Practical Predictive Analytics Seminar: Data preparation

Contents

Intro 1
Set up R's environment 1
1. Data assessment/Formatting variables 1
2. Identify outliers and missing data 2
3. Create new variables 6
4. Year-by-year dataset 6

Intro

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

Set up R's environment

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

```
library(dplyr)
library(ggplot2)
library(car)
library(zoo)
library(lubridate)
options(tinytex.verbose = T)
```

1. Data assessment/Formatting variables

Import data sets and bind together:

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

Take a quick peak at the field names:

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

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

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

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

```
date.form <- "%Y-%m-%d"
data.full$dob <- as.Date(as.character(data.full$dob),
                         date.form)
# dplyr alternative
# data.full <- data.full %>%
#   mutate(dob = as.Date(as.character(data.full$dob),
#                        date.form))
data.full$dod <- as.Date(as.character(data.full$dod),
                         date.form)
data.full$studyenddate <- as.Date(as.character(data.full$studyenddate),
                                  date.form)
data.full$enteredstudydate <- as.Date(as.character(data.full$enteredstudydate),
                                      date.form)
```

Check automatically for character fields, and change to factor vectors. This is not absolutely required, but it's nice when R treats these types of categorical variables in the same way.

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

2. Identify outliers and missing data

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

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

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

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

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

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

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

```
## [1] 4700
```

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

```
data.full <- data.full %>%
  mutate(death_ind = ifelse(!is.na(dod), 1, 0),
         timetodeath2 = ifelse(!is.na(dod),
```

```
                                (dod - enteredstudydate)/365.25, 0))
mean(abs(data.full$timetodeath - data.full$timetodeath2) < 0.01)
```

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

```
#  ~4-day error tolerance
```

10,739 people do not have height or weight (and it is the same people that are missing both). Additionally, those two fields are factors! Let's make them numeric, and then impute the missing values with a flag field to mark them for later. To convert factors to numerics, you have to first convert to characters as we do below.

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

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

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

Many of the disease fields have blanks. It's probably fair to assume that this disease was never reported for those people, and we'll want to supply an "unreported" factor level in place of the blank. Here's a quick, automated way to do that. Note that we also "relevel" the new factor variables so that "unreported" becomes the baseline/reference level.

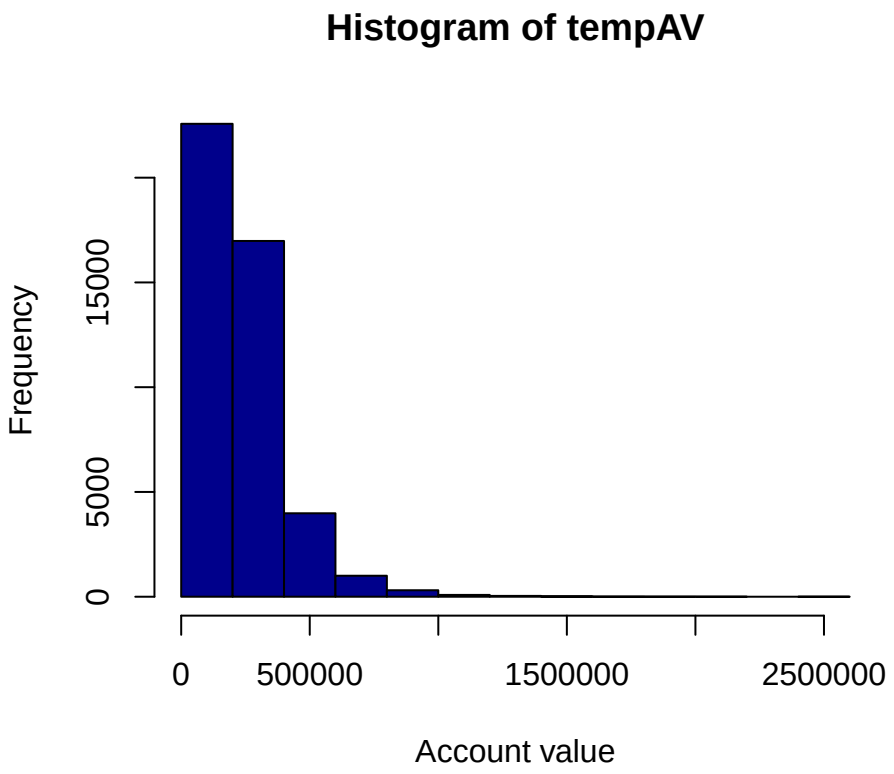
```
(blank.replace <- which(sapply(data.full, function(x){
  ifelse(class(x) == "factor", sum(x == ""), 0)}) > 0))
for(i in blank.replace){
  new.labels <- c("unreported", levels(data.full[,i])[-1])
  data.full[,i] <- relevel(factor(data.full[,i], labels = new.labels),
                           ref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

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

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

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

```
tempAV <- exp(rnorm(n = 45000, mean = 0, sd = .6))*200000
hist(tempAV, xlab = "Account value", col = "blue4")
```

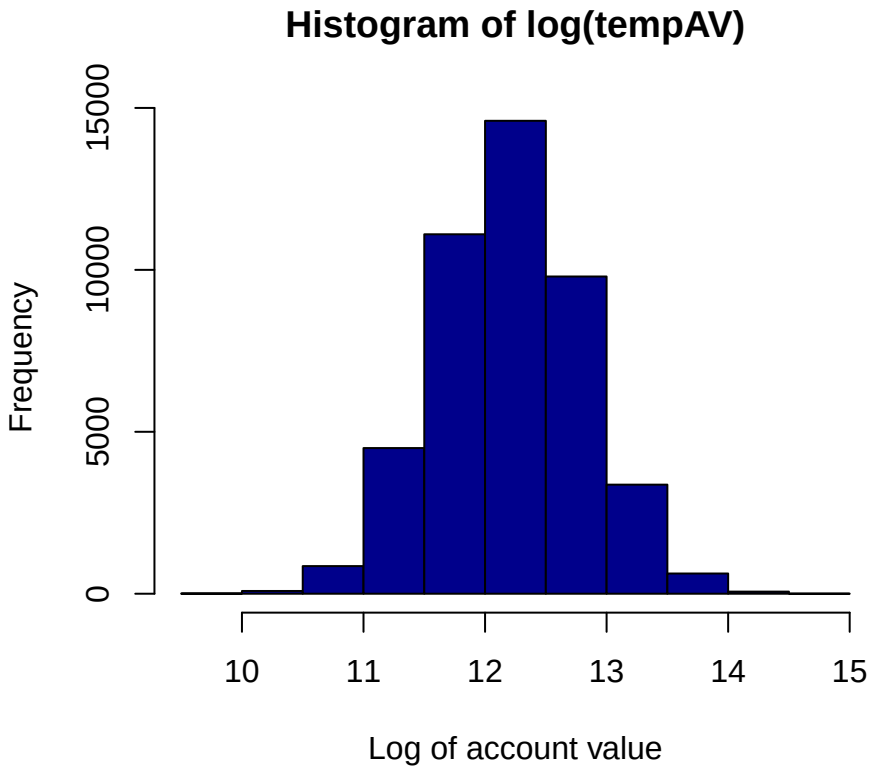


Note that the gap between the maximum value and the 99th percentile value is really wide. For skewed distributions this is often the case, and those outliers can have too much influence on linear models. What we often use in modeling is the natural logarithm of these right-skewed data. This specific AV column was fabricated such that the natural log would generate a normal distribution, so it's not surprising the see the bell shape. But trust us that using log transformations produces more Gaussian distributions when applied to right skewed data in the real world, as well.

```
max(tempAV) - quantile(tempAV,.99)
```

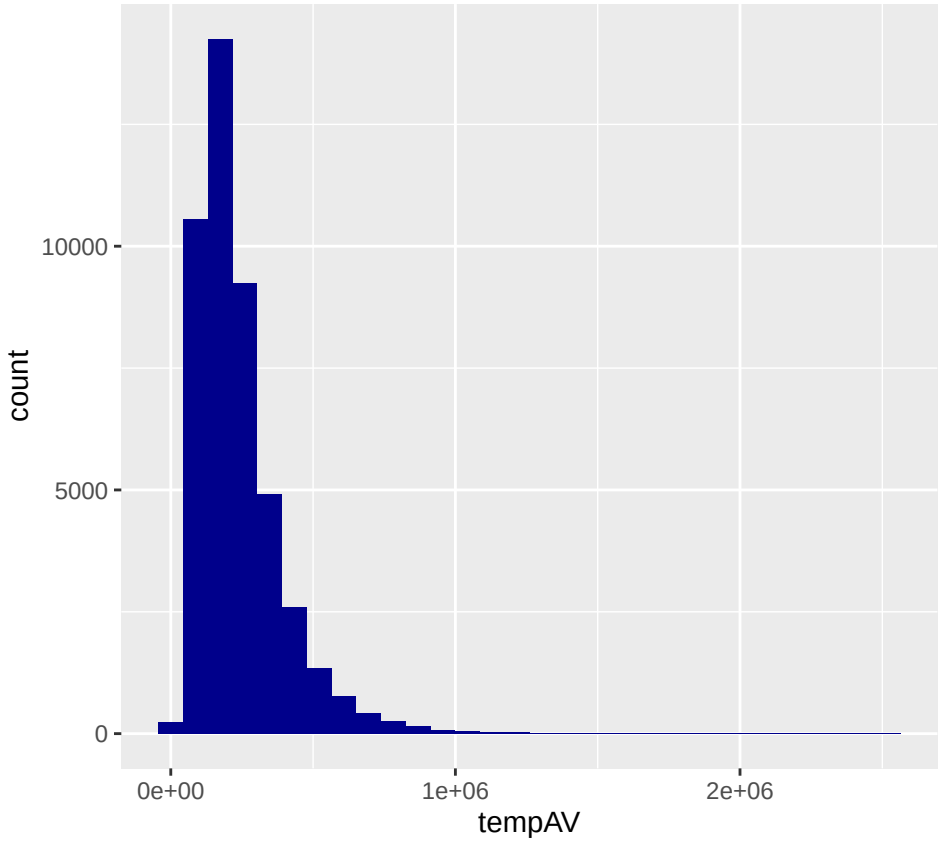
```
##     99%
## 1733297
```

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




```
# ggplot alternative
data.frame(tempAV) %>%
  ggplot(aes(x = tempAV)) +
  geom_histogram(fill = "blue4")
```

```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```



3. Create new variables

"Mutate" a new field for BMI, and squared BMI distance from 27.23 (the average BMI of the group). Also, let's create a grouped variable for all cancers since some are quite rare, and a healthy indicator for those without disease.

```
data.full <- data.full %>%
  mutate(bmi = ifelse(ht.wt.flag == 0, 0, (weight/height^2)*703),
         bmisqrerr = ifelse(ht.wt.flag == 0, 0, (bmi - 27.23)^2))
(cancer_columns <- grep("cancer", names(data.full)))
```

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

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

4. Year-by-year dataset

Create a year-by-year dataset so that we can observe policy holders one year at a time. First, we'll determine each observation's number of years in the dataset so that we know how many rows to make.

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

```
data.full[["PolNum"]] <- 1:45000
data.large <- data.frame(PolNum = rep(data.full$PolNum, data.full$years))
data.large <- data.large %>%
  group_by(PolNum) %>%
  mutate(PolYear = 1:n()) %>%
  ungroup() # Leaving the data frame as ungrouped (after grouping) can help to avoid future issues
```

Join our original static policy information onto our expanded dataset.

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

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

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

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

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

Save dataset for modeling and validation steps. We save it in two formats (.Rdata and .rds) for demonstration. You can read .rds files into the environment with any object name you want, but you can only save one R object per .rds file. With .Rdata files, you can save many R objects into a single file, but you're stuck with the object names you gave them originally. Which method you choose will depend on how you plan to use the objects in the future.

```
save(data.large, file = "PPASExpandedData.Rdata")
saveRDS(data.large, file = "PPASExpandedData.rds")
```