Part 1

```
spam_test <- read.csv("~/ProgramZ/stat365/HW2/spam_test.csv")
spam_train <- read.csv("~/ProgramZ/stat365/HW2/spam_train.csv")
require(FNN)
```

Get the column with the na values in both test and train set. Also get response variable column.

```
nacol_name <- "capital_run_length_average"
nacol_test <- which(colnames(spam_test)==nacol_name)
col_resp <- which(colnames(spam_train)=="spam")
nacol_train <- which(colnames(spam_train)==nacol_name)
```

Scale both test and train data, and remove the response variable.

```
scale_test <- spam_test
scale_train <- spam_train
scale_test[,-nacol_test] <- as.data.frame(scale(scale_test[,-nacol_test]))
scale_train[,-c(nacol_train,col_resp)] <- as.data.frame(scale(scale_train[,-c(nacol_train,col_
scale_train$spam <-NULL
```

Get row in test for which there are na values, run knn and replace the na values in original dataset

```
na_r <- which(is.na(spam_test$capital_run_length_average))
test <- FNN::knn.reg(scale_test[-na_r,-nacol_test],test=scale_test[na_r,-nacol_test],y=scale_
spam_test[na_r,nacol_test] <- test$pred
```

Do the same for train

```
na_r2 <- which(is.na(spam_train$capital_run_length_average))
train <- FNN::knn.reg(scale_train[-na_r2,-nacol_train],test=scale_train[na_r2,-nacol_train],y
spam_train[na_r2,nacol_train] <- train$pred
```

Part 2

Comments are within the function.

```
knnclass <- function(xtrain, xtest, ytrain) {

  #standardize training and test using training set only
  trainMeans <- apply(xtrain, 2, function(y) mean(y))
  trainSD <- apply(xtrain, 2, function(y) sd(y))
  xtest <- (xtest-as.list(trainMeans))/as.list(trainSD)
  xtrain <- (xtrain-as.list(trainMeans))/as.list(trainSD)

  #split training into training and validation
  #set.seed(123)
```

```
ntrain <- nrow(xtrain)
s <- sample(1:ntrain, (4*ntrain)/5, replace = FALSE)
trainSet <- xtrain[s,]
trainY <- ytrain[s]
valSet <- xtrain[-s,]
valY <- ytrain[-s]
testSet <- xtest

#function that calculates euc dist to each row in test. Used in the function below
euc.dist <- function(testrow,traindata) {
  sum <- apply(((as.list(testrow)-traindata)^2),1,sum)
  return(sqrt(sum))
}

#function that gets the distance matrix for ONE row in test to all rows in train
getDistMatrix <- function(xtestrow,xtrain) {
  eucDist<-as.vector(euc.dist(xtestrow,xtrain))
  numberedRows <- c(1:nrow(xtrain))
  eucDistDF <- data.frame(numberedRows,eucDist)
  sortDistDF <- eucDistDF[order(eucDistDF$eucDist),]
  return(sortDistDF)
}

#Sorted distance list containing distance matrix for each test row. Only need to do this or
sortDistDFV <- apply(valSet,1,getDistMatrix,xtrain=trainSet)

#function that gets the classification for each testrow
getClassfn <- function(kDistDF, y) {
  kYClass <- y[kDistDF[,1]]
  class <- as(names(which.max(table(kYClass))), Class=mode(y))
  return(class)
}

#classification
kclass <- function(sortDistDFV,y,k) {
  get.predic <- function(sortDistDF,y,k) {
    sortDistDF <- (sortDistDF)
    kDistDF <- sortDistDF[1:k,]
    class <- getClassfn(kDistDF,y)
    return (class)
  }
  #lapply because sortDistDFV is a list. (apply returns list).
  predicted <- lapply(sortDistDFV,get.predic,y=y,k=k)
  predicted <- (as.vector(unlist(predicted)))
  return (predicted)
}

#function to calculate misclassfication error rate
mer <- function(pred,actual) {
  matchv <- mapply(function(x,y) {ifelse ((x==y),0,1)}, pred, actual,SIMPLIFY=TRUE)
  return(sum(matchv)/length(pred))
```

```
  }

  #We get the misclassfication error rate for each k 2:15.
  nK <- 15
  kCount <- c(2:nK)
  kDF <- data.frame(kCount,rep(NA,nK-1))
  colnames(kDF) <- c("K","MER")
  for (i in 2:nK) {
    pred <- kclass(sortDistDFV,y=trainY,k=i)
    e <- mer(pred,valY)
    kDF$MER[i-1] <- e
  }
  kDF <- kDF[order(kDF$MER),]

  #optimal k is the one with lowest MER, first item in the ordered vector.
  optK <- kDF$K[1]
  #Now get the distance matrix list for the actual test set.
  sortDistDFV_final <- apply(testSet,1,getDistMatrix,xtrain=xtrain)
  finalPred <- kclass(sortDistDFV_final,ytrain,optK)
}
```

Part 3

(cont'd from part 1): We have set up variables such that the UNSCALED train and test data WITH filled in N/A values are called spam_train and spam_test respectively.

```
spam <- spam_train$spam
```

First part (KNN without capital run ave length)

```
spam_train_first <- spam_train
spam_test_first <- spam_test
spam_train_first$spam <- NULL
#get rid of captital_run_length_average predictor
spam_train_first$capital_run_length_average <- NULL
spam_test_first$capital_run_length_average <- NULL
knn_pred1 <- knnclass(spam_train_first,spam_test_first,spam)
```

Second part (KNN with capital run ave length)

```
spam_train_second <- spam_train
spam_test_second <- spam_test
spam_train_second$spam <- NULL
knn_pred2 <- knnclass(spam_train_second,spam_test_second,spam)
#if k = 15, this is the "model" answer, w/o scaling
#knn_pred22 <- as.vector(FNN::knn(spam_train_second,spam_test_second,as.factor(spam), k=15))
```

Third part (Logistic regression without capital run length ave)

```
spam_train_third <- spam_train
spam_test_third <- spam_test
spam_train_third$capital_run_length_average <- NULL
spam_test_third$capital_run_length_average <- NULL
m3 <- glm(spam~.,family=binomial,data=spam_train_third)
```

```
## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred
```

```
logm_pred1 <- predict(m3, newdata=spam_test_third, type="response")
```

Fourth part (Logistic regression with capital run length ave)

```
spam_train_fourth <- spam_train
spam_test_fourth <- spam_test
m4 <- glm(spam~.,family=binomial,data=spam_train_fourth)
```

```
## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred
```

```
logm_pred2 <- predict(m4, newdata=spam_test_fourth, type="response")
```

Let's take a look at the summary of the 4th model. In 3-4 sentences, provide a quick summary of your second logistic regression model. Which predictors appeared to be most significant? Are there any surprises in the predictors that ended up being significant or not significant?

```
summary(m4)
```

```
##
## Call:
## glm(formula = spam ~ ., family = binomial, data = spam_train_fourth)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -4.0919  -0.2125   0.0000   0.1275   4.5985
##
## Coefficients:
##                              Estimate Std. Error z value Pr(>|z|)
## (Intercept)                 -1.753532   0.179306  -9.780  < 2e-16 ***
## word_freq_make              -0.276007   0.247224  -1.116 0.264240
## word_freq_address           -0.152309   0.085467  -1.782 0.074736 .
## word_freq_all                0.198169   0.134344   1.475 0.140190
## word_freq_3d                 2.226130   1.903491   1.169 0.242203
## word_freq_our                0.569887   0.125431   4.543 5.53e-06 ***
## word_freq_over               0.940130   0.321311   2.926 0.003434 **
## word_freq_remove             1.820131   0.336735   5.405 6.47e-08 ***
## word_freq_internet           0.603346   0.237009   2.546 0.010907 *
## word_freq_order              0.437776   0.310839   1.408 0.159022
## word_freq_mail               0.279517   0.109062   2.563 0.010380 *
## word_freq_receive           -0.280360   0.334874  -0.837 0.402474
## word_freq_will              -0.085356   0.085586  -0.997 0.318613
```

```
## word_freq_people           -0.134132   0.276877  -0.484 0.628068
## word_freq_report            0.154409   0.152952   1.010 0.312724
## word_freq_addresses         0.846307   0.714404   1.185 0.236162
## word_freq_free              0.855252   0.162938   5.249 1.53e-07 ***
## word_freq_business          0.922087   0.265315   3.475 0.000510 ***
## word_freq_email             0.219449   0.151455   1.449 0.147354
## word_freq_you               0.068757   0.041988   1.638 0.101522
## word_freq_credit            1.043327   0.666330   1.566 0.117400
## word_freq_your              0.299127   0.064781   4.618 3.88e-06 ***
## word_freq_font              0.223455   0.187618   1.191 0.233650
## word_freq_000               2.339967   0.557147   4.200 2.67e-05 ***
## word_freq_money             0.796148   0.332541   2.394 0.016660 *
## word_freq_hp               -1.630000   0.313691  -5.196 2.03e-07 ***
## word_freq_hpl              -0.965850   0.461613  -2.092 0.036408 *
## word_freq_george           -8.519986   2.034502  -4.188 2.82e-05 ***
## word_freq_650               0.310727   0.236416   1.314 0.188738
## word_freq_lab              -2.047861   1.307796  -1.566 0.117375
## word_freq_labs             -0.283955   0.323785  -0.877 0.380495
## word_freq_telnet           -0.153308   1.455711  -0.105 0.916126
## word_freq_857               2.900174   2.636380   1.100 0.271306
## word_freq_data             -0.584954   0.350046  -1.671 0.094706 .
## word_freq_415               0.423593   1.458512   0.290 0.771488
## word_freq_85               -1.785537   0.848391  -2.105 0.035325 *
## word_freq_technology        0.901174   0.378598   2.380 0.017299 *
## word_freq_1999              0.057831   0.208143   0.278 0.781132
## word_freq_parts            -0.466180   0.403909  -1.154 0.248430
## word_freq_pm               -1.198314   0.523085  -2.291 0.021972 *
## word_freq_direct            0.420933   0.848269   0.496 0.619735
## word_freq_cs              -37.847675  31.672514  -1.195 0.232099
## word_freq_meeting          -2.586338   0.920259  -2.810 0.004947 **
## word_freq_original         -0.760130   0.769854  -0.987 0.323462
## word_freq_project          -1.332537   0.605949  -2.199 0.027871 *
## word_freq_re               -0.837499   0.182046  -4.600 4.22e-06 ***
## word_freq_edu              -1.553544   0.346409  -4.485 7.30e-06 ***
## word_freq_table            -1.268388   2.177947  -0.582 0.560312
## word_freq_conference       -4.022867   1.947205  -2.066 0.038831 *
## char_freq_.                -1.324234   0.571189  -2.318 0.020429 *
## char_freq_..1               0.116952   0.302305   0.387 0.698854
## char_freq_..2              -0.991137   1.384087  -0.716 0.473933
## char_freq_..3               0.244670   0.065219   3.752 0.000176 ***
## char_freq_..4               4.461521   0.748712   5.959 2.54e-09 ***
## char_freq_..5               2.336168   1.153565   2.025 0.042850 *
## capital_run_length_average  0.052178   0.040812   1.278 0.201076
## capital_run_length_longest  0.010145   0.002683   3.781 0.000156 ***
## capital_run_length_total    0.000501   0.000238   2.105 0.035291 *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 4316.6  on 3219  degrees of freedom
```

```
## Residual deviance: 1295.2  on 3162  degrees of freedom
## AIC: 1411.2
##
## Number of Fisher Scoring iterations: 13
```

The residual deviance (1295.2) is lower than the null deviance (4316.6)– this is a good sign that some of the predicors are significant. Capital_run_length_longest, word_freq_edu, word_freq_george, word_freq_money word_freq_free are some of the significant predictors for this model. Somewhat surprising that word_freq_george would be so significant– are there a lot of people named george?

This is data frame that is the .csv file

```
HW2_tk553_results <- data.frame(spam_test$capital_run_length_average,knn_pred1,knn_pred2,logm
#write.csv with row.names=FALSE
```