

Case Study: Prognostic Nomograms from Parametric Survival Modeling

Final Project, DA 5030 Data Mining and Machine Learning

Peter Jemley

4/3/2020

1 Introduction

For the final project I created a nomogram to graphically represent predicted mean and median survival time-windows modeled on biomedical and demographic data taken from the acutely-ill patients in the SUPPORT Study dataset archived in Vanderbilt University's Department of Biostatistics.

A nomogram is a graphical explanation of a prediction process. Nomograms are an easy-to-understand way to present and understand a prognosis. For medical providers making prognoses, and for patients receiving them, nomographic predictions make minimal demands in terms of pre-requisite statistical knowledge.

With such a diagram you can instantly see the relative effect of the different predictor variables (in the form of patients' biomedical and relevant demographic data) on the potential outcome. Even when the diagram isn't needed for any of the simple arithmetic to tally a predictor score it is able to directly display the relative effects of patients' conditions on their prognoses.

The SUPPORT Study (Study to Understand Prognoses, Preferences, Outcomes, and Risks of Treatments) spanned five hospitals and includes data on the critical conditions of 10,000 adult patients, all of whom were studied both for their in-patient outcomes as well as their rates of surviving their critical conditions once discharged.

2 The CRISP-DM Model

The CRISP-DM Model (Cross-Industry Standard Process for Data Mining) was the primary framework for this project. The process of producing a nomogram follows that model quite closely. Each of CRISP-DM's steps, as well as each of the data analysis and machine learning procedures which form this case study, are conducted sequentially here. CRISP-DM is a non-proprietary, documented, and freely available data mining model. Developed by industry leaders with input from more than 200 data mining users and data mining tool and service providers, CRISP-DM is an industry-, tool-, and application-neutral model. This model encourages best practices and offers organizations the structure needed to realize better, faster, reproducible results from data mining.

CRISP-DM organizes the data mining process into six phases: business understanding, data understanding, data preparation, modeling, evaluation, and deployment. These phases help organizations understand the data mining process and provide a road map to follow while planning and carrying out a data mining project.

3 Nomograms and the CRISP-DM Model

It's interesting to note a unique aspect of nomograms in terms of how they fit into the CRISP-DM Model. Predictive data analytics do not usually solve business problems on their own. They generally provide insights which help organizations make apposite decisions and apply them to those problems. The predictions made from nomograms do in fact solve some business problems in health care, namely effectively communicating complex biomedical contexts about patients with them and their families in an effective and low-cost manner. And in this project the business phase of the CRISP-DM Model — predicting survival times for acutely-ill patients modeled on the SUPPORT study — is also the deployment phase, the nomogram itself.

For this case study I took a random sample of 1000 patients from the SUPPORT Study with which to conduct the parametric survival modeling described in detail below. I chose parametric survival modeling's methods primarily because D.R. Cox, the winner of the equivalent of the Nobel Prize for statistics in 2017, and the inventor of the widely used Cox Proportional Hazards Model, was asked after his award if he would do anything differently if he could start his career again. He responded that he would have used parametric survival models more frequently.

His reasoning consisted of parametric survival modeling's abilities to reliably derive predicted values from patient data according to modeling assumptions which are more transparent than other approaches. And as is the case here, parametric modeling's approaches to patients' time-dependent covariates can analyze and interpret them with greater accuracy and reliability than with a semi-parametric Cox model.

4 Accelerated Failure Time Models

The model I'm using is in the class of models referred to as accelerated failure time models because the prognostic modeling for acute patients doesn't fit the patterns needed for a Cox Proportional Hazard Model. A Cox model tends to be more useful for modeling chronic diseases in which risk factors have more constant effects over time.

In the study patients had to survive until Day 3 to qualify for inclusion, and the baselines are the Day 3 measurements. You'll see descriptive statistics for these measurements at the beginning of the analysis. It's important to note that the age distribution goes all the way from 18 years to 95 years, and is a very nice distribution for this case study. The vast majority of subjects were in the intensive care unit (ICU).

5 Libraries and the Computing Environment

Here you scrub the environment, load libraries for visualizations, attach required special transformation and analytic functions from the apposite packages (rms and Hmisc), then scrub the console.

rms, for example, contains a series of special transformation functions, fitting functions, and generic analysis functions which help automate many analysis steps, e.g. fitting restricted interactions and multiple stratification variables, analysis of variance (with tests of linearity of each factor and pooled tests), plotting effects of variables in the model, estimating and graphing effects of variables that appear non-linearly in the model using e.g. inter-quartile-range hazard ratios, bootstrapping model fits, and constructing nomograms for obtaining predictions manually.

```
library(ggplot2)
library(knitr)

suppressMessages({
  require(Hmisc)
  require(rms)
})

rm(list = ls())
cat("\014")
```

6 The SUPPORT Study

There's a lot of information in this dataset and the patients are well-characterized by these data.

Extra precautionary measures were taken by those conducting the study. The data collected were double-checked for accuracy by a second medical abstractor: "Patient identification and data collection procedures included ongoing reliability testing. General physiologic measures were collected by a second nurse abstractor from a random 10 percent sample of patients within 3 to 10 days of the initial data collection." (The SUPPORT Prognostic Model, *Annals of Internal Medicine*, 1995; 122 : 191-203).

The `getHdata()` function from the `Hmisc` package downloads and prepares datasets stored on Vanderbilt's Biostatistics website for use in statistical modeling environments. When using `getHdata()` in R, these datasets are de-compressed with R's `load()` function base, having been compressed for storage with R's `save()` function. After loading, `getHdata()` runs the `cleanup.import()` function, which corrects errors and shrinks sizes of data frames, for example by changing double-precision numeric variables to integers when no fractional components are present.

Acute illnesses are the flagged categories of interest, consisting of — Acute Respiratory Failures ("ARF"), Multiple Organ Systems Failures ("MOSF"), and Coma ("Coma").

Note: to save space in this document I have not implemented the `Hmisc` function `describe()`. It produces tables for each variable which contain the number of occurrences of the variable in the dataset, whether any values are missing, the number of distinct observations, and summary statistics. `describe()` includes Gini's mean difference (denoted as "Gmd"), which quantifies the dispersion among the variables' values in terms of the mean absolute difference between all pairs of observations.

1. Retrieve the study's data.
2. Flagging categories of interest.
3. Display descriptive statistics (code is commented out to save printing space).

```
# 1
getHdata(support)

# 2
acute <- support$dzclass %in% c('ARF/MOSF', 'Coma')

# 3
# describe(support[acute,])
```

7 Hierarchical Cluster Analysis: Showing Patterns in any Missing Data

The `varclus()` function from `Hmisc` performs a hierarchical cluster analysis.

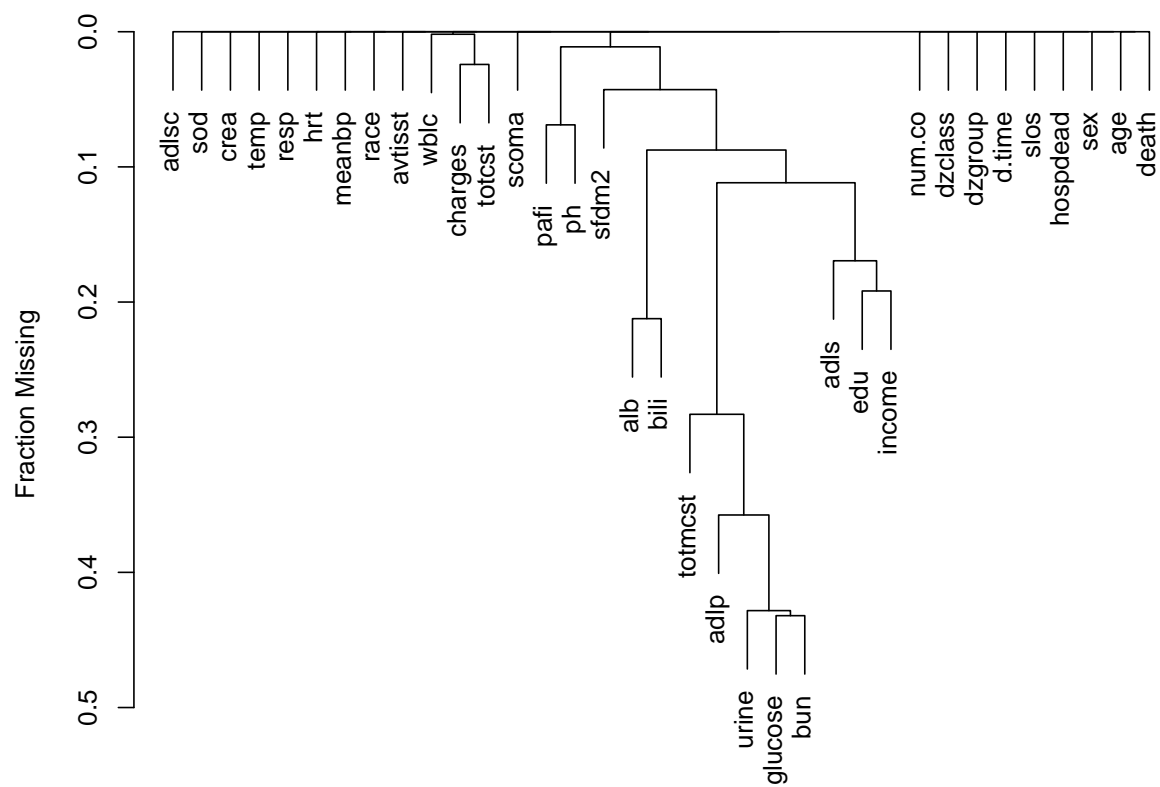
Here you see variable clustering, specifically clustering on whether or not variables were missing. You want to understand any patterns of holes in the data, as well as how many absences occur simultaneously for more than one variable.

We can see that for over 40 percent of the patients, the same patients have missing urine output, glucose, and bun data. Those variables were not collected in the first half of the study.

There is very little missing data on most of the variables, but we can also see that education and income tend to be missing on the same patients about 1/5 of the time. If we were going to use income for imputing education we'd be somewhat out of luck.

1. `naclus()` from `Hmisc` detects similarities among observations with missing variables in a data frame. The similarity measure is the fraction of NAs in common between any two variables.

```
# 1
plot(naclus(support[acute, ]))
```



8 Detecting statistical associations between variables

Here you conduct a variable clustering, in addition to the one above, to see which variables' values may be similar to one another.

This clustering uses Hoeffding's D, a general measure of independence. It's ideal for continuous variables but works somewhat for discrete ones when they are ordered.

`varclus()` implements a hierarchical cluster analysis on variables, using the Hoeffding D statistic, squared Pearson or Spearman correlations, or proportion of observations for which two variables are both positive as similarity measures. Variable clustering is used for assessing collinearity, redundancy, and for separating variables into clusters that can be scored as a single variable, thus resulting in data reduction.

The measure's range is a little hard to understand — but it's so general it can detect relationships between variables having high dependencies.

So we can see that the two variables whose values are the most similar to each other are serum creatinine and serum blood urea nitrogen — both measures of renal function.

This is the closest case in this dataset in which two variables could be considered redundant.

1. You create a new data subset for the modeling.
2. You then drop the unused levels from support's designated acute illness groups.
3. Applying the label Disease Group to the subset.
4. Then you attach the new subset to R's search path before implementing the `varclus()` function.
5. Then you plot the similarity matrix created by `varclus()` as a hierarchical cluster.

```
# 1
ac <- support[acute, ]

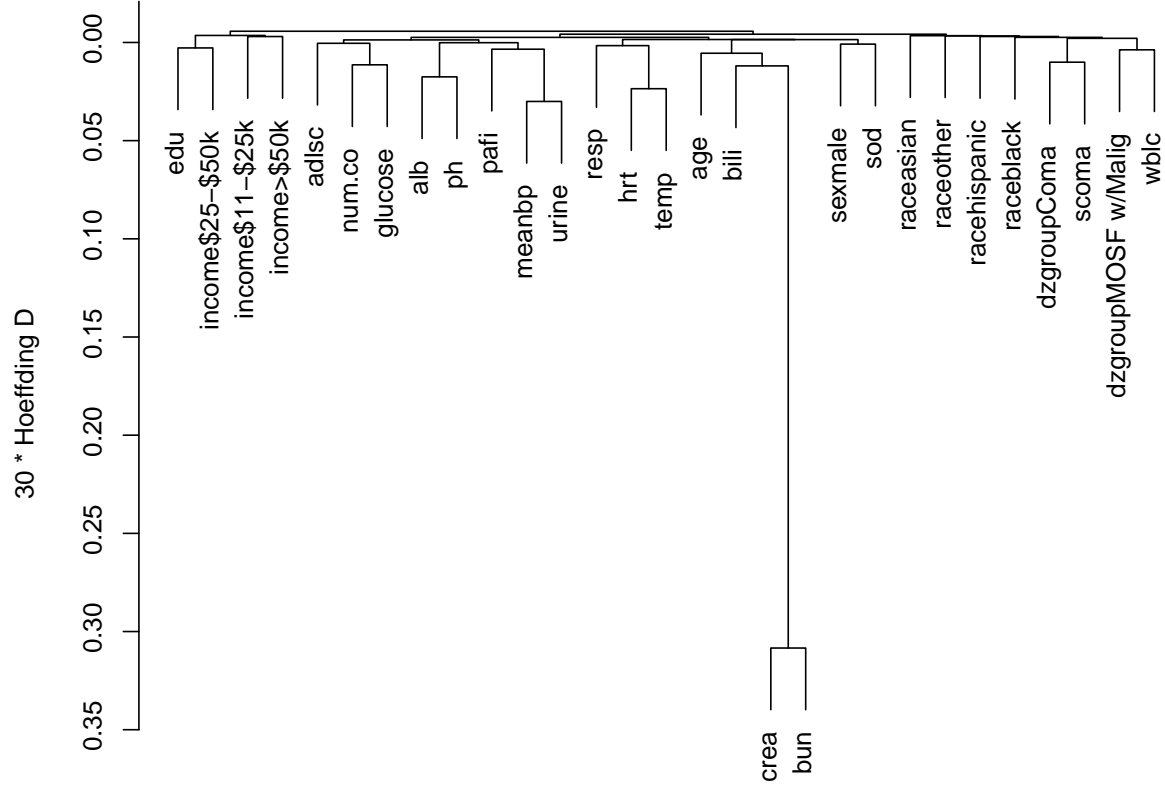
# 2
ac$dzgroup <- ac$dzgroup[drop = TRUE]

# 3
label(ac$dzgroup) <- 'Disease Group'

# 4
attach(ac)
vc <- varclus(
  ~ age + sex + dzgroup + num.co + edu + income +
    scoma + race + meanbp + wblc + hrt + resp +
    temp + pafi + alb + bili + crea + sod + ph +
    glucose + bun + urine + adlsc,
  sim = 'hoeffding'
)

# 5
```

```
plot(vc)
```



9 Verifying variables' reference ranges

After creating a new data subset for modeling you can create boxplots for each variable when apposite, with explanations for what clearly appear to be outliers in the data.

Code to print boxplots is commented out below to save on printing space but they were examined before proceeding.

All conclusions about these outliers accompany the code for each boxplot, as well as explanations for those variables unsuitable to plotting. Running the `describe()` function on the data above, the SUPPORT Study's own abstractors' efforts, along with verifying the reference ranges shown in the boxplots for the numeric variables below, demonstrate that no outliers are present in this subset of the SUPPORT Study's data.

Each of the following are unsuitable to `boxplot()`:

death, sex, death in hospital (`hospdead`), days study entry to discharge (`slos`), days of follow-up (`d.time`), disease group (`dzgroup`), disease class (`dzclass`), number of co-morbidities (`num.co`), income, SUPPORT coma score (`scoma`), hospital charges (`charges`), total micro-cost (`totmcst`), race, white blood cell count (`wblc`), all of the activities of daily living (`adlp`, `adls`, `adlsc`), and `sfdm` (severe functional disability). Their values and ranges were verified using the `describe()` function.

1. For age, a numeric variable.
2. For average TISS (avtisst, the Therapeutic Intervention Scoring System, which is a method for calculating costs in the intensive care unit (ICU) and intermediate care unit (IMCU) of a hospital), a numeric variable.
3. For mean arterial blood pressure, a numeric variable.
4. For heart rate, a numeric variable. As plotted, heart rate appears to have outliers. These rates are not unusual for patients with supraventricular tachycardia, however.
5. For respiration rate, day 3, a numeric variable. Tachypnea, rapid breathing, is also not unusual in acutely-ill patients. The patient with the rate which is the outlier died.
6. For temperature (celcius), day 3, a numeric variable. The one outlier was also associated with patient who died.
7. For pafi, a measure of oxygenation of tissues, a numeric variable. The greatest outlier is over the reference range (5 - 800), but this value should remain as it may in fact be an accurate measurement.
8. For serum albumin (alb), a numeric variable. All of these values are within the reference range for acutely-ill patients.
9. For bilirubin day 3 (bili), a numeric variable. Although these outliers are extreme, they are noted in the medical literature for acutely-ill patients.
10. For serum creatinine day 3 (crea), a numeric variable. These outliers indicate renal failure, and would be part of the Multiple Organ System Failure disease group.
11. For serum sodium day 3 (sod), a numeric variable. These outliers are also within the reference range for acutely-ill patients.
12. For serum pH arterial day 3 (ph), a numeric variable. These outliers are also within the reference range for acutely-ill patients.
13. For glucose day 3 (glucose), a numeric variable. These outliers are within the reference range for diabetic patients.
14. For blood urea nitrogen day 3 (bun), a numeric variable. These outliers are also within the reference range for acutely-ill patients.
15. For urine output day 3 (urine), a numeric variable. The outliers are excessive, but appear in the medical literature.

```
# # 1
# boxplot(ac$age, xlab = "Age")
#
# # 2
# boxplot(ac$avtisst, xlab = "TISS")
#
# # 3
# boxplot(ac$meanbp, xlab = "Mean Blood Pressue")
#
```

```

# #4
# boxplot(ac$hrt, xlab = "Heart Rate")
#
# # 5
# boxplot(ac$resp, xlab = "Respiration")
#
# # 6
# boxplot(ac$temp, xlab = "Temperature")
#
# # 7
# boxplot(ac$pafi, xlab = "PAFI")
#
# # 8
# boxplot(ac$alb, xlab = "Albumin")
#
# # 9
# boxplot(ac$bili, xlab = "Bilirubin")
#
# # 10
# boxplot(ac$crea, xlab = "Serum Creatinine")
#
# # 11
# boxplot(ac$sod, xlab = "Serum Sodium")
#
# # 12
# boxplot(ac$ph, xlab = "pH")
#
# # 13
# boxplot(ac$glucose, xlab = "Glucose")
#
# # 14
# boxplot(ac$bun, xlab = "Blood Urea Nitrogen")
#
# # 15
# boxplot(ac$urine, xlab = "Urine Output")

```

10 Choosing a Tentative Model

Here you can make a tentative model choice of a log normal accelerated failure time model. This model choice would be ordinary linear regression if the study did not involve any censoring. There is censoring here because patients who were not followed up on until their deaths were censored at their last contact date.

When the response variable is the time until an event, subjects not followed long enough for the event to have occurred have their event times censored at the time of last follow-up.

To handle the censoring, one assumes along with the assumptions of a parametric model that the log of the time until death is conditional on the covariates, i.e. we have a normal distribution with a constant variance of sigma squared.

Using the rms package you create the survival time object S which is a 2-column variable. It has the years of follow up or years until death, and the second column is binary indicator of what type of follow up that was, i.e. follow up while still alive, or at death.

Next you make stratified Kaplan-Meier Estimates, transforming the y-axis using the inverse normal transformation, or Z transformation. If you only had disease group (dzgroup) in the model with these 3 categories, which you don't here but we're oversimplifying, then the log normal model would fit if these lines were all straight and parallel.

They all look pretty straight but they don't look perfectly parallel, indicating that the sigma in the model (the residual standard deviation on the log scale) may need to vary and we don't really have a systematic way to approach that. The plots' linearities and approximate parallelisms, however, demonstrate a reasonable fit for the log-normal accelerated failure time model using one predictor.

You can hope that when you adjust for more variables you adjust for more heterogeneity among the patients — and that variables will be fitting the log normal model better. So, then you fit a parametric survival model of type log normal accelerated failure time model, using disease group as a predictor, a restricted cubic spline for age with 5 knots, and the same for blood pressure with 5 knots.

In mathematics, a spline generally refers to a continuous curve constructed so as to pass through a given set of points and have a certain number of continuous derivatives.

There are 3 very powerful variables in this tentative model, and by the time you have 3 powerful variables the residuals should behave similarly to the way they will in a final model.

1. Creating distribution summaries for predictor variables using the `datadist()` function.
2. Describing the distributions of variables to rms.
3. Converting the days of follow up (`d.time`) variable to years.
4. Creating the right-censored survival time object/variable using the `Surv()` function from the survival package.
5. Displaying inverse Kaplan-Meier estimates stratified by disease group (`dzgroup`) using the `survplot()` function from rms. Show normal inverse Kaplan-Meier estimates stratified by `dzgroup`. `npsurv()` from rms computes an estimate of a survival curve for censored data using either the Kaplan-Meier or the Fleming-Harrington method or computes the predicted survivor function.

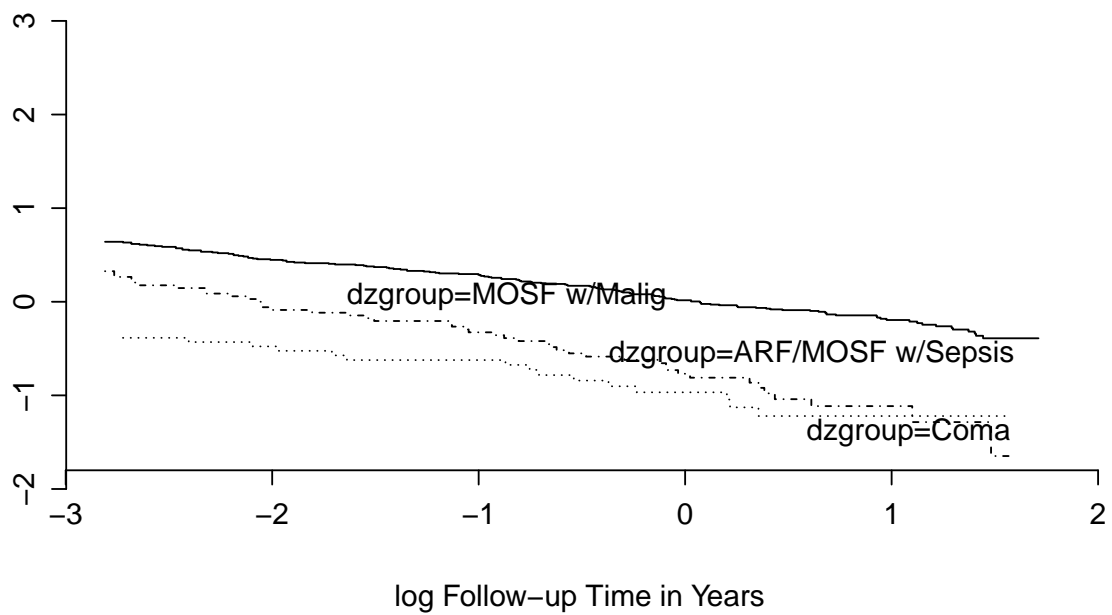
```
# 1
dd <- datadist(ac)

# 2
options(datadist = 'dd')

# 3
years <- d.time / 365.25
units(years) <- 'Year'

# 4
S <- Surv(years, death)

# 5
survplot(npsurv(S ~ dzgroup),
         conf = 'none',
         fun = qnorm,
         logt = TRUE)
```



11 Calculating the tentative model's residuals

Moving on to calculating the tentative model's residuals, when there is censored data in a log normal survival model you can consider that if someone was followed until death the residuals would have the usual meaning: the log survival time minus the predicted log survival time. But if someone has lost a follow up before some event like death the survival time is right censored and the residual is right censored as well.

To look at the distribution of the residuals we can take censoring into account using the Kaplan-Meier estimate just as we would with raw data.

To help with this you generate a random number which can serve as a touchstone for the tentative model. When stratifying by a random number the sampling variation will still be there but it will be pure sampling variation because a random number is unrelated to anything in the dataset — stratifying by a random number should generate identical distributions as those in the data.

So, a random number has to fit the data and the residuals have to have the same distribution. That random number is stratifying the Kaplan-Meier estimates by the random number into quartiles, so for continuous variables this surplus function on the residuals breaks it into quartiles by default. And we see some variation with the random data that's really not different from the variation you get on the real stratifications.

When you stratify by disease group there's a little bit of a shoulder there early on which is slightly problematic. Stratifying by age is a very good fit: all of those 4 curves by quartiles of age are hovering around the thick black curve which is the theoretical 1 minus normal cumulative distribution function that should hold if your model's assumptions are sound

When stratifying by mean arterial blood pressure the curve is off a little bit from the theoretical curve but about the same as when stratifying by the random number.

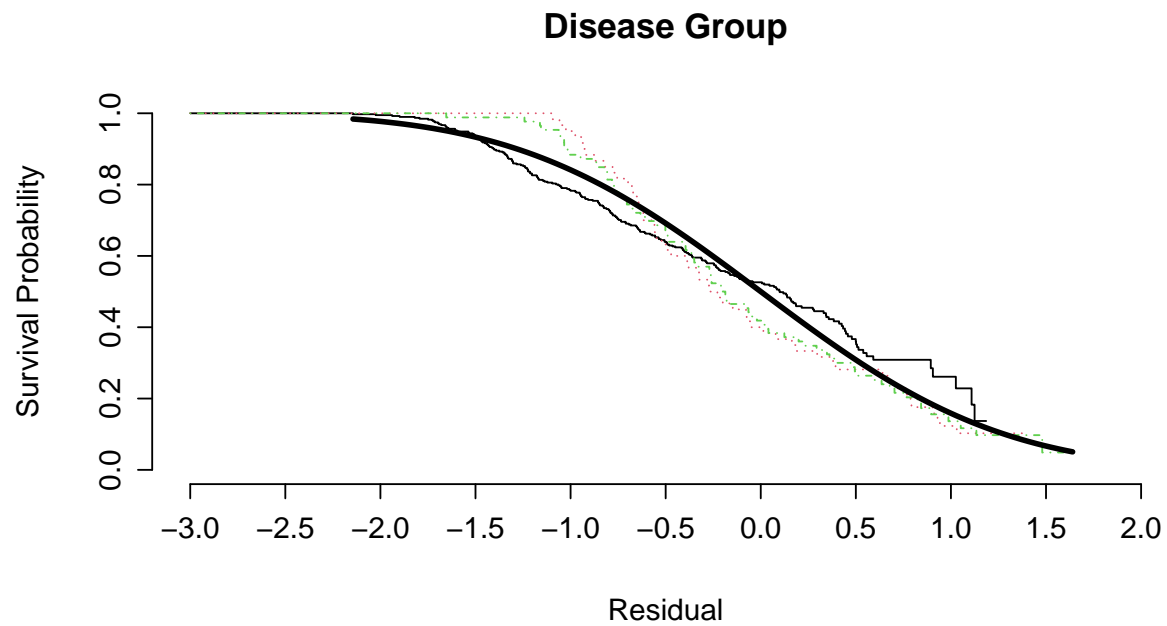
Like all graphical assessments of model fit this is a subjective process but this model fit is as good as you usually see.

1. Creating a parametric survival model object using the `psm()` function from `rms`. `psm()` fits the accelerated failure time family of parametric survival models, and uses the `rms` class for automatic anova, `fastbw`, `calibrate`, `validate`, and other functions. The object `S` is created as a function of the disease group, a restricted cubic spline for age with 5 knots, and the same for mean arterial blood pressure with 5 knots.
2. Extracting the model's residuals using the `residuals()` function from the base stats package.
3. Plotting the curves of the residuals.
4. Creating the random variables and plotting them.

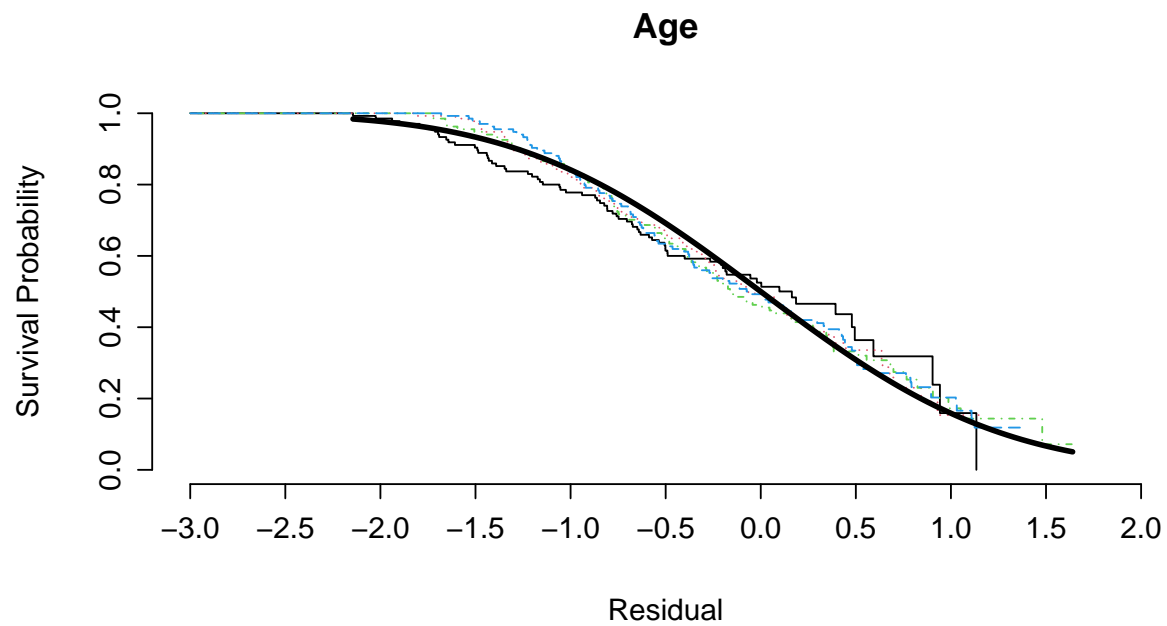
```
# 1
f <- psm(S ~ dzgroup + rcs(age,5) + rcs(meanbp,5),
        dist='lognormal', y=TRUE)

# 2
r <- resid(f)
```

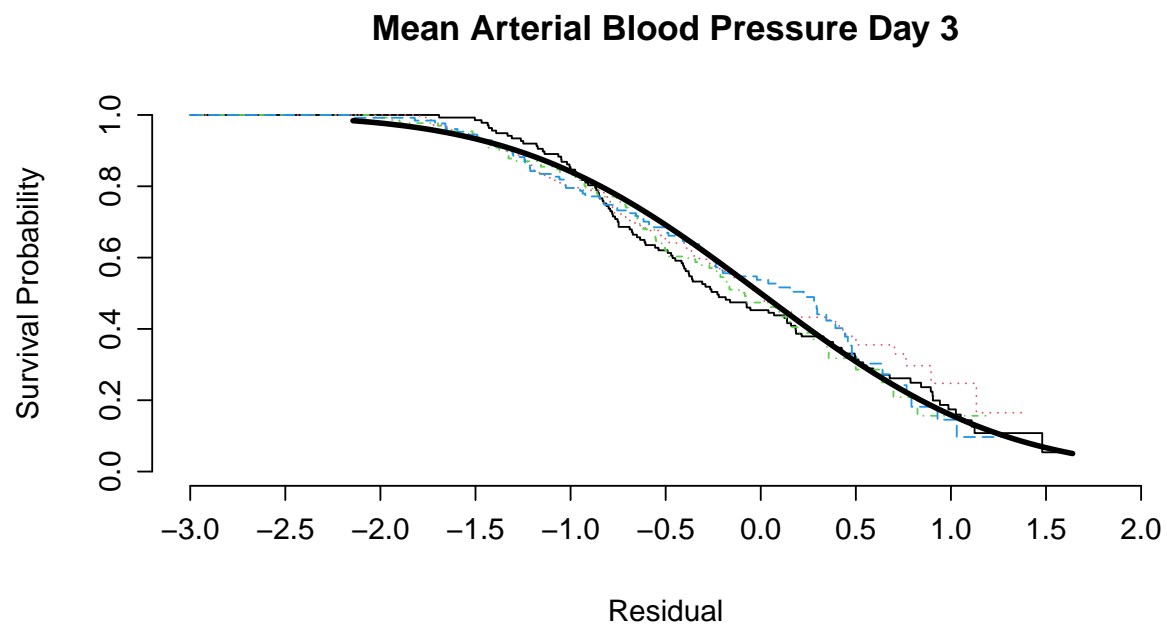
```
# 3  
survplot(r, dzgroup, label.curve = FALSE)
```



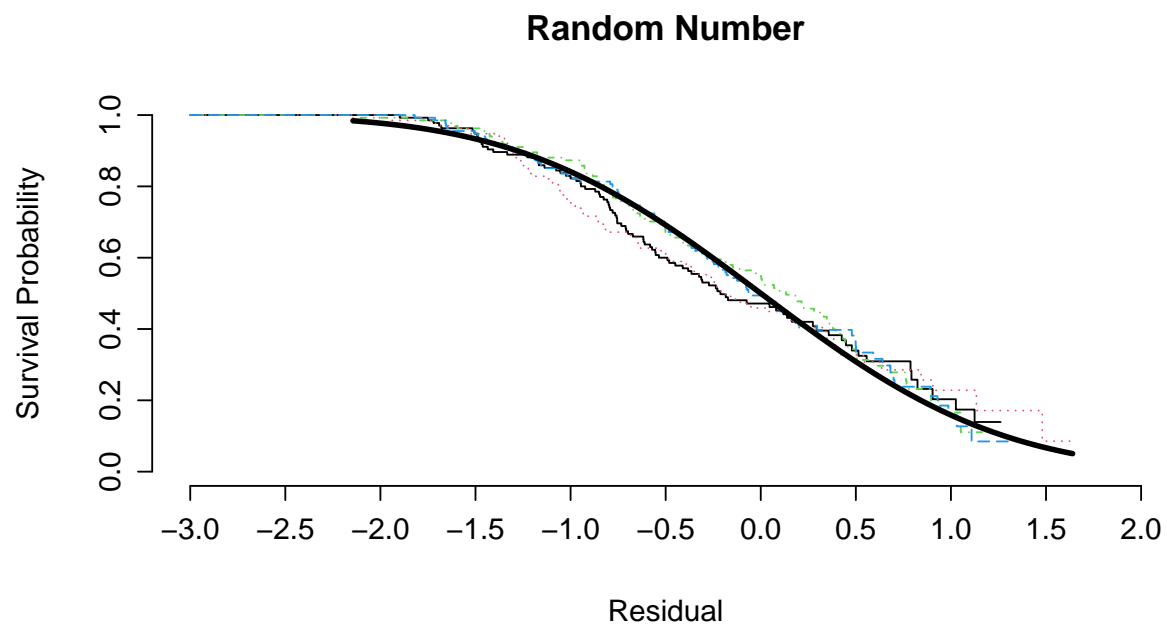
```
survplot(r, age, label.curve = FALSE)
```




```
survplot(r, meanbp, label.curve = FALSE)
```



```
# 4  
random <- runif(length(age))  
label(random) <- 'Random Number'  
survplot(r, random, label.curve = FALSE)
```



12 Building the accelerated failure time parametric survival time model

Next you begin by building the accelerated failure time parametric survival model. A big part of building this model is getting the subject matter input about which of the variables should be used as predictors and another big part is how to assign degrees of freedom to those variables.

In general, degrees of freedom refers to the number of “free floating” parameters, or the number of opportunities a statistical estimator or method was given.

You examine which variables had enough predictive potential to justify giving them many degrees of freedom, i.e. variables which were very steep in their relationship to survival time. You have to try to avoid under-modeling such variables by assuming they’re linear when they may not be.

You can begin by using the pairwise Spearman rho squared method: the generalized one that uses the rank of the predictor and the square of the rank of the predictor against the rank of survival time.

Spearman’s rho squared doesn’t know anything about censoring so you can’t give it the time until event variable because its censored values would be considered the same as the uncensored values. That’s an obvious error.

From the data you find that the shortest follow up time for survivors is a year, so you can truncate the data at one year and just look at the actions between zero and 365 days. That is the period for which there is complete data, and for any patient followed for 365 days we know that they were either alive at day 365 or we know the date of their death within that interval.

This provides complete data, but data with a substantially lower information content because you’re not distinguishing between a death in the truncated data and any death between 366 days and 1000 days, for example.

When the follow up times are truncated at the shortest follow up, one year, the Spearman’s rho squared is calculated — i.e. the 2 degrees of freedom generalization of it.

The variable that has the highest potential is the coma score (scoma), which is a simple translation of the Glasgow Coma Score, and the next highest potential variable is mean arterial blood pressure, and the third is disease group.

The coma score is a bit different because it was derived from an overall analysis of the SUPPORT patients, and although derived from a Cox model it was linear. So you’ll have to end up giving one degree of freedom to that even though that knowledge was gained from an analysis other than this one.

The mean blood pressure is continuous so you want to give it the max number of knots, and for disease group we want to leave in its three categories and not collapse any of those. For the variables with lower potential prediction value they will have only one degree of freedom, forcing them to be linear.

This type of analysis, as well as the next one you’re going to do, consists of looking at predictors one variable at a time, not each adjusted for the other, making this particular part of this case study basically somewhat unreliable, but that will be improved on as we move on.

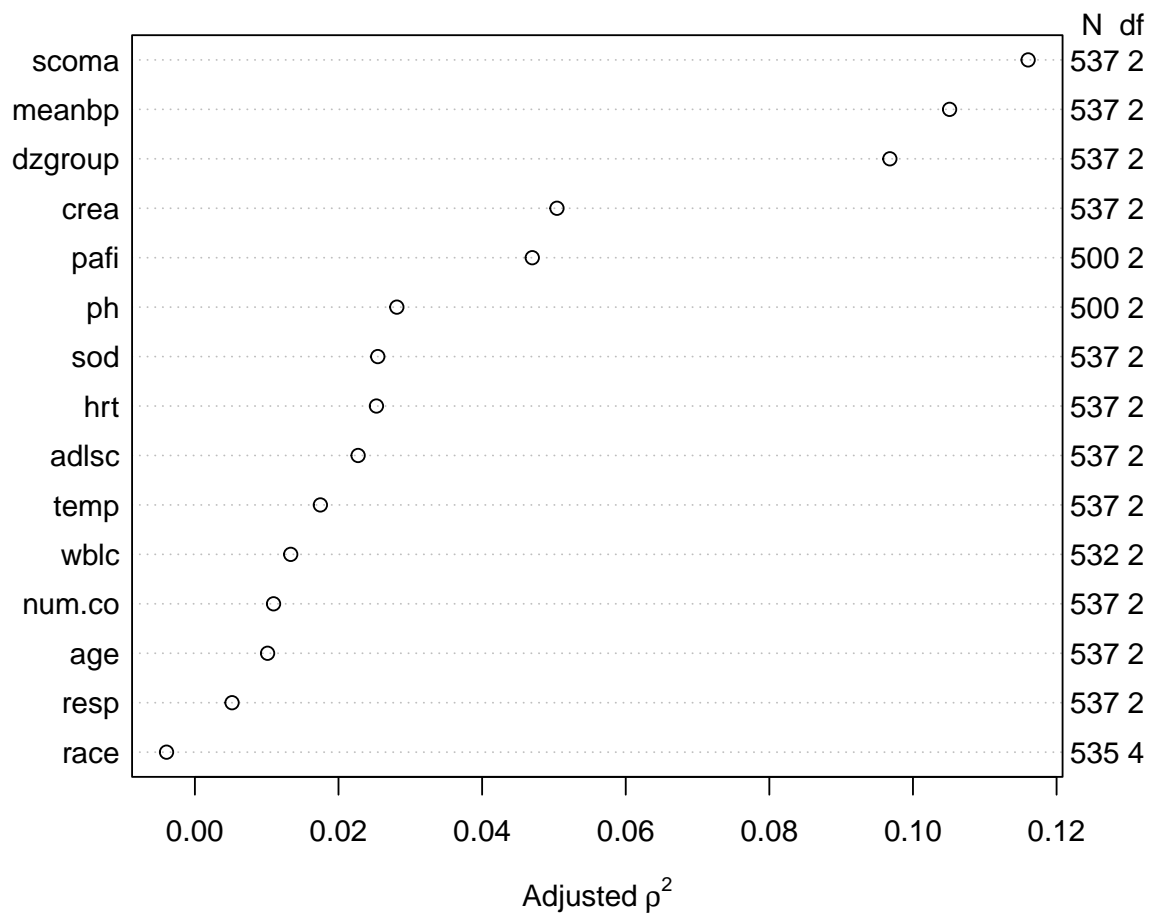
1. Calculating the shortest follow up.
2. Converting shortest follow up object to an object for the `spearman2()` function as a parallel minima object with `pmin()` from the base package.
3. Creating the bivariate Spearman's rho squared object with 2 degrees of freedom.
4. Plot it.

```
# 1
shortest.follow.up <- min(d.time[death == 0], na.rm = TRUE)

# 2
d.timet <- pmin(d.time, shortest.follow.up)

# 3
w <- spearman2(
  d.timet ~ age + num.co + scoma + meanbp +
    hrt + resp + temp + crea + sod + adlsc +
    wblc + pafi + ph + dzgroup + race,
  p = 2
)

# 4
plot(w, main = '')
```



13 A better approach to take censoring into account

You can use a different rank correlation coefficient, Somer's Dxy, which has been adapted to handle right censoring. When calculating concordant and discordant numbers that go into all possible pairs analysis with Somer's D we can take partial information into account.

For example, if someone died before someone else was censored we know that the censored person has already outlived the other person and we know how the ranking between them works in that sense. But the analysis includes two persons who were censored and we cannot compare them, so when calculating concordant pairs there will be one uncertain pair.

So, you're taking into account censoring and looking at the variables one at a time using Somer's Dxy instead of using a Spearman correlation. Although Somer's Dxy is not able to detect nonmonotonic relationships (variables which do not generally vary in the same manner or direction), when you use this approach, using the `rcorrcens()` function in the Hmisc package this makes it easy to take the censoring discussed immediately above into account.

This results in the graph in which you can see that mean blood pressure has the highest prediction potential, creatinine next, then disease group, then coma score: they're all very high.

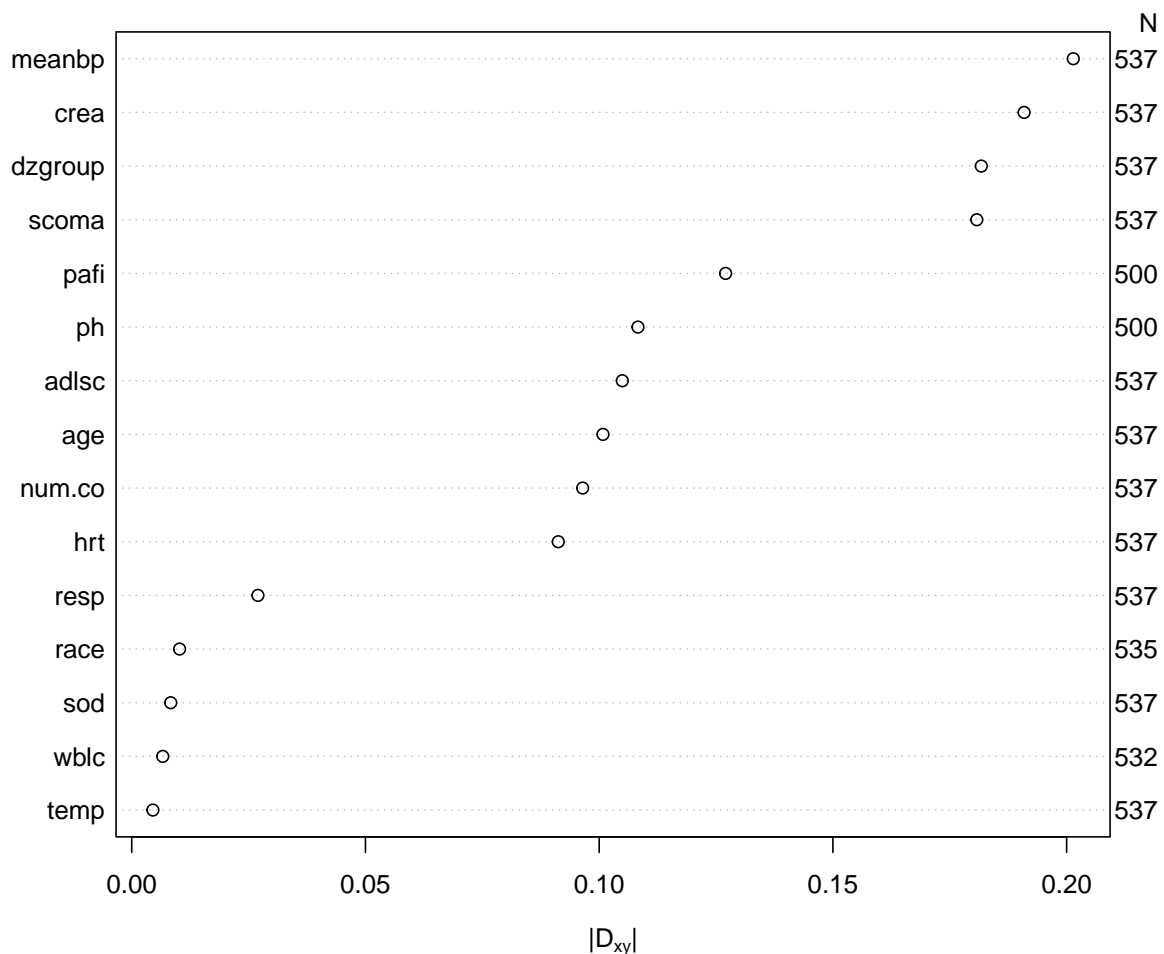
You can have more confidence in this graph than the previous one which used the oversimplified Spearman's rho squared.

1. Taking censoring into account using the `rcorrcens()` function from the Hmisc package. `rcorrcens()` computes the c index and the corresponding generalization of Somers' Dxy rank correlation for a censored response variable.

2. Plot it.

```
# 1
w <- rcorrcens(
  S ~ age + num.co + scoma + meanbp + hrt + resp +
    temp + crea + sod + adlsc + wblc + pafi + ph +
    dzgroup + race
)

# 2
plot(w, main = '')
```



14 Simple Imputation

Here is code for simple imputation based on sound statistical principles and meaningful research.

For example, if white blood count data is missing you enter 9 (which in the original scale was 9000). For this type of imputation there was a federal grant from what is now called AHRQ to study the safest ways to impute data in an ICU population like the population in the SUPPORT Study.

The research found that the physicians were very good at knowing that for certain patients ordering certain tests were unnecessary because they would consistently come back normal. The research found that instead of doing an overly complex imputation it actually worked better to put in the most normal value for these variables as constants.

1. Determining the number of missing values per variable.
2. Imputation.
3. Collapse race into race2 as race turned out not to have explanatory power for this analysis due to the patients' acute conditions — although racism is a very important factor for social determinants of health in other medical contexts.
4. `datadist()` determines summaries of variables for effect and plotting ranges, values to adjust to, and overall ranges for the functions `Predict`, `plot.Predict`, `ggplot.Predict`, `summary.rms`, `survplot`, and `nomogram.rms`.

```
# 1
sapply(llist(
  age,
  num.co,
  scoma,
  meanbp,
  hrt,
  resp,
  temp,
  crea,
  sod,
  adlsc,
  wblc,
  pafi,
  ph
), function(x)
  sum(is.na(x)))
```

```
##   age num.co  scoma meanbp   hrt   resp   temp   crea   sod  adlsc  wblc
##    0      0      0      0      0      0      0      0      0      0      5
##  pafi    ph
##   37    37
```

```
# 2
wblc.i <- impute(wblc, 9)
pafi.i <- impute(pafi, 333.3)
ph.i   <- impute(ph, 7.4)
race2  <- race

# 3
levels(race2) <- list(white = 'white', other = levels(race)[-1])
race2[is.na(race2)] <- 'white'

# 4
dd <- datadist(dd, wblc.i, pafi.i, ph.i, race2)
```

15 Redundancy Analysis

After data imputation the data is ready for a formal multivariable redundancy analysis. This takes each of the potential predictor variables and attempts to predict each from the others.

The `redun()` function from the `Hmisc` package improves on the `varclus` pairwise correlation analysis by allowing for nonmonotonic transformations while predicting each predictor from all the others.

`redun()` uses 4 knots here — knots represent the points at which the coefficients in an equation change.

When doing so you can see that we have the R squared with which each variable can be predicted from the others. So, the variable which is the most predictable from the others, going across, is disease group with a score of 0.45, and `scoma` at 0.42. Those two are somewhat predictable from each other because the coma group from disease group is going to have a very poor `scoma` score due to the coma patients inherent disease characteristics.

No variables were found to be redundant to the level of having an R squared of 0.9 for predictability.

1. `redun()` uses flexible parametric additive models (along with `areg()`'s use of regression splines, also from the `Hmisc` package) to determine how well each variable can be predicted from the remaining variables. Variables are dropped in a stepwise fashion, removing the most predictable variable at each step.

1a. This type of analysis requires missing values to be imputed such as those imputed immediately above in order to preserve the information content of the sample (i.e. to not significantly reduce the sample size).

1b. `redun()` also exceeds the abilities of the `varclus()` pairwise correlation approach, permitting nonmonotonic transformations for predicting each predictor from all other predictors.

```
# 1
# 1a
# 1b
redun(
  ~ crea + age + sex + dzgroup + num.co + scoma + adlsc +
  race2 + meanbp + hrt + resp + temp + sod + wblc.i +
  pafi.i + ph.i,
  nk = 4
)

##
## Redundancy Analysis
##
## redun(formula = ~crea + age + sex + dzgroup + num.co + scoma +
##       adlsc + race2 + meanbp + hrt + resp + temp + sod + wblc.i +
##       pafi.i + ph.i, nk = 4)
##
## n: 537   p: 16   nk: 4
##
## Number of NAs:    0
##
## Transformation of target variables forced to be linear
##
```



```
## R-squared cutoff: 0.9      Type: ordinary
##
## R^2 with which each variable can be predicted from all other variables:
##
##      crea      age      sex dzgroup  num.co   scoma   adlsc   race2  meanbp    hrt
##    0.133    0.246    0.132   0.451   0.147   0.418   0.153   0.151   0.178   0.258
##      resp     temp     sod  wblc.i  pafi.i    ph.i
##    0.131    0.197    0.135   0.093   0.143   0.171
##
## No redundant variables
```

16 Avoiding Bias: Fit the PSM and conduct an ANOVA

Bias is a type of systemic or systematic error. A satisfaction questionnaire that leads patients to never report that they are dissatisfied with their medical care is an example of bias. It typically pertains to the discrepancy between the average of many estimates over repeated sampling and the true value of a parameter. Bias of this general type is on the surface more related to frequentist statistics than to Bayesian statistics.

I spent a lot of time looking at the complexity of the model and allocating degrees of freedom, but in this data context where we have enough events to look at in a model which is saturated. It's considered sound practice to take the saturated model approach in allocating degrees of freedom.

When looking at the ANOVA in the plot you do see that the variable which had the least to say was the sex variable, and the variable which has the most to say is the disease group. Once you've adjusted for everything else the variable which is seeming to have the most extra information is disease group, which is followed by creatinine, then mean bp, then age, then lung function, then coma score. So, you can trust this analysis more than the one above because it's not just looking at marginal univariate or pairwise associations.

Adjusting or controlling for a variable refers to assessing the effect of one variable while accounting for the effect of another (confounding) variable. When an apparent association between two outcomes might be explained by some observed common factor that influences both, this common cause is known as a confounder. Adjustment for the other variable can be carried out by stratifying the analysis (especially if the variable is categorical) or when continuous by statistically estimating the relationship between the variable and the outcome and then subtracting out that effect to study which effects remain.

I proceeded to fit the model on the basis of the latter method (statistically estimating the relationship between the variable and the outcome and then subtracting out that effect), according to the most promising predictors which were allocated 5 knots and then allocated in descending order as they appear on the dot chart.

There are 3 knots for some variables which are not so strong, temp is forced to be linear because it didn't have much potential at all even when allowed to be nonlinear.

You may have thrown away the wrong terms of temperature because what little it has to say might be coming from the nonlinear terms and those are the ones we're discarding. But that's one way to remain unbiased on the average.

1. Create a constant to represent 4 knots for the `psm()` function from the `rms` package.
2. Using the `rcs()` function from the `rsm` package helps to blind researchers from becoming aware of the linear forms of continuous variables' effects, removing a source of bias.

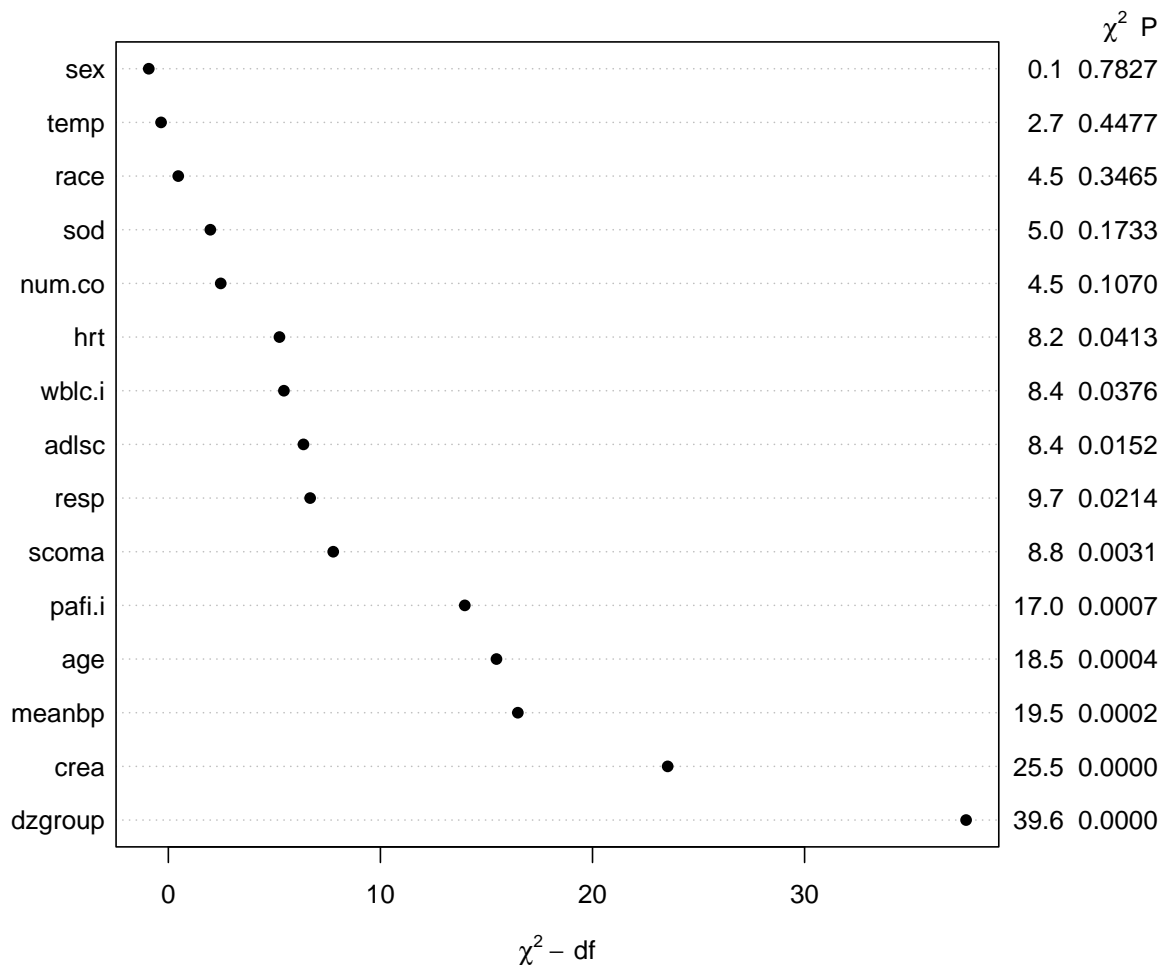
2a `rcs()` is one of a series of functions that set up special attributes (such as knots and nonlinear term indicators) that are carried through to fits (using for example `lrm`, `cph`, `ols`, `psm`). `anova.rms`, `summary.rms`, `Predict`, `survplot`, `fastbw`, `validate`, `specs`, `which.influence`, `nomogram` and `latex.rms` use these attributes to automate certain analyses (e.g., automatic tests of linearity for each predictor are done by `anova.rms`). Many of the functions are called implicitly.

3. Plot the ANOVA — the plot blinds analysts from seeing the forms of coefficients' effects, e.g. linearity test results.

```
# 1
k <- 4

# 2
f <- psm(
  S ~ rcs(age, k) + sex + dzgroup + pol(num.co, 2) + scoma +
    pol(adlsc, 2) + race + rcs(meanbp, k) + rcs(hrt, k) +
    rcs(resp, k) + rcs(temp, k) + rcs(crea, 3) + rcs(sod, k) +
    rcs(wblc.i, k) + rcs(pafi.i, k),
  dist = 'lognormal'
)

# 3
plot(anova(f))
```



17 From the output from the model we see:

There were 537 observations, we didn't have any observations discarded due to missing data.

We had 356 deaths (events).

The standard deviation (sigma) of log survival time was estimated to be 2.23.

The model had a Likelihood Ratio Chi squared of 237 on 30 parameters (i.e, 30 degrees of freedom).

So, there's strong evidence that variables in the model are directly associated with patients' survival windows.

There is also a pseudo R squared of .594, which is a very strong R squared when you're trying to predict the exact date on which someone is going to die — this is unusual for prognostic models. The ICU population is very special in this regard because all of their body system functions were measured very carefully nearly each day.

The Somer's Dxy rank correlation between the predicted and log survival time is also very high at .485.

The discrimination index g sub r demonstrates the typical survival time ratio of two patients chosen at random — how different can you say they are on the basis of their covariates. The log normal survival model, instead of creating hazard ratios, creates survival time ratios, which are most cleanly thought of as ratios of predicted median survival times.

1. Here you can fit another log-normal survival model whose number of parameters (which correspond with the nonlinear effects of the previous model immediately above) you can determined from the ANOVA plot also immediately above.

1a. You can allocate five knots for the predictors which appear in the ANOVA which support this analysis best. You can use five knots because fewer singularity problems exist once you remove the predictors which contribute far less to the goal of the analysis.

```
# 1
# 1a
f <- psm(
  S ~ rcs(age, 5) + sex + dzgroup + num.co +
    scoma + pol(adlsc, 2) + race2 + rcs(meanbp, 5) +
    rcs(hrt, 3) + rcs(resp, 3) + temp +
    rcs(crea, 4) + sod + rcs(wblc.i, 3) + rcs(pafi.i, 4),
  dist = 'lognormal'
)

print(f, coefs = FALSE)

## Parametric Survival Model: Log Normal Distribution
##
## psm(formula = S ~ rcs(age, 5) + sex + dzgroup + num.co + scoma +
##      pol(adlsc, 2) + race2 + rcs(meanbp, 5) + rcs(hrt, 3) + rcs(resp,
##      3) + temp + rcs(crea, 4) + sod + rcs(wblc.i, 3) + rcs(pafi.i,
##      4), dist = "lognormal")
```

```
##
##               Model Likelihood      Discrimination
##               Ratio Test           Indexes
## Obs          537    LR chi2      236.83    R2      0.594
## Events       356    d.f.         30    Dxy      0.485
## sigma 2.230782    Pr(> chi2) <0.0001    g        1.959
##                                           gr        7.095
```

```
a <- anova(f)
print(a)
```

```
##               Wald Statistics           Response: S
##
## Factor          Chi-Square d.f. P
## age             15.99      4 0.0030
## Nonlinear        0.23      3 0.9722
## sex              0.11      1 0.7354
## dzgroup          45.69      2 <.0001
## num.co           4.99      1 0.0255
## scoma            10.58      1 0.0011
## adlsc            8.28      2 0.0159
## Nonlinear        3.31      1 0.0691
## race2            1.26      1 0.2624
## meanbp           27.62      4 <.0001
## Nonlinear        10.51      3 0.0147
## hrt              11.83      2 0.0027
## Nonlinear        1.04      1 0.3090
## resp             11.10      2 0.0039
## Nonlinear        8.56      1 0.0034
## temp             0.39      1 0.5308
## crea             33.63      3 <.0001
## Nonlinear        21.27      2 <.0001
## sod              0.08      1 0.7792
## wblc.i           5.47      2 0.0649
## Nonlinear        5.46      1 0.0195
## pafi.i           15.37      3 0.0015
## Nonlinear        6.97      2 0.0307
## TOTAL NONLINEAR  60.48     14 <.0001
## TOTAL            261.47     30 <.0001
```

18 Summary of the Fitted Model

Partial effects are very good because they handle non-linearity and they illustrate the whole estimated shape of each variable while changing one variable at a time.

The effects are plotted on the log survival scale.

All effects are centered so that they can be compared on a common scale.

This includes a Wald chi squared analysis penalized for degrees of freedom.

The panels made with ggplot show us the partial effects of each predictor, which are especially clean because we don't have interactions in this model.

Activities of daily living score is not very powerful as a predictor, chi squared with 2 df is 8.3, somewhat nonlinear.

Age effect is fairly linear and a stronger variable. The confidence intervals go to zero at the reference value for age, the median age. You can center these using an arbitrary reference to give an effect of zero which is the median (or mode for categorical variables) of the variable. Confidence intervals have to shrink when you're defining the median to be the comparison group.

Creatinine effect is extremely nonlinear and very powerful.

Mean bp has a strong effect.

It's easy to interpret all of these variables from this graph, and in essence they can be played against each other.

The categorical variables are shown in the dot plots.

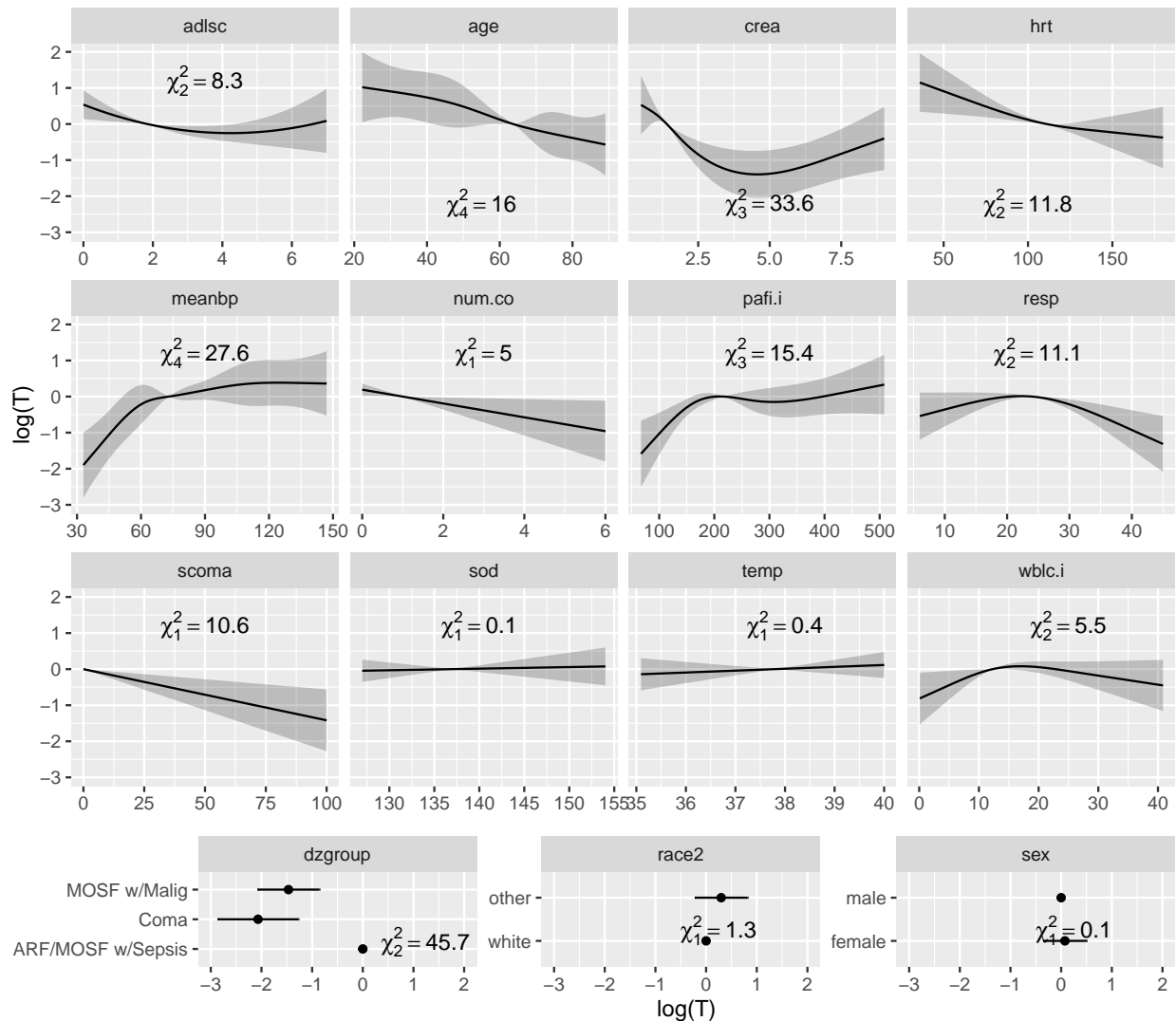
Disease group has a chi squared with 2 df of 45 so it's a very powerful variable adjusted for all the others.

Whereas the race and sex variables are not very relevant.

1. Predict() from the rms package computes predicted values and confidence limits. It allows the user to easily specify which predictors are to vary. Predicted values are the linear predictor (X beta), a user-specified transformation of that scale, or estimated probability of surviving past a fixed single time point given the linear predictor. Predict() is also usually used for plotting predicted values.

1a. This includes a ggplot method for Predict() objects which makes it easy to visualize predicted values and their confidence bands.

```
# 1
# 1a
ggplot(
  Predict(f, ref.zero = TRUE),
  vnames = 'names',
  sepdiscrte = 'vertical',
  anova = a
)
```



19 Wald Statistics

Then there are the nonlinear effects. Looking at serum creatinine, it has an overall chi squared with 3 df of 21.27 adjusted for everything else, which is highly significant. A lot of that significance came from its highly significant nonlinear effect.

Age was fitted to be pretty linear so all those degrees of freedom weren't needed — but that has to remain now because that would bias the analyses and make the confidence intervals too narrow.

The total chi squared for using Wald tests, is 261, which is not too far from the likelihood ratio chi squared of 236.8 in the Parametric Survival Model Log Normal Distribution table above.

The total nonlinearity is 60 with 14 degrees of freedom so, there is very good evidence that somewhere there is a non-linear effect among all the predictors.

```
print(a, size = 'tsz')
```

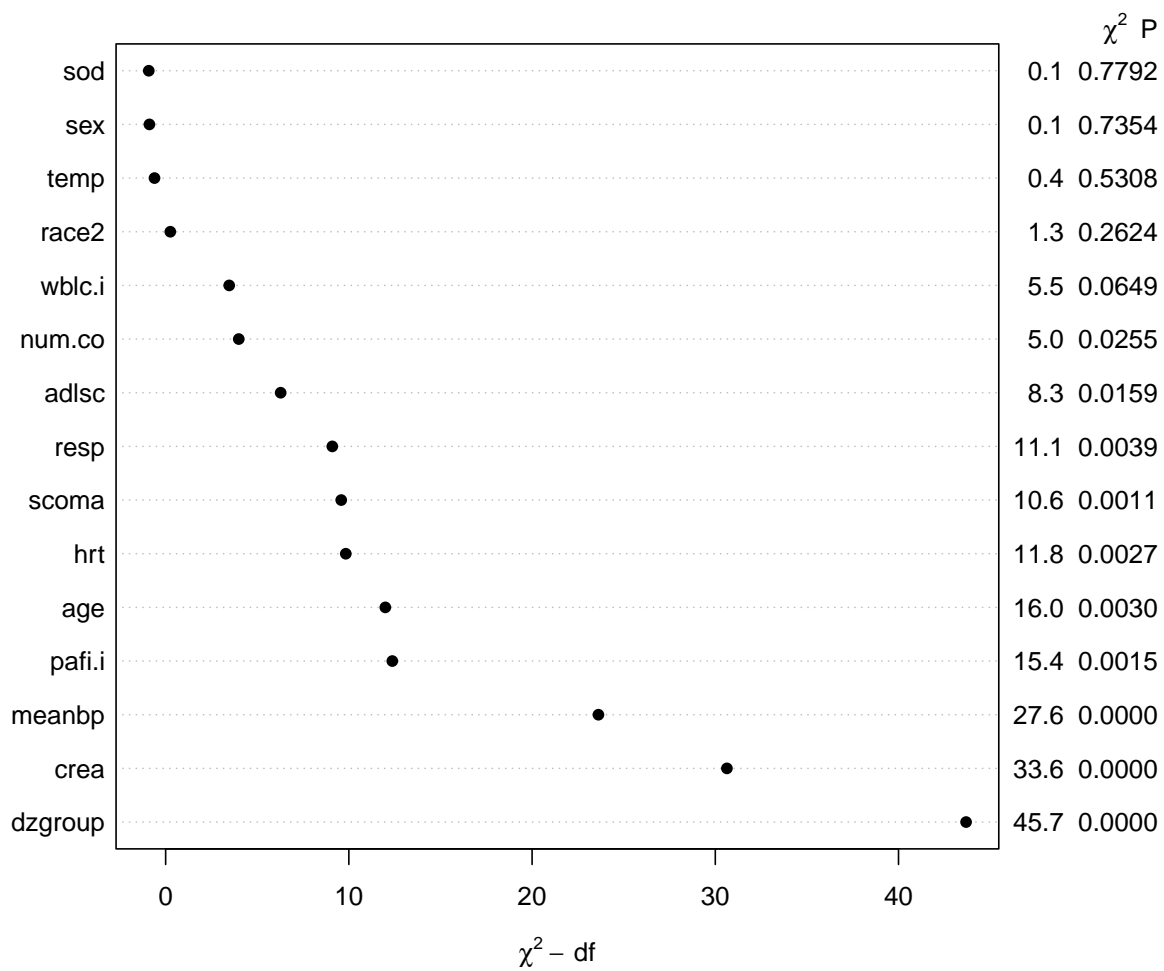
```
##                               Wald Statistics           Response: S
##
## Factor                        Chi-Square d.f. P
## age                          15.99      4  0.0030
##   Nonlinear                   0.23      3  0.9722
## sex                           0.11      1  0.7354
## dzgroup                       45.69      2 <.0001
## num.co                        4.99      1  0.0255
## scoma                         10.58      1  0.0011
## adlsc                         8.28      2  0.0159
##   Nonlinear                   3.31      1  0.0691
## race2                         1.26      1  0.2624
## meanbp                       27.62      4 <.0001
##   Nonlinear                   10.51      3  0.0147
## hrt                           11.83      2  0.0027
##   Nonlinear                   1.04      1  0.3090
## resp                         11.10      2  0.0039
##   Nonlinear                   8.56      1  0.0034
## temp                          0.39      1  0.5308
## crea                         33.63      3 <.0001
##   Nonlinear                   21.27      2 <.0001
## sod                           0.08      1  0.7792
## wblc.i                        5.47      2  0.0649
##   Nonlinear                   5.46      1  0.0195
## pafi.i                       15.37      3  0.0015
##   Nonlinear                   6.97      2  0.0307
## TOTAL NONLINEAR             60.48     14 <.0001
## TOTAL                       261.47     30 <.0001
```

20 Plotting the ANOVA

Plotting the ANOVA is where we see that disease group is the biggest predictor. This is similar to before when the model was saturated and you could assume more degrees of freedom for some of the variables.

This provides us with a brief look/snapshot of the variables involved in predicting time until death. They represent proxy features which biology has itself engineered with which we can indirectly observe important underlying processes in acutely ill patients.

```
plot(a)
```

21 The Summary function

The summary function is used with the option `log equals true` which produces a prediction with the log normal accelerated failure time model for log survival time — if you anti-log that you're getting predicted median survival time, and the effect ratios that are wanted are going to be ratios of median survival time. If the standard deviation is constant these are also ratios of mean survival time. Therefore I used a log scale for plotting.

It gives output with blue bars with point estimates shown with a triangle in the middle with various confidence intervals. For example, as age changes from its lower quartile to its upper quartile (47.9 to 74.5), there is a decrease in median survival time by over 50 percent. The bars' shaded areas refer to different levels of confidence (0.9, 0.95, 0.99).

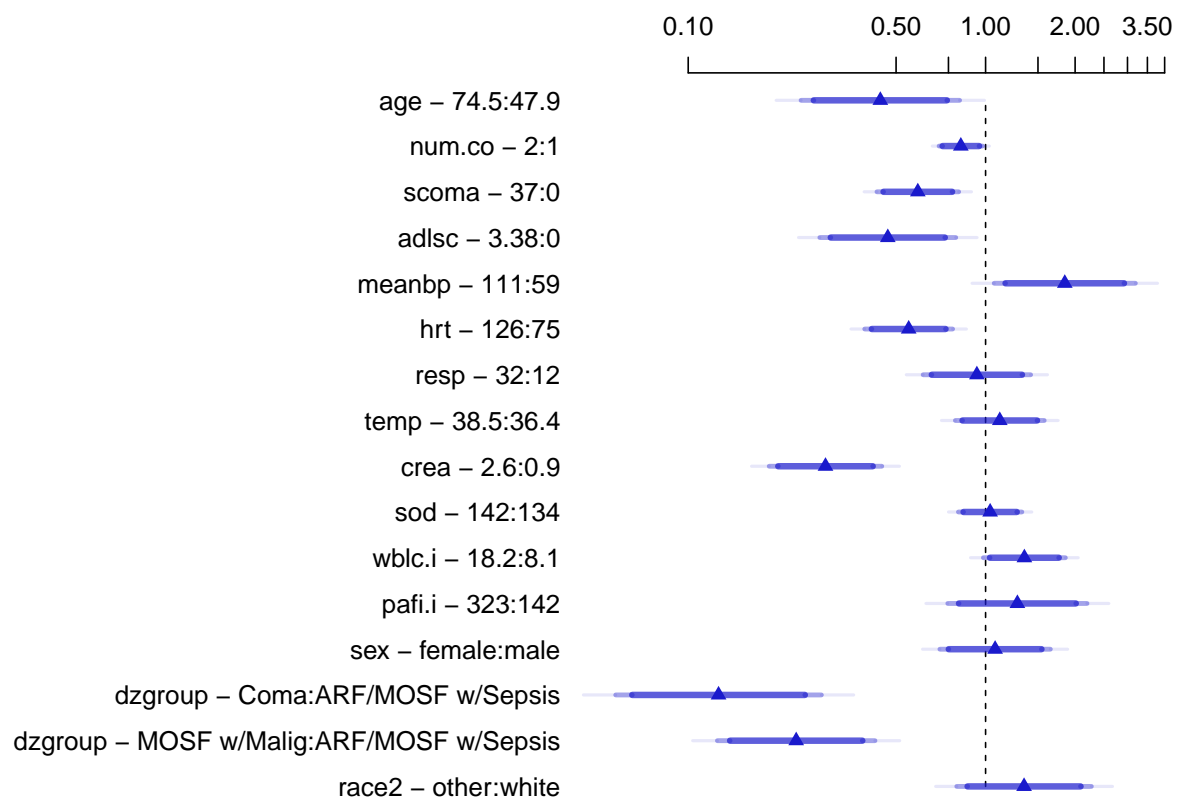
The biggest effect is shown in the difference between having acute respiratory failure with multiple organ system failure with sepsis compared to being septic and in a coma. Being septic and in a coma is worse, with survival cut by almost a factor of 10, so the ratio is a little bigger than 0.1.

1. `options()` base allows users to set and examine a variety of global options which affect the way in which R computes and displays its results.

2. `summary()` base is a generic function used to produce result summaries of the results of various model fitting functions. The function invokes particular methods which depend on the class of the first argument.

```
# 1
options(digits = 3)

# 2
plot(summary(f), log = TRUE, main = '')
```



22 Internal Validation of the Fitted Model Using the Bootstrap

The bootstrap is a simulation technique for studying properties of statistics without the need to have a theoretical infinite population available. The most common use of the bootstrap involves taking random samples (with replacement) from the original dataset and studying how some quantity of interest varies.

You can validate the model internally, which is a very rigorous process with a lot of advantages over external validation.

You can internally validate as soon as the model is finished if you've programmed all the steps used to obtain the model, then they can be included in the bootstrap.

This is a resampling based internal validation which validates all of the steps used to build the model. This process estimates the performance of the procedures you used to build the model. This is, coincidentally, also an estimate of the likely future performance of the model in its final form. External validation is useful when you don't trust the person validating the model to do so without bias. External validation also validates a model's measurement systems and a few other factors.

Here you can use the bootstrap to update the model fit to include the design matrix with all of the cubed terms along with all the indicator variables in the model.

This includes the log survival times and the fit object "g".

You can set a random number seed for reproducibility, then repeat for the recommended 400 iterations.

For each iteration the algorithm also computes the Somer's Dxy measure.

The validation results are as follows:

Somer's Dxy was .49. When sampling with replacement with the bootstrap sample and refitting the model some observations will be duplicated.

This will cause some overfit so the Somer's Dxy increases to .51. When the model is complete and applied to test data the optimism factor is eliminated and it goes back down to .46.

The bootstrap assumes that the difference between .51 and .46 (.46 representing usual overfitting, and .51 representing super overfitting), equals the difference between .49 and what you would like to know to subtract from .49 to give us the index that's corrected for all overfitting. `validate()` estimates that index to be .43 so, .43 represents the likely future performance for patients from the same stream as the types of patients modeled here.

The slope shrinks from 1.00 to .9 which represents an estimated regression to the mean of about 10 percent. This indicates that approximately 10 percent of everything we learned was noise.

So, overall `validate()` indicates a moderate amount of overfitting but the shrinkage factor of .9 is acceptable. The result is an unbiased estimate of the future predictive discrimination on equivalent patients as the corrected Dxy index of .43.

1. Adding data to the fitted model to allow the bootstrap to re-sample.
2. For reproducibility.
3. The `validate()` function, when used on an object created by one of the `rms` series functions, does resampling of the validation of a regression model, with or without backward step-down variable deletion.

```
# 1
g <- update(f, x = TRUE, y = TRUE)

# 2
set.seed(717)

# 3
suppressWarnings({
  print(validate(
    g,
    B = 400,
    dxy = TRUE
  ), digits = 2)
})
```

##	index.orig	training	test	optimism	index.corrected	n
## Dxy	0.49	0.51	0.46	0.05	0.43	400
## R2	0.59	NaN	NaN	NaN	NaN	0
## Intercept	0.00	0.00	-0.05	0.05	-0.05	400
## Slope	1.00	1.00	0.90	0.10	0.90	400
## D	0.48	NaN	NaN	NaN	NaN	0
## U	0.00	NaN	NaN	NaN	NaN	0
## Q	0.48	NaN	NaN	NaN	NaN	0
## g	1.96	2.06	1.87	0.19	1.77	400

23 Validate the absolute predictive accuracy

The next step is to validate the absolute predictive accuracy of the model on the survival probability scale.

Bootstrapping does this by taking into account all the sources of uncertainty, and focuses on validating one prediction only: the one year survival probability.

Calibration refers to the reliability of predicted values, i.e., the extent to which predicted values agree with observed values. For a predictive model such as this one, a calibration curve is constructed by relating predicted to observed values in some smooth manner. The calibration curve is judged against a 45-degree line. Miscalibration is also referred to as bias. Calibration error is frequently assessed for predicted event probabilities.

There are 2 ways to validate the calibration curve. The old way is stratifying the predictions, in this example with 60 patients per group, and within each group you get the average prediction for the group. It's very important not to use the midpoint but the average instead.

But that would bias ("bend") the data, which is always asking for trouble. Stratification involves "binning" of continuous variables which substantially reduces the amount of information in them.

To prevent this you use the bootstrap validation of the calibration curve which is the default in the `calibrate()` function from the `rms` package for survival models. This method is more precise because it effectively uses interpolation across levels of predicted values, unlike the binning/categorization which stratification uses, and which creates the noisy line with a lot of variability.

Here the dots represent the accuracy of the calibration. X's are estimates from the bootstrap with their overfitting corrected. The corrected overfittings were the result of predicted survival probabilities being binned as described immediately above, along with their Kaplan-Meier estimates. The black curve is the relationship of patients' conditions to their predicted survival estimates using `calibrate()`'s "bare" method, and the blue curve is the overfitting corrected estimate of the same, i.e. the actual observed outcomes. The 45 degree gray line is the "ideal" statistical relationship.

The blue curve is a little further from the 45 degree line than the black curve, and the blue curve is our estimate of future performance in terms of absolute prediction ability on the probability of surviving one year.

The smooth blue curve is far smoother than the stratified Kaplan-Meier estimates, which are using a sample size resulting in a lot of noise.

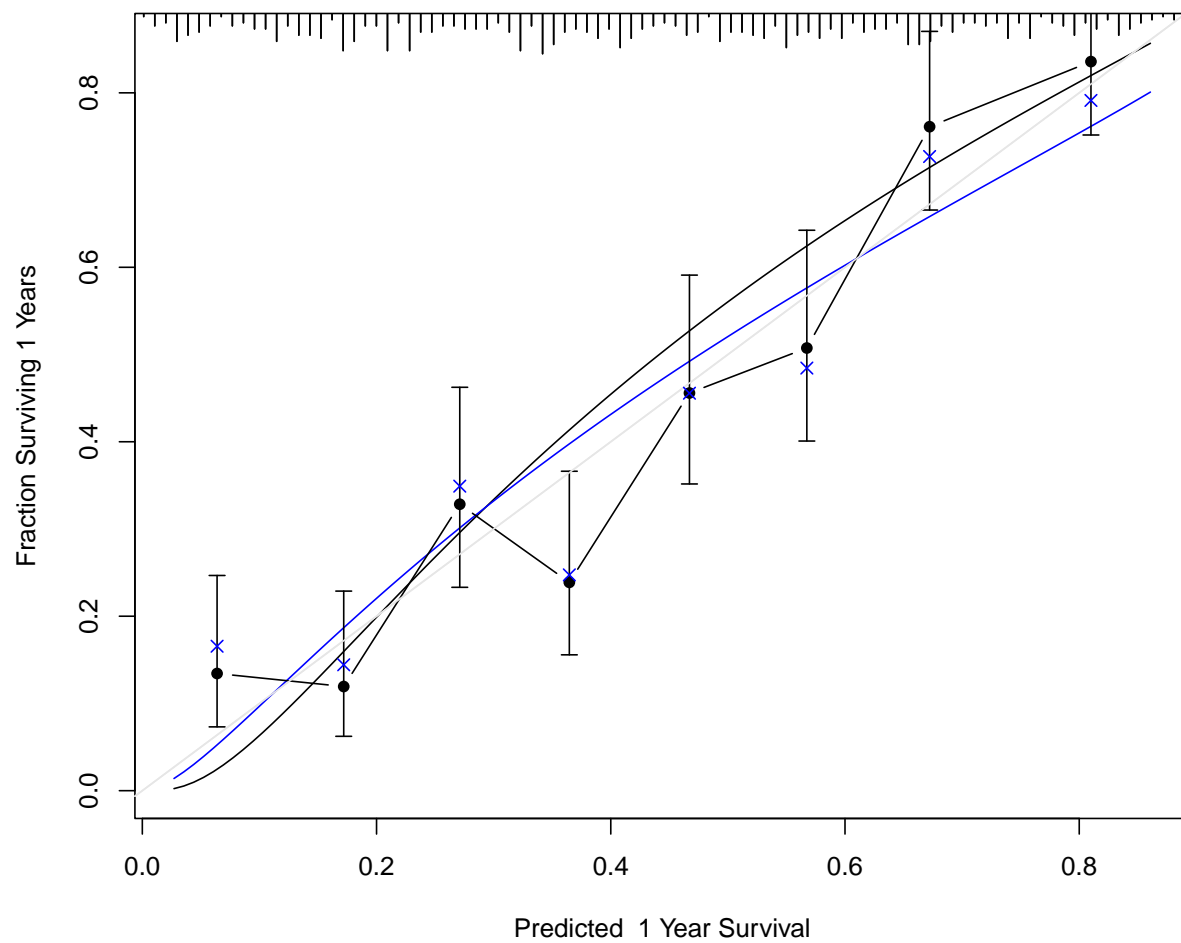
1. `calibrate()` uses bootstrapping here to get bias-corrected, or overfitting corrected, estimates of predicted vs. observed values based on subsetting predictions into intervals (for survival models) or on nonparametric smoothers (for other models).

```
# 1
suppressWarnings({
  set.seed(717)
  cal <- calibrate(g, u = 1, B = 400)
  plot(cal, subtitles = FALSE)
  cal <- calibrate(
    g,
    cmethod = 'KM',
```

```

u = 1,
m = 60,
B = 400,
pr = FALSE
)
plot(cal, add = TRUE)
})

```



24 Simplifying the Full Model

For some use cases parsimony is an important factor. Because it attenuates the quality of predicted discrimination it's a good idea to avoid it. And in the modern computer age you have predictions based on electronic medical record systems which don't need to be parsimonious. If you're trying to use something in the field, however, where you're trying to save time and money collecting data, there can be good use cases for parsimonious models.

If so and you want data to direct you in terms of which variables to discard and are not relying on outside and/or domain knowledge, the results have to be penalized. Bootstrapping will do that, and stepwise variable selection is generally relied on to develop a parsimonious model. It has a number of problems, however.

Here there's an opportunity to take an alternative method by taking the full model as the gold standard. From there you can decide to use the full model for statistical tests, confidence intervals, etc. An equally important decision is to ask whether you can approximate that model with high accuracy to produce a substitute predictive model?

When you're able to approximate the predictions from the full model with a less complex model, along with being able to describe to users what they're ceding when using a less complex model, this alternative approach can often be very helpful.

You can obtain the predictions on the log survival time scale from the model which was just applied to the dataset. Then you can obtain a representation of those predictions from the complete set of original variables.

You can stop at the point which the variables at the top of the list that are dropped are dispensable inasmuch as if we retained those top variables we could obtain predicted values that are accurately equivalent to our original gold standard predictions. This means ceasing the fast backward stepdown process before the R squared drops to 0.924.

This method would also have to be reproducible, but that's beyond the scope of this case study.

1. `predict()` from the stats package (with a lower case p) is a generic function for predictions from the results of various model fitting functions: $X \text{ times } \hat{\beta}$

$$X * \hat{\beta}$$

2. `ols()` from the rms package is a linear model estimation using ordinary least squares. It fits the usual weighted or unweighted linear regression model using the same fitting routines used by `lm()` (the stats package function for fitting linear models), but also storing the variance-covariance matrix and using traditional dummy-variable coding for categorical factors.

2a. Here you use `sigma = 1` to prevent `sigma hat` from being zero when $R^2 = 1$, which is the case here because you begin by approximating Z with all the variables from the full model.

3. `fastbw()` from the rms package performs a numerically stable version of fast backward elimination on factors. It uses a fitted complete model and computes approximate Wald statistics by computing conditional (restricted) maximum likelihood estimates assuming multivariate normality of estimates.

```
# 1 X times Beta hat
Z <- predict(f)
```



```

# 2
a <- ols(
  Z ~ rcs(age, 5) + sex + dzgroup + num.co +
    scoma + pol(adlsc, 2) + race2 +
    rcs(meanbp, 5) + rcs(hrt, 3) + rcs(resp, 3) +
    temp + rcs(crea, 4) + sod + rcs(wblc.i, 3) +
    rcs(pafi.i, 4),
  sigma = 1
)

# 3
fastbw(a, aics = 10000)

```

```

##
## Deleted Chi-Sq d.f. P      Residual d.f. P      AIC      R2
## sod      0.43 1      0.512      0.43 1      0.5117      -1.57 1.000
## sex      0.57 1      0.451      1.00 2      0.6073      -3.00 0.999
## temp     2.20 1      0.138      3.20 3      0.3621      -2.80 0.998
## race2     6.81 1      0.009     10.01 4      0.0402       2.01 0.994
## wblc.i    29.52 2      0.000     39.53 6      0.0000     27.53 0.976
## num.co    30.84 1      0.000     70.36 7      0.0000     56.36 0.957
## resp     54.18 2      0.000    124.55 9      0.0000    106.55 0.924
## adlsc     52.46 2      0.000    177.00 11     0.0000    155.00 0.892
## pafi.i    66.78 3      0.000    243.79 14     0.0000    215.79 0.851
## scoma     78.07 1      0.000    321.86 15     0.0000    291.86 0.803
## hrt       83.17 2      0.000    405.02 17     0.0000    371.02 0.752
## age       68.08 4      0.000    473.10 21     0.0000    431.10 0.710
## crea     314.47 3      0.000    787.57 24     0.0000    739.57 0.517
## meanbp    403.04 4      0.000   1190.61 28     0.0000   1134.61 0.270
## dzgroup   441.28 2      0.000   1631.89 30     0.0000   1571.89 0.000
##
## Approximate Estimates after Deleting Factors
##
##      Coef      S.E. Wald Z P
## [1,] -0.5928 0.04315 -13.74 0
##
## Factors in Final Model
##
## None

```

25 Fitting an approximate Ordinary Least Squares model

Here you have to fit an approximate Ordinary Least Squares model in order to predict the predicted values using the subset of variables which were retained in the less complex model. That model presents the same R squared (.957) as the full model.

You want to inherit the full model's variances to use for the confidence intervals of the less complex model. Because this uses maximum likelihood estimations with censored data (if there were no censored survival times this would just be ordinary least squares on log survival time) the full model and its unbiased variance estimates comes from the full model's fit.

You take the sum of squared residuals, divided by $n - p - 1$, in which p is the largest value we had entertained for the number of degrees of freedom for the prediction. So, the sigma squared figures which comes out of that needs to drive our final confidence intervals, and this approach will take care of that.

Now you can get the variance/covariance matrix for the approximate model, which is really derived from the variance/covariance matrix of the full model, as well as the linear contrasts which are used to map the full model coefficients into the approximate model coefficients. That's what the formula at the bottom of this code does, at "Mapping the full model's coefficients onto the approximate model coefficients".

You can see how this matters with the variance of certain variables using the approximate model versus the original model. You can do an ANOVA (analysis of variance) on this adjusted covariance matrix that you see under the Wald statistics table.

Analysis of variance usually refers to an analysis of a continuous dependent variable where all the predictor variables are categorical.

The equation for the simplified model is coming from this `f.approx` object — it's from an Ordinary Least Squares model and doesn't have to take censoring into account because we're predicting predicted values, not survival time.

This model is .957 accurate. This is the simplest form of the predictions and includes the expansion of the spline terms for the continuous variables.

This model is also the basis for the nomogram, which is coming up.


1. Using `ols()` again, as immediately above, at line 830.
2. Printing the new model's statistics.

```
# 1
f.approx <- ols(
  Z ~ dzgroup + rcs(meanbp, 5) + rcs(crea, 4) +
    rcs(age, 5) + rcs(hrt, 3) + scoma +
    rcs(pafi.i, 4) + pol(adlsc, 2) +
    rcs(resp, 3),
  x = TRUE
)

# 2
f.approx$stats
```

##	n	Model L.R.	d.f.	R2	g	Sigma
##	537.000	1688.225	23.000	0.957	1.915	0.370

26 Mapping the full model's coefficients onto the approximate model coefficients

1. This returns the variance-covariance matrix of the main parameters of a fitted model object — in this case the full model, `f` underscore 2. 
2. This creates the design of the full model.
3. This creates the design of the simplified approximate model.
4. This creates a contrast matrix for the full and approximate models.
5. Creating the variable containing the full model's variances to map onto the approximate model's variances below in the sections which produces Wald statistics for the simplified model.

```
# 1
V <- vcov(f, regcoef.only = TRUE)

# 2
X <- cbind(Intercept = 1, g$x)

# 3
x <- cbind(Intercept = 1, f.approx$x)

# 4
w <- solve(t(x) %*% x, t(x) %*% X)

# 5
v <- w %*% V %*% t(w)
```

27 Approximate model's estimated variances

Here you compare the variance estimates — the diagonals of `v` — with variance estimates from the reduced model fitted against the actual outcomes.

The approximate model's estimated variances are slightly smaller than those that you would have gotten from stepwise variable selection employing a stopping rule to select the same set of variables.

1. Fit the reduced model against the actual outcomes.
2. Comparing the variance estimates (the diagonals of `v`) with variance estimates from the simplified model fitted against the actual outcomes.
3. `diag()` from the base package extracts or replaces the diagonal of a matrix, or constructs a diagonal matrix. Here it constructs a diagonal matrix.

```
# 1
f.sub <- psm(
  S ~ dzgroup + rcs(meanbp, 5) + rcs(crea, 4) +
    rcs(age, 5) + rcs(hrt, 3) + scoma + rcs(pafi.i, 4) +
    pol(adlsc, 2) + rcs(resp, 3),
  dist = 'lognormal'
)

# 2
diag(v) / diag(vcov(f.sub, regcoef.only = TRUE))
```

```
##          Intercept          dzgroup=Coma dzgroup=MOSF w/Malig
##          0.981            0.979            0.979
##          meanbp          meanbp'          meanbp''
##          0.977            0.979            0.979
##          meanbp'''        crea            crea'
##          0.979            0.979            0.979
##          crea''          age            age'
##          0.979            0.982            0.981
##          age''          age'''          hrt
##          0.981            0.980            0.978
##          hrt'          scoma            pafi.i
##          0.976            0.979            0.980
##          pafi.i'        pafi.i''        adlsc
##          0.980            0.980            0.981
##          adlsc^2        resp            resp'
##          0.981            0.978            0.977
```

```
# 3
r <- diag(v) / diag(vcov(f.sub, regcoef.only = TRUE))
r[c(which.min(r), which.max(r))]
```

```
## hrt'    age
## 0.976 0.982
```

28 The Wald statistics for the reduced model

1. Mapping the full model's variances onto the approximate model's variances.
2. In the results you can note the similarities between these statistics and those from the Wald statistics table for the full model. Had they been very collinear with the kept/retained variables these similarities would not be the case.

```
# 1
f.approx$var <- v

# 2
print(anova(f.approx, test = 'Chisq', ss = FALSE), size = 'tsz')
```

```
##          Wald Statistics          Response: Z
##
## Factor      Chi-Square d.f. P
## dzgroup      55.94      2  <.0001
```

##	meanbp	29.87	4	<.0001
##	Nonlinear	9.84	3	0.0200
##	crea	39.04	3	<.0001
##	Nonlinear	24.37	2	<.0001
##	age	18.12	4	0.0012
##	Nonlinear	0.34	3	0.9517
##	hrt	9.87	2	0.0072
##	Nonlinear	0.40	1	0.5289
##	scoma	9.85	1	0.0017
##	pafi.i	14.01	3	0.0029
##	Nonlinear	6.66	2	0.0357
##	adlsc	9.71	2	0.0078
##	Nonlinear	2.87	1	0.0904
##	resp	9.65	2	0.0080
##	Nonlinear	7.13	1	0.0076
##	TOTAL NONLINEAR	58.08	13	<.0001
##	TOTAL	252.32	23	<.0001

29 The final equation on which the nomogram is based:

$$E(Z) = X * \beta, \text{ where}$$

$$\begin{aligned}
X * \hat{\beta} = & -2.51 - 1.94[\text{Coma}] - 1.75[\text{MOSF w/Malig}] + 0.068\text{meanbp}_+ \\
& -3.08 \times 10^{-5}(\text{meanbp} - 41.8)_+^3 + 7.9 \times 10^{-5}(\text{meanbp} - 61)_+^3 \\
& -4.91 \times 10^{-5}(\text{meanbp} - 73)_+^3 + 2.61 \times 10^{-6}(\text{meanbp} - 109)_+^3 \\
& -1.7 \times 10^{-6}(\text{meanbp} - 135)_+^3 - 0.553\text{crea} - 0.229(\text{crea} - 0.6)_+^3 \\
& +0.45(\text{crea} - 1.1)_+^3 - 0.233(\text{crea} - 1.94)_+^3 + 0.0131(\text{crea} - 7.32)_+^3 \\
& -0.0165\text{age} - 1.13 \times 10^{-5}(\text{age} - 28.5)_+^3 + 4.05 \times 10^{-5}(\text{age} - 49.5)_+^3 \\
& -2.15 \times 10^{-5}(\text{age} - 63.7)_+^3 - 2.68 \times 10^{-5}(\text{age} - 72.7)_+^3 + 1.9 \times 10^{-5}(\text{age} - 85.6)_+^3 \\
& -0.0136\text{hrt} + 6.09 \times 10^{-7}(\text{hrt} - 60)_+^3 - 1.68 \times 10^{-6}(\text{hrt} - 111)_+^3 \\
& +1.07 \times 10^{-6}(\text{hrt} - 140)_+^3 - 0.0135\text{scoma} + 0.0161\text{pafi.i} \\
& -4.77 \times 10^{-7}(\text{pafi.i} - 88)_+^3 + 9.11 \times 10^{-7}(\text{pafi.i} - 167)_+^3 \\
& -5.02 \times 10^{-7}(\text{pafi.i} - 276)_+^3 + 6.76 \times 10^{-8}(\text{pafi.i} - 426)_+^3 - 0.369\text{adlsc}_+ \\
& +0.0409\text{adlsc}^2 + 0.0394\text{resp} - 9.11 \times 10^{-5}(\text{resp} - 10)_+^3 \\
& +0.000176(\text{resp} - 24)_+^3 - 8.5 \times 10^{-5}(\text{resp} - 39)_+^3 \\
& \text{and } [c] = 1 \text{ if subject is in group (c),} \\
& 0 \text{ otherwise } (x)_+ = x \text{ if } x > 0, \text{ 0 otherwise}
\end{aligned}$$

30 A simpler representation of the model

Now you've cut out variables which are not accurately predictive of the predicted values from the full model. Once you fit your model you can get more types of predictive quantities from a parametric model than you can from a Cox Proportional Hazards semi-parametric model. You can't really get a mean survival time from a Cox model unless everyone is followed until death or perhaps if your censoring for those still alive is very, very late.

With a parametric model it's able to extrapolate — which is not always apposite — but which allows you to get the area under the whole survival curve which is the mean survival time. You can also get any quantiles of survival time.

You derive a function using the mean function (capital M, noted in the code below) which represents the predicted mean. On the original scale you're going to derive a function which can be used to translate linear predictors into quantiles. It's taking the linear predictor and considering the log sigma value from the model, i.e. it's considering both the mean and the standard deviation on the log scale, which are needed to get the mean on the unlog scale. The quantile function is just solving for when the survival curve hits a certain level.

You can also improve the labels for the axes of the nomogram, which I did.

Next you apply the `nomogram()` function on the approximate model, control where some of the tick marks are placed, and add some functions to the normal output: the median survival time function, and the mean survival time function.

Then you plot that and we see the nomogram in which all the variables have been mapped into a common 0-100 point scale. You see that among the disease groups Acute Respiratory Failures, Multiple Organ Systems Failures w/Sepsis (ARF/MOSF s/Sepsis) actually has the highest survival time, but this is before being adjusted for all the other variables that must be taken into account for how sick the patients are in this group.

You see that mean arterial blood pressure is fairly nonlinear, and a very strong effect because it has a very stretched axis.

Serum creatinine is a nonmonotonic function so it has the worst effect on survival at its lowest value of 5, then as it increases it positively affects survival. Here's an example of where domain knowledge within a multivariate context like this provides additional interpretive help. It's "good" in this patient context to have a high creatinine level because almost all patients with high creatinine are on dialysis and have to already be stable to be so — dialysis is able to keep them living in the short term very well. That's why you get a strong survival effect on the right end of the creatinine scale.

Age is pretty linear, as is heart rate and the coma score.

Lung function is not monotonic.

You have a weaker variable, ADL, and respiratory rate is not monotonic.

You add up the points you get from each variable, calculate total points, and immediately read off the linear predictor which is the patient's predicted log median survival time. When you antilog that you get the predicted median survival time.

If you look at where the linear predictor is zero and carry that straight down the median has to be 1 because the antilog of zero is 1.

The mean survival time when the median is 1 is about 10 years, so the distribution that we're fitting is highly skewed with a very heavy right tail, making the mean much higher than the median survival time. But you can learn a lot about the model and how you can predict different outcome metrics from this nomogram.

1. Derive S functions that express means and quantiles of survival time for specific linear predictors analytically. Mean is a generic function that returns an R function to compute the estimate of the mean of a variable. Its input is typically some kind of model fit object, as you have here.

2. Improve variable labels for the nomogram.

3. Implement the nomogram.

- 3a. `nomogram()` from the `rms` package draws a partial nomogram that can be used to manually obtain predicted values from a regression model that was fitted with `rms`. The nomogram does not have lines representing sums, but it has a reference line for reading scoring points (default range 0–100). Once the reader manually totals the points, the predicted values can be read at the bottom. Non-monotonic transformations of continuous variables are handled (scales wrap around), as are transformations which have flat sections (tick marks are labeled with ranges). If interactions are in the model, one variable is picked as the "axis variable", and separate axes are constructed for each level of the interacting factors (preference is given automatically to using any discrete factors to construct separate axes) and levels of factors which are indirectly related to interacting factors. Thus the nomogram is designed so that only one axis is actually read for each variable, since the variable combinations are disjoint.

4. Plot it.

```
# 1
expected.surv <- Mean(f)
```

```

quantile.surv <- Quantile(f)
median.surv  <- function(x)
  quantile.surv(lp = x)

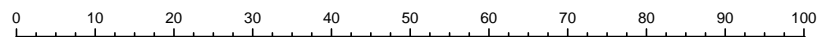
# 2
f.approx <- Newlabels(
  f.approx,
  c(
    'Disease Group',
    'Mean Arterial BP',
    'Creatinine',
    'Age',
    'Heart Rate',
    'SUPPORT Coma Score',
    'PaO2/(.01*FiO2)',
    'ADL',
    'Resp. Rate'
  )
)

# 3
nom <-
  nomogram(
    f.approx,
    pafi.i = c(0, 50, 100, 200, 300, 500, 600, 700, 800, 900),
    fun = list(
      'Median Survival Time' = median.surv,
      'Mean Survival Time'  = expected.surv
    ),
    fun.at = c(.1, .25, .5, 1, 2, 5, 10, 20, 40)
  )

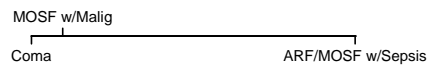
# 4
plot(nom,
  cex.var = 1,
  cex.axis = .6,
  lmgp = .25)

```

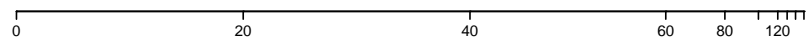

Points



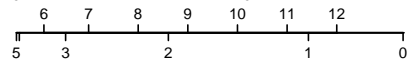
Disease Group



Mean Arterial BP



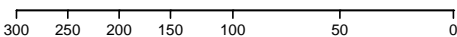
Creatinine



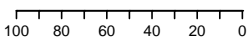
Age



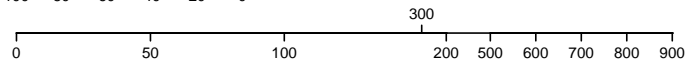
Heart Rate



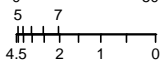
SUPPORT Coma Score



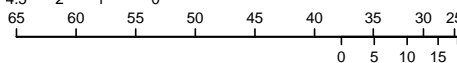
PaO2/(.01*FiO2)



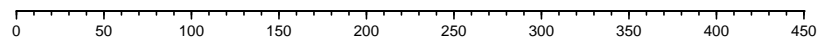
ADL



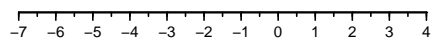
Resp. Rate



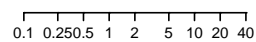
Total Points



Linear Predictor



Median Survival Time



Mean Survival Time

