Alzheimer's Disease Prediction – A Classification Problem

Matthew Warren

Data Set

The data set[1] used in this analysis is a cleaned version of data from the Alzheimer's disease Prediction of Longitudinal Evolution (TADPOLE) Challenge and the Alzheimer's disease neuroimaging initiative (ADNI)[2]. TADPOLE provides a list of individuals at an age that puts them at risk of AD. A history of informative measurements (from imaging, psychology, demographics, genetics, etc.) from each individual is available.

The data contains many attributes, but there are several that we will focus on. The primary attributes to note are the classification of cognitive status (SMC and CN = cognitively normal, (E/L) MCI = mild cognitive impairment, AD = probable Alzheimer's Disease), scores on three different neuropsychological tests administered by a clinical expert (CDR Sum of Boxes (cdrsb_bl), ADAS13 (adas13_bl), and MMSE (mmse_bl)). We also have several measures of brain structural integrity which includes measures of volume and thickness: ventricles_bl, wholebrain_bl, icv_bl, x_hippocampus_l, x_hippocampus_r, x_entorhinal_l, x_entorhinal_r, x_entorhinal_l_thick, x_entorhinal_r_thick, x_hippocampus_l_bl, x_hippocampus_r_bl, x_entorhinal_l_bl, x_entorhinal_r_bl, x_entorhinal_l_thick_bl, x_entorhinal_r_thick_bl. And an attribute named apoe4 that gives the number of copies of the apoe4 gene. Full details on the attributes can be found on the TADPOLE site[3].

Read the Data

```
data <- fread("alz_data.csv")

#Function for use later

normalize <- function(x) {
return ((x - min(x)) / (max(x) - min(x))) }
```

[1] https://github.com/A2ed/springboard_data_science/tree/master/capstone_1_mri

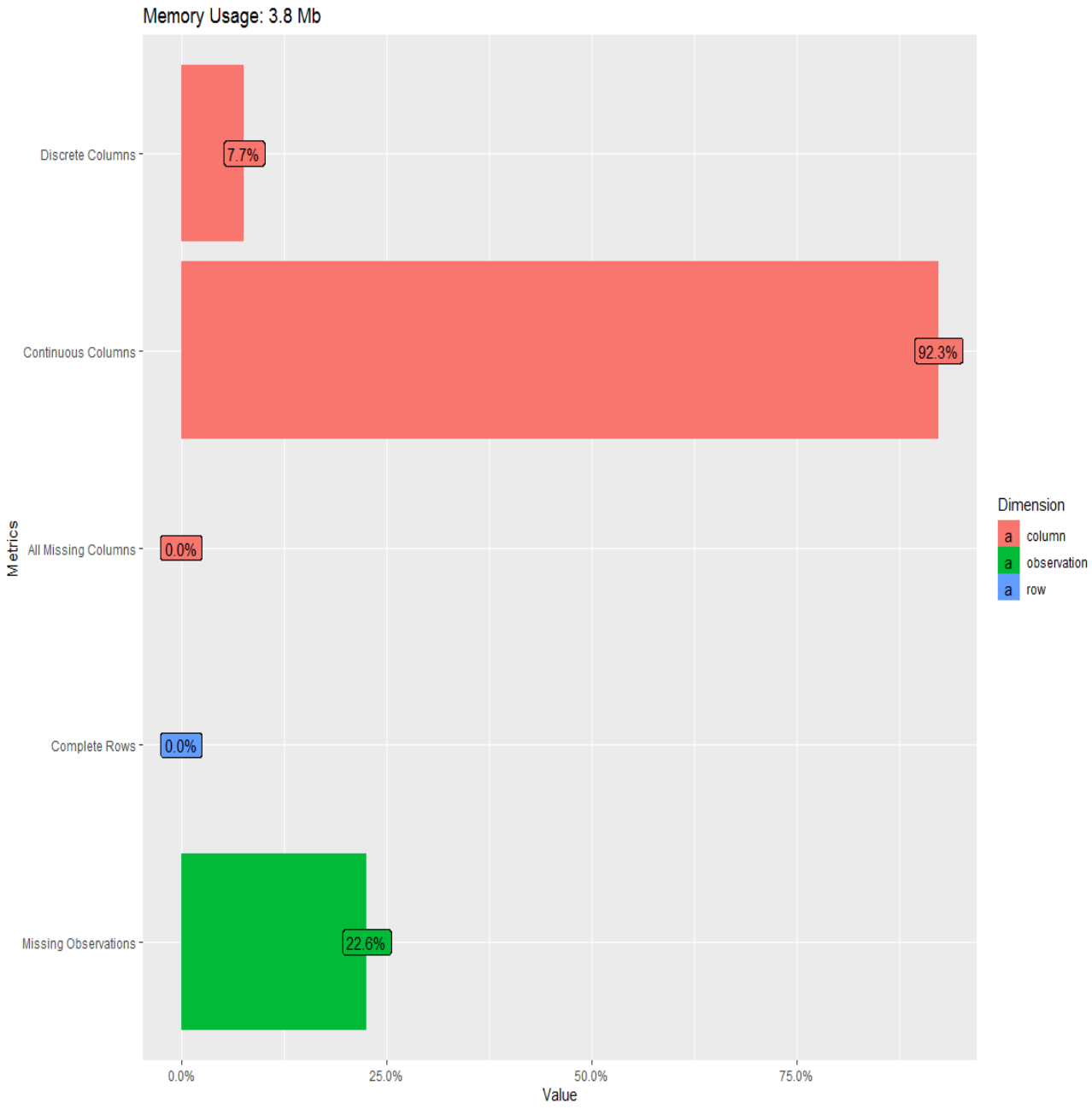
[2] http://adni.loni.usc.edu/data-samples/access-data/

[3] https://tadpole.grand-challenge.org/Data/

Data Cleaning and Preparation

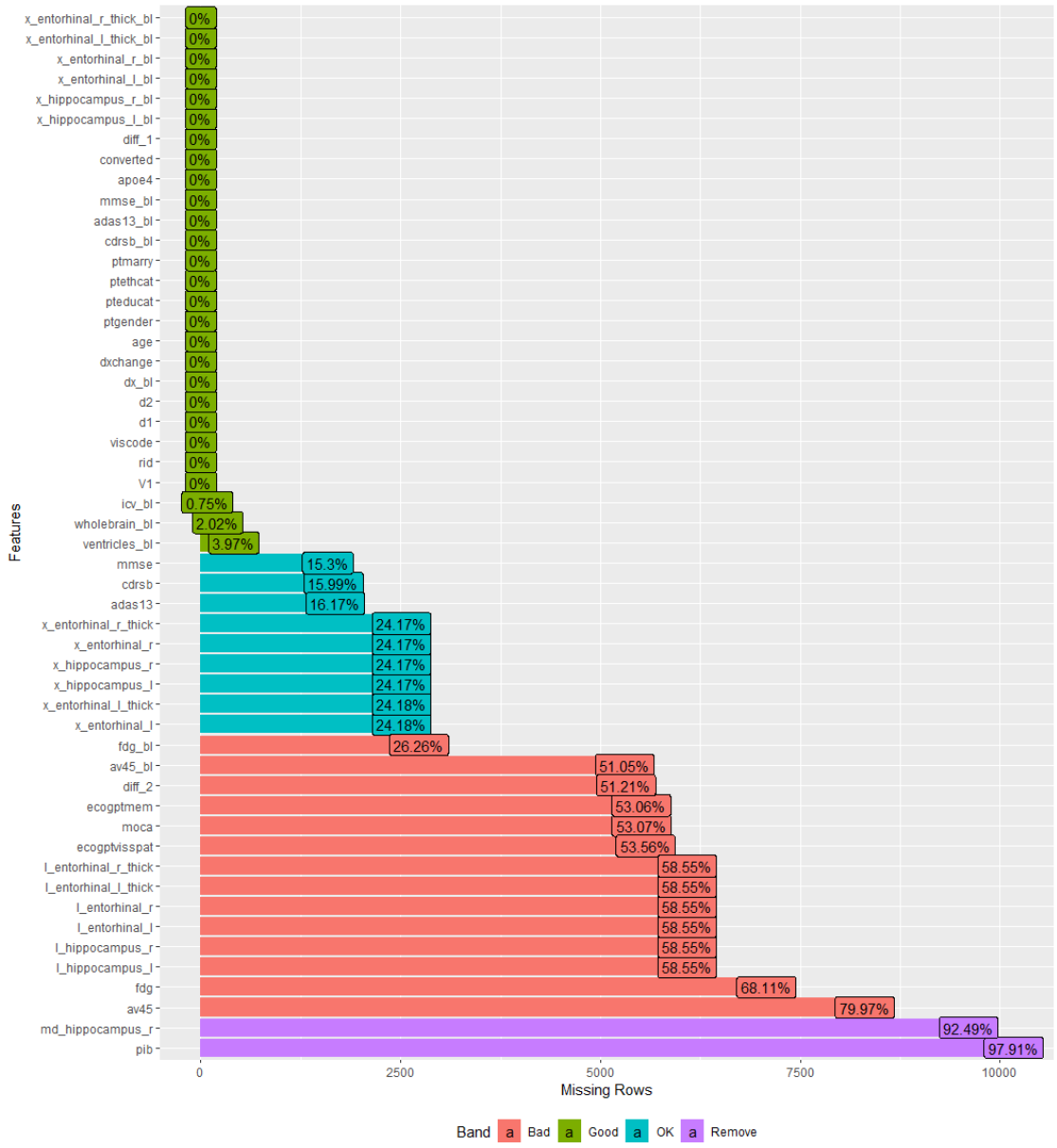
First, let's get a general idea of our data quality and structure. The plot below will give a breakdown of the split between the discrete and continuous columns in the data as well as provide information about missing data.

```
plot_intro(data)
```



We can see that we do have some missing values, according to the chart, about 22.6%. Also, there are no rows where the data is present for every attribute. Let's look at missing's by attribute to see if there are issues with some specific columns.

```
plot_missing(data, group = list(Good = 0.05, OK = 0.25, Bad = 0.8, Remove
  = 1))
```




```
missing_data <- profile_missing(data)
```

We can see that some attributes are missing the majority of their values–we will remove those and keep only attributes that are at least 75% complete. This will leave us with 36 attributes before removing I.D. columns.

```
less_than_25pct_missing <- missing_data %>% filter(pct_missing < 0.25)

data %<>% as.data.frame()
data_2 <- data[,(which(names(data) %in% less_than_25pct_missing$feature))]
```

To deal with the rest of the missing records, we will replace them with the average for the cognitive status associated with the record. This should give us a decent estimate and allow us to move forward.

But first, we need to tag each record with the cognitive status associated with it. We can do this with the attribute dxchange, which tells us what the cognitive status was and what it changed to, allowing us to use this to mark each record with the relevant cognitive status. The attribute dx will be created to hold this value.

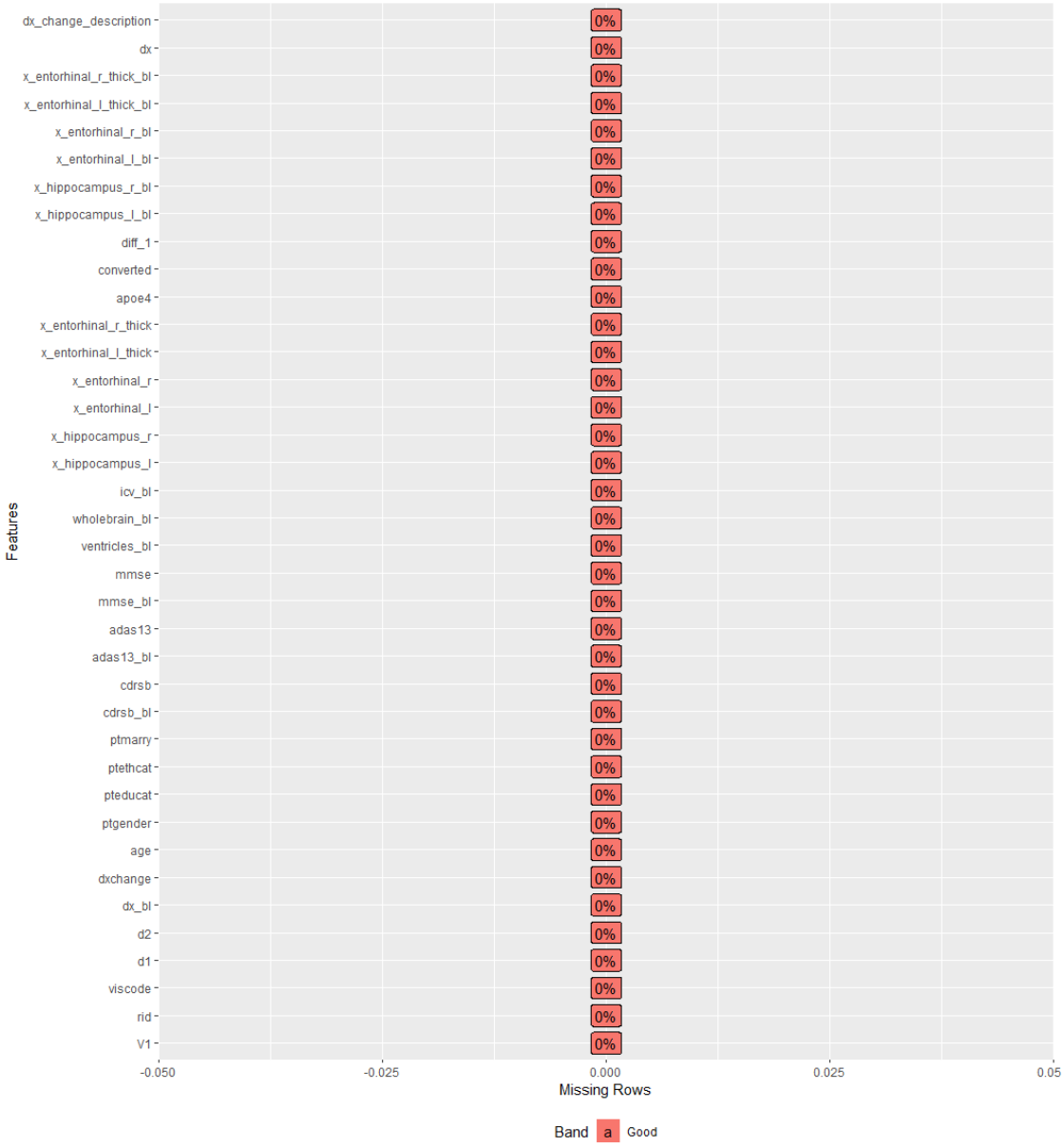
```
data_2 %<>% mutate(dx = case_when(dxchange == 1 ~ "CN",
                                  dxchange == 2 ~ "MCI",
                                  dxchange == 3 ~ "AD",
                                  dxchange == 4 ~ "MCI",
                                  dxchange == 5 ~ "AD",
                                  dxchange == 6 ~ "AD",
                                  dxchange == 7 ~ "CN",
                                  dxchange == 8 ~ "MCI",
                                  dxchange == 9 ~ "CN",
                                  TRUE ~ "NA"),
            dx_change_description =
            case_when(dxchange == 1 ~ "Stable:NL to NL",
                      dxchange == 2 ~ "Stable:MCI to MCI",
                      dxchange == 3 ~ "Stable:AD to AD",
                      dxchange == 4 ~ "Conv:NL to MCI",
                      dxchange == 5 ~ "Conv:MCI to AD",
                      dxchange == 6 ~ "Conv:NL to AD",
                      dxchange == 7 ~ "Rev:MCI to NL",
                      dxchange == 8 ~ "Rev:AD to MCI",
                      dxchange == 9 ~ "Rev:AD to NL",
                      TRUE ~ "NA"))

#Make sure the results make sense
table(data_2$dxchange, useNA = "ifany")
table(data_2$dx, useNA = "ifany")
table(data_2$dx_change_description, useNA = "ifany")
```

Now we can replace missing values with the average for the cognitive status for that record.

```
data_2 <- data_2 %>%  group_by(dx) %>%
  mutate_all(funs(ifelse(is.na(.), mean(., na.rm = TRUE),.)))
```

Make sure we have no missing values after replacing missing's with averages.

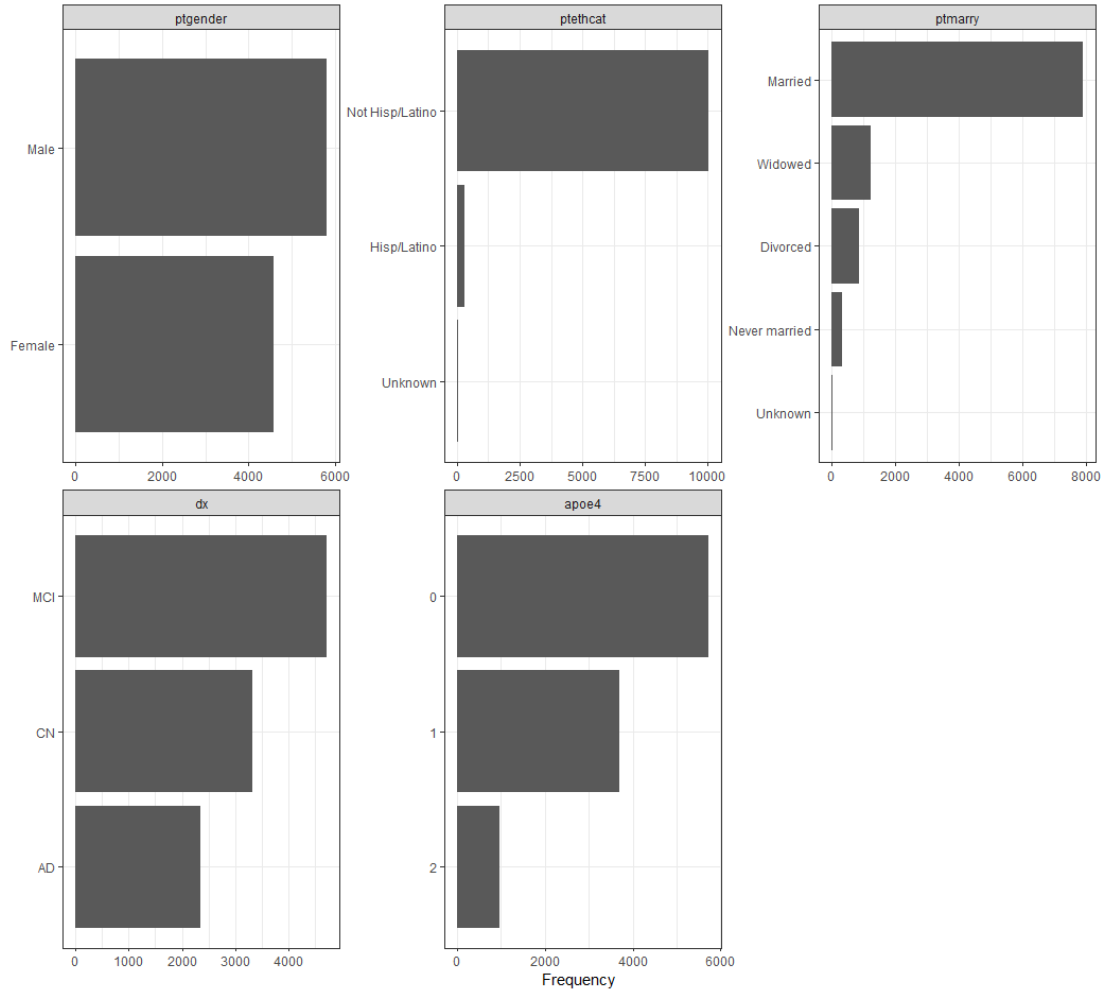
```
plot_missing(data_2)
```



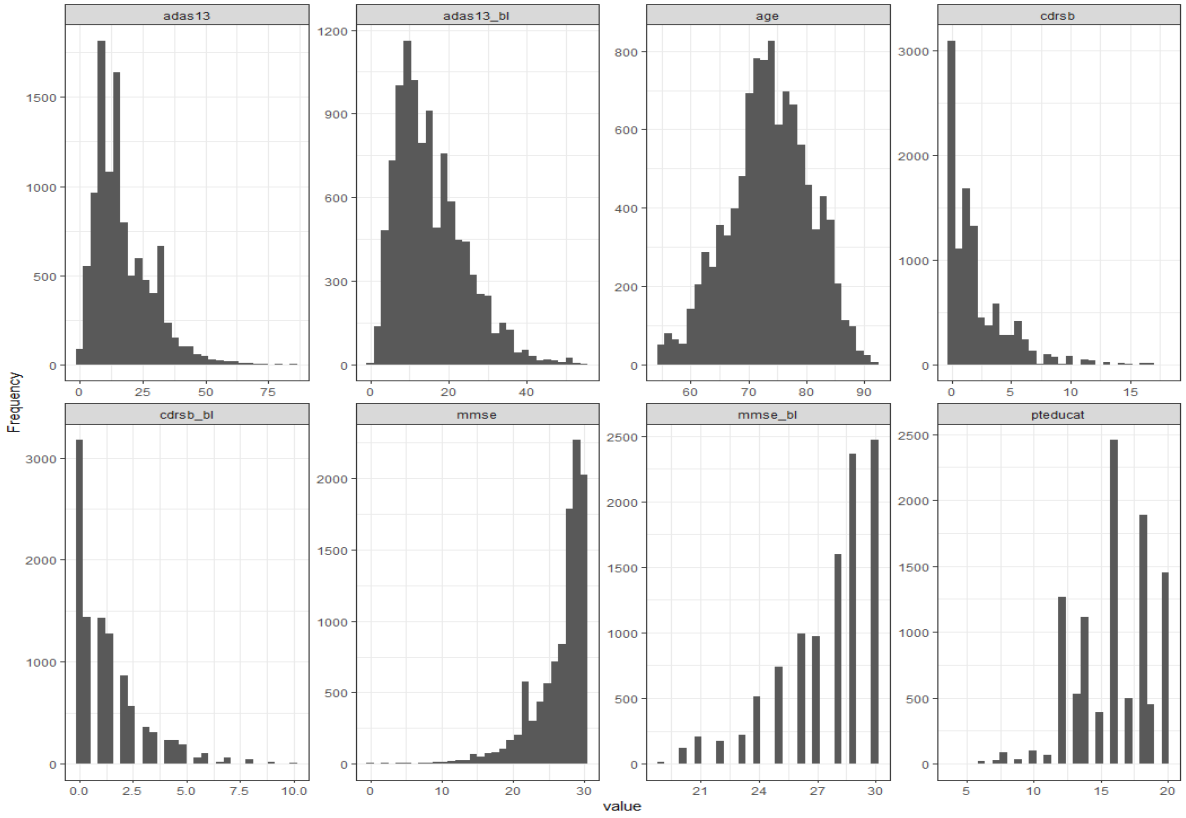
EDA

Now that we have cleaned up our data and dealt with missing values, let's get an idea of the attributes we are dealing with. We will start by looking at the distributions of our demographic variables. Some things to note about the data: the majority of records are male, have a marital status of married, are not Hispanic/Latino, do not carry any copies of the apoe4 gene, are on average 73.66659 years old, and have on average 15.97882 years of education.

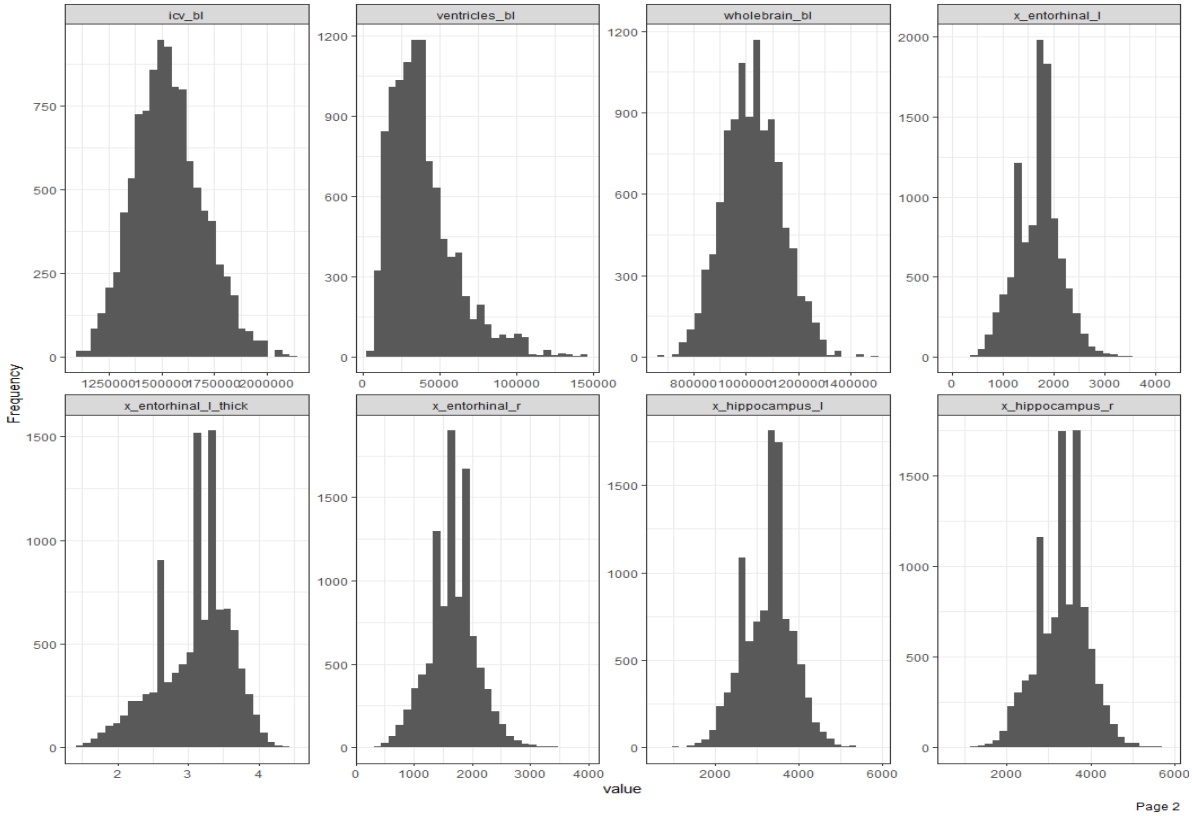
```r
data_2 <- ungroup(data_2)
data_2 <- as.data.table(data_2)

data_2 %<>% mutate(apoe4 = factor(apoe4),
                   converted = factor(converted))

bar_data <- data_2 %>% select(ptgender, ptethcat, ptmarry, dx, apoe4)
hist_data <- data_2 %>% select(c(8,10,13:27,31:36))

plot_bar(bar_data, ggtheme = theme_bw())#, nrow = 2, ncol = 2)
```



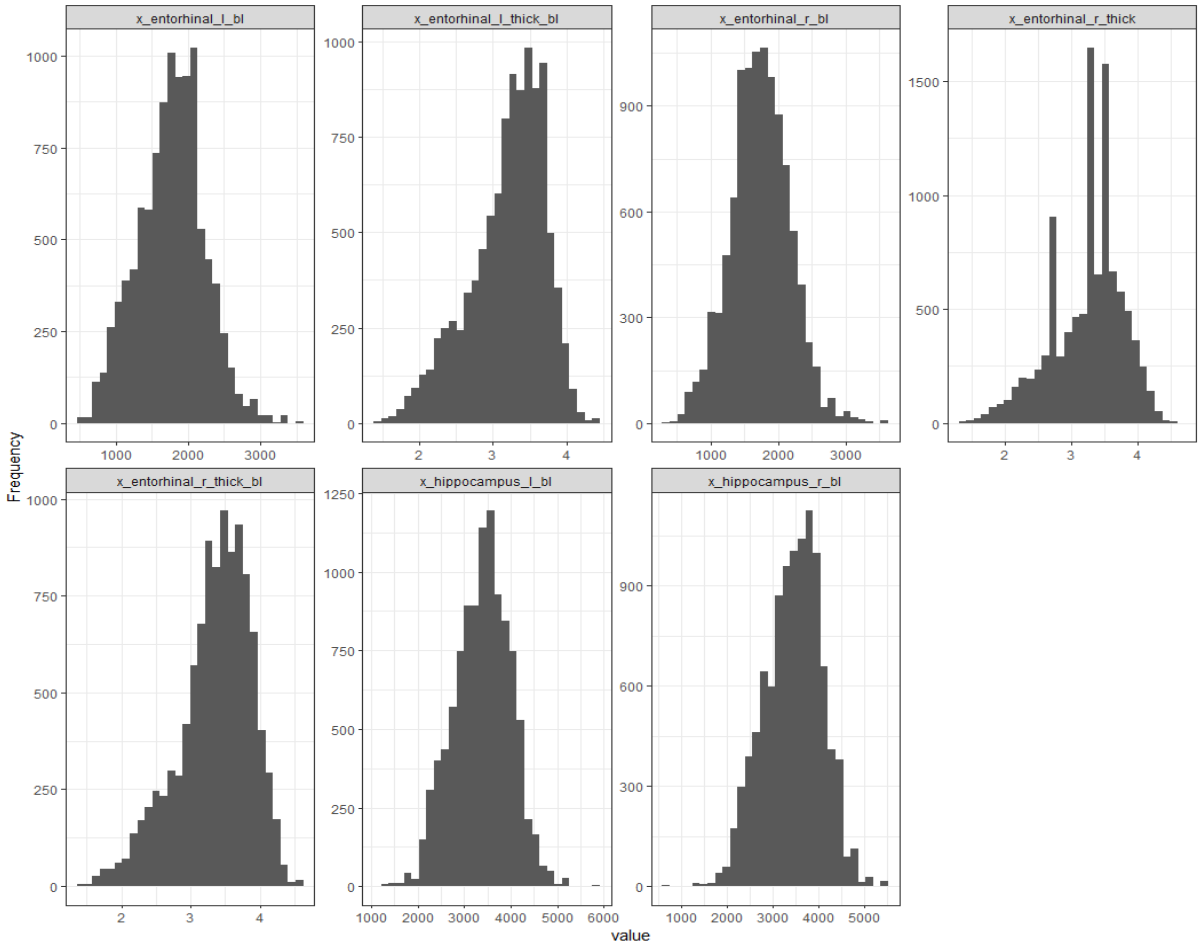

```r
plot_histogram(hist_data, ggtheme = theme_bw(), nrow = 2)#, ncol = 2)
```


adas13
adas13_bl
age
cdrsb
cdrsb_bl
mmse
mmse_bl
pteducat
Frequency
value

Page 1

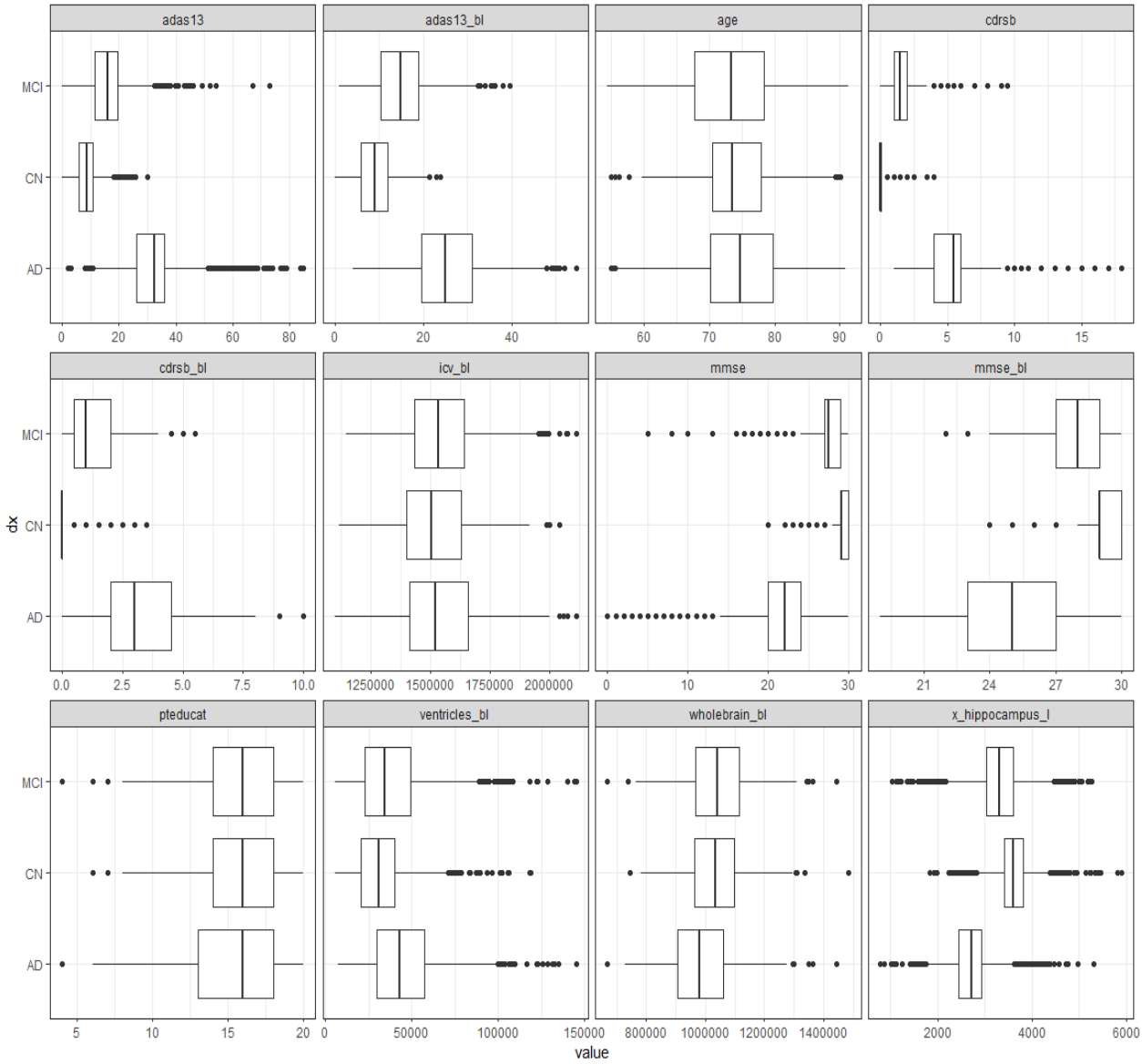

icv_bl
ventricles_bl
wholebrain_bl
x_entorhinal_l
x_entorhinal_l_thick
x_entorhinal_r
x_hippocampus_l
x_hippocampus_r
Frequency
value

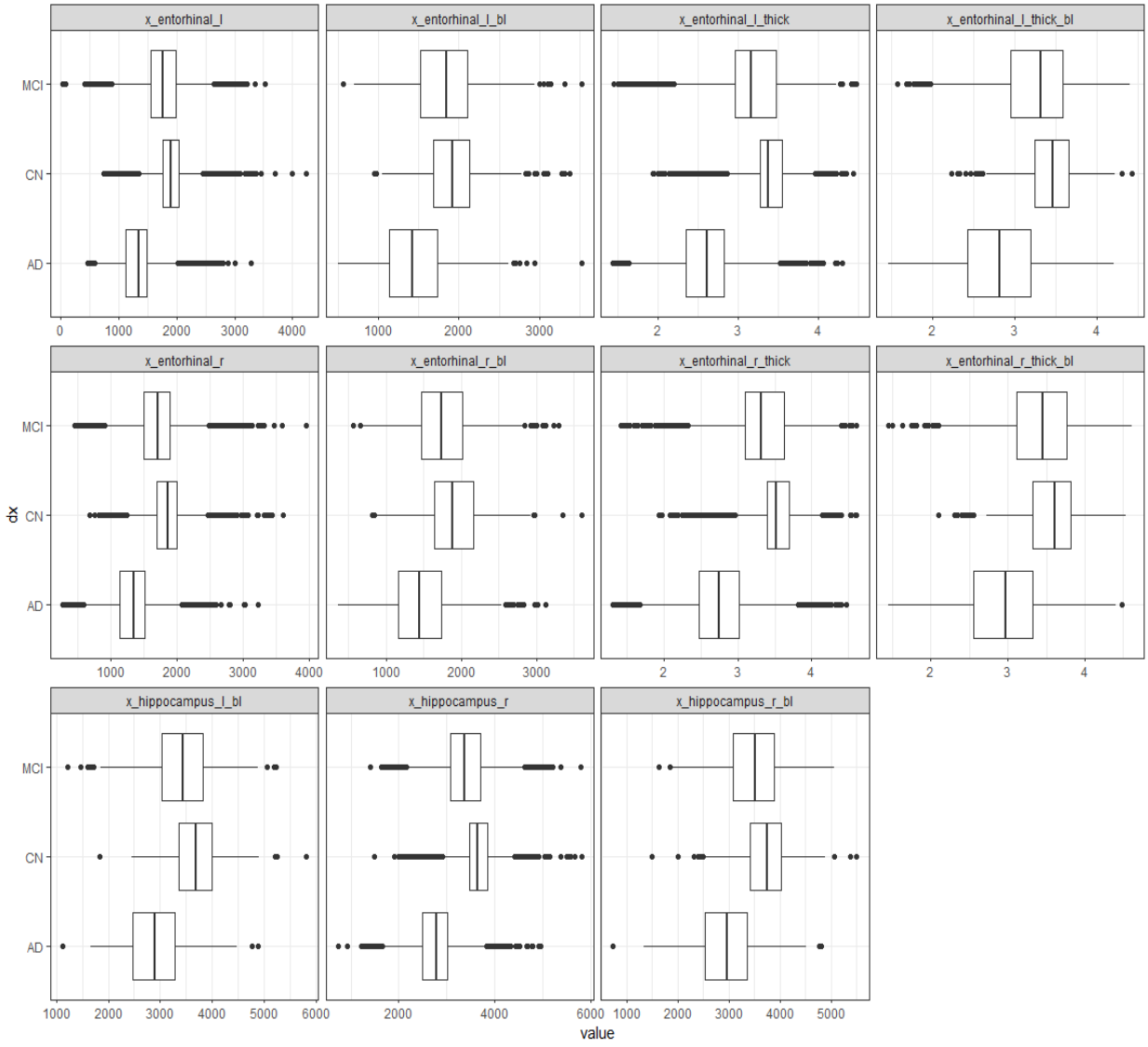
Page 2



Next we will create boxplots for each of the continuous variables by cognitive status. Many of these boxplots reveal some large differences between the various cognitive statuses. In particular, there are large discrepancies between the statuses for the neuropsychological tests–in most cases the IQRs for AD do not overlap with any other status. We see a similar pattern with the measures of brain structural integrity, however there are many more outliers present for some of these measures.

```
box_data <- data_2 %>% select(c(8,10,13:27,31:37))

plot_boxplot(box_data, by = "dx", ggtheme = theme_bw())#, nrow = 2, ncol = 2)
```


adas13
adas13_bl
age
cdrsb
cdrsb_bl
icv_bl
mmse
mmse_bl
pteducat
ventricles_bl
wholebrain_bl
x_hippocampus_l
MCI
CN
AD
dx
value

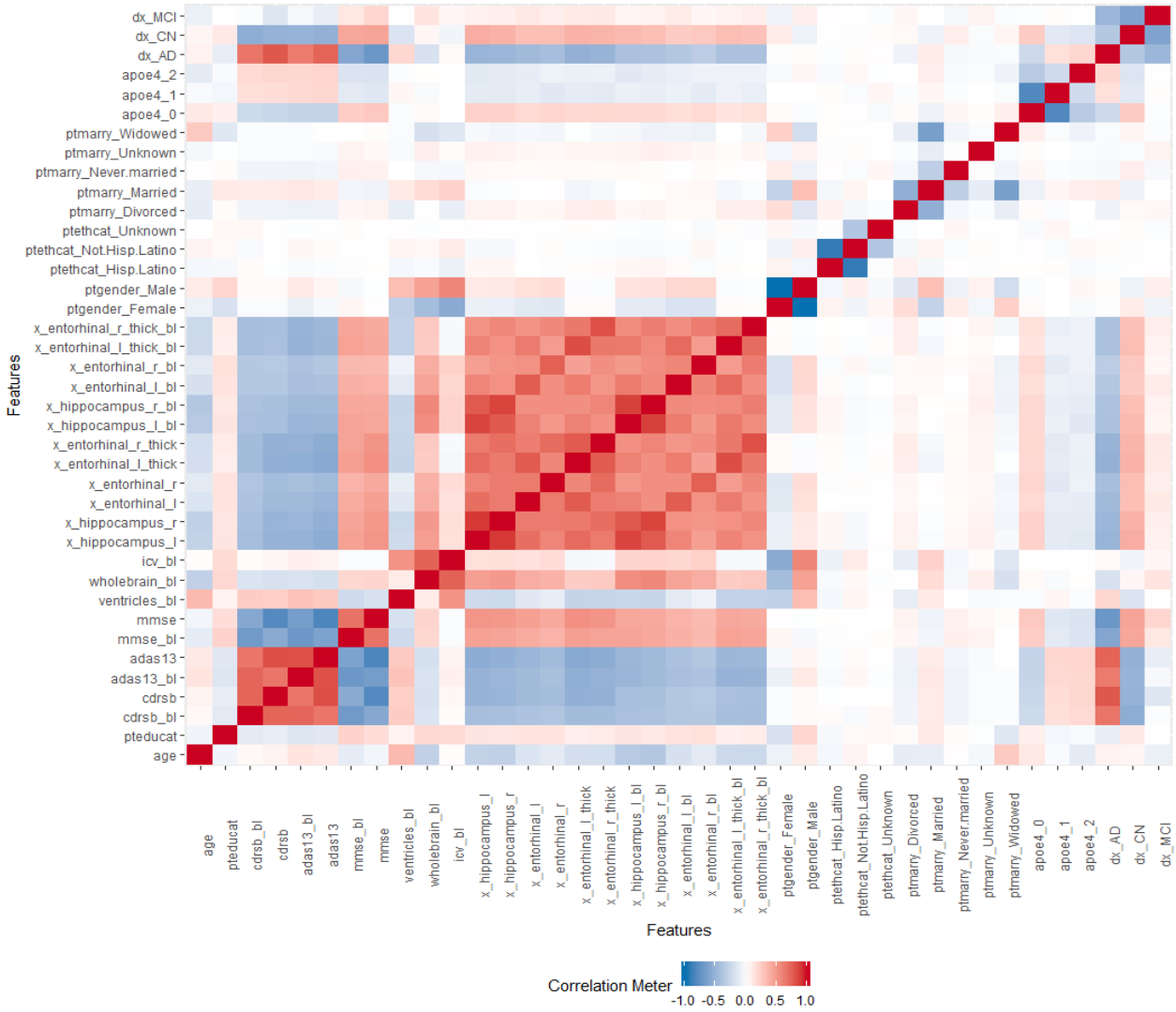



```
data_3 <- data_2 %>% select(c(8:28,31:37))
```

Correlation Analysis

Next we will look at correlations between our attributes. A correlation matrix for the variables is below.

```
plot_correlation(data_3)
```

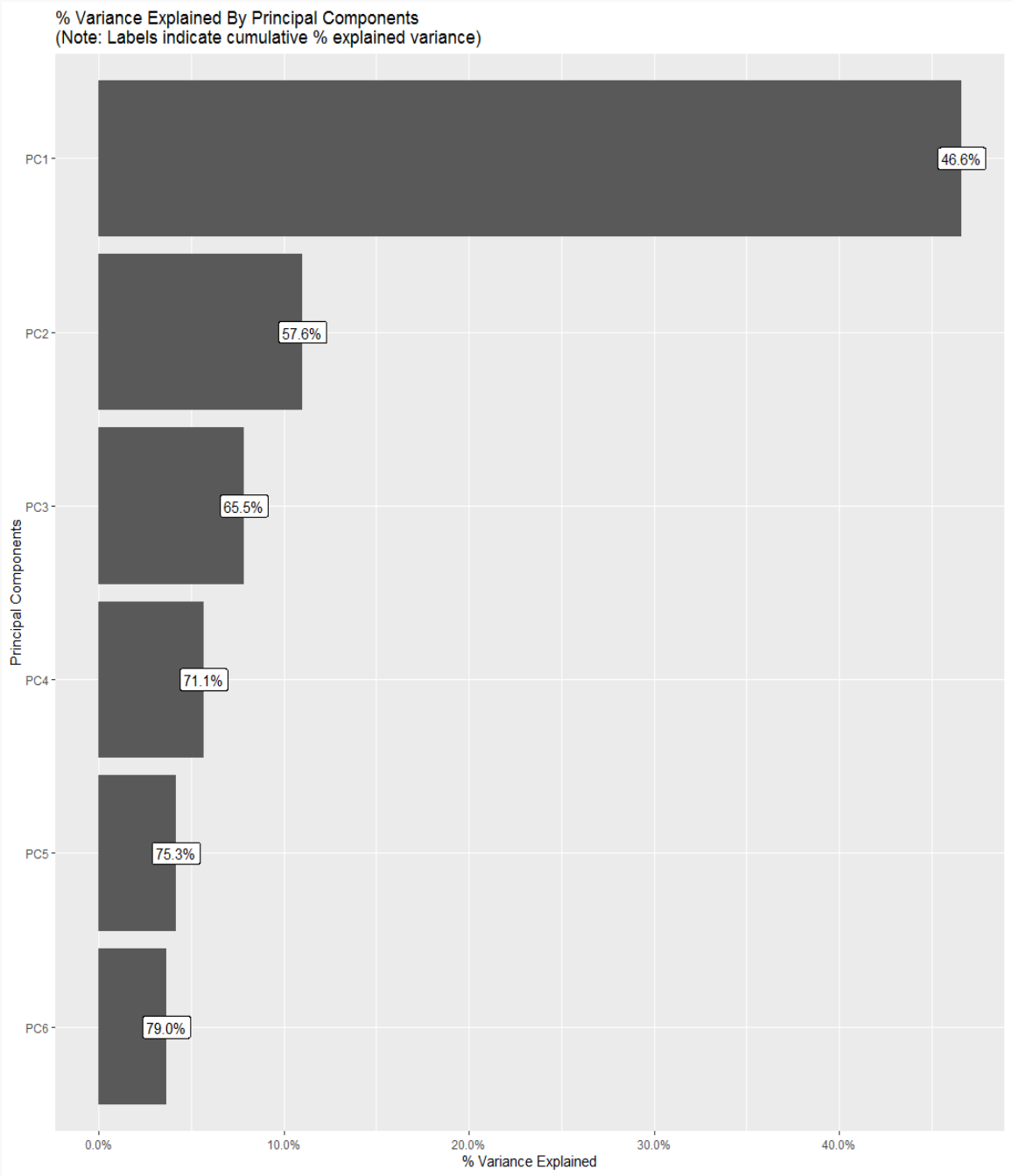


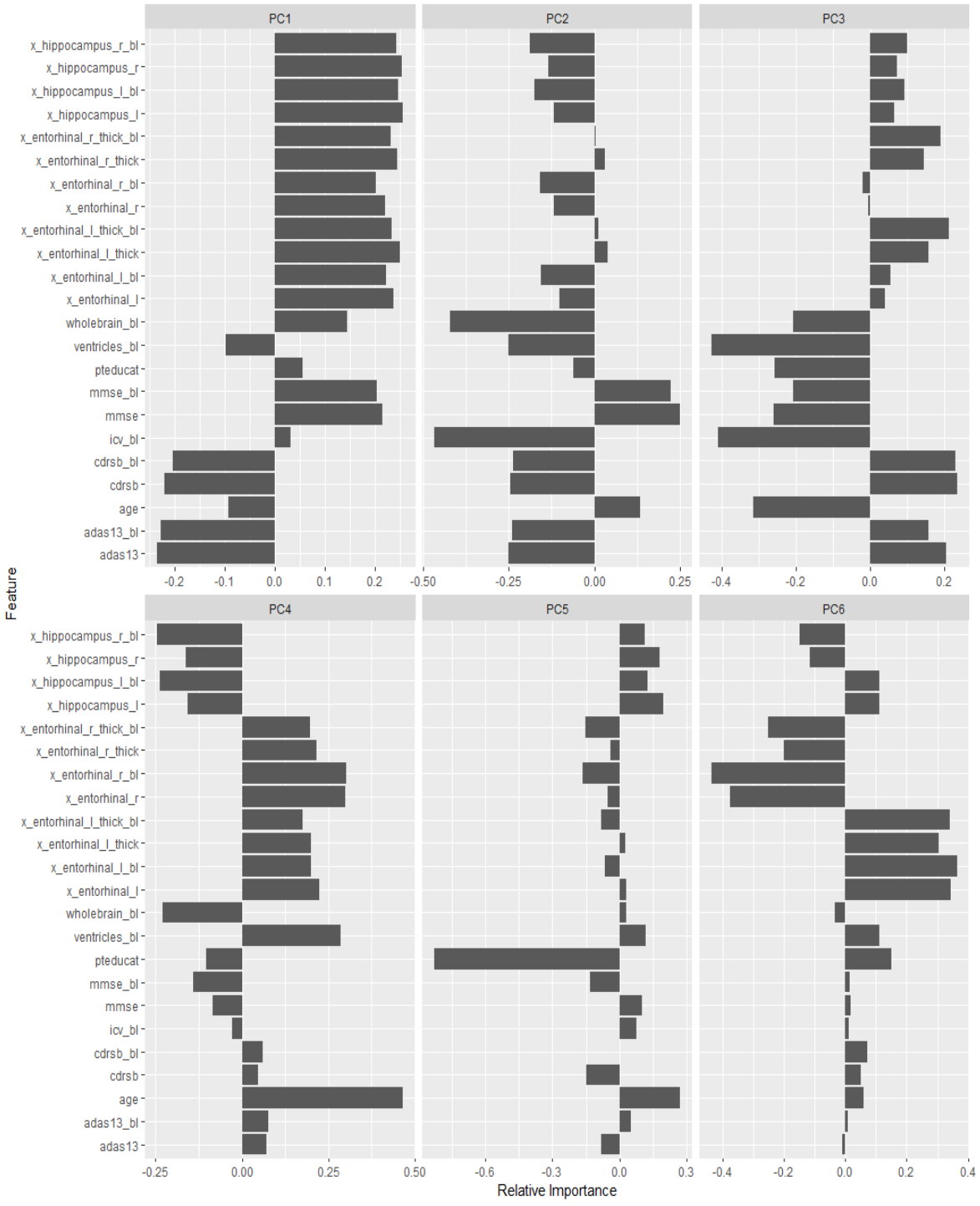
As can be seen, we have a lot of strong correlation occurring between certain variables. This is almost entirely occurring between the measures of brain structural integrity, which can be seen as the red block in the center. This is not too surprising, as we would expect many of these to vary together. We also see correlations between the neuropsychological tests. Moreover, we see correlations between different variables and their baselines, which is denoted by the variable ending in bl. Based on this, we may want to consider eliminating some of these variables. However, since we do not have perfect correlation for any, doing so may remove some information.

PCA

Principal component analysis may help us decide if there are any attributes that it makes sense to remove.

```
plot_prcomp(box_data[-24], nrow = 2)#, ncol = 2)
```





As can be seen, the first two principal components account for over half of the variance, with the first five accounting for over two thirds. The measures of brain structural integrity and the neuropsychological tests carry the majority of the importance in the first component. With this in mind, coupled with some of the strong correlations, we can try to find subsets of attributes that are much smaller than the full set that should still provide good predictive power.

Feature Selection and Engineering

Since what we really want to identify are those that have Alzheimer's versus those that do not, we will change our target variable to be binary. TRUE will indicate records for which the cognitive status is Alzheimer's, and FALSE will indicate all other cognitive statuses.

```
data_4 <- data_3
data_4 %<>% mutate(dx = ifelse(dx == "AD", TRUE, FALSE))
```

With the help of some feature selection and creation tools in RapidMiner, a few subsets of attributes and engineered features were identified. Each of these subsets were used to build a logistic regression model. The results for the subsets were each quite good and with certain measures beat the model built using all attributes. However, when taken holistically, using all of the attributes results in a more accurate model. Since we do not have so many attributes as to cause performance issues, going forward all variables will be used to build the models.

	Full	subset1	subset2	subset3
Accuracy	0.956	0.949	0.945	0.944
Precision	0.879	0.88	0.836	0.836
Recall	0.927	0.884	0.926	0.909
F1	0.815	0.779	0.774	0.761

```
data_subset1 <- data_4 %>%
  mutate(mmse_log = log(mmse_bl),
         cdrsb_2 = 1/cdrsb) %>%
  mutate(cdrsb_2 = ifelse(is.infinite(cdrsb_2),0,cdrsb_2)) %>%
  select(x_entorhinal_r_thick, cdrsb, mmse_bl, x_hippocampus_l, mmse_log, cdr
sb_2, dx)

data_subset2 <- data_4 %>%
  mutate(test_total = cdrsb*adas13) %>%
  select(test_total, dx)

data_subset3 <- data_4 %>%
  mutate(mmse_log = log(mmse_bl),
         cdrsb_2 = 1/cdrsb,
         test_total = cdrsb*adas13) %>%
  mutate(cdrsb_2 = ifelse(is.infinite(cdrsb_2),0,cdrsb_2)) %>%
  select(mmse_bl, x_entorhinal_r_thick, x_hippocampus_l, mmse_log, cdrsb_2, t
est_total, dx)

data_subset1 %<>% mutate(rid = 1:nrow(data_4))
data_subset2 %<>% mutate(rid = 1:nrow(data_4))
data_subset3 %<>% mutate(rid = 1:nrow(data_4))
```

Create Train and Test Split

To guard against overfitting, we will create a train and test split. Our training set will contain 7,500 records and our test set will contain 2,887 records.

```
data_5 <- data_4 %>% mutate(rid = 1:nrow(data_4))

train <- sample_n(data_5, 7500)
test <- data_5 %>% filter(!(rid %in% train$rid))

train_rid <- train$rid
test_rid <- test$rid

train %<>% select(-rid)
test %<>% select(-rid)

score_card <- data.table(model_name = "",test_accuracy = "",
                         test_precision = "",
                         test_recall = "", test_f1 = "")
```

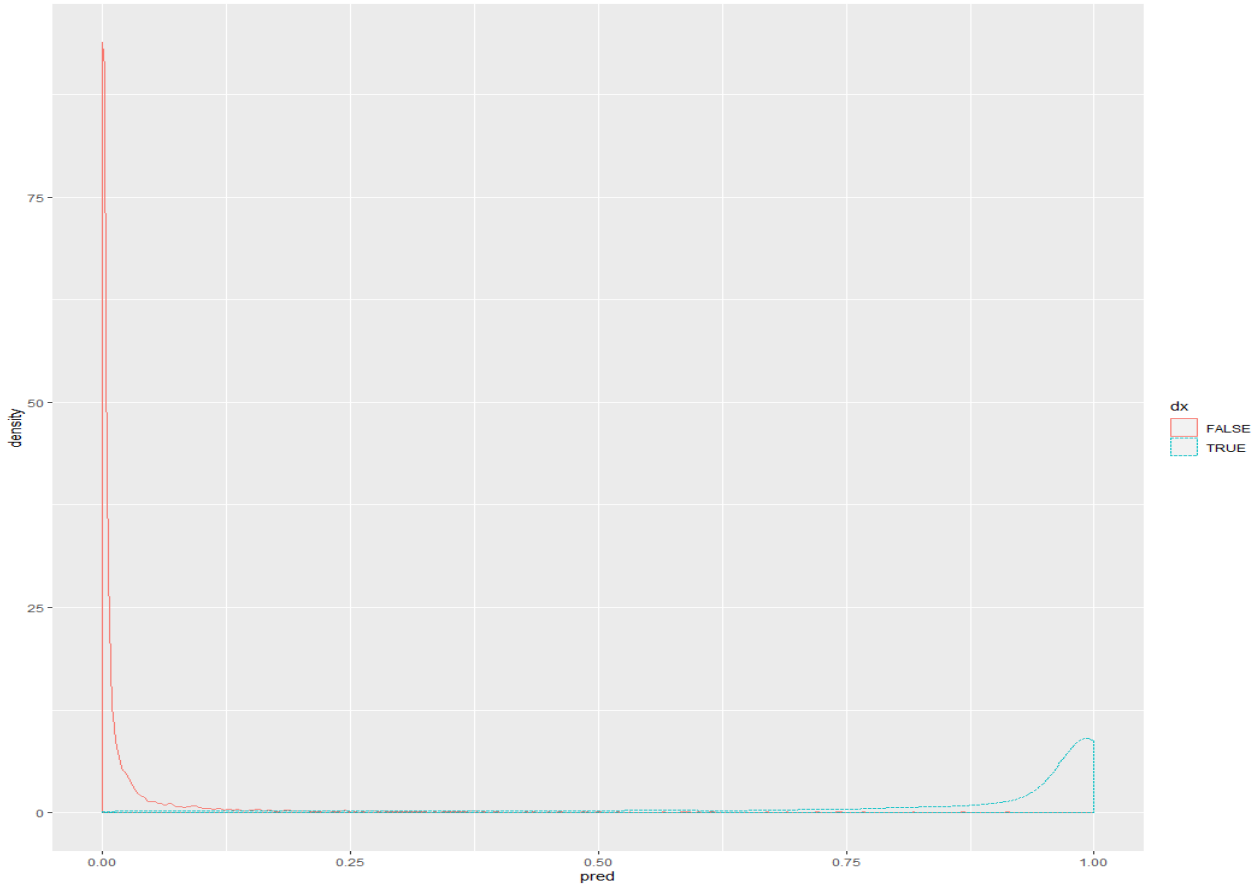
Logistic Regression

Logistic regression will be the first model we try. This will serve as our baseline model by which subsequent ones will be evaluated.

```
vars <- paste(colnames(train)[-28], sep = "", collapse = " + ")
vars <- paste("dx ~ ", vars, sep = "")
formula <- as.formula(vars)
formula

## dx ~ age + ptgender + pteducat + ptethcat + ptmarry + cdrsb_bl +
##     cdrsb + adas13_bl + adas13 + mmse_bl + mmse + ventricles_bl +
##     wholebrain_bl + icv_bl + x_hippocampus_l + x_hippocampus_r +
##     x_entorhinal_l + x_entorhinal_r + x_entorhinal_l_thick +
##     x_entorhinal_r_thick + apoe4 + x_hippocampus_l_bl + x_hippocampus_r_bl
+
##     x_entorhinal_l_bl + x_entorhinal_r_bl + x_entorhinal_l_thick_bl +
##     x_entorhinal_r_thick_bl

glm.fit <- glm(formula=formula, data=train, family=binomial(link="logit"))

glm.train <- train
glm.train$pred <- predict(glm.fit, newdata=train, type="response")
glm.test <- test
glm.test$pred <- predict(glm.fit, newdata=test, type="response")
```

After building the model we will look at the distribution of predictions to see how well separated they are.

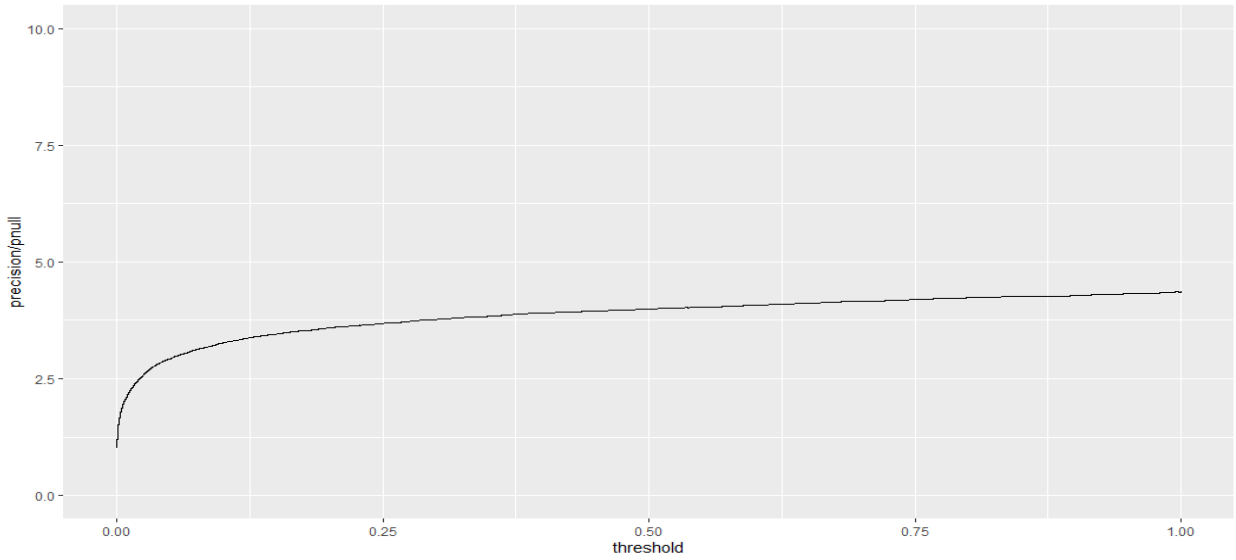
```
ggplot(glm.train, aes(x=pred, color=dx, linetype=dx)) +
  geom_density()
```

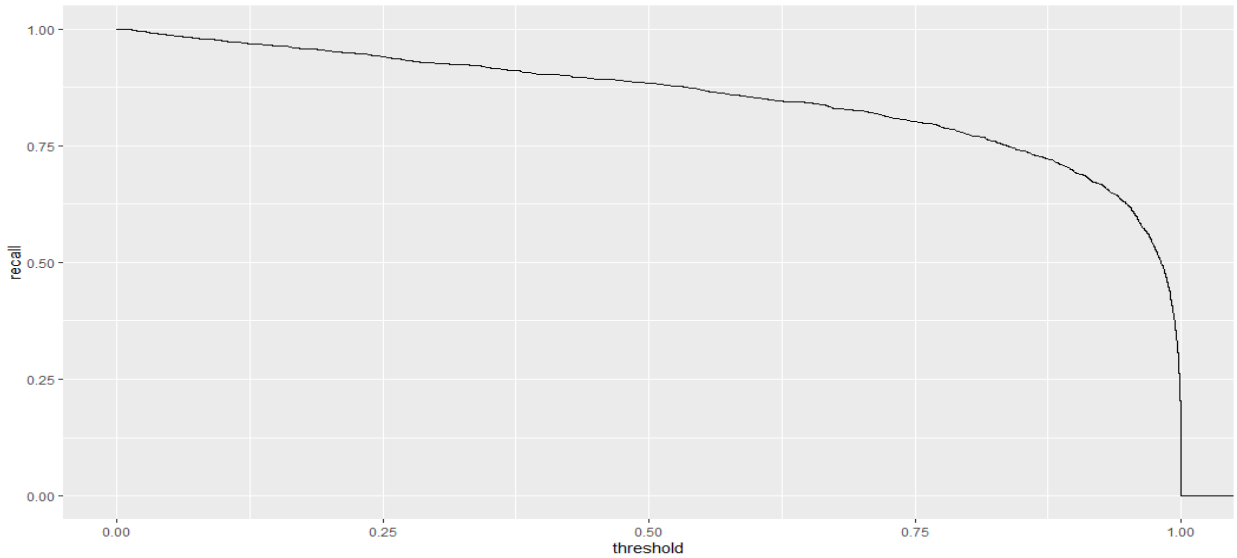


As can be seen, they are well separated with the positive instances concentrated on the right and the negative instances concentrated on the left. We now need to decide on a threshold. Plotting the tradeoffs between precision and recall for various threshold values should help with this decision.

```
predObj <- prediction(glm.train$pred, glm.train$dx)
precObj <- performance(predObj, measure = 'prec')
recObj <- performance(predObj, measure='rec')

precision <- (precObj@y.values)[[1]]
prec.x <- (precObj@x.values)[[1]]
recall <- (recObj@y.values)[[1]]

rocFrame <- data.frame(threshold=prec.x, precision=precision, recall=recall)

nplot <- function(plist) {
  n <- length(plist)
  grid.newpage()
  pushViewport(viewport(layout=grid.layout(n,1)))
  vplayout= function(x,y) {viewport(layout.pos.row=x,  layout.pos.col=y)}
```

```
  for (i in 1:n) {
    print(plist[[i]], vp=vplayout(i,1))
  }
}

pnull<- mean(as.numeric(glm.train$dx))

p1 <- ggplot(rocFrame, aes(x=threshold)) +
  geom_line(aes(y=precision/pnull)) +
  coord_cartesian(xlim = c(0,1), ylim = c(0,10) )

p2 <- ggplot(rocFrame, aes(x=threshold)) +
  geom_line(aes(y=recall)) +
  coord_cartesian(xlim = c(0,1) )

nplot(list(p1,p2))
```





It appears that we can make the threshold quite high without loss in recall. A threshold of 0.5 likely makes sense, as the improvement in precision appears to level off here and we do begin to see a steeper decrease in recall beyond that point.

```
glm_cm_test <- table(pred=glm.test$pred>0.5, dx=glm.test$dx)
glm_cm_train <- table(pred=glm.train$pred>0.5, dx=glm.train$dx)

#cm <- with(glm.fitted,table(y=dx,pred=pred_dx))
glm_cm_train
```

```
##        dx
## pred    FALSE TRUE
##   FALSE  5640  198
##   TRUE    146 1516
```

```
accuracy <- sum(diag(glm_cm_train))/sum(glm_cm_train)
precision <- glm_cm_train[2,2]/sum(glm_cm_train[,2])
recall <- glm_cm_train[2,2]/sum(glm_cm_train[2,])
f1 <- 2*((precision*recall)/(precision+recall))

accuracy
```

```
## [1] 0.9541333
```

```
precision
```

```
## [1] 0.8844807
```

```
recall
```

```
## [1] 0.912154
```

```
f1
```

```
## [1] 0.8067827
```

```
glm_cm_test
```

```
##        dx
## pred    FALSE TRUE
##   FALSE  2189   87
##   TRUE     65  546
```

```
accuracy <- sum(diag(glm_cm_test))/sum(glm_cm_test)
precision <- glm_cm_test[2,2]/sum(glm_cm_test[,2])
recall <- glm_cm_test[2,2]/sum(glm_cm_test[2,])
f1 <- 2*((precision*recall)/(precision+recall))

accuracy
```

```
## [1] 0.9473502
```

```
precision
```

```
## [1] 0.8625592

recall

## [1] 0.893617

f1

## [1] 0.7707976

temp_card <- data.table(model_name = "logistic regression", test_accuracy = a
ccuracy,
                        test_precision = precision, test_recall = recall, tes
t_f1 = f1)

score_card <- rbind(score_card, temp_card)
```

KNN

Next we will build a KNN model. Unlike the other methods implemented in R, KNN will require us to explicitly encode our categorical variables. It is also good practice to normalize our values with KNN to ensure our distance calculations are on the same scale. We will use one-hot encoding to create indicators for each level of the categorical values.

```
dmy <- dummify(data_4)
dmy %<>% select(24:33)

data_6 <- data_4 %>% select(1,3,6:28)

data_6$apoe4 %<>% as.numeric()
data_6$mmse_bl %<>% as.numeric()
data_6$pteducat %<>% as.numeric()

data_6_n <- as.data.frame(lapply(data_6[1:24], normalize))
data_6_dx <- data_6 %>% select(dx)

data_6 <- cbind(data_6_n, dmy, data_6_dx)

data_6[25:34] <- lapply(data_6[25:34], as.numeric)

data_7 <- data_6 %>% mutate(rid = 1:nrow(data_6))

knn_train <- sample_n(data_7, 7500)
knn_test <- data_7 %>% filter(!(rid %in% knn_train$rid))

knn_train %<>% select(-rid)
knn_test %<>% select(-rid)
```

```r
knn_train <- knn_train %>% mutate(dx = ifelse(dx, "TRUE", "FALSE"))
knn_test <- knn_test %>% mutate(dx = ifelse(dx, "TRUE", "FALSE"))

str(knn_train)
```

```
## 'data.frame':    7500 obs. of  35 variables:
##  $ age                     : num  0.459 0.624 0.654 0.254 0.189 ...
##  $ pteducat                : num  1 0.625 0.562 0.75 0.75 ...
##  $ cdrsb_bl                : num  0 0.1 0 0.05 0.1 0.3 0.5 0.2 0.15 0.1 ..
.
##  $ cdrsb                   : num  0 0.0833 0 0.1944 0.0278 ...
##  $ adas13_bl               : num  0.2195 0.2683 0.0792 0.2561 0.2866 ...
##  $ adas13                  : num  0.0471 0.1882 0.0627 0.2353 0.0745 ...
##  $ mmse_bl                 : num  0.818 0.545 1 0.727 0.727 ...
##  $ mmse                    : num  1 0.8 1 0.833 0.967 ...
##  $ ventricles_bl           : num  0.188 0.214 0.126 0.179 0.048 ...
##  $ wholebrain_bl           : num  0.558 0.304 0.296 0.648 0.616 ...
##  $ icv_bl                  : num  0.423 0.364 0.221 0.583 0.515 ...
##  $ x_hippocampus_l         : num  0.552 0.353 0.44 0.495 0.775 ...
##  $ x_hippocampus_r         : num  0.574 0.309 0.472 0.521 0.776 ...
##  $ x_entorhinal_l          : num  0.444 0.269 0.614 0.411 0.411 ...
##  $ x_entorhinal_r          : num  0.43 0.215 0.596 0.39 0.669 ...
##  $ x_entorhinal_l_thick    : num  0.636 0.387 0.8 0.566 0.779 ...
##  $ x_entorhinal_r_thick    : num  0.669 0.378 0.754 0.606 0.872 ...
##  $ apoe4                   : num  0 0.5 0 0 0 0.5 0 0 0 0.5 ...
##  $ x_hippocampus_l_bl      : num  0.601 0.316 0.42 0.404 0.738 ...
##  $ x_hippocampus_r_bl      : num  0.683 0.337 0.5 0.572 0.791 ...
##  $ x_entorhinal_l_bl       : num  0.523 0.24 0.654 0.381 0.524 ...
##  $ x_entorhinal_r_bl       : num  0.522 0.227 0.621 0.424 0.514 ...
##  $ x_entorhinal_l_thick_bl : num  0.647 0.478 0.829 0.572 0.796 ...
##  $ x_entorhinal_r_thick_bl : num  0.729 0.288 0.759 0.612 0.833 ...
##  $ ptgender_Female         : num  0 1 1 0 0 0 1 0 0 1 ...
##  $ ptgender_Male           : num  1 0 0 1 1 1 0 1 1 0 ...
##  $ ptethcat_Hisp.Latino    : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ ptethcat_Not.Hisp.Latino: num  1 1 1 1 1 1 1 1 1 1 ...
##  $ ptethcat_Unknown        : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ ptmarry_Divorced        : num  0 1 0 0 0 1 0 0 0 1 ...
##  $ ptmarry_Married         : num  1 0 1 1 1 0 1 1 0 0 ...
##  $ ptmarry_Never.married   : num  0 0 0 0 0 0 0 0 1 0 ...
##  $ ptmarry_Unknown         : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ ptmarry_Widowed         : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ dx                      : chr  "FALSE" "FALSE" "FALSE" "FALSE" ...
```

```r
knn_train$dx %<>% as.factor()
knn_test$dx %<>% as.factor()

train_lables <- knn_train[,35]
test_lables <- knn_test[,35]

knn_train %<>% select(-dx)
```

```
knn_test %<>% select(-dx)

knn_fit <- knn(train = knn_train, test = knn_test, cl = train_lables, k=5)

tab <- table(knn_fit,test_lables)
tab

##        test_lables
## knn_fit FALSE TRUE
##   FALSE  2198  129
##   TRUE     43  517

accuracy <- sum(diag(tab))/sum(tab)
precision <- tab[1,1]/sum(tab[,1])
recall <- tab[1,1]/sum(tab[1,])
f1 <- 2*((precision*recall)/(precision+recall))

accuracy

## [1] 0.9404226

precision

## [1] 0.9808121

recall

## [1] 0.9445638

f1

## [1] 0.9264397

temp_card <- data.table(model_name = "knn", test_accuracy = accuracy,
                        test_precision = precision,
                        test_recall = recall, test_f1 = f1)

score_card <- rbind(score_card, temp_card)
```

Tree Based Model

Next up is a decision tree. Decisions trees are particularly nice as there is no need to do any additional prep for categorical attributes, they can handle them as-is.

```
tree_model <- rpart(formula, train)

tree_train <- train
tree_test <- test

tree_train$pred <- predict(tree_model, newdata=train)
tree_test$pred <- predict(tree_model, newdata=test)
```

```r
tree_cm_train <- with(tree_train,table(pred=tree_train$pred>0.5,y=dx))
tree_cm_test <- with(tree_test,table(pred=tree_test$pred>0.5,y=dx))

#train set measures and Confusion Matrix
tree_cm_train
```

```
##        y
## pred    FALSE TRUE
##   FALSE  5444  112
##   TRUE    342 1602
```

```r
accuracy <- sum(diag(tree_cm_train))/sum(tree_cm_train)
precision <- tree_cm_train[1,1]/sum(tree_cm_train[,1])
recall <- tree_cm_train[1,1]/sum(tree_cm_train[1,])
f1 <- 2*((precision*recall)/(precision+recall))

accuracy
```

```
## [1] 0.9394667
```

```r
precision
```

```
## [1] 0.9408918
```

```r
recall
```

```
## [1] 0.9798416
```

```r
f1
```

```
## [1] 0.9219249
```

```r
#test set measures and Confusion Matrix
tree_cm_test
```

```
##        y
## pred    FALSE TRUE
##   FALSE  2092   46
##   TRUE    162  587
```

```r
accuracy <- sum(diag(tree_cm_test))/sum(tree_cm_test)
precision <- tree_cm_test[1,1]/sum(tree_cm_test[,1])
recall <- tree_cm_test[1,1]/sum(tree_cm_test[1,])
f1 <- 2*((precision*recall)/(precision+recall))

accuracy
```

```
## [1] 0.9279529
```

```r
precision
```

```
## [1] 0.9281278
```

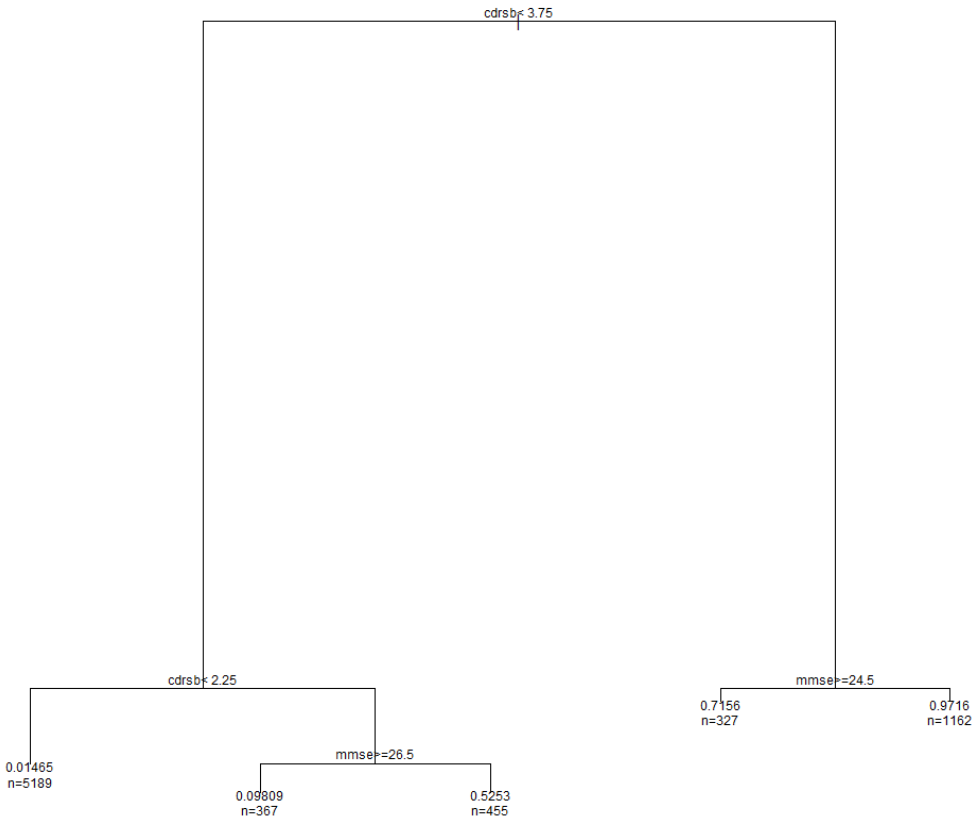
```
recall
```

```
## [1] 0.9784846
```

```
f1
```

```
## [1] 0.9081587
```

```
temp_card <- data.table(model_name = "tree", test_accuracy = accuracy,
                        test_precision = precision,
                        test_recall = recall, test_f1 = f1)

score_card <- rbind(score_card, temp_card)

plot(tree_model)
text(tree_model, use.n = TRUE, cex = 0.75)
```



Overall, I'm pretty happy with these results. The model has good accuracy and the difference between the train and test measures are small, indicating we are likely not overfitting. Additionally, we are getting decent results with a fairly simple tree. It is really

only utilizing two attributes; cdrsb and mmse—both neuropsychological tests, indicating that the tests have strong predictive power.

Bagging the Tree Based Model

The idea behind bagging is to generate many trees from random samples of data. The resulting model is the average of the generated trees. This method should lower variance and improve accuracy, as well as reduce the chances that we see overfitting.

```
#Use bootstrap samples the same size as the training set, with 100 trees.
ntrain <- dim(train)[1]
n <- ntrain
ntree <- 100

#Build the bootstrap samples
samples <- sapply(1:ntree,
FUN = function(iter)
{sample(1:ntrain, size=n, replace=T)})

#Train the individual decision trees and return them in a list. Note: this st
ep can take a few minutes.
treelist <-lapply(1:ntree,
FUN=function(iter)
{samp <- samples[,iter];
rpart(formula, train[samp,])})

#predict.bag assumes the underlying classifier returns decision probabilities
, not decisions.
predict.bag <- function(treelist, newdata) {
preds <- sapply(1:length(treelist),
FUN=function(iter) {
predict(treelist[[iter]], newdata=newdata)})
predsums <- rowSums(preds)
predsums/length(treelist)
}

bagged_tree_train <- train
bagged_tree_test <- test

bagged_tree_train_result <- predict.bag(treelist, newdata=train)
bagged_tree_test_result <- predict.bag(treelist, newdata=test)

bagged_tree_train$pred <- bagged_tree_train_result
bagged_tree_test$pred <- bagged_tree_test_result

bagged_tree_train <- data.table(bagged_tree_train)
bagged_tree_test <- data.table(bagged_tree_test)
```

```r
bagged_tree_cm_train <- with(bagged_tree_train,
                             table(pred=bagged_tree_train$pred>0.5,y=dx))
bagged_tree_cm_test <- with(bagged_tree_test,
                            table(pred=bagged_tree_test$pred>0.5,y=dx))

#train set measures and Confusion Matrix
bagged_tree_cm_train
```

```
##        y
## pred    FALSE TRUE
##   FALSE  5620  219
##   TRUE    166 1495
```

```r
accuracy <- sum(diag(bagged_tree_cm_train))/sum(bagged_tree_cm_train)
precision <- bagged_tree_cm_train[1,1]/sum(bagged_tree_cm_train[,1])
recall <- bagged_tree_cm_train[1,1]/sum(bagged_tree_cm_train[1,])
f1 <- 2*((precision*recall)/(precision+recall))

accuracy
```

```
## [1] 0.9486667
```

```r
precision
```

```
## [1] 0.9713101
```

```r
recall
```

```
## [1] 0.9624936
```

```r
f1
```

```
## [1] 0.9348797
```

```r
#test set measures and Confusion Matrix
bagged_tree_cm_test
```

```
##        y
## pred    FALSE TRUE
##   FALSE  2187   81
##   TRUE     67  552
```

```r
accuracy <- sum(diag(bagged_tree_cm_test))/sum(bagged_tree_cm_test)
precision <- bagged_tree_cm_test[1,1]/sum(bagged_tree_cm_test[,1])
recall <- bagged_tree_cm_test[1,1]/sum(bagged_tree_cm_test[1,])
f1 <- 2*((precision*recall)/(precision+recall))

accuracy
```

```
## [1] 0.9487357
```

```r
precision
```

```
## [1] 0.9702751

recall

## [1] 0.9642857

f1

## [1] 0.9356224

temp_card <- data.table(model_name = "bagged tree", test_accuracy = accuracy,
                        test_precision = precision,
                        test_recall = recall, test_f1 = f1)

score_card <- rbind(score_card, temp_card)
```

Bagging did result in better results than our standard decision tree, though not by a large amount. Given the additional training and scoring time associated with this method, it would likely make sense to stick with our standard decision tree.

Support Vector Machines

Support vector machines improve separation in the data by utilizing a kernel transform. This is typically a good method for cases where interactions and useful combinations of attributes are not know ahead of time. That is certainly the case here, where I lack the domain knowledge to intuit useful engineered features.

```
#Build the support vector model
mSVMV <- ksvm(formula,data=train,kernel='rbfdot')

#Use the model to predict class on held-out data.
svm_train <- train
svm_test <- test

svm_train$pred <- predict(mSVMV,newdata=train,type='response')
svm_test$pred <- predict(mSVMV,newdata=test,type='response')

svm_cm_test <- with(svm_test,table(pred=svm_test$pred>0.5,y=dx))
svm_cm_train <- with(svm_train,table(pred=svm_train$pred>0.5,y=dx))

#train set measures and Confusion Matrix
svm_cm_train

##        y
## pred    FALSE TRUE
##   FALSE  5729  132
##   TRUE     57 1582

accuracy <- sum(diag(svm_cm_train))/sum(svm_cm_train)
precision <- svm_cm_train[1,1]/sum(svm_cm_train[,1])
```

```
recall <- svm_cm_train[1,1]/sum(svm_cm_train[1,])
f1 <- 2*((precision*recall)/(precision+recall))

accuracy
```

```
## [1] 0.9748
```

```
precision
```

```
## [1] 0.9901486
```

```
recall
```

```
## [1] 0.9774782
```

```
f1
```

```
## [1] 0.9678488
```

```
#train set measures and Confusion Matrix
svm_cm_test
```

```
##        y
## pred    FALSE TRUE
##   FALSE  2208   85
##   TRUE     46  548
```

```
accuracy <- sum(diag(svm_cm_test))/sum(svm_cm_test)
precision <- svm_cm_test[1,1]/sum(svm_cm_test[,1])
recall <- svm_cm_test[1,1]/sum(svm_cm_test[1,])
f1 <- 2*((precision*recall)/(precision+recall))

accuracy
```

```
## [1] 0.9546242
```

```
precision
```

```
## [1] 0.9795918
```

```
recall
```

```
## [1] 0.9629307
```

```
f1
```

```
## [1] 0.943279
```

```
temp_card <- data.table(model_name = "svm", test_accuracy = accuracy,
                        test_precision = precision,
                        test_recall = recall, test_f1 = f1)

score_card <- rbind(score_card, temp_card)
```

```r
score_card <- score_card[-1,]

score_card %<>% as.data.frame()
score_card[,-1] <- as.data.frame(lapply(score_card[,-1], as.numeric))
score_card[,-1] <- round(score_card[,-1],4)

tb1 <- tableGrob(glm_cm_test)
tb2 <- tableGrob(tab)
tb3 <- tableGrob(tree_cm_test)
tb4 <- tableGrob(bagged_tree_cm_test)
tb5 <- tableGrob(svm_cm_test)

grid.arrange(arrangeGrob(tb1, top = 'Logistic Regression'), arrangeGrob(tb2,
top = 'KNN'), arrangeGrob(tb3, top = 'Tree'), arrangeGrob(tb4, top = 'Bagged
Tree'),arrangeGrob(tb5, top = 'SVM'), top = "Confusion Matrices”)
```

```r
simple_roc <- function(labels, scores){
  labels <- labels[order(scores, decreasing=TRUE)]
  data.frame(TPR=cumsum(labels)/sum(labels), FPR=cumsum(!labels)/sum(!labels)
, labels)
}
```

```r
tree_roc <- simple_roc(tree_test$dx==TRUE, tree_test$pred)
bagged_roc <- simple_roc(bagged_tree_test$dx==TRUE, bagged_tree_test$pred)
glm_roc <- simple_roc(glm.test$dx==TRUE, glm.test$pred)
svm_roc <- simple_roc(svm_test$dx==TRUE, svm_test$pred)

tree_roc %<>% mutate(name="tree")
bagged_roc %<>% mutate(name="bagged")
glm_roc %<>% mutate(name="logistic regrssion")
svm_roc %<>% mutate(name="svm")

roc_all <- rbind(tree_roc, bagged_roc, glm_roc, svm_roc)

ggplot(roc_all, aes(x=FPR, y=TPR, color=name)) +
  geom_point(size=0.01)
```

Conclusion

To help compare the performance of the various models the confusion matrices, a summary of the performance measures, and a ROC comparison chart are below.

Confusion Matrices

Logistic Regression

	FALSE	TRUE
FALSE	2164	77
TRUE	59	587

KNN

	FALSE	TRUE
FALSE	2187	135
TRUE	80	485

Tree

	FALSE	TRUE
FALSE	2150	80
TRUE	73	584

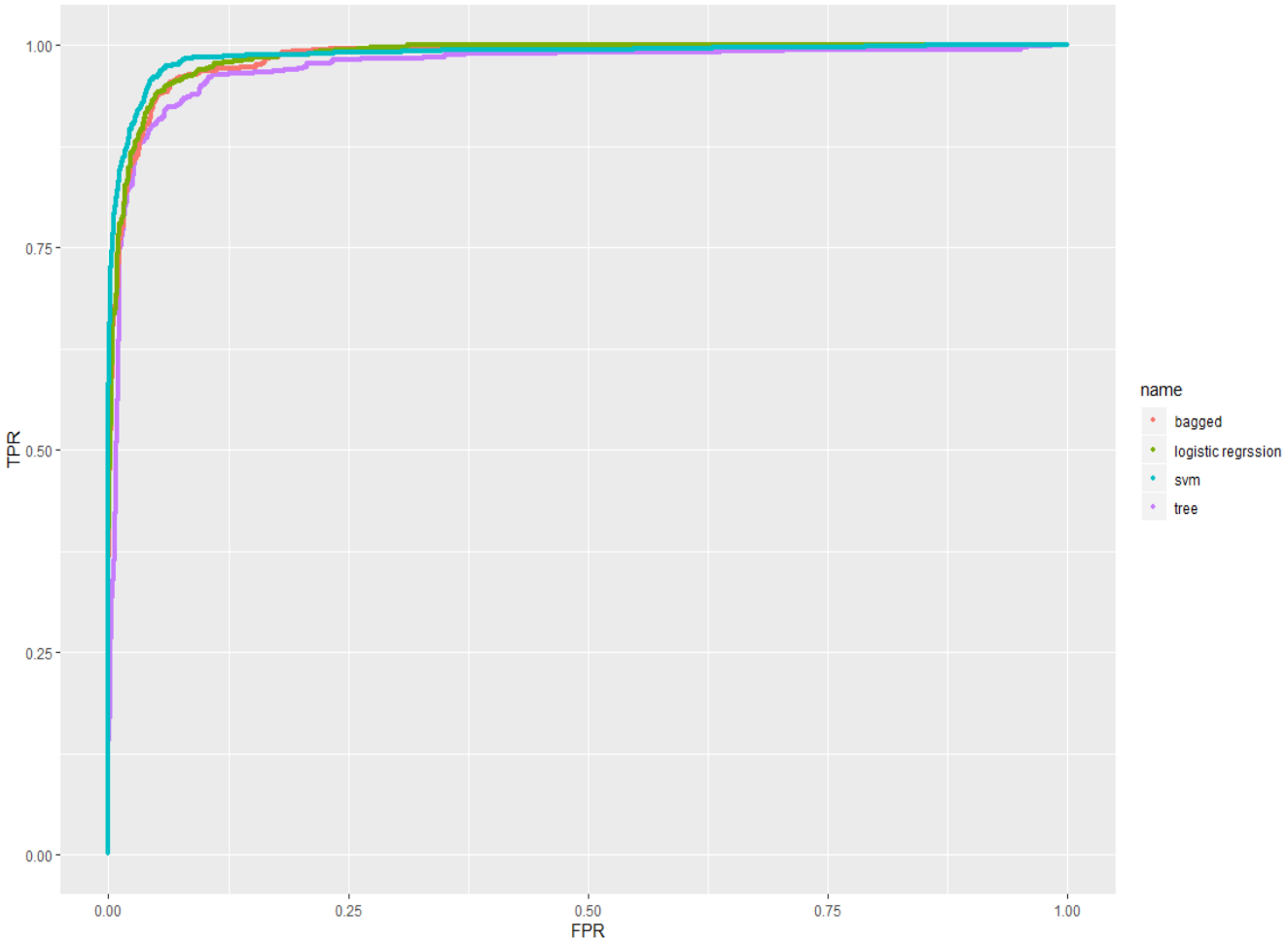
Bagged Tree

	FALSE	TRUE
FALSE	2147	86
TRUE	76	578

SVM

	FALSE	TRUE
FALSE	2181	82
TRUE	42	582

	logistic regression	knn	decision tree	bagged tree	svm
accuracy	0.947	0.94	0.928	0.948	0.95
precision	0.863	0.981	0.928	0.97	0.979
recall	0.894	0.945	0.978	0.96	0.963
f1	0.878	0.963	0.952	0.965	0.971



Accuracy on the test set was quite good for all of the models and did not vary by much. Our lowest, with the decision tree is 92% and our highest with our support vector machine is 95%-not significantly different. However, accuracy does not give us enough nuance to make any decisions on which model to select. When looking at precision, recall, and f1, the differences between the models is much larger—considering these measures will help us make a better informed decision.

First, let's consider precision. Precision is defined as the number of true positives divided by the number of true positives plus the number of false positives. Essentially telling us, when the model predicted true, how precise was it? For something like trying to predict Alzheimer's, precision is key, you do not want to tell someone that does not actually have Alzheimer's that they do, as that would be potentially devastating news. The least precise model was logistic regression, with precision of 86%. Our most precise model was KNN, with precision of 98%, which is quite good.

Second, let's look at recall. Recall is defined as true positives divided by true positives plus false negatives. This essentially tells us how good the model was a finding all of the positive instances in the data. This is also an important measure for our situation. Missing a case where someone does have Alzheimer's can have serious consequences, as it could delay the start of treatment. The difference here is that, while in the case of a false

positive follow-up testing can be done, in the case of a false negative we simply miss the presence of the disease. Taking that into account, we should care a bit more about recall than precision, but balance will be key. The best recall came from our decision tree, at 97%

Third, we can dive in the f1 measure. This measure is good if we want to try and find a good balance between precision and recall. Given that we are most concerned with recall, but still want to achieve a balance, this should be an informative measure. SVM achieved the highest f1 value, at 0.971.

Another important consideration is the ROC curve, which plots the true positive rate against the false positive rate. When you improve the true positive rate, you will typically see an increase in the false positive rate. This makes sense, as you can achieve a 100% hit rate for true positives if you simply classify all instances as true—but that is not a very helpful model. The ROC curve allows us to see this tradeoff, ideally you want to get as much of the true positives as you can without also seeing too many false positives. As can be seen in the chart, the SVM does the best here, gaining the most true positives before we see a larger increase in false positives. All of the models begin to merge as we move along the curve, the only model that does clearly worse than the others is our decision tree.

Finally, we can make a decision on which model to select. Since we are aiming for a balanced model, it should come as no surprise that the recommended model for this application is our SVM. It had the highest accuracy and f1. Additionally, it was only beat out by our decision tree in recall and KNN in precision, and both by a very small margin. Furthermore, when looking at our ROC chart, it has the best tradeoff between true positives and false positives. That said, there is still a lot to consider here. If we truly care about maximizing recall—since any cases marked as true can always have a follow-up visit—the decision tree may be a good bet. However, when taken holistically, the SVM simply performed the best.

I do think that there remain some areas that these models can be improved. The main one being some additional feature engineering. Without domain knowledge, I am unable to select meaningful interactions between variables, but given that we are working with many measures of volume and thickness I would not be surprised if there exists useful information in interactions. Additionally, more tinkering with the selected threshold value could likely result in precision and accuracy scores that better match the needs of the model users. And finally, the utilization of more of the available data. A sizeable portion of the data was removed as it had more than a quarter of the values missing. Results could probably be improved if methods to deal with those missing values were used and the data included in the modeling process.