ADMFirstChallenge

```r
#Importing Dataset
mnist <- read_mnist("~/Mscs 341 S22/Class/Data")
str(mnist)
```

```
## List of 2
##  $ train:List of 2
##   ..$ images: int [1:60000, 1:784] 0 0 0 0 0 0 0 0 0 0 ...
##   ..$ labels: int [1:60000] 5 0 4 1 9 2 1 3 1 4 ...
##  $ test :List of 2
##   ..$ images: int [1:10000, 1:784] 0 0 0 0 0 0 0 0 0 0 ...
##   ..$ labels: int [1:10000] 7 2 1 0 4 1 4 9 5 9 ...
```

Dataset Creation

- **Your dataset should have in total 1000 randomly selected digits (feel free to use a set.seed command so that your results are reproducible).**

First, calculate values for an individual image:

```r
set.seed(12345) #For Reproducible Values
```

```r
index <- sample(1:60000, 60000) #vector of randomly selected indexes to sample from mnist

tester <-
  as_tibble(index) %>%
  mutate(image = mnist$train$images[value, ]) %>%
  mutate(label = mnist$train$labels[value]) %>%
  dplyr::filter(label == 0 | label == 7)

tester$image[10,] #this pulls out a single image. This will be plotted to confirm if it's correct
```

```
##   [1]   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
##  [19]   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
##  [37]   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
##  [55]   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
##  [73]   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
##  [91]   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
## [109]   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
## [127]   0   0  86 255 226 170  29   0   0   0   0   0   0   0   0   0   0   0
## [145]   0   0   0   0   0   0   0   0   0   0   0  86 255 170 170 255 226 114
## [163]   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
## [181]   0   0   0 255 226   0   0   0 170 255  57   0   0   0   0   0   0   0
## [199]   0   0   0   0   0   0   0   0 114 255 198  29   0 255  57   0   0   0
## [217]   0 255 170   0   0   0   0   0   0   0   0   0   0   0   0   0  57 198
## [235] 255 198 255 226   0 255  86   0   0   0   0 170 226   0   0   0   0   0
## [253]   0   0   0   0   0   0   0   0 255 255  86   0 198 255  29  86 170   0
## [271]   0   0   0 141 255  29   0   0   0   0   0   0   0   0   0   0   0  29
```

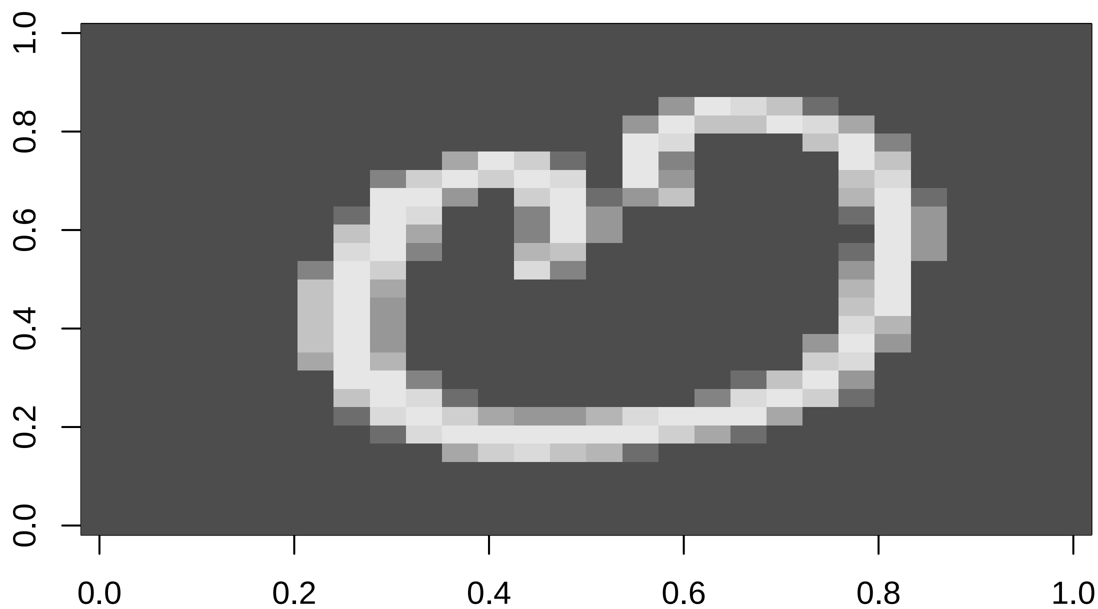
```
## [289] 255 226   0   0  57 255  86   0   0   0   0   0   0  29 255  86   0   0
## [307]   0   0   0   0   0   0   0   0   0 170 255 114   0   0  57 255  86   0
## [325]   0   0   0   0   0   0 255  86   0   0   0   0   0   0   0   0   0   0
## [343]   0 226 255  57   0   0 141 170   0   0   0   0   0   0   0  29 255  86
## [361]   0   0   0   0   0   0   0   0   0   0  57 255 198   0   0   0 226  57
## [379]   0   0   0   0   0   0   0  86 255   0   0   0   0   0   0   0   0   0
## [397]   0   0 170 255 114   0   0   0   0   0   0   0   0   0   0   0   0 141
## [415] 255   0   0   0   0   0   0   0   0   0   0   0 170 255  86   0   0   0
## [433]   0   0   0   0   0   0   0   0   0 170 255   0   0   0   0   0   0   0
## [451]   0   0   0   0 170 255  86   0   0   0   0   0   0   0   0   0   0   0
## [469]   0 226 141   0   0   0   0   0   0   0   0   0   0   0 170 255  86   0
## [487]   0   0   0   0   0   0   0   0   0   0  86 255  86   0   0   0   0   0
## [505]   0   0   0   0   0   0 114 255 141   0   0   0   0   0   0   0   0   0
## [523]   0   0 198 226   0   0   0   0   0   0   0   0   0   0   0   0   0 255
## [541] 255  57   0   0   0   0   0   0   0   0  29 170 255  86   0   0   0   0
## [559]   0   0   0   0   0   0   0   0   0 170 255 226  29   0   0   0   0   0
## [577]   0  57 226 255 198  29   0   0   0   0   0   0   0   0   0   0   0   0
## [595]   0  29 226 255 198 114  86  86 141 226 255 255 255 114   0   0   0   0
## [613]   0   0   0   0   0   0   0   0   0   0   0   0  29 226 255 255 255 255
## [631] 255 255 198 114  29   0   0   0   0   0   0   0   0   0   0   0   0   0
## [649]   0   0   0   0   0   0 114 198 226 170 141  29   0   0   0   0   0   0
## [667]   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
## [685]   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
## [703]   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
## [721]   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
## [739]   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
## [757]   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
## [775]   0   0   0   0   0   0   0   0   0   0
```

```
tester$label[10] #Obtaining label to see if it matches with image. This is labelled 0.
```

```
## [1] 0
```

Plot the image to ensure data is extracted correctly. It should be a 0.

```
#Function that makes the image
plotImage <- function(dat,size=28){
  imag <- matrix(dat,nrow=size)[,28:1]
  image(imag,col=grey.colors(256), xlab = "", ylab="")
}

#Plotting the image of 0.
plotImage(tester$image[10,])
```



Looks like a 0. Data above was extracted correctly.

- **Your training dataset should have 800 observations and your testing should have 200 observations.**

Dividing the dataset into testing and training:

```
#Getting 1000 rows
tester <- tester %>%
  slice(1:1000)

#Setting Seed
set.seed(12345)

#Splitting into training and testing with respective ratio of 4:.1
trainer <- slice_sample(tester, n = 800)
tester <- setdiff(tester, trainer)
```

- Use the mnist dataset from dslabs to create an end-to-end classifier that distinguishes between 7 and 0.

Feature Definition

- **You are allowed to use only 2 features. Notice that you need to calculate those features directly from dataset. Make sure to describe what those features represent and why you chose them. Are those features capturing any intuition that you have about distinguishing those two digits?**

To determine the best way to calculate `x_1` and `x_2`, the quadrants will be calculated individually (q1, q2, q3, and q4) so they can be used to calculate different possible combinations of `x_1` and `x_2`. We will test `x_1` and `x_2` based on which appears to work best when plotted on a scatter plot.

Features:

Feature 1: We test `x_1 = q1+q3-q2-q4` as a measure of symmetry. 0 should be much more symmetrical than 7, so values of `x_1` should be smaller for 0.

Feature 2: `x_2` is the sum of darkness in pixels of the top left quadrant `q2`. We expect this to be smaller for 7 than for 0 because the top-left area of a 7 is not very large.

First, the four quadrants will be calculated. Choose a single vector of 0: (will also do this process for 7 as a comparison: tester$label[1])

```
zeroMatrix <- matrix(trainer$image[4, ],nrow=28)[,28:1]
sum(zeroMatrix[1:14, 15:28])  #q1
sum(zeroMatrix[1:14, 1:14])   #q2
sum(zeroMatrix[15:28, 1:14])  #q3
sum(zeroMatrix[15:28, 15:28]) #q4

#Observing the values of the quadrants+
abs(sum(zeroMatrix[1:14, 15:28]) + sum(zeroMatrix[15:28, 1:14]) - sum(zeroMatrix[1:14, 1:14]) - sum(zer
```

```
q1 = 6588;    q2 = 8456;    q3 = 7982;    q4 = 8562
```

Repeating this process above for 7:

```
sevenMatrix <- matrix(trainer$image[1, ],nrow=28)[,28:1]
sum(sevenMatrix[1:14, 15:28])  #q1
sum(sevenMatrix[1:14, 1:14])   #q2
sum(sevenMatrix[15:28, 1:14])  #q3
sum(sevenMatrix[15:28, 15:28]) #q4

abs(sum(sevenMatrix[1:14, 15:28]) + sum(sevenMatrix[15:28, 1:14]) - sum(sevenMatrix[1:14, 1:14]) - sum(s
```

```
q1 = 4821;    q2 = 1757;    q3 = 7921;    q4 = 4487
```

The approach above (`q1+q3-q2-q4`) gives 2448 for the zero and 6498 for the 7 which is about what we would expect - ideally the value returned for the zero image would be closer to zero. Similarly, the difference in the values of `q1` and `q2` is stark and apparent.

Next, calculate these values for all images and put it in a table:

Making the training dataset

```
#note that directly mutating these values into trainer or tester does not work. need to use a for-loop

trainVectorQ1 <- vector()#making sure the variables are clear
trainVectorQ2 <- vector()
trainVectorQ3 <- vector()
trainVectorQ4 <- vector()

for (i in 1:800) {#note: this value should be 1:200 for tester
  sumMatrix <- matrix(trainer$image[i, ], nrow = 28)[,28:1]
  q1 = sum(sumMatrix[1:14, 15:28])#q1
```

```
  print(q1)
  trainVectorQ1 <- c(trainVectorQ1, q1)
  trainVectorQ1
}
for (i in 1:800) {
  sumMatrix <- matrix(trainer$image[i, ], nrow = 28)[,28:1]
  q2 = sum(sumMatrix[1:14, 1:14])#q2
  trainVectorQ2 <- c(trainVectorQ2, q2)
  trainVectorQ2
}
for (i in 1:800) {
  sumMatrix <- matrix(trainer$image[i, ], nrow = 28)[,28:1]
  q3 = sum(sumMatrix[15:28, 1:14])#q3
  trainVectorQ3 <- c(trainVectorQ3, q3)
  trainVectorQ3
}
for (i in 1:800) {
  sumMatrix <- matrix(trainer$image[i, ], nrow = 28)[,28:1]
  q4 = sum(sumMatrix[15:28, 15:28])#q4
  trainVectorQ4 <- c(trainVectorQ4, q4)
  trainVectorQ4
}
```

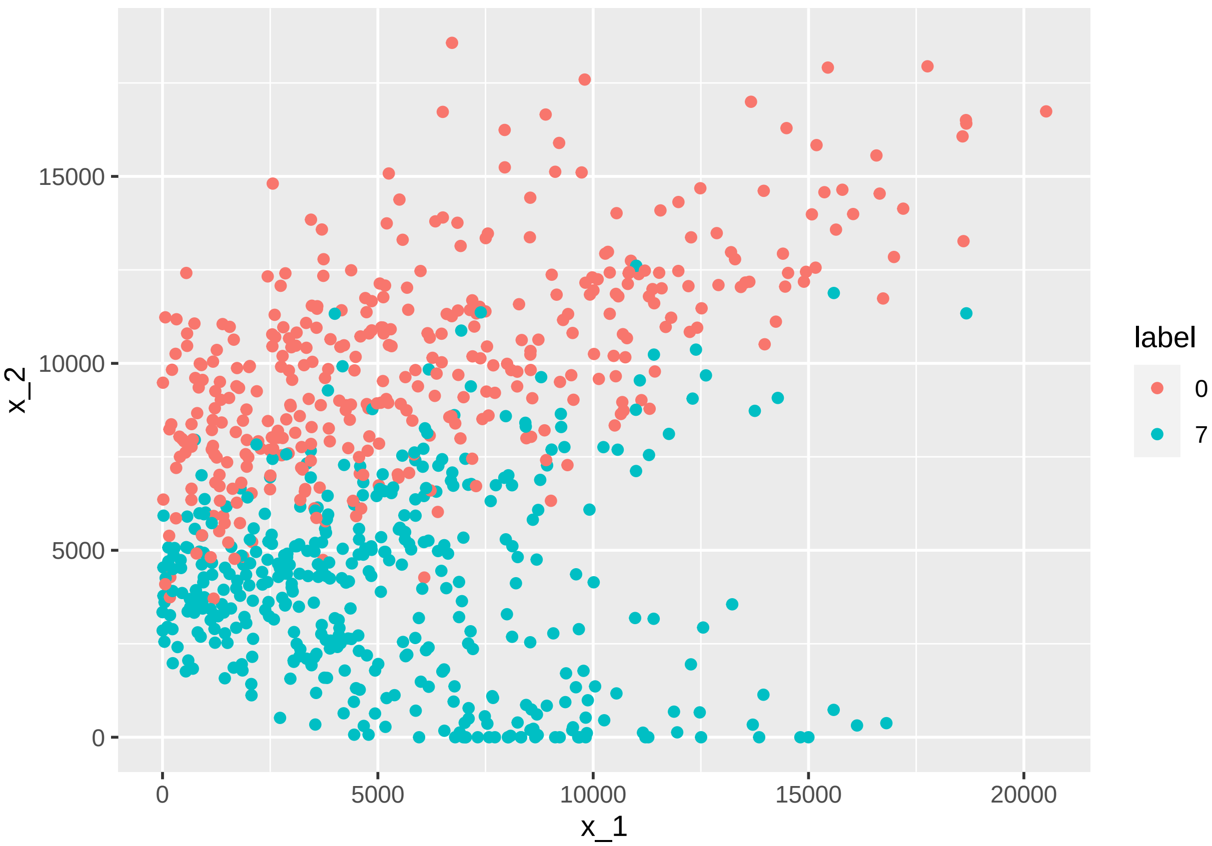
```
trainer <- trainer %>%
  select(label) %>%
  mutate(label=as.factor(label)) %>%
  mutate(row = row_number()) %>%
  mutate(q1 = trainVectorQ1) %>%
  mutate(q2 = trainVectorQ2) %>%
  mutate(q3 = trainVectorQ3) %>%
  mutate(q4 = trainVectorQ4) %>%
  mutate(x_1 = abs(q1+q3-q2-q4)) %>%
  mutate(x_2 = q2)

trainer
```

```
## # A tibble: 800 x 8
##    label   row    q1    q2    q3    q4   x_1   x_2
##    <fct> <int> <int> <int> <int> <int> <int> <int>
##  1 7         1  4821  1757  7921  4487  6498  1757
##  2 7         2  6928   454  8455  4673 10256   454
##  3 0         3  2983  9026  5788  9287  9542  9026
##  4 0         4  6588  8456  7982  8562  2448  8456
##  5 0         5  2339  4741  3223  4547  3726  4741
##  6 0         6  9425 13349 10118 13699  7505 13349
##  7 7         7  1153  4153   893  4778  6885  4153
##  8 0         8  5438  9046  6584  8171  5195  9046
##  9 7         9  8020  3577  7285  8864  2864  3577
## 10 7        10  3584  7693  3360  8285  9034  7693
## # ... with 790 more rows
```

Plot the classifers:

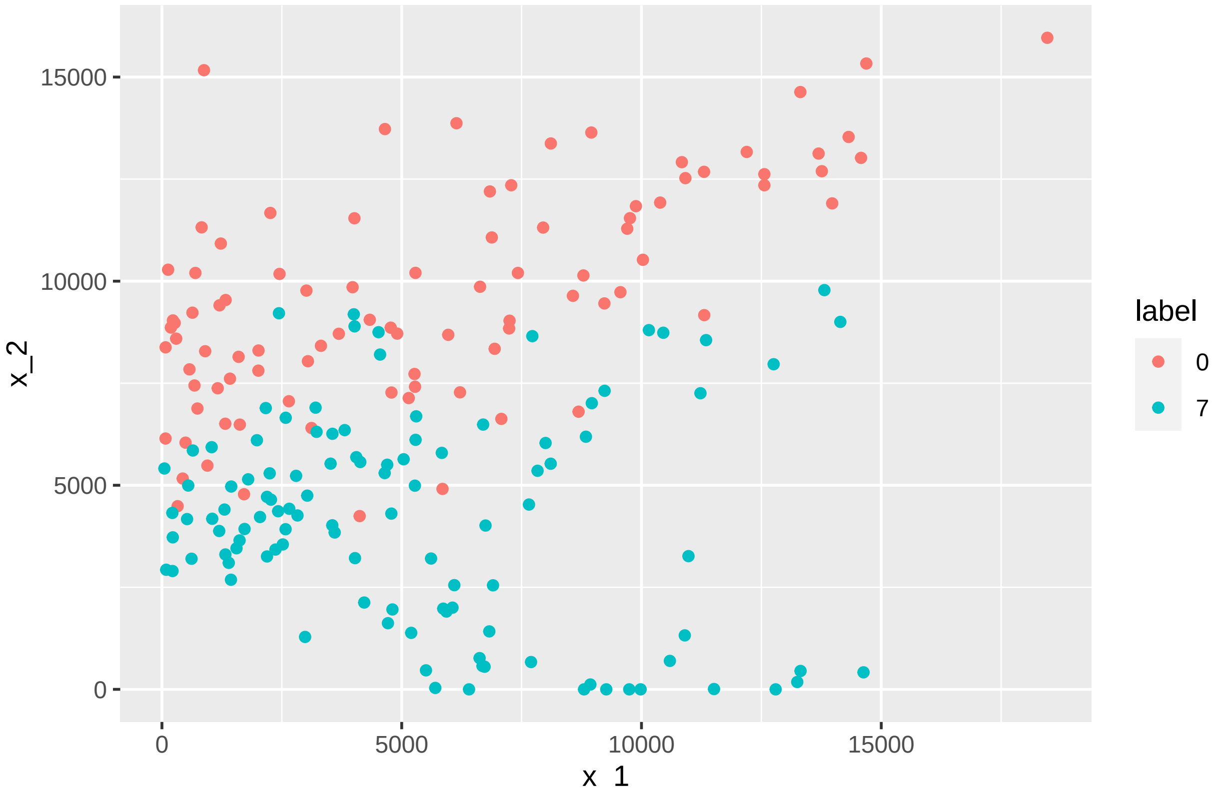
```
trainer %>%
  ggplot(aes(x = x_1, y = x_2, color = label)) +
```

```
geom_point()
```



`x_1` and `x_2` appear to successfully separate 0 and 7. There is not much overlap between the coordinate points of 0 and 7.

Repeat the process above to create the tester tibble and plot:




```
head(tester)
```

```
## # A tibble: 6 x 8
##   label   row    q1    q2    q3    q4   x_1   x_2
##   <fct> <int> <int> <int> <int> <int> <int> <int>
## 1 7         1  2982  6035  2338  7285  8000  6035
## 2 7         2  7729  4013 10411  7379  6748  4013
## 3 0         3  5911  4780  6440  5859  1712  4780
## 4 7         4 12430   417 12408  9791 14630   417
## 5 0         5  7667  4910  8360  5266  5851  4910
## 6 7         6  6891  1977  7587  6635  5866  1977
```

This looks similar to the plot for trainer which is what is expected. Since there are fewer points, the difference between the points for 0 and 7 are not as defined. Next, these tables will be used to create and test models.

Model creation, optimization, and selection

- **Create at least two different models for this classification and make sure to optimize the parameters those models have.**
- **Calculate the misclassification rates for both models and select the model with the lowest error rate.**

Model #1

The first model we will be using is the logistic regression model from parsnip in tidymodels.

```
logit.model <- logistic_reg() %>%
  set_engine("glm") %>%
  set_mode("classification")

default.recipe <-
  recipe(label ~ x_1+x_2, data=trainer)

logit.wflow <- workflow() %>%
  add_recipe(default.recipe) %>%
  add_model(logit.model)

logit.fit <- fit(logit.wflow, trainer)
logit.fit
```

Now, this model can be used on the tester dataset to classify whether images are 0 or 7 (in addition to the probability of being a value being 0 or 7).

```
predict(logit.fit, tester, type = "prob") #Gives probability of 0 and 7
```

```
## # A tibble: 200 x 2
##       .pred_0 .pred_7
##         <dbl>   <dbl>
##  1 0.0778       0.922
##  2 0.0122       0.988
##  3 0.166        0.834
##  4 0.00000933   1.00
##  5 0.0473       0.953
##  6 0.00154      0.998
##  7 0.00206      0.998
##  8 0.884        0.116
##  9 0.00223      0.998
## 10 0.000208     1.00
## # ... with 190 more rows
```

```
predict(logit.fit, tester) #Gives the classsification
```

```
## # A tibble: 200 x 1
##    .pred_class
##    <fct>
##  1 7
##  2 7
##  3 7
##  4 7
##  5 7
##  6 7
##  7 7
##  8 0
##  9 7
## 10 7
## # ... with 190 more rows
```

Calculate the misclassification rate:

```
misclassification.tbl <- augment(logit.fit, tester)
mean(misclassification.tbl$label != misclassification.tbl$.pred_class)
```

```
## [1] 0.115
```

The misclassification rate is 11.5%.

Model #2

Now, we are going to use K nearest neighbours (knn) from parsnip in tidymodels. `nearest_neighbor()` uses k = 5 as default, let us attempt to optimize it

```
library(kknn)

#Making the model
knn.model <- nearest_neighbor(neighbors = tune()) %>%
  set_engine("kknn") %>%
  set_mode("classification")

#Making the workflow
knn.wflow <- workflow() %>%
  add_recipe(default.recipe) %>%
  add_model(knn.model)
```

Optimizing K:

```
#Making 10-fold cross-validation dataset
digits.folds <- vfold_cv(trainer, v = 10)
training(digits.folds$splits[2][[1]])
```

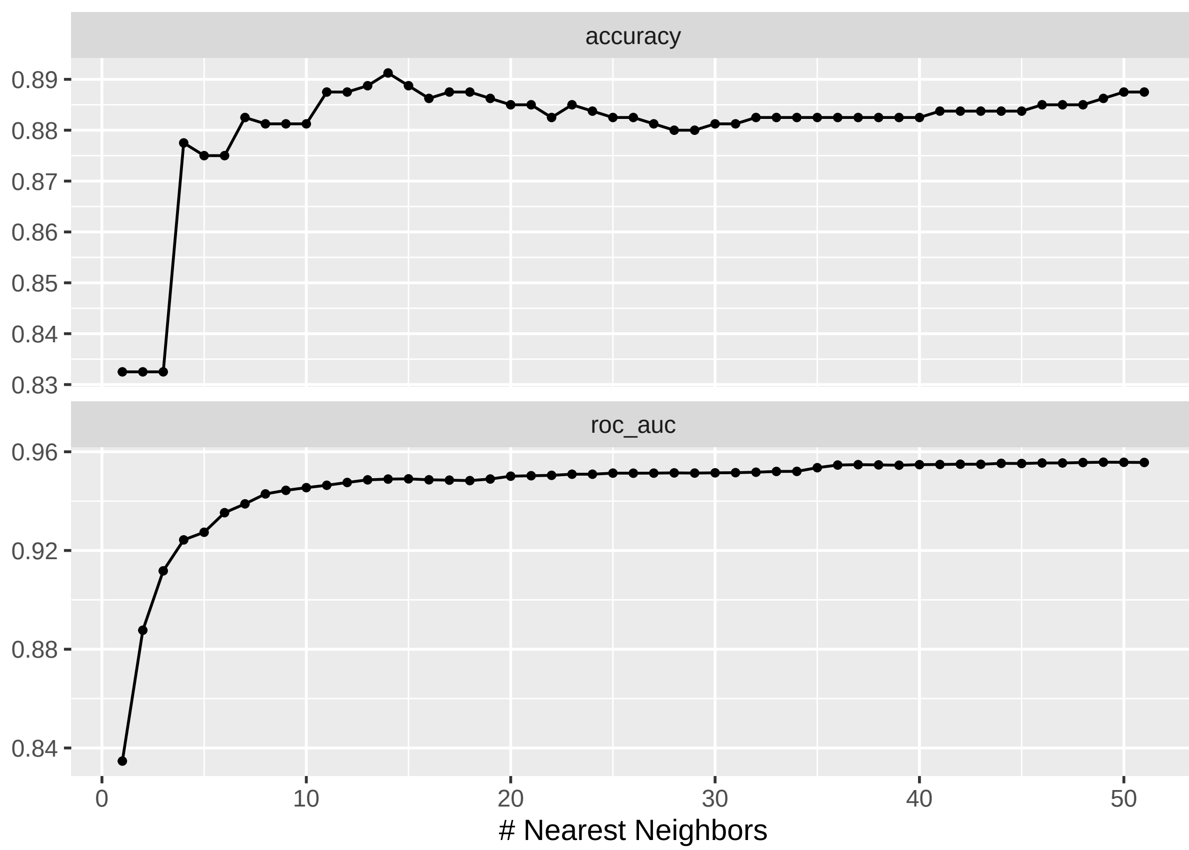
```
## # A tibble: 720 x 8
##    label   row    q1    q2    q3    q4   x_1   x_2
##    <fct> <int> <int> <int> <int> <int> <int> <int>
##  1 7         1  4821  1757  7921  4487  6498  1757
##  2 7         2  6928   454  8455  4673 10256   454
##  3 0         4  6588  8456  7982  8562  2448  8456
##  4 7         7  1153  4153   893  4778  6885  4153
##  5 0         8  5438  9046  6584  8171  5195  9046
##  6 7         9  8020  3577  7285  8864  2864  3577
##  7 7        10  3584  7693  3360  8285  9034  7693
##  8 0        11  4188  7051  6351  8067  4579  7051
##  9 7        12  6906  5074  6678  8374   136  5074
## 10 7        13  4403  7409  5452  8317  5871  7409
## # ... with 710 more rows
```

```
testing(digits.folds$splits[2][[1]])
```

```
## # A tibble: 80 x 8
##    label   row    q1    q2    q3    q4   x_1   x_2
##    <fct> <int> <int> <int> <int> <int> <int> <int>
##  1 0         3  2983  9026  5788  9287  9542  9026
##  2 0         5  2339  4741  3223  4547  3726  4741
##  3 0         6  9425 13349 10118 13699  7505 13349
##  4 0        21 10002 11298  9598 10906  2604 11298
##  5 0        31  8578 11081  7889  8720  3334 11081
##  6 7        32  7112     0  8203  5490  9825     0
```

```
##   7 0         33   4981 11160   7157 10279   9301 11160
##   8 7         36   2586  3558   5306  5712   1378  3558
##   9 0         48   6255 12299   8765 12700   9979 12299
## 10 7         52   4168  6451   3332  7120   6071  6451
## # ... with 70 more rows
```

```r
#Making grid of neighbours across values of K
neighbors.tbl <-  tibble(neighbors = seq(1,51, by = 1))
neighbors.grid.tbl <- grid_regular(neighbors(range = c(1, 51)),
                                   levels = 51)

#Tuning the results accordingly
tune.results <- tune_grid(object = knn.wflow,
                          resamples = digits.folds,
                          grid = neighbors.tbl)

#Having a look at the values of K
autoplot(tune.results)
```




```r
#Show the best Value of K
show_best(tune.results, metric = "accuracy")
```

```
## # A tibble: 5 x 7
##   neighbors .metric  .estimator  mean     n std_err .config
##       <dbl> <chr>    <chr>      <dbl> <int>   <dbl> <fct>
## 1        14 accuracy binary     0.891    10 0.0110  Preprocessor1_Model14
## 2        13 accuracy binary     0.889    10 0.0116  Preprocessor1_Model13
```

```
## 3        15 accuracy binary     0.889    10 0.0106  Preprocessor1_Model15
## 4        11 accuracy binary     0.888    10 0.00986 Preprocessor1_Model11
## 5        12 accuracy binary     0.888    10 0.0113  Preprocessor1_Model12
```

```
best.neighbor <- select_best(tune.results, metric = "accuracy")

#Applying the optimal value of K (14)
knn.final.wflow <- finalize_workflow(knn.wflow, best.neighbor)
knn.fit <- fit(knn.final.wflow, trainer)
```

Finally, getting missclassification rate

```
predict(knn.fit, tester, type = "prob")
```

```
## # A tibble: 200 x 2
##    .pred_0 .pred_7
##      <dbl>   <dbl>
##  1  0.0816  0.918
##  2  0.0867  0.913
##  3  0.332   0.668
##  4  0       1
##  5  0.0255  0.974
##  6  0       1
##  7  0       1
##  8  0.923   0.0765
##  9  0       1
## 10  0       1
## # ... with 190 more rows
```

```
predict(knn.fit, tester)
```

```
## # A tibble: 200 x 1
##    .pred_class
##    <fct>
##  1 7
##  2 7
##  3 7
##  4 7
##  5 7
##  6 7
##  7 7
##  8 0
##  9 7
## 10 7
## # ... with 190 more rows
```

```
misclassification.tbl <- augment(knn.fit, tester)
mean(misclassification.tbl$label != misclassification.tbl$.pred_class)
```

```
## [1] 0.125
```

Result of the models: Logistic Regression gave a misclassification rate of 11.5% and Knn gave 12.5%. Hence Logistic is marginally better. However logistc works with only 2 variables. Hence another model will be tested.

Model #3

```
library(tidymodels)
library(discrim)
tidymodels_prefer()

lda.model <- discrim_linear() %>%
  set_engine("MASS") %>% #MASS is the library, or a type of implementation
  set_mode("classification")

lda.wflow <- workflow() %>%
  add_recipe(default.recipe) %>%
  add_model(lda.model)

lda.fit <- fit(lda.wflow, trainer)
lda.fit
```

```
## == Workflow [trained] ==========================================================
## Preprocessor: Recipe
## Model: discrim_linear()
##
## -- Preprocessor ----------------------------------------------------------------
## 0 Recipe Steps
##
## -- Model -----------------------------------------------------------------------
## Call:
## lda(..y ~ ., data = data)
##
## Prior probabilities of groups:
##       0       7
## 0.47375 0.52625
##
## Group means:
##        x_1       x_2
## 0 6084.715 10057.380
## 7 5187.404  4033.012
##
## Coefficients of linear discriminants:
##               LD1
## x_1  6.103812e-05
## x_2 -3.894835e-04
```

```
predict(lda.fit, tester, type="prob")
```

```
## # A tibble: 200 x 2
##     .pred_0 .pred_7
##       <dbl>   <dbl>
##  1 0.208      0.792
##  2 0.0489     0.951
##  3 0.171      0.829
##  4 0.000689   0.999
##  5 0.115      0.885
##  6 0.00937    0.991
##  7 0.0134     0.987
```

```
## 8 0.828      0.172
## 9 0.0107     0.989
## 10 0.00235   0.998
## # ... with 190 more rows
```

```
predict(lda.fit, tester)
```

```
## # A tibble: 200 x 1
##    .pred_class
##    <fct>
##  1 7
##  2 7
##  3 7
##  4 7
##  5 7
##  6 7
##  7 7
##  8 0
##  9 7
## 10 7
## # ... with 190 more rows
```

```
misclassification.tbl <- augment(lda.fit, tester)
mean(misclassification.tbl$label != misclassification.tbl$.pred_class)
```

```
## [1] 0.125
```

Misclassification = 12.5%

Visualization

- **Plot the probabilities across a grid and the decision boundary for your selected model**

As shown above, both Knn and LDA show an equally good misclassification rate For the purpose of this project, we are going to use Knn. The probability of an image being 7 will be plotted with a decision boundary.

First, we found max and min values for `x_1` and `x_2` to create a grid for plotting with Bayes' boundary. Using max and min, the largest `x_1` is 20518, and the largest `x_2` is 18572. We create grids using seq up to 21000.

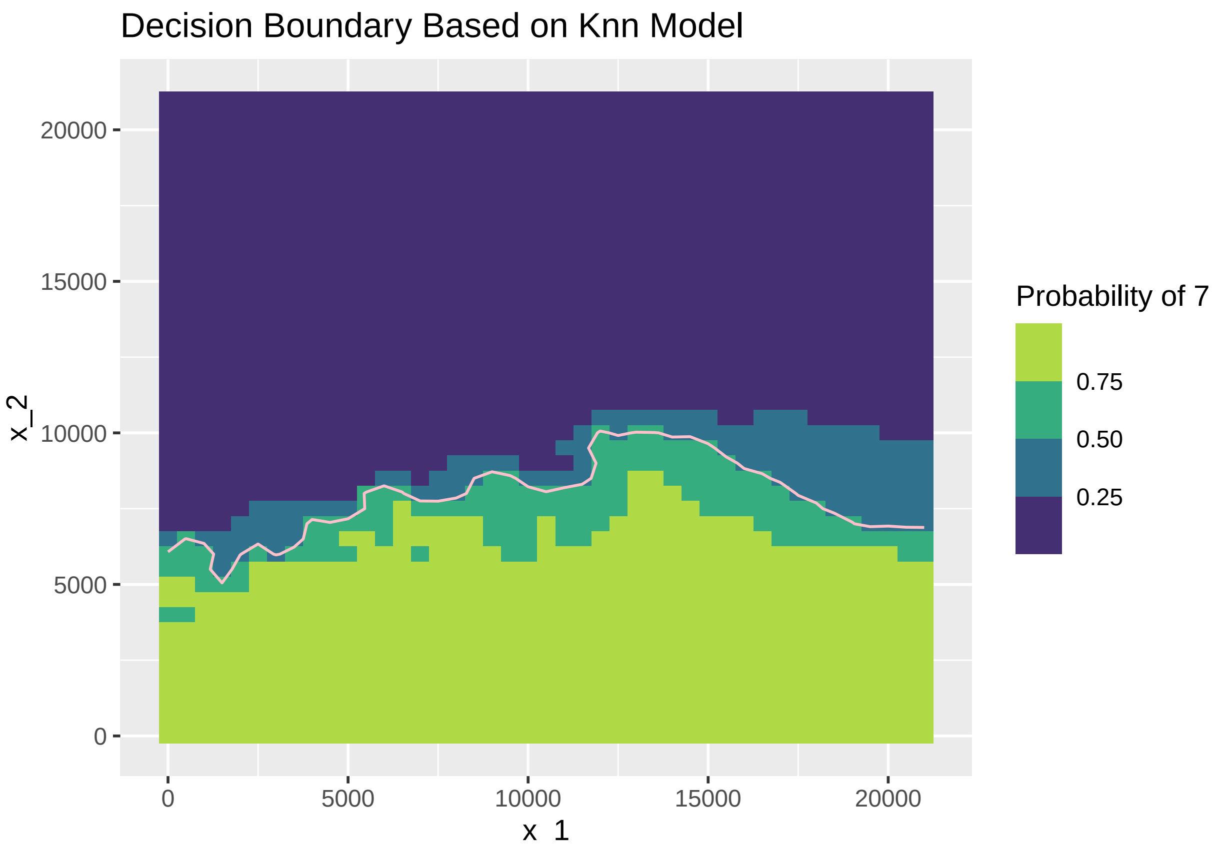
```
grid.vec.x_1 <- seq(from = 0, to = 21000, by = 500)
grid.vec.x_1
```

```
##  [1]     0   500  1000  1500  2000  2500  3000  3500  4000  4500  5000  5500
## [13]  6000  6500  7000  7500  8000  8500  9000  9500 10000 10500 11000 11500
## [25] 12000 12500 13000 13500 14000 14500 15000 15500 16000 16500 17000 17500
## [37] 18000 18500 19000 19500 20000 20500 21000
```

```
grid.vec.x_2 <- seq(from = 0, to = 21000, by = 500)
grid.tbl <- expand_grid(x_1 = grid.vec.x_1, x_2 = grid.vec.x_2)
```

```
pred <- predict(knn.fit, grid.tbl, type="prob")#probability
pred %>%
  mutate(x_1 = grid.tbl$x_1) %>%
  mutate(x_2 = grid.tbl$x_2) %>%
  ggplot(aes(x_1, x_2, z=.pred_7,fill = .pred_7)) +
    geom_raster() +
```

```
    stat_contour(breaks=c(0.5), color="pink")+
    scale_fill_viridis_b()+
    labs(title = "Decision Boundary Based on Knn Model",
         fill = "Probability of 7")
```



Changing things up: adding the number 5

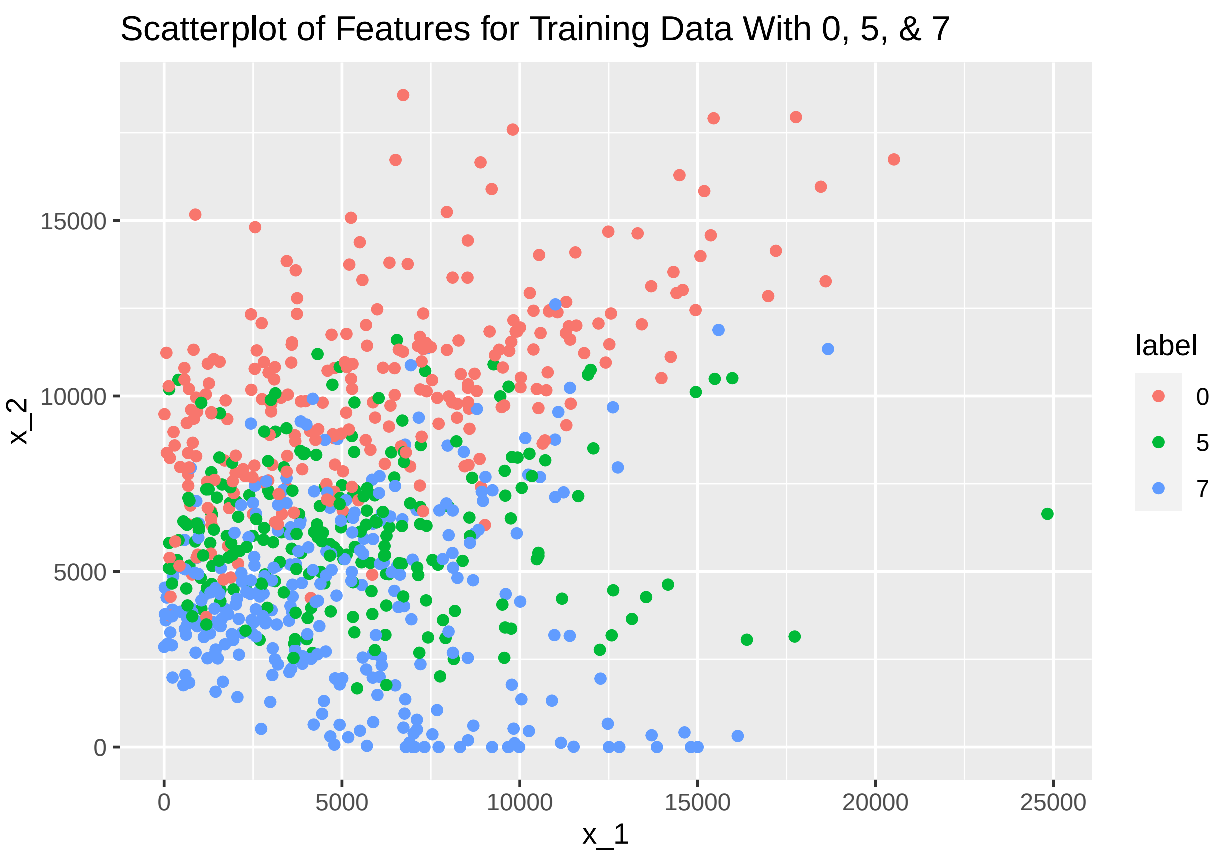
- **Create a new dataset that includes your two chosen digits and the digit 5. Create training and testing datasets that include 5 and your two given digits.**
- **Calculate the same 2 features for this new testing and training dataset.**

We copy and paste the code from near the beginning of this document and then plot it to confirm that everything looks as it should. This time, 5 is also included in the testing and training tables.

Training Dataset:

Having a look at the split between the parameters and the numbers

```
trainer %>%
  ggplot(aes(x = x_1, y = x_2, color = label)) +
    geom_point()+
    labs(title="Scatterplot of Features for Training Data With 0, 5, & 7")
```



5 Appears to be right in the middle of the split of 0 and 7.

Testing Dataset:

```
#Making the model
knn.model <- nearest_neighbor(neighbors = tune()) %>%
  set_engine("kknn") %>%
  set_mode("classification")

#Making the new Recipe
default.recipe <- recipe(label ~ x_1+x_2, data=trainer)

#Making the workflow
knn.wflow <- workflow() %>%
  add_recipe(default.recipe) %>%
  add_model(knn.model)
```

Optimizing K:

```
#Making 10-fold cross-validation dataset
digits.folds <- vfold_cv(trainer, v = 10)
training(digits.folds$splits[2][[1]])
```

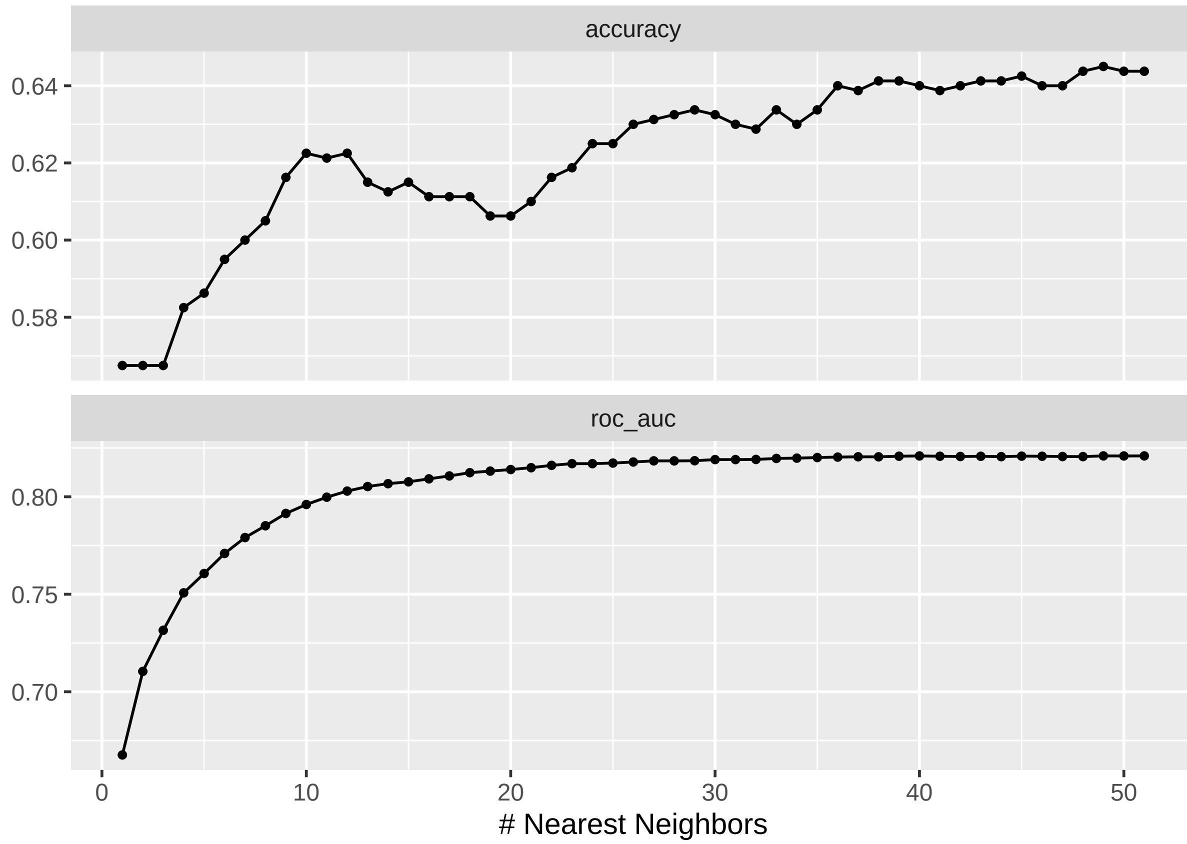
```
## # A tibble: 720 x 8
##    label   row    q1    q2    q3    q4   x_1   x_2
##    <fct> <int> <int> <int> <int> <int> <int> <int>
##  1 5         1  5321  8600  3187  7124  7216  8600
##  2 5         2  9885  6952  9312 10396  1849  6952
##  3 5         4  7586  7314 11195  8544  2923  7314
```

```
##   4 5         7  5481  8984  8312  7936  3127  8984
##   5 7         8 11314  4358 12376  9729  9603  4358
##   6 7         9  7966     0 11823  7284 12505     0
##   7 7        10  7344     8  9854  5677 11513     8
##   8 5        11  9911  5725 10106  8094  6198  5725
##   9 5        12  9662 10191 11176 10790   143 10191
##  10 0        13  6611 12409  7096 12121 10823 12409
## # ... with 710 more rows
```

```r
testing(digits.folds$splits[2][[1]])
```

```
## # A tibble: 80 x 8
##    label   row    q1    q2    q3    q4   x_1   x_2
##    <fct> <int> <int> <int> <int> <int> <int> <int>
##   1 0         3 11956 15078 13652 15784  5254 15078
##   2 0         5  6134  9726  8635 11407  6364  9726
##   3 0         6  9180  8162  8124 10844  1702  8162
##   4 7        21  4522  3276  4380  6828  1202  3276
##   5 5        31  8774  5262  7823  5785  5550  5262
##   6 5        32  3104  4444  4526  4493  1307  4444
##   7 7        33  2376  1980  3842  4000   238  1980
##   8 0        36  5925 11427  8509 10144  7137 11427
##   9 7        48  4409  7759  3274 10162 10238  7759
##  10 0        52  5588 10029  7208  9249  6482 10029
## # ... with 70 more rows
```

```r
#Making grid of neighbours across values of K
neighbors.tbl <-  tibble(neighbors = seq(1,51, by = 1))
neighbors.grid.tbl <- grid_regular(neighbors(range = c(1, 51)),
                                   levels = 51)

#Tuning the results accordingly
tune.results <- tune_grid(object = knn.wflow,
                          resamples = digits.folds,
                          grid = neighbors.tbl)

#Having a look at the values of K
autoplot(tune.results)
```




```r
#Show the best Value of K
show_best(tune.results, metric = "accuracy")
```

```
## # A tibble: 5 x 7
##   neighbors .metric  .estimator  mean     n std_err .config
##       <dbl> <chr>    <chr>      <dbl> <int>   <dbl> <fct>
## 1        49 accuracy multiclass 0.645    10  0.0155 Preprocessor1_Model49
## 2        48 accuracy multiclass 0.644    10  0.0157 Preprocessor1_Model48
## 3        50 accuracy multiclass 0.644    10  0.0153 Preprocessor1_Model50
## 4        51 accuracy multiclass 0.644    10  0.0153 Preprocessor1_Model51
## 5        45 accuracy multiclass 0.643    10  0.0148 Preprocessor1_Model45
```

```r
best.neighbor <- select_best(tune.results, metric = "accuracy")

#Applying the optimal value of K (14)
knn.final.wflow <- finalize_workflow(knn.wflow, best.neighbor)
knn.fit <- fit(knn.final.wflow, trainer)
```

Calculating Misclassification rate:

```r
augment(knn.fit, tester) %>%
  accuracy(truth = label, estimate = .pred_class)
```

```
## # A tibble: 1 x 3
##   .metric  .estimator .estimate
##   <chr>    <chr>          <dbl>
## 1 accuracy multiclass     0.665
```

We get a misclassification of 33.5%

Confusion Matrix:

```
augment(knn.fit, tester) %>%
  conf_mat(truth = label, estimate = .pred_class)
```

```
##           Truth
## Prediction  0  5  7
##          0 56 13  5
##          5  9 29 20
##          7  3 17 48
```

5 appears to be problematic as a lot of them seem to be confused for 7s and a comparable amount seem to be confused with 0s. However, the model seems to work especially well for 0 and 7 isn't that bad either. 7 is very rarely confused for a 0 but is often confused for a 5.

Results:

Accuracy of 0: `56/(56+9+3) = 0.8235 -> 82.3%` Very good!

Accuracy of 5: `29/(13+29+17) = 0.4915 -> 49.1%` Mediocre

Accuracy of 7: `48/(5+20+48) = 0.6575 -> 65.7%` Fair

As we might expect, since our features are based on 0 and 7, the 5s are misclassified the most. Out of the misclassifications of 5s, $26/37 = 70\%$ were classified as 7s. The other 30% were misclassified as 0s. The reason why our model thinks 5s are 7s more often is because, like the average 7, the average 5 is not going to be perfectly symmetric, and we can expect most 5s to have some of their image in the upper left corner. If we ran our model with just 0s and 5s, we would expect to have a misclassification rate similar to our model for classifying 0s and 7s.

Creating the Grid:

```
create_grid <- function(delta) {
  expand_grid(x_1=seq(0,21000, by=delta), x_2 = seq(0,21000, by=delta))
}
grid.tbl <- create_grid(200)
grid.tbl <- grid.tbl %>%
  mutate(x_1 = as.integer(x_1)) %>%
  mutate(x_2 = as.integer(x_2))

augment.tbl <- predict(knn.fit, grid.tbl) %>%
  mutate(row = row_number())
augment2.tbl <- predict(knn.fit, grid.tbl, type = "prob") %>%
  mutate(row = row_number())

augment.tbl <- full_join(augment.tbl, augment2.tbl, by = "row") %>%
  mutate(x_1 = grid.tbl$x_1) %>%
  mutate(x_2 = grid.tbl$x_2)
augment.tbl
```

```
## # A tibble: 11,236 x 7
##    .pred_class   row .pred_0 .pred_5 .pred_7   x_1   x_2
##    <fct>       <int>   <dbl>   <dbl>   <dbl> <int> <int>
##  1 7               1 0.00916  0.0183   0.973     0     0
##  2 7               2 0.0117   0.0208   0.968     0   200
##  3 7               3 0.0150   0.0208   0.964     0   400
##  4 7               4 0.0158   0.0225   0.962     0   600
```

```
##   5 7              5 0.0175   0.0233   0.959     0   800
##   6 7              6 0.0183   0.0250   0.957     0  1000
##   7 7              7 0.0200   0.0283   0.952     0  1200
##   8 7              8 0.0217   0.0292   0.949     0  1400
##   9 7              9 0.0233   0.0333   0.943     0  1600
## 10 7             10 0.0242   0.0350   0.941     0  1800
## # ... with 11,226 more rows
```

Plot Grid With Boundary

```
augment.tbl %>%
    ggplot() +
      geom_raster(aes(x_1, x_2, fill = .pred_class)) +
      geom_point(data=tester, aes(x=x_1, y=x_2, color=label, shape=label))+
      scale_color_manual(values=c("blue","red","orange"))+
      labs(title="Decision Boundary of LDA for 0, 5, & 7",
           fill = "Predicted Class",
           shape = "Label")
```