DATA 605 : Final Exam - Problem2

Ramnivas Singh

12/14/2021

Problem 2 - Digit Recognizer

Digit Recognizer is one of the basic and first problem that a budding Machine Learning engineer should try their hands on. It is a simple problem where the challenge is to recognize hand-written digits.

1. Go to Kaggle.com and build an account if you do not already have one. It is free.

Answer : An existing account used to access Kaggle.com

---

2. Go to https://www.kaggle.com/c/digit-recognizer/overview, accept the rules of the competition, and download the data. You will not be required to submit work to Kaggle, but you do need the data.

Answer :

---

3. Using the training.csv file, plot representations of the first 10 images to understand the data format. Go ahead and divide all pixels by 255 to produce values between 0 and 1. (This is equivalent to min-max scaling.)

```
train <- read.csv("train.csv")
test <- read.csv("test.csv")
# Total Rows of training dataset
nrow(train) # Dataset has 4200 records
```

Answer :

```
## [1] 42000
```

```
# Total columns of training dataset
ncol(train)
```

```
## [1] 785
```

```r
# Total Rows of training dataset
nrow(test) # Dataset has 4200 records
```

```
## [1] 28000
```

```r
# Total columns of training dataset
ncol(test)
```

```
## [1] 784
```

```r
# Print top records from train dataset
head(train[1:10])
```

```
##   label pixel0 pixel1 pixel2 pixel3 pixel4 pixel5 pixel6 pixel7 pixel8
## 1     1      0      0      0      0      0      0      0      0      0
## 2     0      0      0      0      0      0      0      0      0      0
## 3     1      0      0      0      0      0      0      0      0      0
## 4     4      0      0      0      0      0      0      0      0      0
## 5     0      0      0      0      0      0      0      0      0      0
## 6     0      0      0      0      0      0      0      0      0      0
```

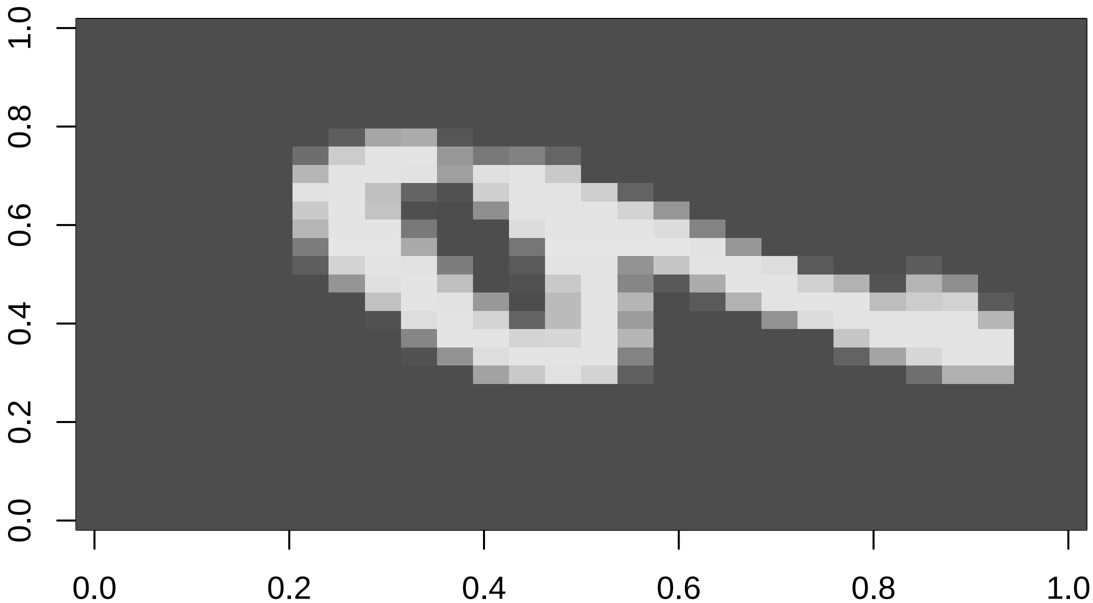
```r
summary(train[train$label==1, 408])
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##     0.0   253.0   253.0   246.5   254.0   255.0
```

```r
summary(train[train$label==0, 408])
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   0.000   0.000   0.000   4.517   0.000 255.000
```

```r
m<-matrix(unlist(train[12,-1]), nrow=28, byrow = T)
image(m, col = grey.colors(255))
```

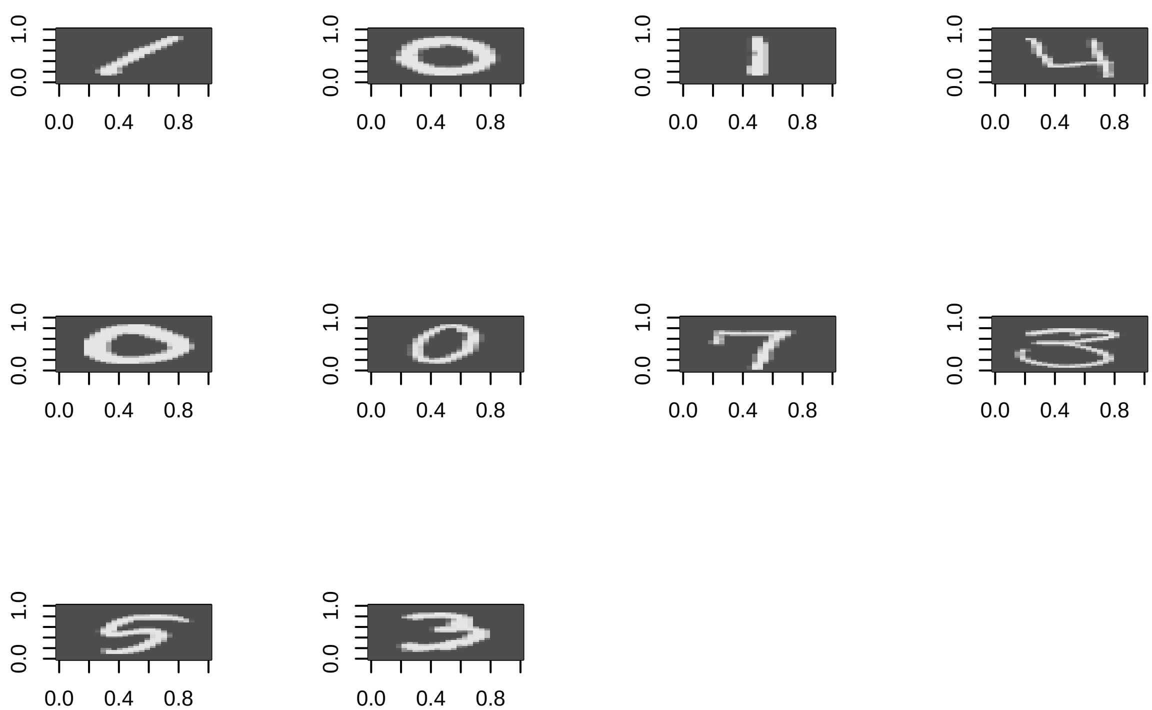



```
flip <- function(matrix){
    apply(matrix, 2, rev)
}

digit<-function(x){
  m<-matrix(unlist(x), nrow=28, byrow=T)
  m<-t(apply(m, 2, rev))
  image(m, col=grey.colors(255))
}

par(mfrow=c(3,4))

for(i in 1:10){
  digit(train[i, -1])
}

# divide all pixels by 255 to produce values between 0 and 1.

train_255 <- train/255.0
test_255 <- test/255.0
head(train_255[1:10])
```

```
##          label pixel0 pixel1 pixel2 pixel3 pixel4 pixel5 pixel6 pixel7 pixel8
## 1 0.003921569      0      0      0      0      0      0      0      0      0
## 2 0.000000000      0      0      0      0      0      0      0      0      0
```

```
## 3 0.003921569        0        0        0        0        0        0        0        0        0
## 4 0.015686275        0        0        0        0        0        0        0        0        0
## 5 0.000000000        0        0        0        0        0        0        0        0        0
## 6 0.000000000        0        0        0        0        0        0        0        0        0
```

```r
# In dataset-remove the first column of label. Create a new data set for this operation apply Min-Max n
normalize <- function(x, na.rm = TRUE) {
    return((x- min(x)) /(max(x)-min(x)))
}

train_b <- train %>% select( 2:ncol(.) )
train_b<- as.data.frame(lapply(train_b[,-1], normalize))
par(mfrow=c(3,4))
```




```r
#Removing Missing values from training(train) dataframe
train[is.na(train)] <- 0
```

---

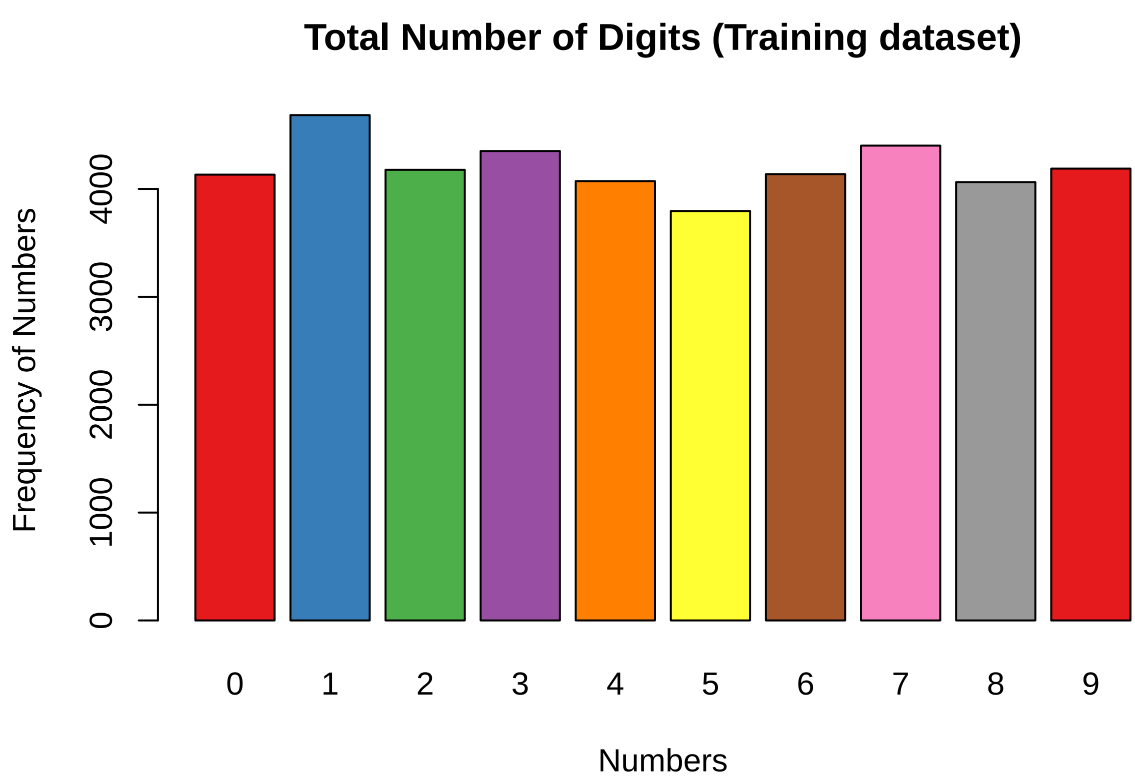
4. What is the frequency distribution of the numbers in the dataset?

Answer : Frequency Distribution : A frequency distribution is a representation, either in a graphical or tabular format, that displays the number of observations within a given interval or categories. It is also called the Frequency Distribution table. Given this dataset, frequency distribution table shows occurrence of various labels in training dataset. To investigate the balance of the data for each label, function will plot all of them along with their name. A plot for the same is below

```
# Table shows numeric representation of frequency distribution
table(train$label)
```

```
##
##    0    1    2    3    4    5    6    7    8    9
## 4132 4684 4177 4351 4072 3795 4137 4401 4063 4188
```

```
# Bar plot chart to show frequency distribution
barplot(table(train$label), main="Total Number of Digits (Training dataset)", col=brewer.pal(10,"Set1")
    xlab="Numbers", ylab = "Frequency of Numbers")
```



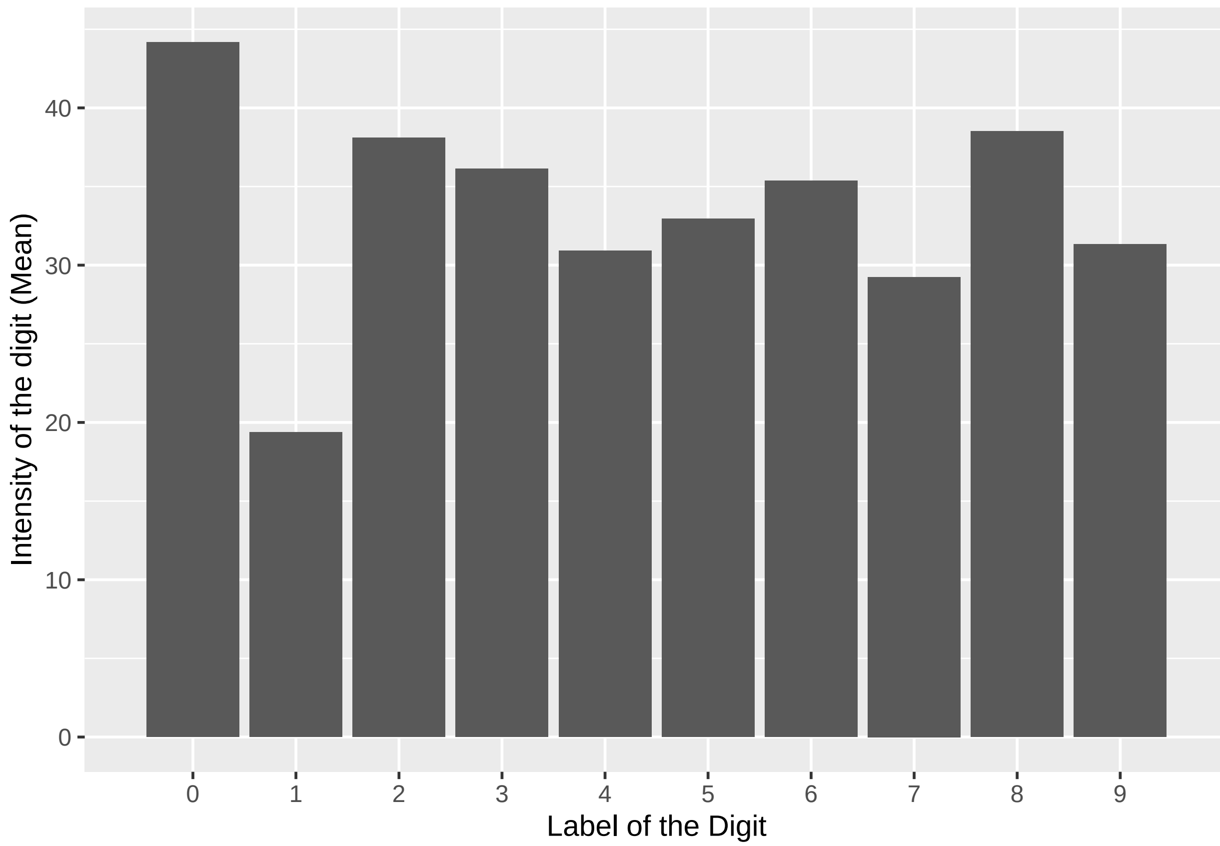
---

5. For each number, provide the mean pixel intensity. What does this tell you?

Answer : To find mean pixel intensity of all of the pixels, we can reshape the dataframe. The output is actually a series, indexed by the original index as well as the pixel label:

```
# Lets calculate mean of each row in training data set to find pixel intensity of all of the pixels
train$intensity <- apply(train[,-1], 1, mean)
intensity_by_label <- aggregate (train$intensity, FUN = mean, by = list(train$label))
```

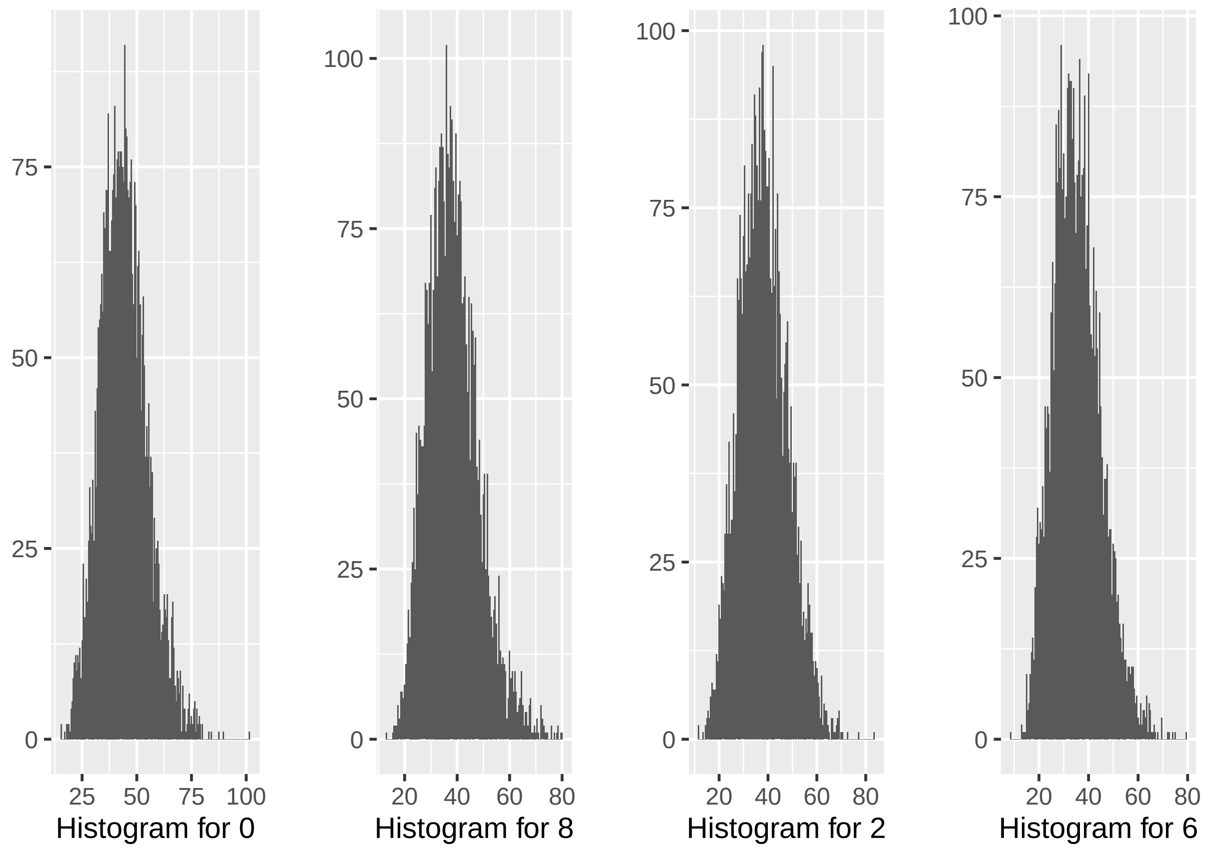
```r
# Now lets plot intensity_by_label
ggplot(data=intensity_by_label, aes(x=Group.1, y = x)) +
    geom_bar(stat="identity")+scale_x_discrete(limits=0:9) + xlab("Label of the Digit") +
    ylab("Intensity of the digit (Mean)")
```



We see that pixel values have different intensity across the dataset. 0, 8, 2, 6 are with higher mean pixel intensity where as 1, 4, 7, and with low mean pixel intensity. If we want to bin the intensity values into statistical quantiles, we can do that. Overall digit "0" is the most intense and "1" is the less intense.

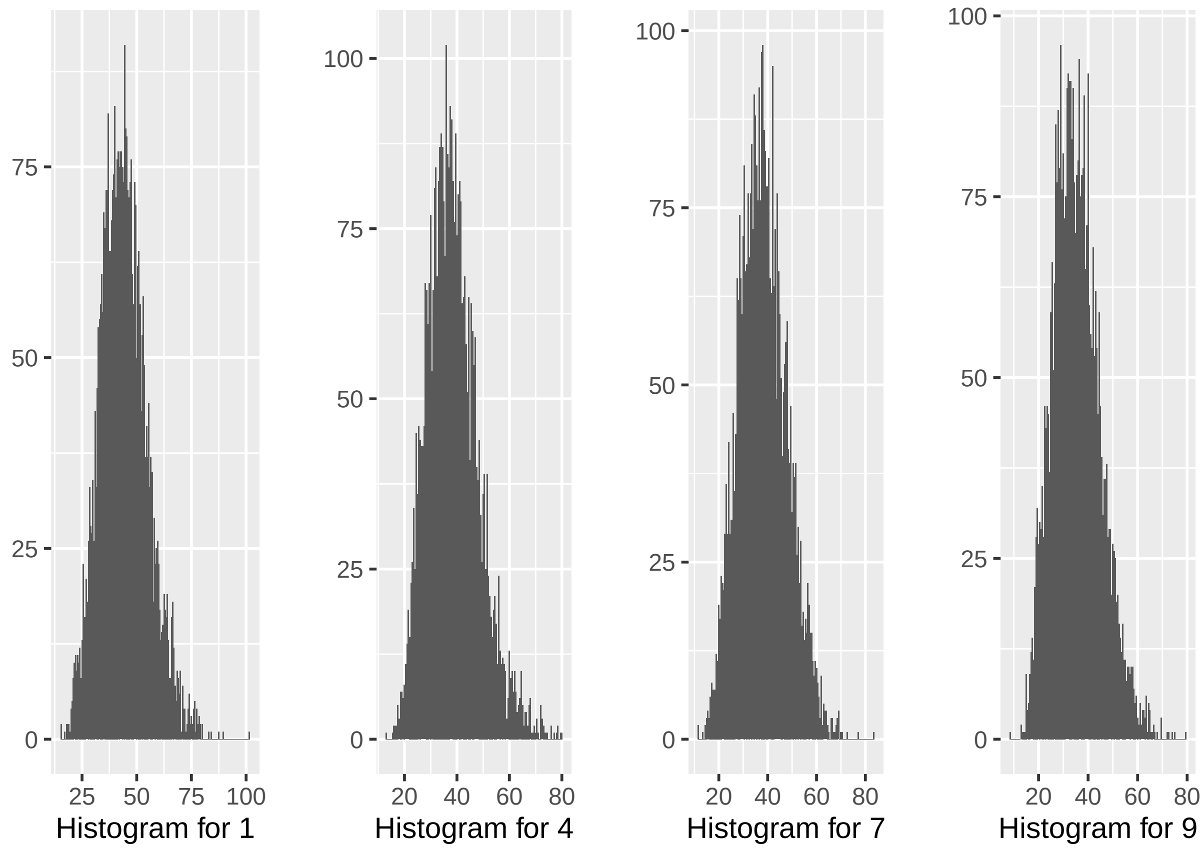
Lets plot Histogram for 0, 8, 2, 6

```r
grid.arrange(qplot(subset(train, label ==0)$intensity, binwidth = .5, xlab = "Histogram for 0"),
qplot(subset(train, label ==8)$intensity, binwidth = .5, xlab = "Histogram for 8"),
qplot(subset(train, label ==2)$intensity, binwidth = .5, xlab = "Histogram for 2"),
qplot(subset(train, label ==6)$intensity, binwidth = .5, xlab = "Histogram for 6"),ncol = 4)
```



Lets plot Histogram for 1, 4, 7, 9

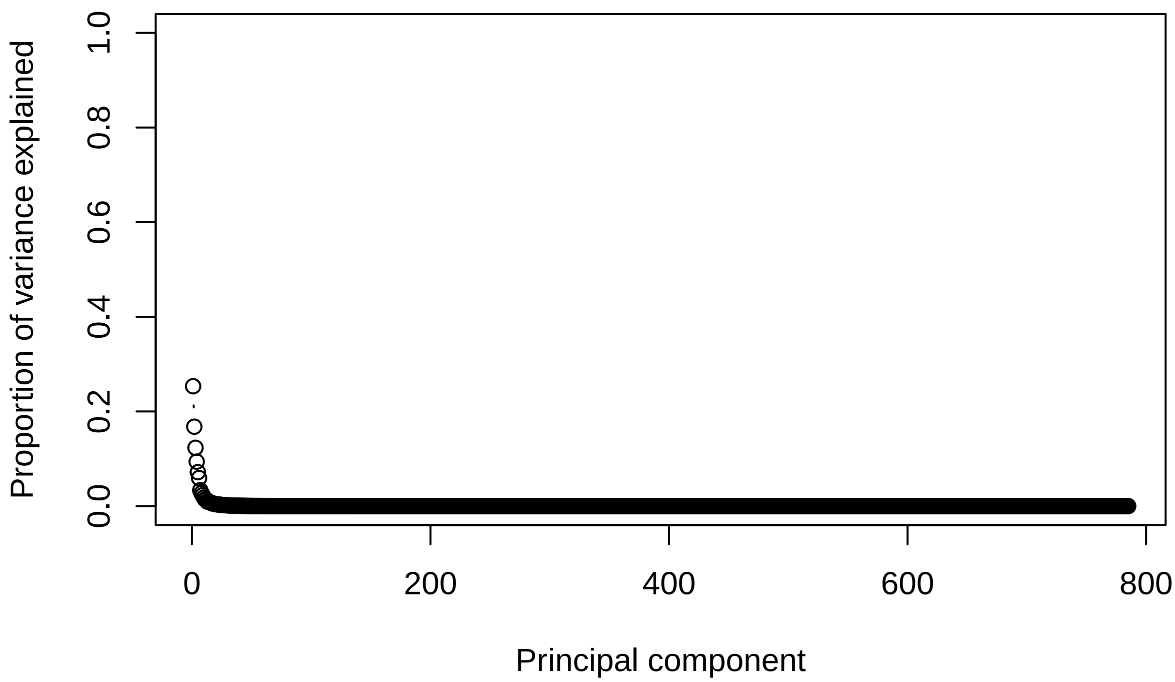
```
grid.arrange(qplot(subset(train, label ==0)$intensity, binwidth = .5, xlab = "Histogram for 1"),
qplot(subset(train, label ==8)$intensity, binwidth = .5, xlab = "Histogram for 4"),
qplot(subset(train, label ==2)$intensity, binwidth = .5, xlab = "Histogram for 7"),
qplot(subset(train, label ==6)$intensity, binwidth = .5, xlab = "Histogram for 9"),ncol = 4)
```



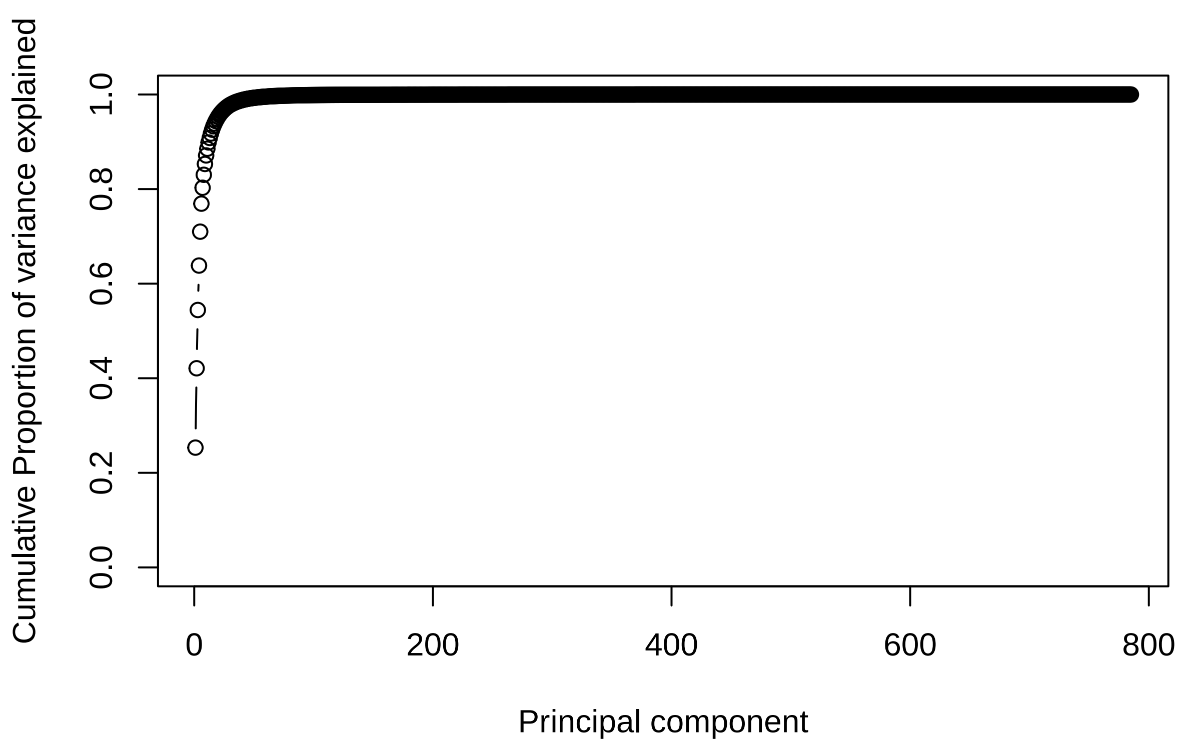
---

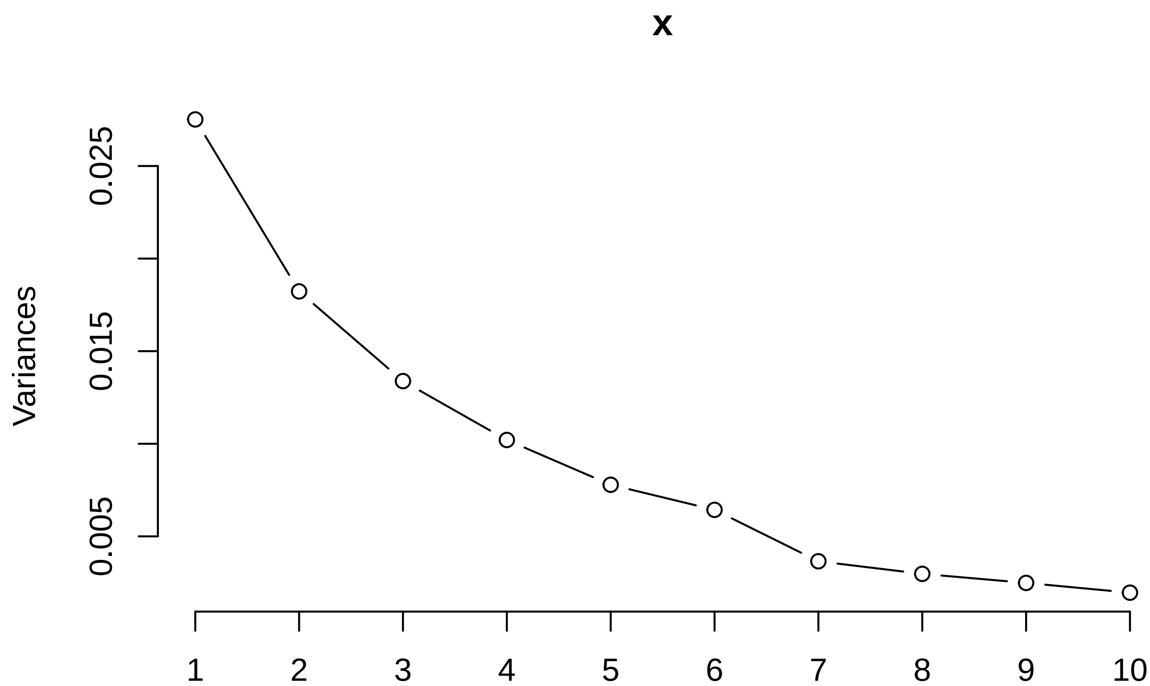
6. Reduce the data by using principal components that account for 95% of the variance. How many components did you generate? Use PCA to generate all possible components (100% of the variance). How many components are possible? Why?

```
pcaCharts <- function(x) {
    x.var <- x$sdev ^ 2
    x.pvar <- x.var/sum(x.var)
    par(mfrow=c(1,1))
    plot(x.pvar,xlab="Principal component", ylab="Proportion of variance explained", ylim=c(0,1), type=
    plot(cumsum(x.pvar),xlab="Principal component", ylab="Cumulative Proportion of variance explained",
    screeplot(x,type="l")
    par(mfrow=c(1,1))
}

#Reducing data using PCA
train_norm<-as.matrix(train[,-1])/255
train_norm_cov <- cov(train_norm)
pca <- prcomp(train_norm_cov)

pcaCharts(pca)
```


Proportion of variance explained
0.0
0.2
0.4
0.6
0.8
1.0
0
200
400
600
800
Principal component

Answer :


Cumulative Proportion of variance explained
0.0
0.2
0.4
0.6
0.8
1.0
0
200
400
600
800
Principal component




```r
# Calculate the variance explained by each principal component
variance_explained<-as.data.frame(pca$sdev^2/sum(pca$sdev^2))
variance_explained<-cbind(1:785, cumsum(variance_explained))
colnames(variance_explained)<-c("Number", "Variance")
variance_explained<-as.data.frame(variance_explained)
head(variance_explained,100)
```

```
##    Number  Variance
## 1       1 0.2533019
## 2       2 0.4211143
## 3       3 0.5443375
## 4       4 0.6382797
## 5       5 0.7099622
## 6       6 0.7691849
## 7       7 0.8028326
## 8       8 0.8302254
## 9       9 0.8531247
## 10     10 0.8711668
## 11     11 0.8853364
## 12     12 0.8986528
## 13     13 0.9080900
## 14     14 0.9174678
## 15     15 0.9256073
## 16     16 0.9328088
## 17     17 0.9384871
## 18     18 0.9438281
```

```
## 19       19 0.9483989
## 20       20 0.9527310
## 21       21 0.9564960
## 22       22 0.9598711
## 23       23 0.9629199
## 24       24 0.9656478
## 25       25 0.9682119
## 26       26 0.9705036
## 27       27 0.9726586
## 28       28 0.9746368
## 29       29 0.9764279
## 30       30 0.9779677
## 31       31 0.9793804
## 32       32 0.9807166
## 33       33 0.9818930
## 34       34 0.9830205
## 35       35 0.9840633
## 36       36 0.9850209
## 37       37 0.9858696
## 38       38 0.9866471
## 39       39 0.9873878
## 40       40 0.9880989
## 41       41 0.9887694
## 42       42 0.9894178
## 43       43 0.9899900
## 44       44 0.9905075
## 45       45 0.9909903
## 46       46 0.9914504
## 47       47 0.9918771
## 48       48 0.9922717
## 49       49 0.9926425
## 50       50 0.9929780
## 51       51 0.9933007
## 52       52 0.9936134
## 53       53 0.9938955
## 54       54 0.9941623
## 55       55 0.9944194
## 56       56 0.9946575
## 57       57 0.9948887
## 58       58 0.9951032
## 59       59 0.9953134
## 60       60 0.9955119
## 61       61 0.9956998
## 62       62 0.9958863
## 63       63 0.9960555
## 64       64 0.9962159
## 65       65 0.9963657
## 66       66 0.9965039
## 67       67 0.9966386
## 68       68 0.9967638
## 69       69 0.9968865
## 70       70 0.9970029
## 71       71 0.9971165
## 72       72 0.9972235
```

```
## 73       73 0.9973257
## 74       74 0.9974232
## 75       75 0.9975134
## 76       76 0.9976006
## 77       77 0.9976845
## 78       78 0.9977623
## 79       79 0.9978322
## 80       80 0.9978985
## 81       81 0.9979637
## 82       82 0.9980280
## 83       83 0.9980911
## 84       84 0.9981509
## 85       85 0.9982082
## 86       86 0.9982642
## 87       87 0.9983192
## 88       88 0.9983697
## 89       89 0.9984186
## 90       90 0.9984655
## 91       91 0.9985092
## 92       92 0.9985516
## 93       93 0.9985929
## 94       94 0.9986328
## 95       95 0.9986710
## 96       96 0.9987087
## 97       97 0.9987440
## 98       98 0.9987787
## 99       99 0.9988124
## 100     100 0.9988451
```

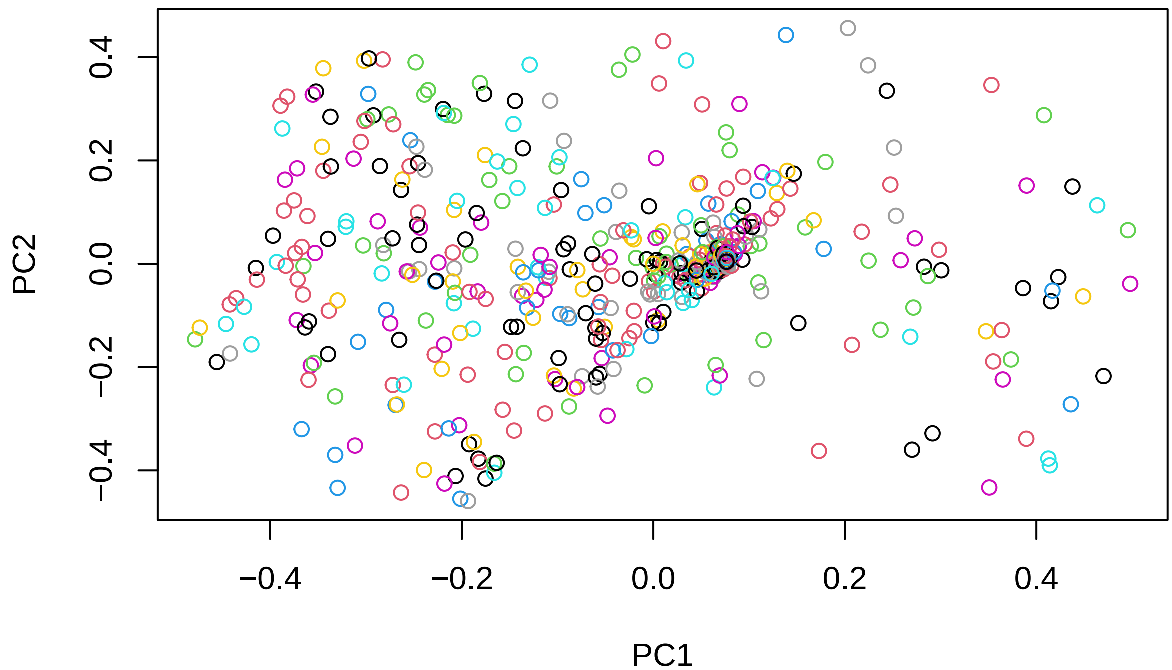
There are around 20 components generated at 95% of variance

---

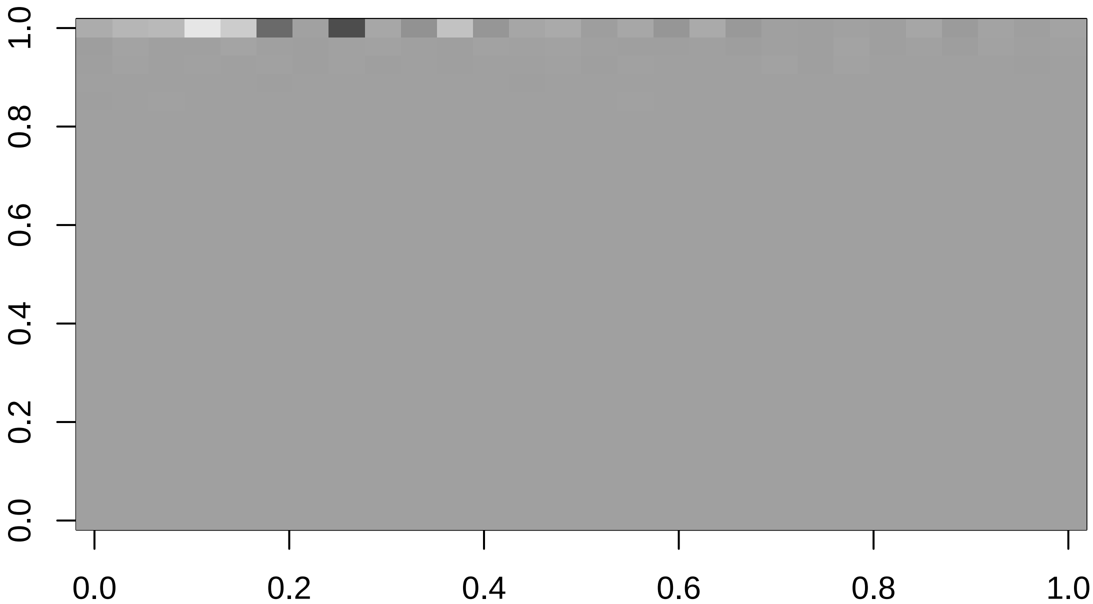
7. Plot the first 10 images generated by PCA. They will appear to be noise. Why?

Answer : Because PCs of higher eigenvalues to show more generic features.The specialized features are also being added for more PCs. So removing removing some PCs with lower eigenvalues actually acting as some sort of regularization and your model is only learning the more general features. If you take all of them the 100% of the data-variations will be restored like the original dimensions.

```
labelClasses <- factor(train$label)
plot(main="",pca$x, col = labelClasses)
```




```
for(i in 1:10){
  digit(pca$x[i, -1])
}
```



---

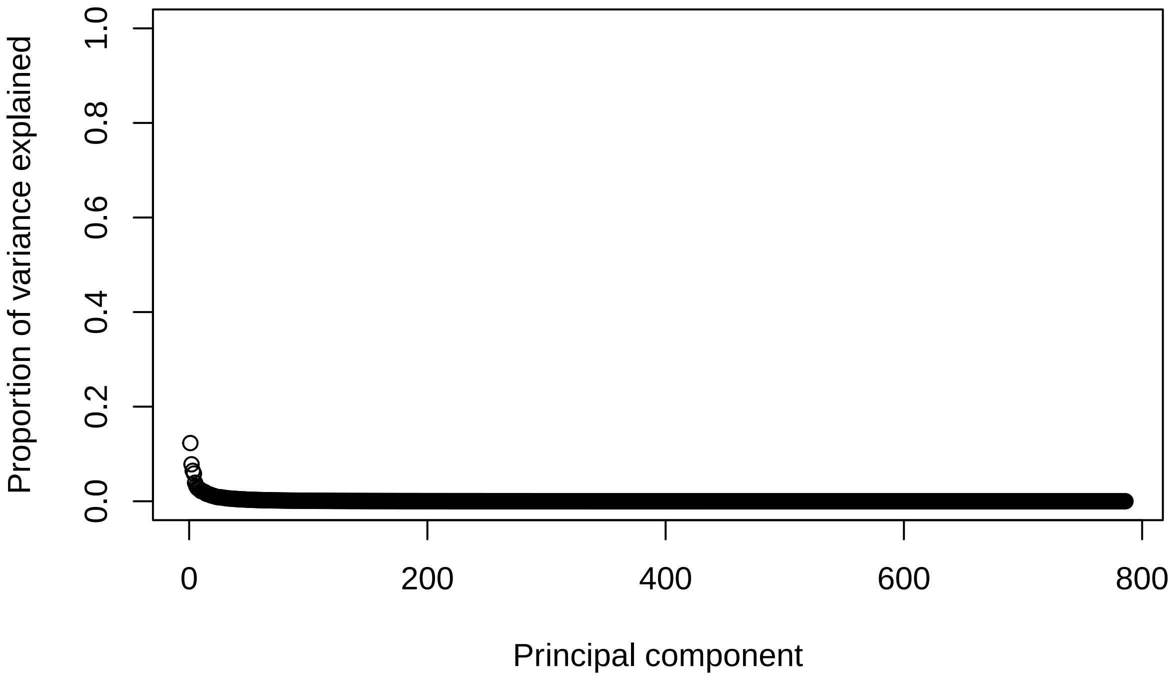
8. Now, select only those images that have labels that are 8's. Re-run PCA that accounts for all of the variance (100%). Plot the first 10 images. What do you see?

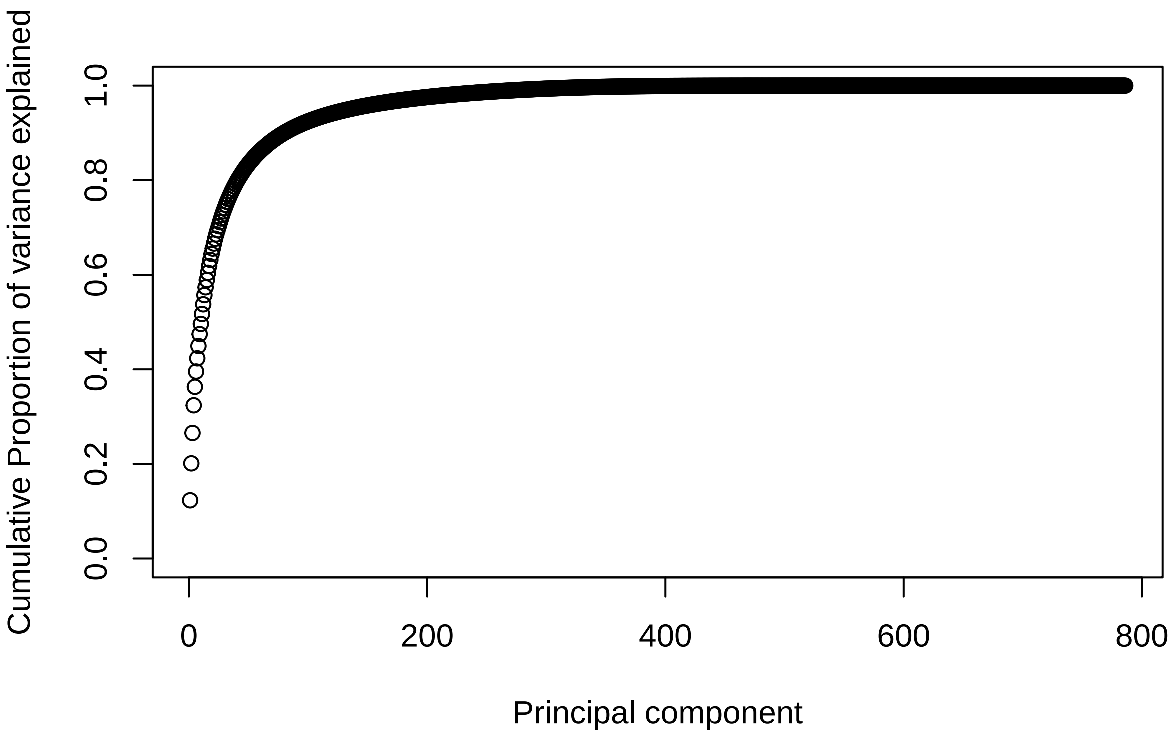
```
train_sub_8<-subset(train, label ==8)
nrow(train_sub_8)
```

Answer :

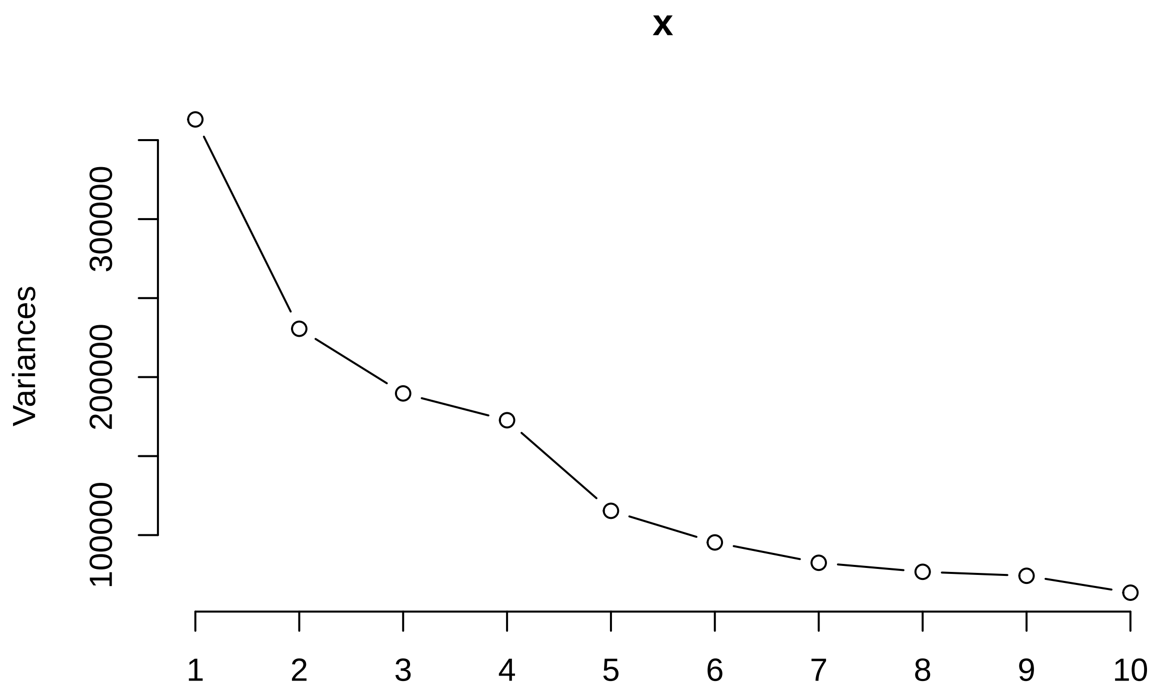
```
## [1] 4063
```

```
pca <- prcomp(train_sub_8)
pcaCharts(pca)
```

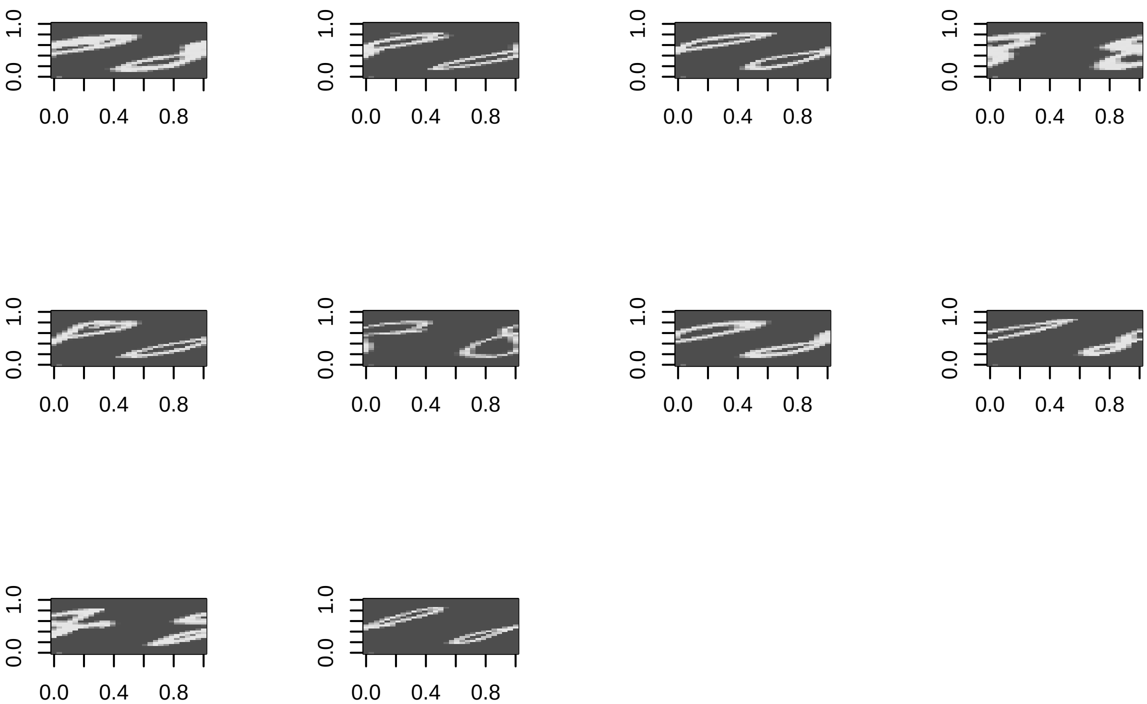

Proportion of variance explained
0.0
0.2
0.4
0.6
0.8
1.0
0
200
400
600
800
Principal component



Cumulative Proportion of variance explained
0.0
0.2
0.4
0.6
0.8
1.0
0
200
400
600
800
Principal component




```r
par(mfrow=c(3,4))
for(i in 1:10){
  digit(train_sub_8[i, -1])
}
```



In above images, different type of 8 letters seems to be visible. ***

9. An incorrect approach to predicting the images would be to build a linear regression model with y as the digit values and X as the pixel matrix. Instead, we can build a multinomial model that classifies the digits. Build a multinomial model on the entirety of the training set. Then provide its classification accuracy (percent correctly identified) as well as a matrix of observed versus forecast values (confusion matrix). This matrix will be a 10 x 10, and correct classifications will

be on the diagonal.

NOTE: Please note models used below for this problem are running on small dataset extracted from train.csv file. Normal laptop doesnt return with the result even after many hours of continuous processing.

Answer : To build a multinomial model on the entirety of the training, I started with training dataset by splitting dataset further into to train and test where 70% data used for training the model and 30% dataset for testing it. Approach is run multinom function which will than return multinomModel than use multinomModel in predict function for probable results. Running this model produced around 93% accuracy.

Lines below for this model are commented, this model for 42K records takes around 4 days of time to return the results.

```
train <- read.csv("train_1.csv")

train[is.na(train)] <- 0
sample_size = round(nrow(train)*.70) # setting what is 70%
```

```r
index <- sample(seq_len(nrow(train)), size = sample_size)

train <- train[index, ]
test <- train[-index, ]

#Build Multinomial Model
#multinomModel <- multinom(label ~., family = "multinomial", data = train, MaxNWts =100000, maxit=10);
#summary (multinomModel) # model summary

#predicted_scores <- predict (multinomModel, test, "probs") # predict on multinomModel data
#predicted_class <- predict (multinomModel, test) # model summary

#Confusion Matrix and Misclassification Error
#table(predicted_class, test$label)
#mean(as.character(predicted_class) != as.character(test$label))
```

A classification accuracy of 93.3% from multinom model is probably too high. We should try other ML approaches as well for this problem.

*** Gradient boosted trees model for multinomial *** : Gradient boosting is a machine learning technique used in regression and classification tasks, among others. It gives a prediction model in the form of an ensemble of weak prediction models, which are typically decision trees. Gradient boosted trees also run directly on the multiclass labels. The model performs much better if I increase the interaction depth slightly. Increasing it past 2-3 is beneficial in large models, but rarely useful with smaller cases like this. I could also play with the learning rate, but won't fiddle with that here for now.

```r
# Model fitting
Xtrain <- as.matrix(train)
Xtest <- as.matrix(test)
ytrain <- train[,1]
ytest <- test[,1]
gbm_result <- gbm.fit(Xtrain,  factor(ytrain), distribution="multinomial", n.trees=500, interaction.dep
```

```
## Iter   TrainDeviance   ValidDeviance   StepSize   Improve
##      1        2.3026             nan     0.0010    0.0161
##      2        2.2939             nan     0.0010    0.0147
##      3        2.2860             nan     0.0010    0.0161
##      4        2.2774             nan     0.0010    0.0120
##      5        2.2704             nan     0.0010    0.0126
##      6        2.2633             nan     0.0010    0.0141
##      7        2.2557             nan     0.0010    0.0131
##      8        2.2487             nan     0.0010    0.0142
##      9        2.2411             nan     0.0010    0.0139
##     10        2.2336             nan     0.0010    0.0134
##     20        2.1644             nan     0.0010    0.0117
##     40        2.0321             nan     0.0010    0.0100
##     60        1.9249             nan     0.0010    0.0077
##     80        1.8270             nan     0.0010    0.0076
##    100        1.7395             nan     0.0010    0.0088
##    120        1.6551             nan     0.0010    0.0079
##    140        1.5792             nan     0.0010    0.0065
##    160        1.5100             nan     0.0010    0.0058
##    180        1.4455             nan     0.0010    0.0073
```

```
##    200        1.3852             nan     0.0010    0.0057
##    220        1.3283             nan     0.0010    0.0059
##    240        1.2753             nan     0.0010    0.0041
##    260        1.2263             nan     0.0010    0.0042
##    280        1.1778             nan     0.0010    0.0053
##    300        1.1306             nan     0.0010    0.0044
##    320        1.0856             nan     0.0010    0.0035
##    340        1.0442             nan     0.0010    0.0037
##    360        1.0028             nan     0.0010    0.0027
##    380        0.9643             nan     0.0010    0.0033
##    400        0.9281             nan     0.0010    0.0032
##    420        0.8923             nan     0.0010    0.0034
##    440        0.8576             nan     0.0010    0.0038
##    460        0.8252             nan     0.0010    0.0027
##    480        0.7931             nan     0.0010    0.0030
##    500        0.7620             nan     0.0010    0.0026
```

```
#summary(gbm_result)
gbm_prediction <- apply(predict(gbm_result, Xtest, n.trees=gbm_result$n.trees),1,which.max) - 1L
# Prediction
gbm_prediction
```

```
##   [1] 4 3 1 7 0 1 1 0 6 0 9 0 3 9 1 3 0 3 9 9 2 5 6 4 0 4 5 7 8 8 2 6 0 2 5 4 1
##  [38] 9 2 9 9 2 4 2 1 0 5 9 7 7 4 0 7 8 4 7 2 4 4 6 2 9 5 6 5 0 7 2 5 3 3 8 5 3
##  [75] 1 2 0 4 4 1 3 8 2 0 9 7 2 2 1 6 2 8 1 0 6 8 8 5 1 0 7 7 3 2 8 4 9 4 4 5 3
## [112] 9 5 2 5 1 1 0 4 6 0 7 3 5 5 7 4 5 1 9 1 9 9 3 2 5 4 4 0 4 6 2 4 6 1 9 5 2
## [149] 6 5 9 6 6 0 1 5 6 7 2 3 9 7 3 9 0 7 2 6 1 2 1 1 5 3 5 6 3 8 4 1 9 5 2 2 5
## [186] 4 3 3 1 2 0 3 6 2 5 5 1 3 4 3 9 0 7 1 7 9 3 4 8 7 0 8 1 1 8 1 7 0 6 9 6 9
## [223] 8 6 7 4 5 5 4 8 9 6 0 7 3 1 6 4 9 5 9 1 1 9 7 2 9 1 7 7 4 6 9 0 2 9 4 2 6
## [260] 1 2 1 7 4 1 6 5 0 0 8 9 2 6 7 0 8 9 0 7 7 9 0 5 2 3 1 3 4 5 0 0 9 9 9 3 2
## [297] 1 7 4 2 9 1 4 6 7 6 6 4 9 5 5 0 3 2 9 8 7 4 9 6 3 2 0 2 5 1 7 2 8 8 4 1 8
## [334] 7 6 3 7 0 8 4 3 6 4 2 9 4 0 7 2 5 1 8 2 4 4 9 6 9 8 0 8 1 5 0 9 0 8 2 3 7
## [371] 2 9 9 1 0 0 4 6 1 8 8 2 7 5 5 8 8 8 4 7 4 8 0 7 8 8 9 7 5 6 5 5 1 3
```

Lets try Ridge regression for prediction and confusion, I'll directly use the multinomial loss function and let the R function do cross validation this time.

```
library(glmnet)
outLm <- cv.glmnet(Xtrain, ytrain, alpha=0, nfolds=3,
                    family="multinomial")
outLm
```

```
##
## Call:  cv.glmnet(x = Xtrain, y = ytrain, nfolds = 3, alpha = 0, family = "multinomial")
##
## Measure: Multinomial Deviance
##
##      Lambda Index Measure      SE Nonzero
## min 0.01624   100  0.8403 0.02915     629
## 1se 0.03753    91  0.8666 0.03246     629
```

```r
predLm <- apply(predict(outLm, Xtest, s=outLm$lambda.min,
                        type="response"), 1, which.max) - 1L
predLm
```

```
##  300 1127 1520  494 1260 1001 1792 1511  790  201 1949 1661  730 1425 1756 1555
##    4    3    1    7    0    1    1    0    6    0    7    0    3    9    1    3
##  268 1536 1102  847  928 1467 1887  216 1730  310  590  243 1979 1208  683 1915
##    0    3    9    9    2    5    6    4    0    4    5    7    8    8    2    6
## 1304 1926  504  352 1341 1308 1686 1562  791 1990   50 1207 1774 1992 1739  771
##    0    2    5    4    1    9    2    9    9    2    4    2    1    0    5    9
##   49 1559  828  328  658  552  674  932 1842  883 1880  635  516 1807 1592  400
##    7    7    4    0    7    8    4    7    2    4    4    6    2    9    5    6
## 1353 1463  812 1055 1611  315 1138 1027  997  482  210  949 1097  704  951 1414
##    5    0    7    2    5    3    3    8    5    3    1    2    0    4    4    1
##  734  717 1614  205  856 1597  600 1867   13  584  398 1738 1025 1228  981  612
##    3    8    2    0    9    7    2    2    1    6    2    8    1    0    6    8
##   11 1150  425  897 1316 1066  334 1262  895 1623  678 1625 1802  298 1184  722
##    8    5    1    0    7    7    3    2    8    4    9    4    4    5    3    9
##  848  665 1679 1737 1995 1429 1711 1705  189 1093 1660 1983 1583 1152 1804  100
##    5    2    5    1    1    0    4    6    0    7    3    5    5    7    4    5
##   16  101  125  800  442 1435 1331 1113 1778 1972  920 1819 1492  560  525 1860
##    1    9    1    9    9    3    2    5    4    4    0    4    6    2    4    6
## 1993 1602 1839   25  217  390 1645  271 1241 1989    1 1833 1688 1359 1395 1626
##    1    9    5    2    6    5    9    6    6    0    1    5    6    7    2    3
## 1489 1460 1858 1258  676  505 1299 1195  709 1176  469  545 1822 1604  666 1565
##    9    7    3    9    0    7    2    6    1    2    1    1    5    3    5    6
##  229  535  483  827 1280  767 1420 1231 1439  497   66 1234 1663 1734  287  943
##    3    8    4    1    9    5    2    2    5    4    3    3    1    2    0    3
## 1728 1013 1622 1628 1436 1922  841  231  211  882 1242 1295 1849  657  388  206
##    6    2    5    5    1    3    4    3    9    0    7    1    7    9    3    4
## 1300  224  986  462 1613 1836  407  591  655 1570  290 1878   46  181  486 1229
##    8    7    0    8    1    1    8    1    7    0    6    9    6    9    8    6
## 1349  468 1766 1468  421 1795  281  160  871  157 1895  839 1309 1815  212 1827
##    7    4    5    5    4    8    9    6    0    7    3    1    6    4    9    5
## 1719 1752 1946 1746 1252 1155 1945 1480 1418  594 1510 1975  840  473 1061  953
##    9    1    1    9    7    2    9    1    7    7    4    6    9    0    2    9
## 1462  909 1921   69  279  976  238 1068 1269  128 1960  274 1286  849 1707  264
##    4    2    6    1    2    1    7    4    1    6    5    0    0    8    9    2
## 1962 1101 1765  450  304  293  680 1588 1683  975  603 1866  972 1932 1664  772
##    6    7    0    8    9    0    7    7    9    0    5    2    3    1    3    4
##  329  784  733 1110 1893 1947  773 1550 1889  885 1485   45 1259  599 1251 1714
##    5    0    0    9    9    9    3    2    1    7    4    2    9    1    4    6
##  967  161  331 1087  438 1164  461  374  285  508  979  933  337  819 1865 1283
##    7    6    5    4    9    5    5    0    3    2    9    8    7    4    9    6
##  607 1603 1077 1680  499  646  857  907 1941  172 1817   61  520  817  631 1821
##    3    2    0    2    5    1    7    2    8    8    4    1    8    7    6    3
##  609 1478  455  805  868  230 1296  474  547 1733 1169 1157  208  801 1434 1191
##    1    0    8    4    3    6    4    2    9    4    0    7    2    5    1    8
##  726 1000   44  162 1202  470 1397 1687  223  372  686  628  357 1671 1179  354
##    2    4    4    9    6    9    8    0    8    1    5    0    9    0    8    2
## 1553 1694  637 1624   54  250  672 1475 1383  899 1506  815  675  480  314  738
##    3    7    2    9    9    1    0    0    4    6    1    8    8    2    7    5
## 1372 1049 1549 1670  811 1919   79 1835  436 1011 1607  851  308   19 1984  605
```

```
##    5    8    8    8    4    7    4    8    0    7    8    8    9    7    5    6
##  175  652  970   71
##    5    5    1    3
```

```r
#Mis-classification rates by class
tapply(predLm != ytest, ytest, mean)
```

```
##          0          1          2          3          4          5          6
## 0.00000000 0.00000000 0.00000000 0.00000000 0.00000000 0.00000000 0.00000000
##          7          8          9
## 0.02500000 0.00000000 0.02083333
```

```r
# Confusion Metric
table(predLm,ytest)
```

```
##       ytest
## predLm  0  1  2  3  4  5  6  7  8  9
##      0 42  0  0  0  0  0  0  0  0  0
##      1  0 44  0  0  0  0  0  1  0  0
##      2  0  0 44  0  0  0  0  0  0  0
##      3  0  0  0 32  0  0  0  0  0  0
##      4  0  0  0  0 44  0  0  0  0  0
##      5  0  0  0  0  0 42  0  0  0  0
##      6  0  0  0  0  0  0 34  0  0  0
##      7  0  0  0  0  0  0  0 39  0  1
##      8  0  0  0  0  0  0  0  0 34  0
##      9  0  0  0  0  0  0  0  0  0 47
```

We might think that a lot 2's, 6's and 9's are being mis-classified as one another (they do look similar in some ways). Looking at the confusion matricies we see that this is not quite the case.
