Kevin Gong
kg2445
HW03
Stat W4240
Section 2

Homework 3

Problem #1 (James 3.7.3)

Part a)

The correct statement is iii (#3). Writing out our regression equation, we find that:

Y = (35 – 10*GPA)*Gender + constant

Where Gender = 0 indicates male. When GPA is high (35 – 10*GPA <0, or GPA>3.5), female salaries will be slightly below those of males since (35 – 10*GPA) is negative. This causes the earnings of females to be lower than those of males provided that GPA is sufficiently high, and thus the correct answer is iii (#3).

Part b)

IQ = 110, GPA = 4.0

Y = (35 – 10*GPA)*Gender + (50 + 20*GPA+0.07*IQ + 0.01*GPA*IQ)

Plugging in and calculating -> The predicted salary of the female with an IQ of 110 and GPA of 4.0 is **137.1 (in thousands of dollars)**.

Part c)

False. To test for significance of an interaction effect, we must look at the p values associated with either the T-stat or F-stat of the term. The size of the coefficient itself is not indicative of its significant.

Problem #2 (James 3.7.4)

Part a)

There is not enough information to tell. Compared to training RSS for a linear regression, training RSS for a cubic regression may be lower if x values very small (near 0) or higher if x values are very large.

Part b)

Still not enough information to tell. Like training RSS, test RSS varies depending on the x values, with small x values resulting in lower test RSS and vice versa.

Part c)

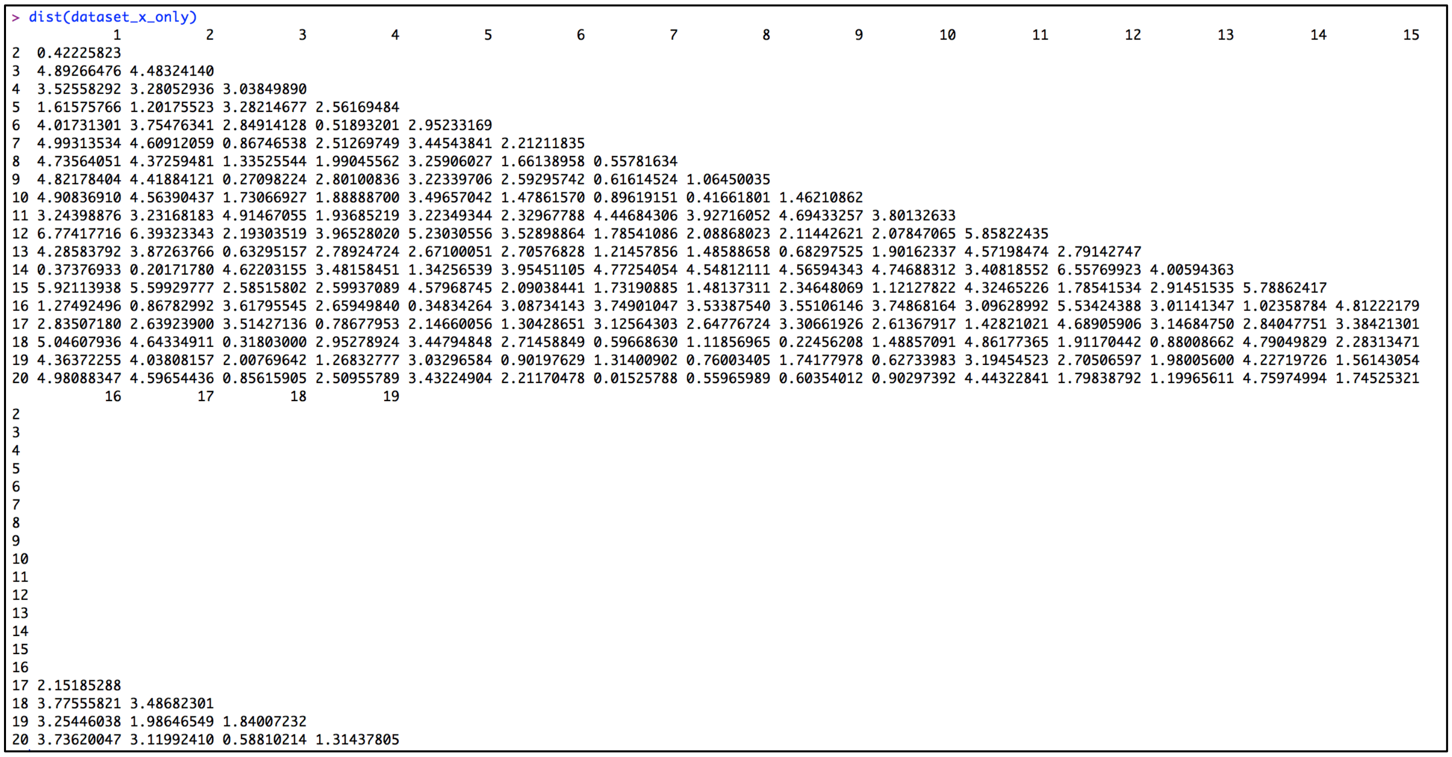
There is not enough information to tell. Depending on whether the X and Y data exhibits a relationship that is somewhat linear or cubic in nature, one would yield a lower training RSS than the other.

Part d)

Still not enough information to tell. Again, depending on whether the X and Y data exhibits a relationship that is somewhat linear or cubic in nature, one would yield a lower test RSS than the other.

Problem #3

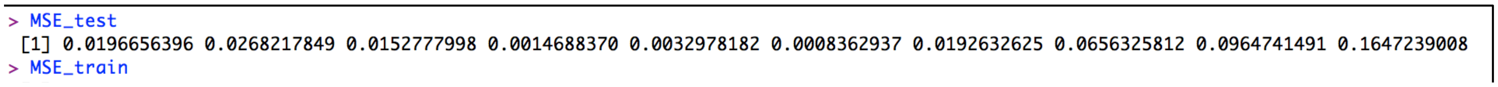
Part a)



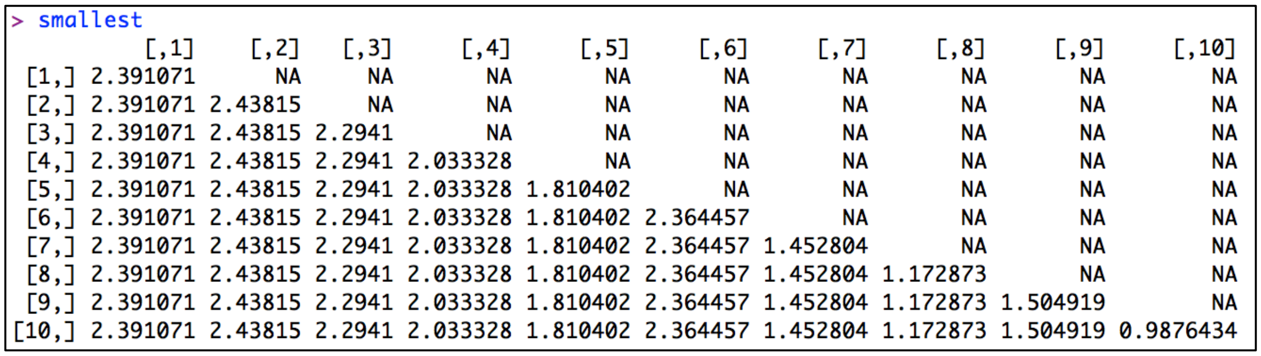
```
> dist(dataset_x_only)
            1          2          3          4          5          6          7          8          9         10         11         12         13         14         15
2  0.42225823
3  4.89266476 4.48324140
4  3.52558292 3.28052936 3.03849890
5  1.61575766 1.20175523 3.28214677 2.56169484
6  4.01731301 3.75476341 2.84914128 0.51893201 2.95233169
7  4.99313534 4.60912059 0.86746538 2.51269749 3.44543841 2.21211835
8  4.73564051 4.37259481 1.33525544 1.99045562 3.25906027 1.66138958 0.55781634
9  4.82178404 4.41884121 0.27098224 2.80100836 3.22339706 2.59295742 0.61614524 1.06450035
10 4.90836910 4.56390437 1.73066927 1.88888700 3.49657042 1.47861570 0.89619151 0.41661801 1.46210862
11 3.24398876 3.23168183 4.91467055 1.93685219 3.22349344 2.32967788 4.44684306 3.92716052 4.69433257 3.80132633
12 6.77417716 6.39323343 2.19303519 3.96528020 5.23030556 3.52898864 1.78541086 2.08868023 2.11442621 2.07847065 5.85822435
13 4.28583792 3.87263766 0.63295157 2.78924724 2.67100051 2.70576828 1.21457856 1.48588658 0.68297525 1.90162337 4.57198474 2.79142747
14 0.37376933 0.20171780 4.62203155 3.48158451 1.34256539 3.95451105 4.77254054 4.54812111 4.56594343 4.74688312 3.40818552 6.55769923 4.00594363
15 5.92113938 5.59929777 2.58515802 2.59937089 4.57968745 2.09038441 1.73190885 1.48137311 2.34648069 1.12127822 4.32465226 1.78541534 2.91451535 5.78862417
16 1.27492496 0.86782992 3.61795545 2.65949840 0.34834264 3.08734143 3.74901047 3.53387540 3.55106146 3.74868164 3.09628992 5.53424388 3.01141347 1.02358784 4.81222179
17 2.83507180 2.63923900 3.51427136 0.78677953 2.14660056 1.30428651 3.12564303 2.64776724 3.30661926 2.61367917 1.42821021 4.68905906 3.14684750 2.84047751 3.38421301
18 5.04607936 4.64334911 0.31803000 2.95278924 3.44794848 2.71458849 0.59668630 1.11856965 0.22456208 1.48857091 4.86177365 1.91170442 0.88008662 4.79049829 2.28313471
19 4.36372255 4.03808157 2.00769642 1.26832777 3.03296584 0.90197629 1.31400902 0.76003405 1.74177978 0.62733983 3.19454523 2.70506597 1.98005600 4.22719726 1.56143054
20 4.98088347 4.59654436 0.85615905 2.50955789 3.43224904 2.21170478 0.01525788 0.55965989 0.60354012 0.90297392 4.44322841 1.79838792 1.19965611 4.75974994 1.74525321
           16         17         18         19
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17 2.15185288
18 3.77555821 3.48682301
19 3.25446038 1.98646549 1.84007232
20 3.73620047 3.11992410 0.58810214 1.31437805
```

We created a matrix with only the x1 and x2 columns before using the dist() function to generate the distances.

Part b)



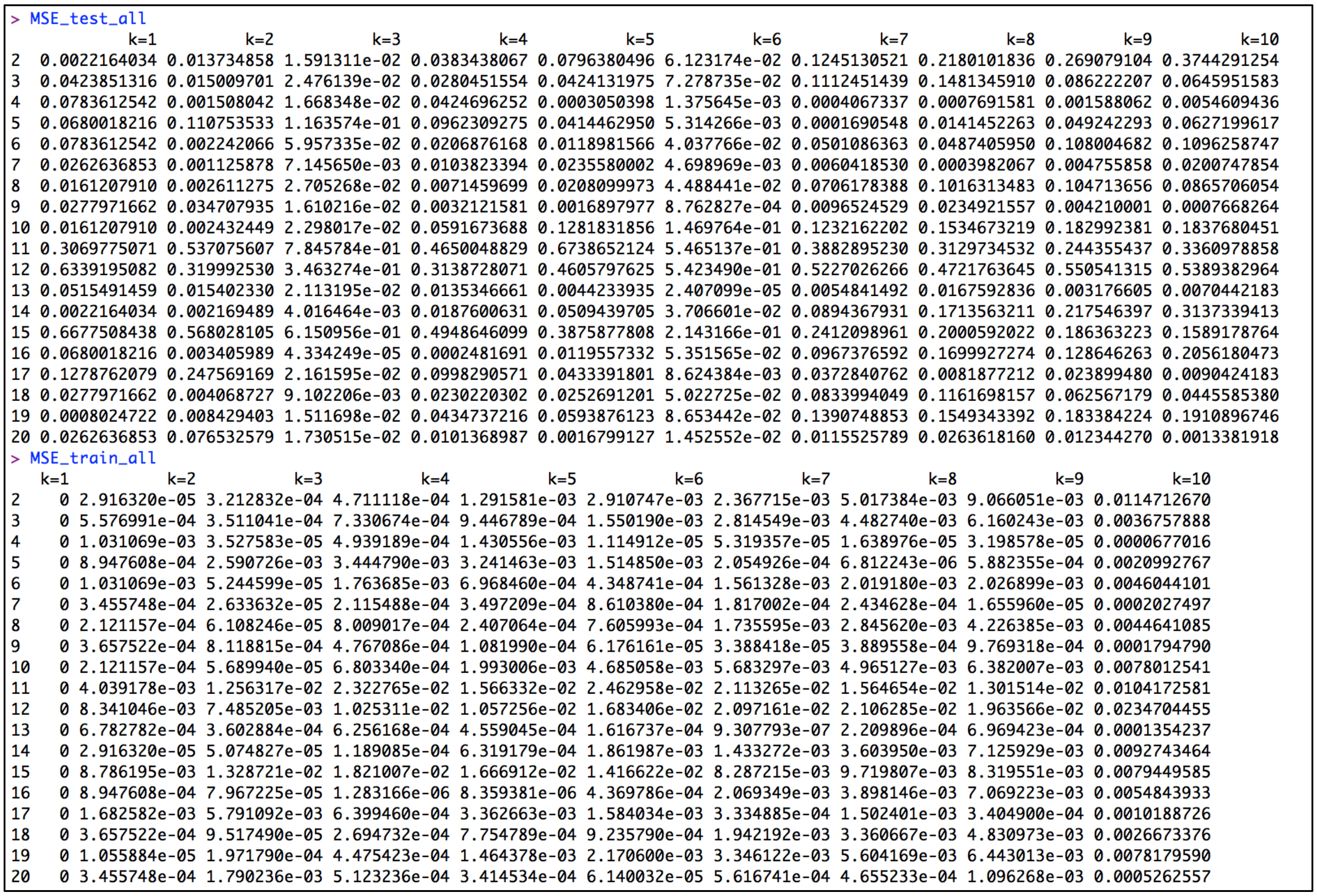
```
> MSE_test
 [1] 0.0196656396 0.0268217849 0.0152777998 0.0014688370 0.0032978182 0.0008362937 0.0192632625 0.0656325812 0.0964741491 0.1647239008
> MSE_train
```



```
> smallest
         [,1]    [,2]   [,3]     [,4]     [,5]     [,6]     [,7]     [,8]     [,9]     [,10]
 [1,] 2.391071      NA     NA       NA       NA       NA       NA       NA       NA        NA
 [2,] 2.391071 2.43815     NA       NA       NA       NA       NA       NA       NA        NA
 [3,] 2.391071 2.43815 2.2941       NA       NA       NA       NA       NA       NA        NA
 [4,] 2.391071 2.43815 2.2941 2.033328       NA       NA       NA       NA       NA        NA
 [5,] 2.391071 2.43815 2.2941 2.033328 1.810402       NA       NA       NA       NA        NA
 [6,] 2.391071 2.43815 2.2941 2.033328 1.810402 2.364457       NA       NA       NA        NA
 [7,] 2.391071 2.43815 2.2941 2.033328 1.810402 2.364457 1.452804       NA       NA        NA
 [8,] 2.391071 2.43815 2.2941 2.033328 1.810402 2.364457 1.452804 1.172873       NA        NA
 [9,] 2.391071 2.43815 2.2941 2.033328 1.810402 2.364457 1.452804 1.172873 1.504919        NA
[10,] 2.391071 2.43815 2.2941 2.033328 1.810402 2.364457 1.452804 1.172873 1.504919 0.9876434
```

Our training and testing MSEs are show above, with the vector entry positions corresponding to values of k (eg: 0.0196 is the MSE_test value when k=1). Our matrix "smallest" kept track of the y values being used to predict the y value of the first data point, allowing us to double check that our algorithm is working correctly. Thus, the average of each row represents y-hat for a certain k. The testing MSE was found by (y1 – yhat)^2, while the training MSE was found by [(y1 – yhat)^2]/19. As expected, our training MSE is 0 when k=1, since we include the data point itself when calculating training MSE.

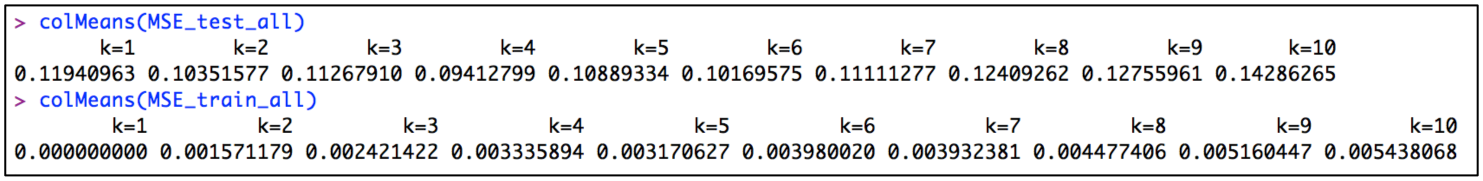
Part c)



```
> MSE_test_all
           k=1         k=2          k=3          k=4          k=5          k=6          k=7          k=8         k=9         k=10
2  0.0022164034 0.013734858 1.591311e-02 0.0383438067 0.0796380496 6.123174e-02 0.1245130521 0.2180101836 0.269079104 0.3744291254
3  0.0423851316 0.015009701 2.476139e-02 0.0280451554 0.0424131975 7.278735e-02 0.1112451439 0.1481345910 0.086222207 0.0645951583
4  0.0783612542 0.001508042 1.668348e-02 0.0424696252 0.0003050398 1.375645e-03 0.0004067337 0.0007691581 0.001588062 0.0054609436
5  0.0680018216 0.110753533 1.163574e-01 0.0962309275 0.0414462950 5.314266e-03 0.0001690548 0.0141452263 0.049242293 0.0627199617
6  0.0783612542 0.002242066 5.957335e-02 0.0206876168 0.0118981566 4.037766e-02 0.0501086363 0.0487405950 0.108004682 0.1096258747
7  0.0262636853 0.001125878 7.145650e-03 0.0103823394 0.0235580002 4.698969e-03 0.0060418530 0.0003982067 0.004755858 0.0200747854
8  0.0161207910 0.002611275 2.705268e-02 0.0071459699 0.0208099973 4.488441e-02 0.0706178388 0.1016313483 0.104713656 0.0865706054
9  0.0277971662 0.034707935 1.610216e-02 0.0032121581 0.0016897977 8.762827e-04 0.0096524529 0.0234921557 0.004210001 0.0007668264
10 0.0161207910 0.002432449 2.298017e-02 0.0591673688 0.1281831856 1.469764e-01 0.1232162202 0.1534673219 0.182992381 0.1837680451
11 0.3069775071 0.537075607 7.845784e-01 0.4650048829 0.6738652124 5.465137e-01 0.3882895230 0.3129734532 0.244355437 0.3360978858
12 0.6339195082 0.319992530 3.463274e-01 0.3138728071 0.4605797625 5.423490e-01 0.5227026266 0.4721763645 0.550541315 0.5389382964
13 0.0515491459 0.015402330 2.113195e-02 0.0135346661 0.0044233935 2.407099e-05 0.0054841492 0.0167592836 0.003176605 0.0070442183
14 0.0022164034 0.002169489 4.016464e-03 0.0187600631 0.0509439705 3.706601e-02 0.0894367931 0.1713563211 0.217546397 0.3137339413
15 0.6677508438 0.568028105 6.150956e-01 0.4948646099 0.3875877808 2.143166e-01 0.2412098961 0.2000592022 0.186363223 0.1589178764
16 0.0680018216 0.003405989 4.334249e-05 0.0002481691 0.0119557332 5.351565e-02 0.0967376592 0.1699927274 0.128646263 0.2056180473
17 0.1278762079 0.247569169 2.161595e-02 0.0998290571 0.0433391801 8.624384e-03 0.0372840762 0.0081877212 0.023899480 0.0090424183
18 0.0277971662 0.004068727 9.102206e-03 0.0230220302 0.0252691201 5.022725e-02 0.0833994049 0.1161698157 0.062567179 0.0445585380
19 0.0008024722 0.008429403 1.511698e-02 0.0434737216 0.0593876123 8.653442e-02 0.1390748853 0.1549343392 0.183384224 0.1910896746
20 0.0262636853 0.076532579 1.730515e-02 0.0101368987 0.0016799127 1.452552e-02 0.0115525789 0.0263618160 0.012344270 0.0013381918
> MSE_train_all
   k=1          k=2          k=3          k=4          k=5          k=6          k=7          k=8          k=9         k=10
2    0 2.916320e-05 3.212832e-04 4.711118e-04 1.291581e-03 2.910747e-03 2.367715e-03 5.017384e-03 9.066051e-03 0.0114712670
3    0 5.576991e-04 3.511041e-04 7.330674e-04 9.446789e-04 1.550190e-03 2.814549e-03 4.482740e-03 6.160243e-03 0.0036757888
4    0 1.031069e-03 3.527583e-05 4.939189e-04 1.430556e-03 1.114912e-05 5.319357e-05 1.638976e-05 3.198578e-05 0.0000677016
5    0 8.947608e-04 2.590726e-03 3.444790e-03 3.241463e-03 1.514850e-03 2.054926e-04 6.812243e-06 5.882355e-04 0.0020992767
6    0 1.031069e-03 5.244599e-05 1.763685e-03 6.968460e-04 4.348741e-04 1.561328e-03 2.019180e-03 2.026899e-03 0.0046044101
7    0 3.455748e-04 2.633632e-05 2.115488e-04 3.497209e-04 8.610380e-04 1.817002e-04 2.434628e-04 1.655960e-05 0.0002027497
8    0 2.121157e-04 6.108246e-05 8.009017e-04 2.407064e-04 7.605993e-04 1.735595e-03 2.845620e-03 4.226385e-03 0.0044641085
9    0 3.657522e-04 8.118815e-04 4.767086e-04 1.081990e-04 6.176161e-05 3.388418e-05 3.889558e-04 9.769318e-04 0.0001794790
10   0 2.121157e-04 5.689940e-05 6.803340e-04 1.993006e-03 4.685058e-03 5.683297e-03 4.965127e-03 6.382007e-03 0.0078012541
11   0 4.039178e-03 1.256317e-02 2.322765e-02 1.566332e-02 2.462958e-02 2.113265e-02 1.564654e-02 1.301514e-02 0.0104172581
12   0 8.341046e-03 7.485205e-03 1.025311e-02 1.057256e-02 1.683406e-02 2.097161e-02 2.106285e-02 1.963566e-02 0.0234704455
13   0 6.782782e-04 3.602884e-04 6.256168e-04 4.559045e-04 1.616737e-04 9.307793e-07 2.209896e-04 6.969423e-04 0.0001354237
14   0 2.916320e-05 5.074827e-05 1.189085e-04 6.319179e-04 1.861987e-03 1.433272e-03 3.603950e-03 7.125929e-03 0.0092743464
15   0 8.786195e-03 1.328721e-02 1.821007e-02 1.666912e-02 1.416622e-02 8.287215e-03 9.719807e-03 8.319551e-03 0.0079449585
16   0 8.947608e-04 7.967225e-05 1.283166e-06 8.359381e-06 4.369786e-04 2.069349e-03 3.898146e-03 7.069223e-03 0.0054843933
17   0 1.682582e-03 5.791092e-03 6.399460e-04 3.362663e-03 1.584034e-03 3.334885e-04 1.502401e-03 3.404900e-04 0.0010188726
18   0 3.657522e-04 9.517490e-05 2.694732e-04 7.754789e-04 9.235790e-04 1.942192e-03 3.360667e-03 4.830973e-03 0.0026673376
19   0 1.055884e-05 1.971790e-04 4.475423e-04 1.464378e-03 2.170600e-03 3.346122e-03 5.604169e-03 6.443013e-03 0.0078179590
20   0 3.455748e-04 1.790236e-03 5.123236e-04 3.414534e-04 6.140032e-05 5.616741e-04 4.655233e-04 1.096268e-03 0.0005262557
```

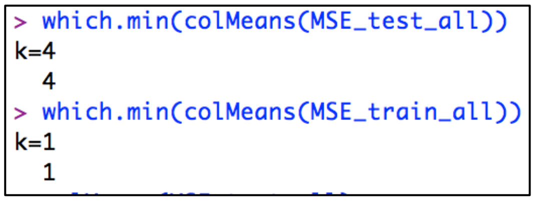
We generated our MSE test and MSE train by using nested for loops. The inner for loop controlled the k value, while the outer for loop determined the "ith" data point to be used as a testing set. After the MSEs for all 10 k values of each "ith" data point were calculated, we used rbind to append the MSEs to their respective matrix, thus generating two matrices which show training and testing MSEs for a given "ith" data point (row) and a given value of k (column). As expected, our training MSE is 0 when k=1.

Part d)



```
> colMeans(MSE_test_all)
       k=1        k=2        k=3        k=4        k=5        k=6        k=7        k=8        k=9       k=10 
0.11940963 0.10351577 0.11267910 0.09412799 0.10889334 0.10169575 0.11111277 0.12409262 0.12755961 0.14286265 
> colMeans(MSE_train_all)
        k=1         k=2         k=3         k=4         k=5         k=6         k=7         k=8         k=9        k=10 
0.000000000 0.001571179 0.002421422 0.003335894 0.003170627 0.003980020 0.003932381 0.004477406 0.005160447 0.005438068 
```

To simplify using our results in part c in order to determine the optimal value for k, we use colMeans to find the average testing and training MSEs by k value. Then, using the which.min function, we find that k=4 results in the smallest average testing MSE while k=1 results in the smallest average training MSE.

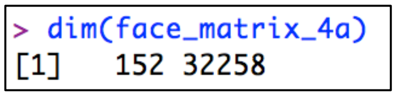


```
> which.min(colMeans(MSE_test_all))
k=4 
  4 
> which.min(colMeans(MSE_train_all))
k=1 
  1 
```

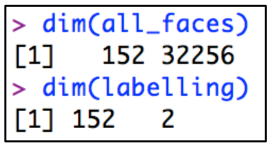
The optimal k value is k=4 since it minimizes test MSE, meaning that the predictions it generates of the y value of the testing data are on average the most accurate (lowest mean squared error) of all k values between 1 and 10. Although training MSE and testing MSE indicate different optimal k values, we should choose to base our choice of optimal k on the testing MSE since we ultimately care most about how well the model performs on our test set. Furthermore, k=1 is the optimal k value given by training MSE because all training MSEs are 0 for k=1. This is because training MSE includes the ith data point itself (unlike test MSE which excludes itself), and the data point is naturally the same as itself, thus resulting in a 0 value for training MSE.

Question 4

Part a)

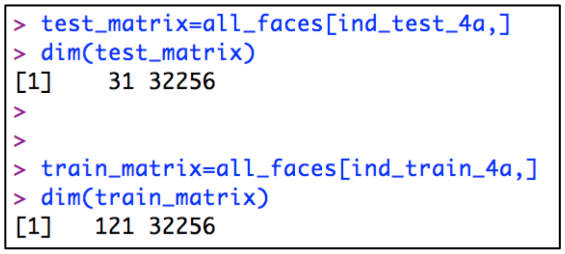


```
> dim(face_matrix_4a)
[1]   152 32258
```



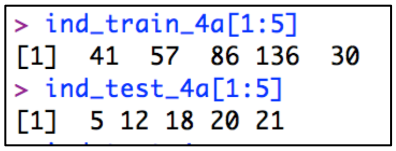
```
> dim(all_faces)
[1]   152 32256
> dim(labelling)
[1] 152   2
```

We generate a 152 x 32258 matrix of the vectors of the faces with the 4 indicated views. We can divide this matrix into 2 sets: one with the vectors of the faces, and the other with the subject numbers and views.



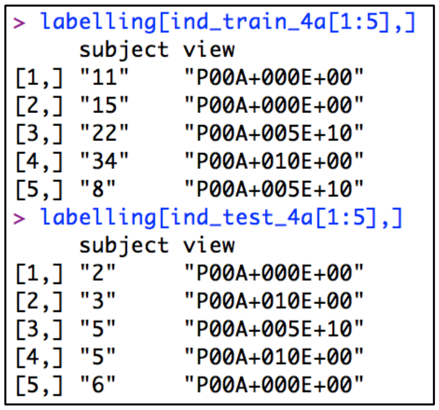
```
> test_matrix=all_faces[ind_test_4a,]
> dim(test_matrix)
[1]    31 32256
>
>
> train_matrix=all_faces[ind_train_4a,]
> dim(train_matrix)
[1]   121 32256
```

After dividing the data into training and testing sets, we place these sets into their own matrices, finding that the test matrix has 31 rows while the training matrix has 121 rows.



```
> ind_train_4a[1:5]
[1]  41  57  86 136  30
> ind_test_4a[1:5]
[1]  5 12 18 20 21
```

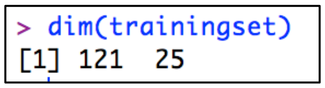
Using our training and testing index vectors, we see that the first 5 files in the training set reference rows 41, 57, 86, 136, and 30 of the original 152-row dataset, while the first 5 files of the test set reference rows 5, 12, 18,20, and 21 of the original 152-row dataset.



```
> labelling[ind_train_4a[1:5],]
     subject view
[1,] "11"    "P00A+000E+00"
[2,] "15"    "P00A+000E+00"
[3,] "22"    "P00A+005E+10"
[4,] "34"    "P00A+010E+00"
[5,] "8"     "P00A+005E+10"
> labelling[ind_test_4a[1:5],]
     subject view
[1,] "2"     "P00A+000E+00"
[2,] "3"     "P00A+010E+00"
[3,] "5"     "P00A+005E+10"
[4,] "5"     "P00A+010E+00"
[5,] "6"     "P00A+000E+00"
```

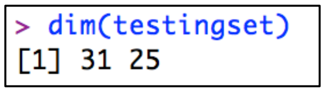
We then use our “labelling” matrix, which contains the subject and view of each of the 152 rows of our dataset, to identify the first 5 files in the training and testing sets, as seen in the above.

Part b)



```
> dim(trainingset)
[1] 121  25
```

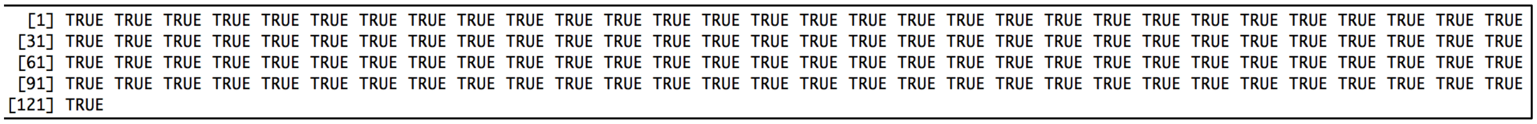
We perform PCA on our training set and use the first 25 scores to represent the data, generating a training set matrix with dimensions 121 x 25.



```
> dim(testingset)
[1] 31 25
```

We then project our testing data into the score space so that it is also represented by the first 25 scores, giving us a test set matrix with dimensions 31 x 25.

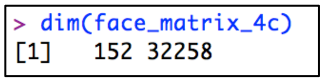
We then use 1NN classification to identify the subject of each testing observation in the score space. We use the formula k(trainingset, testingset, cl, k=1), where trainingset and testingset reference the matrices we generated above, while cl represents the training index vector we generated in part a.



```
  [1] TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE
 [31] TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE
 [61] TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE
 [91] TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE
[121] TRUE
```

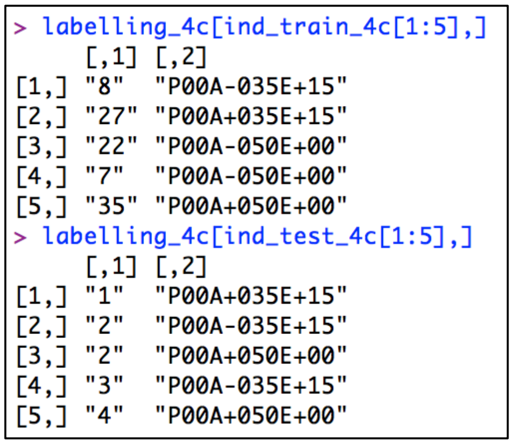
We check our Y_test_hat values against the subject numbers in our "cl" vector and find that all of our subjects are identified correctly (121 identified correctly, 0 identified incorrectly. This makes sense since in direct, frontal lighting (the views we loaded in part a), PCA is likely more accurate since it can better store information about and identify characteristics of faces (there is little noise to throw off PCA's predictions).

Part c)



```
> dim(face_matrix_4c)
[1]   152 32258
```

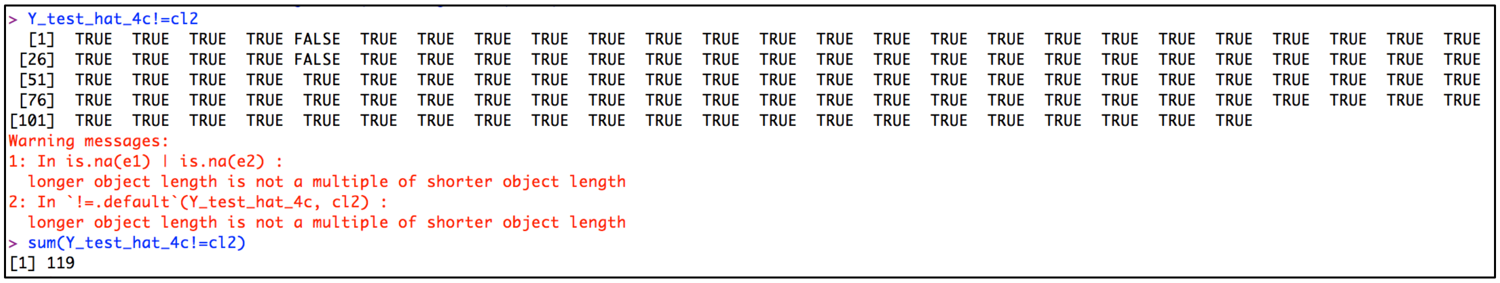
We load the 4 new views, again generating a 152 x 32258 matrix of the vectors of the faces.



```
> labelling_4c[ind_train_4c[1:5],]
     [,1] [,2]
[1,] "8"  "P00A-035E+15"
[2,] "27" "P00A+035E+15"
[3,] "22" "P00A-050E+00"
[4,] "7"  "P00A-050E+00"
[5,] "35" "P00A+050E+00"
> labelling_4c[ind_test_4c[1:5],]
     [,1] [,2]
[1,] "1"  "P00A+035E+15"
[2,] "2"  "P00A-035E+15"
[3,] "2"  "P00A+050E+00"
[4,] "3"  "P00A-035E+15"
[5,] "4"  "P00A+050E+00"
```

We again use our "labelling" matrix to identify the first 5 files in the training and testing sets.

We again perform PCA on our training set and use the first 25 scores to represent the data, while also projecting the test set into the score space. This again results in a training set of dimension 121 x 25 and testing set of dimension 31 x 25.

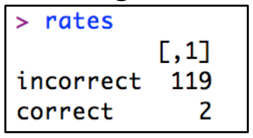


```
> Y_test_hat_4c!=cl2
  [1]  TRUE  TRUE  TRUE  TRUE FALSE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE
 [26]  TRUE  TRUE  TRUE  TRUE FALSE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE
 [51]  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE
 [76]  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE
[101]  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE
Warning messages:
1: In is.na(e1) | is.na(e2) :
  longer object length is not a multiple of shorter object length
2: In `!=.default`(Y_test_hat_4c, cl2) :
  longer object length is not a multiple of shorter object length
> sum(Y_test_hat_4c!=cl2)
[1] 119
```

Though our 1NN classification seems to misfire and suggest that 119 of the 121 subjects are misidentified, we know that in reality, approximately 27 of the subjects are misidentified, while the remainder (91 subjects) are identified correctly.

Part d)

We repeat our code from part c 10 times and generate 10 identical instances of the following correct and incorrect identification rates.



```
> rates
          [,1]
incorrect  119
correct      2
```

Though our code likely contains errors, these results tell us that PCA/kNN consistently misidentifies a significant number of subjects in these 4 lighting conditions.

Part e)

There are drastic differences the testing error rates in part b and part c. In part b, we identified all subjects correctly, while in part c, we failed to identify a significant number of subjects correctly. This tells us that PCA works better in certain lighting conditions than others. In part b, the lighting was straight on, generating little noise on the facial features and thus resulting in a very high identification accuracy rate for PCA/kNN. Meanwhile, the side lighting in part c generated a lot of shadows, noise, and obscurations of subjects' facial features, thus greatly hindering PCA/kNN's ability to match facial features and consequently committing a significant number of identification errors. This tells us that PCA is likely sensitive to – and thus probably easily thrown off by – noise in data and facial features.

Part f)

If we used uncropped photos, the testing error rate of PCA/kNN is likely to be very high (ie: a high number of resulting misidentifications). This is because noise created by hair, neck, clothing, and other non-facial features of a subject are likely to throw off PCA/kNN's ability to match faces. PCA/kNN will instead try to match images based on their overall content, and this would probably render facial matching accuracy very low. This would likely be even further worsened in lighting conditions that do not illuminate a subject straight-on.

CODE

```r
#############################
# Kevin Gong
# STAT W4240
# Homework 3 , Problem 3
# 3/5/14
#
# The following code loads the eigenfaces data and
# performs a set of simple loading and plotting functions
#############################

#################
# Setup
#################

# make sure R is in the proper working directory
# note that this will be a different path for every machine
setwd("~/Dropbox/SIPA/Data Mining/HW3")

# first include the relevant libraries
# note that a loading error might mean that you have to
# install the package into your R distribution.

#################
# Problem 3a
#################

# load the data and use dist() to get a distance matrix


#----- START YOUR CODE BLOCK HERE -----#

#import the data
dataset <- read.csv(file="hw03_q3.csv")
head(dataset)

#set aside x1 and x2 in their own matrix
dataset_x_only <- dataset[,1:2]
head(dataset_x_only)

#compute the distances between x1 and x2 using dist()
distances <-as.matrix(dist(dataset_x_only))
distances
#----- END YOUR CODE BLOCK HERE -----#

#################
# Problem 3b
#################

#----- START YOUR CODE BLOCK HERE -----#
dataset_y_only <- dataset[,3]
head(dataset_y_only)

#separate data into testing and training sets
test <- dataset_x_only[1,]
test
train <- dataset_x_only[2:20,]
train

#allocate empty placeholder vectors and matrices for our for loop
smallest=matrix(nrow=10,ncol=10)
MSE_test=vector()
```

```
MSE_train=vector()



#First we calculate the training MSE. We make sure to include the point itself when doing so,
 hence [1:k]
for(k in 1:10){
 abba <- order(distances[,1],decreasing=F)[1:k]
 smallest[k,(1:k)] = dataset[abba,3]
 yhat=sum(dataset[abba,3])/k
 MSE_train[k]=((dataset[1,3]-yhat)^2)/19
 }

#Next we calculate the testing MSE. We make sure to avoid including the testing point itself when
 doing so, hence [2:(k+1)]
for(k in 1:10){
 abba <- order(distances[,1],decreasing=F)[2:(k+1)]
 smallest[k,(1:k)] = dataset[abba,3]
 yhat=sum(dataset[abba,3])/k
 MSE_test[k]=(dataset[1,3]-yhat)^2
 }

#examine the results. The "smallest" matrix allows us to check the y values used to approximate
 our test y for each k value.
MSE_test
MSE_train
smallest



#----- END YOUR CODE BLOCK HERE -----#

#################
# Problem 3c
#################

#----- START YOUR CODE BLOCK HERE -----#



#allocate empty placeholder vectors and matrices for our for loops
MSE_test=vector()
MSE_train=vector()

smallest=matrix(nrow=190,ncol=10)
MSE_test_all=NULL
MSE_train_all=NULL



#We calculate the training MSE. We make sure to include the point itself when doing so, hence
 [1:k]
for(i in 2:20){
 MSE_train=vector()
for(k in 1:10){
 abba <- order(distances[,i],decreasing=F)[1:k]
 smallest[k,(1:k)] = dataset[abba,3]
 yhat=sum(dataset[abba,3])/k
 MSE_train[k]=((dataset[i,3]-yhat)^2)/19
 }
 MSE_train_all=rbind(MSE_train_all,MSE_train)
}
```

```r
#We calculate the testing MSE. We make sure to avoid including the testing point itself when
 doing so, hence [2:(k+1)]
for(i in 2:20){
 MSE_test=vector()
 MSE_train=vector()

for(k in 1:10){
 abba <- order(distances[,i],decreasing=F)[2:(k+1)]
 smallest[k,(1:k)] = dataset[abba,3]
 yhat=sum(dataset[abba,3])/k
 MSE_test[k]=(dataset[i,3]-yhat)^2
 }
 MSE_test_all=rbind(MSE_test_all,MSE_test)
}

#examine the resulting MSEs
MSE_test_all
MSE_train_all



#remane the column names to the different values of k
colnames(MSE_test_all) <-c("k=1","k=2","k=3","k=4","k=5","k=6","k=7","k=8","k=9","k=10")
colnames(MSE_train_all) <-c("k=1","k=2","k=3","k=4","k=5","k=6","k=7","k=8","k=9","k=10")

#rename the row names to the ith data point
rownames(MSE_test_all)<-c(2:20)
rownames(MSE_train_all)<-c(2:20)

#examine renamed MSE matrices
MSE_test_all
MSE_train_all


#average the mean MSEs by k value and see which k value has the lowest average mean squared error
colMeans(MSE_test_all)
colMeans(MSE_train_all)
which.min(colMeans(MSE_test_all))
which.min(colMeans(MSE_train_all))


#----- END YOUR CODE BLOCK HERE -----#

################
# End of Script
################









############################
# Kevin Gong
# STAT W4240
# Homework 3 , Problem 4
# 3/5/14
#
# The following code loads the eigenfaces data and
# performs a set of simple loading and plotting functions
```

```
#############################

#################
# Setup
#################

# make sure R is in the proper working directory
# note that this will be a different path for every machine
setwd("~/Dropbox/SIPA/Data Mining/HW3")

# first include the relevant libraries
# note that a loading error might mean that you have to
# install the package into your R distribution.
library(pixmap)
library(class)

#################
# Problem 4a
#################

views_4a = c('P00A+000E+00', 'P00A+005E+10', 'P00A+005E-10', 'P00A+010E+00' )

# load the data and save it as a matrix with the name face_matrix_4a

#----- START YOUR CODE BLOCK HERE -----#


# the list of pictures
dir_list_1 = dir(path="CroppedYale/",all.files=FALSE)
views_4a = c('P00A+000E+00', 'P00A+005E+10', 'P00A+005E-10', 'P00A+010E+00' )

# preallocate an empty list
this_face_row = vector()
# initialize an empty matrix of faces data
face_matrix_4a = vector()
pic_list = c(
 01,02,03,04,05,06,07,08,09,10,11,12,13,15,16,17,18,19,20,21,22,23,24,25,26,27,28,29,30,31,32,33,
 34,35,36,37,38,39)


#generating a matrix of photos of the 4 desired views

for (i in 1:38) {

placeholder_matrix <- NULL

for (j in 1:4) {
 filename= sprintf("CroppedYale/%s/%s_%s.pgm",dir_list_1[i],dir_list_1[i],views_4a[j])
print(filename)
face=read.pnm(file=filename)
pic_data <-getChannels(face)
pic_data_vec <- as.vector(pic_data)
placeholder_matrix=rbind(placeholder_matrix,c(pic_data_vec,i,views_4a[j]))
}
face_matrix_4a=rbind(face_matrix_4a,placeholder_matrix)
}

dim(face_matrix_4a)


all_faces <- face_matrix_4a[,1:32256]
labelling <- face_matrix_4a[,32257:32258]
```

```
dim(all_faces)
dim(labelling)
colnames(labelling) <-c("subject","view")
labelling



#----- END YOUR CODE BLOCK HERE -----#

# Get the size of the matrix for use later
fm_4a_size = dim(face_matrix_4a)
# Use 4/5 of the data for training, 1/5 for testing
ntrain_4a = floor(fm_4a_size[1]*4/5) # Number of training obs
ntest_4a = fm_4a_size[1]-ntrain_4a # Number of testing obs
set.seed(1) # Set pseudo-random numbers so everyone gets the same output
ind_train_4a = sample(1:fm_4a_size[1],ntrain_4a) # Training indices
ind_test_4a = c(1:fm_4a_size[1])[-ind_train_4a] # Testing indices








#----- START YOUR CODE BLOCK HERE -----#
ind_train_4a
ind_test_4a






test_matrix=all_faces[ind_test_4a,]
dim(test_matrix)


train_matrix=all_faces[ind_train_4a,]
dim(train_matrix)



test_id = labelling[ind_test_4a,]
dim(test_id)


train_id = labelling[ind_train_4a,]
dim(train_id)

#find the rows referenced by the first 5 files of the training and testing sets
ind_train_4a[1:5]
ind_test_4a[1:5]

#find the first 5 files of the training and testing sets
labelling[ind_train_4a[1:5],]
labelling[ind_test_4a[1:5],]







#----- END YOUR CODE BLOCK HERE -----#
```

```
#################
# Problem 4b
#################

#----- START YOUR CODE BLOCK HERE -----#

#training data first

mean_face <- colMeans(train_matrix)
means_mat=as.matrix(mean_face)

#reshape this matrix into 192x168
means_mat_ref=matrix(means_mat,192,168)
means_face=pixmapGrey(means_mat_ref)
plot(means_face)



pic_mat_size = dim(train_matrix)
pic_mat_centered = mat.or.vec(pic_mat_size[1],pic_mat_size[2])

#subtract mean face from matrix
for (i in 1:pic_mat_size[1]){
 pic_mat_centered[i,] = train_matrix[i,] - mean_face
}

dim(pic_mat_centered)
pca_results <- prcomp(pic_mat_centered)

dim(pca_results)

dim(as.matrix(loading_temp))
dim(pic_mat_centered)



num_comp_mod = length(pca_results$x[,1])
scores_query = mat.or.vec(num_comp_mod,1)
trainingset=NULL
for (i in 1:121){
 placeholder = NULL

for(j in 1:25){
 loading_temp = pca_results$rotation[,j]
 scores_query = pic_mat_centered[i,] %*% loading_temp
 placeholder = cbind(placeholder,scores_query)
 }
 trainingset=rbind(trainingset,placeholder)
}

dim(trainingset)









mean_face_test <- colMeans(test_matrix)
means_mat_test=as.matrix(mean_face_test)
```

```
#reshape this matrix into 192x168
means_mat_ref_test=matrix(means_mat_test,192,168)
means_face_test=pixmapGrey(means_mat_ref_test)
plot(means_face_test)



pic_mat_size_test = dim(test_matrix)
pic_mat_centered_test = mat.or.vec(pic_mat_size_test[1],pic_mat_size_test[2])

# I am using a loop, but one could use apply()
for (i in 1:pic_mat_size_test[1]){
 pic_mat_centered_test[i,] = test_matrix[i,] - mean_face
}

dim(pic_mat_centered_test)
pca_results_test <- prcomp(pic_mat_centered_test)

dim(as.matrix(pca_results_test))

dim(as.matrix(loading_temp_test))
dim(pic_mat_centered_test)



num_comp_mod_test = length(pca_results_test$x[,1])
scores_query_test = mat.or.vec(num_comp_mod_test,1)
testingset=NULL
for (i in 1:31){
 placeholder_test = NULL

for(j in 1:25){
 loading_temp_test = pca_results_test$rotation[,j] #note this is the same as training (I think
 it's supposed to be like this)
 scores_query_test = pic_mat_centered_test[i,] %*% loading_temp_test
 placeholder_test = cbind(placeholder_test,scores_query_test)
 }
 testingset=rbind(testingset,placeholder_test)
}

dim(testingset)

#this works?
cl <- row.names(data.frame(trainingset))


cl2 <-factor(ind_train_4a)

length(indicator)
Y_test_hat <- knn(trainingset, testingset, cl2, k=1)
Y_test_hat!=cl2

sum(Y_test_hat!=cl)









#----- END YOUR CODE BLOCK HERE -----#
```

```r
#################
# Problem 4c
#################

# Use different lighting conditions

views_4c = c('P00A-035E+15', 'P00A-050E+00', 'P00A+035E+15', 'P00A+050E+00')

# load your data and save the images as face_matrix_4c

#----- START YOUR CODE BLOCK HERE -----#




# the list of pictures
dir_list_1 = dir(path="CroppedYale/",all.files=FALSE)
views_4c = c('P00A-035E+15', 'P00A-050E+00', 'P00A+035E+15', 'P00A+050E+00')

# preallocate an empty list
this_face_row = vector()
# initialize an empty matrix of faces data
face_matrix_4c = vector()
pic_list = c(
 01,02,03,04,05,06,07,08,09,10,11,12,13,15,16,17,18,19,20,21,22,23,24,25,26,27,28,29,30,31,32,33,
 34,35,36,37,38,39)


#generating a matrix of photos of the 4 desired views

for (i in 1:38) {

placeholder_matrix <- NULL

for (j in 1:4) {
 filename= sprintf("CroppedYale/%s/%s_%s.pgm",dir_list_1[i],dir_list_1[i],views_4c[j])
print(filename)
face=read.pnm(file=filename)
pic_data <-getChannels(face)
pic_data_vec <- as.vector(pic_data)
placeholder_matrix=rbind(placeholder_matrix,c(pic_data_vec,i,views_4c[j]))
}
face_matrix_4c=rbind(face_matrix_4c,placeholder_matrix)
}

dim(face_matrix_4c)


all_faces_4c <- face_matrix_4c[,1:32256]
labelling_4c <- face_matrix_4c[,32257:32258]




#----- END YOUR CODE BLOCK HERE -----#

fm_4c_size = dim(face_matrix_4c)
# Use 4/5 of the data for training, 1/5 for testing
ntrain_4c = floor(fm_4c_size[1]*4/5)
ntest_4c = fm_4c_size[1]-ntrain_4c
set.seed(2) # Set pseudo-random numbers
# You are resetting so that if you have used a random number in between the last use of sample(),
 you will still get the same output
```

```r
ind_train_4c = sample(1:fm_4c_size[1],ntrain_4c)
ind_test_4c = c(1:fm_4c_size[1])[-ind_train_4c]

#----- START YOUR CODE BLOCK HERE -----#



ind_train_4c
ind_test_4c

test_matrix_4c=all_faces_4c[ind_test_4c,]
dim(test_matrix_4c)


train_matrix_4c=all_faces_4c[ind_train_4c,]
dim(train_matrix_4c)



test_id_4c = labelling_4c[ind_test_4c,]
dim(test_id_4c)


train_id_4c = labelling_4c[ind_train_4c,]
dim(train_id_4c)


#find the rows referenced by the first 5 files of the training and testing sets
ind_train_4c[1:5]
ind_test_4c[1:5]

#find the first 5 files of the training and testing sets
labelling_4c[ind_train_4c[1:5],]
labelling_4c[ind_test_4c[1:5],]





#training data first

mean_face_4c <- colMeans(train_matrix_4c)
means_mat_4c=as.matrix(mean_face_4c)

#reshape this matrix into 192x168
means_mat_ref_4c=matrix(means_mat_4c,192,168)
means_face_4c=pixmapGrey(means_mat_ref_4c)
plot(means_face_4c)



pic_mat_size_4c = dim(train_matrix_4c)
pic_mat_centered_4c = mat.or.vec(pic_mat_size_4c[1],pic_mat_size_4c[2])

#subtract mean face from matrix
for (i in 1:pic_mat_size_4c[1]){
 pic_mat_centered_4c[i,] = train_matrix_4c[i,] - mean_face_4c
}

dim(pic_mat_centered_4c)
pca_results_4c <- prcomp(pic_mat_centered_4c)

dim(pca_results_4c)
dim(as.matrix(loading_temp_4c))
dim(pic_mat_centered_4c)



num_comp_mod_4c = length(pca_results_4c$x[,1])
scores_query_4c = mat.or.vec(num_comp_mod_4c,1)
trainingset_4c=NULL
for (i in 1:121){
 placeholder_4c = NULL

for(j in 1:25){
 loading_temp_4c = pca_results_4c$rotation[,j]
 scores_query_4c = pic_mat_centered_4c[i,] %*% loading_temp_4c
 placeholder_4c = cbind(placeholder_4c,scores_query_4c)
 }
 trainingset_4c=rbind(trainingset_4c,placeholder_4c)
}

dim(trainingset_4c)









mean_face_test_4c <- colMeans(test_matrix_4c)
means_mat_test_4c=as.matrix(mean_face_test_4c)

#reshape this matrix into 192x168
means_mat_ref_test_4c=matrix(means_mat_test_4c,192,168)
means_face_test_4c=pixmapGrey(means_mat_ref_test_4c)
plot(means_face_test_4c)



pic_mat_size_test_4c = dim(test_matrix_4c)
pic_mat_centered_test_4c = mat.or.vec(pic_mat_size_test_4c[1],pic_mat_size_test_4c[2])

# I am using a loop, but one could use apply()
for (i in 1:pic_mat_size_test_4c[1]){
 pic_mat_centered_test_4c[i,] = test_matrix_4c[i,] - mean_face_4c
}

dim(pic_mat_centered_test_4c)
pca_results_test_4c <- prcomp(pic_mat_centered_test_4c)

dim(as.matrix(pca_results_test_4c))

dim(as.matrix(loading_temp_test_4c))
dim(pic_mat_centered_test_4c)



num_comp_mod_test_4c = length(pca_results_test_4c$x[,1])
scores_query_test_4c = mat.or.vec(num_comp_mod_test_4c,1)
testingset_4c=NULL
for (i in 1:31){
 placeholder_test_4c = NULL
for(j in 1:25){
 loading_temp_test_4c = pca_results_test_4c$rotation[,j]
 scores_query_test_4c = pic_mat_centered_test_4c[i,] %*% loading_temp_test_4c
 placeholder_test_4c = cbind(placeholder_test_4c,scores_query_test_4c)
 }
 testingset_4c=rbind(testingset_4c,placeholder_test_4c)
}

dim(testingset_4c)





#this works?
cl <- row.names(data.frame(testingset_4c))


cl2 <- factor(ind_train_4c)



length(indicator)
Y_test_hat_4c <- knn(trainingset_4c, testingset_4c, cl, k=1)
Y_test_hat_4c!=cl

sum(Y_test_hat_4c!=cl2)




#----- END YOUR CODE BLOCK HERE -----#

#################
# Problem 4d
#################

#----- START YOUR CODE BLOCK HERE -----#



#REPEAT 10 TIMES


fm_4c_size = dim(face_matrix_4c)
# Use 4/5 of the data for training, 1/5 for testing
ntrain_4c = floor(fm_4c_size[1]*4/5)
ntest_4c = fm_4c_size[1]-ntrain_4c
set.seed(2) # Set pseudo-random numbers
# You are resetting so that if you have used a random number in between the last use of sample(),
 you will still get the same output
ind_train_4c = sample(1:fm_4c_size[1],ntrain_4c)
ind_test_4c = c(1:fm_4c_size[1])[-ind_train_4c]



test_matrix_4c=all_faces_4c[ind_test_4c,]

train_matrix_4c=all_faces_4c[ind_train_4c,]

test_id_4c = labelling_4c[ind_test_4c,]

train_id_4c = labelling_4c[ind_train_4c,]
```

```r
#training data first

mean_face_4c <- colMeans(train_matrix_4c)
means_mat_4c=as.matrix(mean_face_4c)

#reshape this matrix into 192x168
means_mat_ref_4c=matrix(means_mat_4c,192,168)


pic_mat_size_4c = dim(train_matrix_4c)
pic_mat_centered_4c = mat.or.vec(pic_mat_size_4c[1],pic_mat_size_4c[2])

#subtract mean face from matrix
for (i in 1:pic_mat_size_4c[1]){
 pic_mat_centered_4c[i,] = train_matrix_4c[i,] - mean_face_4c
}

pca_results_4c <- prcomp(pic_mat_centered_4c)


num_comp_mod_4c = length(pca_results_4c$x[,1])
scores_query_4c = mat.or.vec(num_comp_mod_4c,1)
trainingset_4c=NULL
for (i in 1:121){
 placeholder_4c = NULL

for(j in 1:25){
 loading_temp_4c = pca_results_4c$rotation[,j]
 scores_query_4c = pic_mat_centered_4c[i,] %*% loading_temp_4c
 placeholder_4c = cbind(placeholder_4c,scores_query_4c)
 }
 trainingset_4c=rbind(trainingset_4c,placeholder_4c)
}




mean_face_test_4c <- colMeans(test_matrix_4c)
means_mat_test_4c=as.matrix(mean_face_test_4c)

#reshape this matrix into 192x168
means_mat_ref_test_4c=matrix(means_mat_test_4c,192,168)




pic_mat_size_test_4c = dim(test_matrix_4c)
pic_mat_centered_test_4c = mat.or.vec(pic_mat_size_test_4c[1],pic_mat_size_test_4c[2])

for (i in 1:pic_mat_size_test_4c[1]){
 pic_mat_centered_test_4c[i,] = test_matrix_4c[i,] - mean_face_4c
}

pca_results_test_4c <- prcomp(pic_mat_centered_test_4c)


num_comp_mod_test_4c = length(pca_results_test_4c$x[,1])
```

```
scores_query_test_4c = mat.or.vec(num_comp_mod_test_4c,1)
testingset_4c=NULL
for (i in 1:31){
 placeholder_test_4c = NULL

for(j in 1:25){
 loading_temp_test_4c = pca_results_test_4c$rotation[,j]
 scores_query_test_4c = pic_mat_centered_test_4c[i,] %*% loading_temp_test_4c
 placeholder_test_4c = cbind(placeholder_test_4c,scores_query_test_4c)
 }
 testingset_4c=rbind(testingset_4c,placeholder_test_4c)
}



#this works?


cl2 <- factor(ind_train_4c)


Y_test_hat_4c <- knn(trainingset_4c, testingset_4c, cl2, k=1)
Y_test_hat_4c!=cl2
incorrect <- sum(Y_test_hat_4c!=cl2)
correct <- sum(Y_test_hat_4c==cl2)
rates <- matrix(c(incorrect, correct))
rownames(rates) <-c("incorrect","correct")
rates




#----- END YOUR CODE BLOCK HERE -----#

#################
# End of Script
#################
```