Homework 3

BUSN 41204 - 2023

- Aman Krishna
- Christian Pavilanis
- Jingwen Li
- Yazmin Ramirez Delgado

In [340]:
```
knitr::opts_chunk$set(eval = FALSE)
```

Due: end of day Saturday, February 4

Submission instructions: Submit one write-up per group on gradescope.com (gradescope.com).

IMPORTANT:

- Write names of everyone that worked on the assignment on the submission.
- Specify every member of the group when submitting on Gradescope (https://help.gradescope.com/article/m5qz2xsnjy-student-add-group-members (https://help.gradescope.com/article/m5qz2xsnjy-student-add-group-members))

For this homework, we will be using the case *Retention Modeling at Scholastic Travel Company*. Read:

- Case: Retention Modeling at Scholastic Travel Company (A);
- Supplement: Retention Modeling at Scholastic Travel Company (B);

which are available on Canvas.

Your goal is to help David build a model for retention.

The following code will get you started.

Load relevant libraries

In [341]:
```
library(dplyr)
library(caret)
library(glmnet)
```

Load the data

Here we will load the data from the CSV data file, examine its structure, and fix the data types incorrectly identified by R when importing from CSV.

In [342]:
```
STCdata_A<-read.csv('travelData.csv')
STCdata_A<-STCdata_A[,-1]
```

You can use the function `str` to quickly check the internal structure of an R object. Here we are using it to investigate type of data in each column of the loaded data.

In [343]:
```
str(STCdata_A)
```

```
'data.frame':	2389 obs. of  55 variables:
 $ Program.Code                  : chr  "HS" "HC" "HD" "HN" ...
 $ From.Grade                    : int  4 8 8 9 6 10 11 9 8 8 ...
 $ To.Grade                      : int  4 8 8 12 8 12 12 9 8 8 ...
 $ Group.State                   : chr  "CA" "AZ" "FL" "VA" ...
 $ Is.Non.Annual.                : int  0 0 0 1 0 0 1 0 0 0 ...
 $ Days                          : int  1 7 3 3 6 4 6 8 8 4 ...
 $ Travel.Type                   : chr  "A" "A" "A" "B" ...
 $ Departure.Date                : chr  "1/14/2011" "1/14/2011" "1/15/2011" "1/15/2011" ...
 $ Return.Date                   : chr  "1/14/2011" "1/21/2011" "1/17/2011" "1/17/2011" ...
 $ Deposit.Date                  : chr  "8/30/2010" "11/15/2009" "10/15/2010" "1/7/2011" ...
 $ Special.Pay                   : chr  NA "CP" NA NA ...
 $ Tuition                       : int  424 2350 1181 376 865 2025 1977 3379 2200 1428 ...
 $ FRP.Active                    : int  25 9 17 0 40 9 16 10 30 51 ...
 $ FRP.Cancelled                 : int  3 9 6 0 8 4 4 0 0 1 ...
 $ FRP.Take.up.percent.          : num  0.424 0.409 0.708 0 0.494 0.9 0.64 0.769 0.577 0.773 ...
 $ Early.RPL                     : chr  "3/29/2010" "10/20/2009" "4/29/2010" NA ...
 $ Latest.RPL                    : chr  "8/12/2010" "8/10/2010" "8/16/2010" NA ...
 $ Cancelled.Pax                 : int  3 11 6 1 9 3 5 1 0 1 ...
 $ Total.Discount.Pax            : int  4 3 3 0 8 1 2 1 4 6 ...
 $ Initial.System.Date           : chr  "3/26/2010" "10/2/2009" "1/28/2010" "10/19/2010" ...
 $ Poverty.Code                  : chr  "B" "C" "C" "" ...
 $ Region                        : chr  "Southern California" "Other" "Other" "Other" ...
 $ CRM.Segment                   : int  4 10 10 7 10 8 8 7 5 5 ...
 $ School.Type                   : chr  "PUBLIC" "PUBLIC" "PUBLIC" "CHD" ...
 $ Parent.Meeting.Flag           : int  1 1 1 0 1 1 1 1 1 1 ...
 $ MDR.Low.Grade                 : chr  "K" "7" "6" "" ...
 $ MDR.High.Grade                : int  5 8 8 NA 8 12 12 NA 12 8 ...
 $ Total.School.Enrollment       : int  927 850 955 NA 720 939 225 NA 500 635 ...
 $ Income.Level                  : chr  "Q" "A" "O" "" ...
 $ EZ.Pay.Take.Up.Rate           : num  0.17 0.091 0.042 0 0.383 0.1 0.08 0 0.231 0.136 ...
 $ School.Sponsor                : int  1 0 0 0 0 0 0 0 0 0 ...
 $ SPR.Product.Type              : chr  "CA History" "East Coast" "East Coast" "East Coast" ...
 $ SPR.New.Existing              : chr  "EXISTING" "EXISTING" "EXISTING" "EXISTING" ...
 $ FPP                           : int  59 22 24 18 81 10 25 13 52 66 ...
 $ Total.Pax                     : int  63 25 27 18 89 11 27 14 56 72 ...
 $ SPR.Group.Revenue             : int  424 2350 1181 376 865 2025 1977 3379 2200 1428 ...
 $ NumberOfMeetingswithParents   : int  1 2 1 0 1 1 1 1 1 1 ...
 $ FirstMeeting                  : chr  "8/12/2010" "11/17/2009" "9/13/2010" NA ...
 $ LastMeeting                   : chr  "8/12/2010" "8/27/2010" "9/13/2010" NA ...
 $ DifferenceTraveltoFirstMeeting: int  155 423 124 NA 145 91 63 138 143 146 ...
 $ DifferenceTraveltoLastMeeting : int  155 140 124 NA 145 91 63 138 143 146 ...
 $ SchoolGradeTypeLow            : chr  "Elementary" "Middle" "Middle" "High" ...
 $ SchoolGradeTypeHigh           : chr  "Elementary" "Middle" "Middle" "High" ...
 $ SchoolGradeType               : chr  "Elementary->Elementary" "Middle->Middle" "Middle->Middle" "High->High" ...
 $ DepartureMonth                : chr  "January" "January" "January" "January" ...
 $ GroupGradeTypeLow             : chr  "K" "Middle" "Middle" "Undefined" ...
 $ GroupGradeTypeHigh            : chr  "Elementary" "Middle" "Middle" "Undefined" ...
 $ GroupGradeType                : chr  "K->Elementary" "Middle->Middle" "Middle->Middle" "Undefined->Undefined" ...
 $ MajorProgramCode              : chr  "H" "H" "H" "H" ...
 $ SingleGradeTripFlag           : int  1 1 1 0 0 0 0 1 1 1 ...
 $ FPP.to.School.enrollment      : num  0.0636 0.0259 0.0251 NA 0.1125 ...
 $ FPP.to.PAX                    : num  0.937 0.88 0.889 1 0.91 ...
 $ Num.of.Non_FPP.PAX            : int  4 3 3 0 8 1 2 1 4 6 ...
 $ SchoolSizeIndicator           : chr  "L" "L" "L" "" ...
 $ Retained.in.2012.             : int  1 1 1 0 0 1 0 0 1 1 ...
```

Notice that some columns are identified as numerical or integer, but really the should be factors.

For instance, we have that column `From.Grade`

In [344]:
```
n_distinct(STCdata_A$From.Grade, na.rm = FALSE)    ## n_distinct is a function from dplyr package
```

11

only has 11 levels. It might be a better idea to treat it as a factor instead.

You can fix incorrectly classified data types as follows:

In [345]:
```
STCdata_A <- mutate_at(STCdata_A, vars(From.Grade), as.factor)
```

We can check that indeed the column represents a factor:

In [346]:
```
str( STCdata_A$From.Grade )
```

```
 Factor w/ 10 levels "3","4","5","6",..: 2 6 6 7 4 8 9 7 6 6 ...
```

Fix other columns that are numeric at the moment, but could be converted to factors. The following line first finds numeric columns and then identifies the number of unique elements in each one.

In [347]:
```
( unique.per.column <- sapply( dplyr::select_if(STCdata_A, is.numeric), n_distinct ) )
```

To.Grade
11
Is.Non.Annual.
2
Days
12
Tuition
1230
FRP.Active
93
FRP.Cancelled
29
FRP.Take.up.percent.
476
Cancelled.Pax
34
Total.Discount.Pax
26
CRM.Segment
12
Parent.Meeting.Flag
2
MDR.High.Grade
13
Total.School.Enrollment
894
EZ.Pay.Take.Up.Rate
371
School.Sponsor
2
FPP
146
Total.Pax
159
SPR.Group.Revenue
1230
NumberOfMeetingswithParents
3
DifferenceTraveltoFirstMeeting
343
DifferenceTraveltoLastMeeting
252
SingleGradeTripFlag
2
FPP.to.School.enrollment
1910
FPP.to.PAX
306
Num.of.Non_FPP.PAX
26
Retained.in.2012.
2

Let us convert every column that has less than 15 unique values into a factor. The following line identify names of such columns.

In [348]:
```
( column.names.to.factor <- names(unique.per.column)[unique.per.column < 15] )
```

'To.Grade' 'Is.Non.Annual.' 'Days' 'CRM.Segment' 'Parent.Meeting.Flag' 'MDR.High.Grade' 'School.Sponsor' 'NumberOfMeetingswithParents' 'SingleGradeTripFlag' 'Retained.in.2012.'

From this, we can see that the columns `To.Grade` , `Is.Non.Annual.` , `Days` , `CRM.Segment` , `Parent.Meeting.Flag` , `MDR.High.Grade` , `School.Sponsor` , `NumberOfMeetingswithParents` , `SingleGradeTripFlag` can be converted to factors. We can also convert the output `Retained.in.2012.`

Convert these columns into factors.

In [349]:
```
STCdata_A <- mutate_at(STCdata_A, column.names.to.factor, as.factor)
```

Now let's take care of date columns.

In [350]:
```
date.columns = c('Departure.Date', 'Return.Date', 'Deposit.Date', 'Early.RPL', 'Latest.RPL',
                 'Initial.System.Date', 'FirstMeeting', 'LastMeeting')
STCdata_A <- mutate_at(STCdata_A, date.columns, function(x) as.Date(x, format = "%m/%d/%Y"))
```

And finally we change all the character columns to factors as well.

In [351]:
```
STCdata_A <- mutate_if(STCdata_A, is.character, as.factor)
```

Let's see what we have:

In [352]:
```
str(STCdata_A)
```

```
'data.frame':	2389 obs. of  55 variables:
 $ Program.Code                  : Factor w/ 28 levels "CC","CD","CN",..: 15 6 7 12 7 6 25 5 1 7 ...
 $ From.Grade                    : Factor w/ 10 levels "3","4","5","6",..: 2 6 6 7 4 8 9 7 6 6 ...
 $ To.Grade                      : Factor w/ 10 levels "3","4","5","6",..: 2 6 6 10 6 10 10 7 6 6 ...
 $ Group.State                   : Factor w/ 54 levels "AB","AK","AL",..: 7 5 11 49 11 20 21 29 5 47 ...
 $ Is.Non.Annual.                : Factor w/ 2 levels "0","1": 1 1 1 2 1 1 2 1 1 1 ...
 $ Days                          : Factor w/ 12 levels "1","2","3","4",..: 1 7 3 3 6 4 6 8 8 4 ...
 $ Travel.Type                   : Factor w/ 4 levels "A","B","N","T": 1 1 1 2 4 1 1 1 1 1 ...
 $ Departure.Date                : Date, format: "2011-01-14" "2011-01-14" ...
 $ Return.Date                   : Date, format: "2011-01-14" "2011-01-21" ...
 $ Deposit.Date                  : Date, format: "2010-08-30" "2009-11-15" ...
 $ Special.Pay                   : Factor w/ 4 levels "","CP","FR","SA": NA 2 NA NA NA NA NA NA 2 NA ...
 $ Tuition                       : int  424 2350 1181 376 865 2025 1977 3379 2200 1428 ...
 $ FRP.Active                    : int  25 9 17 0 40 9 16 10 30 51 ...
 $ FRP.Cancelled                 : int  3 9 6 0 8 4 4 0 0 1 ...
 $ FRP.Take.up.percent.          : num  0.424 0.409 0.708 0 0.494 0.9 0.64 0.769 0.577 0.773 ...
 $ Early.RPL                     : Date, format: "2010-03-29" "2009-10-20" ...
 $ Latest.RPL                    : Date, format: "2010-08-12" "2010-08-10" ...
 $ Cancelled.Pax                 : int  3 11 6 1 9 3 5 1 0 1 ...
 $ Total.Discount.Pax            : int  4 3 3 0 8 1 2 1 4 6 ...
 $ Initial.System.Date           : Date, format: "2010-03-26" "2009-10-02" ...
 $ Poverty.Code                  : Factor w/ 7 levels "","0","A","B",..: 4 5 5 1 6 5 1 1 1 1 ...
 $ Region                        : Factor w/ 6 levels "Dallas","Houston",..: 6 4 4 4 4 4 4 4 4 2 ...
 $ CRM.Segment                   : Factor w/ 11 levels "1","2","3","4",..: 4 10 10 7 10 8 8 7 5 5 ...
 $ School.Type                   : Factor w/ 4 levels "CHD","Catholic",..: 3 3 3 1 3 3 2 1 1 4 ...
 $ Parent.Meeting.Flag           : Factor w/ 2 levels "0","1": 2 2 2 1 2 2 2 2 2 2 ...
 $ MDR.Low.Grade                 : Factor w/ 13 levels "","1","10","2",..: 12 9 8 1 8 3 11 1 8 13 ...
 $ MDR.High.Grade                : Factor w/ 12 levels "1","2","3","4",..: 5 8 8 NA 8 12 12 NA 12 8 ...
 $ Total.School.Enrollment       : int  927 850 955 NA 720 939 225 NA 500 635 ...
 $ Income.Level                  : Factor w/ 23 levels "","A","B","C",..: 22 2 16 1 4 10 8 1 12 12 ...
 $ EZ.Pay.Take.Up.Rate           : num  0.17 0.091 0.042 0 0.383 0.1 0.08 0 0.231 0.136 ...
 $ School.Sponsor                : Factor w/ 2 levels "0","1": 2 1 1 1 1 1 1 1 1 1 ...
 $ SPR.Product.Type              : Factor w/ 6 levels "CA History","Costa Rica",..: 1 3 3 3 3 3 6 3 3 3 ...
 $ SPR.New.Existing              : Factor w/ 2 levels "EXISTING","NEW": 1 1 1 1 1 2 1 1 1 1 ...
 $ FPP                           : int  59 22 24 18 81 10 25 13 52 66 ...
 $ Total.Pax                     : int  63 25 27 18 89 11 27 14 56 72 ...
 $ SPR.Group.Revenue             : int  424 2350 1181 376 865 2025 1977 3379 2200 1428 ...
 $ NumberOfMeetingswithParents   : Factor w/ 3 levels "0","1","2": 2 3 2 1 2 2 2 2 2 2 ...
 $ FirstMeeting                  : Date, format: "2010-08-12" "2009-11-17" ...
 $ LastMeeting                   : Date, format: "2010-08-12" "2010-08-27" ...
 $ DifferenceTraveltoFirstMeeting: int  155 423 124 NA 145 91 63 138 143 146 ...
 $ DifferenceTraveltoLastMeeting : int  155 140 124 NA 145 91 63 138 143 146 ...
 $ SchoolGradeTypeLow            : Factor w/ 4 levels "Elementary","High",..: 1 3 3 2 3 2 2 2 3 3 ...
 $ SchoolGradeTypeHigh           : Factor w/ 4 levels "Elementary","High",..: 1 3 3 2 3 2 2 2 3 3 ...
 $ SchoolGradeType               : Factor w/ 9 levels "Elementary->Elementary",..: 1 7 7 5 7 5 5 5 7 7 ...
 $ DepartureMonth                : Factor w/ 6 levels "April","February",..: 3 3 3 3 3 3 3 3 3 2 ...
 $ GroupGradeTypeLow             : Factor w/ 6 levels "Elementary","High",..: 3 4 4 6 4 2 2 6 4 5 ...
 $ GroupGradeTypeHigh            : Factor w/ 4 levels "Elementary","High",..: 1 3 3 4 3 2 2 4 2 3 ...
 $ GroupGradeType                : Factor w/ 13 levels "Elementary->Elementary",..: 5 9 9 13 9 4 4 13 8 12 ...
 $ MajorProgramCode              : Factor w/ 4 levels "C","H","I","S": 2 2 2 2 2 2 4 3 1 2 ...
 $ SingleGradeTripFlag           : Factor w/ 2 levels "0","1": 2 2 2 1 1 1 1 2 2 2 ...
 $ FPP.to.School.enrollment      : num  0.0636 0.0259 0.0251 NA 0.1125 ...
 $ FPP.to.PAX                    : num  0.937 0.88 0.889 1 0.91 ...
 $ Num.of.Non_FPP.PAX            : int  4 3 3 0 8 1 2 1 4 6 ...
 $ SchoolSizeIndicator           : Factor w/ 5 levels "","L","M-L","S",..: 2 2 2 1 3 2 4 1 5 3 ...
 $ Retained.in.2012.             : Factor w/ 2 levels "0","1": 2 2 2 1 1 2 1 1 2 2 ...
```

Pretty good!!!

Data preprocessing

The data contains a number of columns with missing values. Let's investigate. The following tells us the number of missing values in each column.

In [353]:
```
sapply(STCdata_A, function(x) sum(is.na(x)))
```

Program.Code
0
From.Grade
127
To.Grade
150
Group.State
0
Is.Non.Annual.
0
Days
0
Travel.Type
0
Departure.Date
0
Return.Date

Dealing with missing values is a challenging problem, which could occupy a quarter of its own. The purpose of this homework is not to investigate in-depth approaches to dealing with missing values, but rather to investigate classification. For that reason, we take the following simple approach.

The function `fixNAs` below fixes missing values. The function defines reactions:

- adds a new category "FIXED_NA" for a missing value of a categorical/factor variable;
- fills zero value for a missing value of a numeric variable;
- fills "1900-01-01" for a missing value of a date variable.

Then it loops through all columns in the dataframe, reads their types, and loops through all the values, applying the defined reaction to any missing data point. In addition, the function creates a surrogate dummy variable for each column containing at least one missing value (for example, `Special.Pay_surrogate`), which takes a value of 1 whenever the original variable (`Special.Pay`) has a missing value, and 0 otherwise.

In [354]:
```
# Create a custom function to fix missing values ("NAs") and
# preserve the NA info as surrogate variables
fixNAs <- function(data_frame){
  # Define reactions to NAs
  integer_reac <- 0
  factor_reac <- "FIXED_NA"
  character_reac <- "FIXED_NA"
  date_reac <- as.Date("1900-01-01")

  # Loop through columns in the data frame
  # and depending on which class the
  # variable is, apply the defined reaction and
  # create a surrogate

  for (i in 1:ncol(data_frame)) {
    if (class(data_frame[,i]) %in% c("numeric","integer")) {
      if (any(is.na(data_frame[,i]))) {
        data_frame[,paste0(colnames(data_frame)[i],"_surrogate")] <-
          as.factor(ifelse(is.na(data_frame[,i]),"1","0"))
        data_frame[is.na(data_frame[,i]), i] <- integer_reac
      }
    } else
      if (class(data_frame[,i]) %in% c("factor")) {
        if (any(is.na(data_frame[,i]))){
          data_frame[,i]<-as.character(data_frame[,i])
          data_frame[,paste0(colnames(data_frame)[i],"_surrogate")] <-
            as.factor(ifelse(is.na(data_frame[,i]),"1","0"))
          data_frame[is.na(data_frame[,i]),i]<-factor_reac
          data_frame[,i]<-as.factor(data_frame[,i])
        }
      } else {
        if (class(data_frame[,i]) %in% c("character")) {
          if (any(is.na(data_frame[,i]))){
            data_frame[,paste0(colnames(data_frame)[i],"_surrogate")]<-
              as.factor(ifelse(is.na(data_frame[,i]),"1","0"))
            data_frame[is.na(data_frame[,i]),i]<-character_reac
          }
        } else {
          if (class(data_frame[,i]) %in% c("Date")) {
            if (any(is.na(data_frame[,i]))){
              data_frame[,paste0(colnames(data_frame)[i],"_surrogate")]<-
                as.factor(ifelse(is.na(data_frame[,i]),"1","0"))
              data_frame[is.na(data_frame[,i]),i]<-date_reac
            }
          }
        }
      }
  }

  return(data_frame)
}
```

We apply the above defined function to our data frame.

In [355]:
```
STCdata_A<-fixNAs(STCdata_A)
```

We can see that the columns do not have any missing values any more.

In [356]:
```
any( sapply(STCdata_A, function(x) sum(is.na(x))) > 0)
```

FALSE

Next, we combine the rare categories. Levels that do not occur often during training tend not to have reliable effect estimates and contribute to over-fit.

Let us check for rare categories in the variable `Group.State` .

In [357]:
```
table(STCdata_A$Group.State)
```

```
            AB             AK             AL             AR             AZ 
             1              5             21             10             53 
       Bermuda             CA             CO             CT Cayman Islands 
             1            718             89             15              1 
            FL             GA             HI             IA             ID 
            62             22              9             35             14 
            IL             IN             KS             KY             LA 
           104             43             26             16             31 
            MA             MD             ME             MI             MN 
            36             15              7             71             51 
            MO             MS             MT             MX             NC 
            43              9              6              3             16 
            ND             NE             NH             NJ             NM 
             5             42              7              6             20 
            NV             NY             OH             OK             OR 
            20             19             53             33             51 
            PA             PR             RI             SC             SD 
             5              1              3             10             11 
            TN             TX             UT             VA             VT 
            38            308              9             18              1 
            WA             WI             WV             WY 
           147             46              1              2 
```

Let us create a custom function to combine rare categories. The function again loops through all the columns in the dataframe, reads their types, and creates a table of counts for each level of the factor/categorical variables. All levels with counts less than the `mincount` are combined into "other." The function combines rare categories into "Other."+the name of the original variable (for example, `Other.State`). This function has two arguments:

- the name of the dataframe; and
- the count of observations in a category to define "rare."

In [358]:
```
combinerarecategories<-function(data_frame,mincount){
  for (i in 1:ncol(data_frame)) {
    a<-data_frame[,i]
    replace <- names(which(table(a) < mincount))
    levels(a)[levels(a) %in% replace] <-
      paste("Other", colnames(data_frame)[i], sep=".")
    data_frame[,i]<-a
  }
  return(data_frame)
}
```

Let us combine categories with < 10 values in `STCdata` into "Other." Ultimately, it is going to depend on the person doing the analysis on what they decide to call ``rare''.

In [359]:
```
STCdata_A<-combinerarecategories(STCdata_A,10)
```

Let us look at `Group.State` again.

In [360]:
```
table(STCdata_A$Group.State)
```

```
Other.Group.State                AL                AR                AZ 
               82                21                10                53 
               CA                CO                CT                FL 
              718                89                15                62 
               GA                IA                ID                IL 
               22                35                14               104 
               IN                KS                KY                LA 
               43                26                16                31 
               MA                MD                MI                MN 
               36                15                71                51 
               MO                NC                NE                NM 
               43                16                42                20 
               NV                NY                OH                OK 
               20                19                53                33 
               OR                SC                SD                TN 
               51                10                11                38 
               TX                VA                WA                WI 
              308                18               147                46 
```

You can investigate other columns to see if everything looks fine.

Split the data into training and testing sets

This is a very important step, both conceptually and technically. Conceptually, because the goal of predictive modeling is not to build a model that fits well the data it trains on, but rather one that would best predict the new data. A test set is in this sense the best representation of what the "new data" may look like. Technically, to facilitate comparison between different models, we need to maintain the same IDs in the corresponding sets at all times. We will accomplishes this through two "tricks":

- a random seed ensures that the random-number generator is initialized identically in each run; and
- the `inTrain` vector is created once and can then be applied anytime the data needs to be split.

By default, the code sets 500 data points in the test set, and the remainder 1,889 into the training set.

In [361]:
```
# set a random number generation seed to
# ensure that the split is the same every time
set.seed(233)

inTrain <- createDataPartition(
  y = STCdata_A$Retained.in.2012.,
  p = 1888/2389,
  list = FALSE)
df.train <- STCdata_A[ inTrain,]
df.test <- STCdata_A[ -inTrain, ]
```

Let us check that both the training and test sets have a similar proportion of positive and negative cases.

In [362]:
```
print('Training set proportion:')
table(df.train$Retained.in.2012.) / nrow(df.train)
print('Test set proportion:')
table(df.test$Retained.in.2012.) / nrow(df.test)
```

```
[1] "Training set proportion:"


        0         1
0.3928004 0.6071996

[1] "Test set proportion:"


    0     1
0.392 0.608
```

Fitting a logistic regression model

Let us fit a logistic regression model with all the variables included on the training set.

In [363]:
```
lgfit.all <- glm(Retained.in.2012.~ .,
                 data=df.train,
                 family="binomial")
summary(lgfit.all)
```

```
Warning message:
"glm.fit: fitted probabilities numerically 0 or 1 occurred"


Call:
glm(formula = Retained.in.2012. ~ ., family = "binomial", data = df.train)

Deviance Residuals:
    Min       1Q   Median       3Q      Max
-2.7206  -0.5092   0.2285   0.5545   3.1577

Coefficients: (44 not defined because of singularities)
                                                                 Estimate
(Intercept)                                                    -1.699e+02
Program.CodeCD                                                  7.495e-01
Program.CodeOther.Program.Code                                  4.205e-01
Program.CodeHC                                                  2.585e-01
Program.CodeHD                                                  2.816e-01
Program.CodeHG                                                 -8.370e-01
Program.CodeHN                                                  5.577e-01
```

The model is overfit. It has too many insignificant variables.

Let us fit a much simpler model. We will use stepwise regressions.

Recall stepwise regression from BUS 41100 Applied regression course. See, for example, [Week 9 slides (https://maxhfarrell.com/bus41100_old/)](https://maxhfarrell.com/bus41100_old/). You can also check Section 6.1.2 of the [ISLR (https://statlearning.com/)](https://statlearning.com/) book.

There are three approaches to running stepwise regressions: backward, forward and both. We need to specify criterion for inclusion/exclusion of variables. We will use one based on Bayesian information criteria.

Observe the process of variables being added to the model, (labeled by "+" in the output), gradual expansion of the model, and improvement of BIC.

In [364]:

```
# Start from a null model with intercept only, and add one covarite at a time until maximum BIC.
lgfit.null <- glm(Retained.in.2012.~ 1,
                  data=df.train, family="binomial")

lgfit.selected <- step(lgfit.null,                  # the starting model for our search
                       scope=formula(lgfit.all),    # the largest possible model that we will consider.
                       direction="forward",
                       k=log(nrow(df.train)),       # by default step() uses AIC, but by
                                                    # multiplying log(n) on the penalty, we get BIC.
                                                    # See ?step -> Arguments -> k
                       trace=1)
```

```
Start:  AIC=2538.74
Retained.in.2012. ~ 1

                                          Df Deviance    AIC
+ SingleGradeTripFlag                      1   2129.3 2144.4
+ Is.Non.Annual.                           1   2236.0 2251.1
+ From.Grade                              10   2196.2 2279.2
+ SPR.New.Existing                         1   2265.7 2280.8
+ Total.Pax                                1   2357.4 2372.5
+ FPP                                      1   2358.5 2373.6
+ FRP.Active                               1   2387.8 2402.8
+ Total.Discount.Pax                       1   2399.9 2415.0
+ Num.of.Non_FPP.PAX                       1   2399.9 2415.0
+ SchoolGradeTypeHigh                      3   2415.0 2445.2
+ SchoolGradeType                          7   2390.5 2450.8
+ To.Grade                                10   2396.0 2479.0
+ DepartureMonth                           5   2446.3 2491.6
+ SchoolGradeTypeLow                       3   2466.2 2496.3
+ CRM.Segment                             10   2416.2 2499.2
```

The algorithm stops once none of the 1-step expanded models lead to a lower BIC.

This is the selected model.

In [365]:

```
summary(lgfit.selected)
```

```
Call:
glm(formula = Retained.in.2012. ~ SingleGradeTripFlag + SPR.New.Existing +
    Is.Non.Annual. + FRP.Active + To.Grade_surrogate + Total.Discount.Pax,
    family = "binomial", data = df.train)

Deviance Residuals:
    Min       1Q   Median       3Q      Max
-2.8150  -0.7108   0.3982   0.6079   2.7149

Coefficients:
                      Estimate Std. Error z value Pr(>|z|)
(Intercept)           0.100495   0.141541   0.710 0.477699
SingleGradeTripFlag1  1.220935   0.130267   9.373  < 2e-16 ***
SPR.New.ExistingNEW  -1.597414   0.129210 -12.363  < 2e-16 ***
Is.Non.Annual.1      -2.427700   0.194144 -12.505  < 2e-16 ***
FRP.Active            0.023528   0.006669   3.528 0.000419 ***
To.Grade_surrogate1   0.738475   0.235902   3.130 0.001745 **
Total.Discount.Pax    0.108888   0.039687   2.744 0.006077 **
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

(Dispersion parameter for binomial family taken to be 1)

    Null deviance: 2531.2  on 1888  degrees of freedom
Residual deviance: 1723.6  on 1882  degrees of freedom
AIC: 1737.6

Number of Fisher Scoring iterations: 5
```

You can predict probabilities from this model using the following.

In [366]:

```
phat.lgfit.selected <- predict(lgfit.selected,
                               newdata = df.test,
                               type = "response")
```

You will use these probabilities later.

While we are investigating variable selection in logistic regression models, let us also use a more modern approach to variable selection. We will use the lasso.

If you have not seen this in BUS 41100 Applied regression course, do not worry. We will provide more details in the Week 5. You can also check Section 6.2.2 of the [ISLR (https://statlearning.com/)](https://statlearning.com/) book.

I provide the code to fit a lasso logistic regression model. We find coefficients β that minimize the deviance loss plus the penalty: [-2\cdot\sum_{i=1}^n \log p(y_i, x_i; \beta) + \lambda \sum_{j=1}^p |\beta_j|.] Here, λ is the user chosen penalty that controls the flexibility of the fit.

First, we need to create a model matrix that will be used as an input to the package.

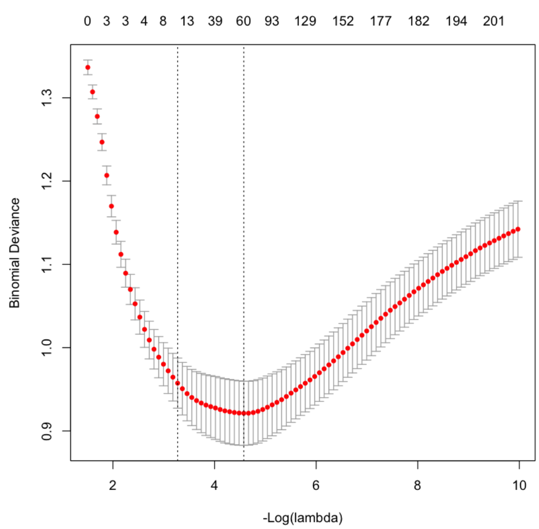
In [367]:
```
X <- model.matrix(formula(lgfit.all), STCdata_A)
#need to subtract the intercept
X <- X[,-1]

X.train = X[ inTrain, ]
X.test = X[ -inTrain, ]
```

Next, we run 5-fold cross-validation.

In [368]:
```
cv.l1.lgfit <- cv.glmnet(
  x       = X.train,
  y       = df.train$Retained.in.2012.,
  family  = "binomial",
  alpha   = 1,    #alpha=0 gives ridge regression
  nfolds  = 5)
```

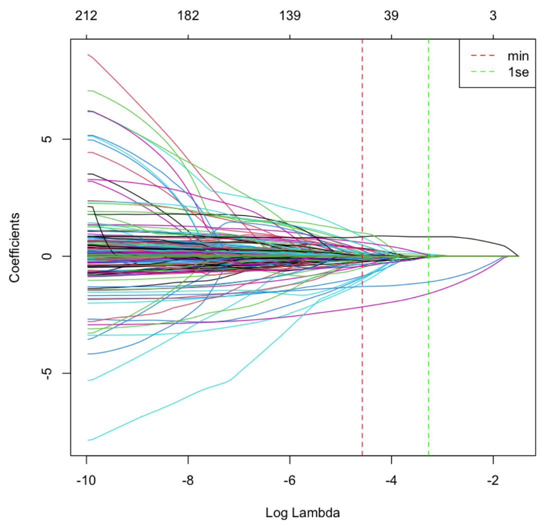
We can plot the cross-validation curve, which shows us an estimate of out-of-sample deviance as a function of the tuning parameter λ. The x-axis represents to $-\log(\lambda)$. Therefore, on the left we have large values of λ and on the right we have small values of λ. At the top, you can see the number variables that were selected into the model. The two vertical dashed lines correspond to λ values that minimize the cross-validation error and the largest value of lambda such that error is within 1 standard error of the minimum.

In [369]:
```
plot(cv.l1.lgfit, sign.lambda=-1)
```



Let us know plot the fitted coefficients as a function of λ. Note that `cv.l1.lgfit$glmnet.fit` corresponds to a fitted glmnet object for the full data.

In [370]:
```
glmnet.fit <- cv.l1.lgfit$glmnet.fit
plot(glmnet.fit, xvar = "lambda")
abline(v = log(cv.l1.lgfit$lambda.min), lty=2, col="red")
abline(v = log(cv.l1.lgfit$lambda.1se), lty=2, col="green")
legend("topright", legend=c("min", "1se"), lty=2, col=c("red", "green"))
```



For our predictive model, we will use 1 standard error λ. Below you can see the variables that are selected by the lasso.

In [371]:
```
betas <- coef(cv.l1.lgfit, s = "lambda.1se")
model.1se <- which(betas[2:length(betas)]!=0)
colnames(X[,model.1se])
```

'From.Grade8' 'Is.Non.Annual.1' 'FRP.Active' 'Total.Discount.Pax' 'CRM.Segment8' 'MDR.High.Grade8' 'Income.LevelP' 'SPR.New.ExistingNEW' 'Total.Pax' 'SchoolGradeTypeHighHigh' 'DepartureMonthJune' 'SingleGradeTripFlag1' 'SchoolSizeIndicatorS'

We now use our model to predict probabilities on the test set.

In [372]:
```
phat.l1.lgfit <- predict(glmnet.fit,
                         newx = X.test,
                         s = cv.l1.lgfit$lambda.1se,
                         type = "response")
```

Questions

How well does logistic regression do?

1. Create a confusion matrix for two logistic regression models build above. Use probabilities `phat.lgfit.selected` and `phat.l1.lgfit` to do so.

   To solve this question, you need to make a major decision. What should the cutoff or "threshold" for the probability be, above which you will label a customer as being classified as "retained?" In our case, the data is slightly unbalanced---about 60.72% of data points are in Class 1. For very unbalanced data, we would first need to balance it (over- or under-sample). In this case, the benefits of balancing are unclear, hence one can implement the average probability of being retained as a cutoff.

   Predict classification using 0.6072 threshold.

   What can we see from the confusion matrices?

In [373]:
```
threshold <- mean(phat.lgfit.selected)
```

In [374]:
```
get_confusion_matrix = function(y, phat, thr=0.5){
  yhat = as.factor(ifelse(phat > thr, 1, 0)) # 1 of greater than thr, 0 o.w.
  confusionMatrix(yhat, y)
}
```

In [375]:
```
get_confusion_matrix(df.test$Retained.in.2012., phat.lgfit.selected, threshold)
```

```
Confusion Matrix and Statistics

          Reference
Prediction   0   1
         0 147  66
         1  49 238

               Accuracy : 0.77
                 95% CI : (0.7306, 0.8062)
    No Information Rate : 0.608
    P-Value [Acc > NIR] : 1.062e-14

                  Kappa : 0.5248

 Mcnemar's Test P-Value : 0.1357

            Sensitivity : 0.7500
            Specificity : 0.7829
         Pos Pred Value : 0.6901
         Neg Pred Value : 0.8293
             Prevalence : 0.3920
         Detection Rate : 0.2940
   Detection Prevalence : 0.4260
      Balanced Accuracy : 0.7664

       'Positive' Class : 0
```

In [376]:
```
get_confusion_matrix(df.test$Retained.in.2012., phat.l1.lgfit, threshold)
```

```
Confusion Matrix and Statistics

          Reference
Prediction   0   1
         0 149  60
         1  47 244

               Accuracy : 0.786
                 95% CI : (0.7474, 0.8212)
    No Information Rate : 0.608
    P-Value [Acc > NIR] : <2e-16

                  Kappa : 0.5563

 Mcnemar's Test P-Value : 0.246

            Sensitivity : 0.7602
            Specificity : 0.8026
         Pos Pred Value : 0.7129
         Neg Pred Value : 0.8385
             Prevalence : 0.3920
         Detection Rate : 0.2980
   Detection Prevalence : 0.4180
      Balanced Accuracy : 0.7814

       'Positive' Class : 0
```

From the confusion matrices, we can see that the lasso model does a better job, though not by much. It is observed that the lasso model has both less false positives and false negatives, increasing the accuracy by .01% which we consider not to be a significant improvement.

2. Plot ROC curves for the two classifiers and report the area under the curve.

   Note that the AUC of an error-free classifier would be 100%, and an AUC of a random guess would be 50%. For values in-between, we can think of AUC as follows:

   - 90%+ = excellent,
   - 80–90% = very good,
   - 70–80% = good,
   - 60–70% = so-so, and
   - below 60% = not much value.

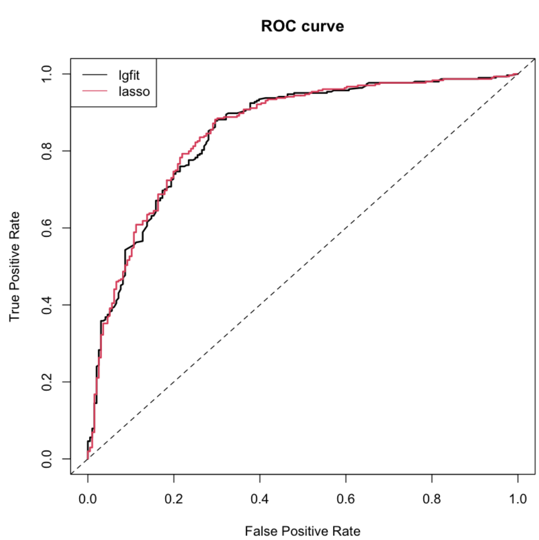
In [377]:
```
library(ROCR)
```

In [378]:
```
# Create a list with the 2 phat vectors
phat_list = list()
phat_list$lgfit =  matrix(phat.lgfit.selected,  ncol = 1)
phat_list$lasso =  matrix(phat.l1.lgfit,  ncol = 1)
nmethod <- length(phat_list)
```

In [379]:
```
#' @param y: should be 0/1
#' @param phat: probabilities obtained by our algorithm
#' @param wht: shrinks probabilities in phat towards .5
#' this helps avoid numerical problems --- don't use log(0)!
#' @return deviance loss
get_deviance = function(y,phat,wht=1e-7) {
  if(is.factor(y)) y = as.numeric(y)-1
  phat = (1-wht)*phat + wht*.5
  py = ifelse(y==1, phat, 1-phat)
  return(-2*sum(log(py)))
}
```

In [380]:
```
phat_best = matrix(0.0,nrow(df.test),nmethod) #pick off best from each method
colnames(phat_best) = names(phat_list)

for(i in 1:nmethod) {
  nrun = ncol(phat_list[[i]])
  lvec = rep(0,nrun)
  for(j in 1:nrun) lvec[j] = get_deviance(df.test$Retained.in.2012.,phat_list[[i]][,j])
    imin = which.min(lvec)
  phat_best[,i] = phat_list[[i]][,imin]
}
```

In [381]:
```
for(i in 1:ncol(phat_best)) {
  pred = prediction(phat_best[,i], df.test$Retained.in.2012.)
  perf = performance(pred, measure = "tpr", x.measure = "fpr")

  if (i == 1) {
    plot(perf, col=1, lwd=2,
    main= 'ROC curve',
    xlab='False Positive Rate',
    ylab='True Positive Rate')
  }

  else {
    plot(perf, add=T, col=i, lwd=2)
  }
}
abline(0, 1, lty=2)
legend("topleft",legend=names(phat_list),col=1:nmethod,lty=rep(1,nmethod))
```



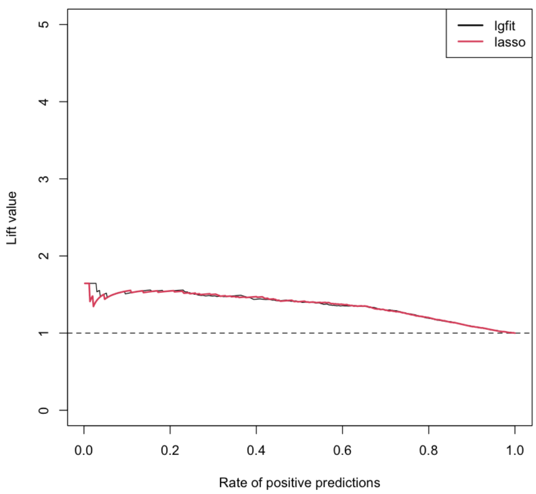
In [382]:
```
for(i in 1:ncol(phat_best)) {
    pred = prediction(phat_best[,i], df.test$Retained.in.2012.)
    perf = performance(pred, measure = "auc")
    print(paste0("AUC ", names(phat_list)[i], " :: ", perf@y.values[[1]]))
}
```

```
[1] "AUC lgfit :: 0.849321965628357"
[1] "AUC lasso :: 0.85170515574651"
```

The AUC (Area Under the Curve) values of 0.849 and 0.852 represent the performance of two different models, "lgfit" and "lasso", in a binary classification task. An AUC of 0.8 is considered a good model performance, and an AUC value close to 1 indicates a perfect classifier. The higher the AUC value, the better the model is at distinguishing between the positive and negative class. In this case, the "lasso" model has a slightly better performance with an AUC of 0.8517 compared to the "lgfit" model with an AUC of 0.8493.

3. Plot lift curves for the two classifiers.

In [383]:
```
pred = prediction(phat_best[,1], df.test$Retained.in.2012.)
perf = performance(pred, measure="lift", x.measure="rpp", lwd=2)
plot(perf, col=1, ylim=c(0,5))
abline(h=1, lty=2)

for(i in 2:ncol(phat_best)) {
  pred = prediction(phat_best[,i], df.test$Retained.in.2012.)
  perf = performance(pred, measure="lift", x.measure="rpp")
  plot(perf, add=T, col=i, lwd=2)
}
legend("topright", legend=names(phat_list),col=1:nmethod, lty=rep(1,nmethod), lwd=2)
```



We can observe from the lift curves that they are very similar.

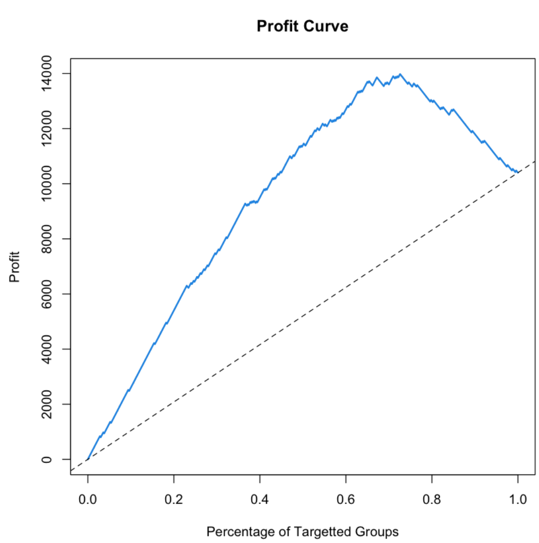
4. Create the profit curve (the amount of net profit vs the number of groups targeted for promotion) for the two classifiers. Suppose that the benefit of retaining a group is $100, whilethecostofapromotionis$40.

   How many groups should be targeted to maximize the profit?

   How would this number change as the ratio between the benefit and cost changes?

In [384]:
```
# Function to plot a profit curve
#
# Inputs:
#   - benefitTP(FN/FP/TN): the net benefit for a true positive (false negative,...)
#       which is positive for a gain, and negative for a loss
#   - y: vector of true labels, which has to be labeled as "0" and "1"
#   - phat: vector of predicted probabilities
# Outputs:
 #    the function returns the profit curve

ProfitCurve <- function(benefitTP, benefitFN, benefitFP, benefitTN, y, phat){
    if(length(y) != length(phat)) stop("Length of y and phat not identical")
    if(length(levels(y))!=2 | levels(y)[1]!="0" | levels(y)[2]!="1") stop("y should be a vector of factors, only with 
    n <- length(y)
    df <- data.frame(y, phat)
# Order phat so that we can pick the k highest groups for promotion
    df <- df[order(df[,2], decreasing = T),]
    TP <- 0; FP <- 0; FN <- table(y)[2]; TN <- table(y)[1]
# Initializing the x and y coordinates of the plot
    ratio.vec <- seq(0,n)/n
    profit.vec <- rep(0,n+1)
    profit.vec[1] <- FN * benefitFN + TN * benefitTN
    for(k in 1:n){
        # k is the number of groups classified as "YES"
        # In every round, we are picking one more group for promotion.
        # If this group was ratained (positive), then in this round, it is classified
        # as a "YES" instead of "NO" before. The confusion matrix is updated each round
        # with one more TP, and one less FN. It's similar when the group was not ratained.
        if(df[k,1]=="1"){TP <- TP + 1; FN <- FN - 1}
        else{FP <- FP + 1; TN <- TN - 1}
        # print(paste(TP, FP, TP-FP, benefitTP, benefitFP))
        profit.vec[k+1] <- TP*benefitTP + FP*benefitFP + FN*benefitFN + TN*benefitTN
        }

    # Get a matrix with profit and ratio
    profit.mat <- cbind(ratio.vec, profit.vec)

    plt <- plot(ratio.vec, profit.vec, type="l", lwd=2, col=4, main="Profit Curve",
            xlab="Percentage of Targetted Groups", ylab="Profit")
    abline(b=(profit.vec[n+1]-profit.vec[1]), a=profit.vec[1], lty=2) #Random guess
    return(profit.mat )
    }
```

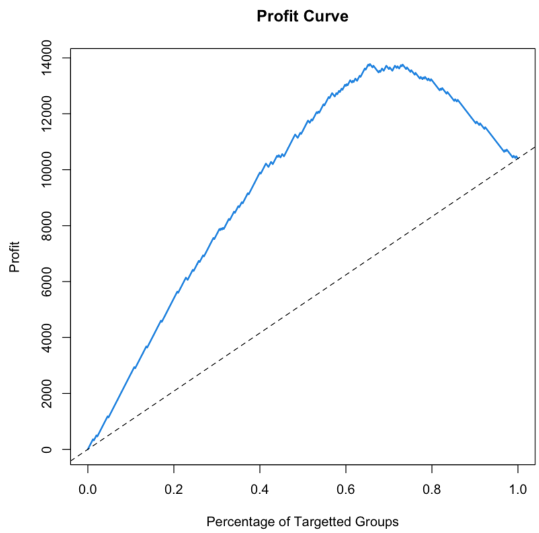
In [385]:
```
curve_1 <- ProfitCurve(60,0,-40,0,df.test$Retained.in.2012.,phat_best[,1])
```



In [386]:
```
# Get the maximum profit and the corresponding ratio with the correspinding row number
max_profit <- max(curve_1[,2])
max_ratio <- curve_1[which.max(curve_1[,2]),1]
max_row <- which.max(curve_1[,2])
print(paste("Maximum profit is", max_profit, "with ratio", max_ratio, "and the number of groups", max_row))
```

```
[1] "Maximum profit is 13980 with ratio 0.726 and the number of groups 364"
```

In [387]:
```
curve_2 <- ProfitCurve(60,0,-40,0,df.test$Retained.in.2012.,phat_best[,2])
```



In [388]:
```
# Get the maximum profit and the corresponding ratio with the correspinding row number
max_profit <- max(curve_2[,2])
max_ratio <- curve_2[which.max(curve_2[,2]),1]
max_row <- which.max(curve_2[,2])
print(paste("Maximum profit is", max_profit, "with ratio", max_ratio, "and the number of groups", max_row))
```

```
[1] "Maximum profit is 13780 with ratio 0.656 and the number of groups 329"
```

Let's suppose that the cost increases, we would expect the ratio of targeted groups to reduce as well. This is because the cost of the promotion is now higher than the benefit of retaining a group. The profit curve is a function of the ratio between the benefit and cost, and as the ratio increases, the number of groups targeted for promotion increases.

5. Develop a decision tree, random forest, and a boosting model using the training data.

   Report ROC, AUC, lift, and profit curves for these models.

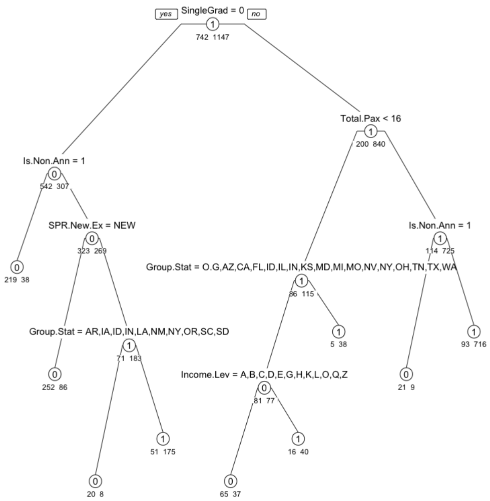
   How do these methods compare to the logistic regression models?

In [389]:
```
library (ranger)
library(rpart)
library(rpart.plot)
```

Decision Tree

In [390]:
```
default.ct <- rpart(Retained.in.2012. ~ ., data = df.train, method = "class")
```

In [391]:
```
prp(default.ct, type = 1, extra = 1, under = TRUE, split.font = 1, varlen = -10)
```



In [392]:
```
deeper.ct <- rpart(Retained.in.2012. ~ ., data = df.train, method = "class", cp = 0, minsplit = 1)
```

In [393]:
```
length(deeper.ct$frame$var[deeper.ct$frame$var == "<leaf>"])
```

229

In [394]:
```
default.ct.point.pred.train <- predict(default.ct, df.train, type = "class")
deeper.ct.point.pred.train <- predict(deeper.ct, df.train, type = "class")
cm.default.train <- confusionMatrix(default.ct.point.pred.train, df.train$Retained.in.2012.)
cm.deeper.train <- confusionMatrix(deeper.ct.point.pred.train, df.train$Retained.in.2012.)
print(cm.default.train)
print(cm.deeper.train)
```

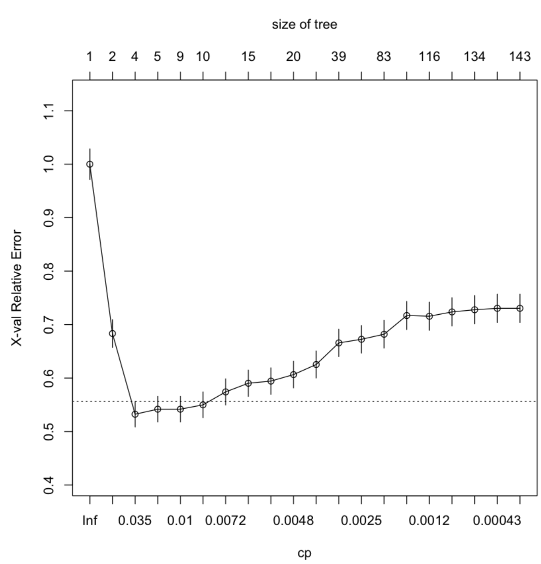
```
Confusion Matrix and Statistics

          Reference
Prediction   0   1
         0 577 178
         1 165 969

               Accuracy : 0.8184
                 95% CI : (0.8003, 0.8356)
    No Information Rate : 0.6072
    P-Value [Acc > NIR] : <2e-16

                  Kappa : 0.6205

 Mcnemar's Test P-Value : 0.517

            Sensitivity : 0.7776
            Specificity : 0.8448
         Pos Pred Value : 0.7642
         Neg Pred Value : 0.8545
             Prevalence : 0.3928
         Detection Rate : 0.3055
   Detection Prevalence : 0.3997
      Balanced Accuracy : 0.8112

       'Positive' Class : 0

Confusion Matrix and Statistics

          Reference
Prediction    0    1
         0  742    0
         1    0 1147

               Accuracy : 1
                 95% CI : (0.998, 1)
    No Information Rate : 0.6072
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 1

 Mcnemar's Test P-Value : NA

            Sensitivity : 1.0000
            Specificity : 1.0000
         Pos Pred Value : 1.0000
         Neg Pred Value : 1.0000
             Prevalence : 0.3928
         Detection Rate : 0.3928
   Detection Prevalence : 0.3928
      Balanced Accuracy : 1.0000

       'Positive' Class : 0
```

In [395]:
```
default.ct.point.pred.valid <- predict(default.ct, df.test, type = "class")
deeper.ct.point.pred.valid <- predict(deeper.ct, df.test, type = "class")
cm.default.valid <- confusionMatrix(default.ct.point.pred.valid, df.test$Retained.in.2012.)
cm.deeper.valid <- confusionMatrix(deeper.ct.point.pred.valid, df.test$Retained.in.2012.)
print(cm.default.valid)
print(cm.deeper.valid)
```

```
Confusion Matrix and Statistics

          Reference
Prediction   0   1
         0 140  56
         1  56 248

               Accuracy : 0.776
                 95% CI : (0.7369, 0.8118)
    No Information Rate : 0.608
    P-Value [Acc > NIR] : 9.848e-16

                  Kappa : 0.5301

 Mcnemar's Test P-Value : 1

            Sensitivity : 0.7143
            Specificity : 0.8158
         Pos Pred Value : 0.7143
         Neg Pred Value : 0.8158
             Prevalence : 0.3920
         Detection Rate : 0.2800
   Detection Prevalence : 0.3920
      Balanced Accuracy : 0.7650

       'Positive' Class : 0

Confusion Matrix and Statistics

          Reference
Prediction   0   1
         0 128  81
         1  68 223

               Accuracy : 0.702
                 95% CI : (0.6598, 0.7418)
    No Information Rate : 0.608
    P-Value [Acc > NIR] : 7.392e-06

                  Kappa : 0.3821

 Mcnemar's Test P-Value : 0.3256

            Sensitivity : 0.6531
            Specificity : 0.7336
         Pos Pred Value : 0.6124
         Neg Pred Value : 0.7663
             Prevalence : 0.3920
         Detection Rate : 0.2560
   Detection Prevalence : 0.4180
      Balanced Accuracy : 0.6933

       'Positive' Class : 0
```

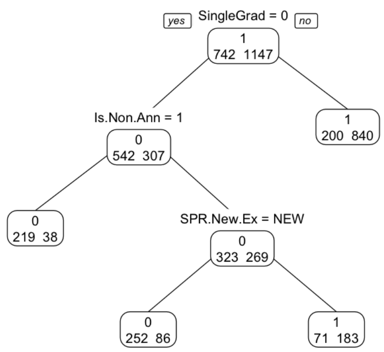
In [396]:
```
cv.ct <- rpart(Retained.in.2012. ~ ., data = df.train, method = "class",
            cp = 0.00001, minsplit = 5, xval = 5)
printcp(cv.ct)
plotcp(cv.ct)
```

```
Classification tree:
rpart(formula = Retained.in.2012. ~ ., data = df.train, method = "class",
    cp = 1e-05, minsplit = 5, xval = 5)

Variables actually used in tree construction:
 [1] CRM.Segment                    Days
 [3] Departure.Date                 Deposit.Date
 [5] DifferenceTraveltoFirstMeeting DifferenceTraveltoLastMeeting
 [7] EZ.Pay.Take.Up.Rate            Early.RPL
 [9] FPP                            FPP.to.PAX
[11] FPP.to.School.enrollment       FRP.Active
[13] FRP.Cancelled                  FRP.Take.up.percent.
[15] FirstMeeting                   From.Grade
[17] Group.State                    GroupGradeType
[19] Income.Level                   Initial.System.Date
[21] Is.Non.Annual.                 LastMeeting
[23] Latest.RPL                     MDR.High.Grade
[25] MDR.Low.Grade                  Poverty.Code
[27] Program.Code                   Region
[29] SPR.New.Existing               School.Sponsor
[31] School.Type                    SchoolGradeType
[33] SchoolSizeIndicator            SingleGradeTripFlag
[35] Special.Pay                    To.Grade
[37] Total.Pax                      Total.School.Enrollment
[39] Tuition

Root node error: 742/1889 = 0.3928

n= 1889

           CP nsplit rel error  xerror     xstd
1  0.31671159      0  1.000000 1.00000 0.028606
2  0.07547170      1  0.683288 0.68329 0.025956
3  0.01617251      3  0.532345 0.53235 0.023821
4  0.01347709      4  0.516173 0.54178 0.023974
5  0.00808625      8  0.462264 0.54178 0.023974
6  0.00763702      9  0.454178 0.54987 0.024104
7  0.00673854     13  0.420485 0.57412 0.024480
8  0.00539084     14  0.413747 0.59030 0.024720
9  0.00494160     16  0.402965 0.59434 0.024779
10 0.00471698     19  0.388140 0.60647 0.024953
11 0.00404313     21  0.378706 0.62534 0.025214
12 0.00269542     38  0.308625 0.66577 0.025741
13 0.00224618     71  0.207547 0.67251 0.025825
14 0.00202156     82  0.176550 0.68194 0.025940
15 0.00134771     92  0.150943 0.71698 0.026347
16 0.00112309    115  0.119946 0.71563 0.026332
17 0.00089847    121  0.113208 0.72372 0.026421
18 0.00067385    133  0.102426 0.72776 0.026466
19 0.00026954    137  0.099730 0.73046 0.026495
20 0.00001000    142  0.098383 0.73046 0.026495
```



In [397]:
```
pruned.ct <- prune(cv.ct, cp = cv.ct$cptable[which.min(cv.ct$cptable[,"xerror"]),"CP"])
length(pruned.ct$frame$var[pruned.ct$frame$var == "<leaf>"])
prp(pruned.ct, type = 1, extra = 1, split.font = 1, varlen = -10)
```

4



In [398]:
```
# this is the cp parameter with smallest cv-errror
index_cp_min = which.min(cv.ct$cptable[,"xerror"])

# one standard deviation rule
# need to find first cp value for which the xerror is below horizontal line on the plot
(val_h = cv.ct$cptable[index_cp_min, "xerror"] + cv.ct$cptable[index_cp_min, "xstd"])
(index_cp_std = Position(function(x) x < val_h, cv.ct$cptable[, "xerror"]))
(cp_std = cv.ct$cptable[ index_cp_std, "CP" ])
```

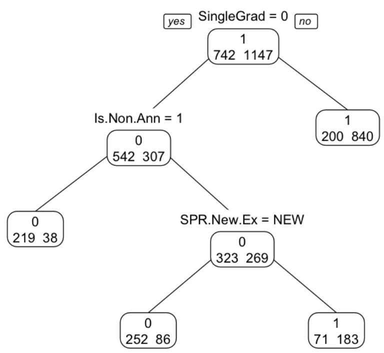
0.556165682910288

3

0.0161725067385445

In [399]:
```
pruned.ct <- prune(cv.ct, cp = cp_std)
length(pruned.ct$frame$var[pruned.ct$frame$var == "<leaf>"])
prp(pruned.ct, type = 1, extra = 1, split.font = 1, varlen = -10)
```

4



In [400]:
```
phat.tree <- predict(pruned.ct, df.test, type = "prob")
# Drop the first column, which is the probability of "NO"
phat.tree <- phat.tree[,2]
phat.tree <- data.frame(phat.tree)
# Add to phat_list the phat tree as a matrix 500 x 1
phat_list <- cbind(phat_list, phat.tree)
```

Random Forest

In [401]:
```
# Run a random forest model
p = ncol(df.train) - 1

grid_rf = expand.grid(
  mtry = c(p, ceiling(sqrt(p))),
  node_size = c(5, 10, 20)
)

for (i in 1:nrow(grid_rf)) {
  rf = ranger(Retained.in.2012. ~ ., data = df.train, mtry = grid_rf$mtry[i],
              min.node.size = grid_rf$node_size[i], probability = TRUE)
  phat.rf <- predict(rf, df.test)$predictions
  phat.rf <- data.frame(phat.rf)
}

# Select the best model
phat.rf <- predict(rf, df.test)$predictions
phat.rf <- data.frame(phat.rf)
# Add to phat_list the phat rf as a matrix 500 x 1 only X1
phat_list <- cbind(phat_list, phat.rf[,1])
```

In [288…

```
import pandas as pd
import numpy as np
# import statsmodels.api as sm
# import statsmodels.formula.api as smf
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, confusion_matrix
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassifier
from sklearn.metrics import roc_curve, auc, precision_recall_curve, confusion_matrix
import matplotlib.pyplot as plt
```

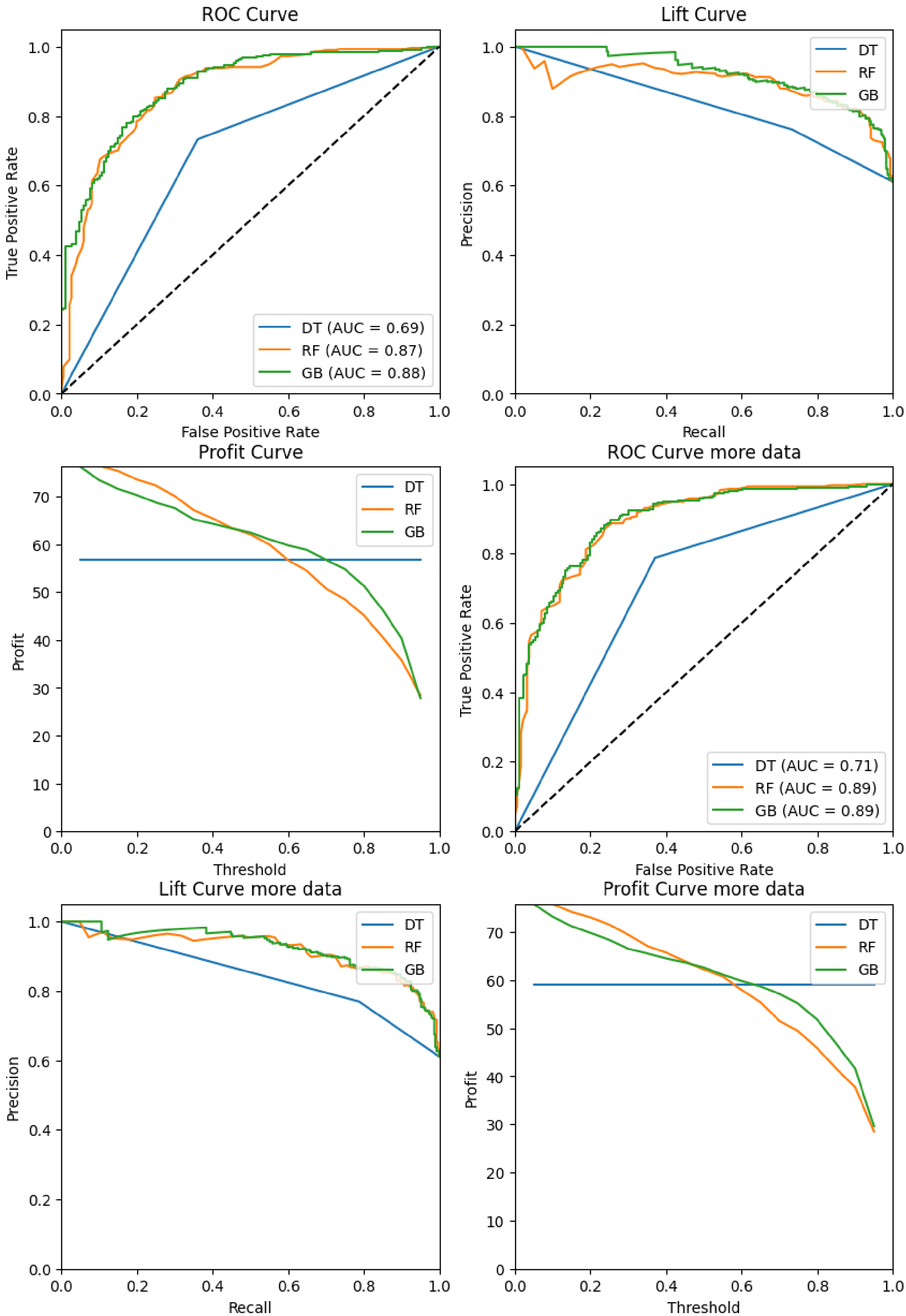
In [340…

```
fig, ax = plt.subplots(nrows=3, ncols=2, figsize=(10, 15))

df = pd.read_csv("~/Downloads/STCdata_A.csv",index_col=0)

le = LabelEncoder()
df = df.apply(le.fit_transform)

X = df.drop("Retained.in.2012.", axis=1)
y = df["Retained.in.2012."]
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=0)

dt = DecisionTreeClassifier(random_state=0)
dt.fit(X_train, y_train)

rf = RandomForestClassifier(random_state=0)
rf.fit(X_train, y_train)

gb = GradientBoostingClassifier(random_state=0)
gb.fit(X_train, y_train)

y_pred_dt = dt.predict_proba(X_test)
y_pred_rf = rf.predict_proba(X_test)
y_pred_gb = gb.predict_proba(X_test)

fpr_dt, tpr_dt, _ = roc_curve(y_test, y_pred_dt[:,1])
roc_auc_dt = auc(fpr_dt, tpr_dt)

fpr_rf, tpr_rf, _ = roc_curve(y_test, y_pred_rf[:,1])
roc_auc_rf = auc(fpr_rf, tpr_rf)

fpr_gb, tpr_gb, _ = roc_curve(y_test, y_pred_gb[:,1])
roc_auc_gb = auc(fpr_gb, tpr_gb)

plt.subplot(3, 2, 1)
plt.plot(fpr_dt, tpr_dt, label='DT (AUC = %0.2f)' % roc_auc_dt)
plt.plot(fpr_rf, tpr_rf, label='RF (AUC = %0.2f)' % roc_auc_rf)
plt.plot(fpr_gb, tpr_gb, label='GB (AUC = %0.2f)' % roc_auc_gb)
plt.plot([0, 1], [0, 1], 'k--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve')
plt.legend(loc="lower right")

precision, recall, thresholds = precision_recall_curve(y_test, y_pred_dt[:,1])
plt.subplot(3, 2, 2)
```

```
plt.plot(recall, precision, label='DT')

precision, recall, thresholds = precision_recall_curve(y_test, y_pred_rf[:,1])
plt.plot(recall, precision, label='RF')

precision, recall, thresholds = precision_recall_curve(y_test, y_pred_gb[:,1])
plt.plot(recall, precision, label='GB')

plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('Recall')
plt.ylabel('Precision')
plt.title('Lift Curve')
plt.legend(loc="upper right")

cost = 40
revenue = 100
thresholds = [0.05*i for i in range(1,20)]
profits = []
for threshold in thresholds:
    y_pred_dt_bin = y_pred_dt[:,1] >= threshold
    cm = confusion_matrix(y_test, y_pred_dt_bin)
    total_cost = cost*(cm[0,1] + cm[1,0]) + revenue*cm[1,1]
    profits.append(total_cost/len(y_test))

plt.subplot(3, 2, 3)
plt.plot(thresholds, profits, label='DT')

thresholds = [0.05*i for i in range(1,20)]
profits = []
for threshold in thresholds:
    y_pred_rf_bin = y_pred_rf[:,1] >= threshold
    cm = confusion_matrix(y_test, y_pred_rf_bin)
    total_cost = cost*(cm[0,1] + cm[1,0]) + revenue*cm[1,1]
    profits.append(total_cost/len(y_test))

plt.plot(thresholds, profits, label='RF')

thresholds = [0.05*i for i in range(1,20)]
profits = []
for threshold in thresholds:
    y_pred_gb_bin = y_pred_gb[:,1] >= threshold
    cm = confusion_matrix(y_test, y_pred_gb_bin)
    total_cost = cost*(cm[0,1] + cm[1,0]) + revenue*cm[1,1]
    profits.append(total_cost/len(y_test))

plt.plot(thresholds, profits, label='GB')

plt.xlim([0.0, 1.0])
plt.ylim([0.0, max(profits)])
plt.xlabel('Threshold')
plt.ylabel('Profit')
plt.title('Profit Curve')
plt.legend(loc="upper right")

# again with extra data
df_m = pd.read_csv("~/Downloads/STCdata_merged.csv",index_col=0)

le = LabelEncoder()
df_m = df_m.apply(le.fit_transform)

X = df_m.drop("Retained.in.2012.", axis=1)
y = df_m["Retained.in.2012."]
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=0)
```

```
dt = DecisionTreeClassifier(random_state=0)
dt.fit(X_train, y_train)

rf = RandomForestClassifier(random_state=0)
rf.fit(X_train, y_train)

gb = GradientBoostingClassifier(random_state=0)
gb.fit(X_train, y_train)

y_pred_dt = dt.predict_proba(X_test)
y_pred_rf = rf.predict_proba(X_test)
y_pred_gb = gb.predict_proba(X_test)

fpr_dt, tpr_dt, _ = roc_curve(y_test, y_pred_dt[:,1])
roc_auc_dt = auc(fpr_dt, tpr_dt)

fpr_rf, tpr_rf, _ = roc_curve(y_test, y_pred_rf[:,1])
roc_auc_rf = auc(fpr_rf, tpr_rf)

fpr_gb, tpr_gb, _ = roc_curve(y_test, y_pred_gb[:,1])
roc_auc_gb = auc(fpr_gb, tpr_gb)

plt.subplot(3, 2, 4)
plt.plot(fpr_dt, tpr_dt, label='DT (AUC = %0.2f)' % roc_auc_dt)
plt.plot(fpr_rf, tpr_rf, label='RF (AUC = %0.2f)' % roc_auc_rf)
plt.plot(fpr_gb, tpr_gb, label='GB (AUC = %0.2f)' % roc_auc_gb)
plt.plot([0, 1], [0, 1], 'k--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve more data')
plt.legend(loc="lower right")

precision, recall, thresholds = precision_recall_curve(y_test, y_pred_dt[:,1])
plt.subplot(3, 2, 5)
plt.plot(recall, precision, label='DT')

precision, recall, thresholds = precision_recall_curve(y_test, y_pred_rf[:,1])
plt.plot(recall, precision, label='RF')

precision, recall, thresholds = precision_recall_curve(y_test, y_pred_gb[:,1])
plt.plot(recall, precision, label='GB')

plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('Recall')
plt.ylabel('Precision')
plt.title('Lift Curve more data')
plt.legend(loc="upper right")

cost = 40
revenue = 100
thresholds = [0.05*i for i in range(1,20)]
profits = []
for threshold in thresholds:
    y_pred_dt_bin = y_pred_dt[:,1] >= threshold
    cm = confusion_matrix(y_test, y_pred_dt_bin)
    total_cost = cost*(cm[0,1] + cm[1,0]) + revenue*cm[1,1]
    profits.append(total_cost/len(y_test))
plt.subplot(3, 2, 6)
plt.plot(thresholds, profits, label='DT')
```

```
thresholds = [0.05*i for i in range(1,20)]
profits = []
for threshold in thresholds:
    y_pred_rf_bin = y_pred_rf[:,1] >= threshold
    cm = confusion_matrix(y_test, y_pred_rf_bin)
    total_cost = cost*(cm[0,1] + cm[1,0]) + revenue*cm[1,1]
    profits.append(total_cost/len(y_test))

plt.plot(thresholds, profits, label='RF')

thresholds = [0.05*i for i in range(1,20)]
profits = []
for threshold in thresholds:
    y_pred_gb_bin = y_pred_gb[:,1] >= threshold
    cm = confusion_matrix(y_test, y_pred_gb_bin)
    total_cost = cost*(cm[0,1] + cm[1,0]) + revenue*cm[1,1]
    profits.append(total_cost/len(y_test))

plt.plot(thresholds, profits, label='GB')

plt.xlim([0.0, 1.0])
plt.ylim([0.0, max(profits)])
plt.xlabel('Threshold')
plt.ylabel('Profit')
plt.title('Profit Curve more data')
plt.legend(loc="upper right")
plt.show()
```



How do these methods compare to the logistic regression models?

The logistic model is underperforming these models (gradient boost and random forest). We can see this especially from the ROC curve. These models, especially gradient boosting and random forest, are doing better as seen by their AUC (0.87 for RF and 0.88 for GB, while 0.85 for logistic regression) where we know that a higher AUC is better.

Comment on the improvement (or lack thereof) from incorporating the NPS data

Including the data from Emily was a net improvement to our performance. For example, AUC went from 0.69,0.87,0.88 to 0.71,0.89,0.89 for decision tree, random forest and gradient boost respectively. We can see some to no improvement in lift curve and profit curve. In sum, while the improvement from getting more data is not large, it is still an improvement.

In [ ]:

```

```