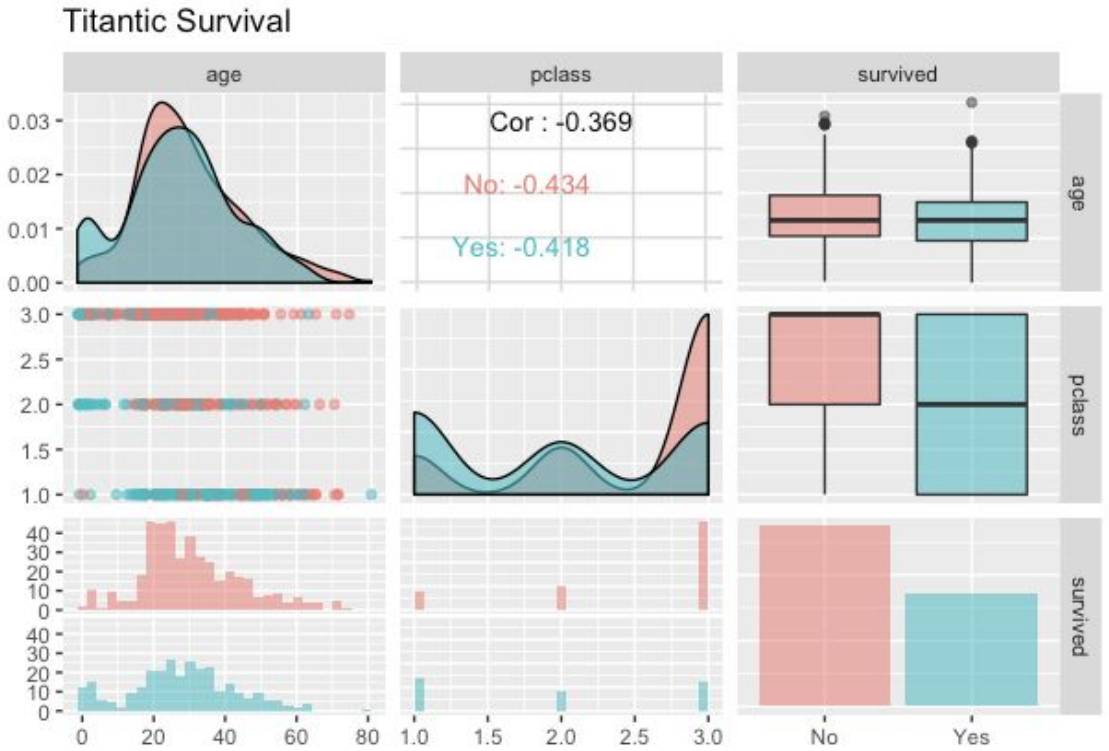
Unit6: For Live Session

DS6306
Garrity

PART 1 - Titanic k-NN

EDA

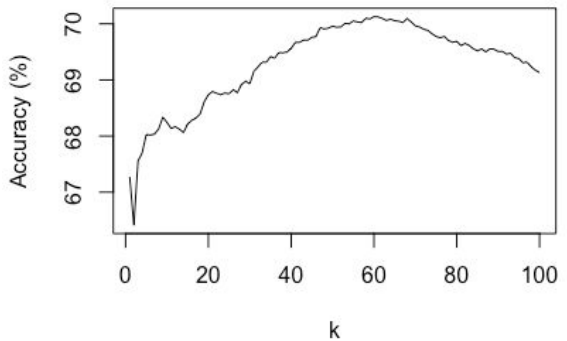
```
library(jsonlite)
library(RCurl)
library(XML)

### Import and tidy the data
data <-
getURL("https://public.opendatasoft.com/api/records/1.0/search/?dataset=titanic-passengers&rows=2000&facet=survived&facet=pclass&facet=sex&facet=age&facet=embarked")

temp <- fromJSON(data, flatten = FALSE)
class(temp) # it is a list.

df <- as.data.frame(temp$records$fields)

df$survived <- as.factor(df$survived)
       rm(temp, data)

library(GGally)

df %>% select(age, pclass, survived) %>%
ggpairs(aes(color=survived, alpha=0.2)) +
ggtitle("Titantic Survival")
```



PART 1 - Titanic k-NN

Tune hyperparameter “k”

```
#### First, omit any rows where one of the observations is NA
df_reduced <- df %>% select(age, pclass, survived) %>% na.omit()
df_reduced_scaled <- data.frame(Zage = scale(df_reduced$age), Zpclass = scale(df_reduced$pclass), Survived =
df_reduced$survived)

#### Tune hyperparameter, "k"
splitPerc = .75
iterations = 250
numks = 100
masterAcc = matrix(nrow = iterations, ncol = numks)

for(j in 1:iterations)
{
# accs = data.frame(accuracy = numeric(30), k = numeric(30)) # I don't think we need this!
trainIndices = sample(1:dim(df_reduced_scaled)[1],round(splitPerc * dim(df_reduced_scaled)[1]))
train = df_reduced_scaled[trainIndices,]
test = df_reduced_scaled[-trainIndices,]
for(i in 1:numks)
{
  classifications = knn(train[,c(1,2)],test[,c(1,2)],train$Survived, prob = TRUE, k = i)
  table(classifications,test$Survived)
  CM = confusionMatrix(table(classifications,test$Survived))
  masterAcc[j,i] = CM$overall[1]
}
}

MeanAcc = colMeans(masterAcc)

plot(seq(1,numks,1),MeanAcc*100, type = "l", xlab = "k", ylab = "Accuracy (%)")
```



which.max(MeanAcc)
[1] 60 ----------> k=60 produces the highest classification accuracy (70%).

PART 1 - Titanic k-NN

Would I have survived?

```
### predict self-survival based on age...iterate for each class (need to scale my age and class first!!!)

scaled_age <- scale(c(min(df_reduced$age), 40, max(df_reduced$age)))
scaled_class <- scale(c(1,2,3))

for (i in 1:length(scaled_class))
{
  age_class = data.frame(Age = scaled_age[2], Class = scaled_class[i])
  print(knn(df_reduced_scaled[,1:2], age_class, df_reduced_scaled$Survived, k = 60, prob = TRUE))
}
```

```
[1] Yes -------> If I were a first class passenger, then it is likely that I would have survived.
attr(,"prob")
[1] 0.7741935
Levels: No Yes
[1] No ------> If I were a second or third class passenger, then it is likely that I would not have survived.
attr(,"prob")
[1] 0.5645161
Levels: No Yes
[1] No ------> If I were a second or third class passenger, then it is likely that I would not have survived.
attr(,"prob")
[1] 0.6923077
Levels: No Yes
```

PART 1 - Titanic k-NN

Classify the “test” data set

```
temp <- read.csv('/Users/stevengarrity/SMU_MSDS/DS6306_DoingDataScience/DDS_Git/Unit 6/titanic_test.csv',header = TRUE)

df_test <- temp
dim(df_test)

# remove 'NA' observations
df_test_reduced <- df_test %>% select(Age, Pclass) %>% na.omit()
dim(df_test_reduced)

test_classifications <- knn(df_reduced_scaled[,1:2], df_test_reduced, df_reduced_scaled$Survived, k = 60, prob = TRUE)
```

```
> test_classifications
  [1] No No No No No No No No No No No No No No No No No No No No No No No No No No No No No No No No No No No
 [38] No No No No No No No No No No No No No No No No No No No No No No No No No No No No No No No No No No No
 [75] No No No No No No No No No No No No No No No No No No No No No No No No No No No No No No No No No No
[112] No No No No No No No No No No No No No No No No No No No No No No No No No No No No No No No No No No
[149] No No No No No No No No No No No No No No No No No No No No No No No No No No No No No No No No No No
[186] No No No No No No No No No No No No No No No No No No No No No No No No No No No No No No No No No No
[223] No No No No No No No No No No No No No No No No No No No No No No No No No No No No No No No No No No
[260] No No No No No No No No No No No No No No No No No No No No No No No No No No No No No No No No No No
[297] No No No No No No No No No No No No No No No No No No No No No No No No No No No No No No No No No No
```

According to our k-nn model, no one in the test dataset was predicted to have survived.

PART 1 - Titanic k-NN

Confusion Matrix, accuracy, misclassification rate, sensitivity and specificity

```
> classifications <- knn.cv(df_reduced_scaled[,1:2],df_reduced_scaled$Survived, k = 60, prob = TRUE)
> confusionMatrix(classifications,df_reduced$survived)
```

Confusion Matrix and Statistics

```
          Reference
Prediction  No Yes
       No  371 159
       Yes  53 131

               Accuracy : 0.7031
                 95% CI : (0.6681, 0.7364)
    No Information Rate : 0.5938
    P-Value [Acc > NIR] : 8.964e-10

                  Kappa : 0.3468

 Mcnemar's Test P-Value : 5.537e-13

            Sensitivity : 0.8750
            Specificity : 0.4517
         Pos Pred Value : 0.7000
         Neg Pred Value : 0.7120
             Prevalence : 0.5938
         Detection Rate : 0.5196
   Detection Prevalence : 0.7423
      Balanced Accuracy : 0.6634

       'Positive' Class : No
```

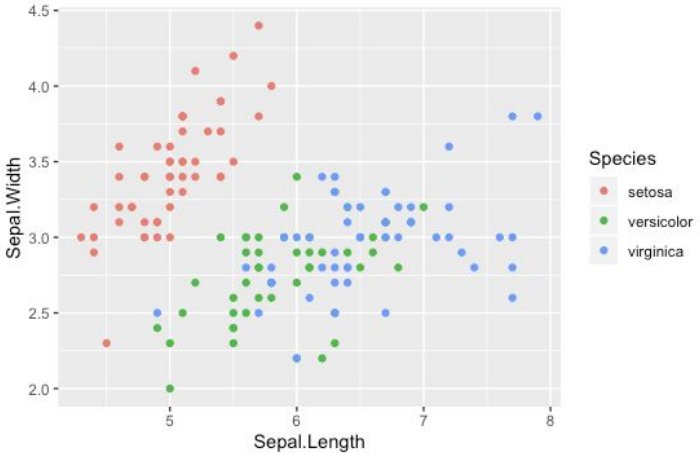
Accuracy:	0.70
Misclassification rate:	0.30
Sensitivity:	0.875
Specificity:	0.4517

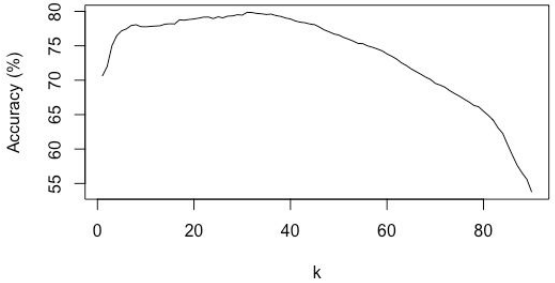
PART 2 - Iris k-NN

Multinomial classification using 70/30 train/test split

```
iris_df <- iris %>% select(Sepal.Length, Sepal.Width, Species)

iris_df %>% ggplot(aes(x=Sepal.Length, y=Sepal.Width, col=Species)) +
  geom_point()

#### Grid search to tune hyperparameter, "k"
splitPerc = .70
iterations = 250
numks = 90

masterAcc = matrix(nrow = iterations, ncol = numks)

for(j in 1:iterations)
{
trainIndices = sample(1:dim(iris_df)[1],round(splitPerc * dim(iris_df)[1]))
train = iris_df[trainIndices,]
test = iris_df[-trainIndices,]
for(i in 1:numks)
{
  classifications = knn(train[,c(1,2)],test[,c(1,2)],train$Species, prob = TRUE, k = i)
  table(classifications,test$Species)
  CM = confusionMatrix(table(classifications,test$Species))
  masterAcc[j,i] = CM$overall[1]
}
}

MeanAcc = colMeans(masterAcc)
plot(seq(1,numks,1),MeanAcc*100, type = "l", xlab = "k", ylab = "Accuracy (%)")

# find index of k that produces highest accuracy:
which.max(MeanAcc)
```

[1] 31 ------------> k = 31 produces the highest accuracy (79.8%) for a 70/30 train/test k-nn classifier

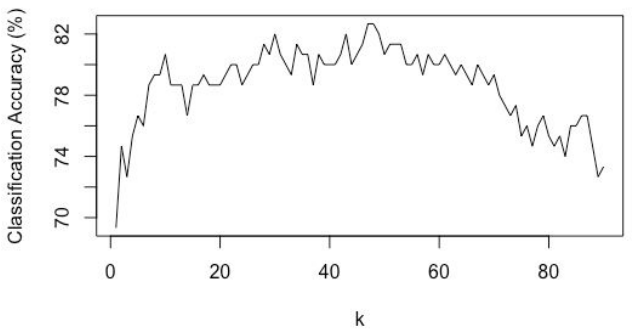




PART 2 - Iris k-NN

Multinomial classification using leave-one-out cross validation

```
numks = 90
accs = data.frame(accuracy = numeric(numks), k = numeric(numks))

for(i in 1:numks)
{
  classifications = knn.cv(iris_df[,c(1,2)],iris_df$Species, prob = TRUE, k = i)
  table(iris_df$Species,classifications)
  CM = confusionMatrix(table(iris_df$Species,classifications))
  accs$accuracy[i] = CM$overall[1]
  accs$k[i] = i
}
plot(accs$k,accs$accuracy, type = "l", xlab = "k")

# find index of k that produces highest accuracy:
> which.max(accs$accuracy)
[1] 47 ------> k = 47 produces the highest classification accuracy (82.7%)
```




```
classifications <- knn.cv(iris_df[,c(1,2)], iris_df$Species, prob = TRUE, k = which.max(accs$accuracy))

> confusionMatrix(classifications,iris_df$Species)

Confusion Matrix and Statistics

            Reference
Prediction    setosa versicolor virginica
  setosa          50          0         1
  versicolor       0         37        12
  virginica        0         13        37

Overall Statistics

               Accuracy : 0.8267
```

We performed a sweep of the “k” parameter for 70/30 and leave-one-out cross validation classification models. The leave-one-out model suggested a larger “k” (k=47 vs. k=31) and produced a slightly higher classification accuracy (82.7% vs 79.8%) relative to the model fit using a 70/30 train/test split.

Takeaways & Questions

Any rules of thumb on when to use leave-one-out verses a different train/test ratio (80/20, 70/30, etc) for cross validation?

I hear a lot about overfitting. My understanding is that overfitting should never be an issue for a k-nn model. Or, thinking from a slightly different perspective, k-nn models are by definition overfit. The only way to improve model performance when we feed it new (unlabeled) observations, is to collect additional training data (labeled data) and refit the model, rather than refit model parameters on the original data...do I have this correct?