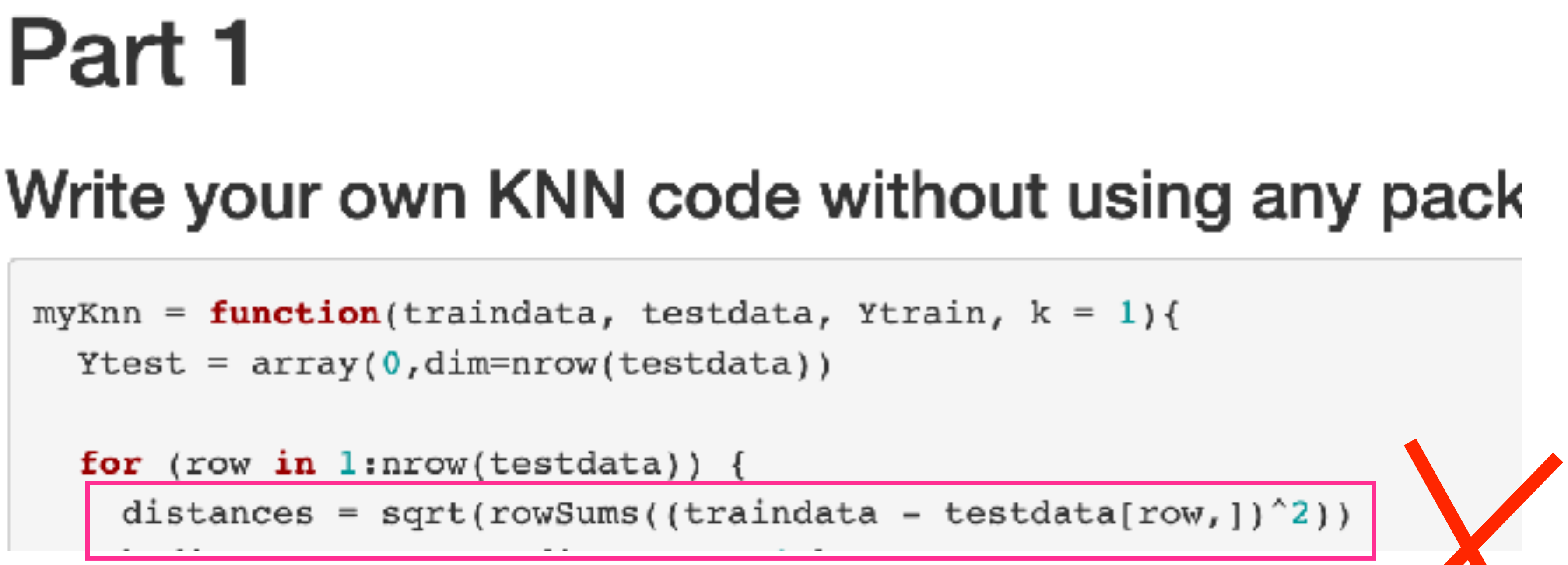
Student 1



Part 1

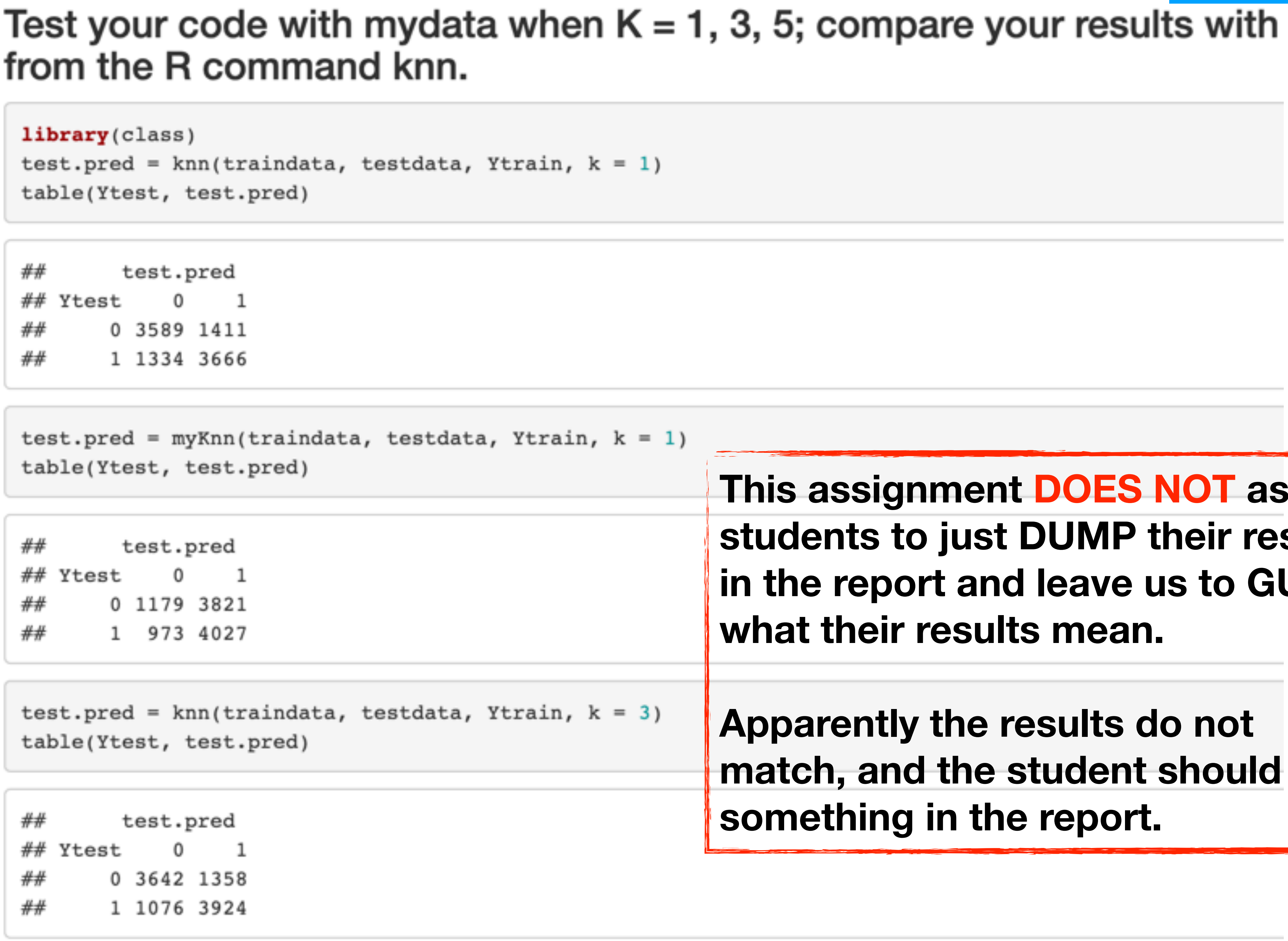
Write your own KNN code without using any pack

```
myKnn = function(traindata, testdata, Ytrain, k = 1){
  Ytest = array(0,dim=nrow(testdata))

  for (row in 1:nrow(testdata)) {
    distances = sqrt(rowSums((traindata - testdata[row,])^2))
```

As shown on the next page, the student’s KNN result is not correct. This is because the calculation for “`distances`” is not correct.

```
traindata = matrix(1:8, 4, 2)
testdata = matrix(1:4, 2, 2)
traindata
testdata[1, ]
traindata - testdata[1,]

# instead one should use
t(t(traindata) -testdata[1,])
```

Student 1



Test your code with mydata when K = 1, 3, 5; compare your results with from the R command knn.

```
library(class)
test.pred = knn(traindata, testdata, Ytrain, k = 1)
table(Ytest, test.pred)
```

```
##       test.pred
## Ytest     0    1
##       0 3589 1411
##       1 1334 3666
```

```
test.pred = myKnn(traindata, testdata, Ytrain, k = 1)
table(Ytest, test.pred)
```

```
##       test.pred
## Ytest     0    1
##       0 1179 3821
##       1  973 4027
```

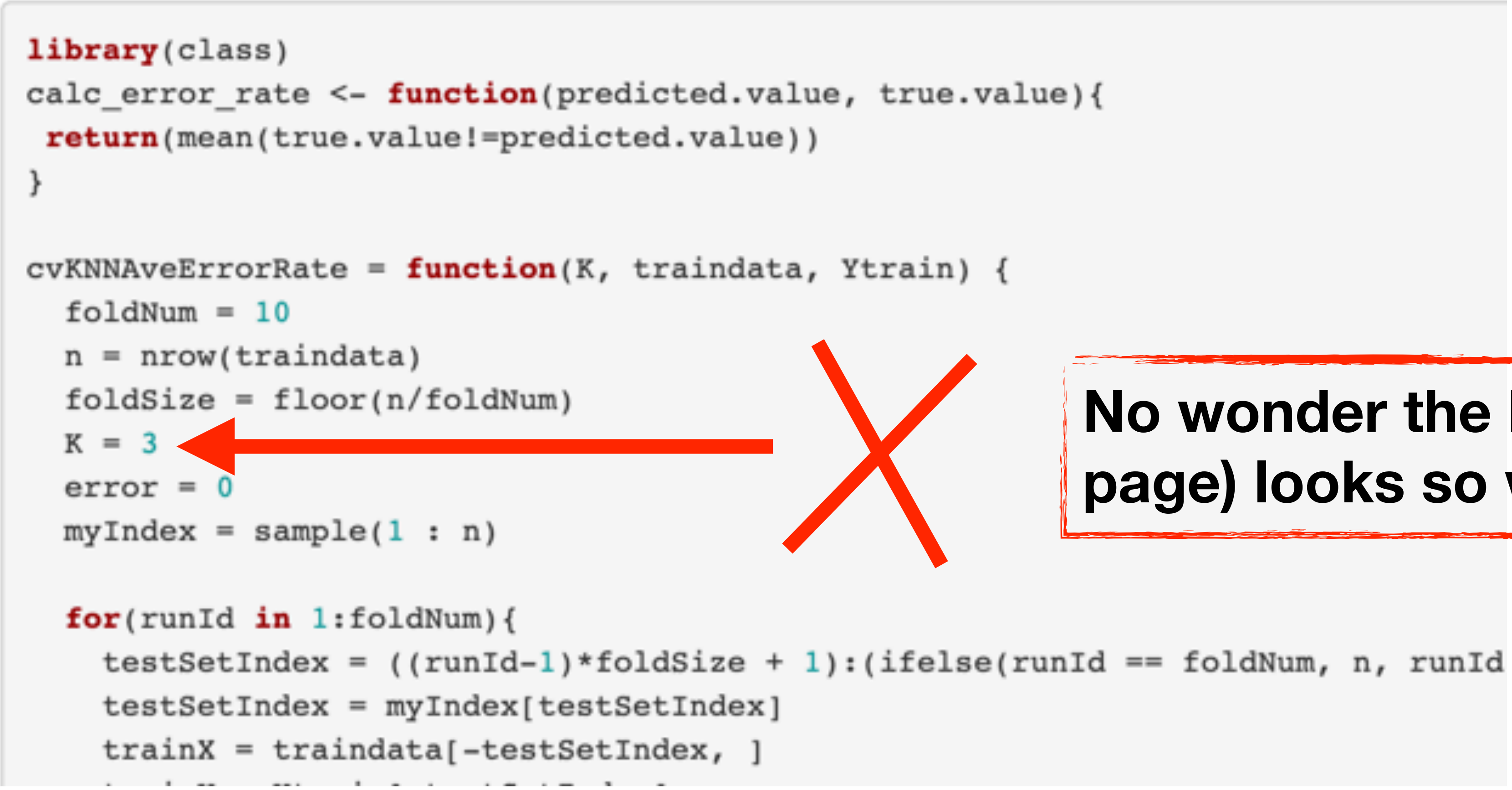
```
test.pred = knn(traindata, testdata, Ytrain, k = 3)
table(Ytest, test.pred)
```

```
##       test.pred
## Ytest     0    1
##       0 3642 1358
##       1 1076 3924
```

This assignment DOES NOT ask students to just DUMP their results in the report and leave us to GUESS what their results mean.

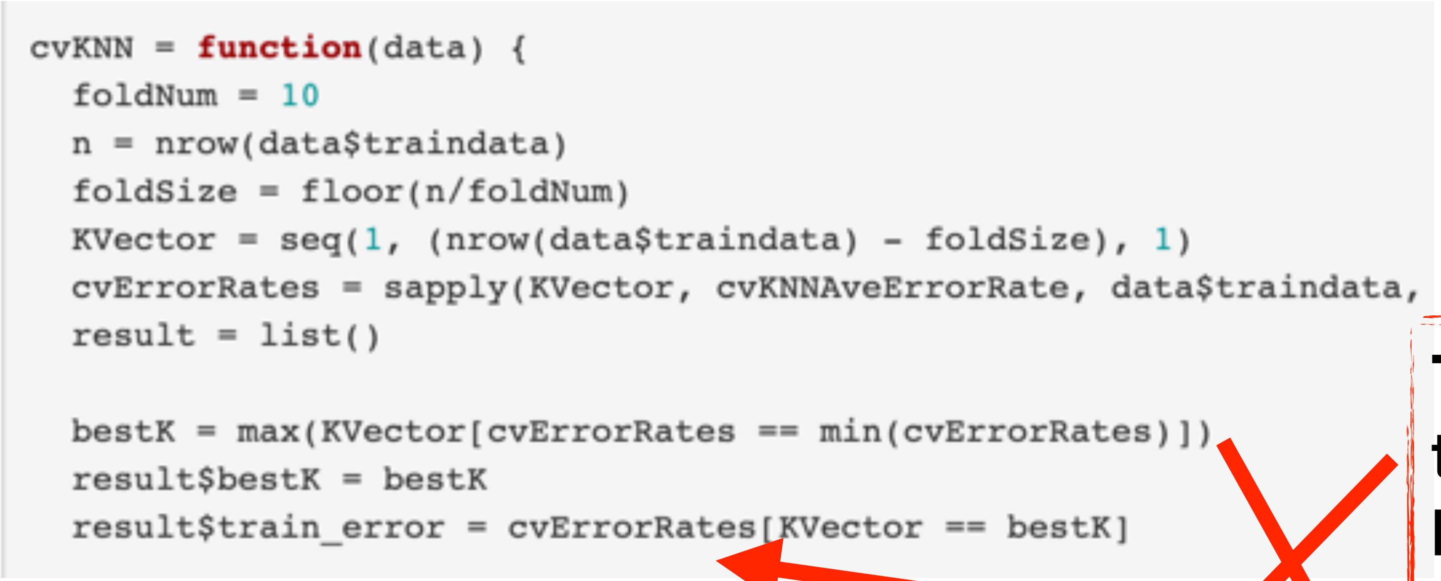
Apparently the results do not match, and the student should say something in the report.

Student 1



```
library(class)
calc_error_rate <- function(predicted.value, true.value){
 return(mean(true.value!=predicted.value))
}

cvKNNAveErrorRate = function(K, traindata, Ytrain) {
  foldNum = 10
  n = nrow(traindata)
  foldSize = floor(n/foldNum)
  K = 3
  error = 0
  myIndex = sample(1 : n)

  for(runId in 1:foldNum){
    testSetIndex = ((runId-1)*foldSize + 1):(ifelse(runId == foldNum, n, runId
    testSetIndex = myIndex[testSetIndex]
    trainX = traindata[-testSetIndex, ]
```

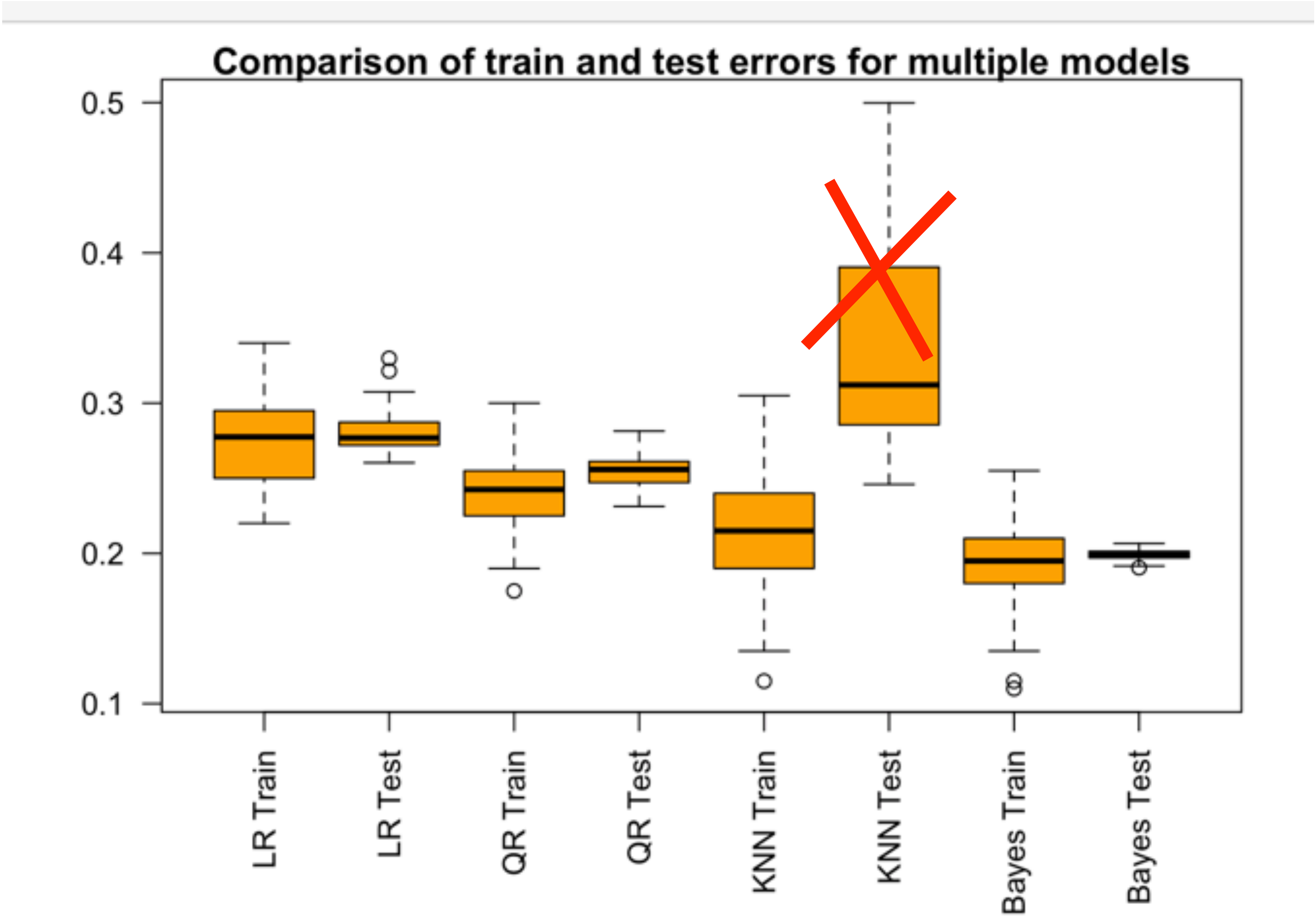
No wonder the box plot (on the next page) looks so weird.



```
cvKNN = function(data) {
  foldNum = 10
  n = nrow(data$traindata)
  foldSize = floor(n/foldNum)
  KVector = seq(1, (nrow(data$traindata) - foldSize), 1)
  cvErrorRates = sapply(KVector, cvKNNAveErrorRate, data$traindata,
  result = list()

  bestK = max(KVector[cvErrorRates == min(cvErrorRates)])
  result$bestK = bestK
  result$train_error = cvErrorRates[KVector == bestK]
```

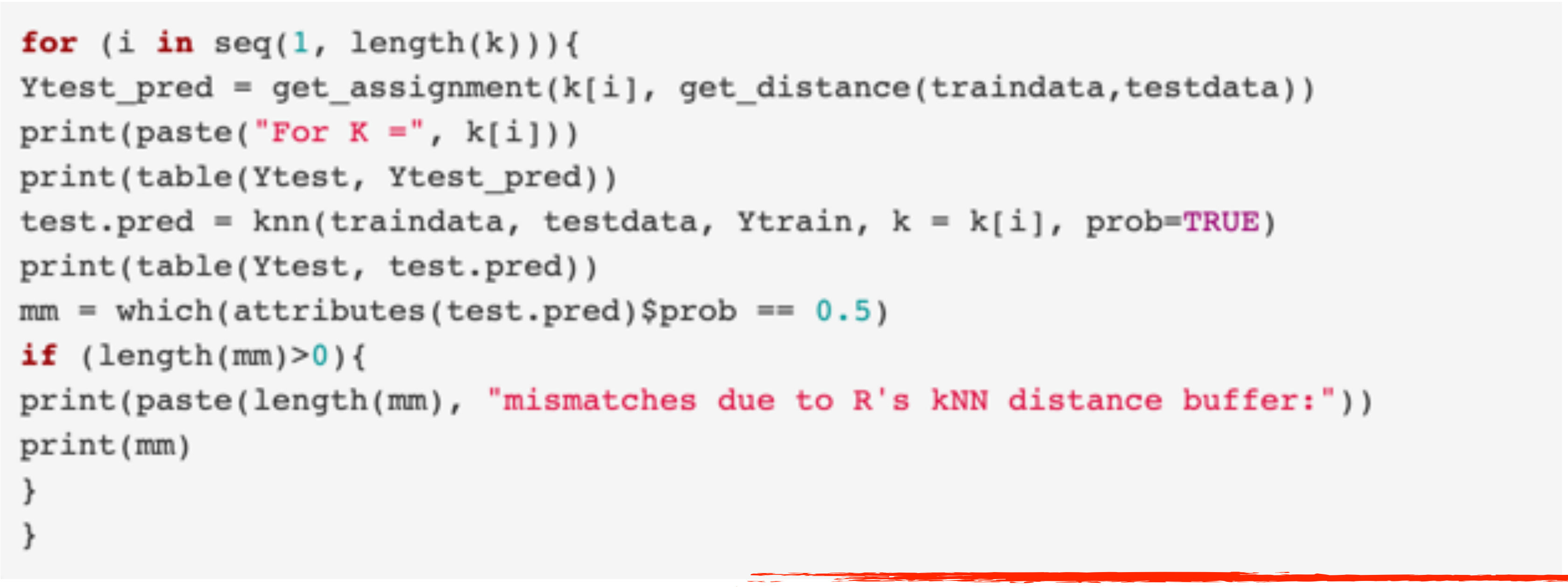
This is the CV error NOT the training error. Run knn with bestK on the training data, and compute the corresponding error.

Student 1

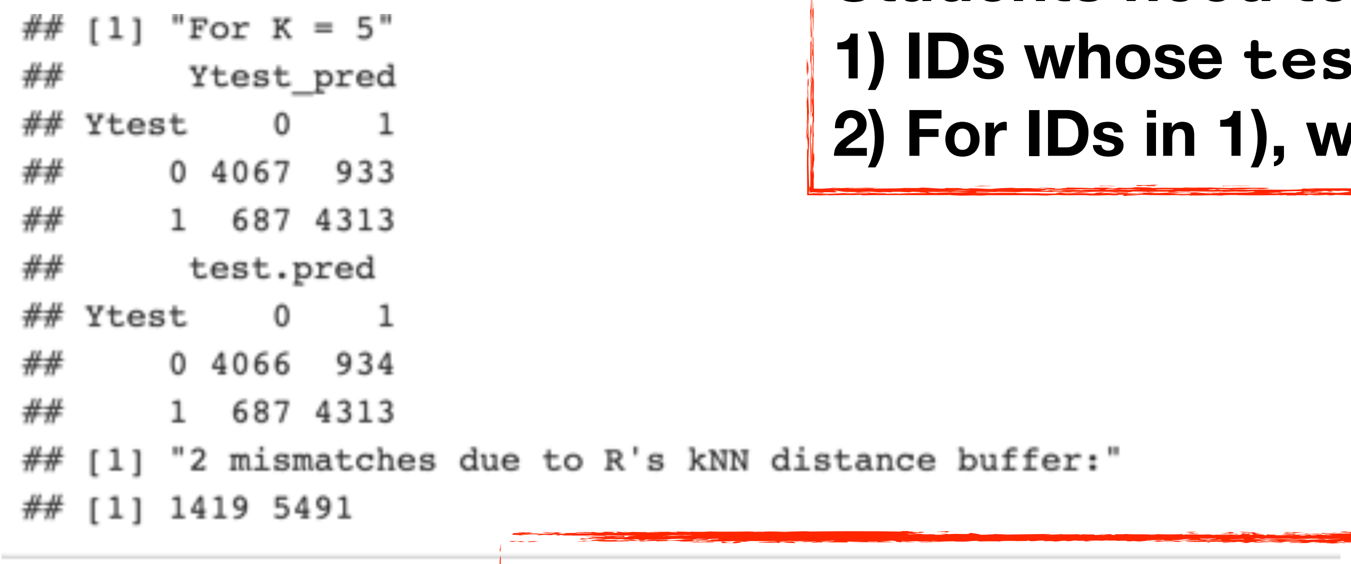


Rotate X-tick label 45 or 60 degree

Student 2



```
for (i in seq(1, length(k))){
Ytest_pred = get_assignment(k[i], get_distance(traindata,testdata))
print(paste("For K =", k[i]))
print(table(Ytest, Ytest_pred))
test.pred = knn(traindata, testdata, Ytrain, k = k[i], prob=TRUE)
print(table(Ytest, test.pred))
mm = which(attributes(test.pred)$prob == 0.5)
if (length(mm)>0){
print(paste(length(mm), "mismatches due to R's kNN distance buffer:"))
print(mm)
}
}
```

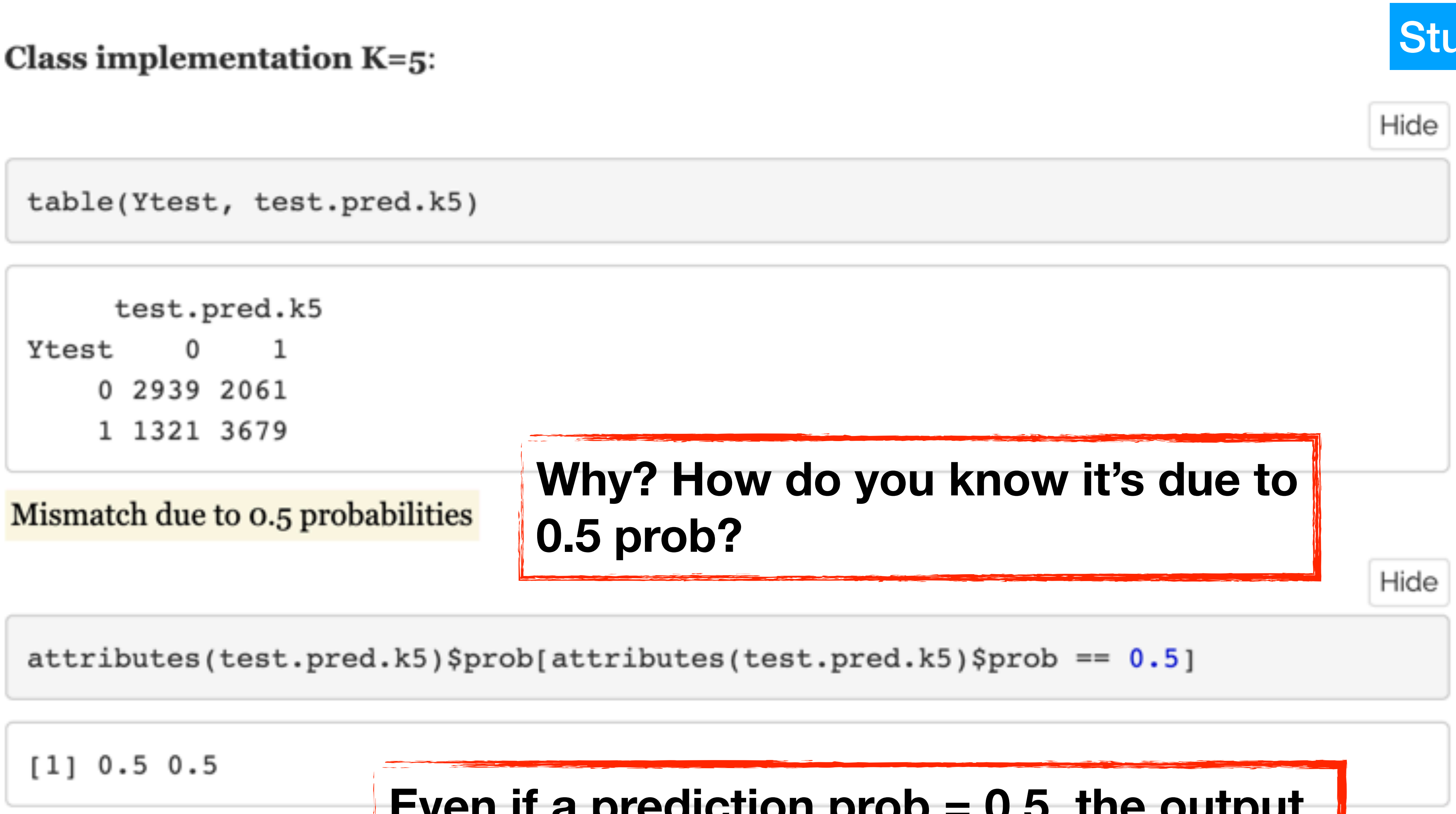


```
## [1] "For K = 5"
##        Ytest_pred
## Ytest     0    1
##       0 4067  933
##       1  687 4313
##        test.pred
## Ytest     0    1
##       0 4066  934
##       1  687 4313
## [1] "2 mismatches due to R's kNN distance buffer:"
## [1] 1419 5491
```

Students need to provide
1) IDs whose `test.pred != Ytest_pred`
2) For IDs in 1), whose prob from knn are 0.5

Are there two mismatches in the table shown above? Or you mean 2 potential mismatches…

Student 3



Class implementation K=5:

```
table(Ytest, test.pred.k5)
```

```
      test.pred.k5
Ytest     0    1
     0 2939 2061
     1 1321 3679
```

Mismatch due to 0.5 probabilities

```
attributes(test.pred.k5)$prob[attributes(test.pred.k5)$prob == 0.5]
```

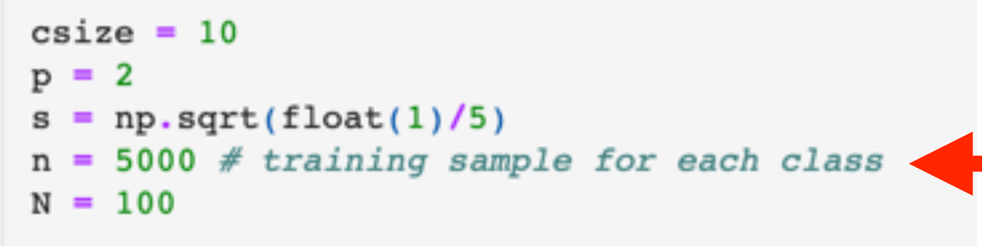
```
[1] 0.5 0.5
```

Why? How do you know it’s due to 0.5 prob?

Even if a prediction prob = 0.5, the output from knn could still match with yours.

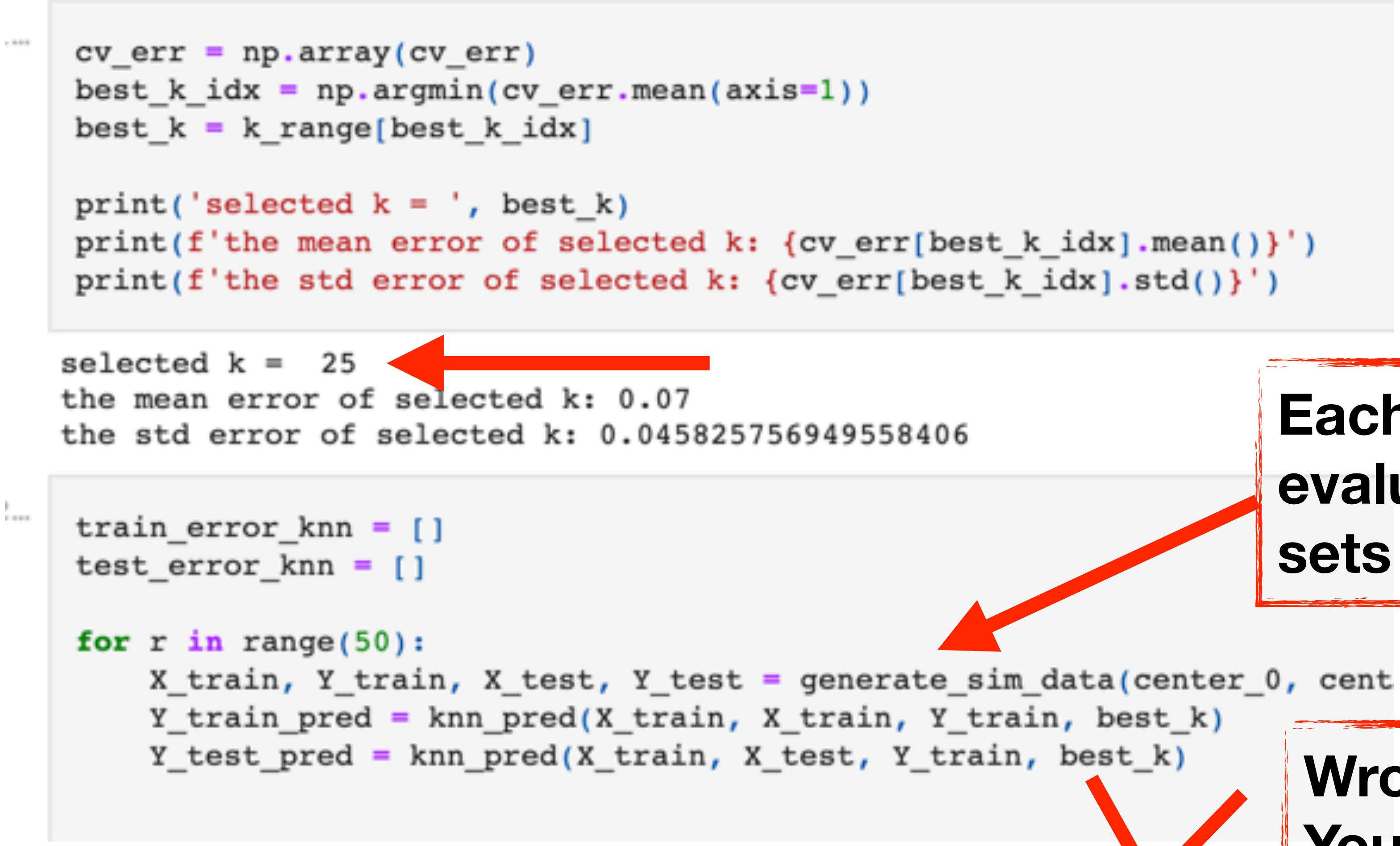
You have to show that the prediction prob (from knn) of that mismatched test point is equal to 0.5

Student 4



```
csize = 10
p = 2
s = np.sqrt(float(1)/5)
n = 5000 # training sample for each class
N = 100
```

Wrong values for n and N



```
cv_err = np.array(cv_err)
best_k_idx = np.argmin(cv_err.mean(axis=1))
best_k = k_range[best_k_idx]

print('selected k = ', best_k)
print(f'the mean error of selected k: {cv_err[best_k_idx].mean()}')
print(f'the std error of selected k: {cv_err[best_k_idx].std()}')
```

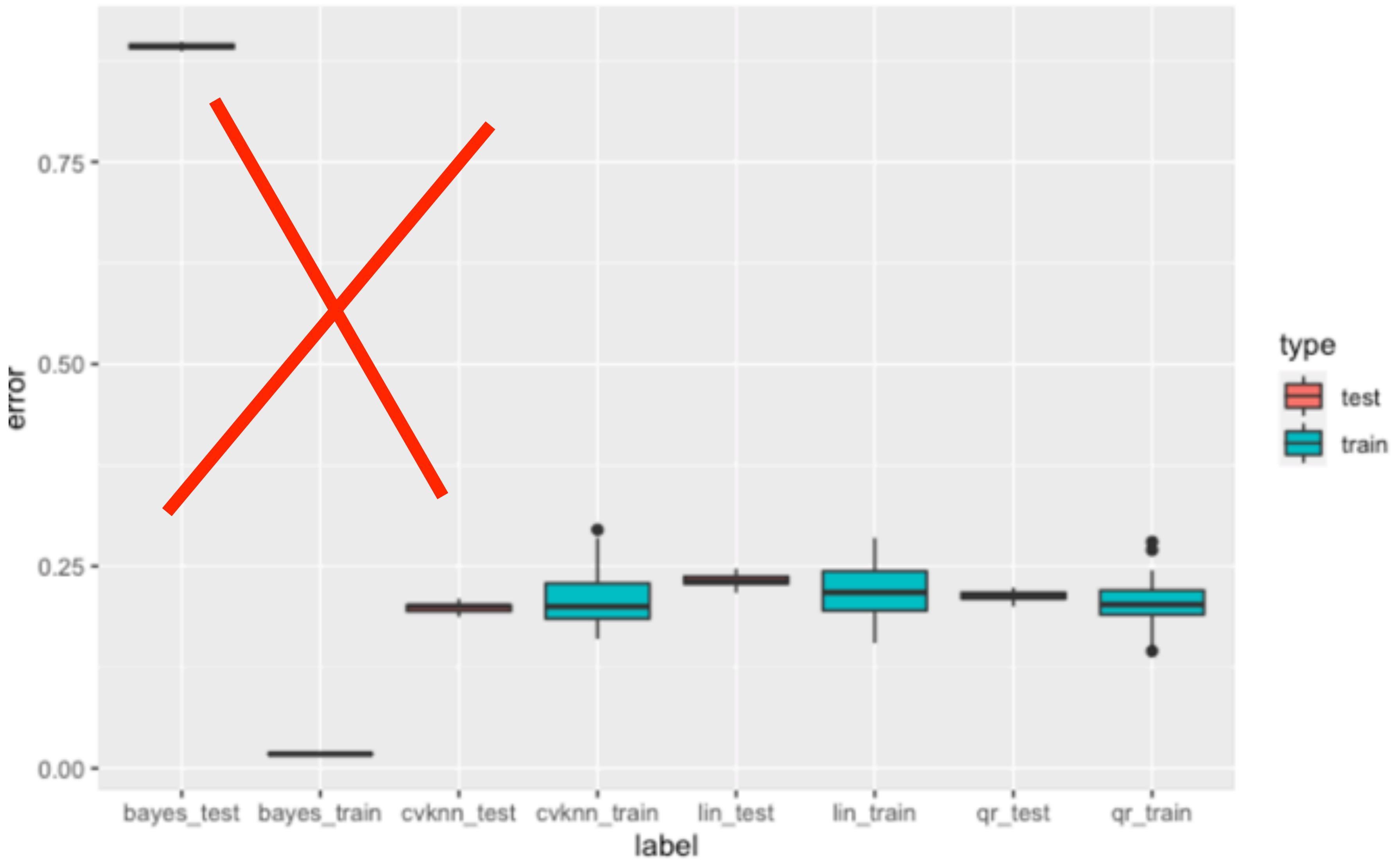
```
selected k =  25
the mean error of selected k: 0.07
the std error of selected k: 0.045825756949558406
```

```
train_error_knn = []
test_error_knn = []

for r in range(50):
    X_train, Y_train, X_test, Y_test = generate_sim_data(center_0, cent
    Y_train_pred = knn_pred(X_train, X_train, Y_train, best_k)
    Y_test_pred = knn_pred(X_train, X_test, Y_train, best_k)
```

Each procedure should be evaluated on the same 50 sets of training/test data

Wrong CVkNN procedure: You need to select best_k for each r

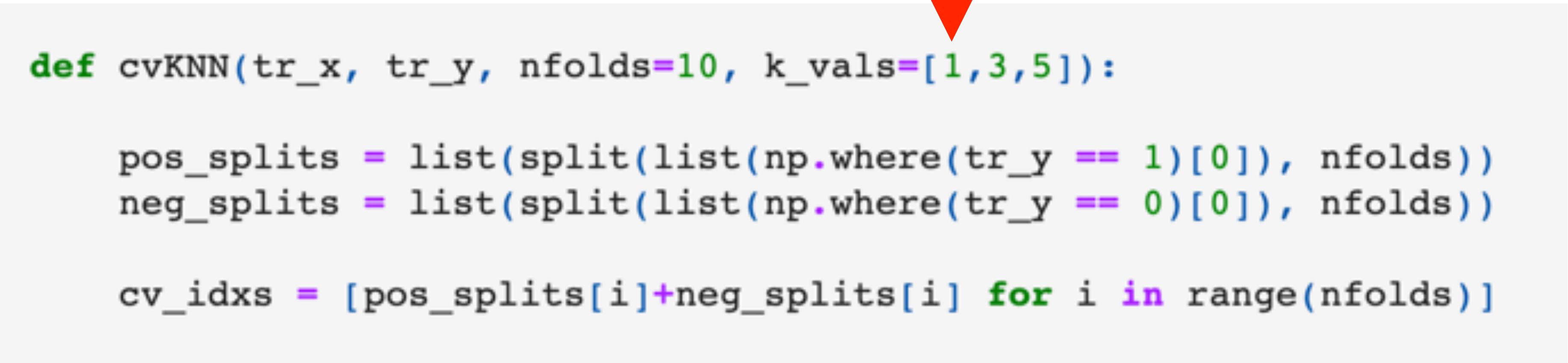
Student 5



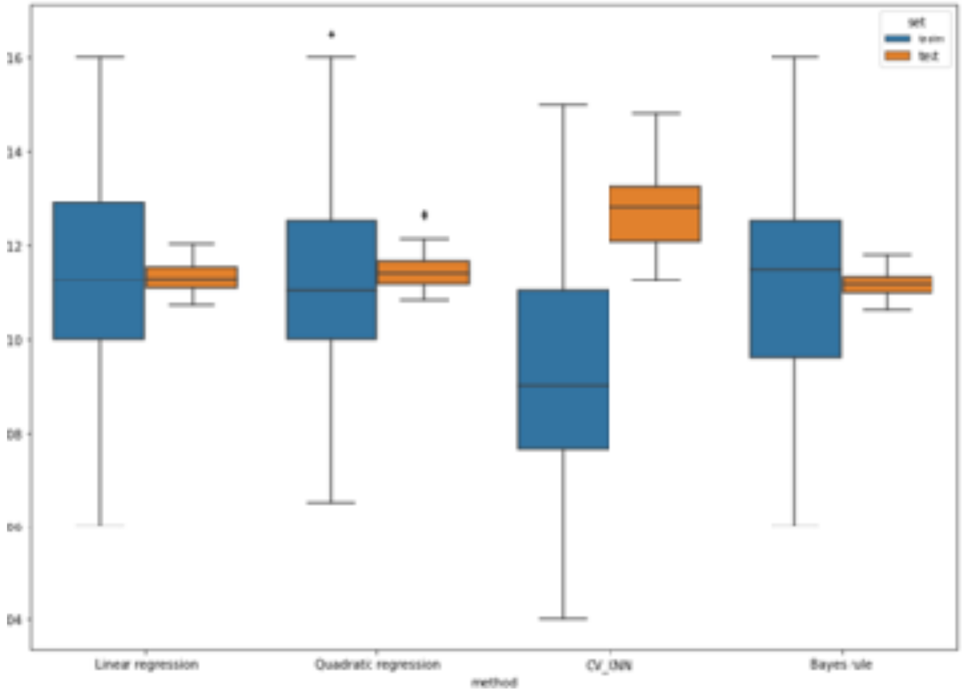
Apparently something is wrong.

Student 6

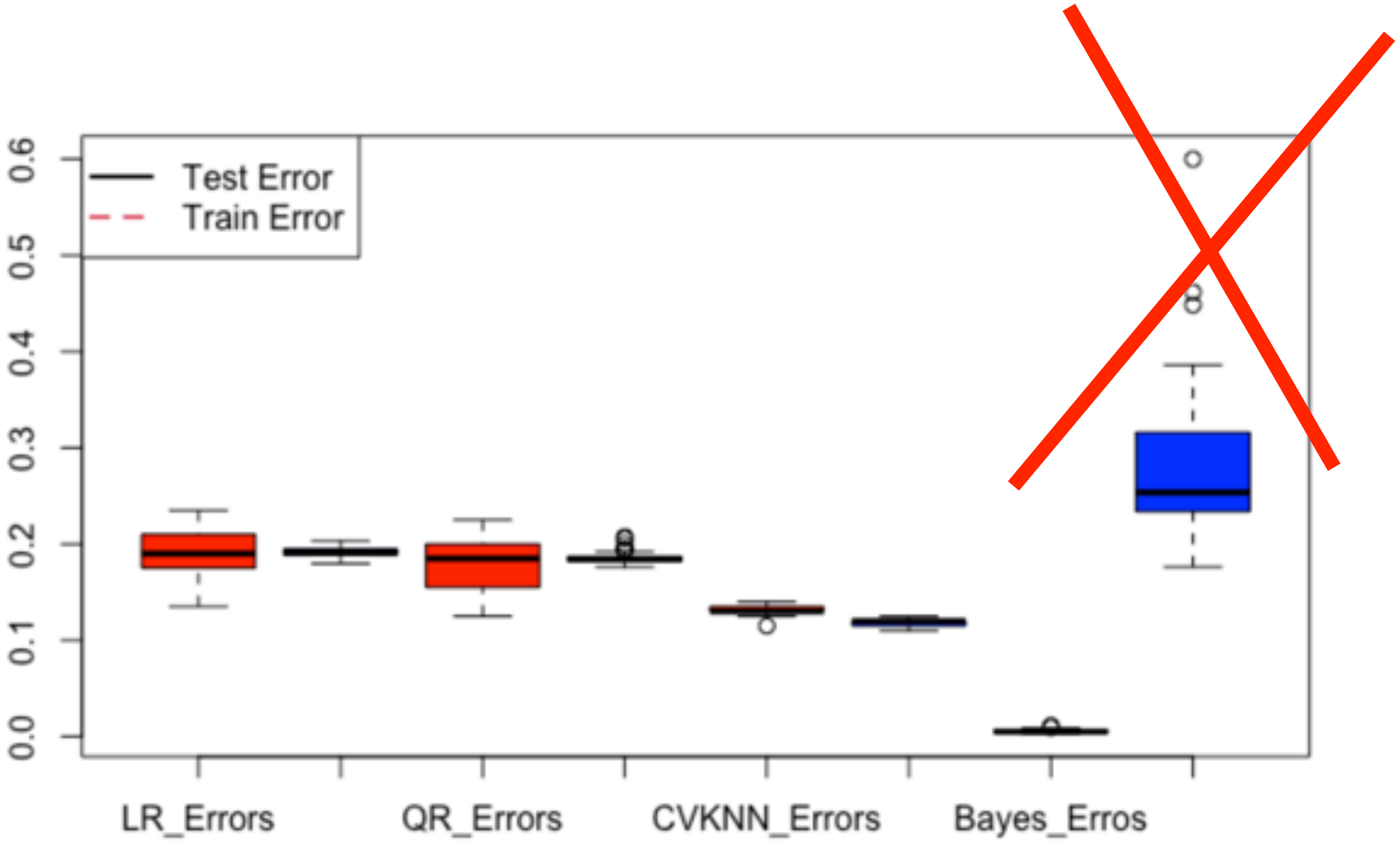
The three k-values, 1, 3, 5, used in Part I has nothing do with cvKNN in part II. No wonder your box plot does not look right.



```
def cvKNN(tr_x, tr_y, nfolds=10, k_vals=[1,3,5]):

    pos_splits = list(split(list(np.where(tr_y == 1)[0]), nfolds))
    neg_splits = list(split(list(np.where(tr_y == 0)[0]), nfolds))

    cv_idxs = [pos_splits[i]+neg_splits[i] for i in range(nfolds)]
```

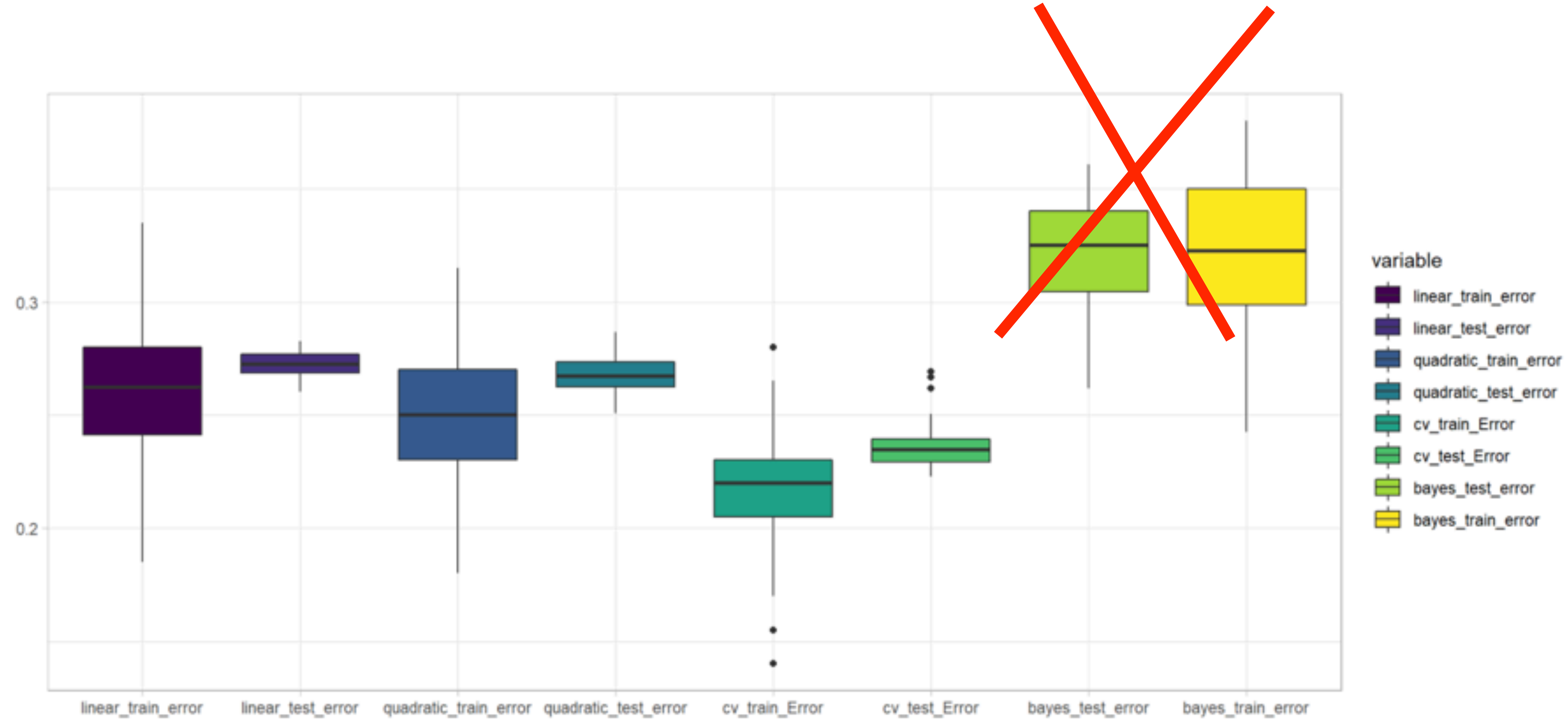


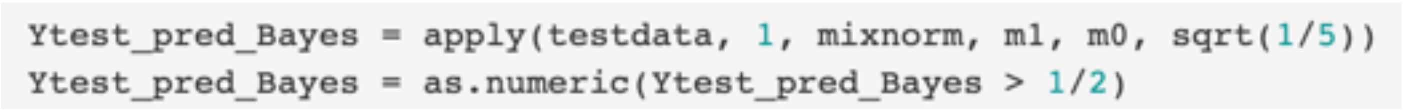
Student 7



Apparently something is wrong.

Student 8



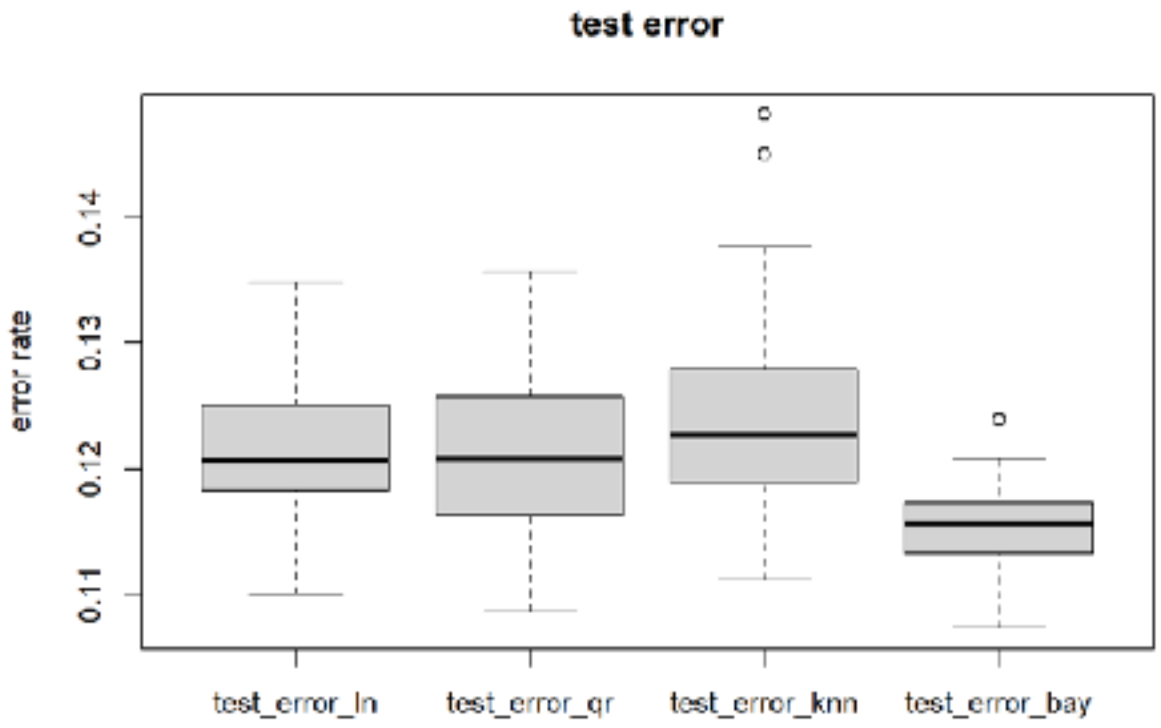


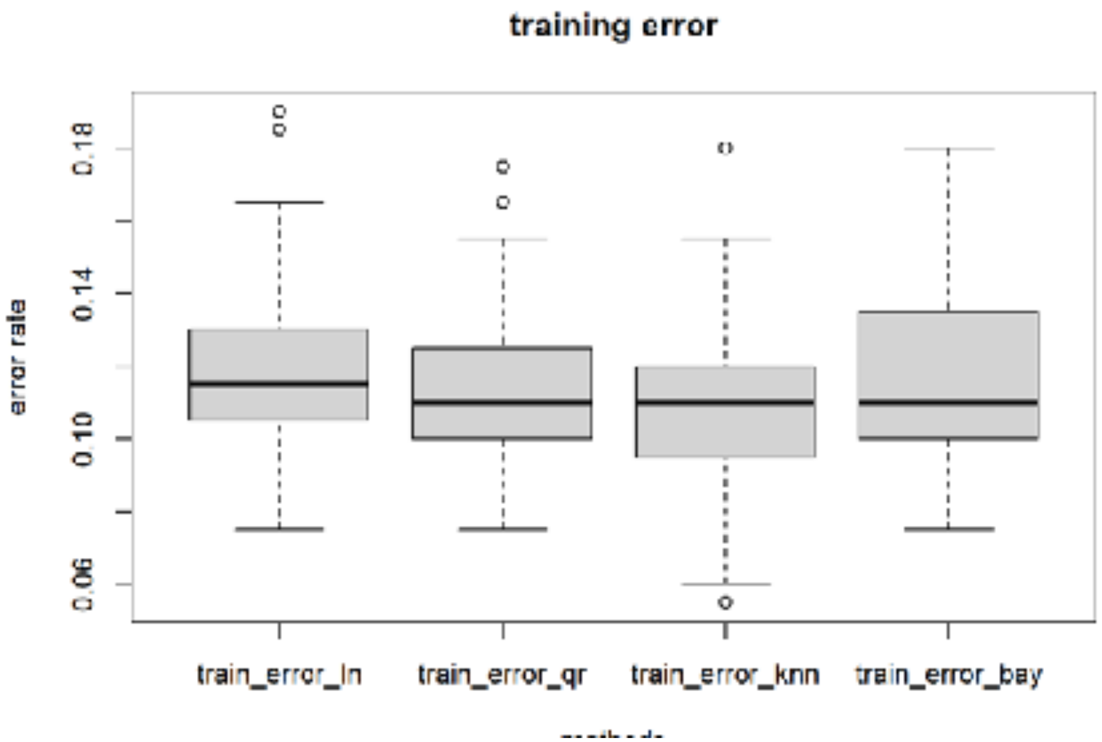
```
Ytest_pred_Bayes = apply(testdata, 1, mixnorm, m1, m0, sqrt(1/5))
Ytest_pred_Bayes = as.numeric(Ytest_pred_Bayes > 1/2)
```

Apparently something is wrong.
The cut-off for mixnorm is NOT 1/2.

Student 9

Suggest to display the 8 sets of errors in ONE figure.


test error
error rate
test_error_ln
test_error_qr
test_error_knn
test_error_bay


training error
error rate
train_error_ln
train_error_qr
train_error_knn
train_error_bay