5) Other

# Calculate 1st princ comp from each cluster (except other)
cluster1 <- pco(v = c("BM02", "BM05"), data_tr = ord_scan$transformed)
```

```
## Fraction of variance explained by PC1: 0.72
##
```

```
cluster2 <- pco(v = c("BM18", "BM08"), data_tr = ord_scan$transformed)
```

```
## Fraction of variance explained by PC1: 0.71
##
```

```
cluster3 <- pco(v = c("BM13", "BM04"), data_tr = ord_scan$transformed)
```

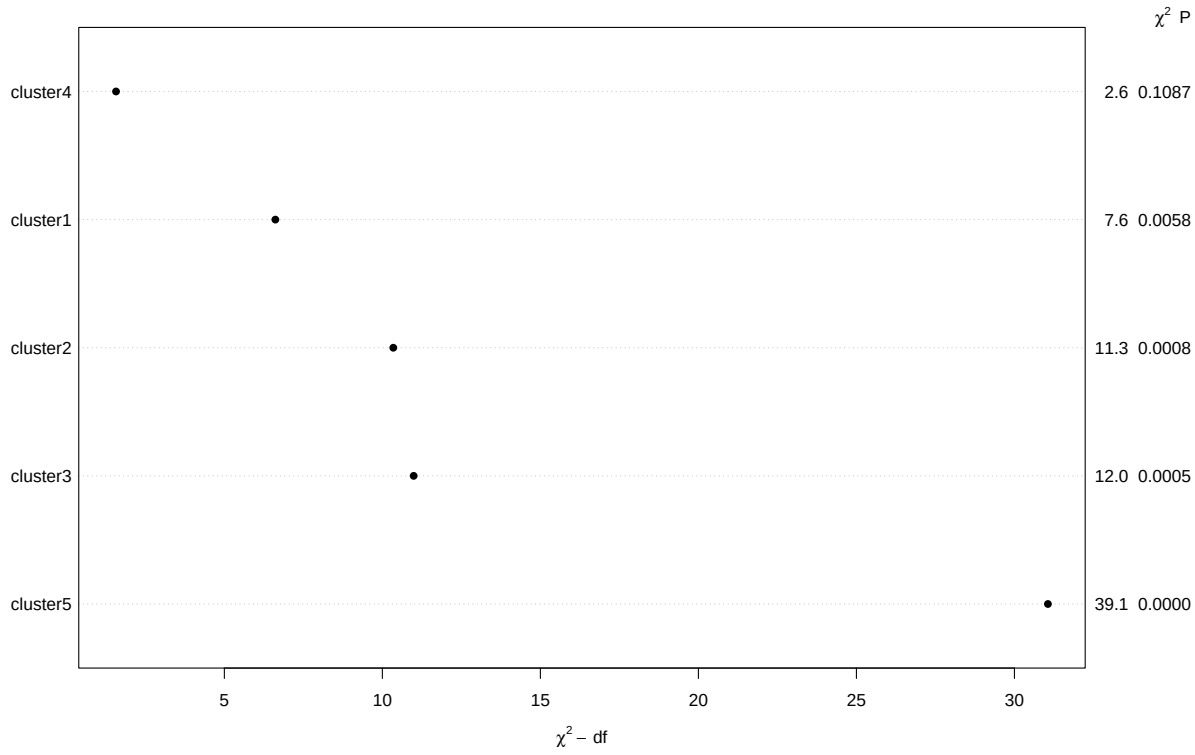
```
## Fraction of variance explained by PC1: 0.7
```

```
##
```

```
cluster4 <- pco(v = c("BM11", "BM03"), data_tr = ord_scan$transformed)
```

```
## Fraction of variance explained by PC1: 0.67
##
```

```
cluster5 <- ord_scan$transformed[,c("ARM", "BM12", "BM15", "BM01", "BM09", "BM06", "BM07", "BM10")]

ddist<- datadist(ord_imp, adjto.cat = "first")
options(datadist='ddist')

# Run ordinal logistic regression model
mod_ord <- lrm(BOR ~ cluster1 + cluster2 + cluster3 + cluster4 + cluster5, data = ord_imp)


plot(anova(mod_ord))
```




```
print(mod_ord)
```

```
## Logistic Regression Model
##
## lrm(formula = BOR ~ cluster1 + cluster2 + cluster3 + cluster4 +
##     cluster5, data = ord_imp)
##
##
## Frequencies of Responses
##
```

```
##  PD  SD  PR  CR
## 297 143 312 178
##
##                         Model Likelihood       Discrimination    Rank Discrim.
##                               Ratio Test              Indexes          Indexes
## Obs           930    LR chi2     116.20       R2       0.126    C       0.654
## max |deriv| 2e-10    d.f.            12      R2(12,930)0.106    Dxy     0.308
##                      Pr(> chi2) <0.0001    R2(12,854.7)0.115    gamma   0.308
##                                               Brier    0.226    tau-a   0.223
##
##          Coef    S.E.   Wald Z Pr(>|Z|)
## y>=SD     0.7328 0.0791   9.26 <0.0001
## y>=PR     0.0108 0.0752   0.14 0.8862
## y>=CR    -1.6591 0.0945 -17.56 <0.0001
## cluster1 -0.1519 0.0551  -2.76 0.0058
## cluster2 -0.2026 0.0602  -3.37 0.0008
## cluster3  0.2058 0.0594   3.46 0.0005
## cluster4  0.0880 0.0549   1.60 0.1087
## ARM       0.2505 0.0612   4.09 <0.0001
## BM12      0.1015 0.0647   1.57 0.1168
## BM15      0.3555 0.1569   2.27 0.0235
## BM01     -0.0457 0.0652  -0.70 0.4829
## BM09      0.1059 0.0630   1.68 0.0927
## BM06      0.0720 0.0643   1.12 0.2630
## BM07      0.1464 0.0620   2.36 0.0181
## BM10      0.0544 0.0591   0.92 0.3575
```

```
# p_ord <- Predict(mod_ord)
```

9 Visualizing Results

Now that we have our model, the next step is to provide helpful visualizations of the results. In particular we recommend 3 helpful visualizations:

1. Anova plot - Useful for ranking strengths of association amongst predictors. By default the X-axis is the $\chi^2 - df$ which should equal 0 under $H_0 : \beta = 0$. This provides more information than the p-value since it gives a greater sense of distinction between "significant" predictors. See '?plot.anova.rms' for more information.
2. Partial Effects plot - Useful for visualizing the direction and form of the relationship between each predictor and outcome while adjusting for all other predictors. By default all other continuous predictors are set to median value and categorical predictors are set to their reference value. See '?ggplot.Predict' for more information.
3. Nomogram - Provides high level overview of relationships between variables and response. Allows you to convert the scale from "prediction score" to probability of response on an individual patient level. Clinicians tend to have strong preference for this visualization. See '?rms.nomogram' for more information.

We illustrate these visualizations below using the model built on the multiply imputed dataset from above.

Note that we use `fit.mult.impute` to run the model on the multiple imputed data. We include `fitter = lrm` as an argument telling the function to perform logistic regression. We also include `xtrans = MI5` telling it to use the `MI5` object we created from `aregimpute` above (section 3.3). Finally, we include `data = mod_dat` telling the function which dataset we used to derive the `xtrans` object.

```r
ddist<- datadist(mod_dat, adjto.cat = "first")
options(datadist='ddist')

# For the multiple imputed data, we use fit.mult.impute
sat_mod <- fit.mult.impute(Outcome ~ rcs(BM16, 4) +  rcs(BM18, 4) + ARM2 + BM07 + BM15 +
                             rcs(BM08, 4) + rcs(BM09, 4) + rcs(BM10, 4) + rcs(BM11, 4) +
                             rcs (BM12, 4) + rcs(BM13, 4) +
                             BM02 + BM17 + BM03 + rcs(BM04, 4) + BM01 + BM05 + rcs(BM06, 4) +
                             BM16 %ia% ARM2 + BM18 %ia% ARM2,
                           fitter = lrm, xtrans = MI5, data = mod_dat)
```

```
##
## Wald Statistic Information
##
## Variance Inflation Factors Due to Imputation:
##
##         Intercept              BM16             BM16'              BM18
##              1.01              1.39              1.46              1.00
##             BM18'            BM18''        ARM2=ARM B        ARM2=ARM C
##              1.00              1.00              1.02              1.01
##        ARM2=ARM D       BM07=Type B       BM07=Type C              BM15
##              1.01              1.01              1.00              1.00
##              BM08             BM08'            BM08''              BM09
##              1.02              1.01              1.01              1.01
##             BM09'            BM09''              BM10             BM10'
##              1.03              1.03              1.28              1.31
##            BM10''              BM11             BM11'            BM11''
##              1.31              1.01              1.01              1.01
##              BM12             BM12'            BM12''              BM13
##              1.01              1.00              1.00              1.01
##             BM13'            BM13''              BM02          BM17=YES
##              1.01              1.01              1.02              1.05
##          BM03=YES              BM04             BM04'            BM04''
##              1.02              1.01              1.01              1.01
##          BM01=>=1       BM05=Type B       BM05=Type C       BM05=Type D
##              1.01              1.01              1.01              1.05
##              BM06             BM06'            BM06'' BM16 * ARM2=ARM B
##              1.01              1.00              1.00              1.10
## BM16 * ARM2=ARM C BM16 * ARM2=ARM D BM18 * ARM2=ARM B BM18 * ARM2=ARM C
##              1.05              1.04              1.02              1.01
## BM18 * ARM2=ARM D
##              1.01
##
## Fraction of Missing Information:
##
##         Intercept              BM16             BM16'              BM18
##              0.01              0.28              0.31              0.00
##             BM18'            BM18''        ARM2=ARM B        ARM2=ARM C
##              0.00              0.00              0.02              0.01
##        ARM2=ARM D       BM07=Type B       BM07=Type C              BM15
##              0.01              0.01              0.00              0.00
##              BM08             BM08'            BM08''              BM09
##              0.02              0.01              0.01              0.01
##             BM09'            BM09''              BM10             BM10'
```

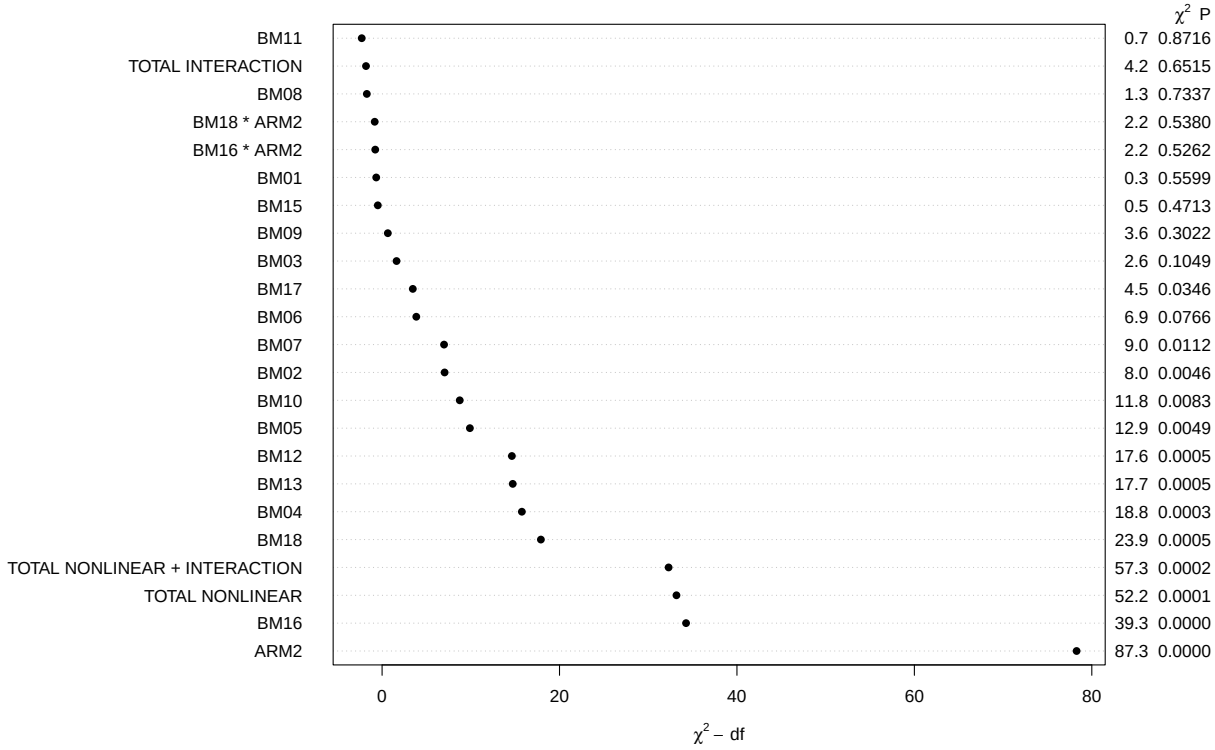
```
##             0.03             0.03             0.22             0.24
##           BM10''             BM11            BM11'           BM11''
##             0.24             0.01             0.01             0.01
##             BM12            BM12'           BM12''             BM13
##             0.01             0.00             0.00             0.01
##            BM13'           BM13''             BM02         BM17=YES
##             0.01             0.01             0.02             0.05
##         BM03=YES             BM04            BM04'           BM04''
##             0.02             0.01             0.01             0.01
##         BM01=>=1      BM05=Type B      BM05=Type C      BM05=Type D
##             0.01             0.01             0.01             0.05
##             BM06            BM06'           BM06'' BM16 * ARM2=ARM B
##             0.01             0.00             0.00             0.09
## BM16 * ARM2=ARM C BM16 * ARM2=ARM D BM18 * ARM2=ARM B BM18 * ARM2=ARM C
##             0.05             0.04             0.02             0.01
## BM18 * ARM2=ARM D
##             0.01
##
## d.f. for t-distribution for Tests of Single Coefficients:
##
##         Intercept             BM16            BM16'             BM18
##          98682.84            50.75            40.42        491088.20
##            BM18'           BM18''       ARM2=ARM B       ARM2=ARM C
##         507805.60        339441.44         14532.62         42084.62
##        ARM2=ARM D      BM07=Type B      BM07=Type C             BM15
##          41970.08        103973.61       1067072.19        208756.46
##             BM08            BM08'           BM08''             BM09
##          15580.42         41806.45         52717.42         18463.04
##            BM09'           BM09''             BM10            BM10'
##           4954.45          4045.00            85.51            70.73
##           BM10''             BM11            BM11'           BM11''
##            70.21         86121.71         49775.77         24081.69
##            BM12            BM12'           BM12''             BM13
##          82485.72        278270.49        312538.46         18629.32
##            BM13'           BM13''             BM02         BM17=YES
##          29781.63         40519.28          9552.06          1636.17
##          BM03=YES             BM04            BM04'           BM04''
##          15760.25         79824.79         66111.69         70425.33
##          BM01=>=1      BM05=Type B      BM05=Type C      BM05=Type D
##         119913.76         48886.27         28142.45          1754.19
##             BM06            BM06'           BM06'' BM16 * ARM2=ARM B
##          95835.27        201006.18        249787.19           465.22
## BM16 * ARM2=ARM C BM16 * ARM2=ARM D BM18 * ARM2=ARM B BM18 * ARM2=ARM C
##           1575.30          3244.42         10912.08         27568.83
## BM18 * ARM2=ARM D
##          42705.57
##
## The following fit components were averaged over the 5 model fits:
##
##   stats linear.predictors
```

```
# ANOVA Plot - We can now visualize the results including the test for non-linearity
# to convince our collaborators that it was worth allowing for BUT NOT to influence
# our decision whether or not to include non-linear terms
```

```r
a_sat <- anova(sat_mod)

# You can also visualize the anova output in text form
# This is much more clear (and useful) of a summary than that provided by summary(model)
print(a_sat)
```

```
##                 Wald Statistics          Response: Outcome
##
##  Factor                                     Chi-Square d.f. P
##  BM16  (Factor+Higher Order Factors)         39.26      5   <.0001
##   All Interactions                            2.23      3   0.5262
##   Nonlinear                                  21.20      1   <.0001
##  BM18  (Factor+Higher Order Factors)         23.90      6   0.0005
##   All Interactions                            2.17      3   0.5380
##   Nonlinear                                   0.58      2   0.7492
##  ARM2  (Factor+Higher Order Factors)         87.29      9   <.0001
##   All Interactions                            4.19      6   0.6515
##  BM07                                         8.99      2   0.0112
##  BM15                                         0.52      1   0.4713
##  BM08                                         1.28      3   0.7337
##   Nonlinear                                   0.78      2   0.6774
##  BM09                                         3.65      3   0.3022
##   Nonlinear                                   1.90      2   0.3868
##  BM10                                        11.75      3   0.0083
##   Nonlinear                                   6.96      2   0.0309
##  BM11                                         0.71      3   0.8716
##   Nonlinear                                   0.34      2   0.8440
##  BM12                                        17.63      3   0.0005
##   Nonlinear                                   8.96      2   0.0113
##  BM13                                        17.73      3   0.0005
##   Nonlinear                                   4.84      2   0.0888
##  BM02                                         8.05      1   0.0046
##  BM17                                         4.46      1   0.0346
##  BM03                                         2.63      1   0.1049
##  BM04                                        18.76      3   0.0003
##   Nonlinear                                   3.85      2   0.1459
##  BM01                                         0.34      1   0.5599
##  BM05                                        12.90      3   0.0049
##  BM06                                         6.86      3   0.0766
##   Nonlinear                                   0.37      2   0.8308
##  BM16 * ARM2  (Factor+Higher Order Factors)   2.23      3   0.5262
##  BM18 * ARM2  (Factor+Higher Order Factors)   2.17      3   0.5380
##  TOTAL NONLINEAR                             52.18     19   0.0001
##  TOTAL INTERACTION                            4.19      6   0.6515
##  TOTAL NONLINEAR + INTERACTION               57.30     25   0.0002
##  TOTAL                                      293.38     48   <.0001
```

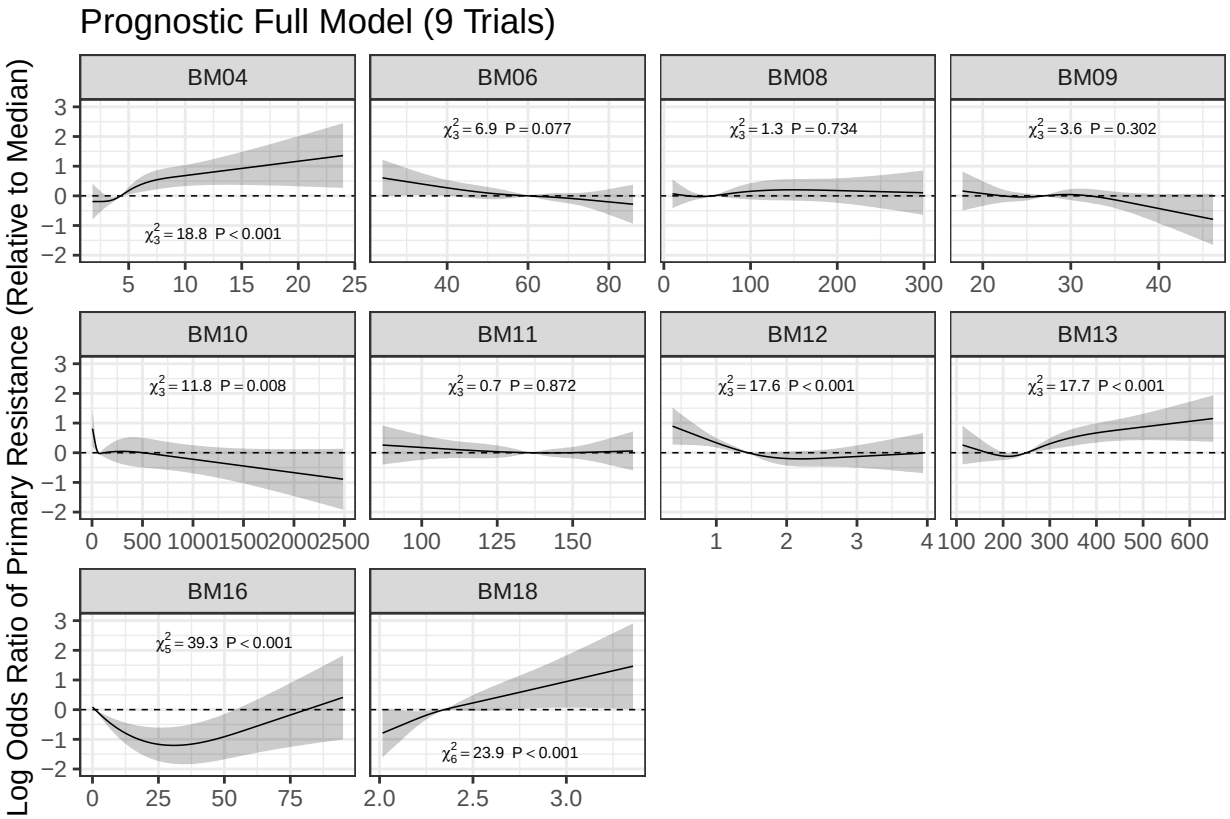
```r
print(plot(a_sat, rm.totals = F, rm.other = "TOTAL"))
```



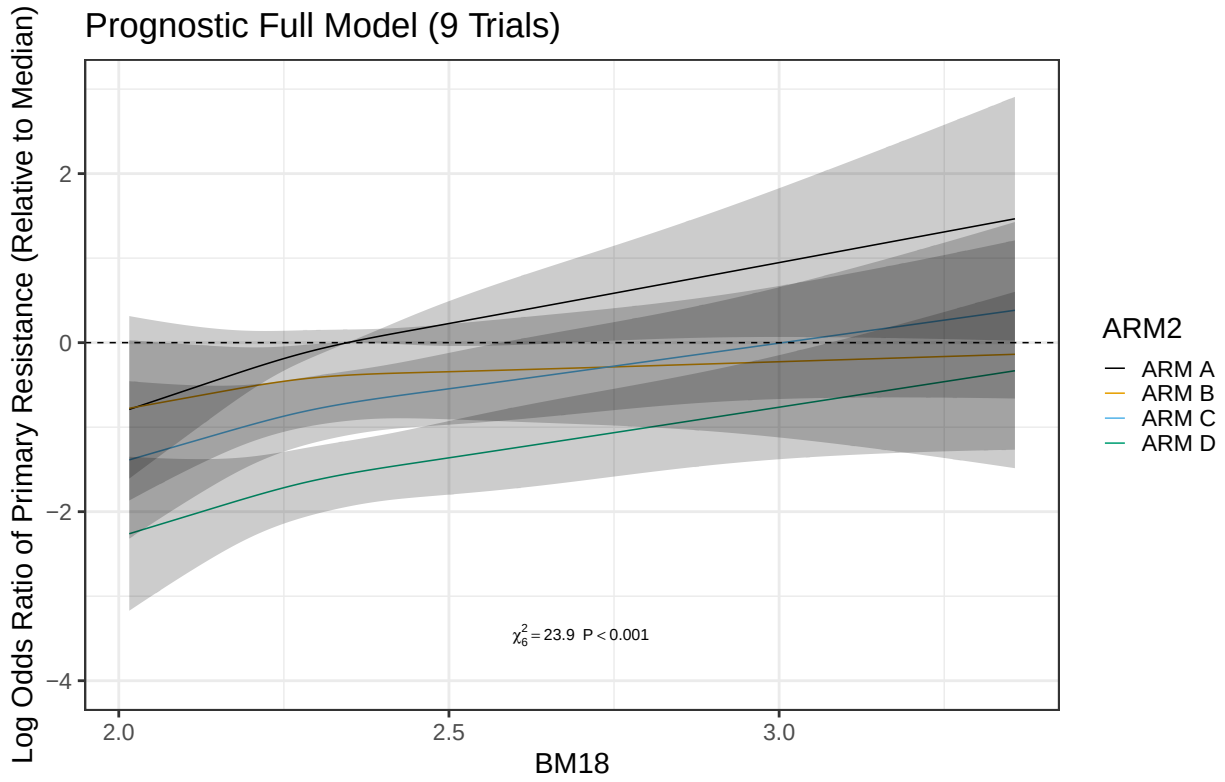

```
##                       ARM2                       BM16
##                 78.2860773                 34.2637891
##  TOTAL NONLINEAR TOTAL NONLINEAR + INTERACTION
##                 33.1822996                 32.2972891
##                       BM18                       BM04
##                 17.9006873                 15.7559944
##                       BM13                       BM12
##                 14.7280774                 14.6303820
##                       BM05                       BM10
##                  9.8977399                  8.7541921
##                       BM02                       BM07
##                  7.0478362                  6.9857267
##                       BM06                       BM17
##                  3.8568564                  3.4626822
##                       BM03                       BM09
##                  1.6298833                  0.6471221
##                       BM15                       BM01
##                 -0.4811718                 -0.6600468
##                  BM16 * ARM2                BM18 * ARM2
##                 -0.7705008                 -0.8305007
##                       BM08           TOTAL INTERACTION
##                 -1.7193835                 -1.8136168
##                       BM11
##                 -2.2929858
```

```
# Partial Effects Plot - We use ref.zero = T to plot the log(OR) relative to median
# instead of relative odds for each predictor.
# This therefore constrains each facet to have log(OR) = 0 at the median for each predictor.
```

```
# Due to the many predictors, we also separate the continuous from the categorical
# predictors into 2 distinct plots via setting sepdiscrete = "list"
# This plot uses ggplot, so we can customize it via regular ggplot sintax as illustrated below
pred_red <- Predict(sat_mod, ref.zero = T)
p_pe_red <- ggplot(pred_red, vnames = "names", sepdiscrete = "list", anova = a_sat, pval = T)

# Continuous partial effects plot
p_pe_red_cont <- p_pe_red$continuous +
ggtitle("Prognostic Full Model (9 Trials)") +
geom_hline(yintercept = 0, linetype = "dashed") +
ylab("Log Odds Ratio of Primary Resistance (Relative to Median)")

print(p_pe_red_cont)
```

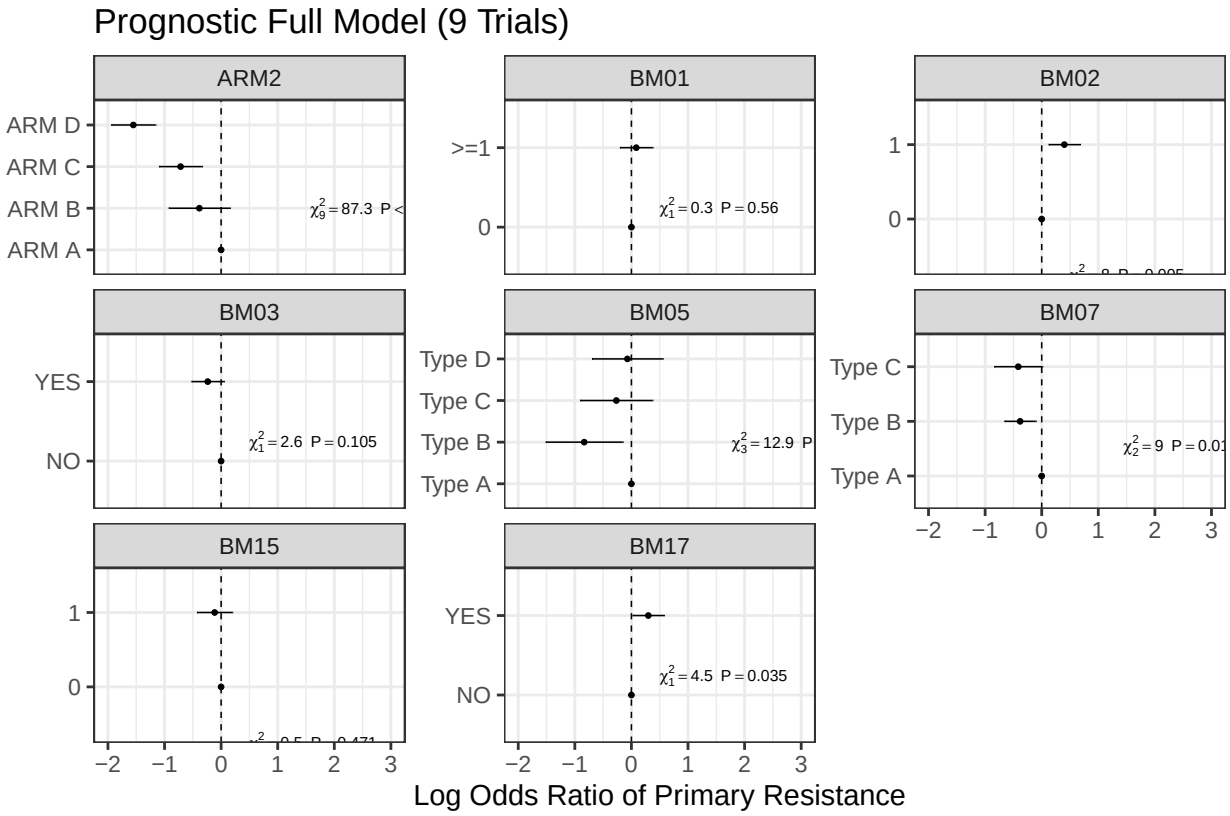



```
# Example of Partial Effects Plot with Interaction
# Here we allowed for interaction between BM18 and Arm
# so we view that plot individually (since not all terms in model have interactions)
pred_intBM18 <- Predict(sat_mod, BM18, ARM2, ref.zero = T)
p_pe_BM18 <- ggplot(pred_intBM18, vnames = "names", anova = a_sat, pval = T)

p_pe_BM18 <- p_pe_BM18 +
ggtitle("Prognostic Full Model (9 Trials)") +
geom_hline(yintercept = 0, linetype = "dashed") +
ylab("Log Odds Ratio of Primary Resistance (Relative to Median)")

print(p_pe_BM18)
```

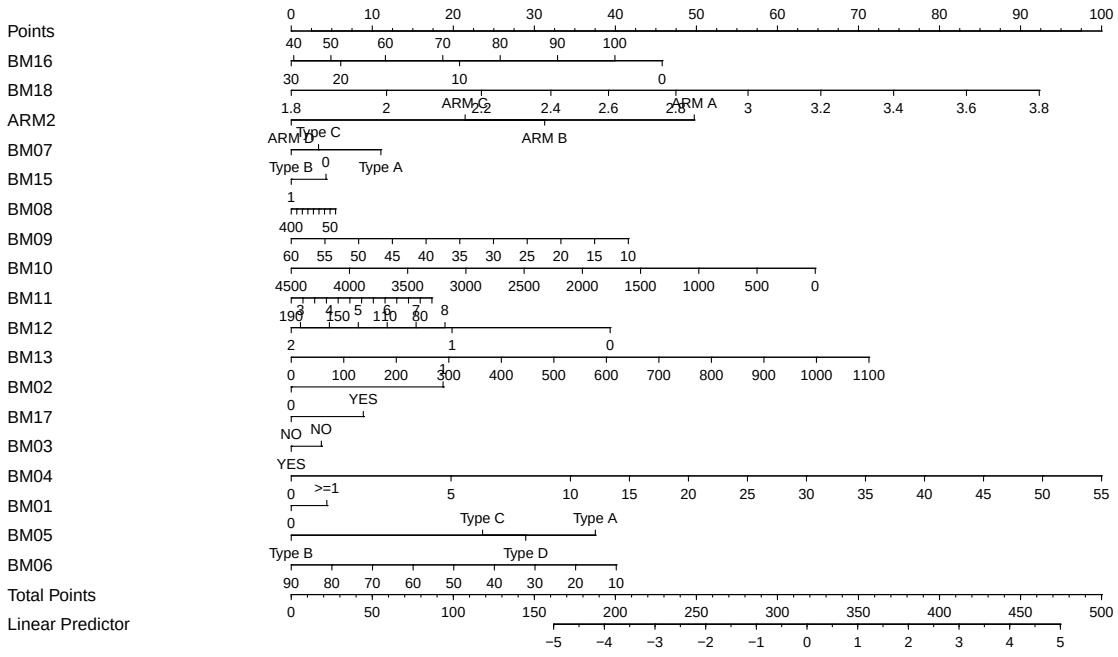


BM12=1.45 BM13=250 BM02=0 BM17=NO BM03=NO BM04=4.292 BM01=0 BM05=Type A BM06=60

```
# Categorical partial effects plot
p_pe_red_disc <- p_pe_red$discrete +
  ggtitle("Prognostic Full Model (9 Trials)") +
  geom_vline(xintercept = 0, linetype = "dashed") +
  xlab("Log Odds Ratio of Primary Resistance")

print(p_pe_red_disc)
```




```
# Partial Effects Plot with interaction


# Nomogram
plot(nomogram(red_mod))
```



10 Performance Evaluation via Internal Validation

It is of utmost importance to assess the performance of your model in a generalizable, robust way. That means not to simply use the performance metrics from the data you used to train your model. In an ideal world, you would have an external dataset that became available just as you finished your analysis for you to validate your model on (if it became available before you finished your analysis it would be preferable to use all available data in training your model). This would be known as external validation and is the gold standard of validation. However, considering that is not the case, in general we recommend one of two ways to perform internal validation:

1. Optimism-adjusted bootstrap
2. Repeated Cross-validation (CV)

The following is the algorithm used in the optimism-adjusted bootstrap:

1. Run full model on all of data
2. Determine performance metrics on full model
3. Draw bootstrap sample (with replacement) Build model on bootstrap sample and Calculate "apparent" performance on same bootstrap sample
4. Apply model from bootstrap sample on original (full) data
5. Calculate Optimism by subtracting difference between performance metric Calculated in step 3 from that Calculated in step 4
6. After repeating steps 3 – 5 many times, Calculate average amount of optimism

7. Subtract average optimism from performance metric Calculated in step 2

Although intuitively there is some leakage in step 4 since on average 63.2% of data from full model is in each bootstrap sample as well, nonetheless both Harrell Jr (2015) and Steyerberg et al. (2019) assert this method of internal validation has strong basis both in theory and based on many simulations. There are also 2 reasons to prefer this method over repeated CV:

1. It's easier to validate with imputed data
2. It uses the full dataset and therefore you can expect a more realistic estimate to true model performance whereas the estimate from repeated CV is a conservative one since it's based on a model built from only a fraction of the data

Combining internal validation with multiple imputation is a challenging problem. Here we take the approach of Steyerberg et al. (2019) to perform an optimism-adjusted bootstrap for each imputed dataset. We then report the mean and distribution of the performance metrics across all imputed datasets.

The `rms` package makes it simple to implement the optimism-adjusted bootstrap. We demonstrate below.

```
# First we set a seed
set.seed(4155)

# This can be time consuming, so we save our output as an rds object
if(!file.exists(file.path(results, "PubDcomplete9.rds"))) {
full <- list()
mod <- list()
val <- list()
for (i in seq_len(MI5$n.impute)) {

  # Pull out each imputed dataset from our MI5 object. See ?impute.transcan for more details.
full[[i]] <- as.data.frame(impute.transcan(MI5, imputation=i, data=mod_dat,
                                           list.out=TRUE, pr=FALSE, check=FALSE))

# As above, we tell the rms package the distribution of our data
ddist<- datadist(full[[i]], adjto.cat = "first")
options(datadist='ddist')

# We run the same model as above, however this time via the lrm function since we are
# looking at only one dataset at a time instead of all of them together
# We set x = T, y = T so that the model object will include the matrix of predictors (x)
# and response (y), this is necessary for the validation steps which follow
mod[[i]] <- lrm(Outcome ~ rcs(BM16, 4) +  rcs(BM18, 4) + ARM2 + BM07 + BM15 +
                  rcs(BM08, 4) + rcs(BM09, 4) + rcs(BM10, 4) + rcs(BM11, 4) +
                  rcs (BM12, 4) + rcs(BM13, 4) +
                  BM02 + BM17 + BM03 + rcs(BM04, 4) + BM01 + BM05 + rcs(BM06, 4) +
                  BM16 %ia% ARM2 + BM18 %ia% ARM2, data = full[[i]], x = T, y = T)

# Calculate various performance metrics for our model via optimism-adjusted bootstrap.
# For illustration purposes we set B = 50 bootstrap samples.
# In practice it may be preferable to set B to a higher number, e.g. B = 500.
val[[i]] <- validate(mod[[i]], B = 50)
}
complete9 <- list(full = full, mod = mod, val = val)
saveRDS(complete9, file.path(results, "PubDcomplete9.rds"))
} else {
  complete9 <- readRDS(file.path(results, "PubDcomplete9.rds"))
```

```
  # "full" is a list of the individual imputed datasets used in validation
  full <- complete9$full

  # "mod" is a list of model outputs corresponding to each dataset
  mod <- complete9$mod

  # "val" is a list of performance metric summaries corresponding to each dataset
  val <- complete9$val
}

# View a sample output from the validate function
print(val[[1]])
```

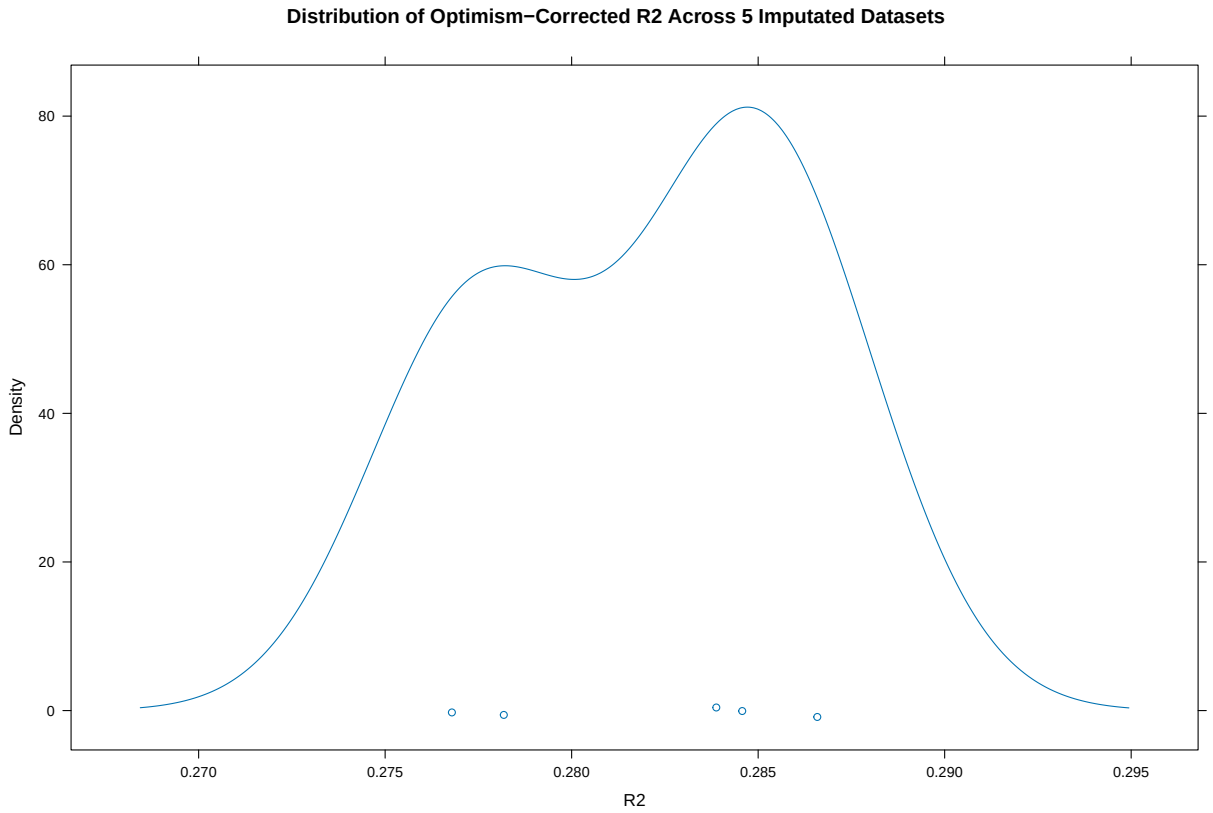
```
##           index.orig training    test optimism index.corrected  n
## Dxy           0.5991   0.6273  0.5729   0.0545          0.5446 50
## R2            0.3383   0.3693  0.3092   0.0601          0.2782 50
## Intercept     0.0000   0.0000 -0.0760   0.0760         -0.0760 50
## Slope         1.0000   1.0000  0.8550   0.1450          0.8550 50
## Emax          0.0000   0.0000  0.0480   0.0480          0.0480 50
## D             0.2823   0.3132  0.2546   0.0585          0.2238 50
## U            -0.0012  -0.0012  0.0046  -0.0058          0.0046 50
## Q             0.2835   0.3144  0.2500   0.0643          0.2192 50
## B             0.1684   0.1628  0.1744  -0.0116          0.1799 50
## g             1.5437   1.6804  1.4383   0.2421          1.3016 50
## gp            0.2760   0.2890  0.2629   0.0261          0.2499 50
```

We now have created `val` a list of performance metric summaries corresponding to each dataset. we now illustrate below how to pull out any individual performance metric and show its distribution across the multiple imputed datasets. This serves as a sort of sensitivity analysis to see how much the performance varies across imputed datasets. We will illustrate with R^2. We also like to see the estimate for optimism to get a sense how overfit the model is.

```
R2 <- data.frame(Optimism = rep(NA, MI5$n.impute), Corrected = rep(NA, MI5$n.impute))
for(i in seq_len(MI5$n.impute)) {
  R2$Corrected[i] <- val[[i]]["R2", "index.corrected"]
  R2$Optimism[i] <- val[[i]]["R2", "optimism"]
}
median(R2$Corrected)
```

```
## [1] 0.2838783
```

```
print(densityplot(R2$Corrected, xlab = "R2",
            main = "Distribution of Optimism-Corrected R2 Across 5 Imputated Datasets"))
```




```
median(R2$Optimism)
```

```
## [1] 0.05974982
```

11 Cox Regression with Interaction

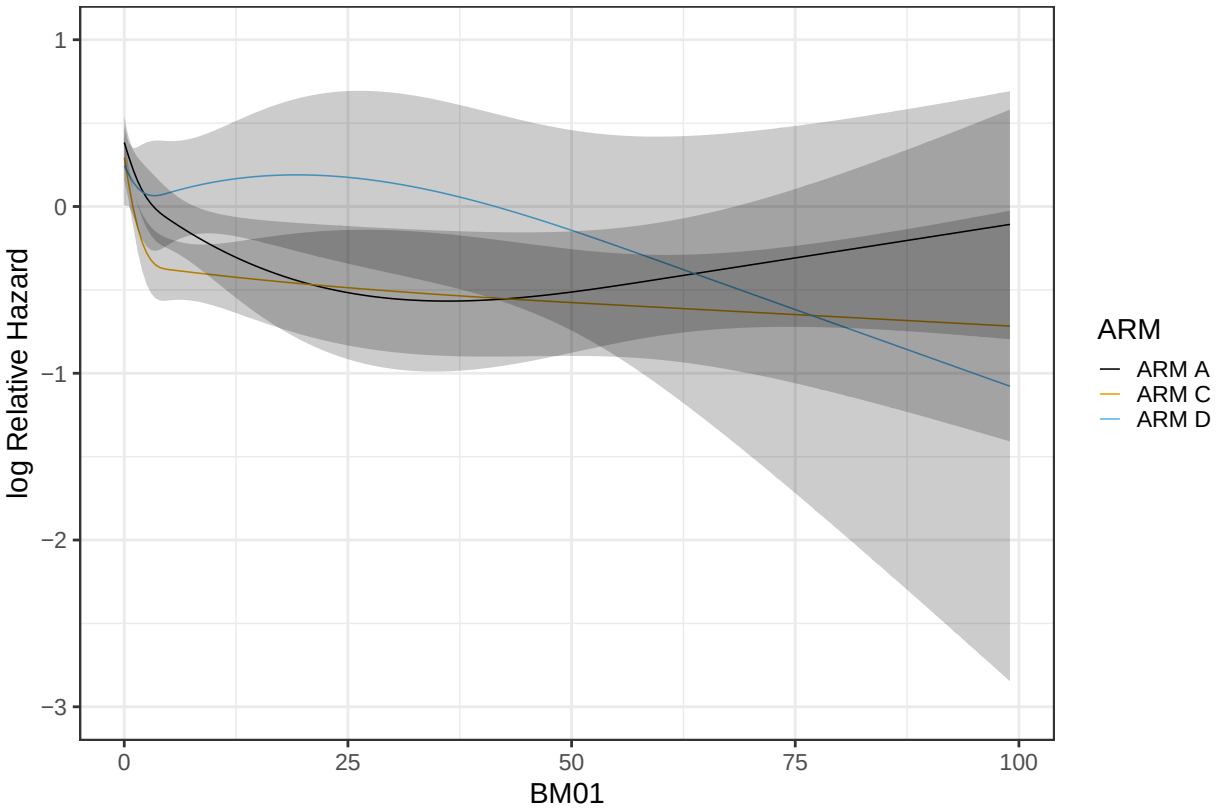
Here we give a brief example of running a cox model on a continuous biomarker interacting with treatment, with partial effects plot via the `rms` package. We illustrate using the `surv_dat` dataset.

```
library(rms)

# Here we inform the rms package of the distribution of our dataset to aid in visualizations
ddist<- datadist(surv_dat, adjto.cat = "first")
options(datadist='ddist')

# We provide a description of the data
describe(surv_dat)
```

```
## surv_dat
##
##  4  Variables      2310  Observations
## ---------------------------------------------------------------------------
## BM01
##        n  missing distinct     Info     Mean      Gmd      .05      .10
##     2086      224       36    0.942    10.18    16.26        0        0
##      .25      .50      .75      .90      .95
##        0        1        5       40       60
```

```
##
## lowest :   0   1   2   3   4, highest:  90  95  98  99 100
## ---------------------------------------------------------------------------
## ARM
##        n  missing distinct
##     2310        0        3
##
## Value      ARM A ARM C ARM D
## Frequency    834  1101   375
## Proportion 0.361 0.477 0.162
## ---------------------------------------------------------------------------
## OS_CONF
##        n  missing distinct     Info     Mean      Gmd      .05      .10
##     2186      124     1066        1     33.1    21.92    3.000    5.651
##      .25      .50      .75      .90      .95
##   15.939   38.470   42.310   62.800   64.200
##
## lowest : 0.1      0.262834 0.3      0.4      0.5
## highest: 67.6     67.7     67.8     67.9     68.7
## ---------------------------------------------------------------------------
## OSCENSOR
##        n  missing distinct     Info      Sum     Mean      Gmd
##     2186      124        2    0.725     1293   0.5915   0.4835
##
## ---------------------------------------------------------------------------
```

```r
# We run a cox model using restricted cubic spline for BM16 with 5 knots and
# interaction with Treatment (ARM)

mod <- cph(Surv(OS_CONF, 1 - OSCENSOR) ~ rcs(BM01, 5) * ARM, data = surv_dat)

# We make Prediction matrix and plot partial effects plot
surv_preds <- Predict(mod, BM01, ARM)
surv_plot <- ggplot(surv_preds)
print(surv_plot)
```



12 References

Harrell Jr, F.E., 2015. Regression modeling strategies: With applications to linear models, logistic and ordinal regression, and survival analysis. Springer.

Moons, K.G., Donders, R.A., Stijnen, T., Harrell Jr, F.E., 2006. Using the outcome for imputation of missing predictor values was preferred. Journal of clinical epidemiology 59, 1092–1101.

Steyerberg, E.W., others, 2019. Clinical prediction models. Springer.

Van Houwelingen, J., Le Cessie, S., 1990. Predictive value of statistical models. Statistics in medicine 9, 1303–1325.

13 System Information

Time required to process this report: *28.06407 secs*

R session information:

```
## R version 4.3.1 (2023-06-16)
## Platform: x86_64-pc-linux-gnu (64-bit)
## Running under: Ubuntu 18.04.6 LTS
##
## Matrix products: default
## BLAS:   /opt/tbio/domino_202310/binaries/R-4.3.1/lib/R/lib/libRblas.so
## LAPACK: /opt/tbio/domino_202310/binaries/R-4.3.1/lib/R/lib/libRlapack.so;  LAPACK version 3.11.0
##
## locale:
##  [1] LC_CTYPE=en_US.UTF-8       LC_NUMERIC=C
##  [3] LC_TIME=en_US.UTF-8        LC_COLLATE=en_US.UTF-8
##  [5] LC_MONETARY=en_US.UTF-8    LC_MESSAGES=en_US.UTF-8
##  [7] LC_PAPER=en_US.UTF-8       LC_NAME=C
##  [9] LC_ADDRESS=C               LC_TELEPHONE=C
## [11] LC_MEASUREMENT=en_US.UTF-8 LC_IDENTIFICATION=C
##
## time zone: America/New_York
## tzcode source: system (glibc)
##
## attached base packages:
## [1] stats     graphics  grDevices utils     datasets  methods   base
##
## other attached packages:
##  [1] bmsPURR_0.1.34   pcaPP_2.0-3      rms_6.7-1        Hmisc_5.1-1
##  [5] corrplot_0.92    lattice_0.21-9   tidyr_1.3.0      tibble_3.2.1
##  [9] stringr_1.5.0    ggplot2_3.4.3    dplyr_1.1.3      conflicted_1.2.0
##
## loaded via a namespace (and not attached):
##   [1] gridExtra_2.3       remotes_2.4.2.1     sandwich_3.0-2
##   [4] rlang_1.1.1         magrittr_2.0.3      multcomp_1.4-25
##   [7] polspline_1.1.23    compiler_4.3.1      getPass_0.2-2
##  [10] callr_3.7.3         vctrs_0.6.3         CatMisc_1.1.5
##  [13] quantreg_5.97       profvis_0.3.8       pkgconfig_2.0.3
##  [16] crayon_1.5.2        fastmap_1.1.1       backports_1.4.1
##  [19] ellipsis_0.3.2      labeling_0.4.3      utf8_1.2.3
##  [22] promises_1.2.1      rmarkdown_2.25      sessioninfo_1.2.2
##  [25] ps_1.7.5            MatrixModels_0.5-2  purrr_1.0.2
##  [28] xfun_0.40           cachem_1.0.8        jsonlite_1.8.7
##  [31] later_1.3.1         prettyunits_1.2.0   cluster_2.1.4
##  [34] R6_2.5.1            stringi_1.7.12      pkgload_1.3.3
##  [37] rpart_4.1.19        Rcpp_1.0.11         knitr_1.44
##  [40] zoo_1.8-12          usethis_2.2.2       base64enc_0.1-3
##  [43] httpuv_1.6.11       Matrix_1.6-1.1      splines_4.3.1
##  [46] nnet_7.3-19         tidyselect_1.2.0    rstudioapi_0.15.0
##  [49] yaml_2.3.7          codetools_0.2-19    miniUI_0.1.1.1
##  [52] processx_3.8.2      pkgbuild_1.4.2      shiny_1.7.5
##  [55] withr_2.5.1         askpass_1.2.0       evaluate_0.22
##  [58] EventLogger_1.15.3  foreign_0.8-85      desc_1.4.2
##  [61] survival_3.5-7      urlchecker_1.0.1    pillar_1.9.0
```

```
##  [64] ParamSetI_1.17.0    checkmate_2.2.0     pingr_2.0.2
##  [67] generics_0.1.3      rprojroot_2.0.3     myRepository_1.34.0
##  [70] munsell_0.5.0       scales_1.2.1        xtable_1.8-4
##  [73] glue_1.6.2          tools_4.3.1         data.table_1.14.8
##  [76] SparseM_1.81        fs_1.6.3            mvtnorm_1.2-3
##  [79] grid_4.3.1          devtools_2.4.5      colorspace_2.1-0
##  [82] nlme_3.1-163        htmlTable_2.4.1     Formula_1.2-5
##  [85] cli_3.6.1           fansi_1.0.4         gtable_0.3.4
##  [88] digest_0.6.33       dynamictable_1.5    TH.data_1.1-2
##  [91] farver_2.1.1        htmlwidgets_1.6.2   memoise_2.0.1
##  [94] htmltools_0.5.6     lifecycle_1.0.3     httr_1.4.7
##  [97] DominoR_0.0.2       mime_0.12           openssl_2.1.1
## [100] MASS_7.3-60
```

13.1 Domino environment details

Domino User: apfela

Domino Project:

Domino Run ID:

Domino Run #: