Jared Castaneda
Kurt Prutsman
Khoa Do

Homework 2

Due: Tuesday 10/15, 11:55pm on Titanium. Prepare your answers as a single PDF file.

1. Consider the “Auto MPG” which “concerns city-cycle fuel consumption in miles per gallon, to be predicted in terms of 3 multivalued discrete and 5 continuous attributes.”1 The goal is to model mpg given engine displacement and number of cylinders. Answer the following questions.
a. Load the autompg.csv file on Titanium and convert cylinders variable to a factor. (code, output of str())

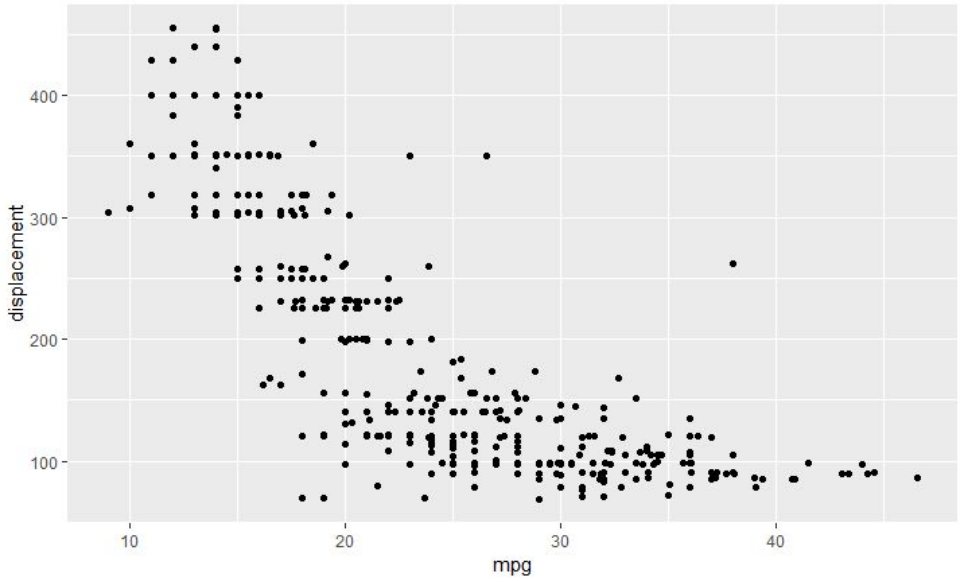
```
> cylinders_factor <- as.factor(autompg$cylinders)
> str(cylinders_factor)
 Factor w/ 5 levels "3","4","5","6",..: 5 5 5 5 5 5 5 5 5 5 ...
```

b. Which is the dependent variable? Which are the independent variables?

Cylinders is the independent variable and engine displacement is the dependent variable.

c. Plot mpg vs. displacement (code, plot)

```
>  ggplot(data=autompg) + geom_point(mapping = aes(x=mpg, y=displacement))
```



d. Create a linear model of mpg vs. displacement (only one independent variable). What is the R2? (code, output of summary(model), R2 value)

```
> lm(data=autompg, displacement~mpg)

Call:
lm(formula = displacement ~ mpg, data = autompg)

Coefficients:
(Intercept)          mpg
    445.70       -10.73

> summary(lm(data=autompg, displacement~mpg))

Call:
lm(formula = displacement ~ mpg, data = autompg)

Residuals:
    Min      1Q  Median      3Q     Max
-182.589  -37.575   -2.589   39.548  223.982

Coefficients:
             Estimate Std. Error t value Pr(>|t|)
(Intercept) 445.7029     9.8722   45.15   <2e-16 ***
mpg         -10.7285     0.3984  -26.93   <2e-16 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 62.05 on 396 degrees of freedom
Multiple R-squared:  0.6467,          Adjusted R-squared:  0.6459
F-statistic:   725 on 1 and 396 DF,  p-value: < 2.2e-16
```

e. Create a new transformed variable that is sqrt(displacement). Create a linear model of mpg vs. log(displacement).

```
> sqrtdisplacement <- sqrt(autompg$displacement)
```

```
> str(sqrtdisplacement)
 num [1:398] 17.5 18.7 17.8 17.4 17.4 …
> logdisplacement <- log(autompg$displacement)

> str(logdisplacement)
 num [1:398] 5.73 5.86 5.76 5.72 5.71 ...
> lm(data=autompg, mpg~logdisplacement)

> lm(data = autompg, logdisplacement~mpg)

Call:
lm(formula = logdisplacement ~ mpg, data = autompg)

Coefficients:
(Intercept)          mpg
    6.44725     -0.05631
```

i. Give R code, output of summary(model)

```
> model <- lm(data=autompg, mpg~logdisplacement)
> summary(model)

Call:
lm(formula = mpg ~ logdisplacement, data = autompg)

Residuals:
     Min       1Q   Median       3Q      Max
-16.1743  -2.5461  -0.4326   2.1949  19.9107

Coefficients:
                Estimate Std. Error t value Pr(>|t|)
(Intercept)      85.9507     2.1329   40.30   <2e-16 ***
logdisplacement -12.1870     0.4141  -29.43   <2e-16 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 4.384 on 396 degrees of freedom
Multiple R-squared:  0.6862,          Adjusted R-squared:  0.6854
F-statistic: 866.1 on 1 and 396 DF,  p-value: < 2.2e-16
```

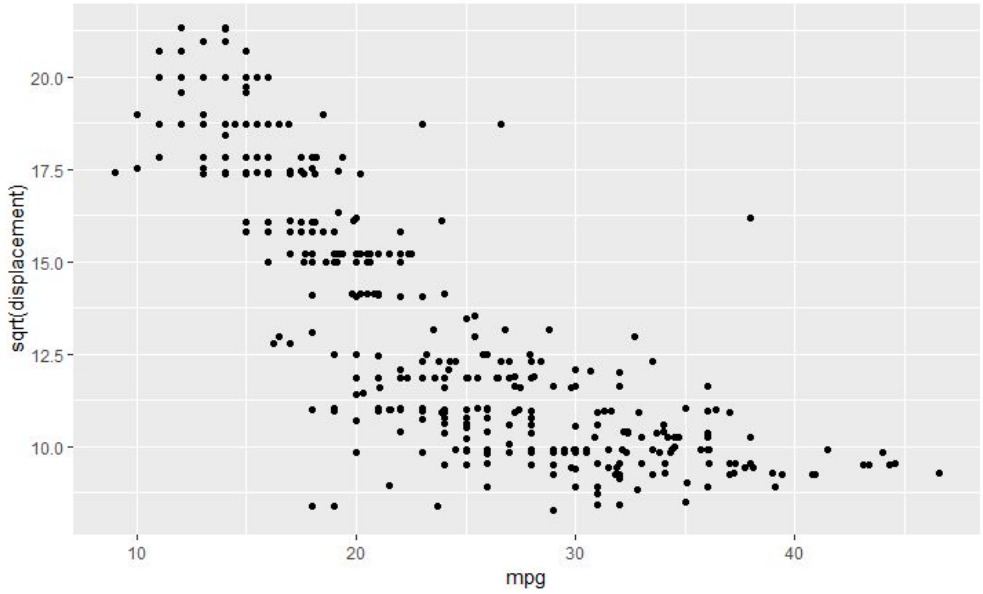
ii. Is this a better fit than in part (d)?

The R squared value from part (d) is 0.6467. The R squared value from part(i) is 0.6862. In general, the higher the R-squared, the better the model fits your data. **Therefore, part(i) is a better fit than part(d) because it has a higher R-squared.**

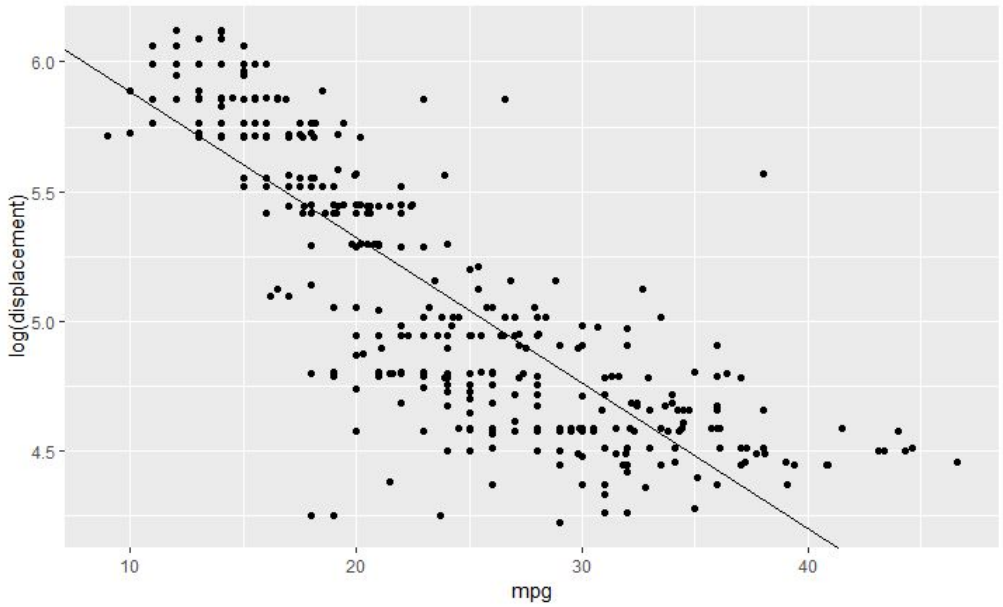
iii. Plot mpg vs. sqrt(displacement) and overlay the best fit model as a straight line. (code, plot)

The best fit model is **lm(data=autompg, mpg~logdisplacement)**

```
model <- lm(data=autompg, mpg~logdisplacement)
> ggplot(data=autompg) + geom_point(mapping=aes(x=mpg, y=log(displacement))) +
geom_abline(slope=model$coefficients[2], intercept = model$coefficients[1])
```



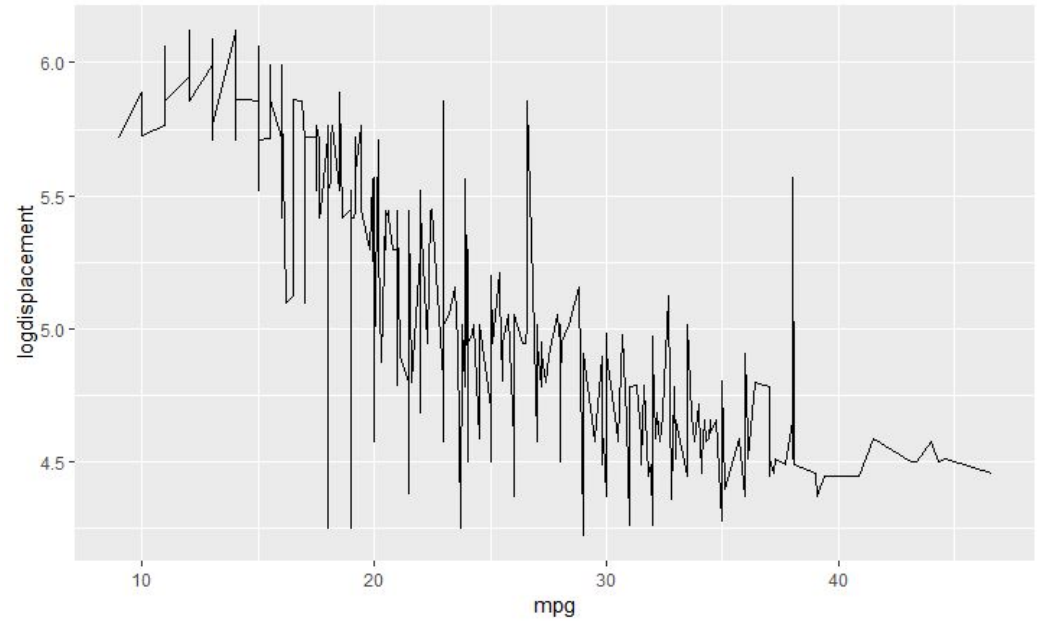

```
> ggplot(data=autompg) + geom_point(mapping=aes(x=mpg, y=log(displacement))) +
geom_abline(slope=model$coefficients[2], intercept = model$coefficients[1])
```



iv. Plot mpg vs. displacement and overlay the best fit model as a curve. (code, plot) [Hint: plot the predictions; use add_predictions() and geom_line(). You don't have to use data_grid()]

The best fit model is **lm(data=autompg, mpg~logdisplacement)**

```
> model <- lm(data=autompg, mpg~logdisplacement)
> autompgcopy <- autompg %>% add_predictions(model)
> ggplot(data = autompgcopy) + geom_line(mapping = aes(model))
```



f. Create a linear model of mpg vs. sqrt(displacement) and cylinders.

i. Give R code, output of summary(model)

```
modelF <- lm(data=autompg, mpg~sqrtdisplacement + cylinders_factor) #y vs x

summary(modelF)

Call:
lm(formula = mpg ~ sqrtdisplacement + cylinders_factor, data = autompg)

Residuals:
    Min      1Q  Median      3Q     Max
-10.3601  -2.6105  -0.2589   2.1420  20.7138

Coefficients:
                Estimate Std. Error t value Pr(>|t|)
(Intercept)       36.2894     2.7096  13.393  < 2e-16 ***
sqrtdisplacement  -1.8493     0.1956  -9.452  < 2e-16 ***
cylinders_factor4 12.2841     2.1909   5.607 3.89e-08 ***
cylinders_factor5 13.2524     3.3355   3.973 8.44e-05 ***
cylinders_factor6 10.9303     2.5032   4.367 1.62e-05 ***
cylinders_factor8 12.9472     2.9312   4.417 1.30e-05 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 4.275 on 392 degrees of freedom
Multiple R-squared:  0.7046,  Adjusted R-squared:  0.7008
F-statistic:   187 on 5 and 392 DF,  p-value: < 2.2e-16
```

ii. How many dummy (i.e., 0-1) variables were created in the model?

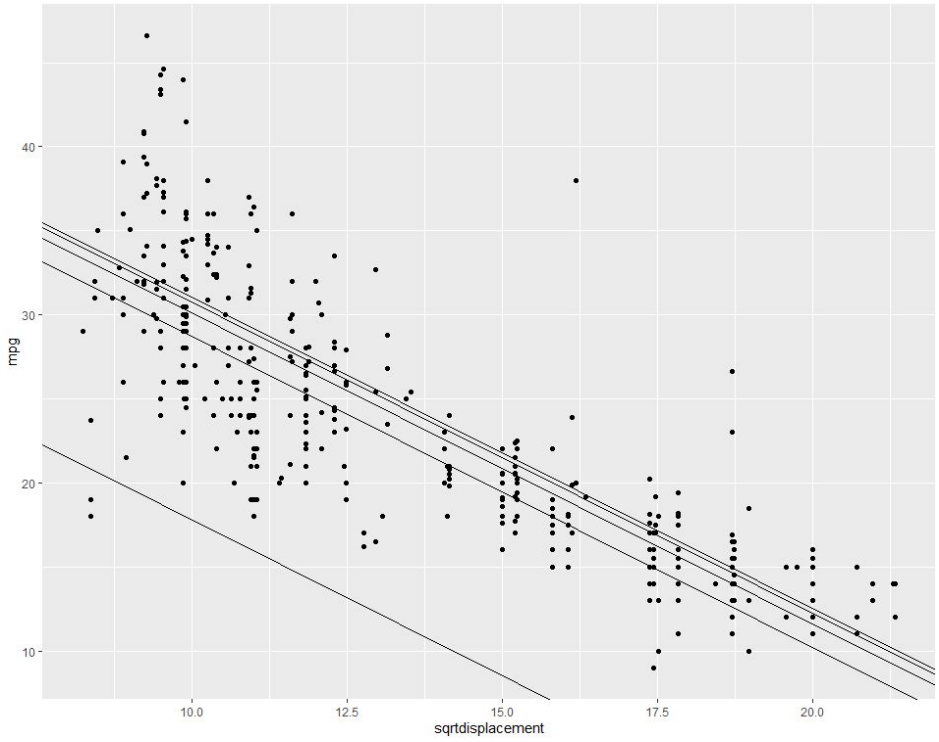
There were 6 dummy variables created.

iii. Is this a better fit than in part (e)?

Part e had an r-squared of .6862, while this has an r-squared of .7046. This is better.

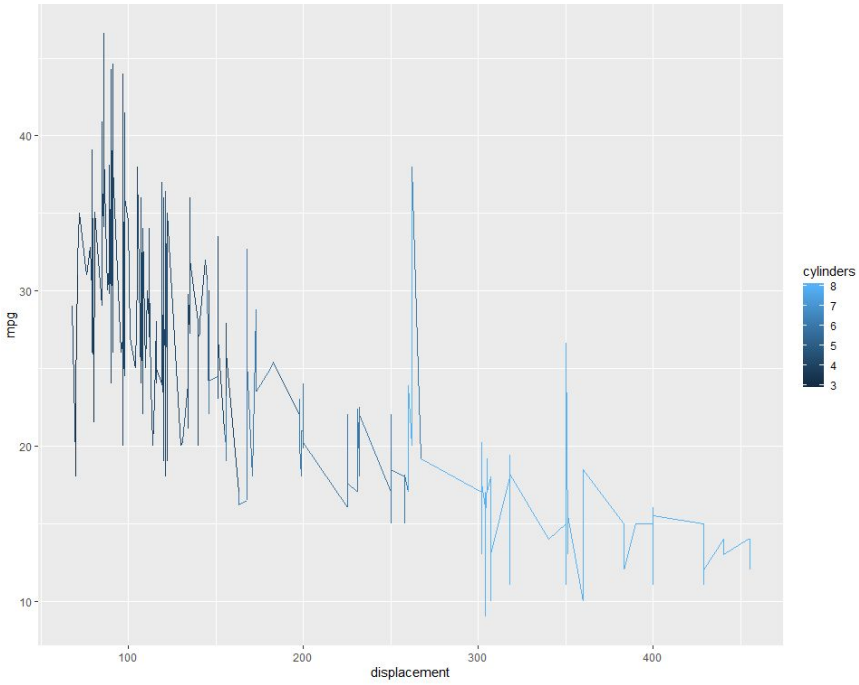
iv. Plot mpg vs. sqrt(displacement) and overlay the multiple linear fit lines:
one for each value of the discrete variable. (code, plot)

```
f <- modelF$coefficients
> ggplot(data = autompg) + geom_point(mapping=aes(x=sqrtdisplacement, y=mpg)) +
geom_abline(slope=f[2], intercept=f[1]) + geom_abline(slope=f[2], intercept=f[1] + f[3]) +
geom_abline(slope=f[2], intercept=f[1] + f[4]) + geom_abline(slope=f[2], intercept=f[1] + f[5]) +
geom_abline(slope=f[2], intercept=f[1] + f[6])
```



v. Plot mpg vs. displacement and overlay the best fit model as a curve. (code, plot) [Hint: plot the predictions; use add_predictions() and geom_line() and use the color aesthetic for cylinders]

ggplot(autompgcopy, aes(displacement, mpg)) + geom_line(aes(color=cylinders))



2. Consider the toy dataset below which shows if 4 subjects have diabetes or not, along with two diagnostic measurements. [This question is meant to be completed with a calculator; no need to write any R code.]

Preg	**BP**	**HasDiabetes**	**Preg.Norm**	**BP.Norm**
2	74	No		
3	58	Yes		
2	58	Yes		
1	54	No		
2	70	?		

a. Which variable is the “Class” variable?

HasDiabetes is the “Class” variable.

b. Normalize the Preg and BP values by scaling the minimum-maximum range of each column to 0-1. Fill in the empty columns in the table.
normalizeFunction = (x - min)/(max - min)

max(Preg) = 3
min(Preg) = 1

max(BP) = 74
min(BP) = 54

Preg
(2 - 1)/(3-1) = 0.5
(3-1)/(3-1) = 1
(2-1)/(3-1) = 0.5
(1-1)/(3-1) = 0
(2 - 1)/(3-1) = 0.5

BP
(74-54)/(74-54) = 1
(58-54)/(74-54) = 0.2
(58-54)/(74-54) = 0.2
(54-54)/(74-54) = 0
(70-54)/(74-54) = 0.8

Preg	BP	HasDiabetes	Preg.Norm	BP.Norm
2	74	No	0.5	1
3	58	Yes	1	0.2
2	58	Yes	0.5	0.2
1	54	No	0	0
2	70	?	0.5	0.8

c. Predict whether a subject with Preg=2, BP=70 will have diabetes using the 1-NN algorithm and

i. Using Euclidean distance on the original variables:

No. (Class of nearest neighbor (2,74); nearest distance=4)

ii. Using Manhattan distance on the original variables:

No. (Class of nearest neighbor (2,74); nearest distance=4)

iii. Using Euclidean distance on the normalized variables:

No. (Class of nearest neighbor (0.5,1); nearest distance=0.2)

iv. Using Manhattan distance on the normalized variables:

No. (Class of nearest neighbor (0.5,1); nearest distance=0.2)

3. The data_banknote_authentication.csv file attached on Titanium contains instances of genuine and forged banknotes. The first four columns are features calculated from an industrial camera2 ; the fifth column indicates if the banknote is forged or not. The goal is to see if it is possible to detect a forgery from only the features.

a. Load and pre-process the data. Show code to:

i. Load the data file on Titanium.

```
data <- read_csv("data_banknote_authentication.csv")
```

ii. How many rows and columns are there?

answer: 1372 rows and 5 columns.

```
> nrow(data)
[1] 1372
> ncol(data)
[1] 5
```

b. Split the dataset into train and test datasets with the rows 1, 3, 5, ... for training, and the remaining rows for test (i.e, test using rows 2, 4, 6, …). Do NOT randomly sample the data (though resampling is usually done, this hw problem does not use this step for ease of grading). (code)

code:

```
> library(dplyr)

# 1, 3, 5, ... for training
# odd
traindata <- data %>% dplyr::filter(row_number() %% 2 == 1)

# remaining rows for test (i.e, test using rows 2, 4, 6, .)
# even
testdata <- data %>% dplyr::filter(row_number() %% 2 == 0)
```

c. Train and test a k-nearest neighbor classifier with the above datasets. Consider only variance and skewness columns. Set k=1. What is the error rate (number of misclassifications)? (code)

code:

```
#only variance and skewness columns 1:2
colsToConsider <- 1:2
k <- 1
trainfeatures <- traindata[colsToConsider]
trainlabels <- factor(traindata$forged)

testfeatures <- testdata[colsToConsider]
testlabels <- factor(testdata$forged)

predictedlabels <- knn(train = trainfeatures, cl = trainlabels, test=testfeatures, k = k)

actualVsPredicted <- table(testlabels, predictedlabels)
actualVsPredicted

errorRate <- sum(actualVsPredicted) - sum(diag(actualVsPredicted))
```

output:

```
          predictedlabels
testlabels   0   1
         0 356  25
         1  15 290
```

What is the error rate (number of misclassifications)?

answer: The error rate = 25 + 15 = 40

This is the sum of the values in the table that are not on the diagonal.

	Predictedlabels	
Testlabels	**false = “0”**	**true = “1”**
false = “0”	356	25
true = “1”	15	290

d. Repeat part (c) but consider only variance , skewness, and curtosis columns. Set k=1. (show code.) What is the error rate? Will the error rate always decrease with larger number of parameters? Why or why not: answer in 2-3 sentences?

code:

```
#consider only variance , skewness, and curtosis columns 1:3
colsToConsider <- 1:3
k <- 1
trainfeatures <- traindata[colsToConsider]
trainlabels <- factor(traindata$forged)

testfeatures <- testdata[colsToConsider]
testlabels <- factor(testdata$forged)

predictedlabels <- knn(train = trainfeatures, cl = trainlabels, test=testfeatures, k = k)

actualVsPredicted <- table(testlabels, predictedlabels)
actualVsPredicted

errorRate <- sum(actualVsPredicted) - sum(diag(actualVsPredicted))
```

output:

```
          predictedlabels
testlabels   0   1
         0 379   2
         1   0 305
```

What is the error rate (number of misclassifications)?

answer: The error rate = 2 + 0 = 2

This is the sum of the values in the table that are not on the diagonal.

	Predictedlabels	
Testlabels	**false = "0"**	**true = "1"**
false = "0"	379	2
true = "1"	0	305

Will the error rate always decrease with larger number of parameters?
answer: No

Why or why not: answer in 2-3 sentences?
answer: (TODO: check for better answer)
Because certain features may not contribute to the prediction. In fact, the excess features may contribute to overfitting which may cause inaccurate predictions.

e. Repeat part (d) but set k=5. What is the error rate?

code:

```
#consider only variance , skewness, and curtosis columns 1:3
colsToConsider <- 1:3
k <- 5
trainfeatures <- traindata[colsToConsider]
trainlabels <- factor(traindata$forged)

testfeatures <- testdata[colsToConsider]
testlabels <- factor(testdata$forged)

predictedlabels <- knn(train = trainfeatures, cl = trainlabels, test=testfeatures, k = k)

actualVsPredicted <- table(testlabels, predictedlabels)
actualVsPredicted

errorRate <- sum(actualVsPredicted) - sum(diag(actualVsPredicted))
```

output:

```
          predictedlabels
testlabels   0   1
         0 377   4
         1   2 303
```

What is the error rate (number of misclassifications)?
answer: The error rate = 4 + 2 = 6
This is the sum of the values in the table that are not on the diagonal.

	Predictedlabels	
Testlabels	**false = "0"**	**true = "1"**
false = "0"	377	4
true = "1"	2	303

f. Repeat part (e) but set k=11. What is the error rate? Considering your observations from (d)-(f), which is the best value for k?

code:

```
#consider only variance , skewness, and curtosis columns 1:3
colsToConsider <- 1:3
k <- 11
trainfeatures <- traindata[colsToConsider]
trainlabels <- factor(traindata$forged)

testfeatures <- testdata[colsToConsider]
testlabels <- factor(testdata$forged)

predictedlabels <- knn(train = trainfeatures, cl = trainlabels, test=testfeatures, k = k)

actualVsPredicted <- table(testlabels, predictedlabels)
actualVsPredicted

errorRate <- sum(actualVsPredicted) - sum(diag(actualVsPredicted))
```

output:

```
          predictedlabels
testlabels   0   1
         0 377   4
         1   1 304
```

What is the error rate (number of misclassifications)?
answer: The error rate = 4 + 1 = 5
This is the sum of the values in the table that are not on the diagonal.

	Predictedlabels	
Testlabels	**false = "0"**	**true = "1"**
false = "0"	377	4
true = "1"	1	304

Considering your observations from (d)-(f), which is the best value for k?
answer: The best value for k is 1

g. Consider only the ranges of the features - is normalization required?
answer: Yes

h. Normalize each column by scaling the minimum-maximum range of each column to 0-1. (Hint: the built-in R function scale() can be used for this) (code)

code:

```
colsToConsider <- 1:3
trainfeatures <- traindata[colsToConsider]
trainlabels <- factor(traindata$forged)

testfeatures <- testdata[colsToConsider]
testlabels <- factor(testdata$forged)

trainfeatures$variance.scaled <- scale(trainfeatures$variance)
trainfeatures$skewness.scaled <- scale(trainfeatures$skewness)
trainfeatures$curtosis.scaled <- scale(trainfeatures$curtosis)

testfeatures$variance.scaled <- scale(testfeatures$variance)
testfeatures$skewness.scaled <- scale(testfeatures$skewness)
testfeatures$curtosis.scaled <- scale(testfeatures$curtosis)

# adjust the cols to use the scaled columns
colsToConsider <- 4:6
trainfeatures <- trainfeatures[colsToConsider]
testfeatures <- testfeatures[colsToConsider]
```

i. Train and test a k-nearest neighbor classifier with the normalized dataset. Consider only variance, skewness, and curtosis columns. Set k=1. What is the error rate?

code:

```
k <- 1
predictedlabels <- knn(train = trainfeatures, cl = trainlabels, test=testfeatures, k = k)

actualVsPredicted <- table(testlabels, predictedlabels)
actualVsPredicted

errorRate <- sum(actualVsPredicted) - sum(diag(actualVsPredicted))
```

output:

```
          predictedlabels
testlabels   0   1
         0 380   1
         1   0 305
```

What is the error rate (number of misclassifications)?
answer: The error rate = 1 + 0 = 1
This is the sum of the values in the table that are not on the diagonal.

	Predictedlabels	
Testlabels	**false = "0"**	**true = "1"**
false = "0"	380	1
true = "1"	0	305