

# PTP Classification for People who were hospitalized for TBI

Ethan Straub: ethan.straub@icloud.com and Ronnie Delgado: RonnieDelgado26@outlook.com

2025-06-29

## **Abstract:**

Traumatic brain injuries (TBI) can have adverse effects on people's mental health, and many patients post-TBI do not receive the proper preventative medical care from clinicians. The goal of this paper is to create a statistical learning model that accurately provides physicians with a probability that their patient develops a post traumatic psychopathological disorder. To deal with some of the missing values in the data, continuous variables were imputed through the Mice package in R. Due to the large number of predictors in the dataset, feature selection methods such as fitting an initial logistic Regression with an L1 penalty, and Random Forest were used to determine which predictors should be included or left out when fitting models. Four total models were then fit on the imputed and non-imputed versions of three different predictor subsets. These models are: logistic regression with an L1 penalty, random forests, XGBoost, and bayesian additive regression trees. 5-fold cross validation was used to tune hyperparameters and determine the best subset of predictors for each model. A train/test split was used to estimate model performance. We concluded that the Bayesian Additive Regression Trees using all predictor variables and imputations for missing values was the best performing model with a test accuracy of 0.885 and cross entropy loss of 0.292.

## **Introduction:**

Traumatic brain injuries (TBI) can have serious negative impacts on a person's mental health. TBI has been associated with many hospitalizations and deaths in the United States. Unfortunately, many people who have experienced TBI do not receive the medical care that is sometimes needed to prevent harmful psychological issues from developing. It is important for clinicians to understand which patients are at a higher risk for mental health problems so they can provide the proper preventative care to the patients who need it most. The main objective of this analysis is to provide clinicians with accurate probabilities of a patient developing any post traumatic psychopathology (PTP) disorders within 12 months of hospitalization among Colorado adults with TBI, given information on the patient's physical characteristics, medical history, and mental state. PTP is a broad term for mental health issues, including depression, anxiety, substance abuse, PTSD, and other more severe psychiatric disorders. This is an observational study with a sample size of 11,309 patients, all of whom were hospitalized for TBI between 2018 and 2022. For persons with more than one TBI-related admission, the data from the first hospitalization was used. There are several missing or unknown data points in the dataset. The data was sourced from De-identified Electronic Medical Record (EMR) data from UC Health, CO all-payer claims data, Access via Health Data Compass data warehouse, and data-use agreements & IRB approval.

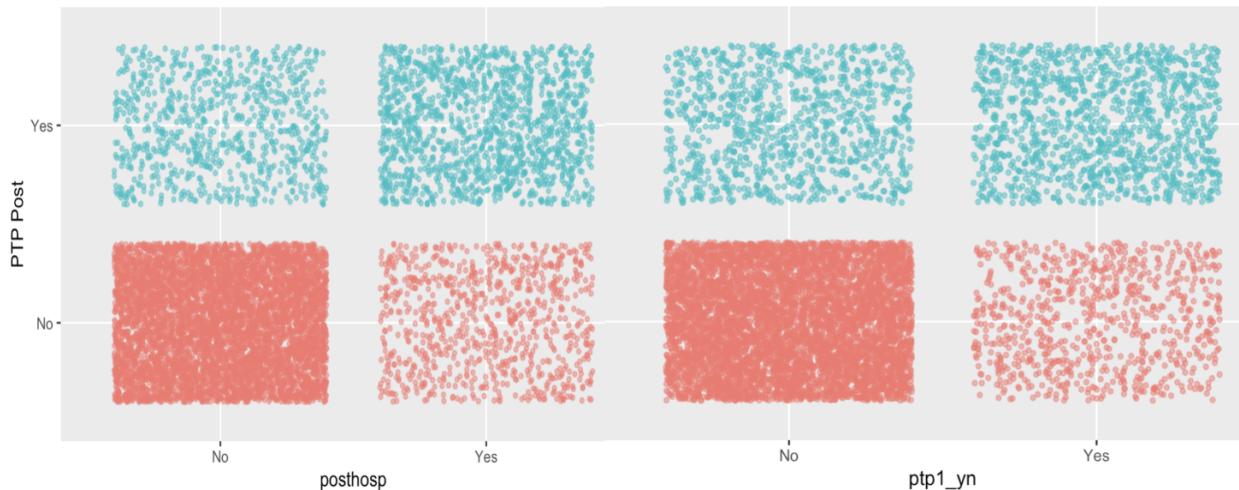
## **Summary Statistics:**

Table 1 (shown below) gives information on all predictor variables, the variable type, the levels of the variable, and the proportion of the variable that is missing or unknown in the training dataset. We can see that there are large proportions of missing values in the scores and scales that medical professionals may or may not evaluate.

Name	Variable	Type	Levels if Nominal or Range if Numeric	% Missing/Unknown/NA
age	Age	Numeric	[18,122]	0
race7	Race	Nominal	Unknown, Multi-Race, Pacific Islander, American Indian, Asian, Black, White	13
ethnic3	Ethnicity	Nominal	Unknow, Hispanic, Non-Hispanic	1.9
sex2	Sex	Nominal	Unknown, Female, Male	0
sig_other	Significant other	Nominal	Unknown, No significant other, Significant other	8.7
tobacco	Tobacco use	Nominal	Missing, Yes, No	7.7
alcohol	Alcohol use	Nominal	Missing, Yes, No	8.4
drugs	Elicit drug use	Nominal	Missing, Yes, No	9.7
MedianIncomeForZip	Median Income by Zipcode	Numeric	[11000,222930]	2
PercentAboveHighSchoolEducationForZip	% Above Highschool Ed by Zipcode	Numeric	[0,25]	0
PercentAboveBachelorsEducationForZip	% Higher Ed by Zipcode	Numeric	[0,8]	0
payertype	Payer category	Nominal	Work Comp/Self Pay, Medicare, Medicaid, VA/Indian Services, Commercial	0
tbiS02	TBI S02 - fracture	Numeric	[0,31]	0
tbiS06	TBI S06 - intracranial	Numeric	[0,30]	0
tbiS09	TBI S09 - unspecified	Numeric	[0,7]	0
ptp1_yn	PTP pre	Nominal	Yes, No	0
ptp2_yn	PTP index	Nominal	Yes, No	0
ptp0_yn	PTP pre or index	Nominal	Yes, No	0
ed_yn	ED visit	Nominal	Missing, Yes, No	23.4
icu	ICU stay	Nominal	Yes, No	0
delirium	Delirium	Nominal	Missing, Yes, No	53.4
agitated	Agitation	Nominal	Missing, Yes, No	2.2
lethargic	Lethargic	Nominal	Missing, Yes, No	2.2
comatose	Comatose	Nominal	Missing, Yes, No	2.2
disoriented	Disoriented	Nominal	Missing, Yes, No	17.7
gcs_min	Glasgow Coma Scale Minimum	Numeric	[3,15]	4.2
gcs_max	Glasgow Coma Scale Maximum	Numeric	[3,15]	4.2
adl_min	Activities of Daily Living Score Minimum	Numeric	[6,24]	35.3
adl_max	Activities of Daily Living Score Maximum	Numeric	[6,24]	35.3
mobility_min	Mobility Score Minimum	Numeric	[6,24]	20
mobility_max	Mobility Score Maximum	Numeric	[6,24]	20
los_total	Length of stay	Numeric	[0,413]	0
dc_setting	DC setting	Nominal	Died, Unknown, Hospice, Other, AMA, IRF, SNF, mental/psych, total	0.01
prehosp	Prior hospitalization	Nominal	Yes, No	0
posthosp	Rehospitalization	Nominal	Yes, No	0

**Table 1**

Predictor variables that appear to have a strong relationship with the response variable include posthosp (whether or not the patient was rehospitalized) and ptp1\_yn (whether or not the patient was diagnosed with TBI before the recorded brain injury). The relationships between these variables and the response variable is shown in figure 1 below. The response variable (PTP Post) is a binary variable indicating whether or not a patient developed PTP symptoms after their hospital visit for a TBI.

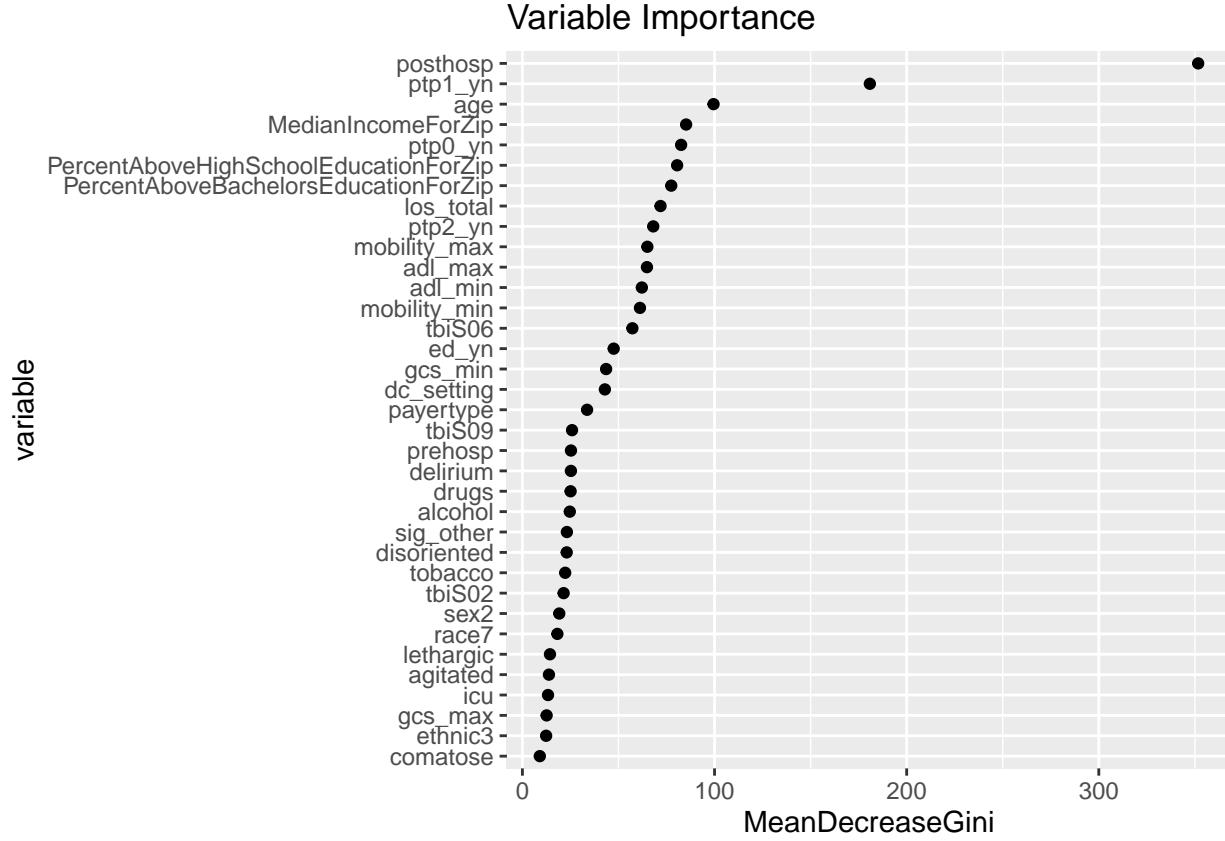


**Figure 1**

### Statistical Methods:

Before model fitting, some predictors were transformed to either summarize correlated variables, ensure that every observation is on the same scale, or to unskew predictors with heavily skewed right distributions, which is particularly beneficial for logistic regression. All values greater than 1 in the predictors “Percent above bachelor’s education for zip” and “Percent above high school education for zip” were divided by 100

to ensure that all observations are on a decimal scale, instead of some being on a decimal scale and some on a percentage scale. The predictors los\_total, tbiS02, tbiS06, and tbiS09 were log transformed. And new potential predictors were added by averaging the minimum and maximums for the glasgow coma scale, activities of daily living score, and mobility score. Instead of avoiding predictors with many missing values or removing patients who have missing values, we decided to treat missing/unknown nominal variables as their own separate category and impute the missing continuous variables using multivariate imputation by chained equations with the mice package in R. Due to the large number of predictors in the dataset, variable selection methods were conducted in order to avoid the potential issue of overfitting. The first method utilized was a Random Forest model with 100 trees using the entire predictor space (More details on how this model functions will be elaborated on once we get to model fitting but for right now we are merely using this as a means for feature selection.). From this, we were able to assess variable importance by looking at the mean decreasing gini and node impurity. The mean decreasing gini measures how much each variable reduces node impurity on average across all trees. The node impurity refers to how mixed the classes are at each node, where a very pure node is all one class, and an impure node is one of mixed classes.



**Figure 1**

We can see here that certain clinical variables such as rehospitalization and whether or not patients were diagnosed with PTP prior to their TBI, carry great importance when it came to making decisions on the purity of the node in the random forest. Meanwhile, variables that looked at the ethnicity of the patient and whether they were comatose did very little to increase the purity of the nodes. The second variable selection method that was used was fitting a logistic regression with an L1 penalty (otherwise known as the LASSO) over the entire predictor space. What this method focused on was applying a penalty to the model's loss function.

$$Loss = \sum_{i=1}^n (y_i - \hat{y}_i)^2 + \lambda \sum_{j=1}^p |\beta_j|$$

This in turn shrinks the coefficients of certain variables while setting the non impactful ones to zero. Although the results of this method were not exactly the same as the random forest we saw multiple similarities when it came to understanding the importance of certain predictors. To acknowledge these differences in results we created three different subsets of predictors to train our models on. The first was the inclusion of all of the variables with the transformed ones replacing their originals. The second subset excluded the following predictors: ethnicity, icu, agitated, lethargic, comatose, disoriented, the Glasgow Coma Scale minimum and maximum, the Activities of Daily Living score minimum and maximum, and the percent above bachelor's education for zipcode. The last subset excluded the same group of variables but kept the disoriented and the percent of above bachelor's education for zipcode variables. Going into model fitting, the first approach we took was one similar to our variable selection methods and that was the LASSO. Though this was not mentioned in variable selection it is important to note that we found our optimal lambda value for model fitting through cross-validation on a range of possible lambda values. As discussed earlier, the intuition behind this method was that with the optimal hyperparameter value for lambda, we could shrink our coefficients to see which variables were largely contributing to a patient having PTP. This model is fit with Ordinary Least Squares and still follows the assumptions held in linear regression which are linearity, independence of errors, and constant variance. The second model fitted was also a method used in variable selection which was the Random Forest, an improvement from decision trees that uses bootstrapped aggregated samples to fit multiple trees. Additionally, this method implements another layer of randomness by randomly selecting a subset of predictors for each split across each tree. The size of the subset we decided to use was the standard square root of the number of predictors to fit on 100 trees for this model. The strength of this model comes from using only a subset of variables at each split which reduces the correlation amongst trees.

The third model, XGBoost, is a tree based method that works by iteratively updating its predictions by fitting decision trees to the model's residuals and using each new tree's decision to update the previous prediction. XGBoost is known to give state of the art accuracy with tabular data, and has many hyperparameters. To tune these hyperparameters, we used a random search over plausible values. Models were trained using each randomly generated set of hyperparameters, and the model with the lowest cross entropy loss was chosen. The hyperparameters that were tuned include the learning rate, the maximum depth of a tree, the minimum loss reduction required to make a further partition on a leaf node, the minimum sum of instance weight needed in a child, the subsample ratio of the training instance, the subsample ratio of columns when constructing each tree, and the L2 regularization term on weights. To prevent overfitting, we used early stopping with cross validation using train/validation/test splits. For each training iteration, the current model is evaluated using cross entropy loss on the validation set. When the performance on that validation set hasn't improved for 10 iterations, the training process stops, and the current model is used to predict the labels of the test set. The chosen number of iterations was used with only a train/test split. With our train/test split, the training set is larger than when we use a train/validation/test split, which usually yields better performance since there is more data for the model to use in its training.

The final model is Bayesian Additive Regression Trees (abbreviated as BART), which is a nonparametric Bayesian method that sums predictions from many trees. BART is known to find well calibrated probabilities compared to non-statistical machine learning methods. To be honest, neither of us authors (Ethan or Ronnie) have a firm grasp of how BART works. However, we don't believe this lack of understanding should stop us from using it. In this study, we care about model performance rather than sound statistical assumptions and methods that would allow us to understand how predictors are related to the response. And complex models such as BART could give us the high performance that we want. However, a better understanding of BART may have allowed us to tune hyperparameters more effectively and ensure the MCMC sampling converged. Just like the XGBoost training, a random search with train/validation/test cross validation was done to tune hyperparameters, and a train/test split was used to estimate model performance. Two main evaluation metrics were considered: Accuracy and Cross Entropy Loss (Also known as Log Loss). Accuracy is percentage of classifications that the model assigns correctly, and cross entropy loss is a measure of how well the model's predicted probability distribution aligns with the true probability distribution. The formula for cross entropy loss is:

$$\frac{1}{n} \sum_{i=1}^n y_i(\log(p_i) + (1 - y_i)\log(1 - p_i))$$

In this formula,  $n$  is the sample size,  $y_i$  is the true probability of subject  $i$  developing PTP (0 or 1), and  $p_i$  is the model's predicted probability of subject  $i$  developing PTP. The closer to 0 the cross entropy loss is, the closer the model's predicted probability distribution aligns with the true probability distribution. It is important to evaluate the model's probabilities since these probabilities can give physicians more information than a binary classification. For example, predicted probabilities of 1% and 49% will lead to the same classification (using the traditional cutoff of 50%) even though the probabilities are considerably different and may lead to different, more well informed decisions on how a patient is treated.

### **Limitations and Alternatives:**

This analysis is limited by the missing nominal values in our dataset, which we did not impute. Only continuous variables were imputed. Additionally, including indicator variables that communicate whether or not a data point was imputed may have been beneficial to include since many missing values are likely MNAR (missing not at random), in which case the missingness of a data point may provide valuable information to the model. Values were imputed using only the chosen subset of predictors instead of the entire set of predictors. Imputations would likely have been more accurate if the program had access to all predictors when imputing. An alternative method of feature selection could be implementing a Principal Components Regression model to reduce the feature space instead of using the L1 penalty for regularization. It is also possible that a higher performing model could be achieved by using the other models (BART, XGBoost, and Random Forests) with principal components as features instead of a subset of predictors. Another form that could've been utilized was a Ridge Regression, which is very similar to the L1 penalty for regularization. However, in this method, no coefficients are shrunk to exactly zero. This method might have been preferred if all predictors did have some effect on our response instead of thinking it was only a few that had a real sizable effect on response. Hyperparameters were tuned using a subset of 9044 out of the 11309 total observations. Slightly higher performance could likely be achieved if all observations were used in the cross validation process for tuning hyperparameters.

### **Results and Conclusions**

Of the 6 potential feature matrices, the 3 imputed ones performed better than the 3 non-imputed ones for every model. The best performance for LASSO used the smallest set of features. The best performance for Random Forests used the second largest/second smallest set of features. And the best performance for XGBoost and BART used the largest set of features.. A table summarizing the performance of each model using imputed versions of their best performing sets of features is shown in table 2 below.

Models	Accuracy	True_Negative_Rate	True_Positive_Rate	Cross_Entropy_Loss
LASSO	0.872	0.943	0.622	0.302
Random Forest	0.877	0.951	0.620	1.889
XGBoost	0.879	0.939	0.669	0.301
BART	0.885	0.942	0.685	0.292

**Table 2**

Note that the cross entropy losses were estimated from cross validation from a subset of 9044 observations while all other metrics were estimated by training on the subset of 9044 observations and evaluating the model's predictions on a test set of the remaining 2265 observations. Additionally, we are not sure why the cross entropy loss for the random forest is abnormally large compared to the other 3 models.

The best performing model was determined to be Bayesian Additive Regression Trees using the largest subset of predictors with imputed missing values. BART had the lowest cross entropy loss and accuracy of any model. Bayesian Additive Regression Trees have an estimated True Negative Rate of 0.942 and an estimated True Positive Rate of 0.685. So if the model predicts that a patient will not develop PTP, we can be fairly confident that the model is correct. But if the model predicts that a patient will not develop PTP, we are less certain of the model's prediction. Due to the notable potential for error, this model should never be

used alone to predict the future mental health of hospital patients. Physicians can gain valuable information and intuitions on a patient's condition that computers do not have access to. The model should act as supplementary information to help physicians determine the risk of PTP in their patient's future.

## Author's Statements:

Both authors contributed to statistical analysis and writing for this report. Both authors read and approved the final report. Ronnie Delgado created the variable selection methods and implemented the LASSO and Random Forest Model. Ethan Straub completed the variable transformations, imputation, and cross validation splits. He also implemented the gradient boosting and bayesian additive regression trees models.

## References:

Chen T, He T, Benesty M, Khotilovich V, Tang Y, Cho H, Chen K, Mitchell R, Cano I, Zhou T, Li M, Xie J, Lin M, Geng Y, Li Y, Yuan J, Cortes D (2025). xgboost: Extreme Gradient Boosting. R package version 3.0.2.1, <https://github.com/dmlc/xgboost>. Graham, J. (2025). Practical tools for predicting risk of mental health issues following TBI. [PowerPoint slides]. Department of Occupational Therapy, Colorado State University. Sparapani R, Spanbauer C, McCulloch R (2021). “Nonparametric Machine Learning and Efficient Computation with Bayesian Additive Regression Trees: The BART R Package.” Journal of Statistical Software, 97(1), 1–66. doi:10.18637/jss.v097.i01

## Appendix:

```
knitr::opts_chunk$set(echo = FALSE)
knitr::include_graphics(normalizePath("variables_table.PNG"))
knitr::include_graphics(normalizePath("figure_1.PNG"))
set.seed(556)
library(ggplot2)
library(dplyr)
library(tidyverse)
library(glmnet)
library(randomForest)
library(ggcorrplot)
library(mice)
library(xgboost)
library(dbarts)
library(BART)
library(knitr)

train <- read.csv("tbistaa556_training.csv")
validate <- read.csv("tbistaa556_validate.csv")

# removing the died and subject id columns
train = train[,-1]
validate = validate[,-1]
train = train[,-36]
validate = validate[,-35]
# removing na values from train
trainClean = na.omit(train)
trainClean = trainClean[,-c(37,38,39,40,41,42,43,44,45)]

# fitting random forest
trainClean$ptp3_yn = as.factor(trainClean$ptp3_yn)
rf_fit <- randomForest(ptp3_yn ~ ., data = trainClean)
```

```

# plotting which predictors are the most important
data.frame(rf_fit$importance )%>%
  mutate(variable = rownames(rf_fit$importance)) %>%
  mutate(variable = factor(variable, levels = variable[order(MeanDecreaseGini)])) %>%
  ggplot() +
  geom_point(aes(MeanDecreaseGini, variable))+
  labs(title = "Variable Importance")
knitr:::include_graphics(normalizePath("table_2.PNG"))
library(ggplot2)
library(dplyr)
library(tidyverse)
library(glmnet)
library(randomForest)
library(ggcorrplot)
library(mice)
library(xgboost)
library(dbarts)
library(BART)
library(knitr)

train <- read.csv("tbistaa556_training.csv")
validate <- read.csv("tbistaa556_validate.csv")

# removing the died and subject id columns
train = train[,-1]
validate = validate[,-1]
train = train[,-36]
validate = validate[,-35]

# converting NA in the race column to unknown
indR = is.na(train$race7)
train$race7[indR] = 0

# Making new Data Frame with strings instead of numeric indicators

train1 = train
train1 = train1 %>% mutate(race7 = case_when(
  race7 == 0 ~ "Unknown",
  race7 == 1 ~ "Multi-race",
  race7 == 2 ~ "Pacific Islander",
  race7 == 3 ~ "American Indian",
  race7 == 4 ~ "Asian",
  race7 == 5 ~ "Black",
  race7 == 6 ~ "White"))

train1 = train1 %>% mutate(ethnic3 = case_when(
  ethnic3 == 0 ~ "Unknown",
  ethnic3 == 1 ~ "Hispanic",
  ethnic3 == 2 ~ "Non-Hispanic"))

train1 = train1 %>% mutate(sex2 = case_when(

```

```

sex2 == 0 ~ "Unknown",
sex2 == 1 ~ "Female",
sex2 == 2 ~ "Male"))

train1 = train1 %>% mutate(sig_other = case_when(
  sig_other == 0 ~ "Unknown",
  sig_other == 1 ~ "No Significant Other",
  sig_other == 2 ~ "Significant Other"))

train1 = train1 %>% mutate(tobacco = case_when(
  tobacco == 0 ~ "Missing",
  tobacco == 1 ~ "Yes",
  tobacco == 2 ~ "No"))

train1 = train1 %>% mutate(alcohol = case_when(
  alcohol == 0 ~ "Missing",
  alcohol == 1 ~ "Yes",
  alcohol == 2 ~ "No"))

train1 = train1 %>% mutate(drugs = case_when(
  drugs == 0 ~ "Missing",
  drugs == 1 ~ "Yes",
  drugs == 2 ~ "No"))

train1 = train1 %>% mutate(payertype = case_when(
  payertype == 1 ~ "Work Comp and Self Pay",
  payertype == 2 ~ "Medicaid",
  payertype == 3 ~ "Medicare",
  payertype == 4 ~ "VA and Indian Services",
  payertype == 5 ~ "Commercial"))

train1 = train1 %>% mutate(ptp1_yn = case_when(
  ptp1_yn == 1 ~ "Yes",
  ptp1_yn == 2 ~ "No"))

train1 = train1 %>% mutate(ptp2_yn = case_when(
  ptp2_yn == 1 ~ "Yes",
  ptp2_yn == 2 ~ "No"))

train1 = train1 %>% mutate(ptp0_yn = case_when(
  ptp0_yn == 1 ~ "Yes",
  ptp0_yn == 2 ~ "No"))

train1 = train1 %>% mutate(ed_yn = case_when(
  ed_yn == 0 ~ "Missing",
  ed_yn == 1 ~ "Yes",
  ed_yn == 2 ~ "No"))

train1 = train1 %>% mutate(icu = case_when(
  icu == 1 ~ "Yes",
  icu == 2 ~ "No"))

train1 = train1 %>% mutate(delirium = case_when(

```

```

delirium == 0 ~ "Missing",
delirium == 1 ~ "Yes",
delirium == 2 ~ "No"))

train1 = train1 %>% mutate(agitated = case_when(
  agitated == 0 ~ "Missing",
  agitated == 1 ~ "Yes",
  agitated == 2 ~ "No"))

train1 = train1 %>% mutate(lethargic = case_when(
  lethargic == 0 ~ "Missing",
  lethargic == 1 ~ "Yes",
  lethargic == 2 ~ "No"))

train1 = train1 %>% mutate(comatose = case_when(
  comatose == 0 ~ "Missing",
  comatose == 1 ~ "Yes",
  comatose == 2 ~ "No"))

train1 = train1 %>% mutate(disoriented = case_when(
  disoriented == 0 ~ "Missing",
  disoriented == 1 ~ "Yes",
  disoriented == 2 ~ "No"))

train1 = train1 %>% mutate(dc_setting = case_when(
  dc_setting == 0 ~ "Died",
  dc_setting == 1 ~ "Unknown",
  dc_setting == 2 ~ "Hospice",
  dc_setting == 3 ~ "Other",
  dc_setting == 4 ~ "AMA",
  dc_setting == 5 ~ "IRF",
  dc_setting == 6 ~ "SNF",
  dc_setting == 7 ~ "Mental/Psych",
  dc_setting == 8 ~ "Home",))

train1 = train1 %>% mutate(prehosp = case_when(
  prehosp == 1 ~ "Yes",
  prehosp == 2 ~ "No"))

train1 = train1 %>% mutate(posthosp = case_when(
  posthosp == 1 ~ "Yes",
  posthosp == 2 ~ "No"))

train1 = train1 %>% mutate(ptp3_yn = case_when(
  ptp3_yn == 0 ~ "No",
  ptp3_yn == 1 ~ "Yes"))

#Graphs of each predictor and response

graphs = function(train, numUnique = 8){
  for(col in 1:ncol(train)){
    uvs = sort(unique(train[,col]))
    # skip continuous variables

```

```

if(length(uvs) <= numUnique){
  next
}
colname = colnames(train)[col]
print(ggplot(train, aes(x = .data[[colname]], y = as.factor(ptp3_yn), color = as.factor(ptp3_yn))) +
  geom_jitter(size = 1, alpha = 0.5) +
  labs(x = colname, y = "PTP Post") + theme(legend.position = "none"))
}

#Transforming predictors

# performance summaries
train$summaryGCS = (train$gcs_min + train$gcs_max)/2
train$summaryADL = (train$adl_min + train$adl_max)/2
train$summaryMOB = (train$mobility_min + train$mobility_max)/2

# put ceiling on Percentages
train$PABEFZ = ifelse(train$PercentAboveBachelorsEducationForZip>1,
                      train$PercentAboveBachelorsEducationForZip/100,
                      train$PercentAboveBachelorsEducationForZip)
train$PAHSEFZ = ifelse(train$PercentAboveHighSchoolEducationForZip>1,
                      train$PercentAboveHighSchoolEducationForZip/100,
                      train$PercentAboveHighSchoolEducationForZip)

# log skewed right predictors
train$loglos_total = log(train$los_total+1)
train$logtbiS02 = log(train$tbiS02+1)
train$logtbiS06 = log(train$tbiS06+1)
train$logtbiS09 = log(train$tbiS09+1)
data = train

#Creating variable importance graphs with LASSO and Random Forest

set.seed(556)
# removing na values from train
trainClean = na.omit(train)
trainClean = trainClean[, -c(37, 38, 39, 40, 41, 42, 43, 44, 45)]

# fitting random forest
trainClean$ptp3_yn = as.factor(trainClean$ptp3_yn)
rf_fit <- randomForest(ptp3_yn ~ ., data = trainClean)

# plotting which predictors are the most important
data.frame(rf_fit$importance )%>%
  mutate(variable = rownames(rf_fit$importance)) %>%
  mutate(variable = factor(variable, levels = variable[order(MeanDecreaseGini)])) %>%
  ggplot() +
  geom_point(aes(MeanDecreaseGini, variable))+
  labs(title = "Variable Importance")-> Rf_image

ggsave("Rf_image.png")

```

```

# Using model.matrix to create a matrix of all predictors suitable for lasso GLM
X = model.matrix(ptp3_yn ~ ., data = trainClean)[,-36]
X = X[,-1]

y <- as.numeric(trainClean[,36])

# creating a vector of lambda values
lambda <- 10^seq(-4, -1, length.out = 100)

# creating lasso fit
lassofit = glmnet(X, y, alpha = 1, lambda = lambda, family = "binomial")

# doing 10 fold cv on lasso model for best lambda value
cv_lasso <- cv.glmnet(X, y, alpha = 1, lambda = lambda)

# Plotting cv test mse and lambda values
ggplot() +
  geom_line(aes(cv_lasso$lambda, cv_lasso$cvm)) +
  geom_point(aes(cv_lasso$lambda, cv_lasso$cvm)) +
  scale_x_log10()+
  labs(x = "Lambda", y = "CV Error", title = "CV Error Across Lambdas") -> lasso_image

ggsave("lasso_image.png")

#cv_lasso$lambda.min

# Getting coefficients at best lambda
best_lambda <- cv_lasso$lambda.min
best_model_coefs <- coef(cv_lasso, s = best_lambda)

Allnames = c("ptp3_yn", "age", "race7", "ethnic3", "sex2",
           "sig_other", "tobacco", "alcohol", "drugs",
           "MedianIncomeForZip",
           "PercentAboveHighSchoolEducationForZip",
           "PercentAboveBachelorsEducationForZip", "payertype", "tbiS02", "tbiS06",
           "tbiS09", "ptp1_yn", "ptp2_yn", "ptp0_yn", "ed_yn", "icu", "delirium",
           "agitated", "lethargic", "comatose", "disoriented", "gcs_min",
           "gcs_max", "adl_min", "adl_max", "mobility_min", "mobility_max",
           "los_total", "dc_setting", "prehosp", "posthosp", "summaryGCS",
           "summaryADL", "summaryMOB", "loglos_total", "PABEFZ", "PAHSEFZ",
           "logtbiS02", "logtbiS06", "logtbiS09")

names1 = c("ptp3_yn", "age", "race7", "ethnic3", "sex2", "sig_other",
          "tobacco", "alcohol", "drugs", "MedianIncomeForZip",
          "payertype", "ptp1_yn", "ptp2_yn", "ptp0_yn", "ed_yn",
          "icu", "delirium", "agitated", "lethargic",
          "comatose", "disoriented", "dc_setting", "prehosp",
          "posthosp", "gcs_min", "gcs_max", "adl_min", "adl_max",
          "mobility_min", "mobility_max", "loglos_total", "PABEFZ",
          "PAHSEFZ", "logtbiS02", "logtbiS06", "logtbiS09")

```

```

names2 = c("ptp3_yn", "age", "race7", "sex2", "sig_other", "tobacco",
         "alcohol",
"drugs", "MedianIncomeForZip", "payertype", "ptp1_yn", "ptp2_yn",
"ptp0_yn", "ed_yn", "delirium", "dc_setting", "prehosp", "posthosp",
"summaryADL", "mobility_min", "mobility_max", "loglos_total",
"PAHSEFZ", "logtbiS02", "logtbiS06", "logtbiS09")

names3 = c("ptp3_yn", "age", "race7", "sex2", "sig_other", "tobacco",
         "alcohol",
"drugs", "MedianIncomeForZip", "payertype", "ptp1_yn", "ptp2_yn",
"ptp0_yn", "ed_yn", "delirium", "disoriented", "dc_setting", "prehosp",
"posthosp", "summaryADL", "mobility_min", "mobility_max",
"loglos_total", "PAHSEFZ", "PABEFZ", "logtbiS02", "logtbiS06", "logtbiS09")

#Function for cross validation splits

kfoldData = function(impute = FALSE, colnames, data, seed = 556, k = 5, printImputer = FALSE){
  set.seed(seed)

  # shuffling rows of data frame
  ind = sample(nrow(data))
  data = data[ind,]

  # selecting columns in colnames
  dta = data %>% select(all_of(colnames))

  # initializing list to return
  listReturn = list()

  # convert to factors
  for (col in names(dta)) {
    if(col == "ptp3_yn"){
      next
    }
    if (length(unique(dta[[col]])) < 8) {
      dta[[col]] = as.factor(dta[[col]])
    }
  }
  dta$dc_setting = as.factor(dta$dc_setting)
  mf = model.frame(~.-1, data = dta, na.action = na.pass)
  dta = model.matrix(~.-1, data = mf)
  dta = as.data.frame(dta)

  # If not imputing, remove rows with NA
  if(impute == FALSE){
    ind = complete.cases(dta)
    dta = dta[ind,]
  }

  # Splitting features and labels
  X = dta %>% select(-"ptp3_yn")
  Y = dta %>% select("ptp3_yn")
}

```

```

# Number of rows per fold
numPerFold = round(nrow(dta)/k)

#loop through each split
for(i in 1:k){

  # Get indices of train and test set for the ith iteration
  startInd = (i-1)*numPerFold+1
  endInd = ifelse(i == k, nrow(dta), i*numPerFold)
  testInd = startInd:endInd

  # split features and labels into train and test sets
  Xtest = X[testInd,]
  Xtrain = X[-testInd,]
  Ytest = Y[testInd,]
  Ytrain = Y[-testInd,]

  # imputing
  if(impute == TRUE){

    # imputing Xtrain
    imputeObj = mice(Xtrain, m = 1, seed = seed, printFlag = printImputer)
    #print(imputeObj$loggedEvents)
    Xtrain = complete(imputeObj, 1)
    # imputing Xtest using the already imputed Xtrain
    full = rbind(Xtrain, Xtest)
    fullImputeObj = mice(full, m = 1, maxit = 1, seed = seed, printFlag = printImputer)
    fullImputed = complete(fullImputeObj, 1)
    testStart = nrow(Xtrain)+1
    Xtest = fullImputed[testStart:nrow(fullImputed),]
  }

  # Append to the list that is returned
  listReturn[[4*i-3]] = Xtrain
  listReturn[[4*i-2]] = Ytrain
  listReturn[[4*i-1]] = Xtest
  listReturn[[4*i]] = Ytest
}

# naming elements in the list
for(i in 1:k){
  start = 4*i-3
  end = start + 3
  names(listReturn)[start:end] = c(paste0("Xtrain",i), paste0("Ytrain",i),
                                   paste0("Xtest",i), paste0("Ytest",i))
}
return(listReturn)
}

CrossEntropyLoss = function(probabilities,truth){
  probabilities <- pmin(pmax(probabilities, 1e-15), 1 - 1e-15)
  return(-mean((truth*log(probabilities)) + (1-truth)*log(1-probabilities)))
}

#Get splits for Imputed/not imputed for all 3 sets of predictors

```

```

noImp1 = kfoldData(colnames = names1, data = data, impute = FALSE, k = 5)
noImp2 = kfoldData(colnames = names2, data = data, impute = FALSE, k = 5)
noImp3 = kfoldData(colnames = names3, data = data, impute = FALSE, k = 5)
Imp1 = kfoldData(colnames = names1, data = data, impute = TRUE, k = 5)
Imp2 = kfoldData(colnames = names2, data = data, impute = TRUE, k = 5)
Imp3 = kfoldData(colnames = names3, data = data, impute = TRUE, k = 5)

#Lasso GLM Results

LASSO = function(kfold_list, lambda = 0.00231013, seed = 556){

error_rates <- numeric(5)
cross_entropies_losses <- numeric(5)
true_positive_rates <- numeric(5)
true_negative_rates <- numeric(5)
all_predictions <- c()
all_truths <- c()
lasso_coefs <- list()

# looping through each kfold split
for (i in 1:5) {
  Xtrain <- as.matrix(kfold_list[[paste0("Xtrain", i)]])
  Ytrain <- as.vector(kfold_list[[paste0("Ytrain", i)]])
  Xtest <- as.matrix(kfold_list[[paste0("Xtest", i)]])
  Ytest <- as.vector(kfold_list[[paste0("Ytest", i)]])

  lasso_fit <- glmnet(Xtrain, Ytrain, alpha = 1, lambda = 0.002009233, family = binomial)

  pred_probs <- predict(lasso_fit, s = 0.002009233, newx = Xtest, type = "response")

  predictions <- ifelse(pred_probs > 0.5, 1, 0)

  error_rates[i] <- mean(predictions != Ytest)

  cross_entropies_losses[i] <- CrossEntropyLoss(pred_probs, Ytest)

  TP <- sum(predictions == 1 & Ytest == 1)
  TN <- sum(predictions == 0 & Ytest == 0)
  FP <- sum(predictions == 1 & Ytest == 0)
  FN <- sum(predictions == 0 & Ytest == 1)

  true_positive_rates[i] <- ifelse((TP + FN) > 0, TP / (TP + FN))
  true_negative_rates[i] <- ifelse((TN + FP) > 0, TN / (TN + FP))

}

mean_error_rate <- mean(error_rates)
mean_cross_entropy_loss <- mean(cross_entropies_losses)
mean_tp_rate <- mean(true_positive_rates)
mean_tn_rate <- mean(true_negative_rates)

return( kable(data.frame(mean_error_rate,mean_cross_entropy_loss,
  mean_tn_rate, mean_tp_rate), digits = 3))
}

```

```

}

# LASSO(Imp1)
# LASSO(Imp2)
# LASSO(Imp3)

#Random Forest Results

RF = function(kfold_list, seed = 556){

error_rates <- numeric(5)
cross_entropies_losses <- numeric(5)
true_positive_rates <- numeric(5)
true_negative_rates <- numeric(5)

for (i in 1:5) {
  Xtrain <- kfold_list[[paste0("Xtrain", i)]]
  Ytrain <- kfold_list[[paste0("Ytrain", i)]]
  Xtest <- kfold_list[[paste0("Xtest", i)]]
  Ytest <- kfold_list[[paste0("Ytest", i)]]

  train_df <- data.frame(Xtrain, ptp3_yn = as.factor(Ytrain))
  test_df <- data.frame(Xtest)
  Ytest <- as.factor(Ytest)

  fit <- randomForest(ptp3_yn ~ ., data = train_df, )

  pred_probs <- predict( fit, newdata = test_df, type = "prob")[, "1"]

  predictions <- ifelse(pred_probs > 0.5, 1, 0)

  error_rates[i] <- mean(predictions != Ytest)

  cross_entropies_losses[i] <- CrossEntropyLoss(pred_probs, as.numeric(Ytest))

  TP <- sum(predictions == 1 & Ytest == 1)
  TN <- sum(predictions == 0 & Ytest == 0)
  FP <- sum(predictions == 1 & Ytest == 0)
  FN <- sum(predictions == 0 & Ytest == 1)

  true_positive_rates[i] <- ifelse((TP + FN) > 0, TP / (TP + FN))
  true_negative_rates[i] <- ifelse((TN + FP) > 0, TN / (TN + FP))
}

mean_error_rate <- mean(error_rates)
mean_cross_entropy_loss <- mean(cross_entropies_losses)
mean_tp_rate <- mean(true_positive_rates)
mean_tn_rate <- mean(true_negative_rates)

return( kable(data.frame(mean_error_rate,mean_cross_entropy_loss,

```

```

        mean_tn_rate, mean_tp_rate), digits = 3)
    )
}

# RF(Imp1)
# RF(Imp2)
# RF(Imp3)

#Gradient Boosting Results

BOOST = function(kfold_list, nsims = 1, nrounds = 10000, verbose = 0,
                 k = 5, early_stopping_rounds = 10){

  # get all parameters
  boostParams = data.frame(eta = numeric(),
                           max_depth = numeric(),
                           min_child_weight = numeric(),
                           subsample = numeric(),
                           colsample_bytree = numeric(), gamma = numeric(), lambda = numeric())
  for(i in 1:nsims){
    eta = rgamma(1,1,50) + 0.005
    max_depth = rpois(1,5) + 1
    min_child_weight = runif(1,1,10)
    subsample = runif(1,0.5,1)
    colsample_bytree = runif(1,0.5,1)
    gamma = runif(1,0,2.1)
    lambda = runif(1,0,10)
    boostParams[i,] = c(eta, max_depth, min_child_weight, subsample,
                       colsample_bytree, gamma, lambda)
  }

  # initialize data frame for results
  results = data.frame(id = numeric(), eta = numeric(),
                        max_depth = numeric(),
                        min_child_weight = numeric(),
                        subsample = numeric(),
                        colsample_bytree = numeric(), gamma = numeric(), lambda = numeric(),
                        bestIter = numeric(), CEloss = numeric(), accuracy = numeric(),
                        TPR = numeric(), TNR = numeric())

  # loop through each row of parameters
  for(row in 1:nrow(boostParams)){
    params = as.list(boostParams[row,])

    # loop through each fold
    for (i in 1:k){

      # separate train, validate, and test sets
      Xt = as.matrix(kfold_list[[paste0("Xtrain", i)]])
      Yt = as.vector(kfold_list[[paste0("Ytrain", i)]])
      # The original train is split into train and validation with roughly
      # 80% in train and 20% in validate
    }
  }
}

```

```

cutoff = round(nrow(Xt)*0.8)
c1 = cutoff+1
Xtrain = Xt[1:cutoff,]
Ytrain = Yt[1:cutoff]
Xvalid = Xt[c1:nrow(Xt),]
Yvalid = Yt[c1:nrow(Xt)]
Xtest = as.matrix(kfold_list[[paste0("Xtest", i)]])
Ytest = as.vector(kfold_list[[paste0("Ytest", i)]])

#convert train and validate into xgb matrices
trainXG = xgb.DMatrix(data = Xtrain, label = Ytrain)
validXG = xgb.DMatrix(data = Xvalid, label = Yvalid)
wl = list(train = trainXG, eval = validXG)

# train with early stopping using the validation set.
trainFit = xgb.train(params = params, data = trainXG, watchlist = wl,
                     nrounds = nrounds, verbose = verbose,
                     early_stopping_rounds = early_stopping_rounds)

# predict using the test set
probabilities = predict(trainFit, Xtest, iteration_range = c(0, trainFit$best_iteration))
classes = ifelse(probabilities > 0.5, 1, 0)

# calculate performance metrics
ceL = CrossEntropyLoss(probabilities, Ytest)
acc = mean(classes == Ytest)
TP = sum(classes == 1 & Ytest == 1)
TN = sum(classes == 0 & Ytest == 0)
FP = sum(classes == 1 & Ytest == 0)
FN = sum(classes == 0 & Ytest == 1)
TPR = ifelse((TP + FN) > 0, TP / (TP + FN))
TNR = ifelse((TN + FP) > 0, TN / (TN + FP))

# row bind performance metrics to the data frame
results = rbind(results, list(id = row, eta = params$eta,
                               max_depth = params$max_depth,
                               min_child_weight = params$min_child_weight,
                               subsample = params$subsample,
                               colsample_bytree = params$colsample_bytree,
                               gamma = params$gamma, lambda = params$lambda,
                               bestIter = trainFit$best_iteration, accuracy = acc,
                               CELoss = ceL, TPR = TPR, TNR = TNR))
}

}
return(results)
}

#boostnoImp1 = BOOST(noImp1, nsims = 5)
#boostnoImp2 = BOOST(noImp2, nsims = 5)
#boostnoImp3 = BOOST(noImp3, nsims = 5)
#boostImp1 = BOOST(Imp1, nsims = 5)
#boostImp2 = BOOST(Imp2, nsims = 5)
#boostImp3 = BOOST(Imp3, nsims = 5)

```

```

#boostImp1Big = BOOST(Imp1, nsims = 100)
#boostImp3Big = BOOST(Imp3, nsims = 100)

bigboi = rbind(boostImp1Big, boostImp3Big)
predictorSubset = as.factor(c(rep(1,250), rep(3,250)))
bigboi$predictorSubset = predictorSubset
ggplot(bigboi, aes(x = CELoss, y = accuracy, color = predictorSubset)) + geom_point() +
  labs(title = "Accuracy vs Cross Entropy Loss", x = "Cross Entropy Loss", y = "Accuracy")

smalldf = data.frame(eta = numeric(), max_depth = numeric(),
min_child_weight = numeric(),
subsample = numeric(),
colsample_bytree = numeric(), gamma = numeric(),
lambda = numeric(), bestIter = numeric(), MeanCELoss = numeric(),
Meanaccuracy = numeric(), uniqueid = numeric())

bigboi$uniqueid = paste(bigboi$id,bigboi$predictorSubset)

count = 0
for(id in unique(bigboi$uniqueid)){
  count = count+5
  subset = bigboi[bigboi$uniqueid==id,]
  smalldf[count/5,] = c(subset$eta[1], subset$max_depth[1], subset$min_child_weight[1],
                        subset$subsample[1], subset$colsample_bytree[1], subset$gamma[1],
                        subset$lambda[1],
                        mean(subset$bestIter), mean(subset$CELoss), mean(subset$accuracy),
                        subset$uniqueid[1])
}

best = smalldf[smalldf$MeanCELoss<0.305,]

ggplot(bigboi, aes(x = eta, y = CELoss)) + geom_point()

#boostImp1Big = BOOST(Imp1, nsims = )
#boostImp3Big = BOOST(Imp3, nsims = 2)

bigboi1 = rbind(boostImp1Big, boostImp3Big)
smalldf1 = data.frame(eta = numeric(), max_depth = numeric(),
min_child_weight = numeric(), subsample = numeric(),
colsample_bytree = numeric(),
gamma = numeric(), lambda = numeric(), bestIter = numeric(),
MeanCELoss = numeric(), Meanaccuracy = numeric(), uniqueid = numeric())

bigboi1$uniqueid = paste(bigboi1$id,bigboi1$predictorSubset)

count = 0
for(id in unique(bigboi$uniqueid)){
  count = count+5
  subset = bigboi1[bigboi1$uniqueid==id,]
  smalldf1[count/5,] = c(subset$eta[1], subset$max_depth[1],
                        subset$min_child_weight[1], subset$subsample[1],
                        subset$colsample_bytree[1], subset$gamma[1],
                        subset$lambda[1], mean(subset$bestIter),

```

```

        mean(subset$CELoss), mean(subset$accuracy),
        subset$uniqueid[1])
}

best = smalldf[smalldf$Meanaccuracy>0.8,]
bartt = function(kfold_list, nsims = 1, seed = 556, numfolds = 5){

  bartParams = expand.grid(k = numeric(), power = numeric(), base = numeric(), ntree = numeric())
  results = data.frame(id = numeric(), k = numeric(), power = numeric(),
  base = numeric(), ntree = numeric())
  for(i in 1:nsims){
    k = runif(1,1,1.5)
    power = runif(1,1,2.5)
    base = runif(1,0.7,1)
    ntree = round(runif(1,99.6,500.4))
    bartParams[i,] = c(k,power,base,ntree)
  }
  # loop through each row of parameters
  for(row in 1:nrow(bartParams)){
    params = as.list(bartParams[row,])

    # loop through each fold
    for (i in 1:numfolds){
      # separate train, validate, and test sets
      Xt = as.matrix(kfold_list[[paste0("Xtrain", i)]])
      Yt = as.vector(kfold_list[[paste0("Ytrain", i)]])
      # The original train is split into train and validation with roughly
      #80% in train and 20% in validate
      cutoff = round(nrow(Xt)*0.8)
      c1 = cutoff+1
      Xtrain = Xt[1:cutoff,]
      Ytrain = Yt[1:cutoff]
      Xvalid = Xt[c1:nrow(Xt),]
      Yvalid = Yt[c1:nrow(Xt)]
      Xtest = as.matrix(kfold_list[[paste0("Xtest", i)]])
      Ytest = as.vector(kfold_list[[paste0("Ytest", i)]])
      trainFit = bart(x.train = Xtrain, y.train = Ytrain, k = params$k,
      power = params$power, base = params$base, ntree = params$ntree,
      keeptrees = TRUE)
      # predict using the test set and validation set
      posteriorValidate = predict(trainFit, newdata = Xvalid)
      probsValidate = colMeans(posteriorValidate)
      classesValidate = ifelse(probsValidate > 0.5, 1, 0)

      posteriorTest = predict(trainFit, newdata = Xtest)
      probsTest = colMeans(posteriorTest)
      classesTest = ifelse(probsTest > 0.5, 1, 0)

      # calculate performance metrics
      ceLValidate = CrossEntropyLoss(probsValidate, Yvalid)
      ceLTest = CrossEntropyLoss(probsTest, Ytest)
      accValidate = mean(classesValidate == Yvalid)
      accTest = mean(classesTest == Ytest)
    }
  }
}

```

```

TP = sum(classesTest == 1 & Ytest == 1)
TN = sum(classesTest == 0 & Ytest == 0)
FP = sum(classesTest == 1 & Ytest == 0)
FN = sum(classesTest == 0 & Ytest == 1)
TPR = ifelse((TP + FN) > 0, TP / (TP + FN))
TNR = ifelse((TN + FP) > 0, TN / (TN + FP))

# row bind performance metrics to the data frame
results = rbind(results, list(id = row, k = params$k, power = params$power,
                             base = params$base, ntree = params$ntree,
                             CElossValidate = ceLValidate, CElossTest = ceLTest,
                             accValidate = accValidate, accTest = accTest,
                             TPR = TPR, TNR = TNR))
}

}
return(results)
}

#bartnoImp1 = bartt(kfold_list = noImp1, nsims = 5)
#bartnoImp2 = bartt(kfold_list = noImp2, nsims = 5)
#bartnoImp3 = bartt(kfold_list = noImp3, nsims = 5)
#bartImp2 = bartt(kfold_list = Imp2, nsims = 5)
#bartImp1 = bartt(kfold_list = Imp1, nsims = 50)
#bartImp3 = bartt(kfold_list = Imp3, nsims = 50)
bigbart = rbind(bartImp1,bartImp3)
smalldf = data.frame(k = numeric(), power = numeric(),
                      base = numeric(), ntree = numeric(),
                      MeanCELossValidate = numeric(), MeanCELossTest = numeric(),
                      MeanaccValidate = numeric(), MeanaccTest = numeric(), uniqueid = numeric())

predictorSubset = as.factor(c(rep(1,10), rep(3,10)))
bigbart$predictorSubset = predictorSubset
bigbart$uniqueid = paste(bigbart$id,bigbart$predictorSubset)

count = 0
for(id in unique(bigbart$uniqueid)){
  count = count+5
  subset = bigbart[bigbart$uniqueid==id,]
  smalldf[count/5,] = c(subset$k[1], subset$power[1],
                        subset$base[1],
                        subset$ntree[1], mean(subset$CELossValidate),
                        mean(subset$CElossTest), mean(subset$accValidate),
                        mean(subset$accTest), subset$uniqueid[1])
}
best = smalldf[smalldf$MeanCELossValidate<0.29,]

predictTest = function(impute = FALSE, colnames, data, Y, seed = 556, printImputer = FALSE){
  set.seed(seed)

  # selecting columns in colnames
  dta = data %>% select(all_of(colnames))
}

```

```

# initializing list to return
listReturn = list()

# convert to factors
for (col in names(dta)) {
  if (length(unique(dta[[col]])) < 8) {
    dta[[col]] = as.factor(dta[[col]])
  }
}
dta$dc_setting = as.factor(dta$dc_setting)
#mf = model.frame(~.-1, data = dta, na.action = na.pass)
#dta = model.matrix(~.-1, data = mf, na.action = na.pass)
#dta = as.data.frame(dta)
#return(dta)
# If not imputing, remove rows with NA
if(impute == FALSE){
  ind = complete.cases(dta)
  dta = dta[ind,]
}

# Splitting features and labels
X = dta

# split features and labels into train and test sets
train = X[1:9044,]
validate = X[9045:11309,]

# imputing
if(impute == TRUE){

  # imputing Xtrain
  imputeObj = mice(train, m = 1, seed = seed, printFlag = printImputer)
  #print(imputeObj$loggedEvents)
  train = complete(imputeObj, 1)
  # imputing Xtest using the already imputed Xtrain
  full = rbind(train, validate)
  fullImputeObj = mice(full, m = 1, maxit = 1, seed = seed, printFlag = printImputer)
  fullImputed = complete(fullImputeObj, 1)
  testStart = nrow(train)+1
  validate = fullImputed[testStart:nrow(fullImputed),]
}

Xtrain = train
Ytrain = Y
Xvalidate = validate
# Append to the list that is returned
listReturn[[1]] = Xtrain
listReturn[[2]] = Ytrain
listReturn[[3]] = Xvalidate
return(listReturn)
}

CrossEntropyLoss = function(probabilities, truth){
  probabilities <- pmin(pmax(probabilities, 1e-15), 1 - 1e-15)
}

```

```

    return(-mean((truth*log(probabilities)) + (1-truth)*log(1-probabilities)))
}

Imp1 = predictTest(colnames = names1, data = data, Y = trainY, impute = TRUE)

Xtrain = Imp1[[1]]
Xtrain = model.matrix(~., data = Xtrain)
Ytrain = Imp1[[2]]
Xvalidate = Imp1[[3]]
Xvalidate = model.matrix(~., data = Xvalidate)

verbose = FALSE
nrounds = 259
params = list(objective = "binary:logistic", eta = 0.028635479810182,
              max_depth = 4, subsample = 0.809286485658959,
              colsample_bytree = 0.711043330607936,
              min_child_weight = 8.33287097513676,
              gamma = 0.595529325027019, lambda = 4.07533963210881)

trainXG1 = xgb.DMatrix(data = Xtrain, label = Ytrain)

trainFit1 = xgb.train(params = params, data = trainXG1, nrounds = nrounds, verbose = verbose)

# predict using the test set
probabilities1 = predict(trainFit1, Xvalidate)

classes1 = ifelse(probabilities1 > 0.5, 1, 0)

k = 1
power = 1.6752053586533
base = 0.821862926776521
ntree = 120
trainFit = bart(x.train = Xtrain, y.train = Ytrain, k = k,
                 power = power, base = base, ntree = ntree,
                 keeptrees = TRUE)

posteriorTest = predict(trainFit, newdata = Xvalidate)
probsTest = colMeans(posteriorTest)
classesTest = ifelse(probsTest > 0.5, 1, 0)

predictionsXGBoost = classes1
predictionsBART = classesTest
subj_id = v$subj_id

df = data.frame(subj_id, predictionsXGBoost, predictionsBART)

```