Project 2

Kyle Bartsch

November 5, 2015

Summary

This dataset was pulled from the UC Irvine Machine Learning Repository: http://archive.ics.uci.edu/ml/index.html (http://archive.ics.uci.edu/ml/index.html). It is compiled from data related to direct marketing campaigns of a Portuguese banking institution. The marketing campaigns were based on phone calls….The classification goal is to predict if the client will subscribe to a term deposit.

Data Import and Coding

Dataset Attributes:

The data consists of *45211* observations of *21* variables. Half of the data was withheld as a test dataset for model validation.

<u>Input variables:</u>
bank client data:
1 - age (numeric)
2 - job : type of job (categorical: 'admin.','blue-collar','entrepreneur','housemaid','management','retired','self-employed','services','student','technician','unemployed','unknown')
3 - marital : marital status (categorical: 'divorced','married','single','unknown'; note: 'divorced' means divorced or widowed)
4 - education (categorical: 'basic.4y','basic.6y','basic.9y','high.school','illiterate','professional.course','university.degree','unknown')
5 - default: has credit in default? (categorical: 'no','yes','unknown')
6 - housing: has housing loan? (categorical: 'no','yes','unknown')
7 - loan: has personal loan? (categorical: 'no','yes','unknown')
related with the last contact of the current campaign:
8 - contact: contact communication type (categorical: 'cellular','telephone')
9 - month: last contact month of year (categorical: 'jan', 'feb', 'mar', …, 'nov', 'dec')
10 - day_of_week: last contact day of the week (categorical: 'mon','tue','wed','thu','fri')
11 - duration: last contact duration, in seconds (numeric). Important note: this attribute highly affects the output target (e.g., if duration=0 then y='no'). Yet, the duration is not known before a call is performed. Also, after the end of the call y is obviously known. Thus, this input should only be included for benchmark purposes and should be discarded if the intention is to have a realistic predictive model.
other attributes:
12 - campaign: number of contacts performed during this campaign and for this client (numeric, includes last contact)

13 - pdays: number of days that passed by after the client was last contacted from a previous campaign (numeric; 999 means client was not previously contacted)
14 - previous: number of contacts performed before this campaign and for this client (numeric)
15 - poutcome: outcome of the previous marketing campaign (categorical: 'failure','nonexistent','success')
social and economic context attributes
16 - emp.var.rate: employment variation rate - quarterly indicator (numeric)
17 - cons.price.idx: consumer price index - monthly indicator (numeric)
18 - cons.conf.idx: consumer confidence index - monthly indicator (numeric)
19 - euribor3m: euribor 3 month rate - daily indicator (numeric)
20 - nr.employed: number of employees - quarterly indicator (numeric)

Output variable (desired target):
21 - y - has the client subscribed a term deposit? (binary: 'yes','no')

The online directions state that the "Duration" variable highly affects the output target and should be only used as a benchmark; it should removed if the intention is to develop a truly predictive model:

```
d = d[,-which(names(d) %in% c("X","Duration"))] #remove index and duration
summary(d)
```

```
##       Age                Job              Marital
##  Min.   :17.0   admin.     :5189   divorced: 2285
##  1st Qu.:32.0   blue-collar:4592   married :12500
##  Median :38.0   technician :3417   single  : 5767
##  Mean   :40.1   services   :1967   unknown :   41
##  3rd Qu.:47.0   management :1490
##  Max.   :98.0   retired    : 874
##                 (Other)    :3064
##                Education        Default          Housing
##  university.degree  :6091   no     :16317   no     : 9346
##  high.school        :4777   unknown: 4275   unknown:  497
##  basic.9y           :3001   yes    :    1   yes    :10750
##  professional.course:2660
##  basic.4y           :2073
##  basic.6y           :1117
##  (Other)            : 874
##       Loan            Contact            Month         Day
##  no     :17011   cellular :13157   may    :6907   fri:3905
##  unknown:  497   telephone: 7436   jul    :3590   mon:4178
##  yes    : 3085                     aug    :3148   thu:4317
##                                    jun    :2600   tue:4102
##                                    nov    :2046   wed:4091
##                                    apr    :1306
##                                    (Other): 996
##     Campaign          pDays           Previous            pOutcome
##  Min.   : 1.000   Min.   :  0.0   Min.   :0.000   failure    : 2090
##  1st Qu.: 1.000   1st Qu.:999.0   1st Qu.:0.000   nonexistent:17822
##  Median : 2.000   Median :999.0   Median :0.000   success    :  681
##  Mean   : 2.563   Mean   :962.7   Mean   :0.173
##  3rd Qu.: 3.000   3rd Qu.:999.0   3rd Qu.:0.000
##  Max.   :43.000   Max.   :999.0   Max.   :7.000
##
##    EmpVarRate         ConsPriceIdx    ConsConfIndx      Euribor3M
##  Min.   :-3.40000   Min.   :92.20   Min.   :-50.80   Min.   :0.634
##  1st Qu.:-1.80000   1st Qu.:93.08   1st Qu.:-42.70   1st Qu.:1.344
##  Median : 1.10000   Median :93.75   Median :-41.80   Median :4.857
##  Mean   : 0.08379   Mean   :93.57   Mean   :-40.47   Mean   :3.623
##  3rd Qu.: 1.40000   3rd Qu.:93.99   3rd Qu.:-36.40   3rd Qu.:4.961
##  Max.   : 1.40000   Max.   :94.77   Max.   :-26.90   Max.   :5.045
##
##    NREmployed   Subscribed
##  Min.   :4964   no :18228
##  1st Qu.:5099   yes: 2365
##  Median :5191
##  Mean   :5167
##  3rd Qu.:5228
##  Max.   :5228
##
```

```r
str(d)
```

```
## 'data.frame':    20593 obs. of  20 variables:
##  $ Age         : int  46 28 44 38 34 31 33 44 40 40 ...
##  $ Job         : Factor w/ 12 levels "admin.","blue-collar",..: 1 8 10 10 10 1 1 11 1
10 ...
##  $ Marital     : Factor w/ 4 levels "divorced","married",..: 1 3 2 2 2 2 2 2 2 2 ...
##  $ Education   : Factor w/ 8 levels "basic.4y","basic.6y",..: 7 4 6 7 6 7 3 4 7 7 ...
##  $ Default     : Factor w/ 3 levels "no","unknown",..: 2 1 2 1 1 1 1 2 2 1 ...
##  $ Housing     : Factor w/ 3 levels "no","unknown",..: 1 3 3 2 3 1 2 3 3 1 ...
##  $ Loan        : Factor w/ 3 levels "no","unknown",..: 1 3 1 2 1 1 2 1 3 1 ...
##  $ Contact     : Factor w/ 2 levels "cellular","telephone": 1 1 2 1 1 2 2 1 2 1 ...
##  $ Month       : Factor w/ 10 levels "apr","aug","dec",..: 4 7 5 7 2 4 7 2 7 8 ...
##  $ Day         : Factor w/ 5 levels "fri","mon","thu",..: 3 5 1 3 1 2 4 5 5 1 ...
##  $ Campaign    : int  1 1 2 1 1 1 1 1 5 1 ...
##  $ pDays       : int  999 999 999 999 999 999 999 999 999 999 ...
##  $ Previous    : int  0 0 0 1 0 0 0 0 0 1 ...
##  $ pOutcome    : Factor w/ 3 levels "failure","nonexistent",..: 2 2 2 1 2 2 2 2 2 1
...
##  $ EmpVarRate  : num  1.4 -1.8 1.4 -1.8 1.4 -1.7 1.1 1.4 1.1 -0.1 ...
##  $ ConsPriceIdx: num  93.9 92.9 94.5 92.9 93.4 ...
##  $ ConsConfIndx: num  -42.7 -46.2 -41.8 -46.2 -36.1 -40.3 -36.4 -36.1 -36.4 -42 ...
##  $ Euribor3M   : num  4.96 1.28 4.97 1.33 4.97 ...
##  $ NREmployed  : num  5228 5099 5228 5099 5228 ...
##  $ Subscribed  : Factor w/ 2 levels "no","yes": 1 2 1 1 1 1 1 1 1 1 ...
```

The data appears to be accurately coded: continuous variables are coded as integers or numbers and categorical variables are coded as factors.

Missing Values

Let’s see look at which variables are missing. For this dataset, missing values have been coded as “unknown”:

```r
m = matrix(data=NA,nrow=length(names(d)),ncol=2)
dimnames(m) = list(names(d),c("Total Missing","Percent Missing"))
for(i in 1:nrow(m)){
    l = length(d[,i])
    u = length(which(d[,i]=="unknown"))
    m[i,"Total Missing"] = u
    m[i,"Percent Missing"] = round(u/l,digits=4)
}
m[m[,"Percent Missing"]!=0,]
```

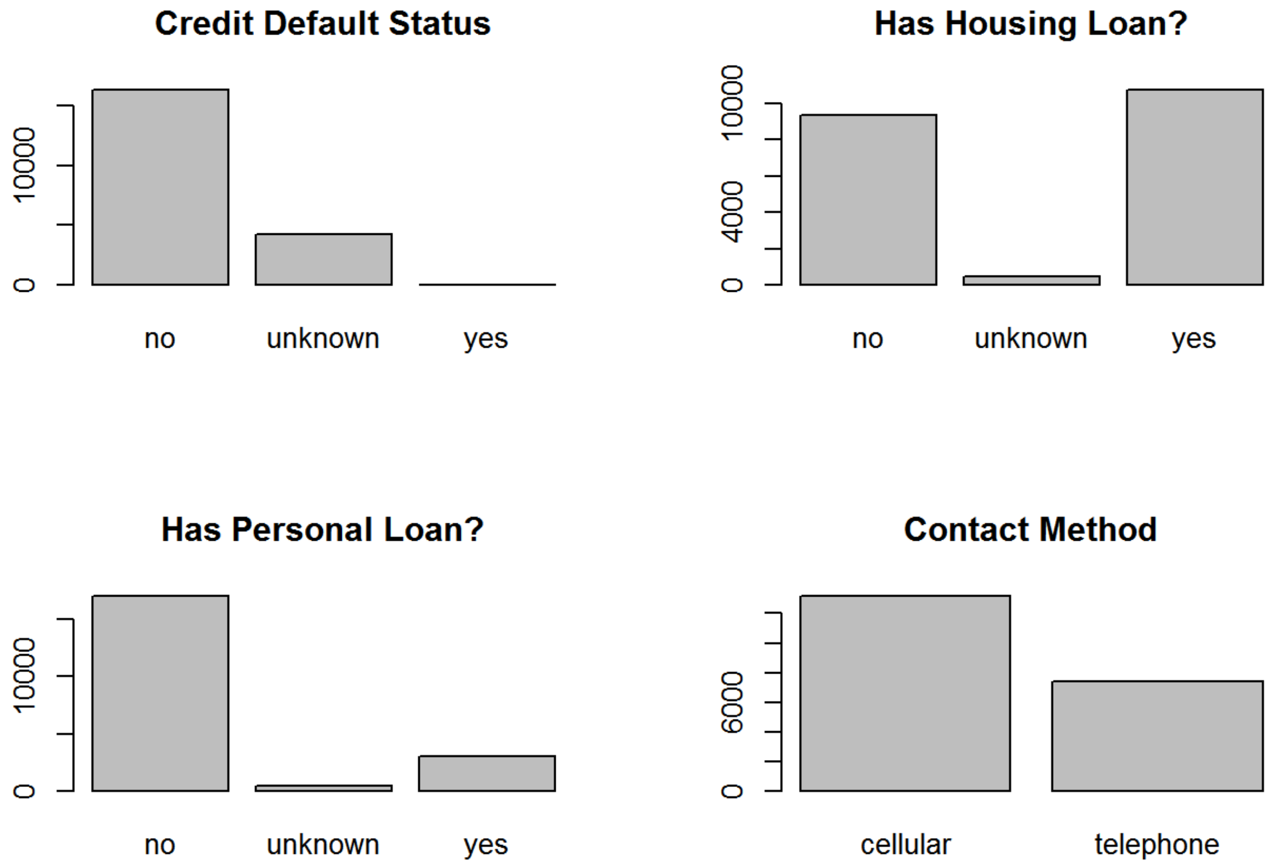
```
##            Total Missing Percent Missing
## Job                  162          0.0079
## Marital               41          0.0020
## Education            865          0.0420
## Default             4275          0.2076
## Housing              497          0.0241
## Loan                 497          0.0241
```

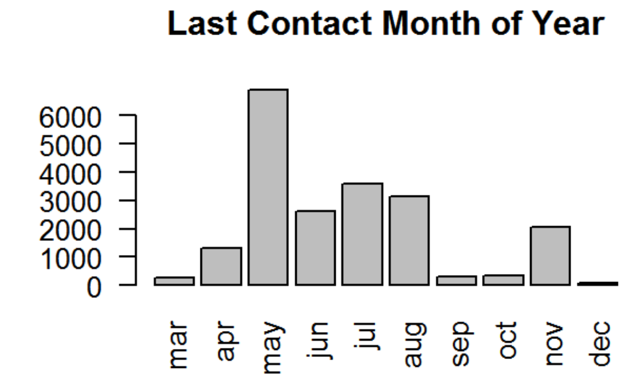
Here we see that we have 6 variables with missing values, ranging from under 1% missing up to just over 20% missing. “Default” is the most problematic variable here, with 20% of observations missing. At this point, we could try and imput missing values using techniques like tree-based methods or K-Nearest Neighbors. However, in the interest of exloring how different models can handle missing values, we will not attemp to make any imputations.

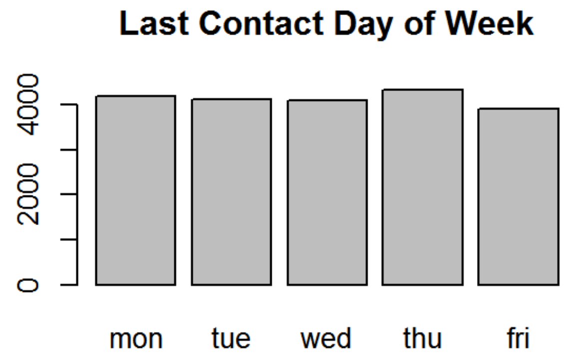
Exploratory Data Analysis

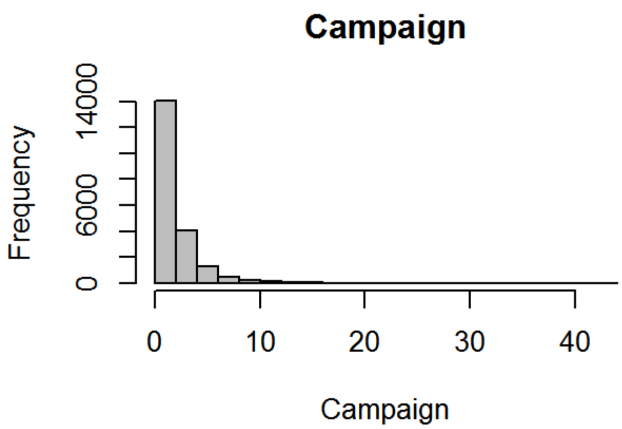
Now that we have updated incorrect datatypes and recoded missing data, let’s do some exploratory data analysis by looking at univariate plots of the data.

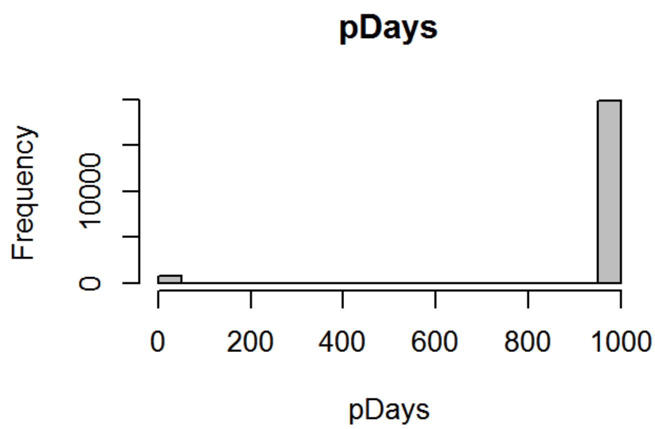
Univariate Plots

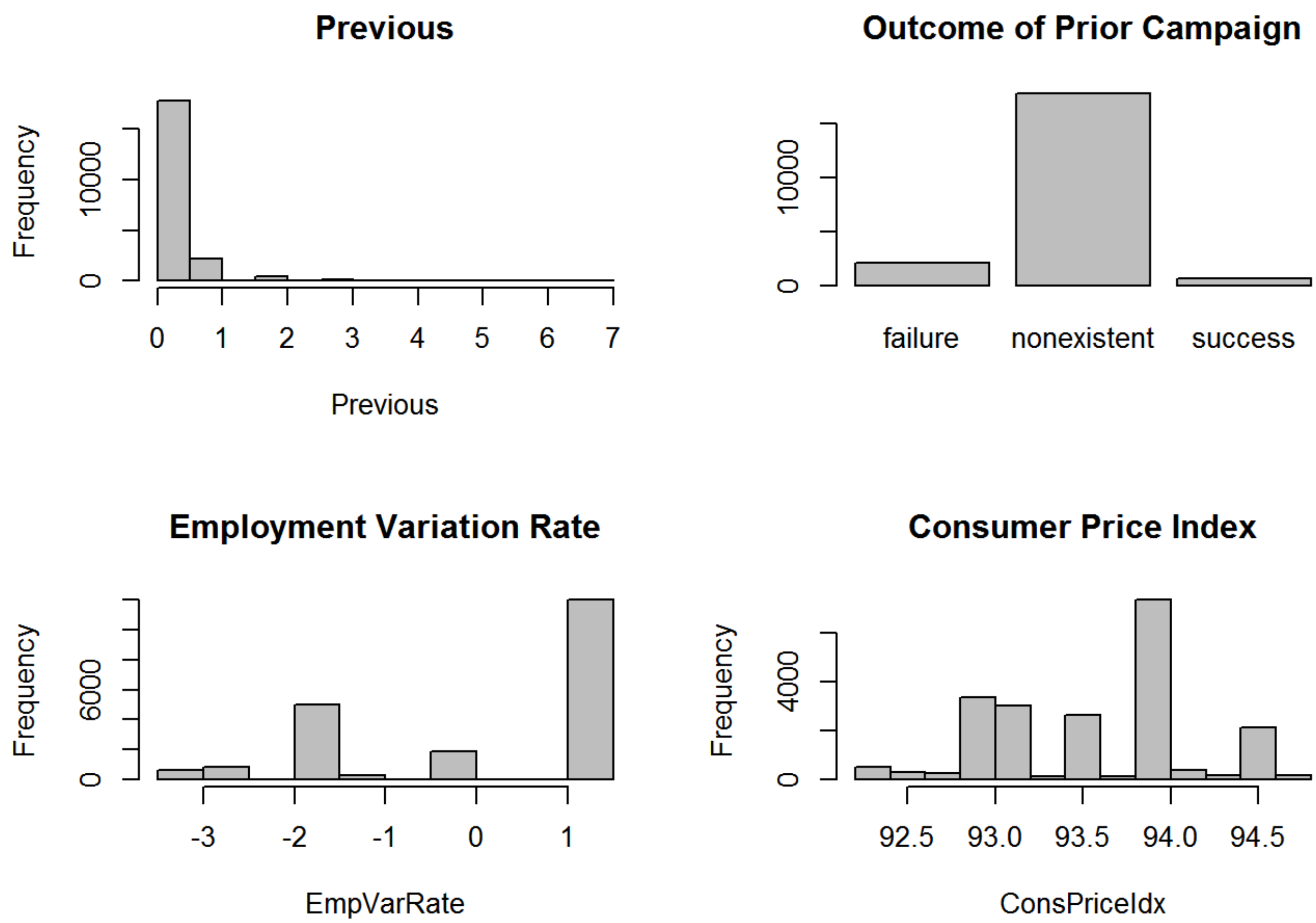

Credit Default Status
0
10000
no
unknown
yes
Has Housing Loan?
0
4000
10000
no
unknown
yes
Has Personal Loan?
0
10000
no
unknown
yes
Contact Method
0
6000
cellular
telephone


Last Contact Month of Year
6000
5000
4000
3000
2000
1000
0
mar
apr
may
jun
jul
aug
sep
oct
nov
dec


Last Contact Day of Week
4000
2000
0
mon
tue
wed
thu
fri


Campaign
Frequency
14000
6000
0
0
10
20
30
40
Campaign

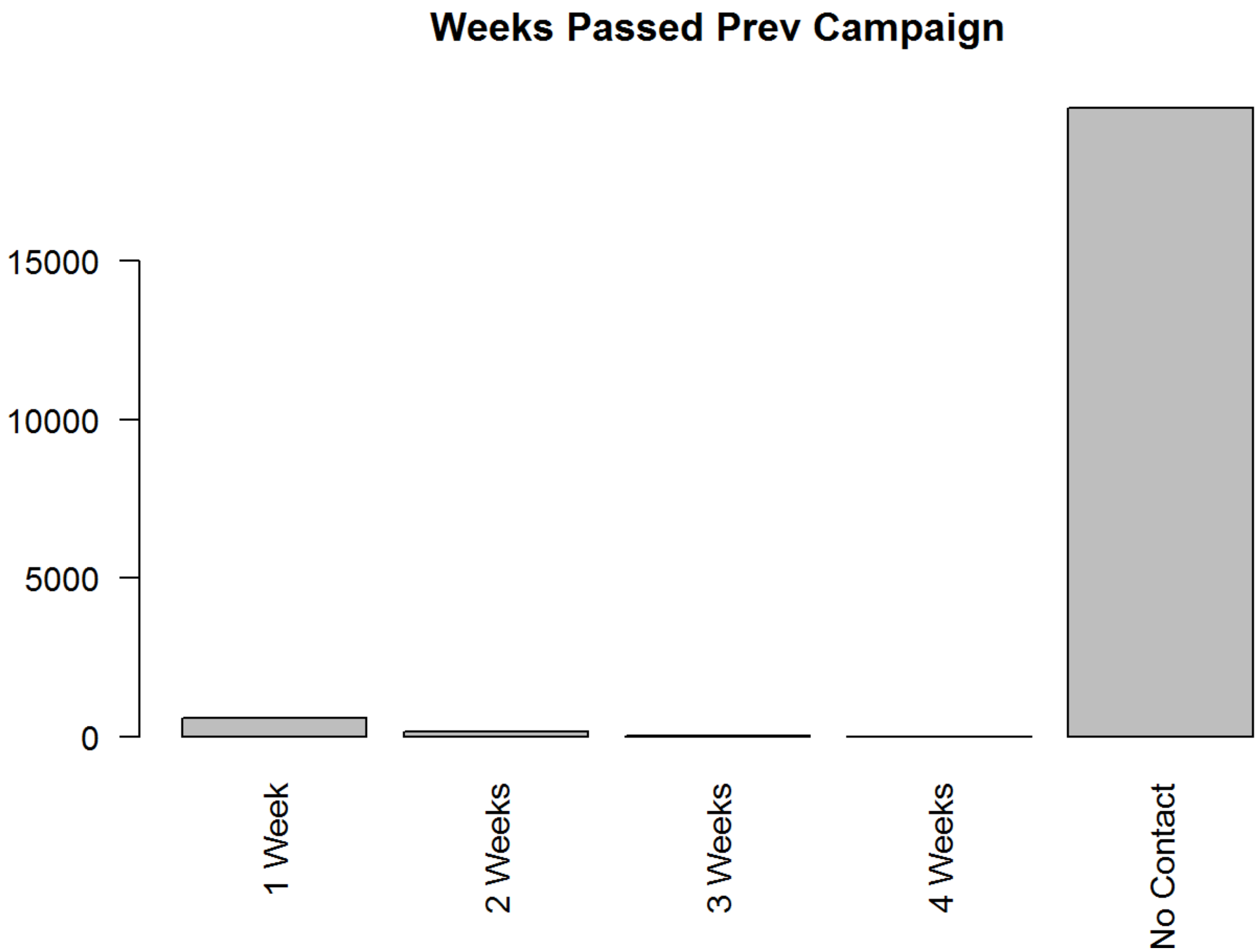

pDays
Frequency
10000
0
0
200
400
600
800
1000
pDays


Previous
Frequency
0
10000
0
1
2
3
4
5
6
7
Previous
Outcome of Prior Campaign
0
10000
failure
nonexistent
success
Employment Variation Rate
Frequency
0
6000
-3
-2
-1
0
1
EmpVarRate
Consumer Price Index
Frequency
0
4000
92.5
93.0
93.5
94.0
94.5
ConsPriceIdx

<u>A few generalized conclusions from looking at univariate plots:</u>

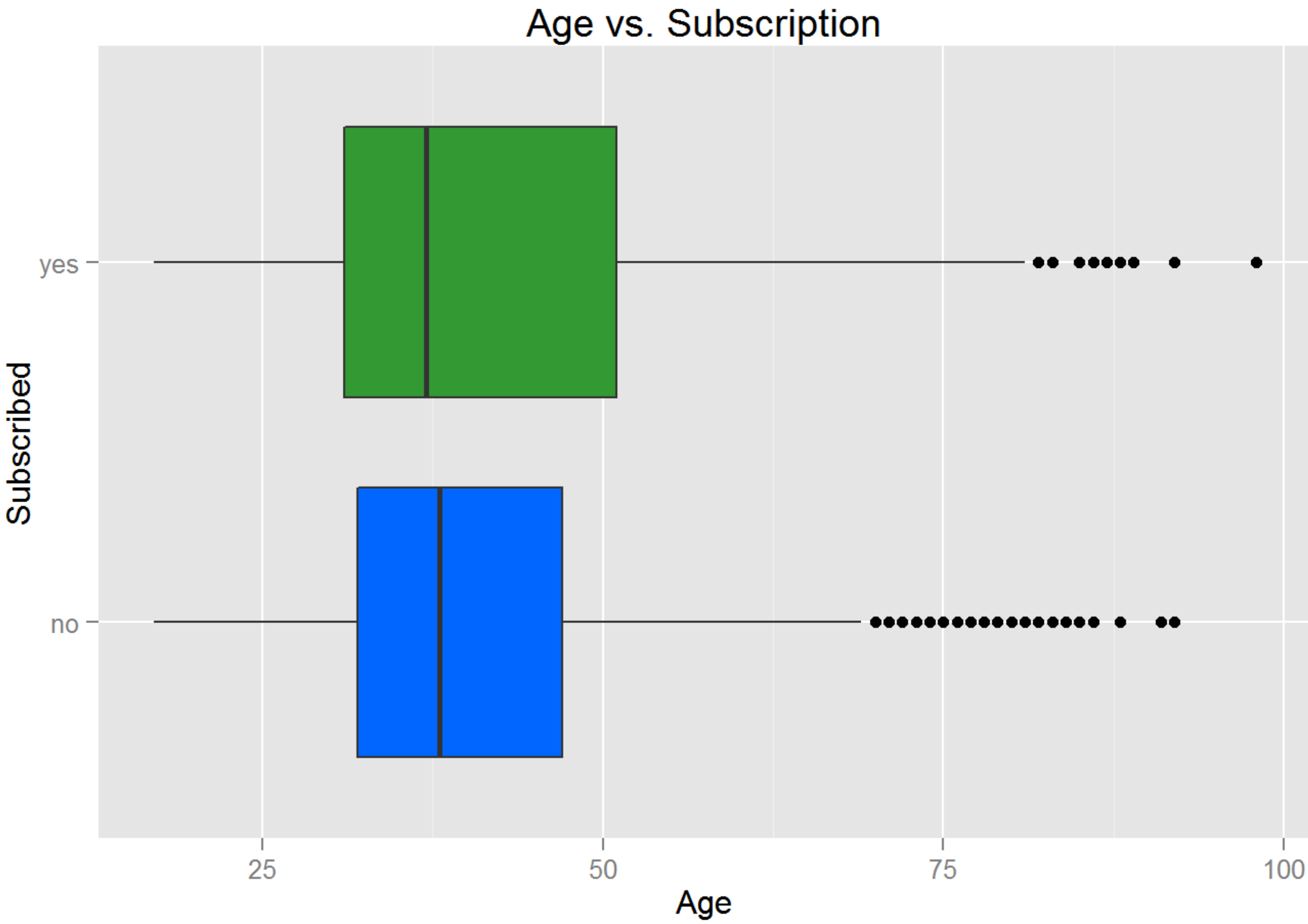
- Several variables are quite skewed/disproportionate:
  Default, Personal Loan, Campaign, pDays, Previous
- Most of these clients were not previously contacted
- pDays needs to be addressed. The instructions state that “if the client was not previously contacted, the variable is coded as”999“. We will recode this varaible and make it more simple by splitting it into categories based on weeks since last contact:

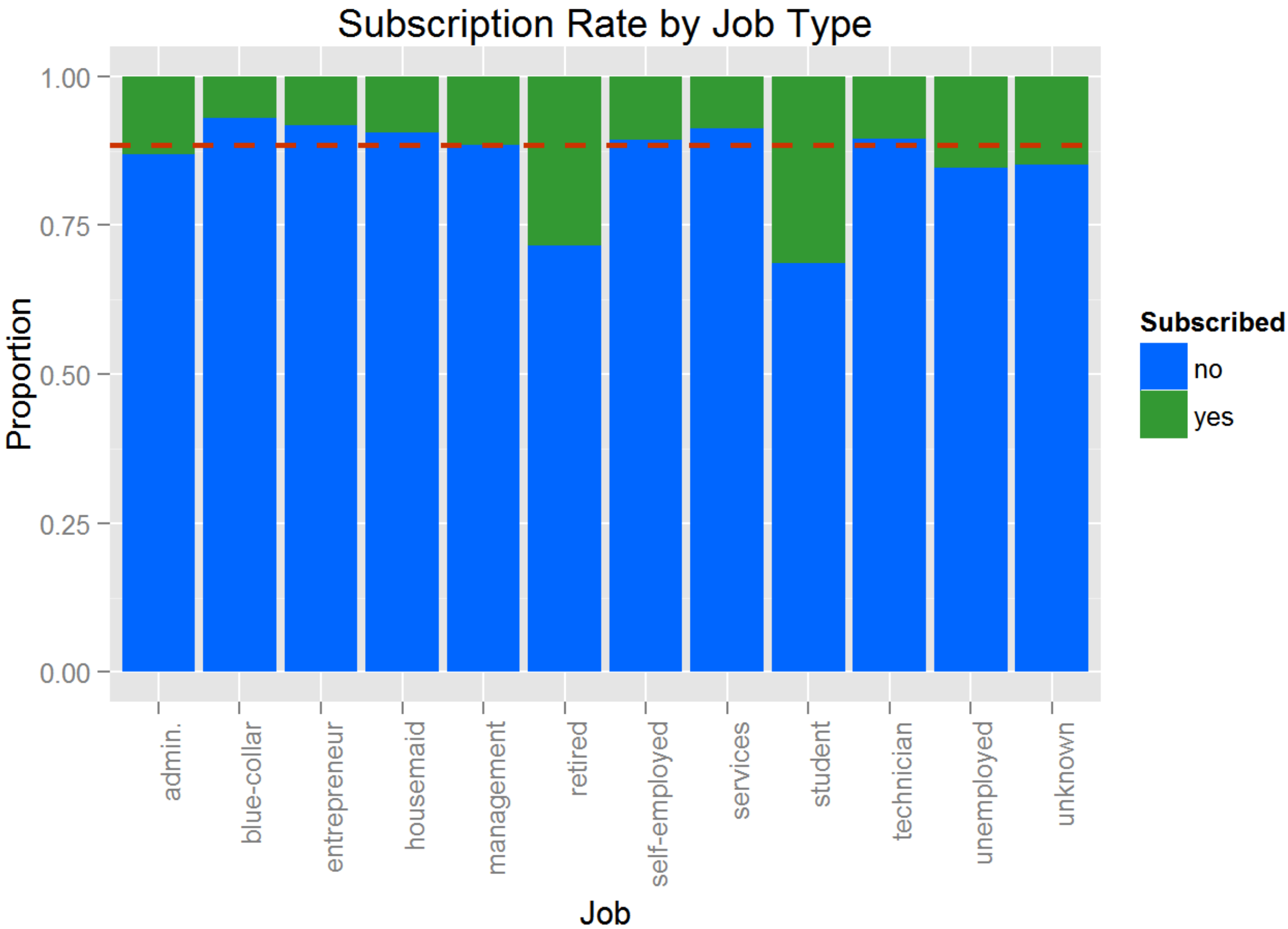
```
d$pContact = as.factor(ifelse(d$pDays<=7,"1 Week",ifelse(d$pDays<=14,"2 Weeks",ifelse(d$pDays<=21,"3 Weeks",ifelse(d$pDays<=28,"4 Weeks","No Contact")))))
d=d[,-which(names(d) %in% c("pDays"))] #remove pDays variable
barplot(table(d$pContact),main="Weeks Passed Prev Campaign",las=2)
```

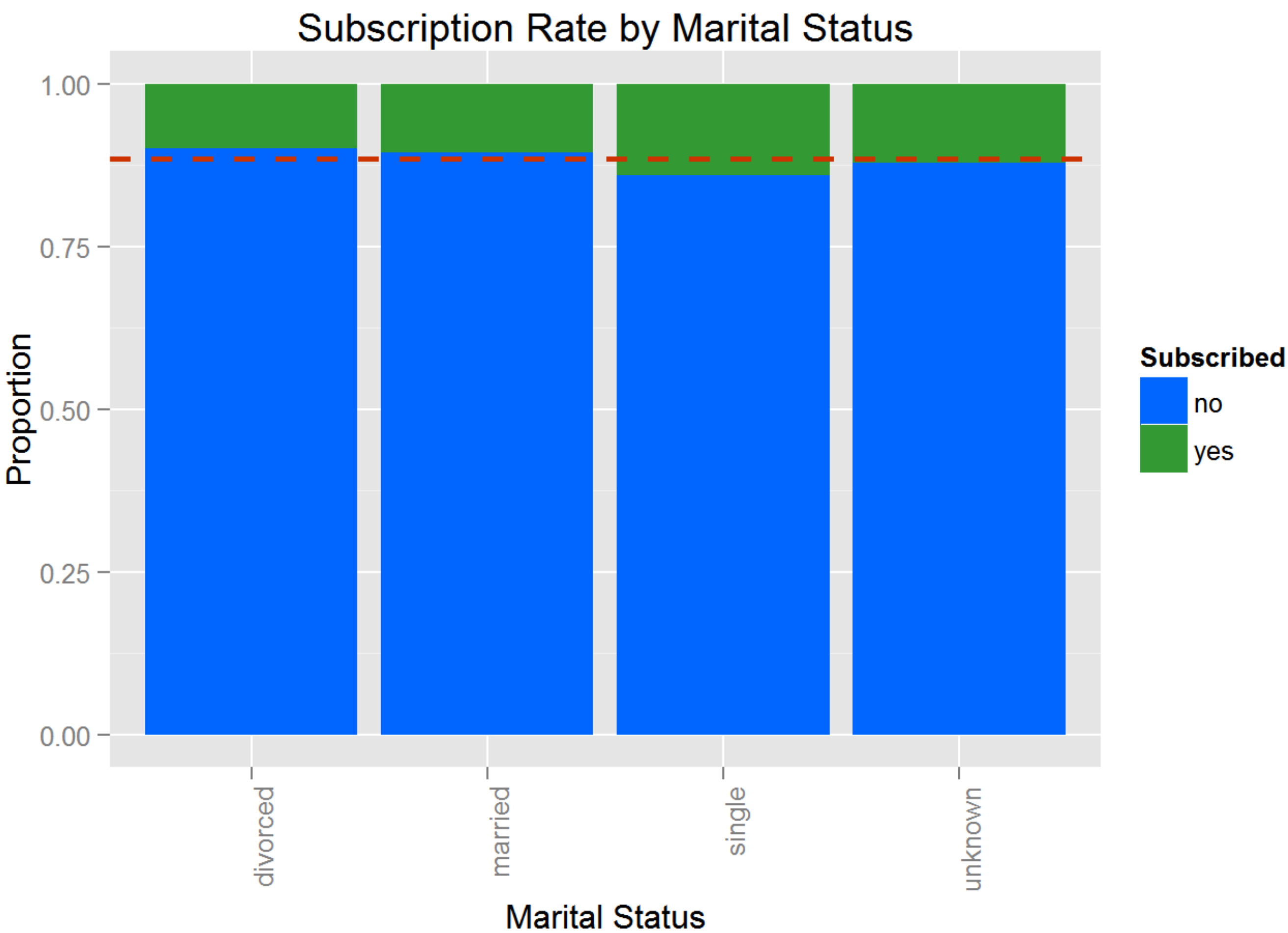


Bivariate Plots

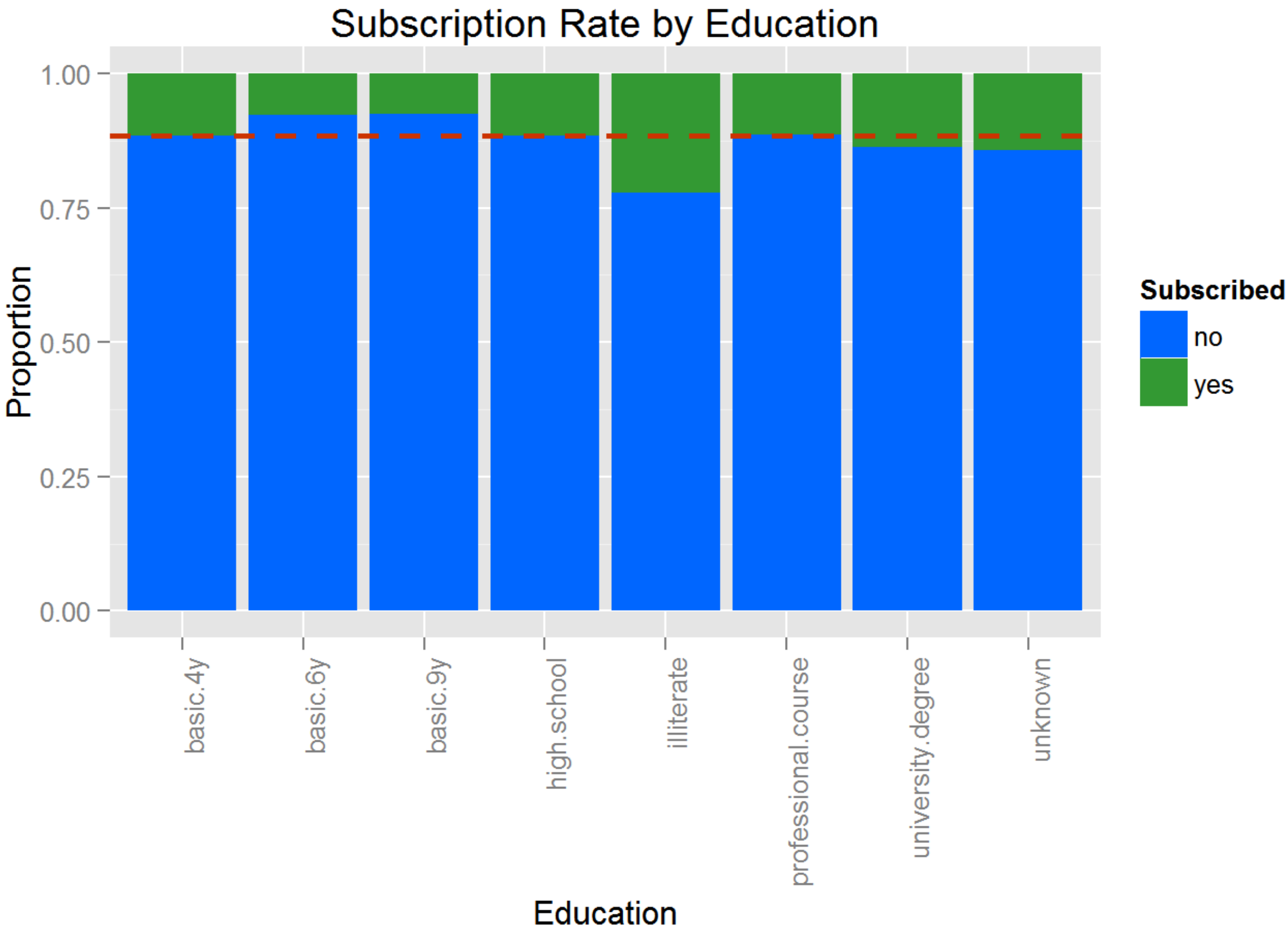
Now let's examine how each of our variables relates to the subscription rate by examining bivariate plots. NOTE: *for each of the mosaic plots below (categorical predictors), a dotted red line has been included that marks the proportion of yes/no for the response accross all observations (~88.52%).*

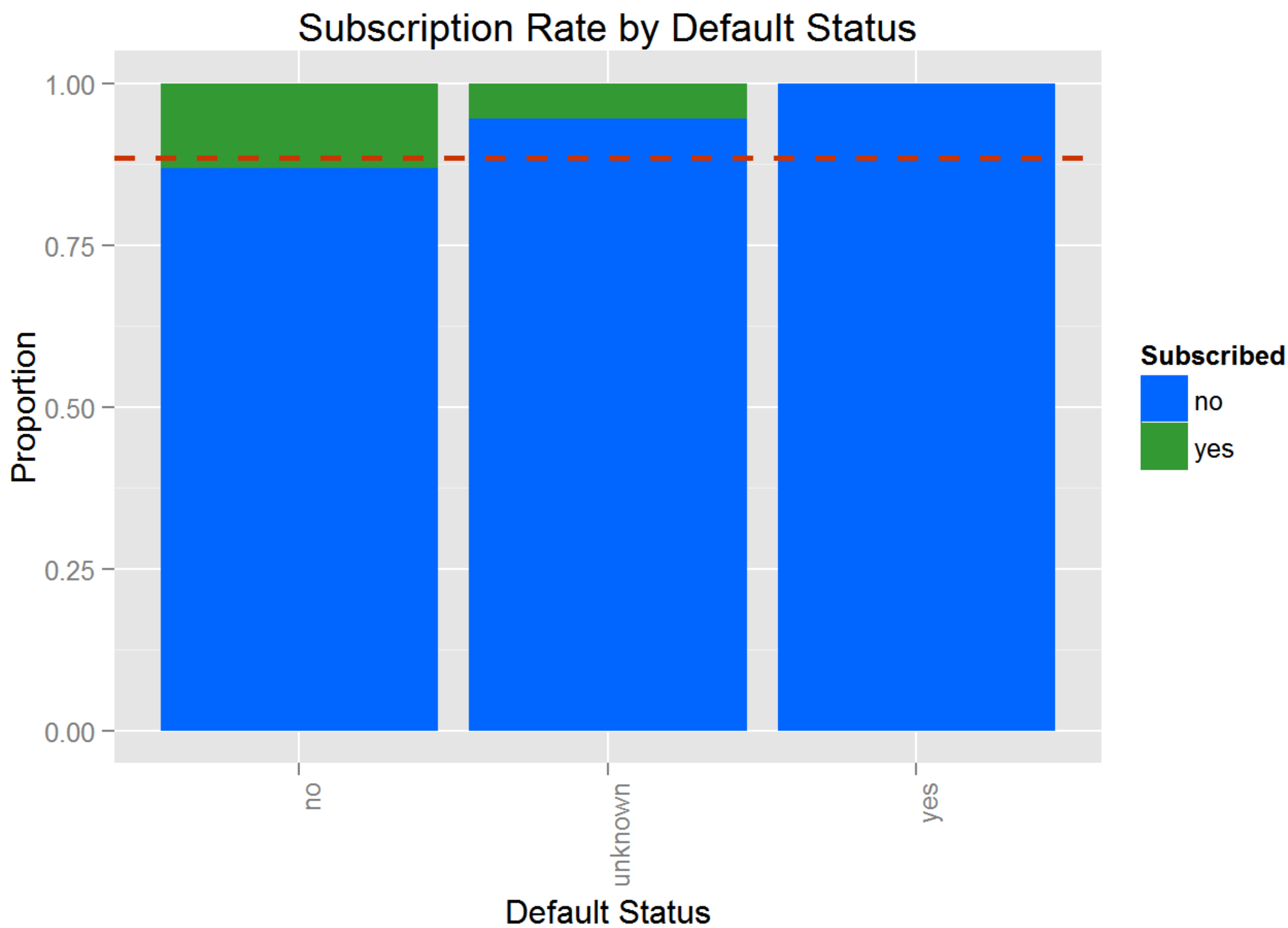

Age vs. Subscription
yes
no
Subscribed
25
50
75
100
Age

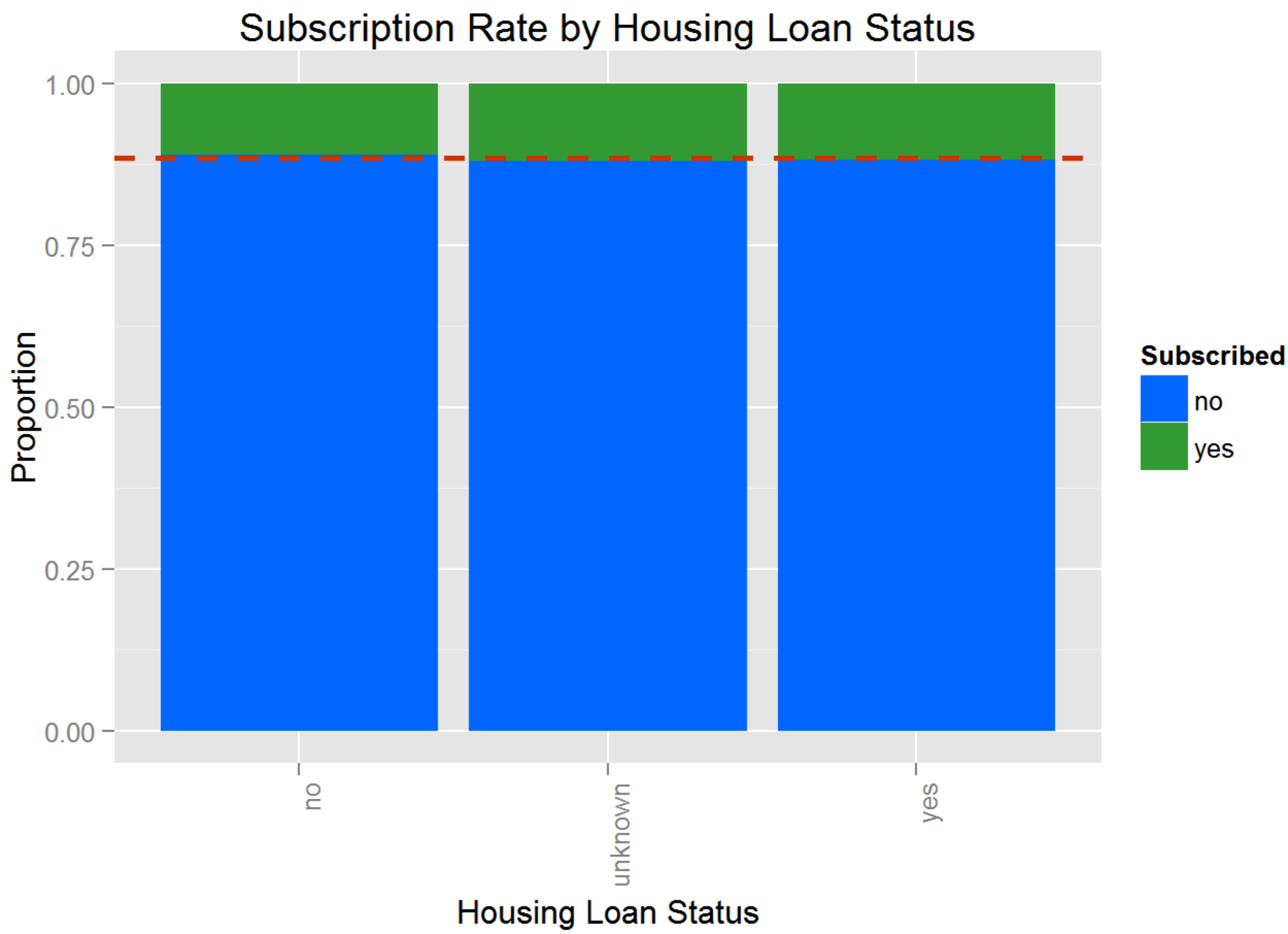

Subscription Rate by Job Type
Proportion
1.00
0.75
0.50
0.25
0.00
admin.
blue-collar
entrepreneur
housemaid
management
retired
self-employed
services
student
technician
unemployed
unknown
Job
Subscribed
no
yes

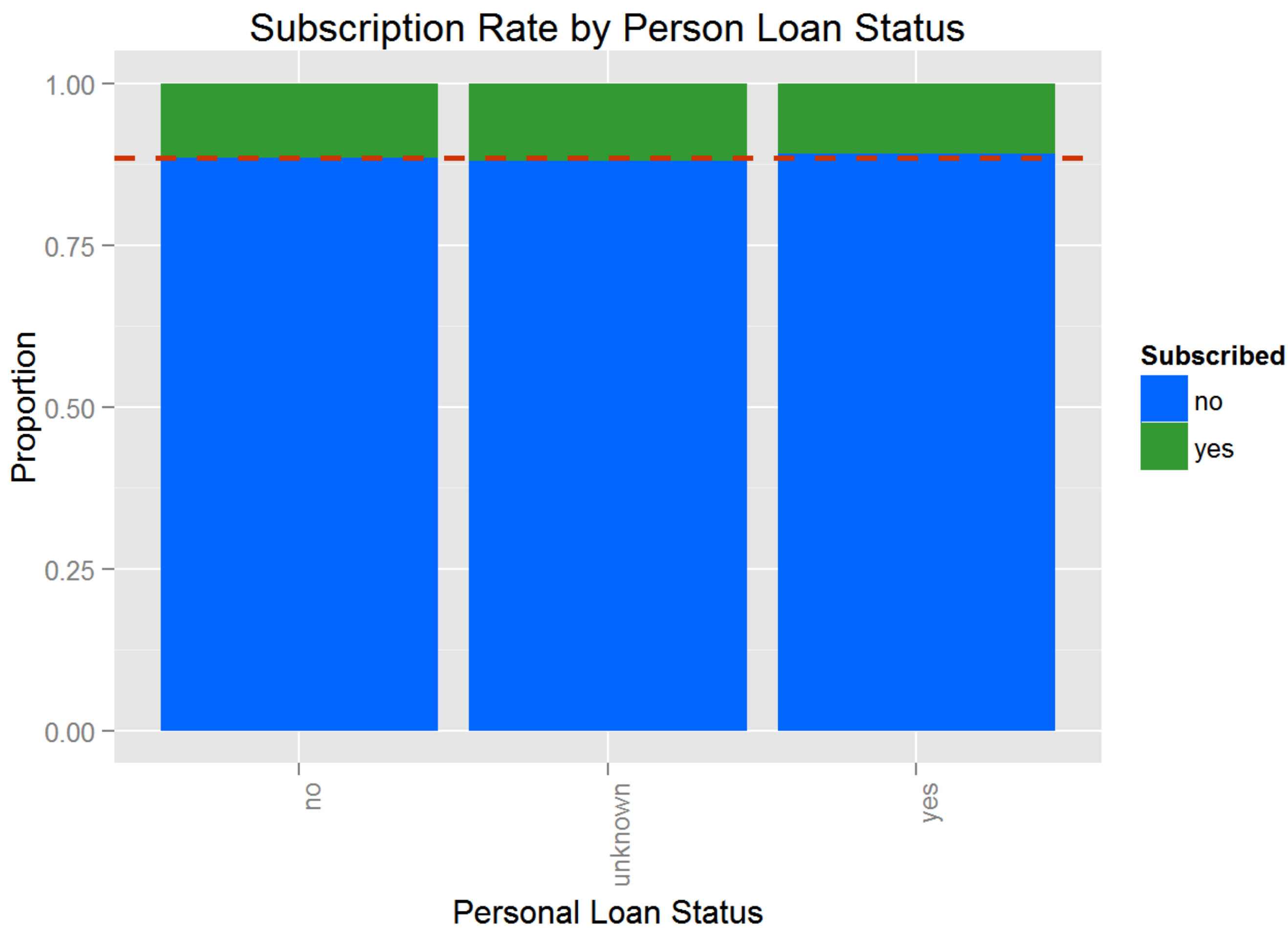


Subscription Rate by Marital Status
1.00
0.75
0.50
0.25
0.00
Proportion
divorced
married
single
unknown
Marital Status
Subscribed
no
yes

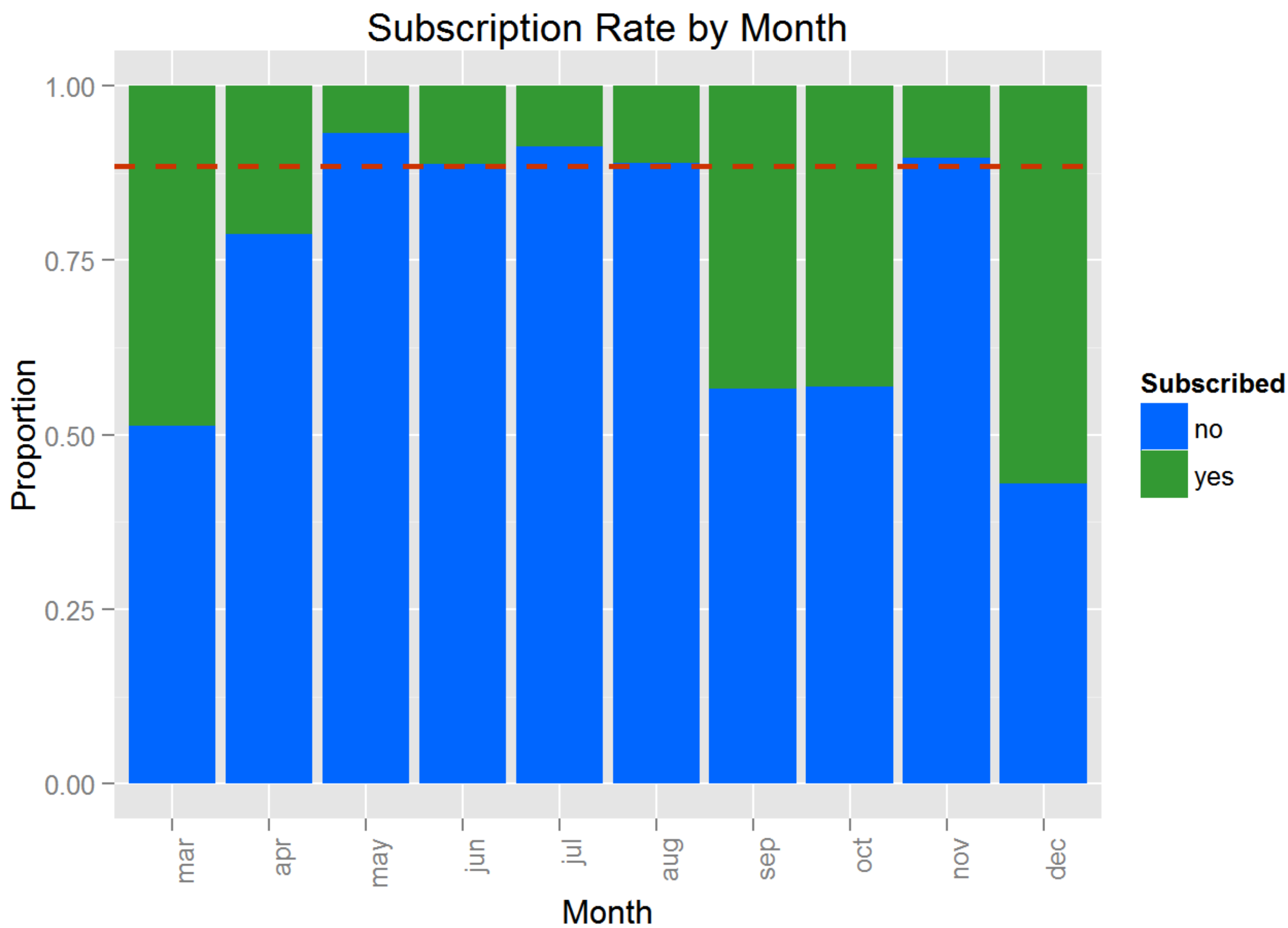

Subscription Rate by Education
1.00
0.75
0.50
0.25
0.00
Proportion
basic.4y
basic.6y
basic.9y
high.school
illiterate
professional.course
university.degree
unknown
Education
Subscribed
no
yes

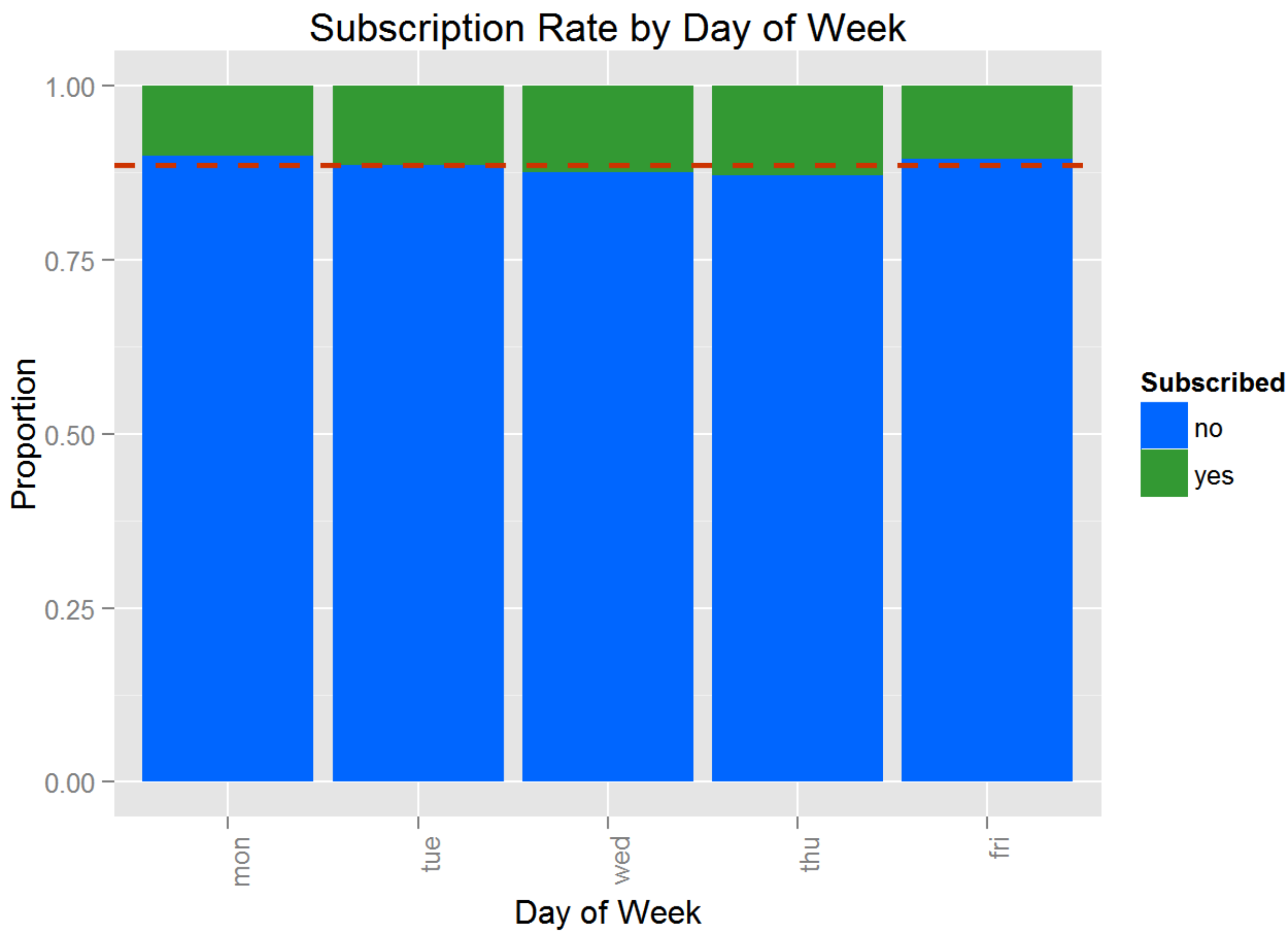

Subscription Rate by Default Status
Proportion
1.00
0.75
0.50
0.25
0.00
no
unknown
yes
Default Status
Subscribed
no
yes

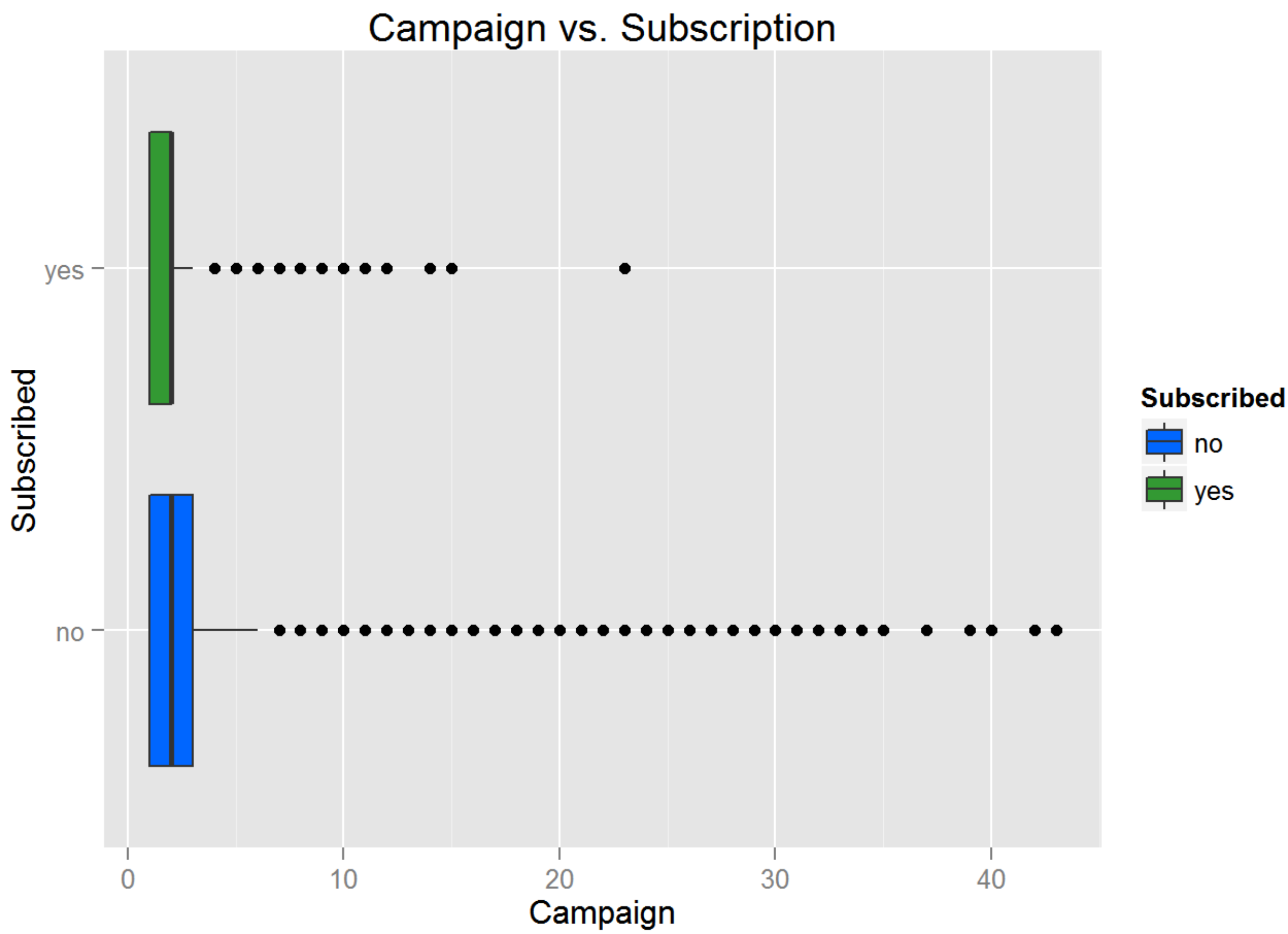

Subscription Rate by Housing Loan Status
1.00
0.75
0.50
0.25
0.00
Proportion
no
unknown
yes
Housing Loan Status
Subscribed
no
yes


Subscription Rate by Person Loan Status
Proportion
1.00
0.75
0.50
0.25
0.00
no
unknown
yes
Personal Loan Status
Subscribed
no
yes

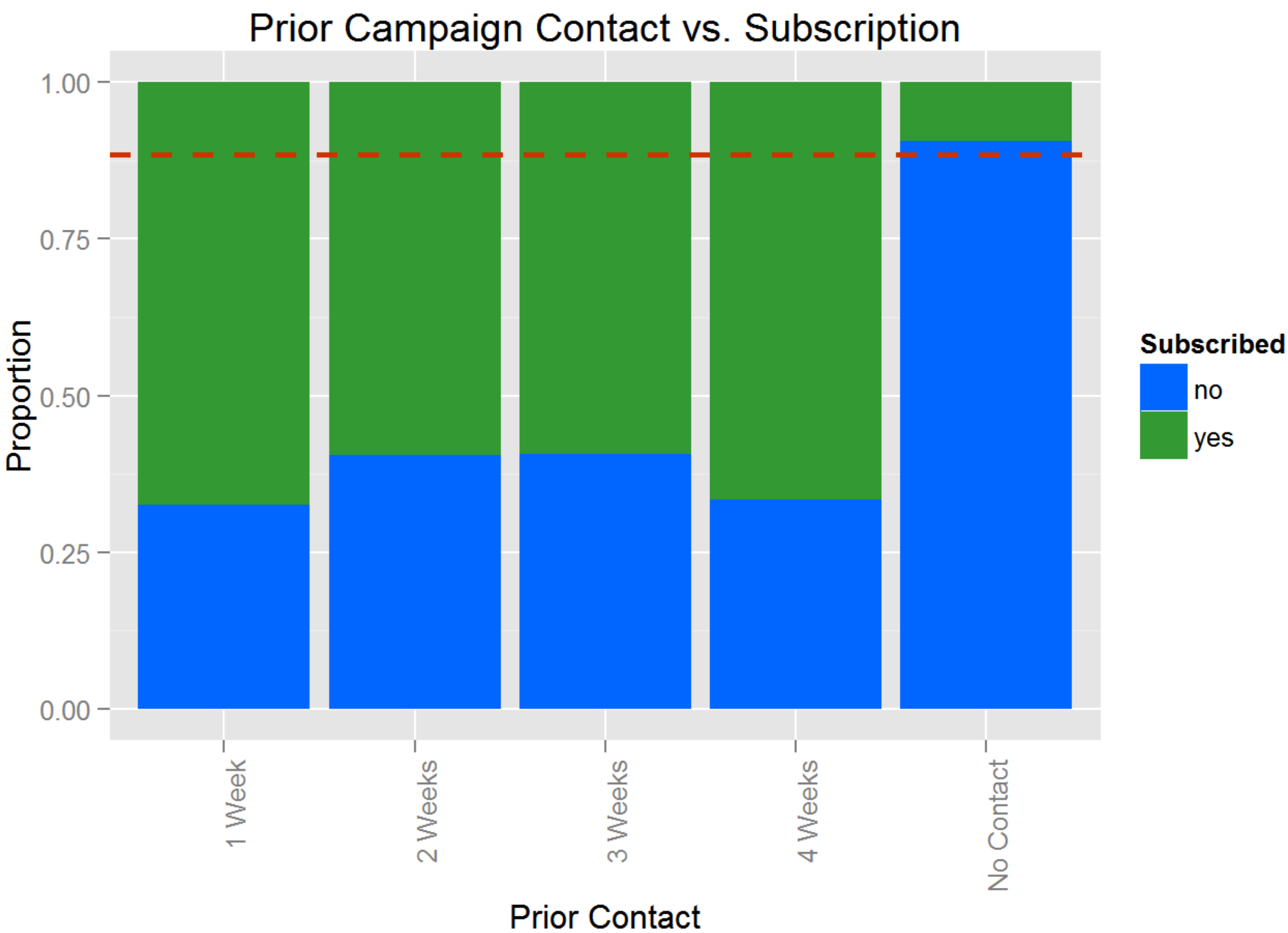

Subscription Rate by Contact Method
Proportion
1.00
0.75
0.50
0.25
0.00
cellular
telephone
Contact Method
Subscribed
no
yes


Subscription Rate by Month
Proportion
1.00
0.75
0.50
0.25
0.00
mar
apr
may
jun
jul
aug
sep
oct
nov
dec
Month
Subscribed
no
yes

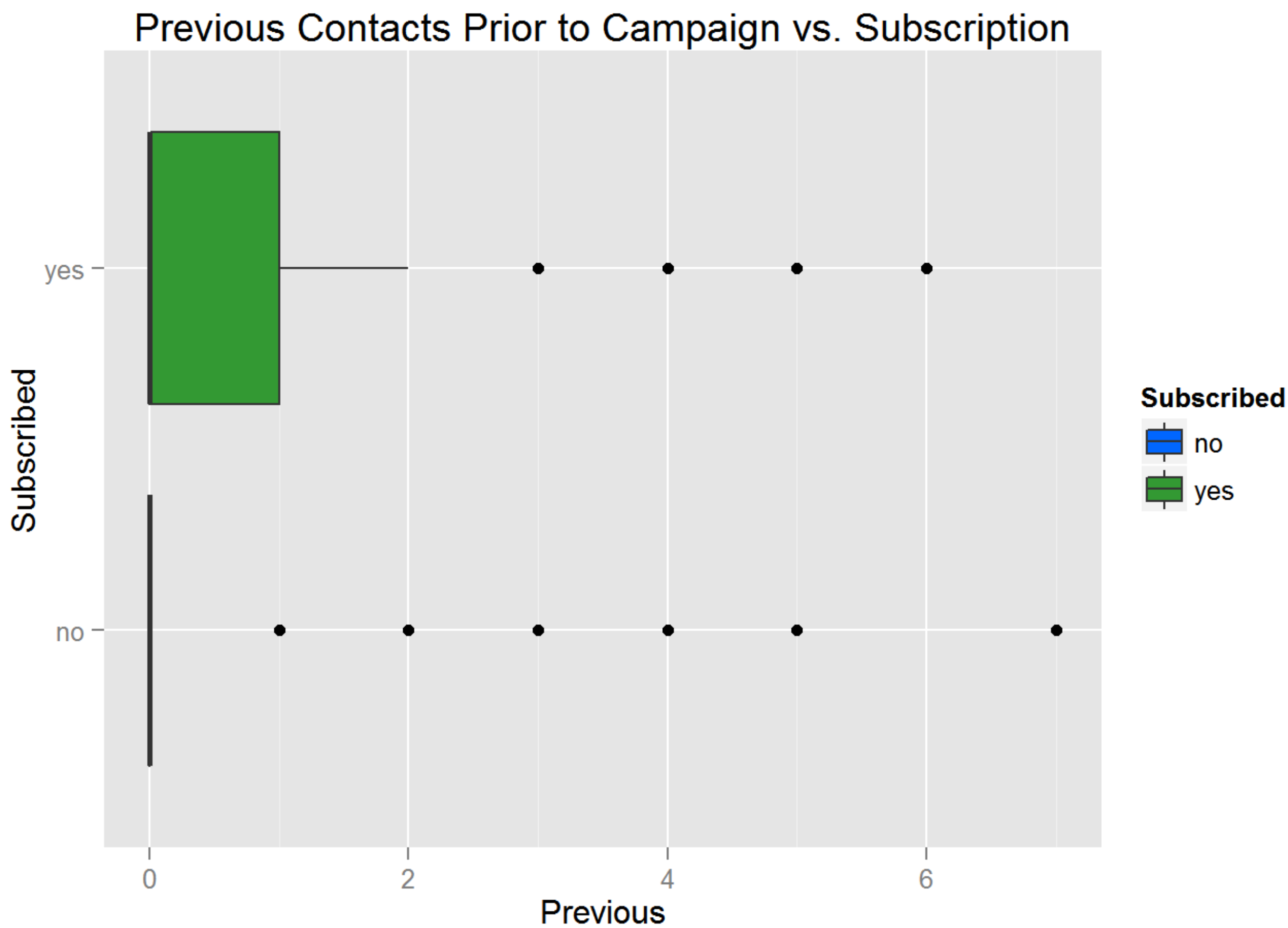

Subscription Rate by Day of Week
Proportion
1.00
0.75
0.50
0.25
0.00
mon
tue
wed
thu
fri
Day of Week
Subscribed
no
yes



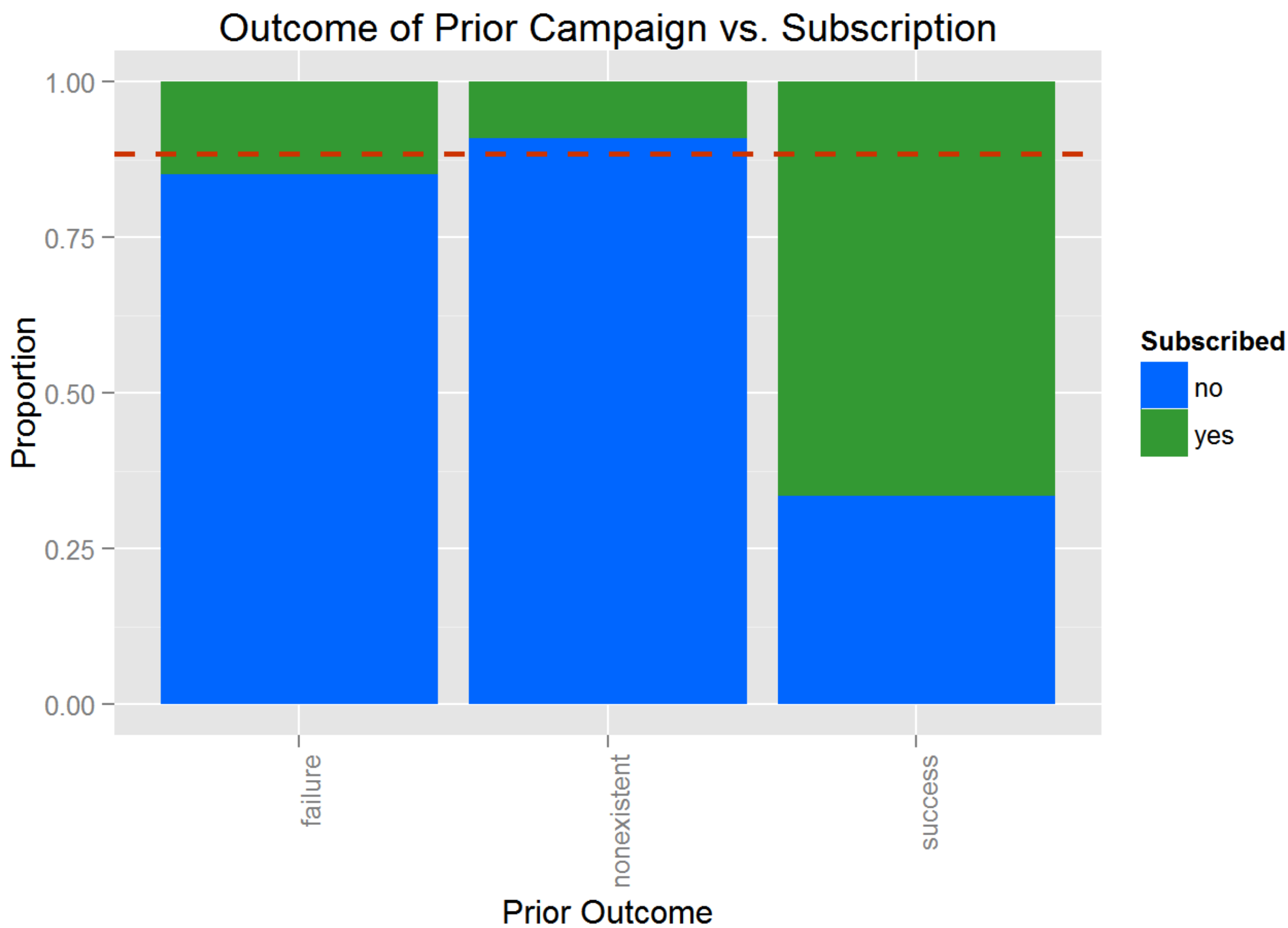
Campaign vs. Subscription
yes
no
Subscribed
0
10
20
30
40
Campaign
Subscribed
no
yes

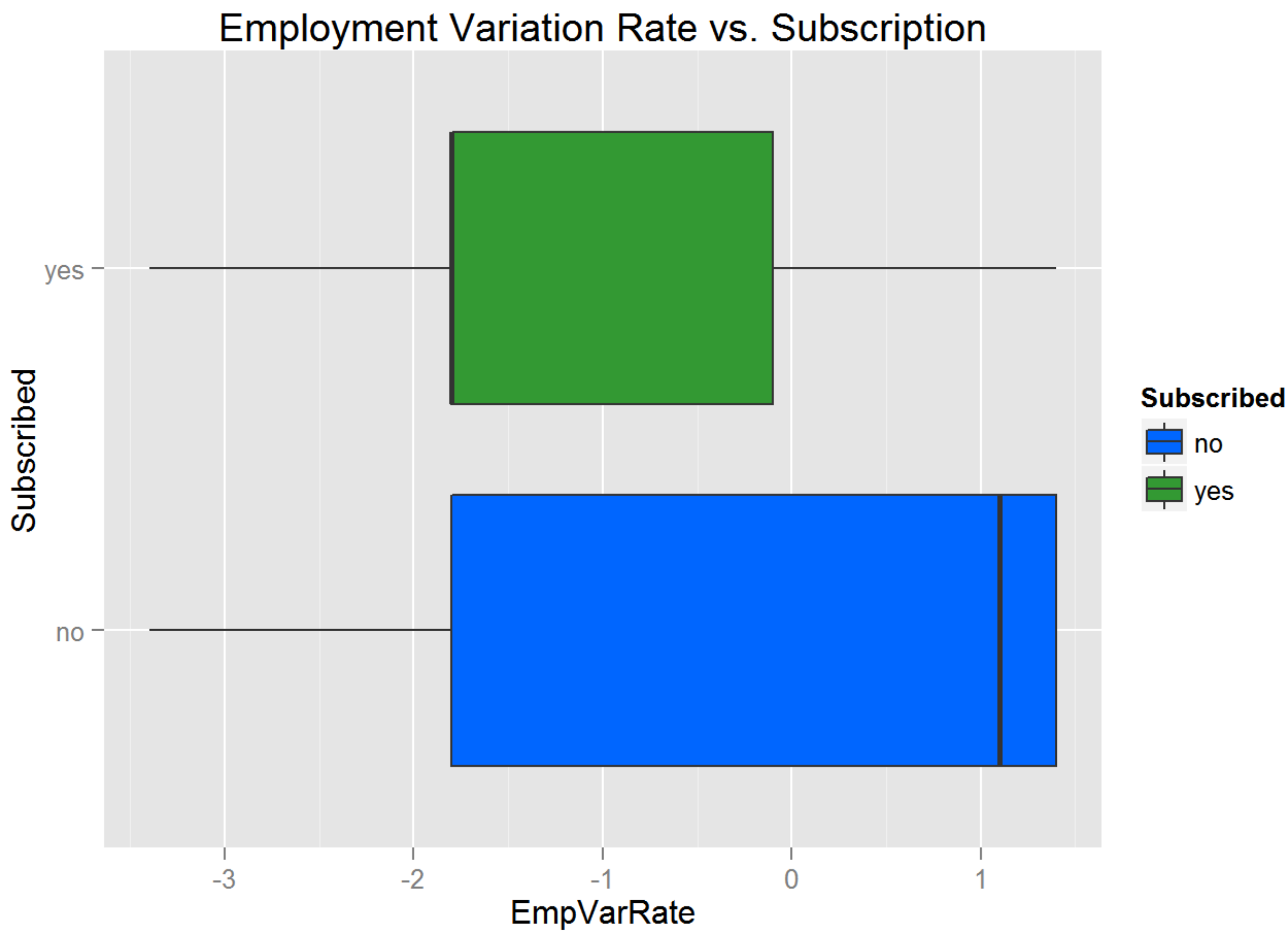


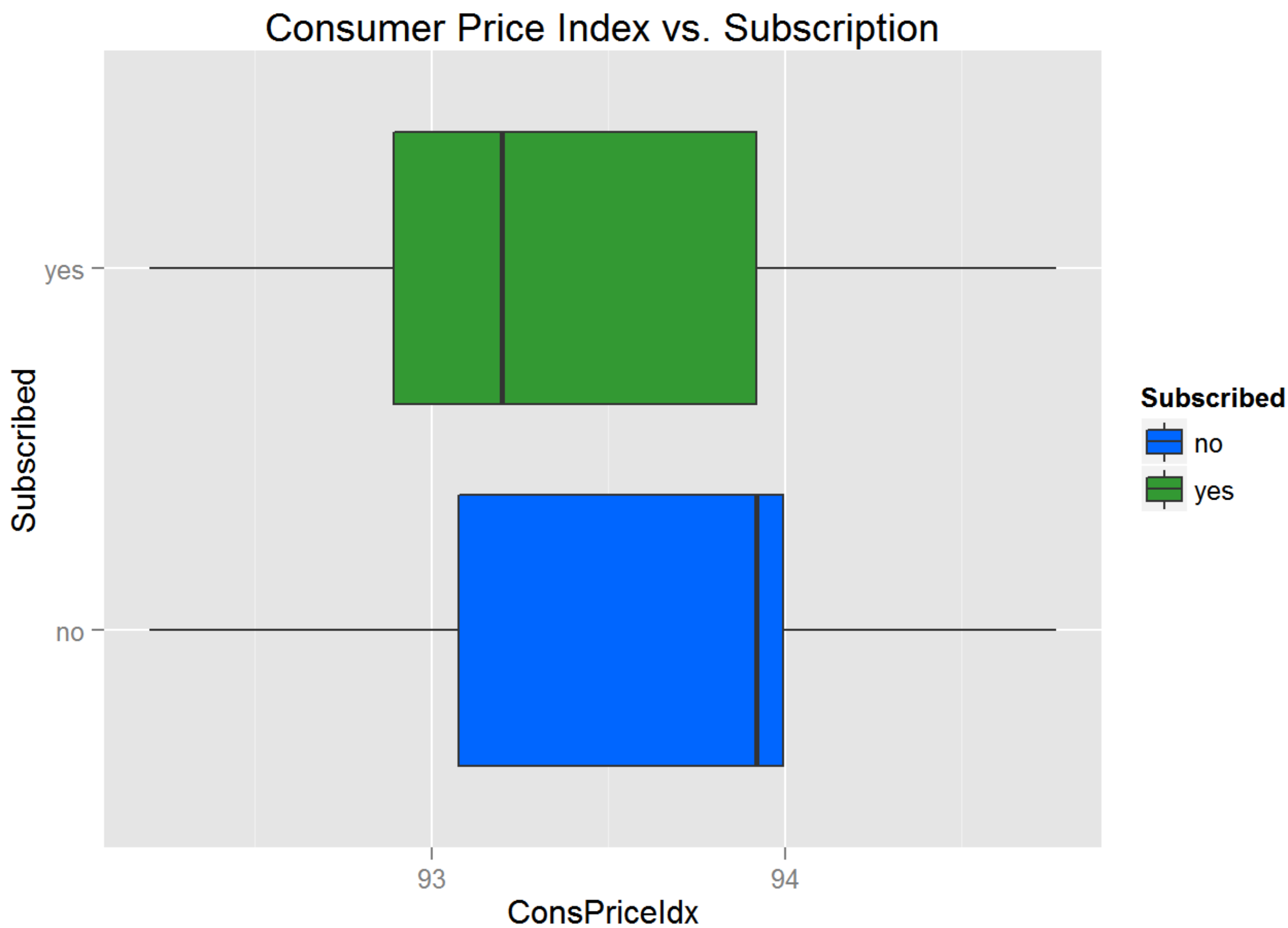
Prior Campaign Contact vs. Subscription
Proportion
1.00
0.75
0.50
0.25
0.00
1 Week
2 Weeks
3 Weeks
4 Weeks
No Contact
Prior Contact
Subscribed
no
yes

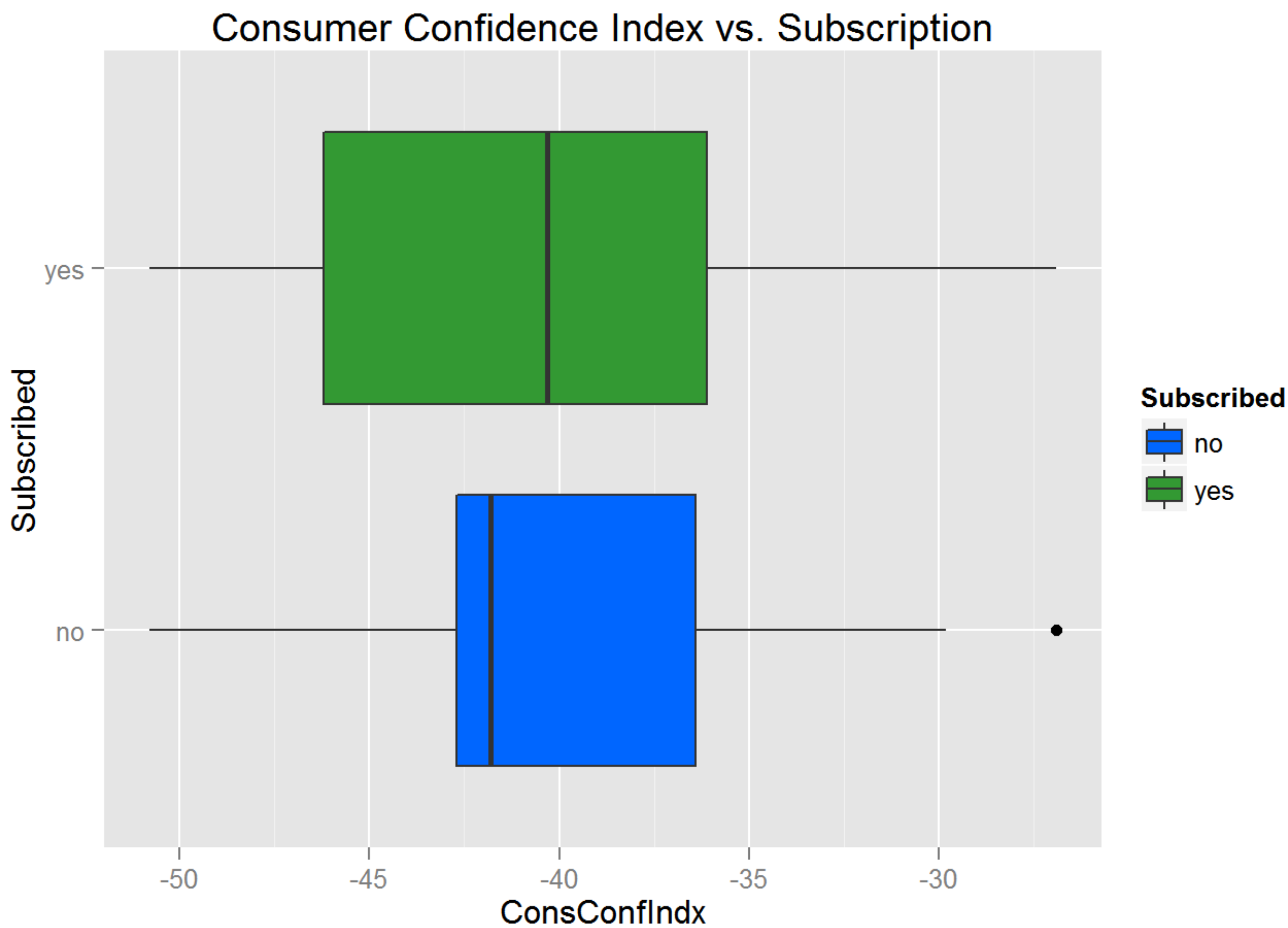


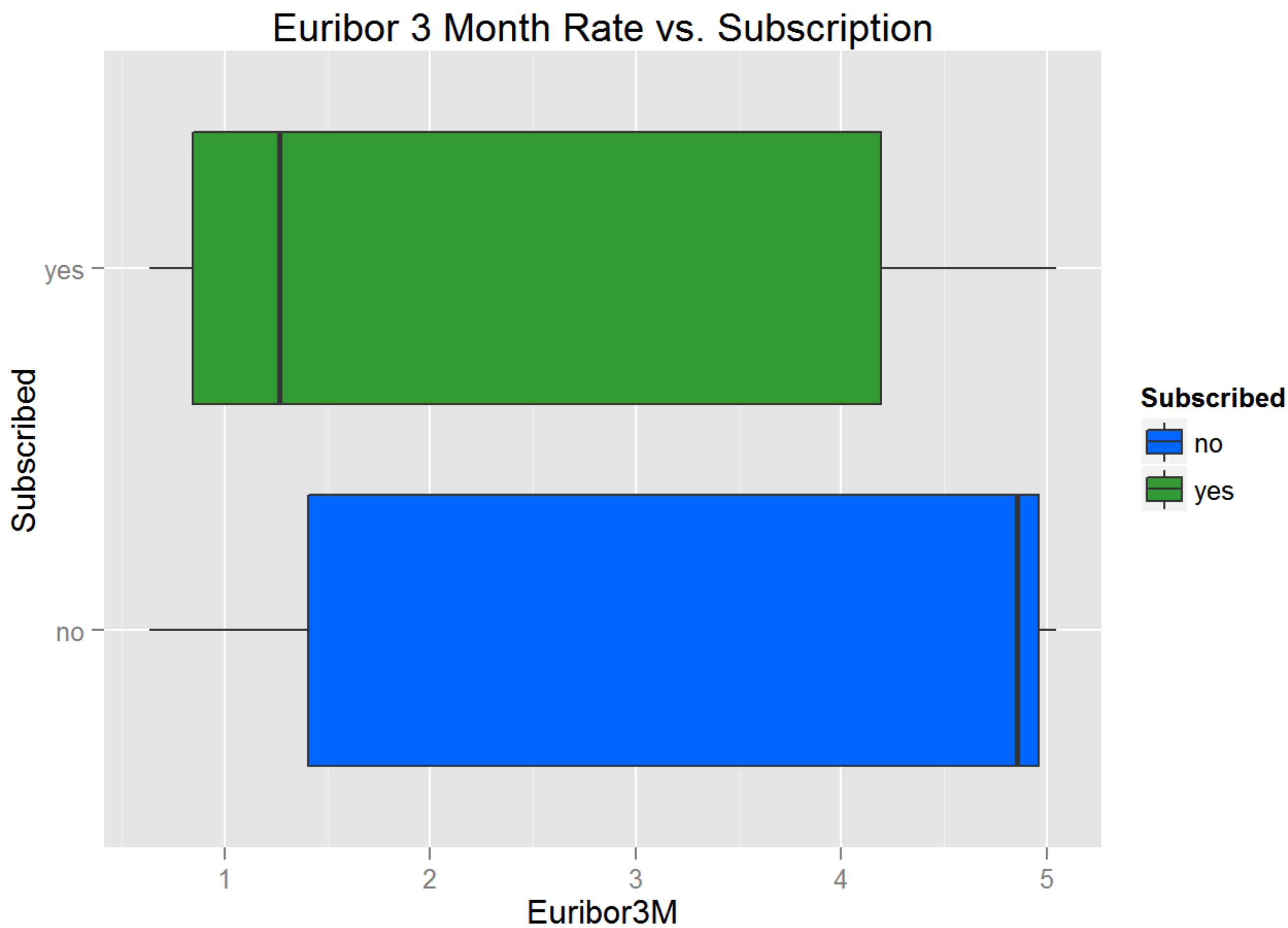
Previous Contacts Prior to Campaign vs. Subscription
Subscribed
yes
no
0
2
4
6
Previous
Subscribed
no
yes

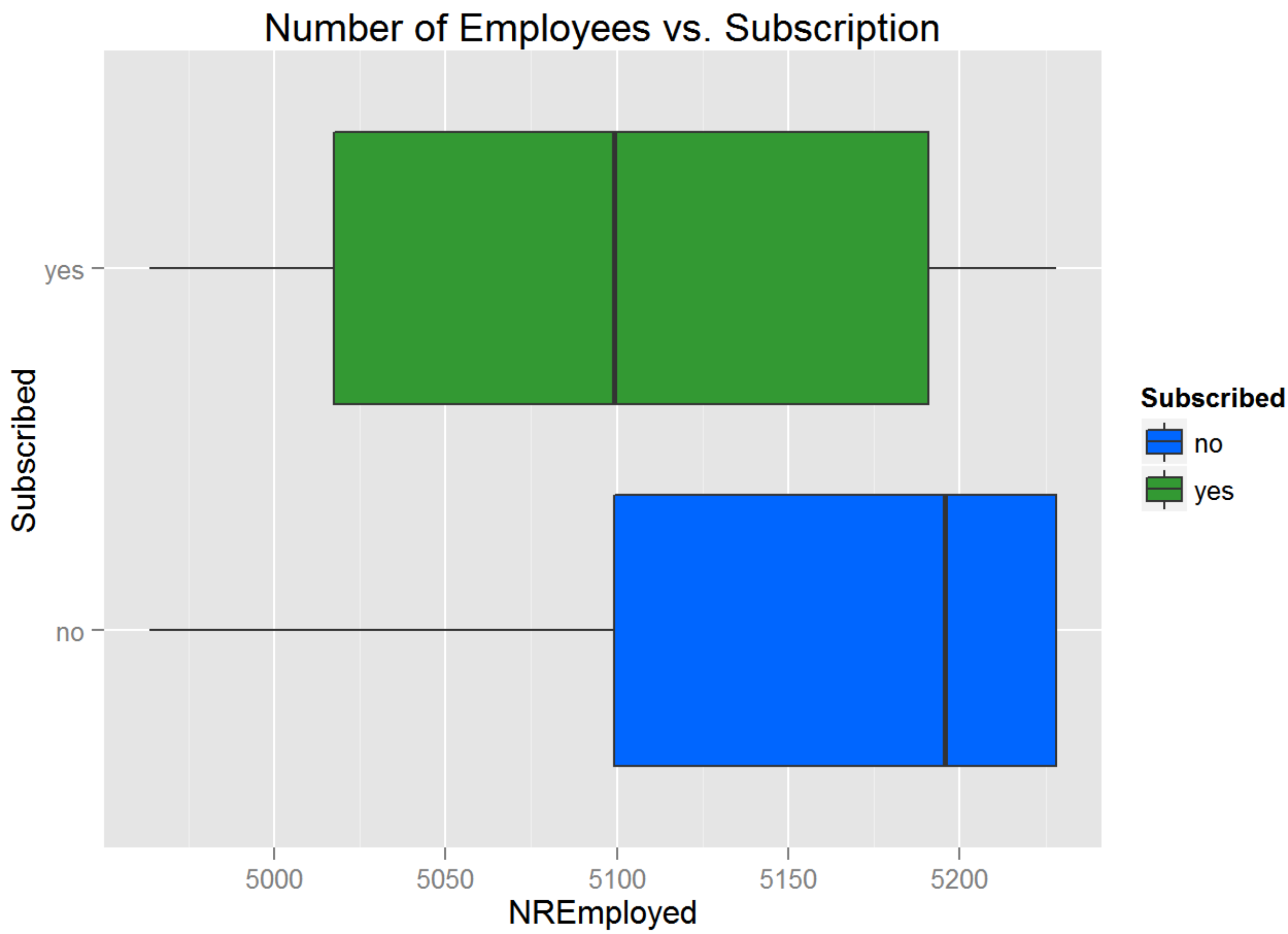

Outcome of Prior Campaign vs. Subscription
1.00
0.75
0.50
0.25
0.00
Proportion
failure
nonexistent
success
Prior Outcome
Subscribed
no
yes


Employment Variation Rate vs. Subscription
yes
no
Subscribed
-3
-2
-1
0
1
EmpVarRate
Subscribed
no
yes


Consumer Price Index vs. Subscription
Subscribed
yes
no
93
94
ConsPriceIdx
Subscribed
no
yes


Consumer Confidence Index vs. Subscription
Subscribed
yes
no
-50
-45
-40
-35
-30
ConsConfIndx
Subscribed
no
yes


Euribor 3 Month Rate vs. Subscription
yes
no
Subscribed
1
2
3
4
5
Euribor3M
Subscribed
no
yes



A few generalized conclusions from looking at bivariate plots:
The following variables exhibit variation in the response and may be strong predictors:

- Job - Retired and Student job types had higher rates of subscription compared to other job types.
- Default Status - Clients who have defaulted did not subscribe at all.
- Contact Method - Clients who were contacted by cell phone subscribed more than those by telephone.
- Month - March, April, September October and December have much higher rates of subscription than others.
- Prior Contact - Clients that had contact of any kind had much higher rates of subscription than those that were never contacted.
- Pervious - Clients that had been contacted in the previous campaign subscribed at a higher rate.
- Prior Outcome - Clients that subscribed in the prior campaign subscribed in this campaign at a higher rate.
- Euibor 3 Month Rate - The lower the rate, it appears the subscription rate increases.
- Number of Employees - Subscription rates seem to be higher at lower employment levels.

Modeling

There are serveral different ways we can model a binary response variable. The classic model is Logistic Regression. For this project, we will be going further and also applying Classification Trees, Bagging, Random Forests and Gradient Boosting. The goal of using five different models is to compare how each model performs on the test dataset we have withheld.

Assumptions

For sceneraios that involve classification, there needs to be special consideration to the costs of False Positives vs. False Negatives. In this case, we intend to use the results of our models to send a direct mailer to potential bank customers. For a real-world scenario, we would have to determine the costs and benefits of producing a direct mailer, gaining a customer, not gaining a customer, increasing direct competition, etc. For our sake, we will assume that the cost of generating the mailer is much less than that of not gaining new customers. Furhter, if we assume that a new customer would generate revenue well beyond the cost of the mailer, and that customers not receiving the mailer will not consider our bank, then we can assume that False Negatives (not sending a mailer when they would have responded) are more costly than False Positives (sending the mailer, not having the customer respond). Additionally, we can assume that unless we reach out to a customer, perhaps they would choose a different bank, which would result in an increase in direct competition. Again, here we would want to minimize False Negatives. Therefore, for all modeling, we will set the cost of False Negatives as **2x** that of False Positives.

Train Set and Validation Set

For modeling, we will split our dataset into 80% for training and 20% for validation:

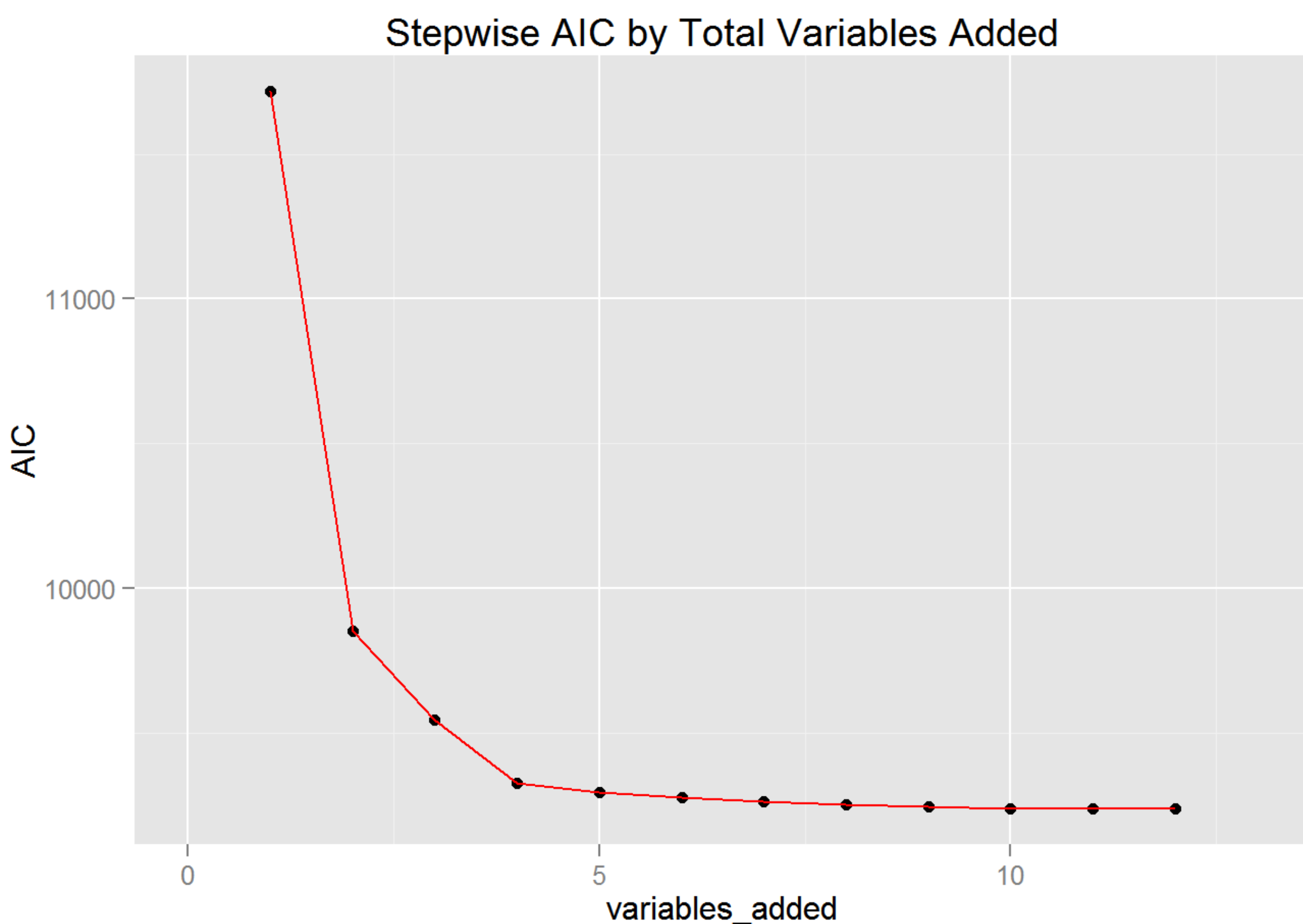
```
set.seed(72) #set seed for reproducibility
train.ind = sample(nrow(d), nrow(d)/5) #split into 5ths
train = d[-train.ind,] #assign training to 80%
val = d[train.ind,] #assign validation to 20%
uc.val = c(0,1)[unclass(val$Subscribed)] #create an unclassed vector of responses from va
l set
```

Logistic Regression

We already have a good idea of which variables might be benficial in a model for this problem. However, we have to be wary of overfitting the model and should look at which variables might be the most predictive. Here we use forward selection to build the best models based on the AIC criterion:

```
full.model = glm(Subscribed~.-Subscribed,data=train,family=binomial(link="logit"))
null.model = glm(Subscribed~1,data=train,family=binomial(link="logit"))
step.model = step(null.model,scope=list(upper=full.model),data=train,direction="both")
```

```
##              Step Df    Deviance Resid. Df Resid. Dev       AIC
## 1                  NA          NA     16474  11714.546 11716.546
## 2     + NREmployed -1 1867.123982     16473   9847.422  9851.422
## 3       + pOutcome -2  308.434291     16471   9538.988  9546.988
## 4          + Month -9  236.612471     16462   9302.375  9328.375
## 5        + Contact -1   36.612642     16461   9265.763  9293.763
## 6            + Day -4   24.749391     16457   9241.013  9277.013
## 7       + pContact -4   20.921131     16453   9220.092  9264.092
## 8       + Campaign -1   13.514836     16452   9206.577  9252.577
## 9  + ConsConfIndx -1    9.764857     16451   9196.812  9244.812
## 10       + Default -1    7.171042     16450   9189.641  9239.641
## 11      + Previous -1    2.760272     16449   9186.881  9238.881
## 12           + Age -1    2.407898     16448   9184.473  9238.473
```

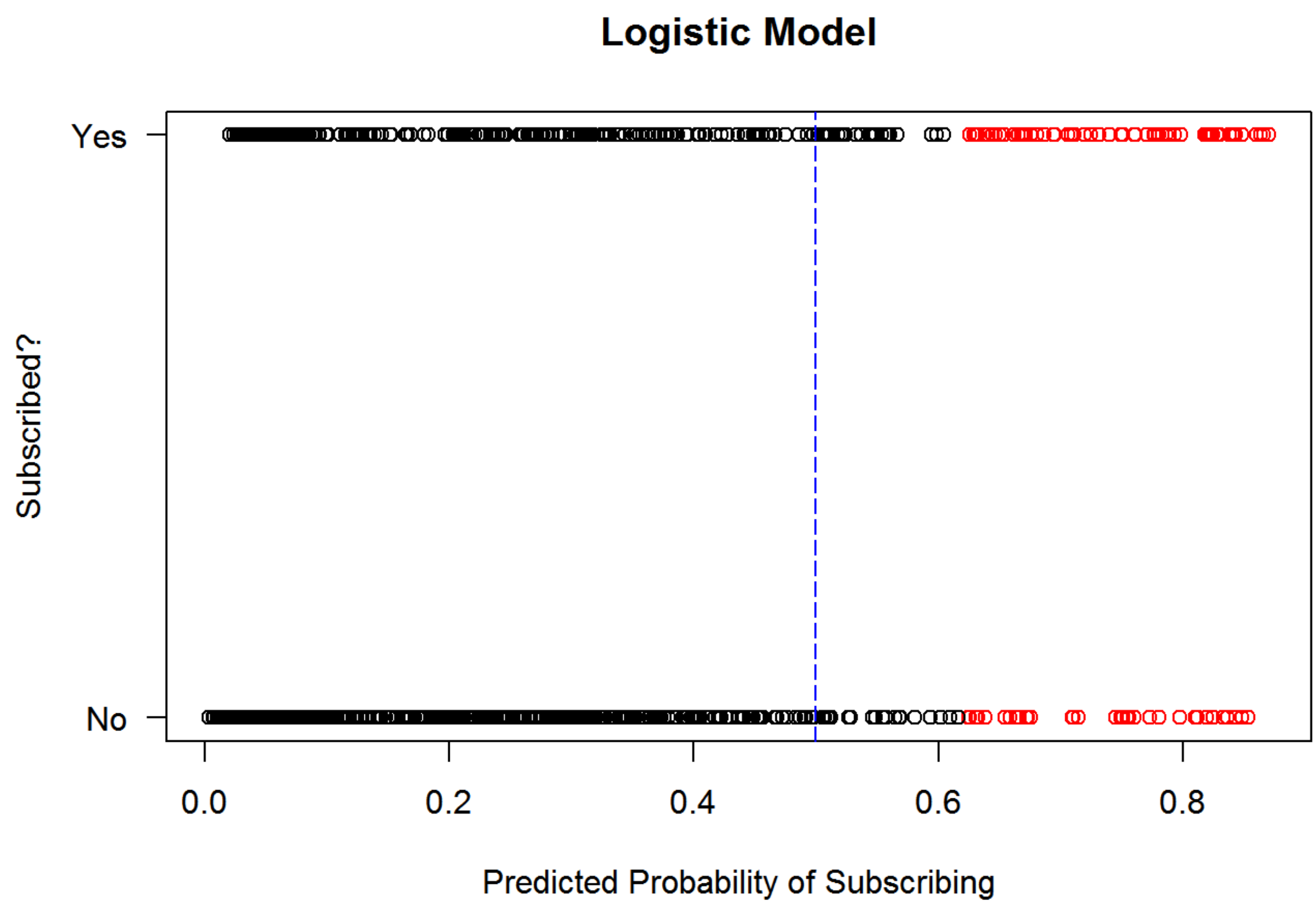


Looks like we hit the minimum AIC around 6 or 7 variables added. There is little benfit to including more variables as we would increase the risk in overfitting the model. Let's build a Logistic Model using the first 7 varaibles from the stepwide method:

```
log.mod = glm(Subscribed~NREmployed+pOutcome+Month+Contact+Day+pContact+Campaign,
              data=train, family = binomial(link="logit"))
summary(log.mod)
```

```
##
## Call:
## glm(formula = Subscribed ~ NREmployed + pOutcome + Month + Contact +
##     Day + pContact + Campaign, family = binomial(link = "logit"),
##     data = train)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -2.0662  -0.3936  -0.3306  -0.2468   3.0124
##
## Coefficients:
##                      Estimate Std. Error z value Pr(>|z|)
## (Intercept)         58.923495   2.273638  25.916  < 2e-16 ***
## NREmployed          -0.011540   0.000452 -25.529  < 2e-16 ***
## pOutcomenonexistent  0.460486   0.087396   5.269 1.37e-07 ***
## pOutcomesuccess      0.566559   0.312982   1.810 0.070265 .
## Monthapr            -0.624848   0.170236  -3.670 0.000242 ***
## Monthmay            -1.315812   0.162775  -8.084 6.29e-16 ***
## Monthjun            -0.331252   0.175609  -1.886 0.059253 .
## Monthjul            -0.548479   0.175951  -3.117 0.001826 **
## Monthaug            -0.521819   0.171911  -3.035 0.002402 **
## Monthsep            -1.153767   0.210564  -5.479 4.27e-08 ***
## Monthoct            -0.640090   0.205460  -3.115 0.001837 **
## Monthnov            -0.825131   0.179564  -4.595 4.32e-06 ***
## Monthdec            -0.094731   0.291133  -0.325 0.744888
## Contacttelephone    -0.463587   0.079821  -5.808 6.33e-09 ***
## Daytue               0.259679   0.090812   2.860 0.004243 **
## Daywed               0.387518   0.089859   4.313 1.61e-05 ***
## Daythu               0.352325   0.087047   4.048 5.18e-05 ***
## Dayfri               0.231371   0.091763   2.521 0.011689 *
## pContact2 Weeks     -0.015303   0.251065  -0.061 0.951398
## pContact3 Weeks     -0.521656   0.463884  -1.125 0.260784
## pContact4 Weeks     -0.726967   1.242801  -0.585 0.558587
## pContactNo Contact  -1.383215   0.322503  -4.289 1.79e-05 ***
## Campaign            -0.051952   0.015072  -3.447 0.000567 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 11714.5  on 16474  degrees of freedom
## Residual deviance:  9206.6  on 16452  degrees of freedom
## AIC: 9252.6
##
## Number of Fisher Scoring iterations: 6
```

Here is a plot showing the data for the Logistic Model and how it would perform if we used a 50% probability threshold (blue vertical line) to classify our response:



Obviously, this is without regard to the cost of False Postives and False Negatives. Instead, let's figure out the optimal threshold taking into account our assumption that False Negatives are twice as costly compared to False Positives:

```
log.pr = predict(log.mod,val,type="response")
log.pred = prediction(log.pr,val$Subscribed)
log.perf = performance(log.pred,"tpr","fpr")
cost.perf = performance(log.pred, "cost", cost.fp = 1, cost.fn = 2)
log.cut = log.pred@cutoffs[[1]][which.min(cost.perf@y.values[[1]])]
```

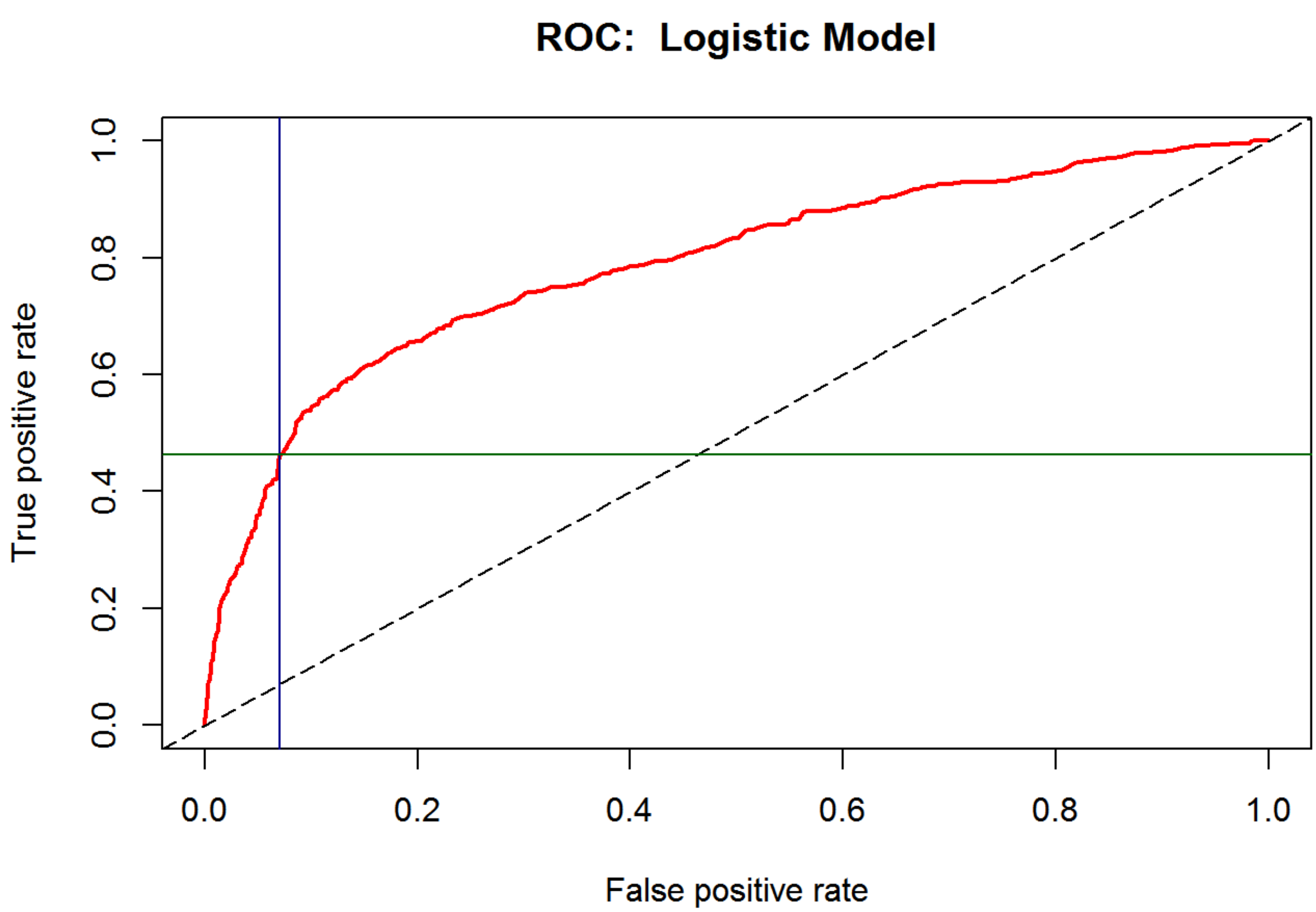
We should use a cutoff of 25.89% to optimize our model with regard to False Negatives. At this cutoff, the Logistic Model produces the following confusion matrix:

```
x = confusion.matrix(uc.val, log.pr, log.cut)[1:2,1:2] #create a confusion matrix
x
```

```
##      obs
## pred    0   1
##    0 3381 258
##    1  256 223
```

Now, let's see how using this cutoff affects our true postive and false positive rates on the ROC curve for the Logistic Model:

```
plot(log.perf,lwd=2, main="ROC:  Logistic Model",col="red")
abline(a=0,b=1,col="black",lty=5)
abline(v=log.fpr,col="dark blue",lty=1)
abline(h=log.tpr,col="dark green",lty=1)
```



<u>The Logistic Model has a True Positive Rate of 46.36%, a False Positive Rate of 7.04%, and is 87.52% accurate.</u>

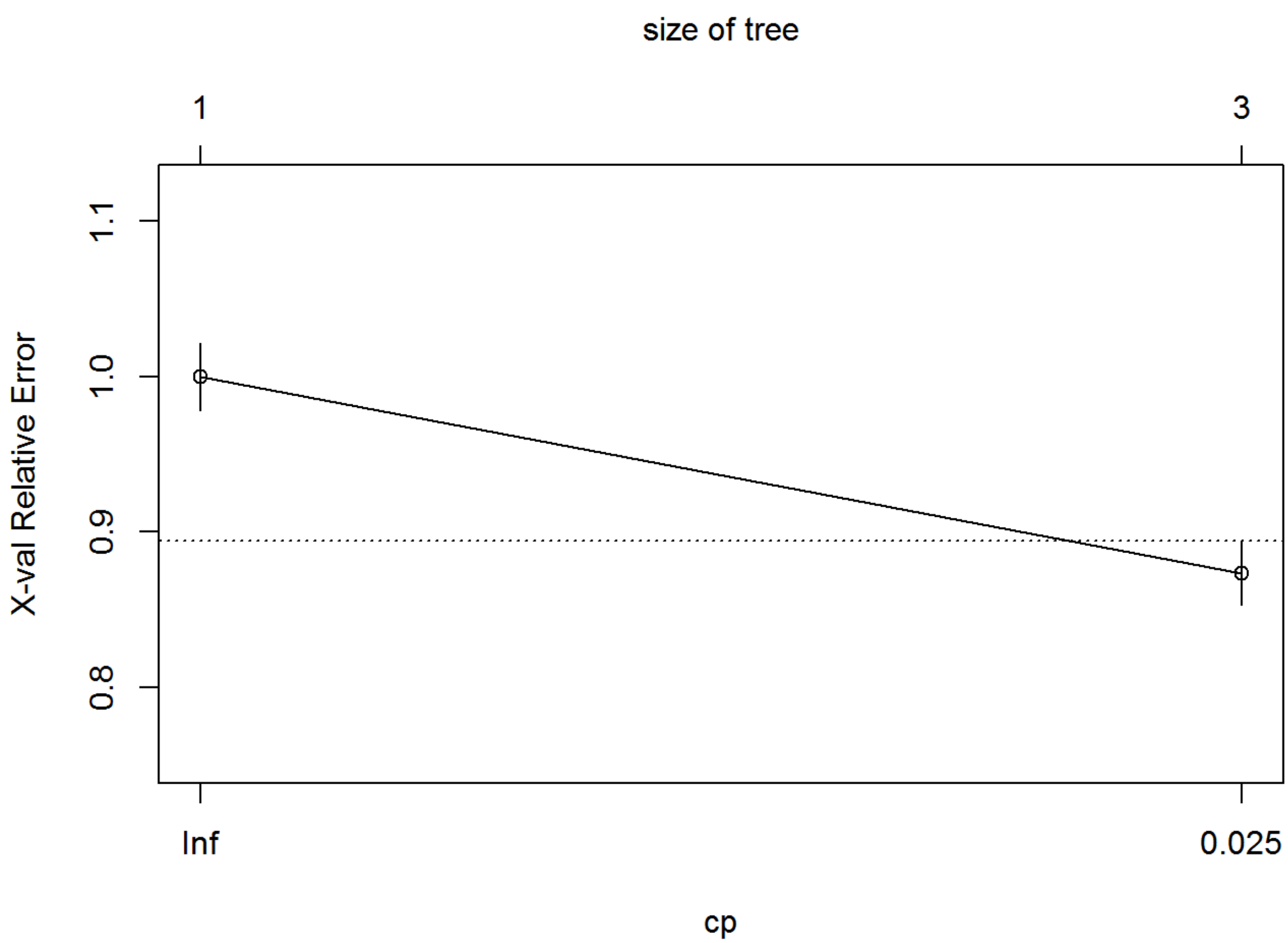
Classification Trees

Next we will build a decision tree to model the response. We start by building a decision tree using all of the predictors:

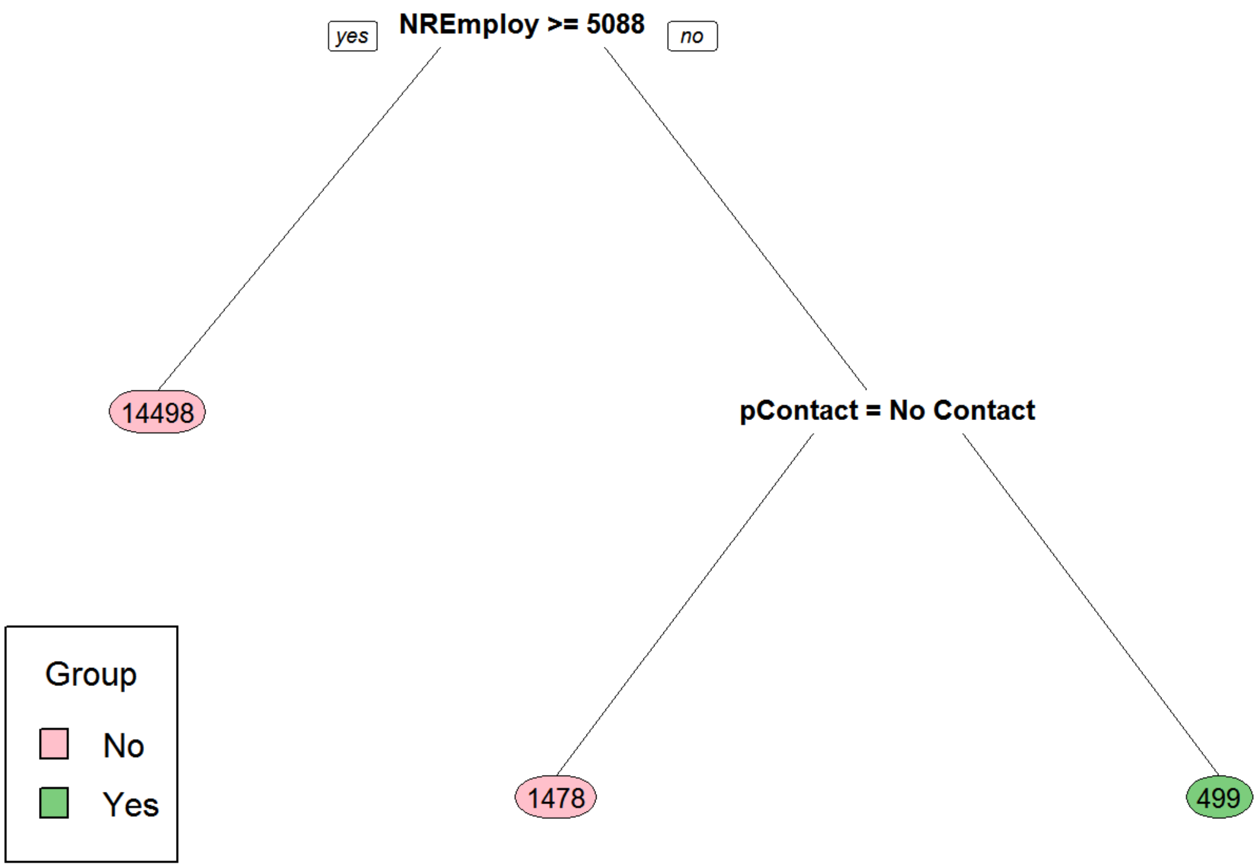
```
tree.mod = rpart(Subscribed~.-Subscribed, method = "class",data = train)
printcp(tree.mod)
```

```
##
## Classification tree:
## rpart(formula = Subscribed ~ . - Subscribed, data = train, method = "class")
##
## Variables actually used in tree construction:
## [1] NREmployed pContact
##
## Root node error: 1884/16475 = 0.11436
##
## n= 16475
##
##         CP nsplit rel error  xerror     xstd
## 1 0.063429      0   1.00000 1.00000 0.021682
## 2 0.010000      2   0.87314 0.87367 0.020430
```

```
plotcp(tree.mod)
```

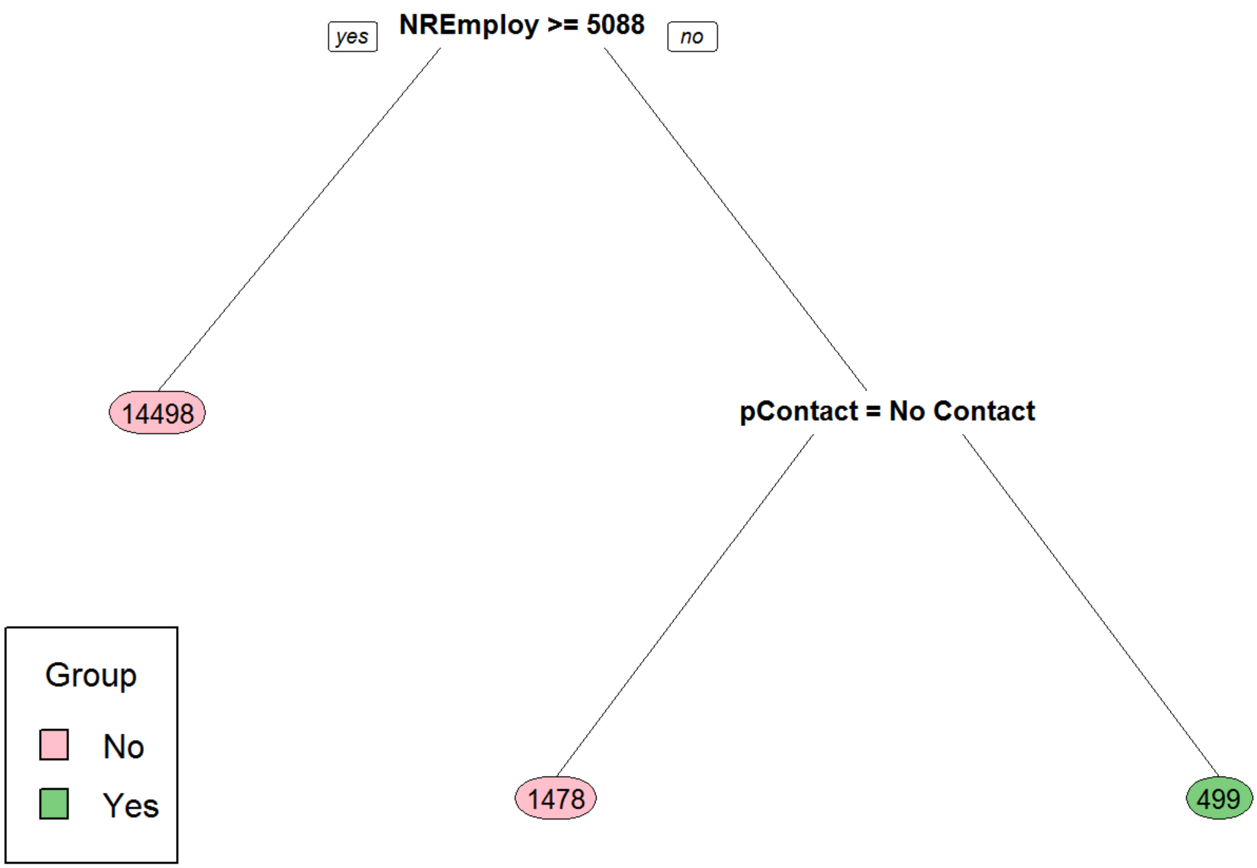


```
boxcols <- c("pink", "palegreen3")[tree.mod$frame$yval]
par(xpd=TRUE)
prp(tree.mod, faclen = 0, cex = 0.8, node.fun=only_count, box.col = boxcols)
legend("bottomleft", legend = c("No","Yes"), fill = c("pink", "palegreen3"),
       title = "Group")
```



Here we see that the Tree Model only used two variables: NREmployed and pContact. With classification trees, it is possible that we could overfit our model. In this case, our model is quite simple: if NREmployed is less than 5088 and the client was contacted previously, we predict they will subscribe. However, just to be sure we are not overfitting the data, let's prune the tree and see if there is a more simple version with just as much predictive power:

```
min.cp = tree.mod$cptable[which.min(tree.mod$cptable[,"xerror"]),"CP"]
p.tree.mod = prune(tree.mod,cp=min.cp)
boxcols = c("pink", "palegreen3")[p.tree.mod$frame$yval]
par(xpd=TRUE)
prp(p.tree.mod, faclen = 0, cex = 0.8, node.fun=only_count, box.col = boxcols)
legend("bottomleft", legend = c("No","Yes"), fill = c("pink", "palegreen3"),
       title = "Group")
```



In this case, our original tree does not need to be pruned. Now let's see which cutoff should be used for classification:

```
tree.pr = predict(p.tree.mod,newdata=val,type="prob")[,2]
tree.pred = prediction(tree.pr,val$Subscribed)
tree.perf = performance(tree.pred,"tpr","fpr")
cost.perf = performance(tree.pred, "cost", cost.fp = 1, cost.fn = 2)
tree.cut = tree.pred@cutoffs[[1]][which.min(cost.perf@y.values[[1]])]
```

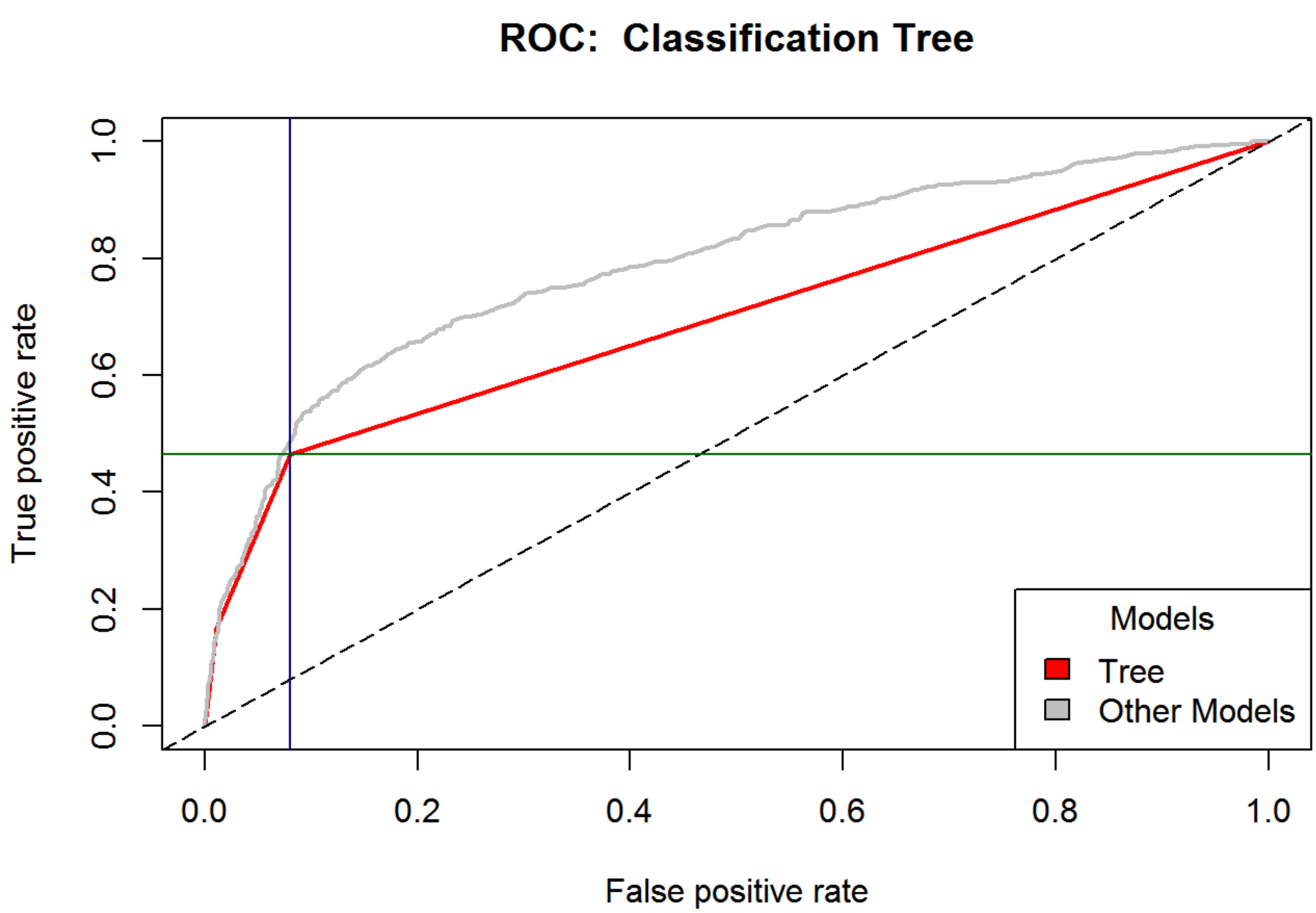
We should use a cutoff of 35.66% to optimize our Tree model with regard to False Negatives. At this cutoff, the Tree Model produces the following confusion matrix:

```
x = confusion.matrix(uc.val, tree.pr, tree.cut)[1:2,1:2]
x
```

```
##      obs
## pred    0   1
##    0 3345 257
##    1  292 224
```

Now, let’s see how using this cutoff affects our true postive and false positive rates on the ROC curve for the Tree Model:

```
plot(tree.perf,lwd=2,col="red", main="ROC:  Classification Tree")
plot(log.perf, lwd=2,col="grey",add=TRUE)
abline(a=0,b=1,col="black",lty=5)
abline(v=tree.fpr,col="dark blue",lty=1)
abline(h=tree.tpr,col="dark green",lty=1)
legend("bottomright", legend = c("Tree","Other Models")
       , fill = c("Red", "Grey")
       ,title = "Models")
```



<u>The Tree Model has a True Positive Rate of 46.57%, a False Positive Rate of 8.03%, and is 86.67% accurate.</u>

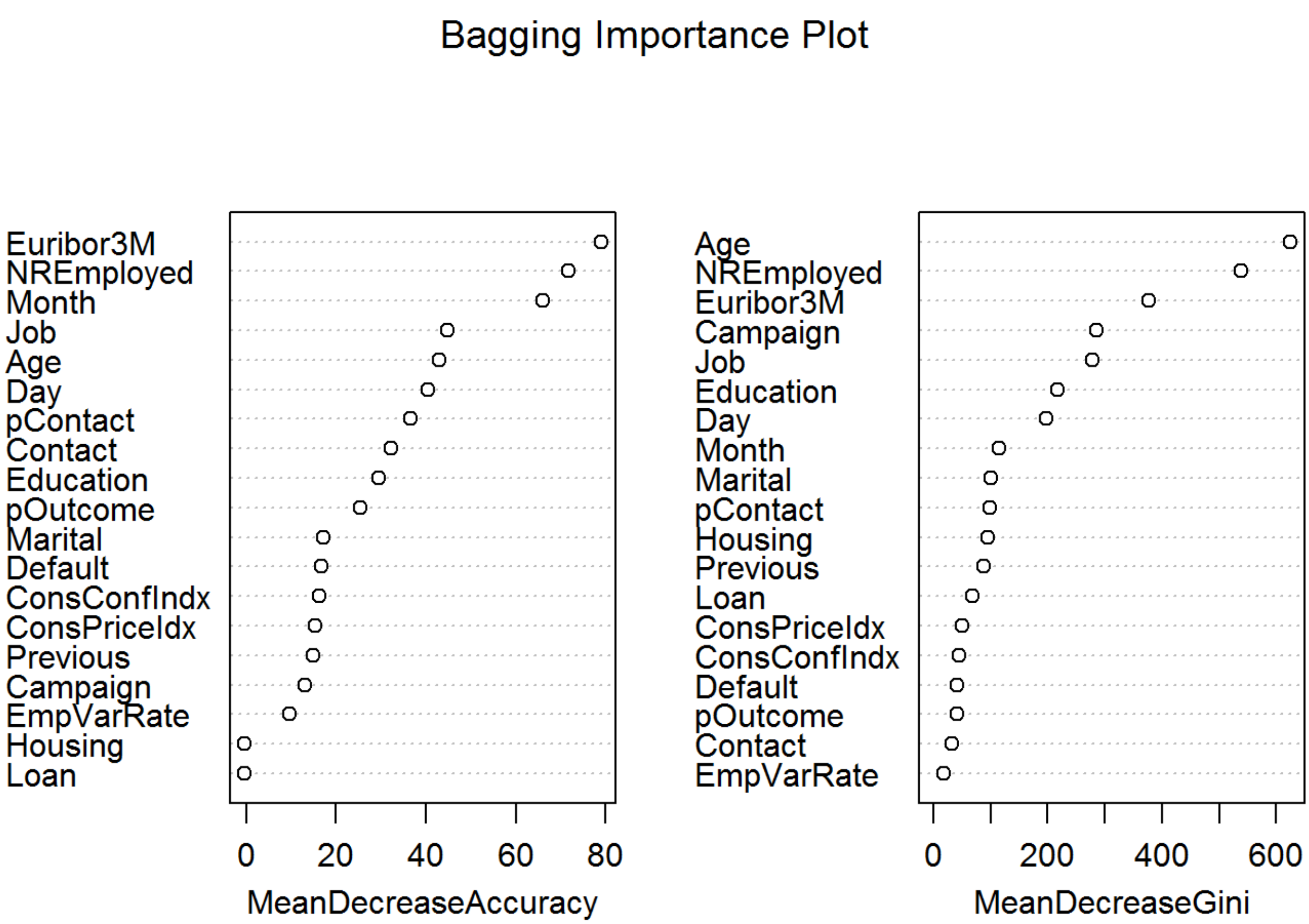
Bagging Model

With Bagging, we will use bootstrap aggregating of our data to build an ensemble of classification trees. We will let each tree be built with all the varaibles in the data:

```
p = dim(d)[2]-1 #how many variables are available?
bagging.mod = randomForest(Subscribed~.,data=train,mtry=p,importance=TRUE)
print(bagging.mod)
```

```
##
## Call:
##  randomForest(formula = Subscribed ~ ., data = train, mtry = p,      importance = TRU
E)
##                Type of random forest: classification
##                      Number of trees: 500
## No. of variables tried at each split: 19
##
##         OOB estimate of  error rate: 10.79%
## Confusion matrix:
##        no yes class.error
## no  14103 488  0.03344527
## yes  1290 594  0.68471338
```

```
varImpPlot(bagging.mod,main="Bagging Importance Plot")
```



```
importance(bagging.mod)
```

```
##                     no         yes MeanDecreaseAccuracy MeanDecreaseGini
## Age         44.4613056  -7.4869754           42.9706112        624.69337
## Job         47.2157994  -1.3211240           44.8720299        277.84432
## Marital     20.1663069  -6.5871988           17.1960001        101.17366
## Education   32.8998565  -4.1328240           29.5894906        216.97507
## Default     18.6937990  -6.2999596           16.6208131         41.83688
## Housing      0.9078339  -3.2915190           -0.3821018         95.62891
## Loan         1.5600673  -4.7585824           -0.3827360         68.54893
## Contact     27.6546386  23.1908017           32.3116559         31.71951
## Month       64.0547980   0.8832198           66.0697660        115.60983
## Day         42.9736979  -1.9809804           40.3989891        197.52650
## Campaign    13.1106432   3.0595766           13.1592638        284.57481
## Previous    17.6506649 -10.2030846           15.0057772         87.38060
## pOutcome    19.4928365  10.2385774           25.3779661         40.65697
## EmpVarRate  10.5512137  -7.9992314            9.5468836         18.74822
## ConsPriceIdx 17.6369055 -13.1197086          15.2829720         49.46836
## ConsConfIndx 16.9885939  -4.9766023          16.3840451         44.61306
## Euribor3M   75.3509250   0.3507828           79.0835979        376.01870
## NREmployed  55.2978777  56.0799319           71.7205225        538.87199
## pContact    -7.3474838  44.3780779           36.6203436         98.38339
```

Next, let’s figure out what threshold we should use:

```
bag.pr = predict(bagging.mod,val,type="prob")[,2]
bag.pred = prediction(bag.pr,val$Subscribed)
bag.perf = performance(bag.pred,"tpr","fpr")
cost.perf = performance(bag.pred, "cost", cost.fp = 1, cost.fn = 2)
bag.cut = bag.pred@cutoffs[[1]][which.min(cost.perf@y.values[[1]])]
```

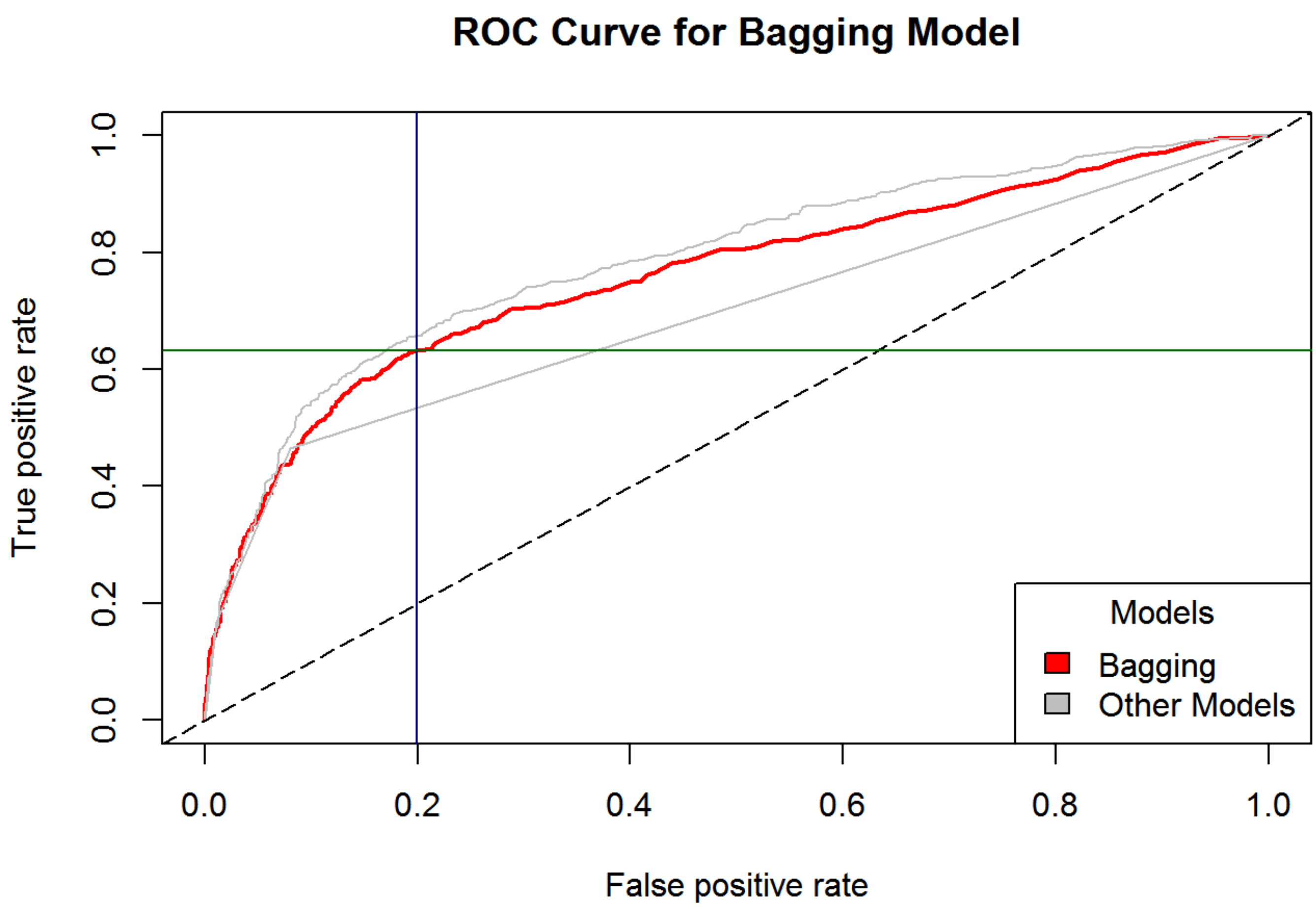
We should use a cutoff of 41.8% to optimize our Bagging model with regard to False Negatives. At this cutoff, the Bagging Model produces the following confusion matrix:

```
x = confusion.matrix(uc.val, bag.pr, bag.cut)[1:2,1:2]
x
```

```
##     obs
## pred    0   1
##    0 3432 295
##    1  205 186
```

Now, let’s see how using this cutoff affects our True Postive and False Positive rates on the ROC curve for the Bagging Model:

```
plot(bag.perf,main="ROC Curve for Bagging Model",col="red",lwd=2)
plot(log.perf,col="grey",add=TRUE)
plot(tree.perf,col="grey",add=TRUE)
abline(a=0,b=1,col="black",lty=5)
abline(v=bag.fpr,col="dark blue",lty=1)
abline(h=bag.tpr,col="dark green",lty=1)
legend("bottomright", legend = c("Bagging","Other Models")
       , fill = c("Red", "Grey")
       ,title = "Models")
```



<u>The Bagging Model has a True Positive Rate of 38.67%, a False Positive Rate of 5.64%, and is 87.86% accurate.</u>

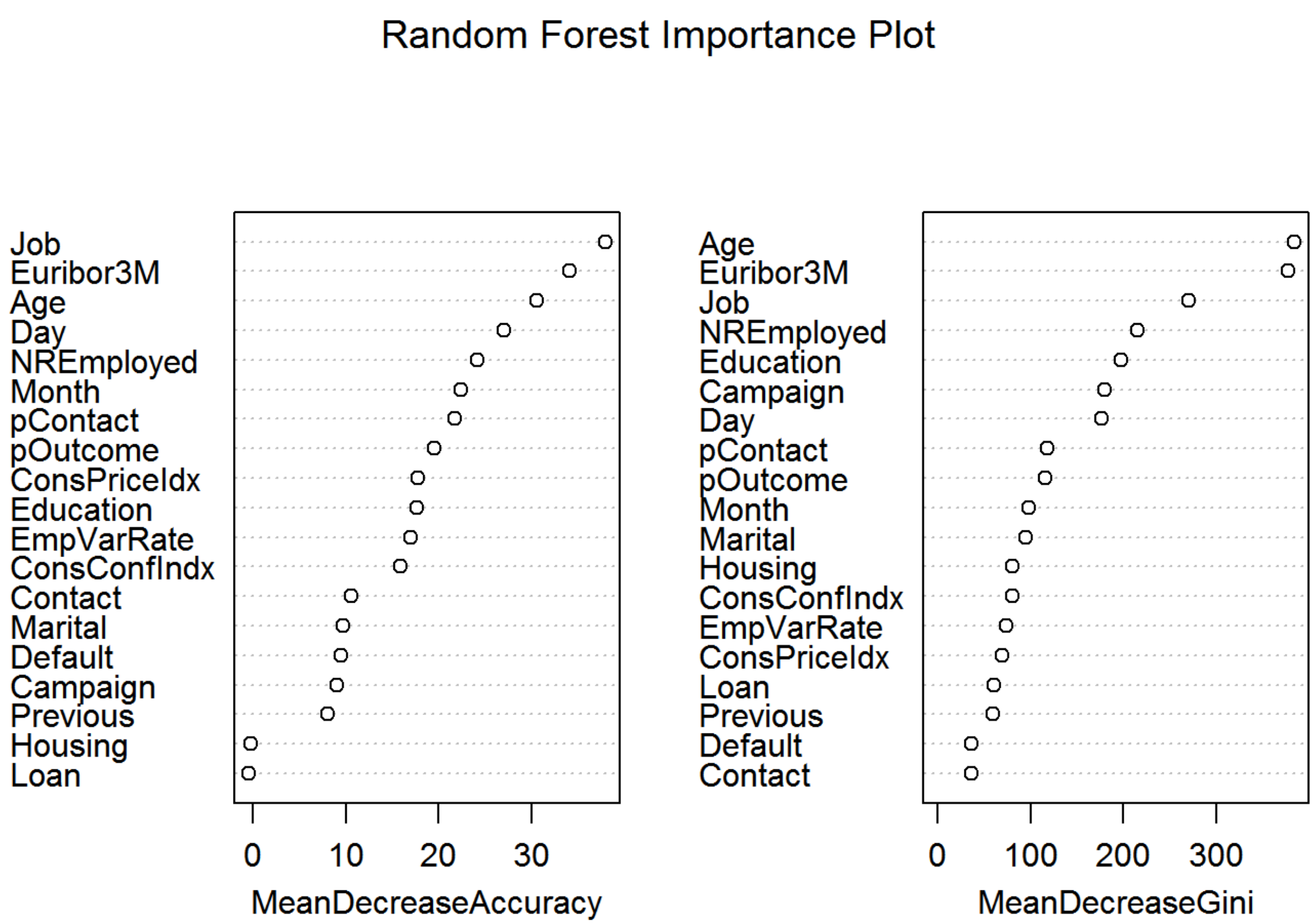
Random Forest

For a Random Forest model, we will use bootstrap aggregating to build an ensemble of classification trees but, unlike Bagging, we will not allow the model to use all the parameters at each node split. Instead, we will provide a random sample of our independent variables which will be equal to the square root of the total variables:

```
p = sqrt(p)
rf.mod = randomForest(Subscribed~.,data=train,mtry=p,importance=TRUE,type="prob")
print(rf.mod)
```

```
##
## Call:
##  randomForest(formula = Subscribed ~ ., data = train, mtry = p,      importance = TRU
E, type = "prob")
##                Type of random forest: classification
##                      Number of trees: 500
## No. of variables tried at each split: 4
##
##         OOB estimate of  error rate: 10.31%
## Confusion matrix:
##        no yes class.error
## no  14198 393  0.02693441
## yes  1306 578  0.69320594
```

The Random Forests model includes Importance Plots indicating which variables are most important to the overall ensemble model:



```
##                      no         yes MeanDecreaseAccuracy MeanDecreaseGini
## Age          31.9347070  -3.4425370           30.5726184        383.74077
## Job          39.3073235  -0.6763428           37.9714328        269.94899
## Marital      12.1076530  -4.2234375            9.7022884         94.61495
## Education    20.4649715  -4.9852913           17.6686759        196.96304
## Default       7.1678536   6.2590625            9.4688764         36.79147
## Housing       1.2100750  -3.0514433           -0.2121166         80.43401
## Loan          0.9157372  -3.2635213           -0.4564909         60.11576
## Contact       6.4669909  26.0534373           10.5284280         36.54371
## Month        21.7888004  -1.8168775           22.4272059         97.67867
## Day          27.0763199   3.6496349           27.0552475        176.00648
## Campaign      6.9706085   5.2993095            9.0402053        179.28037
## Previous      6.1856733   4.8128410            8.0091652         59.68711
## pOutcome     12.4541730  15.2385215           19.4687092        115.51932
## EmpVarRate   15.8173023   7.6727308           16.9584703         73.52494
## ConsPriceIdx 17.7051198 -10.5161032           17.7031747         69.27382
## ConsConfIndx 14.9825554  -1.9756448           15.8685825         80.12470
## Euribor3M    31.5231007   7.8432736           34.1391710        377.10101
## NREmployed   20.3605041  18.8713881           24.1179712        215.20640
## pContact      4.8608989  29.1260540           21.6722250        118.09618
```

Now let’s figure out the optimal cutoff for the Random Forest Model:

```
rf.pr = predict(rf.mod,val,type="prob")[,2]
rf.pred = prediction(rf.pr,val$Subscribed)
rf.perf = performance(rf.pred,"tpr","fpr")
cost.perf = performance(rf.pred, "cost", cost.fp = 1, cost.fn = 2)
rf.cut = rf.pred@cutoffs[[1]][which.min(cost.perf@y.values[[1]])]
```

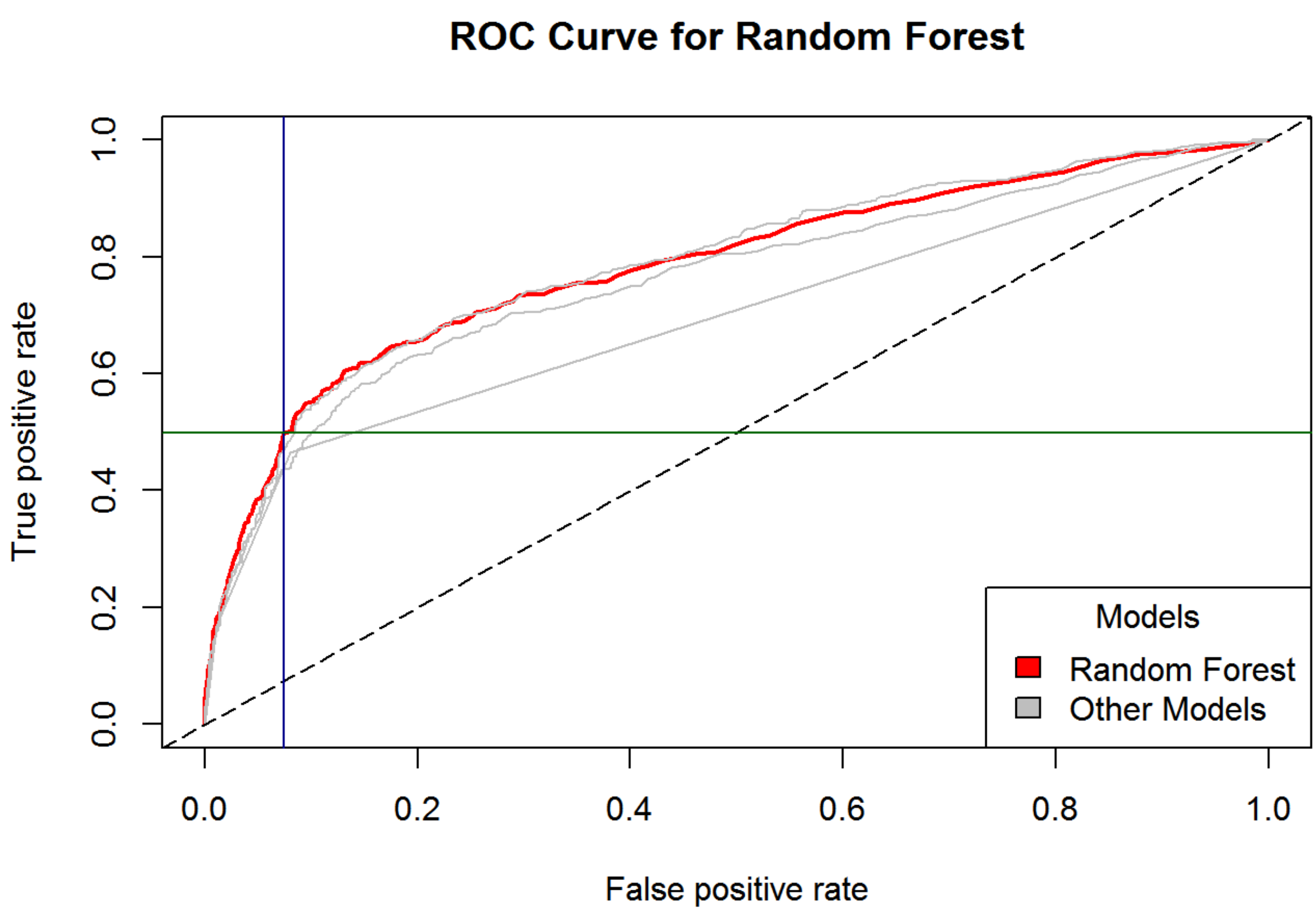
We should use a cutoff of 27.8% to optimize our Random Forest model with regard to False Negatives. At this cutoff, the Random Forest Model produces the following confusion matrix:

```
x = confusion.matrix(uc.val, rf.pr, rf.cut)[1:2,1:2]
x
```

```
##     obs
## pred    0   1
##    0 3366 241
##    1  271 240
```

Now, let’s see how using this cutoff affects our true postive and false positive rates on the ROC curve for the Random Forest Model:

```
plot(rf.perf,main="ROC Curve for Random Forest",col="red",lwd=2)
plot(log.perf,col="grey",add=TRUE)
plot(tree.perf,col="grey",add=TRUE)
plot(bag.perf,col="grey",add=TRUE)
abline(a=0,b=1,col="black",lty=5)
abline(v=rf.fpr,col="dark blue",lty=1)
abline(h=rf.tpr,col="dark green",lty=1)
legend("bottomright", legend = c("Random Forest","Other Models")
       , fill = c("Red", "Grey")
       ,title = "Models")
```



The Random Forest Model has a True Positive Rate of 49.9%, a False Positive Rate of 7.45%, and is 87.57% accurate.

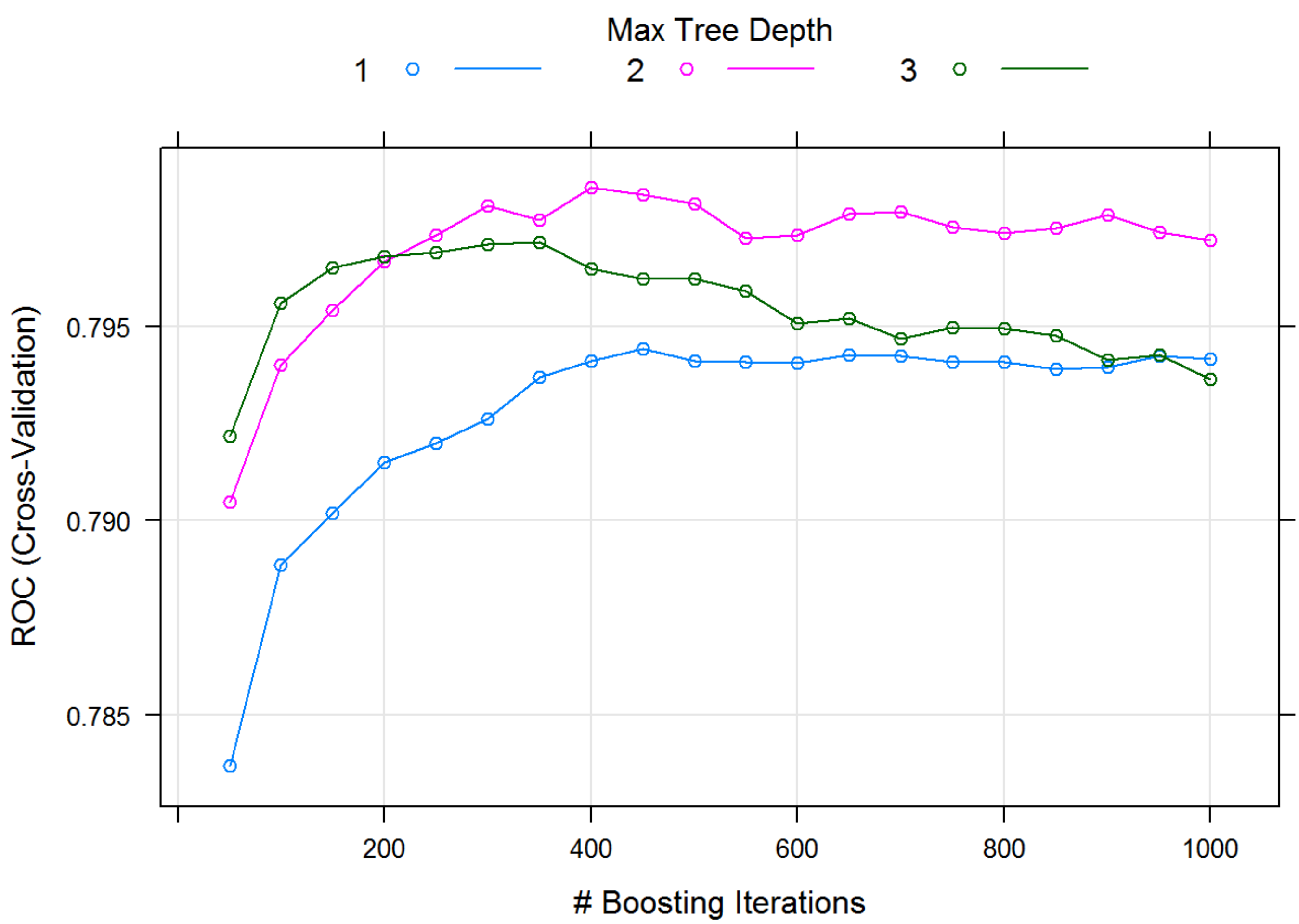
Gradient Boosting

Gradient Boosing is another tree based method that uses ensemble aggregating. However, unlike Random Forests, the model can be easily overfit. There are three parameters that must be tuned to optimize performance: 1) Tree Depth 2) Number of Trees to Build and 3) Shrinkage. Here we use the caret package to tune the three parameters using 5-fold cross-validation:

```
fitControl = trainControl(method='CV', #use cross validation
                          number=5, #set the number of folds
                          summaryFunction = twoClassSummary, #use two-class classificati
on
                          classProbs = TRUE) #return probabilities
gbmGrid =  expand.grid(interaction.depth=c(1,2,3), #which tree depth values to try
                       n.trees = (1:20)*50, #how many values of n.trees to try
                       shrinkage = 0.1, #what shrinkage value to try
                       n.minobsinnode = 20)
gbmFit = train(Subscribed ~ .,data = train,method='gbm',trControl=fitControl
                ,metric="ROC",tuneGrid=gbmGrid,verbose=FALSE)
gbmFit$bestTune
```

```
##    n.trees interaction.depth shrinkage n.minobsinnode
## 28     400                 2       0.1             20
```

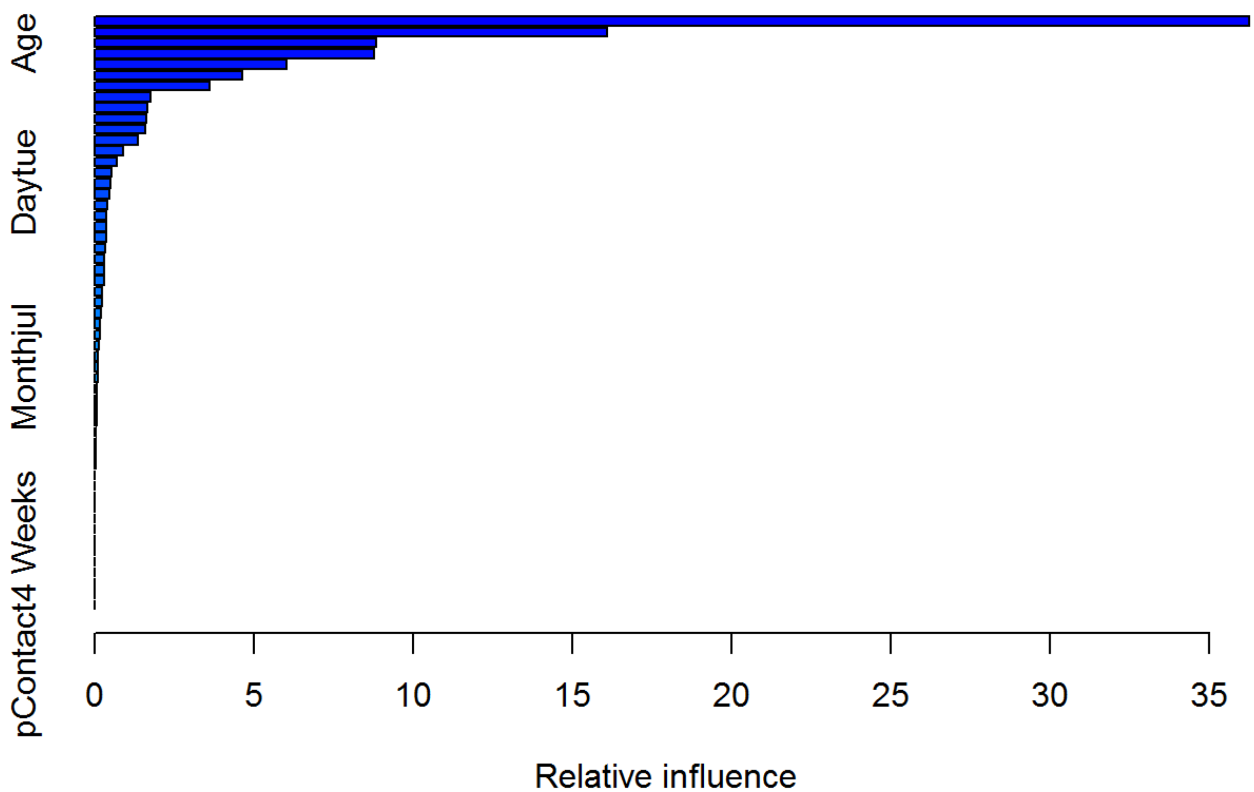
```
plot(gbmFit)
```



```
r = which.max(gbmFit$results[,"ROC"])#which combo had the best ROC?
ntrees = gbmFit$results[r,"n.trees"] #what was the # of trees?
depth = gbmFit$results[r,"interaction.depth"] #how deep were the trees?
shrink = gbmFit$results[r,"shrinkage"] #what was the shrinkage?
```

We see that the optimal Boosting Model includes 400 trees, each built 2 nodes deep with a shinkage parameter of 0.1. With Boosting, we can also look at which variables had the most significant influence in the model. Here are the top 10 variables:

```
summary(gbmFit)[1:10,]
```



```
##                                   var   rel.inf
## NREmployed                 NREmployed 36.263746
## Euribor3M                   Euribor3M 16.088964
## Age                               Age  8.854626
## pContactNo Contact pContactNo Contact  8.792334
## ConsConfIndx             ConsConfIndx  6.030205
## pOutcomesuccess       pOutcomesuccess  4.634868
## ConsPriceIdx             ConsPriceIdx  3.602976
## EmpVarRate                 EmpVarRate  1.770673
## Previous                     Previous  1.649505
## Monthoct                     Monthoct  1.613087
```

Now that we have tuned the Boosting parameters, we can build the actual model:

```
gmb.model = gbm(Subscribed2~., data=train2,n.trees=ntrees,interaction.depth =depth,shrink
age=shrink
                ,distribution = "bernoulli")
```

Let's see what cutoff should be used:

```
boost.probs = predict(gmb.model, val2, n.trees=ntrees,interaction.depth=depth,shrinkage=s
hrink
                      , type="response")
gbm.pred = prediction(boost.probs,val$Subscribed)
gbm.perf = performance(gbm.pred,"tpr","fpr")
cost.perf = performance(gbm.pred, "cost", cost.fp = 1, cost.fn = 2)
gbm.cut = gbm.pred@cutoffs[[1]][which.min(cost.perf@y.values[[1]])]
```

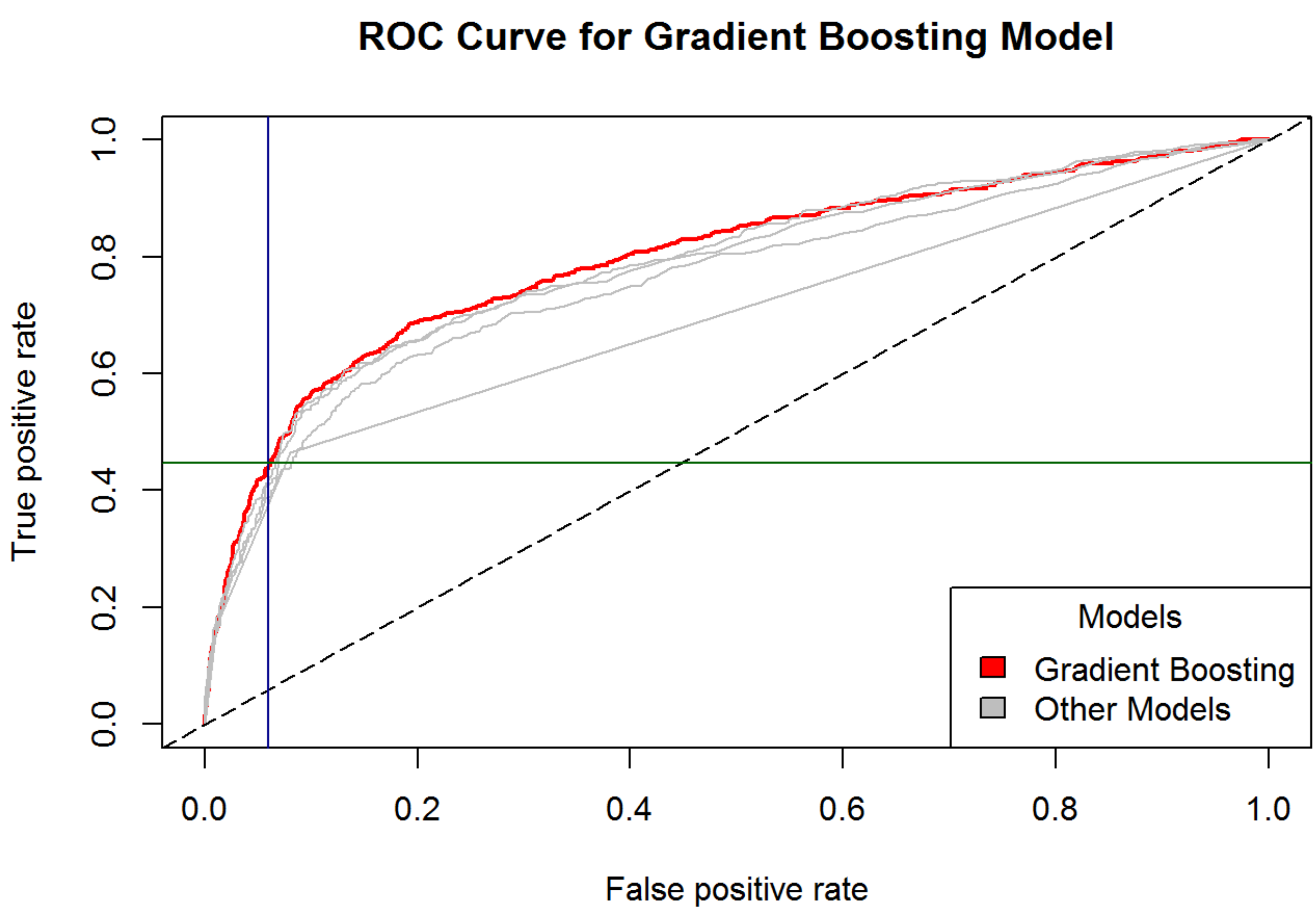
We should use a cutoff of 34.41% to optimize our Boosting model with regard to False Negatives. At this cutoff, the Boosting Model produces the following confusion matrix:

```
x = confusion.matrix(uc.val, boost.probs, gbm.cut)[1:2,1:2]
x
```

```
##     obs
## pred    0   1
##    0 3419 266
##    1  218 215
```

Now, let's see how using this cutoff affects our true postive and false positive rates on the ROC curve for the Boosting Model:

```
plot(gbm.perf,main="ROC Curve for Gradient Boosting Model",col="red",lwd=2)
plot(log.perf,col="grey",add=TRUE)
plot(tree.perf,col="grey",add=TRUE)
plot(bag.perf,col="grey",add=TRUE)
plot(rf.perf,col="grey",add=TRUE)
abline(a=0,b=1,col="black",lty=5)
abline(v=gbm.fpr,col="dark blue",lty=1)
abline(h=gbm.tpr,col="dark green",lty=1)
legend("bottomright", legend = c("Gradient Boosting","Other Models")
       , fill = c("Red", "Grey")
       ,title = "Models")
```

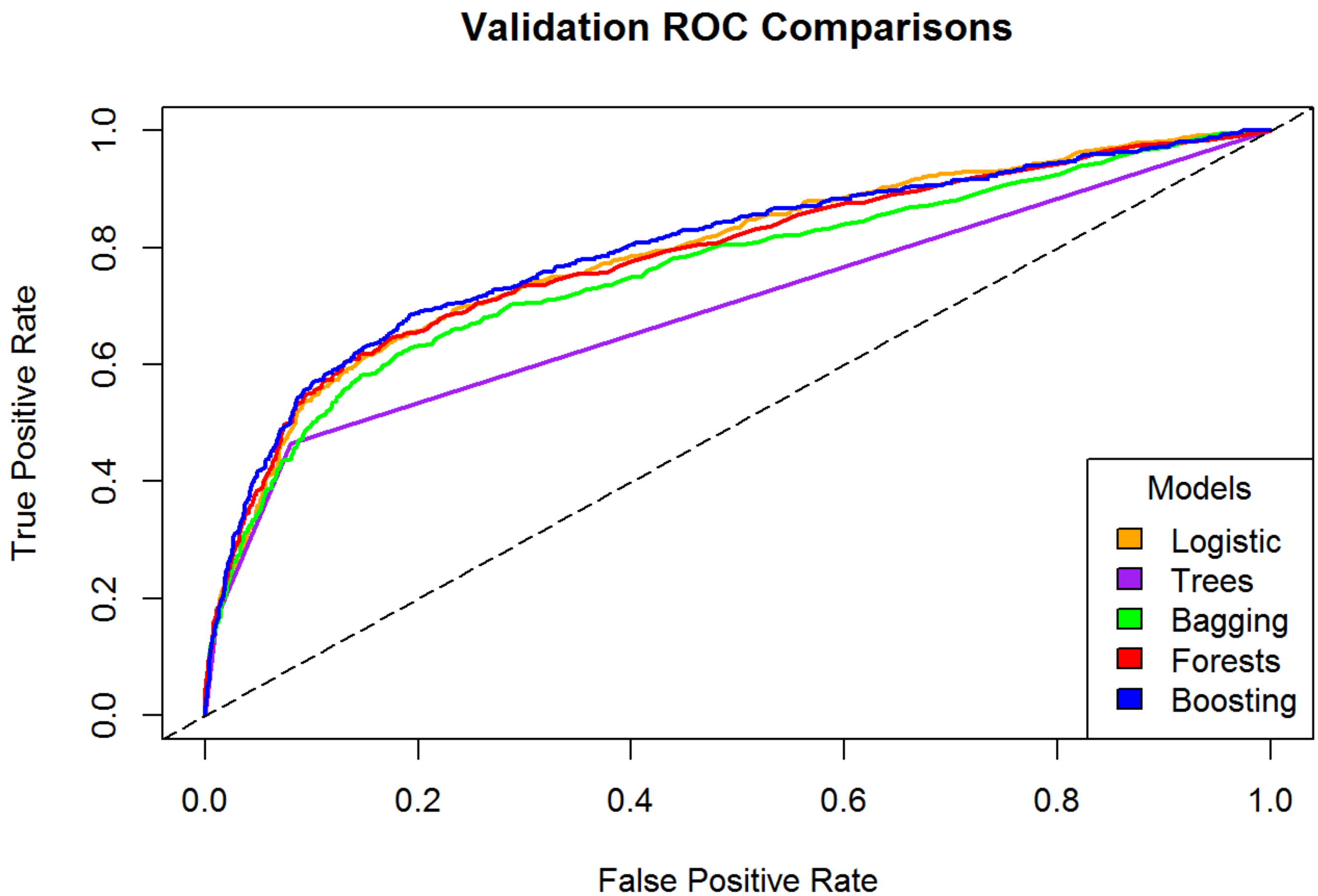


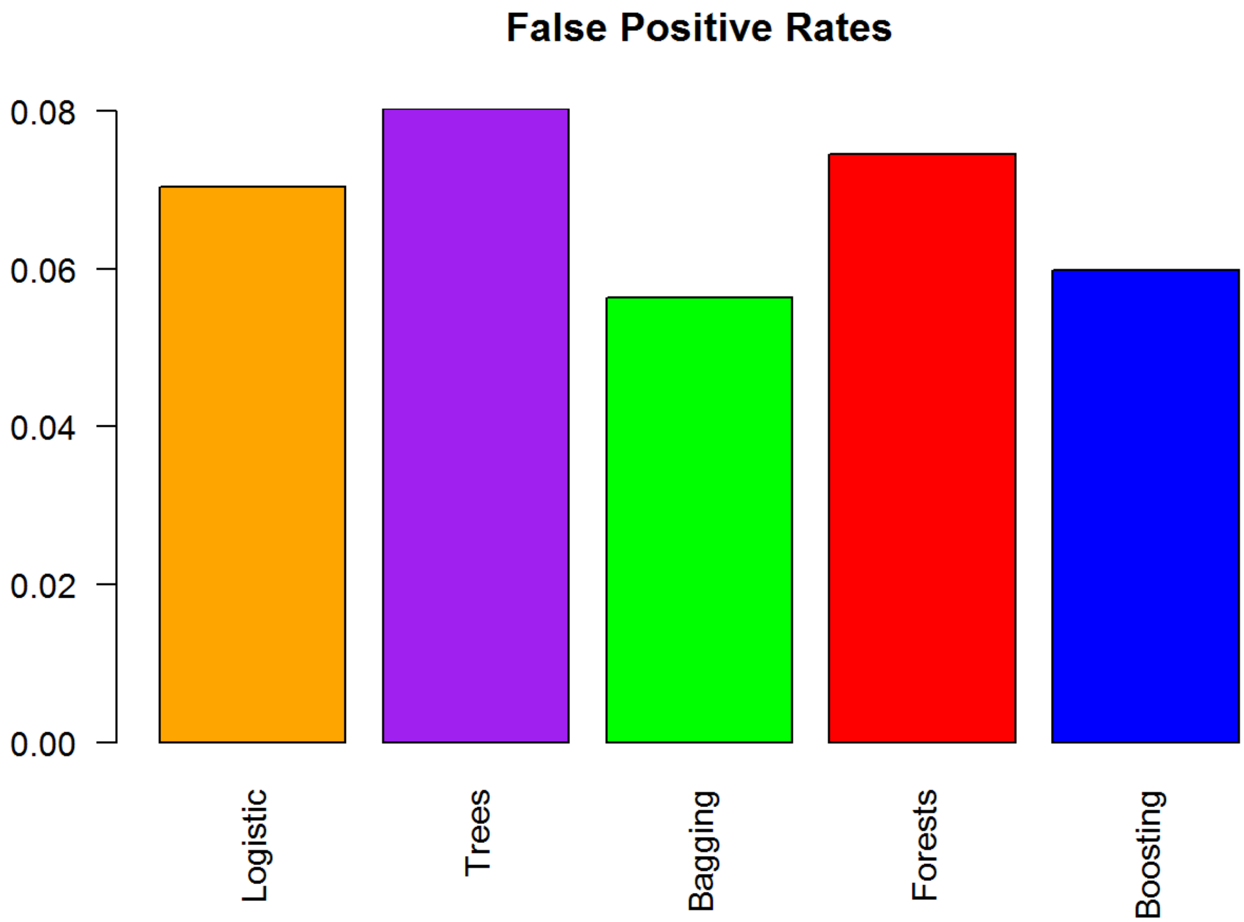
The Boosting Model has a True Positive Rate of 44.7%, a False Positive Rate of 5.99%, and is 88.25% accurate.

Model Comparision Summary

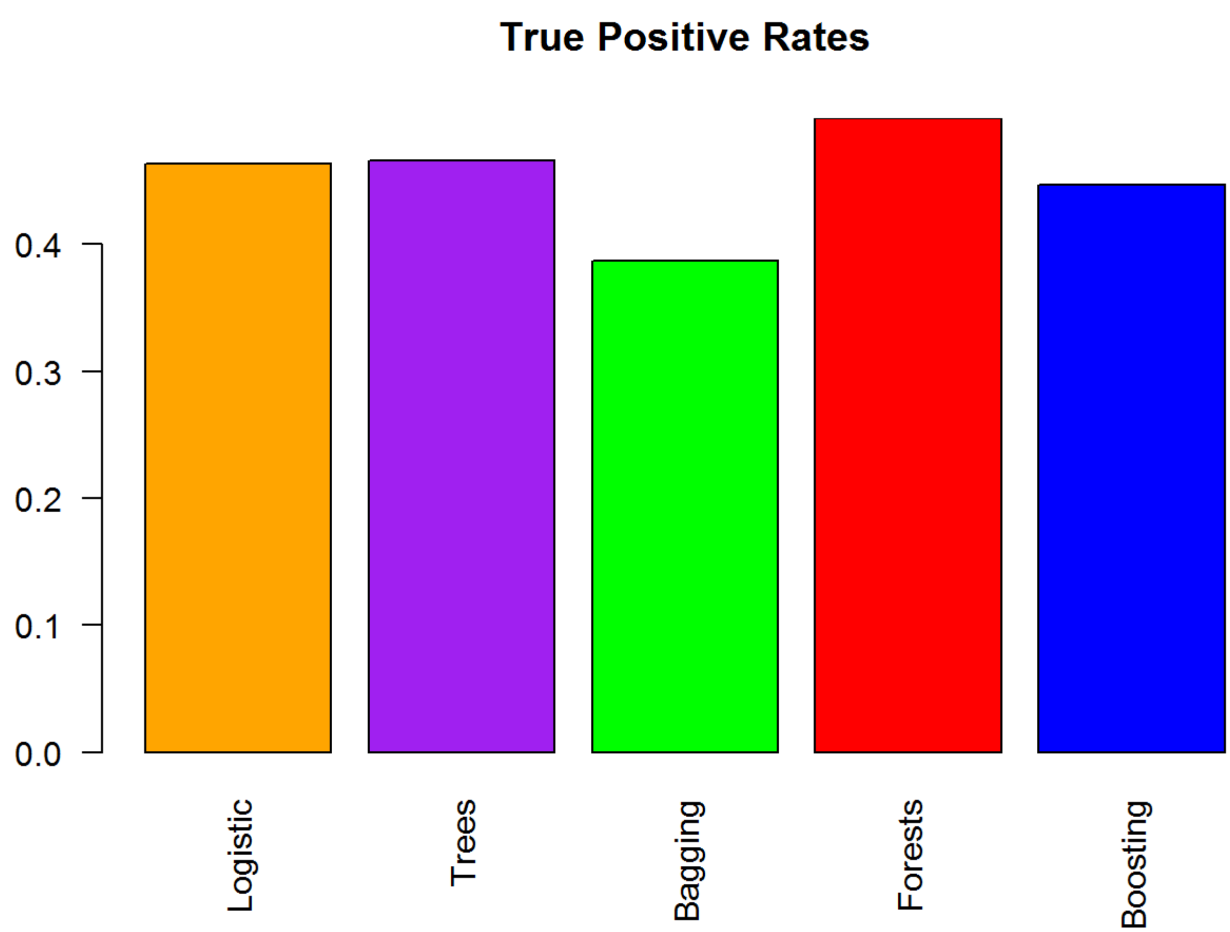
Top 5 Variables from Each Model:

```
##      log.vars  tree.vars   bag.vars     rf.vars      gbm.vars
## 1 NREmployed NREmployed         Age         Age    NREmployed
## 2   pOutcome   pContact NREmployed   Euribor3M     Euribor3M
## 3      Month         NA   Euribor3M         Job           Age
## 4    Contact         NA    Campaign NREmployed      pContact
## 5        Day         NA         Job   Education ConsConfIndx
```

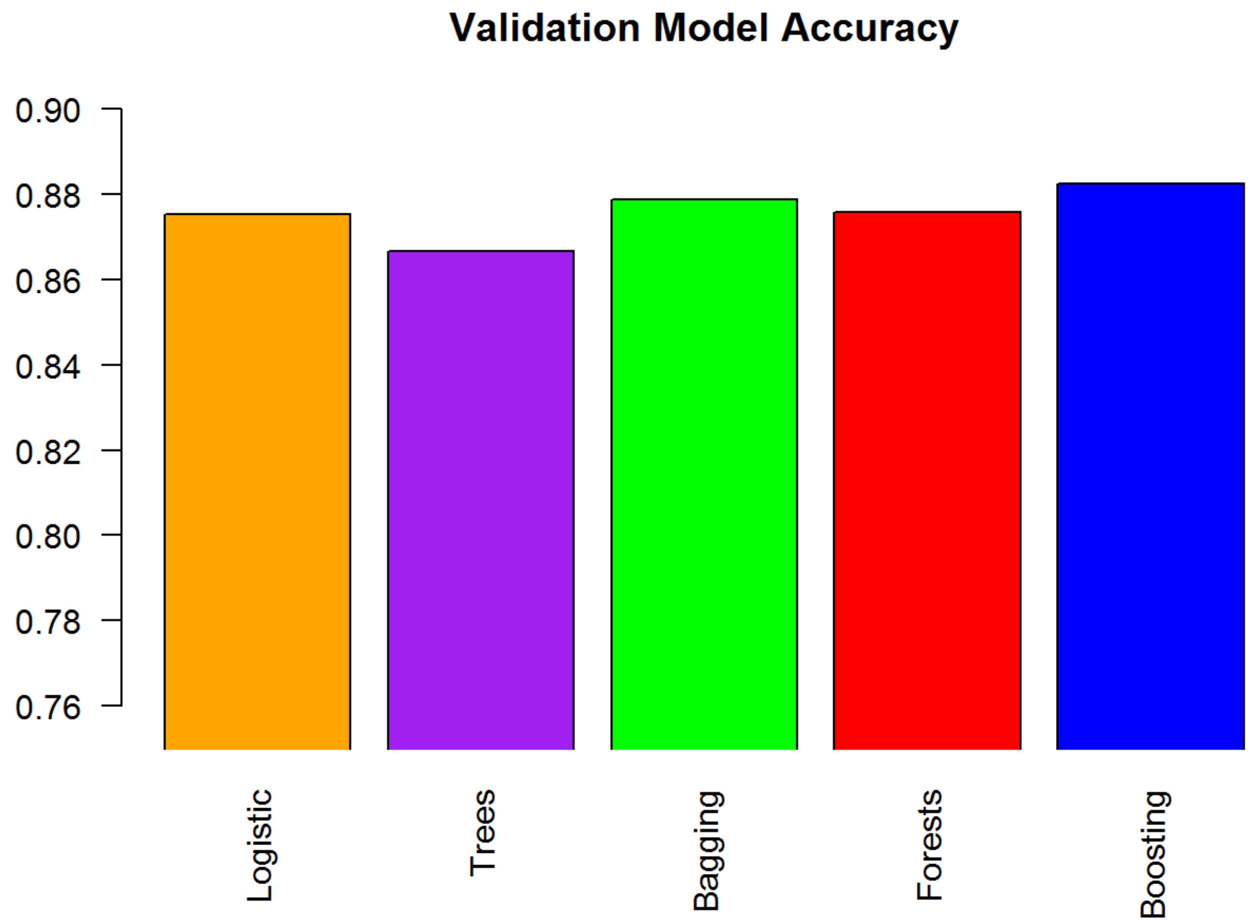





```
##        mod      a
## 1 Logistic 0.0704
## 2    Trees 0.0803
## 3  Bagging 0.0564
## 4  Forests 0.0745
## 5 Boosting 0.0599
```




```
##        mod      a
## 1 Logistic 0.4636
## 2    Trees 0.4657
## 3  Bagging 0.3867
## 4  Forests  0.499
## 5 Boosting  0.447
```




```
##        mod      a
## 1 Logistic 0.8752
## 2    Trees 0.8667
## 3  Bagging 0.8786
## 4  Forests 0.8757
## 5 Boosting 0.8825
```

Expectations for Test Data

From these results, we expect that the Gradient Boosting Model will perform best: it has the lowest False Postive Rate, has a True Postive rate comparable to most other models, and has the highest accuracy. The next best model might be Bagging for the same reasons.

Model Testing

Logistic:

```
log.test =  predict(log.mod,test,type="response")
x = confusion.matrix(uc.test, log.test, log.cut)[1:2,1:2] #create a confusion matrix
x
```

```
##     obs
## pred     0    1
##    0 17121 1194
##    1  1199 1081
```

Classification Trees:

```
tree.test = predict(p.tree.mod,newdata=test,type="prob")[,2]
x = confusion.matrix(uc.test, tree.test, tree.cut)[1:2,1:2]
x
```

```
##     obs
## pred     0    1
##    0 16938 1186
##    1  1382 1089
```

Bagging:

```
bag.test = predict(bagging.mod,test,type="prob")[,2]
x = confusion.matrix(uc.test,bag.test, bag.cut)[1:2,1:2]
x
```

```
##     obs
## pred     0    1
##    0 15642 1369
##    1  2678  906
```

```
bag.t.tpr = round(x[2,2]/(x[1,2] + x[2,2]),4)
bag.t.fpr = round(x[2,1]/(x[2,1] + x[1,1]),4)
bag.t.accuracy = round((x[1,1] + x[2,2])/(x[1,1] + x[1,2] + x[2,1] + x[2,2]),4)
bag.t.pred = prediction(bag.test,test$Subscribed)
bag.t.perf = performance(bag.t.pred,"tpr","fpr")
```

Random Forests:

```
rf.test = predict(rf.mod,test,type="prob")[,2]
x = confusion.matrix(uc.test, rf.test, rf.cut)[1:2,1:2]
x
```

```
##     obs
## pred     0    1
##    0 16753 1104
##    1  1567 1171
```

```
rf.t.tpr = round(x[2,2]/(x[1,2] + x[2,2]),4)
rf.t.fpr = round(x[2,1]/(x[2,1] + x[1,1]),4)
rf.t.accuracy = round((x[1,1] + x[2,2])/(x[1,1] + x[1,2] + x[2,1] + x[2,2]),4)
rf.t.pred = prediction(rf.test,test$Subscribed)
rf.t.perf = performance(rf.t.pred,"tpr","fpr")
```

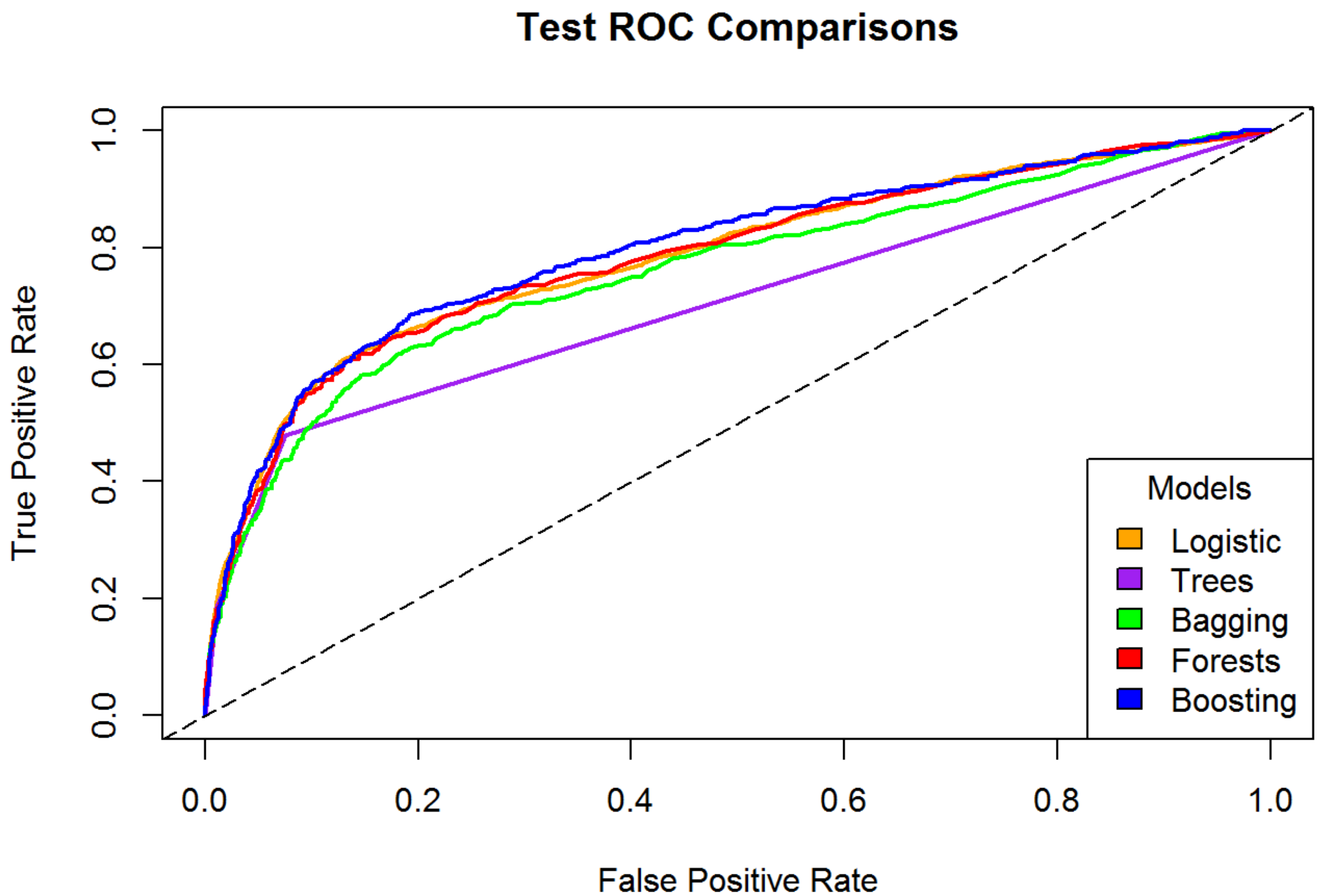
Boosting:

```
boost.test = predict(gmb.model, test2, n.trees=ntrees,interaction.depth=depth,shrinkage=s
hrink
                      , type="response")
x = confusion.matrix(test2$Subscribed2, boost.test,gbm.cut)[1:2,1:2]
x
```

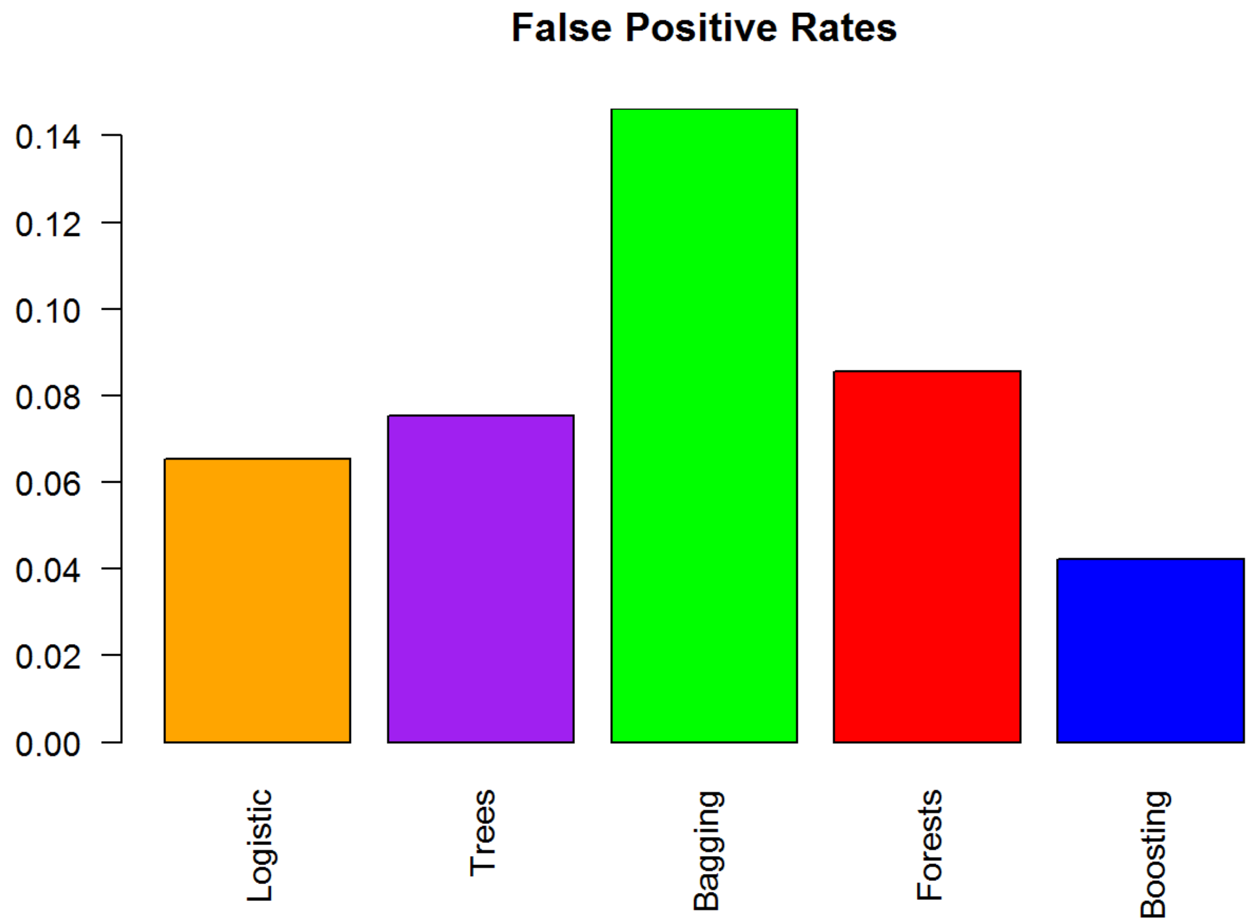
```
##     obs
## pred     0    1
##    0 17546 1389
##    1   774  886
```

```
gbm.t.tpr = round(x[2,2]/(x[1,2] + x[2,2]),4)
gbm.t.fpr = round(x[2,1]/(x[2,1] + x[1,1]),4)
gbm.t.accuracy = round((x[1,1] + x[2,2])/(x[1,1] + x[1,2] + x[2,1] + x[2,2]),4)
gbm.t.pred = prediction(boost.test,test$Subscribed)
gbm.t.perf = performance(gbm.t.pred,"tpr","fpr")
```

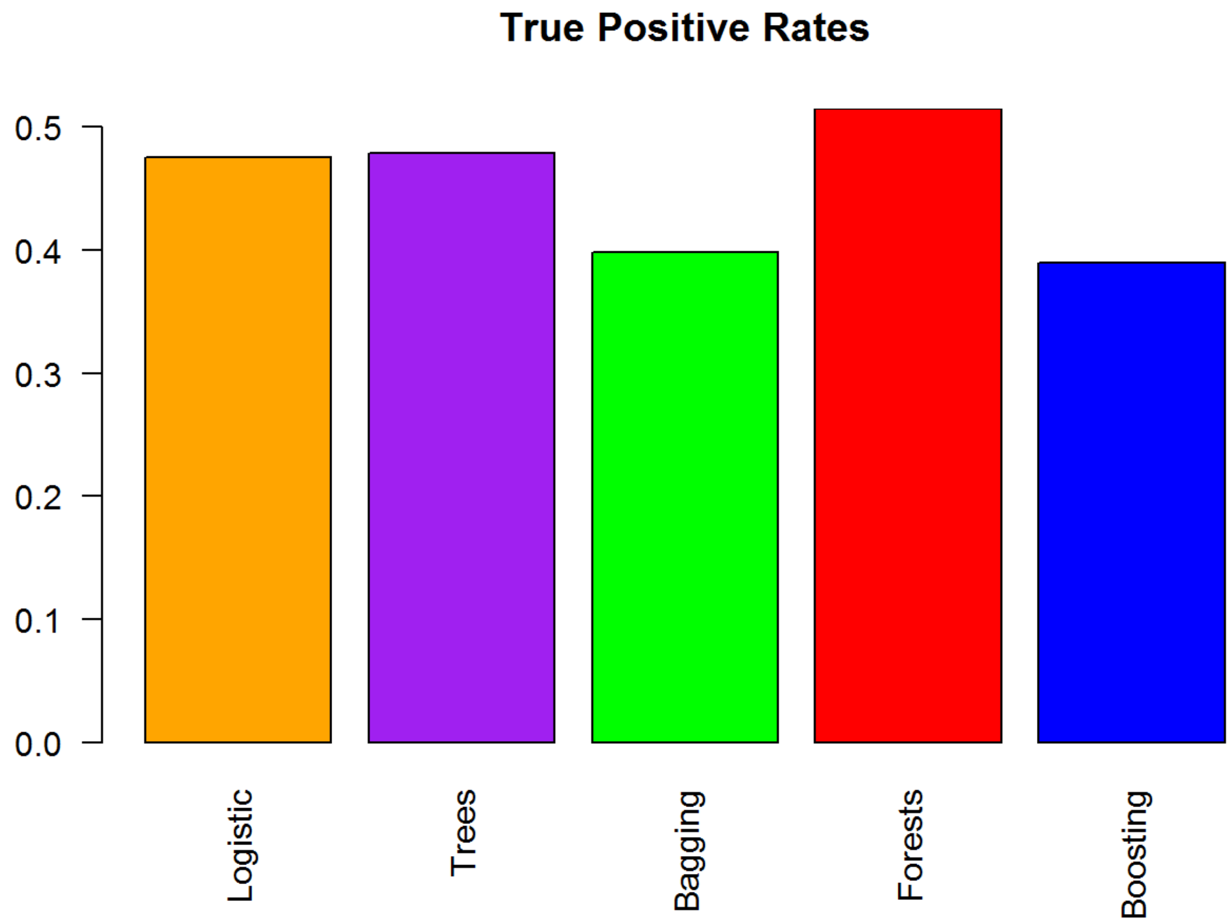
Model Comparision Summary



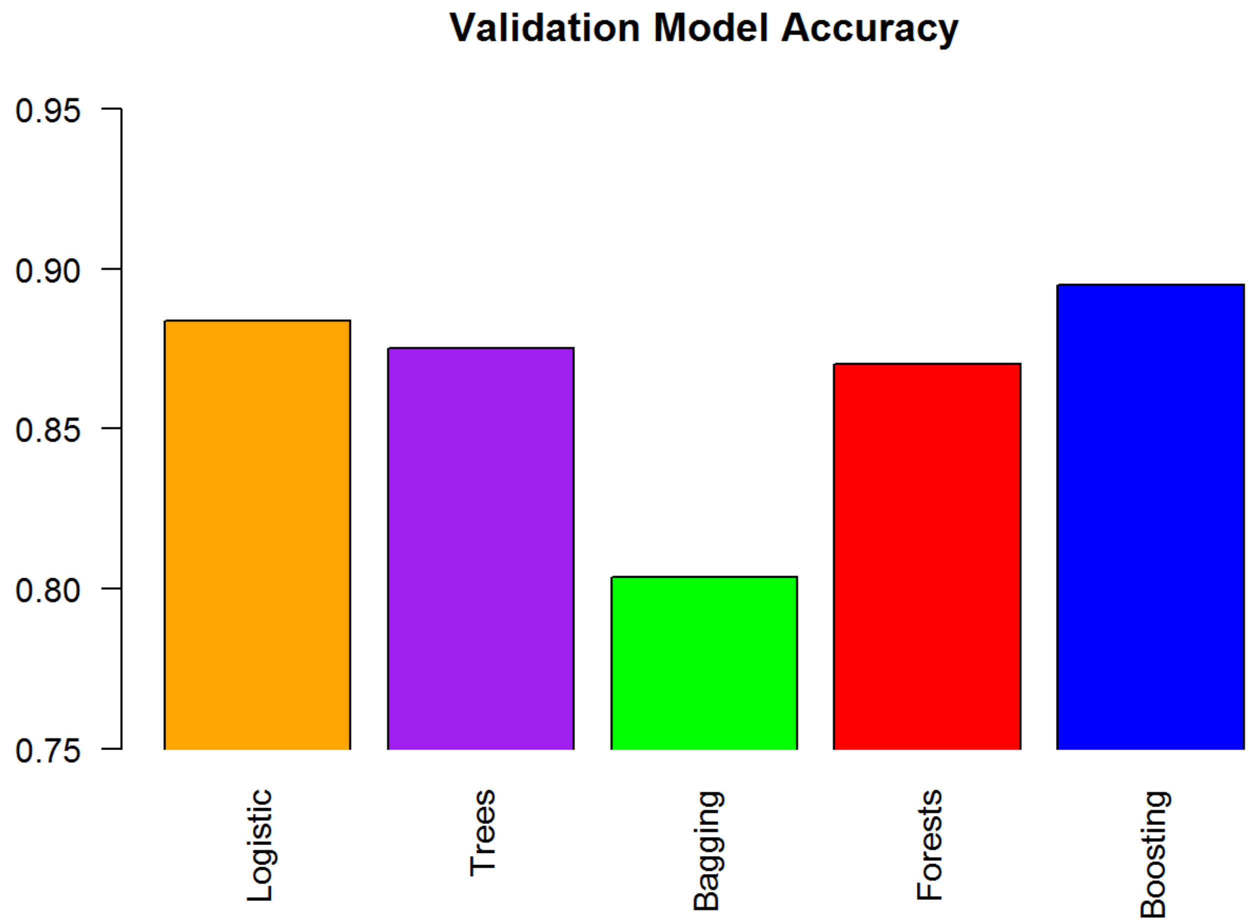
Test ROC Comparisons
True Positive Rate
0.0
0.2
0.4
0.6
0.8
1.0
False Positive Rate
0.0
0.2
0.4
0.6
0.8
1.0
Models
Logistic
Trees
Bagging
Forests
Boosting




```
##        mod      a
## 1 Logistic 0.0654
## 2    Trees 0.0754
## 3  Bagging 0.1462
## 4  Forests 0.0855
## 5 Boosting 0.0422
```




```
##        mod      a
## 1 Logistic 0.4752
## 2    Trees 0.4787
## 3  Bagging 0.3982
## 4  Forests 0.5147
## 5 Boosting 0.3895
```




```
##        mod      a
## 1 Logistic 0.8838
## 2    Trees 0.8753
## 3  Bagging 0.8035
## 4  Forests 0.8703
## 5 Boosting  0.895
```

Results

As we anticipated, the Gradient Boosting Model had the best performance on the Test Data: it had the lowest False Positive Rate and highest accuracy. The fact that it had the lowest True Positive Rate is not concerning as our assumptions guided us to mimimize the False Positives. Interestingly, the Bagging model did not perform nearly as well as anticipated. Its False Positive Rate was the highest and its accuracy was the lowest. Perhaps the model was overfit, which is always a conern with Bagging. Instead, the Logistic Model was second best: it had the second lowest False Positive Rate, almost tied for second highest True Positive Rate, and the second highest Accuracy.

Conclusion

We have developed several classification models to predict a binary response. Each of the models we used have particular aspects that must be tuned to ensure the model performs well on unseen data. Even when these models are properly tuned, they must also be adapted to the inherient costs and tradeoffs of False Positives and False Negatives. In this case, we choose to minimize False Positives and tuned our models accordingly.