Problem 1

Problem 1

Read in the follow bitimage

```
library(bmp)
img = read.bmp('~/Downloads/image1.bmp')
rotate = function(x) t(apply(x, 2, rev))
img = rotate(img)
img = scale(img, center=TRUE, scale=FALSE)
```

Plot image in greyscale

```
gs = grey(seq(0, 1, length=256))
image(img, asp=1, col=gs)
```

Part A

Compute the principal components using prcomp and list objects in the function output: i.e. str function would be useful.

```
pca.img <- prcomp(img)
names(pca.img)
```

```
## [1] "sdev"     "rotation" "center"   "scale"    "x"
```

Part B

Recall that principal components were linear combination of data columns. Verify that this is true by multiplying data matrix (original bitmap image img or

a.k.a X) by loadings (pca.img$rotation object or a.k.a matrix of ij) and compare to computed principal components (pca.img$x object or a.k.a Z's)

```
prod <- img * pca.img$rotation
prin_comp <- pca.img$x

norm(prin_comp - (prod), type = "F")
```

```
## [1] 24025.71
```

Part C

Check that rotation of the prcomp output is indeed a rotation matrix, say Q, by verifying a crucial property of orthonormal rotation matrices:

```
Q <- pca.img$rotation
norm((t(Q) * Q) - diag(512))
```

```
## [1] 1.691255
```

Part D

Using this fact, reconstruct the image from 10 and 100 principal components and plot the reconstructed image.

```
# Stuff goes here
```

Part E

Plot proportion of variance explained as function of number of principal components and also cumulative proportional variance explained. The function summary returns helpful objects including PVE. Using this information, find out how many principal components are needed to explain 90% of the variance.

```
percent_var_expl <- pca.img$sdev^2 / sum(pca.img$sdev^2)
percent_var_expl
```

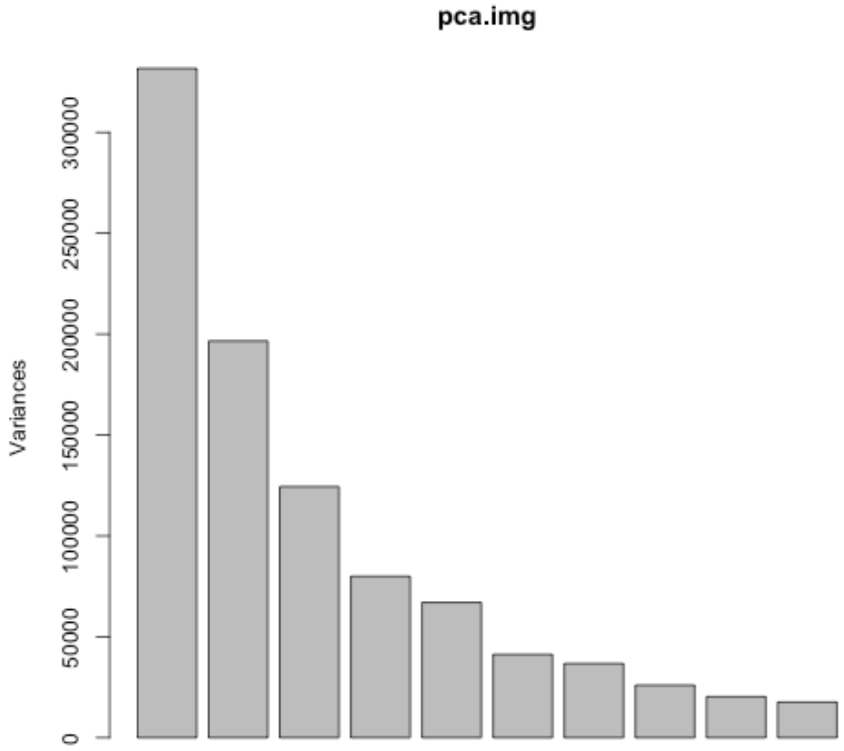
```
##  [1] 2.936665e-01 1.740617e-01 1.101491e-01 7.077572e-02 5.932366e-02
##  [6] 3.653734e-02 3.253358e-02 2.304659e-02 1.797740e-02 1.560507e-02
## [11] 1.186749e-02 1.035898e-02 9.562788e-03 8.905462e-03 8.063259e-03
## [16] 7.734146e-03 7.119611e-03 5.954311e-03 5.204289e-03 5.000616e-03
```

```
##  [21] 4.638838e-03 4.347220e-03 3.637689e-03 3.093697e-03 3.006981e-03
##  [26] 2.802544e-03 2.562521e-03 2.517216e-03 2.275200e-03 2.164776e-03
##  [31] 1.955951e-03 1.904237e-03 1.884262e-03 1.796665e-03 1.736167e-03
##  [36] 1.714203e-03 1.522302e-03 1.507679e-03 1.458674e-03 1.350296e-03
##  [41] 1.251350e-03 1.211058e-03 1.183220e-03 1.102305e-03 1.093069e-03
##  [46] 1.025292e-03 9.639417e-04 9.157397e-04 9.069664e-04 8.537456e-04
##  [51] 8.219966e-04 7.941990e-04 7.403003e-04 7.095454e-04 6.979263e-04
##  [56] 6.874087e-04 6.583020e-04 6.514870e-04 6.384573e-04 6.107118e-04
##  [61] 5.589814e-04 5.383518e-04 5.340009e-04 5.237322e-04 5.015006e-04
##  [66] 4.761315e-04 4.722661e-04 4.554102e-04 4.212711e-04 4.024129e-04
##  [71] 3.945006e-04 3.864169e-04 3.807451e-04 3.676953e-04 3.614507e-04
##  [76] 3.431657e-04 3.359264e-04 3.271601e-04 3.203473e-04 3.109284e-04
##  [81] 3.018827e-04 2.937124e-04 2.789043e-04 2.745495e-04 2.659863e-04
##  [86] 2.624420e-04 2.543587e-04 2.480902e-04 2.459407e-04 2.436194e-04
##  [91] 2.288326e-04 2.258693e-04 2.192537e-04 2.145963e-04 2.107062e-04
##  [96] 2.091023e-04 1.999779e-04 1.974334e-04 1.846152e-04 1.795878e-04
## [101] 1.735445e-04 1.708492e-04 1.697462e-04 1.631356e-04 1.534233e-04
## [106] 1.485045e-04 1.473409e-04 1.451566e-04 1.389646e-04 1.380610e-04
## [111] 1.340478e-04 1.299345e-04 1.274016e-04 1.223676e-04 1.199619e-04
## [116] 1.128680e-04 1.122368e-04 1.096635e-04 1.093274e-04 1.062498e-04
## [121] 1.058551e-04 1.020506e-04 9.908484e-05 9.781567e-05 9.398544e-05
## [126] 9.092109e-05 8.801853e-05 8.730404e-05 8.477241e-05 8.347562e-05
## [131] 8.152581e-05 7.884544e-05 7.650638e-05 7.554365e-05 7.511911e-05
## [136] 7.400247e-05 7.056715e-05 6.814857e-05 6.776279e-05 6.510434e-05
## [141] 6.447453e-05 6.292571e-05 6.180542e-05 6.002372e-05 5.782466e-05
## [146] 5.717674e-05 5.582381e-05 5.527942e-05 5.354580e-05 5.303856e-05
## [151] 5.068964e-05 4.968891e-05 4.850314e-05 4.647284e-05 4.623400e-05
## [156] 4.584140e-05 4.495049e-05 4.390563e-05 4.247953e-05 4.146432e-05
## [161] 4.084874e-05 4.019019e-05 3.964720e-05 3.878329e-05 3.843498e-05
## [166] 3.688040e-05 3.664952e-05 3.583290e-05 3.515011e-05 3.505366e-05
## [171] 3.391365e-05 3.335851e-05 3.241005e-05 3.169786e-05 3.125514e-05
## [176] 3.087783e-05 3.026978e-05 2.931055e-05 2.886484e-05 2.863206e-05
## [181] 2.826342e-05 2.797779e-05 2.721897e-05 2.668047e-05 2.617461e-05
## [186] 2.604711e-05 2.555737e-05 2.516140e-05 2.455633e-05 2.420708e-05
## [191] 2.376005e-05 2.355783e-05 2.264225e-05 2.255969e-05 2.226836e-05
## [196] 2.202312e-05 2.148573e-05 2.115703e-05 2.081364e-05 2.042004e-05
## [201] 1.998892e-05 1.971410e-05 1.960419e-05 1.918296e-05 1.888142e-05
## [206] 1.881138e-05 1.827687e-05 1.795803e-05 1.786769e-05 1.759405e-05
## [211] 1.723535e-05 1.707539e-05 1.676827e-05 1.644661e-05 1.617017e-05
## [216] 1.602972e-05 1.594666e-05 1.551631e-05 1.519890e-05 1.473886e-05
## [221] 1.455203e-05 1.427374e-05 1.421165e-05 1.406782e-05 1.384752e-05
## [226] 1.377286e-05 1.364857e-05 1.331185e-05 1.309351e-05 1.295245e-05
## [231] 1.279037e-05 1.256365e-05 1.244013e-05 1.233795e-05 1.215590e-05
## [236] 1.204701e-05 1.173703e-05 1.155968e-05 1.141257e-05 1.124531e-05
## [241] 1.114667e-05 1.104177e-05 1.097895e-05 1.080645e-05 1.056180e-05
## [246] 1.049260e-05 1.037609e-05 1.020813e-05 1.005888e-05 9.895360e-06
```

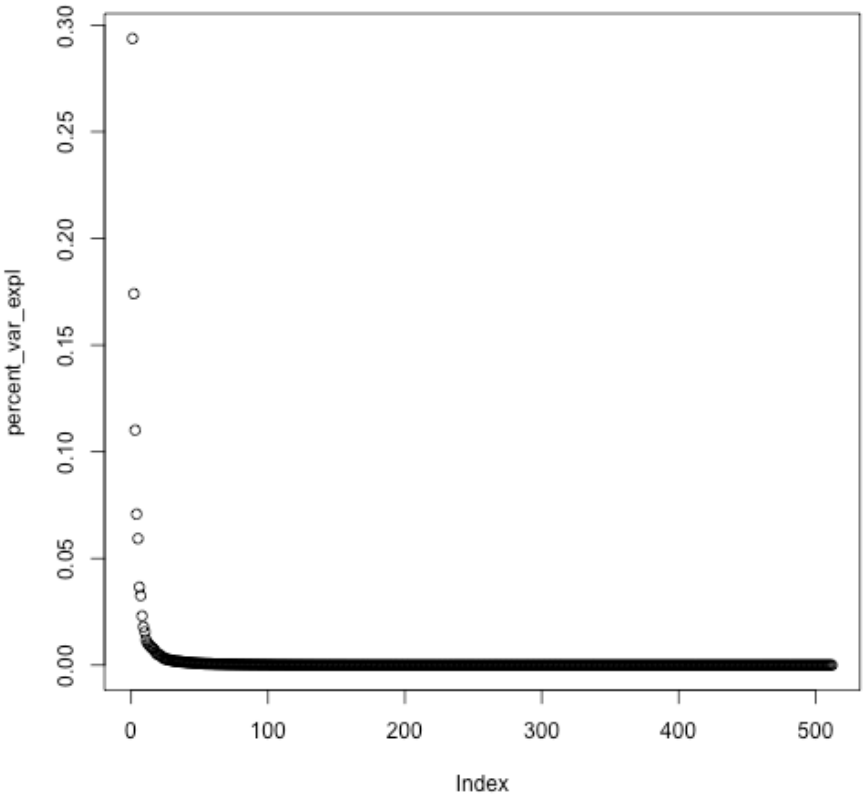
```
## [251] 9.811865e-06 9.613193e-06 9.479352e-06 9.378986e-06 9.153023e-06
## [256] 9.051764e-06 8.983743e-06 8.886154e-06 8.809972e-06 8.740157e-06
## [261] 8.671141e-06 8.531156e-06 8.392633e-06 8.196453e-06 8.140076e-06
## [266] 8.011649e-06 7.953435e-06 7.829323e-06 7.654039e-06 7.621937e-06
## [271] 7.485725e-06 7.467723e-06 7.405480e-06 7.333401e-06 7.240292e-06
## [276] 7.194989e-06 7.065241e-06 6.919291e-06 6.848880e-06 6.730607e-06
## [281] 6.619176e-06 6.520126e-06 6.514401e-06 6.446207e-06 6.337150e-06
## [286] 6.212606e-06 6.153051e-06 6.114325e-06 5.965903e-06 5.875938e-06
## [291] 5.750273e-06 5.718543e-06 5.705740e-06 5.630204e-06 5.502511e-06
## [296] 5.483741e-06 5.437633e-06 5.351984e-06 5.255711e-06 5.171789e-06
## [301] 5.093410e-06 5.039120e-06 4.978832e-06 4.934095e-06 4.854954e-06
## [306] 4.766914e-06 4.735275e-06 4.667886e-06 4.640746e-06 4.608033e-06
## [311] 4.532642e-06 4.457300e-06 4.425859e-06 4.350250e-06 4.304556e-06
## [316] 4.232501e-06 4.186806e-06 4.142825e-06 3.992970e-06 3.966928e-06
## [321] 3.918861e-06 3.875472e-06 3.813711e-06 3.789161e-06 3.713193e-06
## [326] 3.618302e-06 3.607885e-06 3.585161e-06 3.547684e-06 3.468618e-06
## [331] 3.454495e-06 3.426584e-06 3.339830e-06 3.326485e-06 3.296869e-06
## [336] 3.267922e-06 3.185928e-06 3.172358e-06 3.100947e-06 3.074761e-06
## [341] 3.041534e-06 2.969036e-06 2.936903e-06 2.872892e-06 2.864929e-06
## [346] 2.836839e-06 2.733565e-06 2.699403e-06 2.681674e-06 2.640943e-06
## [351] 2.587934e-06 2.540061e-06 2.530734e-06 2.497620e-06 2.468617e-06
## [356] 2.369951e-06 2.340699e-06 2.321952e-06 2.285857e-06 2.277737e-06
## [361] 2.247472e-06 2.211561e-06 2.180942e-06 2.115342e-06 2.063383e-06
## [366] 2.047221e-06 2.043374e-06 1.999642e-06 1.937509e-06 1.917652e-06
## [371] 1.903005e-06 1.876474e-06 1.853056e-06 1.810489e-06 1.799600e-06
## [376] 1.772286e-06 1.712245e-06 1.683251e-06 1.665486e-06 1.632569e-06
## [381] 1.595026e-06 1.588047e-06 1.546560e-06 1.543800e-06 1.509531e-06
## [386] 1.487594e-06 1.460506e-06 1.429576e-06 1.405621e-06 1.380913e-06
## [391] 1.353048e-06 1.333895e-06 1.296814e-06 1.277592e-06 1.262235e-06
## [396] 1.254935e-06 1.237104e-06 1.190919e-06 1.174660e-06 1.138799e-06
## [401] 1.118371e-06 1.117058e-06 1.092217e-06 1.055656e-06 1.051370e-06
## [406] 1.027865e-06 1.023800e-06 9.836733e-07 9.614077e-07 9.529311e-07
## [411] 9.099847e-07 8.997733e-07 8.729718e-07 8.557769e-07 8.416204e-07
## [416] 8.238636e-07 7.929532e-07 7.878549e-07 7.774256e-07 7.519726e-07
## [421] 7.198474e-07 7.118414e-07 6.909226e-07 6.849600e-07 6.650570e-07
## [426] 6.455979e-07 6.209687e-07 6.122259e-07 5.981632e-07 5.741324e-07
## [431] 5.690735e-07 5.588889e-07 5.541817e-07 5.414364e-07 5.277923e-07
## [436] 5.209020e-07 4.927836e-07 4.746133e-07 4.574205e-07 4.539547e-07
## [441] 4.380748e-07 4.298880e-07 4.235944e-07 4.142757e-07 3.852488e-07
## [446] 3.678109e-07 3.601637e-07 3.567490e-07 3.490720e-07 3.290939e-07
## [451] 3.181668e-07 3.110072e-07 3.020185e-07 2.838791e-07 2.738314e-07
## [456] 2.655085e-07 2.532097e-07 2.393193e-07 2.345060e-07 2.295656e-07
## [461] 2.260420e-07 2.164226e-07 2.000145e-07 1.984350e-07 1.827973e-07
## [466] 1.810178e-07 1.723663e-07 1.601232e-07 1.452324e-07 1.391262e-07
## [471] 1.348422e-07 1.282642e-07 1.195902e-07 1.139635e-07 1.086175e-07
## [476] 1.013361e-07 9.308676e-08 8.760459e-08 8.014736e-08 7.856149e-08
```

```
## [481] 7.412638e-08 6.649942e-08 6.379514e-08 5.936378e-08 5.569176e-08
## [486] 5.277864e-08 4.132369e-08 3.901318e-08 3.651215e-08 3.211513e-08
## [491] 3.140609e-08 2.821035e-08 2.625296e-08 2.061852e-08 1.998224e-08
## [496] 1.465441e-08 1.254871e-08 1.163492e-08 9.143222e-09 8.421834e-09
## [501] 6.377670e-09 4.822587e-09 3.499786e-09 2.680442e-09 1.598980e-09
## [506] 1.211690e-09 4.854767e-10 1.376272e-10 1.291291e-33 1.291291e-33
## [511] 1.291291e-33 1.291291e-33
```

```
screeplot(pca.img, npcs = 10)
```




```
plot(percent_var_expl)
```



Problem 2

Discuss whether or not each of the following activities is a data mining task.

Part A

Dividing the customers of a company according to their profitability.

This is not a data mining task; this is simply a sorting task which does not require predicting a feature based on others, nor does it require exploring the relationships between variables.

Part B

Computing the total sales of a company.

This is not a data mining task; this is a simply addition task in which you are not predicting a feature based on others, nor are you exploring the relationships between variables.

Part C

Predicting the future stock price of a company using historical records.

This is a data mining task, because you are trying to make a prediction for a feature based on previous data on that particular feature, along with data on other features.

Part D

Sorting a student database based on student identification numbers.

This is not a data mining task; this is a simple sorting task and you are not predicting a feature based on others, nor are you exploring the relationships between variables.

Part E

Predicting the outcomes of tossing a (fair) pair of dice.

This is not a data mining task; since the die are fair, the probabilities can be predicted easily since we assume that P(heads) = P(tails) = 1/2.

Problem 3

Consider the Boston housing data http://archive.ics.uci.edu/ml/machine-learning-databases/housing/housing.data from the UCI Machine Learning Repository http://archive.ics.uci.edu/ml/ . Data described at http://archive.ics.uci.edu/ml/machine-learning-databases/housing/housing.names

Part A

Describe this data set: how many observations? how many variables (or attributes)?, what is the unit analyzed?

There are 506 observations of 14 variables (13 categorical, 1 binary-valued attribute), and the features are all numerical.

Part B

Load the data into R and can call it Boston.Housing. Consider the fact that the columns of the dataset are unequally separated by white spaces.

```
Boston.Housing <- read.table("/var/folders/xr/ykjyy2_n71n7rmxhgnytgv0r0000gn/T//RtmpcYz1Ln/d
```

```
## Warning in file(file, "rt"): cannot open file '/var/folders/xr/
## ykjyy2_n71n7rmxhgnytgv0r0000gn/T//RtmpcYz1Ln/data5b6669088b1': No such file
## or directory
```

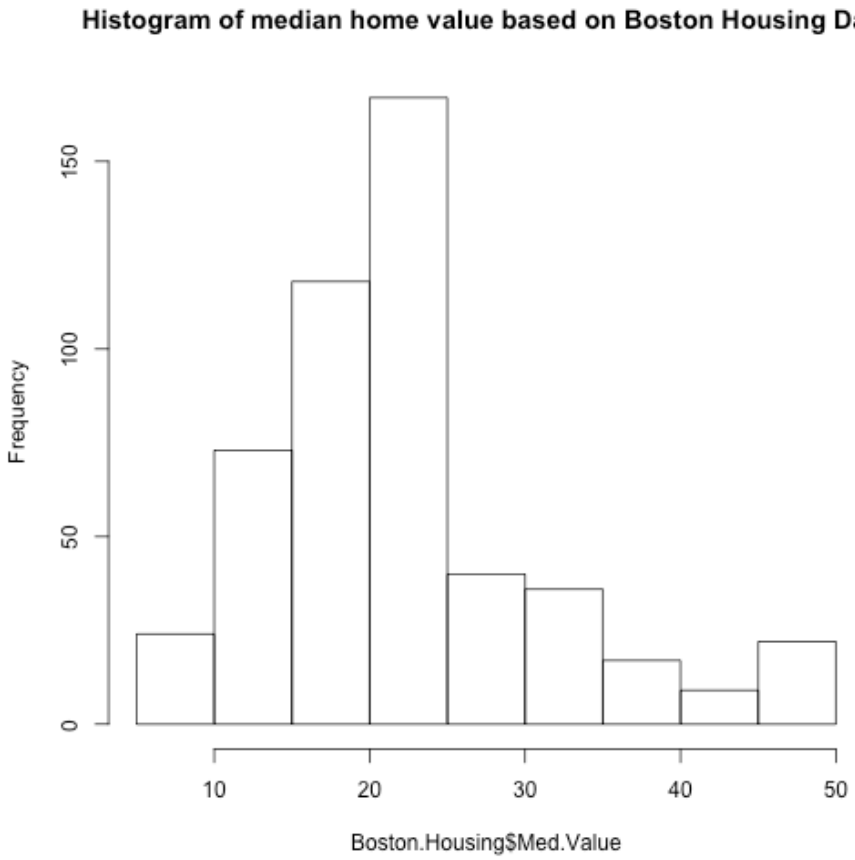
```
## Error in file(file, "rt"): cannot open the connection
```

```
View(Boston.Housing)
str(Boston.Housing)
```

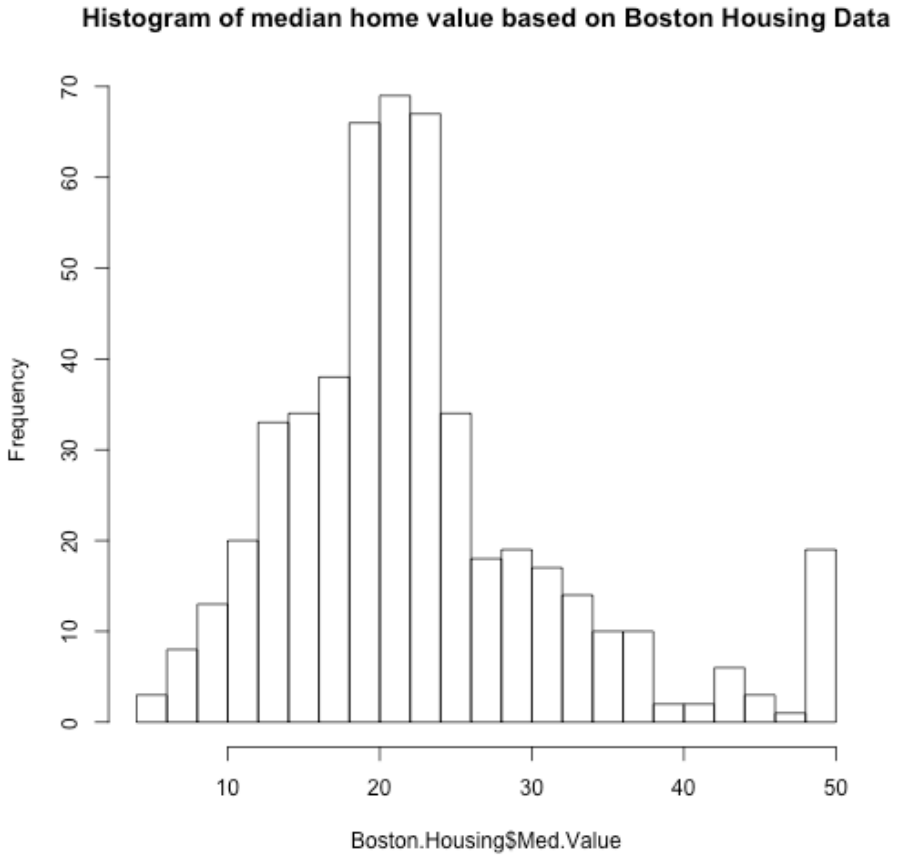
Part C

Produce a histogram of the median value of owner-occupied homes with the title "Histogram of median home value based on Boston Housing Data". Using binwidth argument, gradually increase the number of bins (create four different histograms). What happens to the histogram?

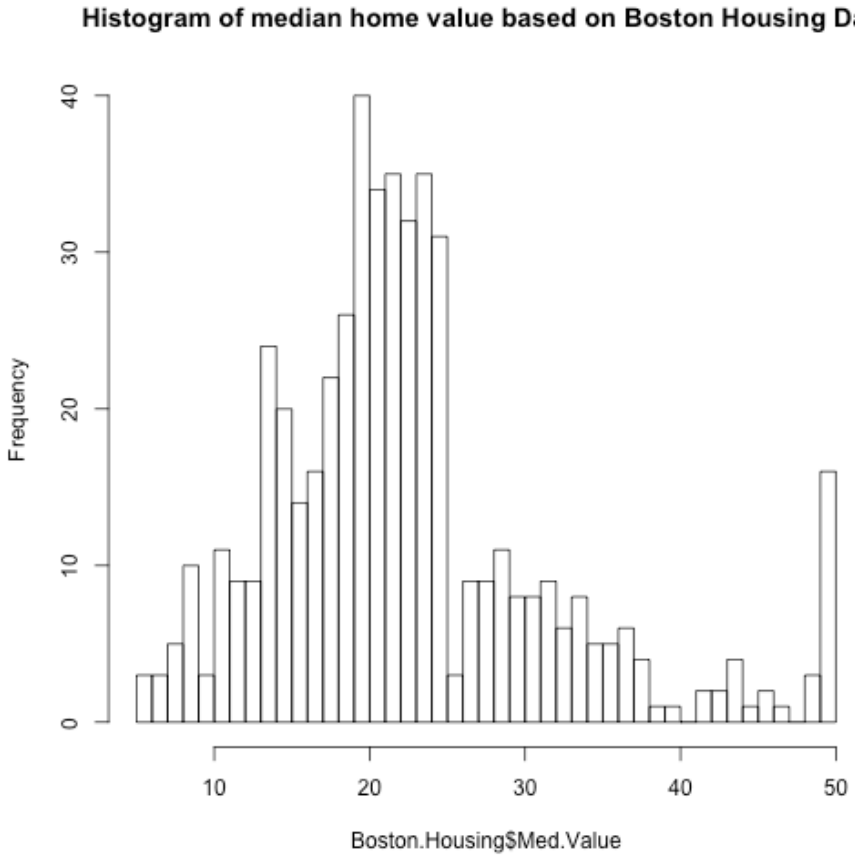
```
# Original Plot
boston.hist <- hist(Boston.Housing$Med.Value, main="Histogram of median home value based on Bos
```



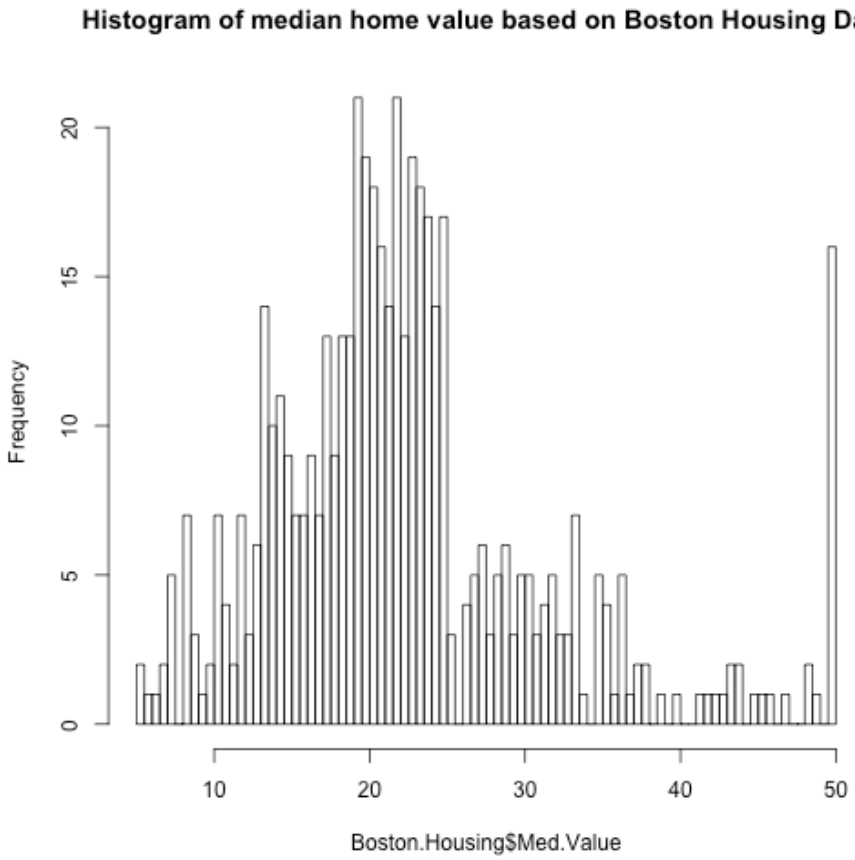

```
# Second Plot
boston.hist.2 <- hist(Boston.Housing$Med.Value, main="Histogram of median home value based on B
```




```
# Third Plot
boston.hist.3 <- hist(Boston.Housing$Med.Value, main="Histogram of median home value based on B
```




```
# Fourth and Final Plot
boston.hist.4 <- hist(Boston.Housing$Med.Value, main="Histogram of median home value based on B
```

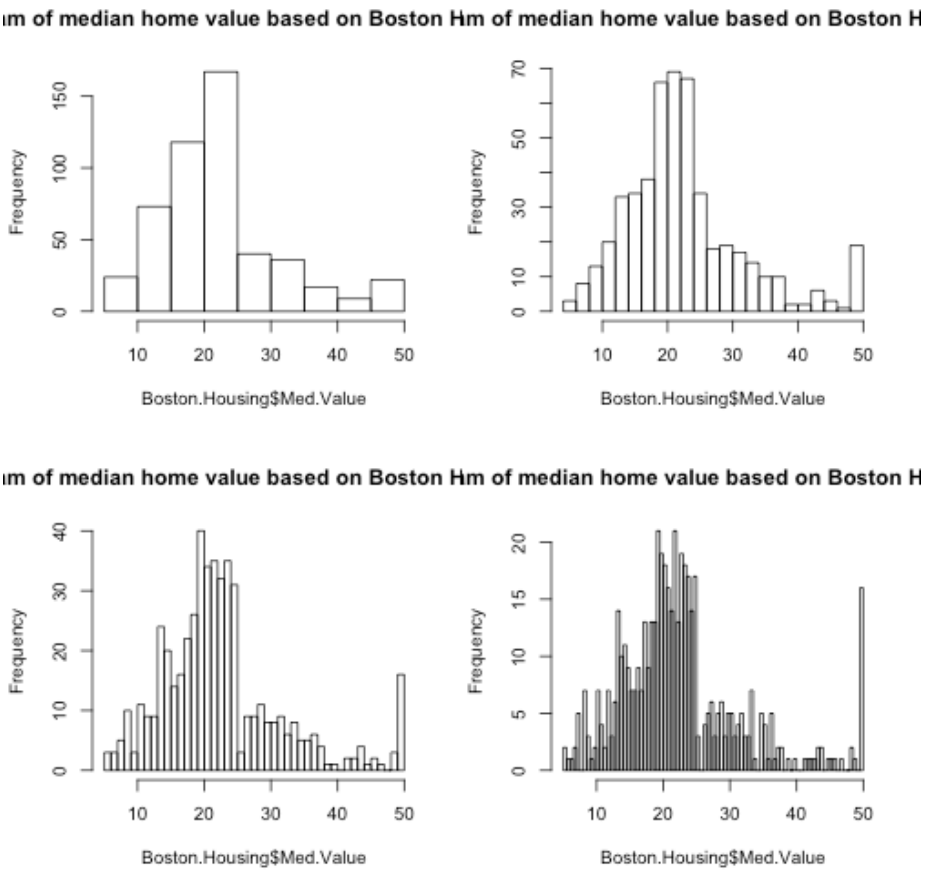


As you increase the number of bins, you begin to get a clearer picture of where the exact data points are, and a better idea of the distribution of the data as a result.

Part D

Show all histograms plot in one chart.

```
par(mfrow = c(2,2))
plot(boston.hist, main="Histogram of median home value based on Boston Housing Data")
plot(boston.hist.2, main="Histogram of median home value based on Boston Housing Data")
plot(boston.hist.3, main="Histogram of median home value based on Boston Housing Data")
plot(boston.hist.4, main="Histogram of median home value based on Boston Housing Data")
```



Part E

Using R, compute mean, median, standard deviation and interquartile range of the median home value. What is a good measure of center and spread of your data? Explain why. Note that you are asked to compute median of the median home value. Does this make sense? Explain.

```
mean(Boston.Housing$Med.Value)
```

```
## [1] 22.53281
```

```
median(Boston.Housing$Med.Value)
```

```
## [1] 21.2
```

```
sd(Boston.Housing$Med.Value)
```

```
## [1] 9.197104
```

```
IQR(Boston.Housing$Med.Value)
```

```
## [1] 7.975
```

A good measure of center for this data is median because the data is highly skewed right, and therefore the mean would be highly influenced by outliers.

Part F

Create 5 equally distributed ranks of Crime.Rate variable. Then use a boxplot to analyze if the median value of the house significantly differs across the levels of each rank of crime rate by town. Hint: Use the quantile() function.

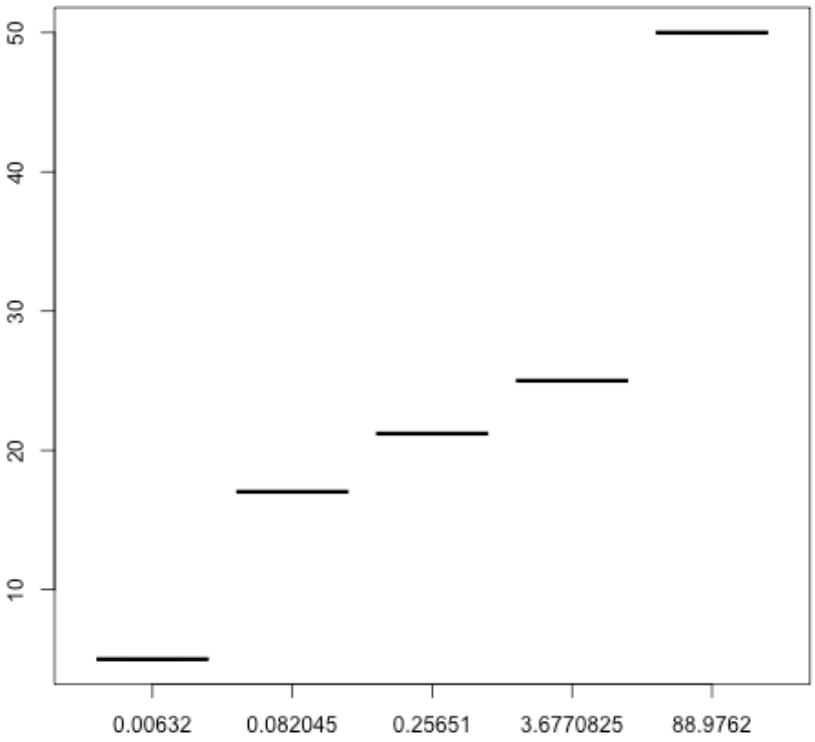
```
quantiles.crime <- quantile(Boston.Housing$Crime.Rate)
quantiles.crime
```

```
##        0%       25%       50%       75%      100%
##  0.006320  0.082045  0.256510  3.677083 88.976200
```

```
quantiles.value <- quantile(Boston.Housing$Med.Value)
quantiles.value
```

```
##     0%    25%    50%    75%   100%
##  5.000 17.025 21.200 25.000 50.000
```

```
boxplot.crime <- boxplot(quantiles.value ~ quantiles.crime)
```



Problem 4

In this problem, you will develop a model to predict whether a given car gets high or low gas mileage based on the Auto data set, which is part of the ISLR package. You will also need to download class package for part d).

```
require(ISLR)
```

```
## Loading required package: ISLR
```

```
require(data.table)
```

```
## Loading required package: data.table
```

```
## data.table 1.9.6  For help type ?data.table or https://github.com/Rdatatable/data.table/wiki

## The fastest way to learn (by data.table authors): https://www.datacamp.com/courses/data-anal

attach(Auto)
```

Part A

Create a binary variable, mpg01, that contains a 1 if mpg contains a value above its median, and a 0 if mpg contains a value below its median. You can compute the median using the median() function. Make sure that you make mpg01 a factor variable. Also, use the data.table() function to create a single data set containing both mpg01 and the other Auto variables.

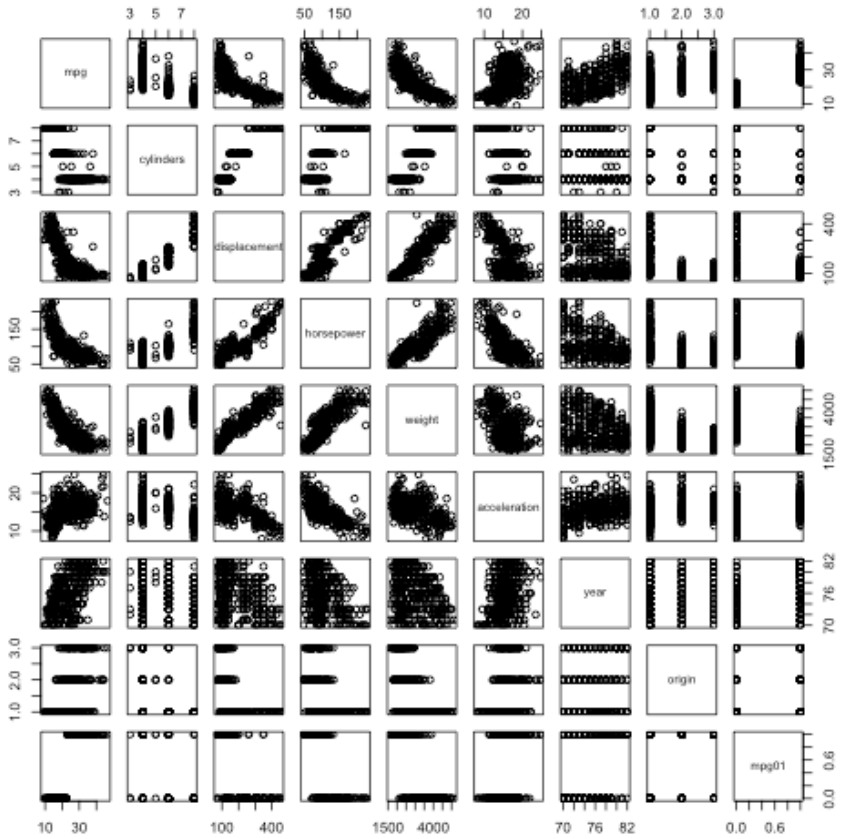
```
median.auto <- median(Auto$mpg)
median.auto

mpg01 <- ifelse(Auto$mpg > median.auto, 1, 0)
mpg01

require(tidyr)

## Loading required package: tidyr

require(dplyr)

## Loading required package: dplyr

## ------------------------------------------------------------------------

## data.table + dplyr code now lives in dtplyr.
## Please library(dtplyr)!

## ------------------------------------------------------------------------

##
## Attaching package: 'dplyr'

## The following objects are masked from 'package:data.table':
##
##     between, last
```

```
## The following objects are masked from 'package:stats':
##
##     filter, lag

## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union

new.auto.data <- Auto %>% mutate(mpg01)
new.auto.data %>% head(8)
View(new.auto.data)
```

Part B

Explore the data graphically in order to investigate the association between mpg01 and the other features. Which of the other features seem most likely to be useful in predicting mpg01? Scatterplots and boxplots and other graphical devices discussed in section may be useful tools to answer this question (you should at least include 3 different graphs). Describe your findings.

```
pairs(new.auto.data[,-9])
```




```
prcomp(new.auto.data[,-9])
```

```
## Standard deviations:
## [1] 855.6833897  38.9157794  16.1763416   4.8256407   2.3556055   1.6903342
## [7]   0.6015086   0.5105322   0.2526710
##
## Rotation:
##                          PC1           PC2           PC3           PC4
## mpg           0.0075959055  0.017579260  0.041915378 -0.830649911
## cylinders    -0.0017925745 -0.013322407  0.007280814  0.004355385
## displacement -0.1143381921 -0.945571556  0.303869295 -0.009291657
## horsepower   -0.0389660857 -0.298327547 -0.947540008 -0.063348792
## weight       -0.9926446456  0.120867641  0.002658325 -0.003548265
## acceleration  0.0013528120  0.034829902  0.076884030  0.020802239
## year          0.0013368989  0.023950015  0.044124423 -0.551152391
```

```
## origin       0.0005515271  0.003246773 -0.012371683 -0.020055224
## mpg01        0.0004437396  0.001528987 -0.001797744 -0.035858877
##                       PC5          PC6           PC7           PC8
## mpg        -0.548144018 -0.055960553  0.0360166014  0.0299940051
## cylinders   0.009315518 -0.015364715 -0.3869998251  0.9134992864
## displacement  0.003362603  0.010685368 -0.0005648496 -0.0165075486
## horsepower   0.005862309  0.086334248  0.0081645440  0.0086521631
## weight     -0.003931791 -0.003528969  0.0001869075 -0.0001269088
## acceleration -0.125417533  0.988149496 -0.0092690666  0.0135159819
## year        0.824606812  0.111930547 -0.0296676880 -0.0171849687
## origin     -0.052966404 -0.008755572 -0.9182580928 -0.3910888052
## mpg01      -0.029645099 -0.002752326  0.0685777929 -0.1041325103
##                         PC9
## mpg          -4.590006e-02
## cylinders     1.231855e-01
## displacement  1.598754e-04
## horsepower   -2.773840e-03
## weight       -1.926289e-05
## acceleration  1.891949e-03
## year          5.324454e-03
## origin        2.008495e-02
## mpg01         9.910981e-01
```

```
boxplot()
```

```
## Error in boxplot.default(): argument "x" is missing, with no default
```

It seems that Horsepower, Weight and Acceleration seem to have the most correlation between themselves and the mpg01 variable. All the other variables seem highly uncorrelated.

Part C

Split the data into a training set (75%) and a test set (25%). Call them train.set and test.set, respectively. The sample() command may be useful for answering this question.

```
require(base)
# Determine number and index of training points to use
train.number <- 0.75 * length(mpg01)
train.number
```

```
## [1] 294
```

```
train.indices <- sample.int(392, train.number)
train.indices
```

```
##  [1]  21 383 167  83 284 304 144 316  52 114 171 116 106 172 382 184 373
## [18] 155 203 274  23 354  93   9 282 327  45 279 149 377 151 290   8 321
## [35] 255 260 270 291 176 112 257  35 372 154 345 230 187 275 165 103 197
## [52] 206  53 253  28 289 198 179 215 147 164  26 199 320 125 351  43 375
## [69] 238 347 264 357 234 248 133 392 168 111 113 363 319 318  17 301 229
## [86] 276 190 200 371  78  71 134 378 173 359  81 288  18 191 329 175  84
## [103] 139 335  67  27 169 245 196 246  33  68 204 121 140 366 273 380 385
## [120] 362  14 367 349  10 332 365  44  50 391 115  82 157  60  42 251 129
## [137] 118  66   5  22  91  97  95 350  80 376  76  15 348  65 174 205 236
## [154]  75 269 266  56 182 263   1 352 369 307  89  37 296 145 254 105 353
## [171] 328 298   4 231 228 285 100  34 239 225 277 223 338 261  36 356 210
## [188]  99 343 272 156  61 294 388 283 150 186 102  51 310 303 300 137 339
## [205] 128 374  30 370  59  16 180 130  74 244  72  79 232 295  96  73 342
## [222] 207 299 163  70  54 323  19 322 381 166 355 189 341 384  86 240 101
## [239] 252 136 135 242 192 161 208 123  11  94 268 110 222   6 361 107 249
## [256] 227 226 331 278 281 265  57 344 219 317 209  69  32 324 358 340 224
## [273]  12   7  55 138 237  87  90 153  98 325 212 152  62 218  49 333  20
## [290] 178  24 160 146 312
```

```
# Extract the training set using our train indices
train.set <- new.auto.data[train.indices,]
str(train.set)
```

```
## 'data.frame':    294 obs. of  10 variables:
##  $ mpg         : num  25 26 23 28 17 28.4 32 34.3 30 26 ...
##  $ cylinders   : num  4 4 4 4 8 4 4 4 4 4 ...
##  $ displacement: num  110 156 140 98 305 151 83 97 88 98 ...
##  $ horsepower  : num  87 92 83 80 130 90 61 78 76 90 ...
##  $ weight      : num  2672 2585 2639 2164 3840 ...
##  $ acceleration: num  17.5 14.5 17 15 15.4 16 19 15.8 14.5 15.5 ...
##  $ year        : num  70 82 75 72 79 79 74 80 71 73 ...
##  $ origin      : num  2 1 1 1 1 1 3 2 2 2 ...
##  $ name      : Factor w/ 304 levels "amc ambassador brougham",..: 211 71 156 104 44 37 88 18 123
##  $ mpg01       : num  1 1 1 1 0 1 1 1 1 1 ...
```

```
# Get the test set from the rest
test.set <- new.auto.data[-c(train.indices),]
str(test.set)
```

```
## 'data.frame':    98 obs. of  10 variables:
##  $ mpg         : num  15 18 15 21 9 28 14 14 14 14 ...
```

```
##  $ cylinders   : num  8 8 8 6 8 4 8 8 8 8 ...
##  $ displacement: num  350 318 400 199 304 140 350 400 351 318 ...
##  $ horsepower  : num  165 150 150 90 193 90 165 175 153 150 ...
##  $ weight      : num  3693 3436 3761 2648 4732 ...
##  $ acceleration: num  11.5 11 9.5 15 18.5 15.5 12 11.5 13.5 13 ...
##  $ year        : num  70 70 70 70 70 71 71 71 71 71 ...
##  $ origin      : num  1 1 1 1 1 1 1 1 1 1 ...
##  $ name      : Factor w/ 304 levels "amc ambassador brougham",..: 36 231 57 7 163 65 54 242 141
##  $ mpg01       : num  0 0 0 0 0 1 0 0 0 0 ...
```

Part D

Using train.set and test.set perform k-NN on the training data, with several values of k, in order to predict mpg01. Use only the variables that seemed most associated with mpg01 in (b) (Justify). What test errors do you obtain? Which value of K seems to perform the best on this data set?

```
require(class)
```

```
## Loading required package: class
```

```
require(base)
train.knn <- train.set[,c("horsepower","weight","acceleration")]
test.knn <- test.set[,c("horsepower","weight","acceleration")]

# Perform K Nearest Neighbors Analysis
# Beginning with 1 neighbor
knn.mpg <- knn(train.knn, test.knn, train.set$mpg01, k = 1)
table(knn.mpg, test.set$mpg01)
```

```
##
## knn.mpg  0  1
##       0 41  7
##       1  7 43
```

```
mean(knn.mpg == test.set$mpg01)
```

```
## [1] 0.8571429
```

```
# Now let's try 2 neighbors
knn.mpg.2 <- knn(train.knn, test.knn, train.set$mpg01, k = 2)
table(knn.mpg.2, test.set$mpg01)
```

```
##
## knn.mpg.2  0  1
##         0 41  9
##         1  7 41
```

```
mean(knn.mpg.2 == test.set$mpg01)
```

```
## [1] 0.8367347
```

```
# Let's try 3
knn.mpg.3 <- knn(train.knn, test.knn, train.set$mpg01, k = 3)
table(knn.mpg.3, test.set$mpg01)
```

```
##
## knn.mpg.3  0  1
##         0 42  6
##         1  6 44
```

```
mean(knn.mpg.3 == test.set$mpg01)
```

```
## [1] 0.877551
```

```
# Accuracy decreased with K = 3
# One more shot with K = 4
knn.mpg.4 <- knn(train.knn, test.knn, train.set$mpg01, k = 4)
table(knn.mpg.4, test.set$mpg01)
```

```
##
## knn.mpg.4  0  1
##         0 42  6
##         1  6 44
```

```
mean(knn.mpg.4 == test.set$mpg01)
```

```
## [1] 0.877551
```

```
# More accurate than K = 3, but not as much as K = 2
# K = 5; last one
knn.mpg.5 <- knn(train.knn, test.knn, train.set$mpg01, k = 5)
table(knn.mpg.5, test.set$mpg01)
```

```
##
## knn.mpg.5  0  1
##         0 41  7
##         1  7 43
```

```
mean(knn.mpg.5 == test.set$mpg01)
```

```
## [1] 0.8571429
```

I used the three variables that seemed to have the highest correlation with mpg01 according to the scatterplots and boxplots from Part B.

The test errors obtained are…

According to the outputs from above, the K Nearest Neighbors algorithm with K = 2 ended up with the highest probability of 0.9591837.