```
#load mlbench library and e1071 (needed later)
library(mlbench)
#set working directory if needed (modify path as needed)
setwd("C:/Users/Kailash/Documents/NaiveBayes")
#load HouseVotes84 dataset
data("HouseVotes84")

#barplots for specific issue
plot(as.factor(HouseVotes84[,2]))
title(main="Votes cast for issue", xlab="vote", ylab="# reps")
#by party
plot(as.factor(HouseVotes84[HouseVotes84$Class=='republican',2]))
title(main="Republican votes cast for issue 1", xlab="vote", ylab="# reps")
plot(as.factor(HouseVotes84[HouseVotes84$Class=='democrat',2]))
title(main="Democrat votes cast for issue 1", xlab="vote", ylab="# reps")

#Functions needed for imputation
#function to return number of NAs by vote and class (democrat or republican)
na_by_col_class <- function (col,cls){return(sum(is.na(HouseVotes84[,col]) &
HouseVotes84$Class==cls))}
#function to compute the conditional probability that a member of a party will cast
a 'yes' vote for
#a particular issue. The probability is based on all members of the party who
#actually cast a vote on the issue (ignores NAs).
p_y_col_class <- function(col,cls){
sum_y<-sum(HouseVotes84[,col]=='y' & HouseVotes84$Class==cls,na.rm = TRUE)
sum_n<-sum(HouseVotes84[,col]=='n' & HouseVotes84$Class==cls,na.rm = TRUE)
return(sum_y/(sum_y+sum_n))}

#impute missing values.
for (i in 2:ncol(HouseVotes84)) {
if(sum(is.na(HouseVotes84[,i])>0)) {
c1 <- which(is.na(HouseVotes84[,i])& HouseVotes84$Class=='democrat',arr.ind = TRUE)
c2 <- which(is.na(HouseVotes84[,i])& HouseVotes84$Class=='republican',arr.ind =
TRUE)
HouseVotes84[c1,i] <-
ifelse(runif(na_by_col_class(i,'democrat'))<p_y_col_class(i,'democrat'),'y','n')
HouseVotes84[c2,i] <-
ifelse(runif(na_by_col_class(i,'republican'))<p_y_col_class(i,'republican'),'y','n'
)}
}

#divide into test and training sets
#create new col "train" and assign 1 or 0 in 80/20 proportion via random uniform
dist
HouseVotes84[,"train"] <- ifelse(runif(nrow(HouseVotes84))<0.80,1,0)
#get col number of train / test indicator column (needed later)
trainColNum <- grep("train",names(HouseVotes84))
#separate training and test sets and remove training column before modeling
trainHouseVotes84 <- HouseVotes84[HouseVotes84$train==1,-trainColNum]
testHouseVotes84 <- HouseVotes84[HouseVotes84$train==0,-trainColNum]

#train model
nb_model <- naiveBayes(Class~.,data = trainHouseVotes84)

nb_model
summary(nb_model)
str(nb_model)

#...and the moment of reckoning
nb_test_predict <- predict(nb_model,testHouseVotes84[,-1])
#confusion matrix
table(pred=nb_test_predict,true=testHouseVotes84$Class)
```

```
#fraction of correct predictions
mean(nb_test_predict==testHouseVotes84$Class)

#function to create, run and record model results
nb_multiple_runs <- function(train_fraction,n){
fraction_correct <- rep(NA,n)
for (i in 1:n){
HouseVotes84[,"train"] <- ifelse(runif(nrow(HouseVotes84))<train_fraction,1,0)
trainColNum <- grep("train",names(HouseVotes84))
trainHouseVotes84 <- HouseVotes84[HouseVotes84$train==1,-trainColNum]
testHouseVotes84 <- HouseVotes84[HouseVotes84$train==0,-trainColNum]
nb_model <- naiveBayes(Class~.,data = trainHouseVotes84)
nb_test_predict <- predict(nb_model,testHouseVotes84[,-1])
fraction_correct[i] <- mean(nb_test_predict==testHouseVotes84$Class)
}
return(fraction_correct)
}

#20 runs, 80% of data randomly selected for training set in each run
fraction_correct_predictions <- nb_multiple_runs(0.8,20)
fraction_correct_predictions

#summary of results
summary(fraction_correct_predictions)

#standard deviation
sd(fraction_correct_predictions)
```