Homework #5

1. In this problem, we will make use of the Auto data set, which is part of the ISLR2 package.

Instructions:

You may discuss the homework problems in small groups, but you must write up the final solutions and code yourself. Please turn in your code for the problems that involve coding. However, code without written answers will receive no credit. To receive credit, you must explain your answers and show your work. All plots should be appropriately labeled and legible, with axis labels, legends, etc., as needed.

On this assignment, some of the problems involve random number generation. Be sure to set a random seed (using the command `set.seed()`*) before you begin.*

1. Suppose we have an $n \times p$ data matrix X, and a continuous-valued response $y \in \mathbb{R}^n$. We saw in lecture that the mth principal component score vector is a linear combination of the p features, of the form

$$z_{im} = \phi_{1m} x_{i1} + \phi_{2m} x_{i2} + ... + \phi_{pm} x_{ip} \tag{1}$$

(e.g. see (12.2) and (12.4) in textbook). In principal components regression, we fit a linear model to predict y, but instead of using the columns of X as predictors, we use the first M principal component score vectors, where $M < p$.

A note before you begin: In this problem, I will ask you to "write out an expression for a linear model." For instance, if I asked you to write out an expression for a linear model to predict an n-vector y using the columns of an $n \times p$ matrix X, then here's what I'd want to see: $y_i = \beta_0 + \beta_1 x_{i1} + ... + \beta_p x_{ip} + \epsilon_i$, where ϵ_i is a mean-zero noise term.

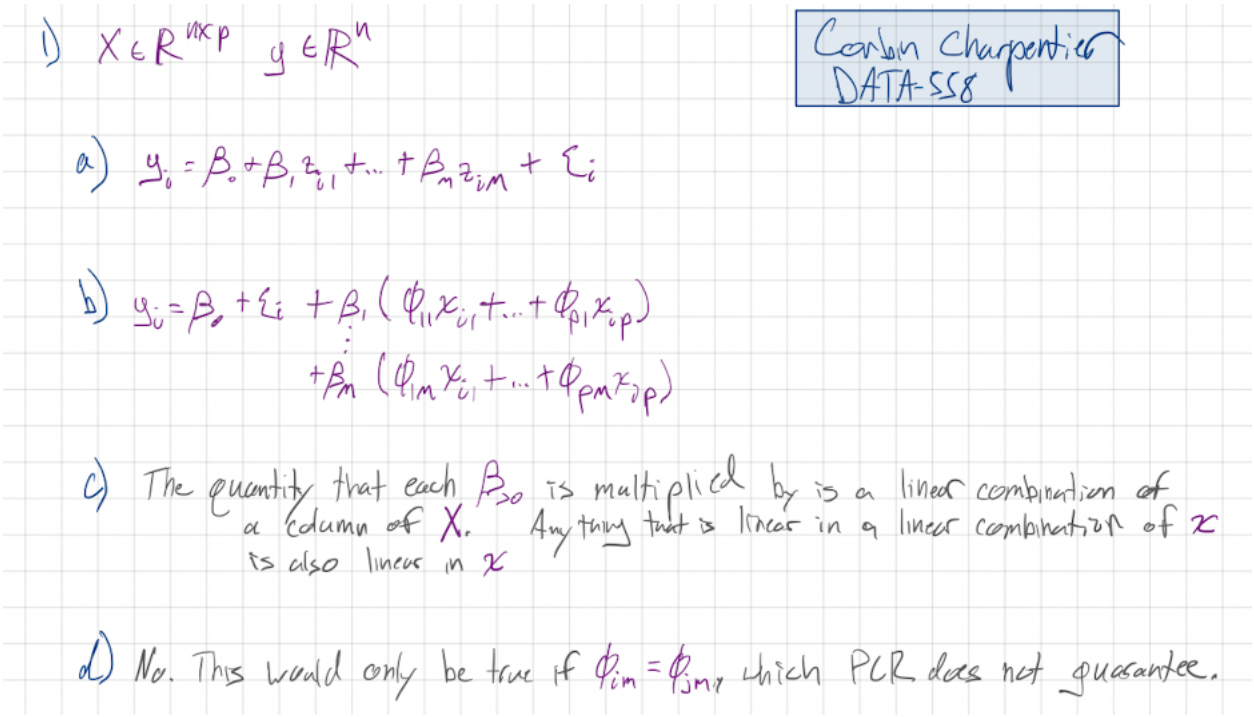


Figure 1: Problem 1

2. We saw in class that K-means clustering minimizes the within-cluster sum of squares, given in (12.17) of the textbook. We can better understand the meaning of the within-cluster sum of squares by looking at (12.18) of the textbook. This shows us that the within-cluster sum of squares is (up to a scaling by a factor of two) the sum of squared distances from each observation to its cluster centroid.

(a) Show *computationally* that (12.18) holds. You can do this by repeating this procedure a whole bunch of times:

- Simulate an $n \times p$ data matrix, as well as some clusters $C_1, ..., C_K$. (It doesn't matter whether there are any "true clusters" in your data, nor whether $C_1, ..., C_K$ correspond to these true clusters — (12.18) is a mathematical identity that should hold no matter what.)
- Compute the left-hand side of (12.18) on this data.
- Compute the right-hand side of (12.18) on this data.
- Verify that the left- and right-hand sides are equal. (If they aren't, then you have done something wrong!)

```
n <- 1000
p <- 100
k <- 10
c_size <- n/k

X <- matrix(rnorm(n*p), ncol=p)

left_sum <- 0
right_sum <- 0

for(i_start in seq(1, n, by=c_size)) { # move across the rows by partition size
    i_end <- c_size + i_start - 1
```

```
    # left sum stuff
    for(i_l in i_start:(i_end - 1)) { # left index
        for(i_r in (i_l + 1):i_end) { # right index
            left_sum <- left_sum + sum((X[i_l,] - X[i_r,])^2) / c_size
        }
    }

    # right sum stuff
    for(j in 1:p) { # partition-element index
        x_bar <- mean(X[(i_start:i_end), j])
        for(i in i_start:i_end) { # left index
            # Note: we don't multiply this quantity by two because we single
            #       count each combination of i and i_prime above.
            right_sum <- right_sum + ((X[i,j] - x_bar)^2)
        }
    }
}

print(left_sum)
```

```
## [1] 98723.54
```

```
print(right_sum)
```

```
## [1] 98723.54
```

(b) *Extra Credit*: Show *analytically* that (12.18) holds. In other words, use algebra to prove (12.18).

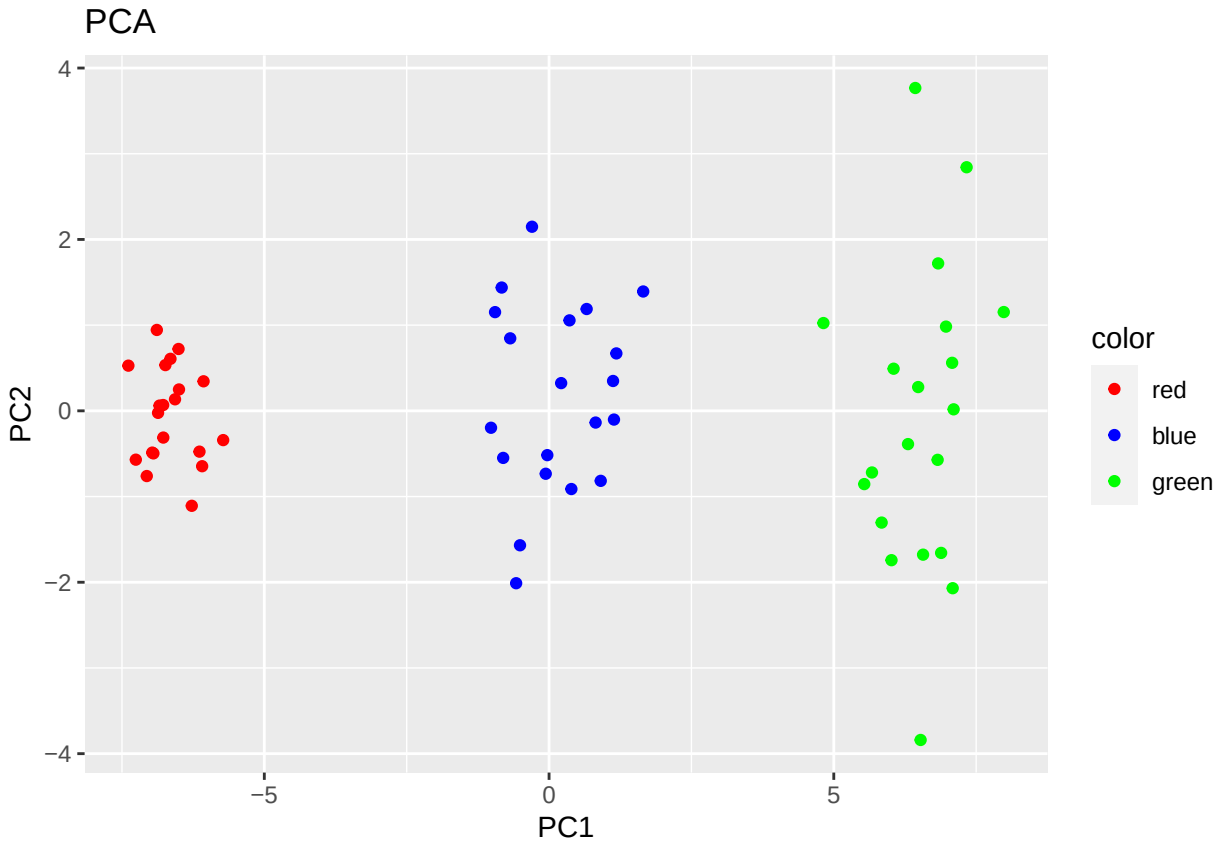
3. In this problem, you will generate simulated data, and then perform PCA and K-means clustering on the data.

(a) Generate a simulated data set with 20 observations in each of three classes (i.e. 60 observations total), and 50 variables.

```
generate_q3_data <- function() {
    gen_data <- function(color, mean, sd=1) {
        colord <- rep(color, 20)
        df <- data.frame(colord, rnorm(20, mean, sd))

        for(i in 2:50) {
            df <- cbind(df, variable=rnorm(20, mean, sd))
        }
        return(df)
    }

    rbind(
       gen_data("red", 0, 0.75),
       gen_data("blue", 1.5, 1),
       gen_data("green", 3, 1.25)
    )
}

df <- generate_q3_data()
```

(b) Perform PCA on the 60 observations and plot the first two principal component score vectors. Use a different color to indicate the observations in each of the three classes. If the three classes appear separated in this plot, then continue on to part (c). If not, then return to part (a) and modify the simulation so that there is greater separation between the three classes. Do not continue to part (c) until the three classes show at least some separation in the first two principal component score vectors.

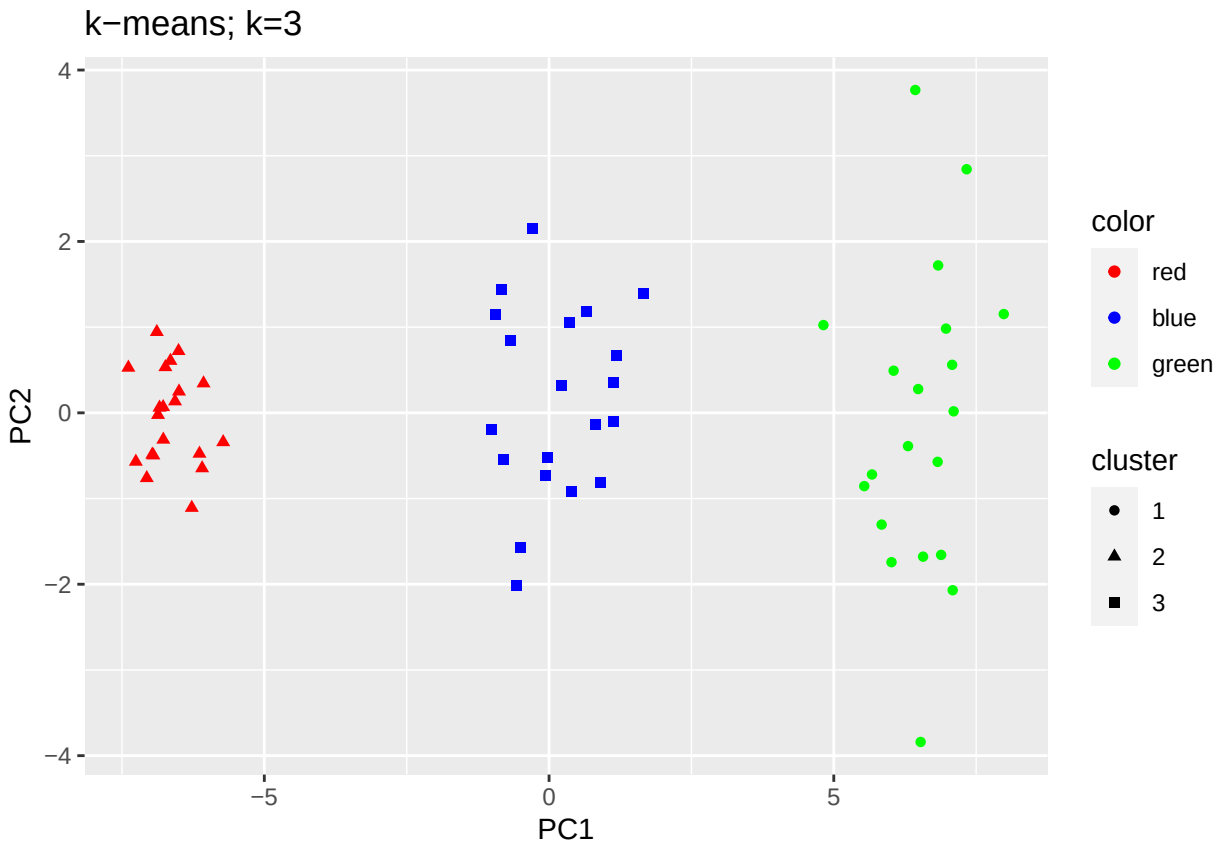
```
pcs <- prcomp(df[-1], scale=TRUE)

df_pcs <- cbind(as.data.frame(pcs$x), color=df$colord)

qplot(PC1, PC2, main="PCA", color=color, data=df_pcs) +
    scale_color_manual(values=c("red"="red",
                                "blue"="blue", "green"="green"))
```



(c) Perform K-means clustering of the observations with $K = 3$. How well do the clusters that you obtained in K-means clustering compare to the true class labels?

Hint: You can use the `table` *function in* `R` *to compare the true class labels to the class labels obtained by clustering. Be careful how you interpret the results: K-means clustering will arbitrarily number the clusters, so you cannot simply check whether the true class labels and clustering labels are the same.*

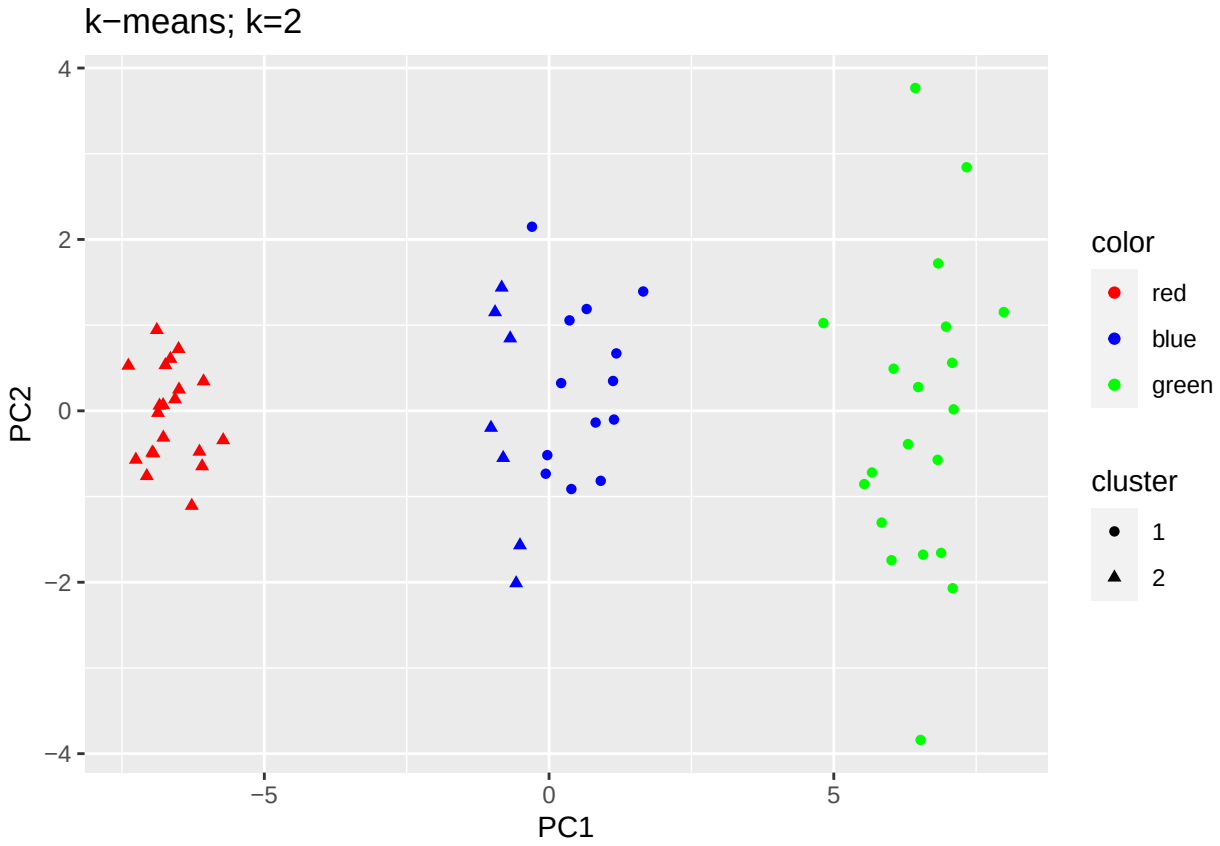
```
set.seed(1)
km.out <- kmeans(df[-1], 3)

qplot(PC1, PC2, main="k-means; k=3", color=color, shape=cluster,
      data=cbind(
          df_pcs,
          cluster=factor(km.out$cluster))) +
    scale_color_manual(values=c("red"="red",
                                "blue"="blue",
                                "green"="green"))
```



K-means performed perfectly.

(d) Perform K-means clustering with $K = 2$. Describe your results.

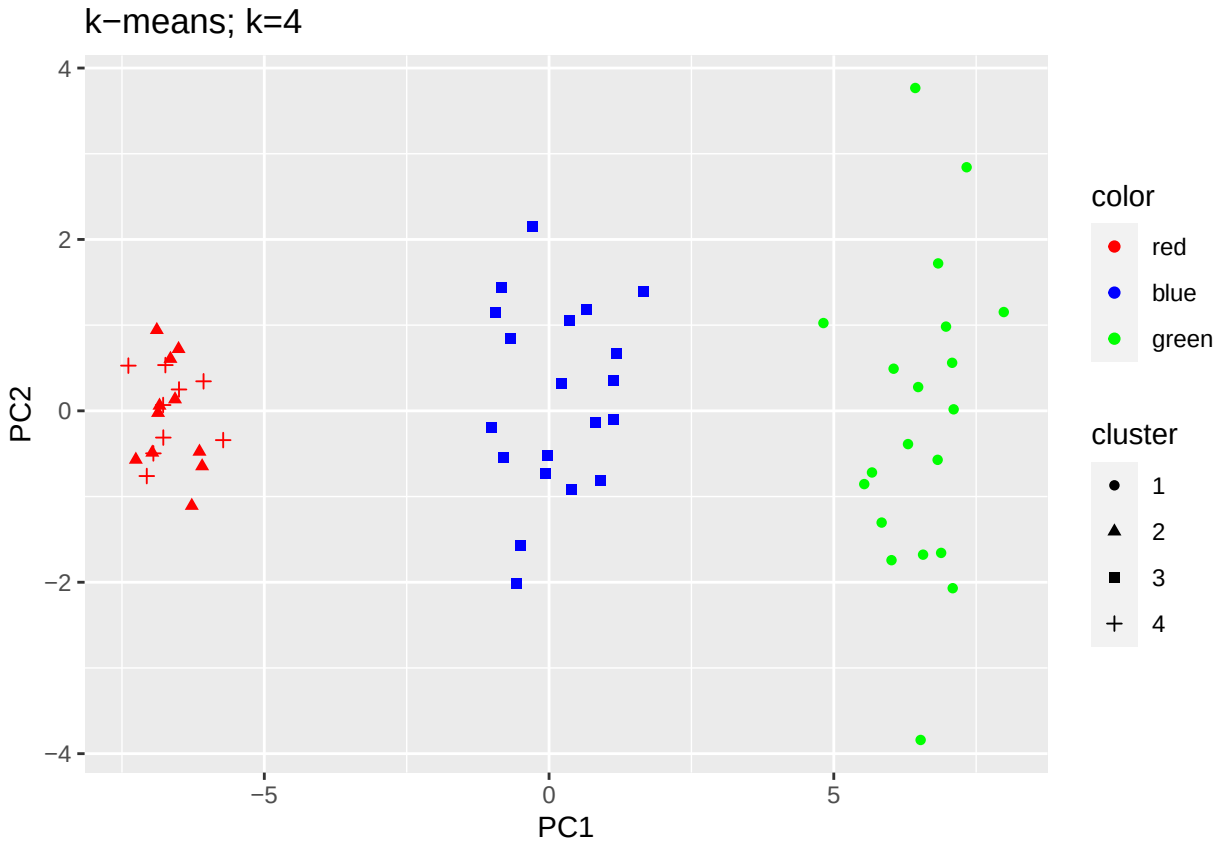
```
set.seed(1)
km.out <- kmeans(df[-1], 2)

qplot(PC1, PC2, main="k-means; k=2", color=color, shape=cluster,
      data=cbind(
          df_pcs,
          cluster=factor(km.out$cluster))) +
    scale_color_manual(values=c("red"="red",
                                "blue"="blue",
                                "green"="green"))
```



The clustering algorithm absorbed the middle class, blue, into one of the outer clusters. The color that blue ultimated associated with depends on initial cluster assignments, which is random.

(e) Now perform K-means clustering with $K = 4$, and describe your results.

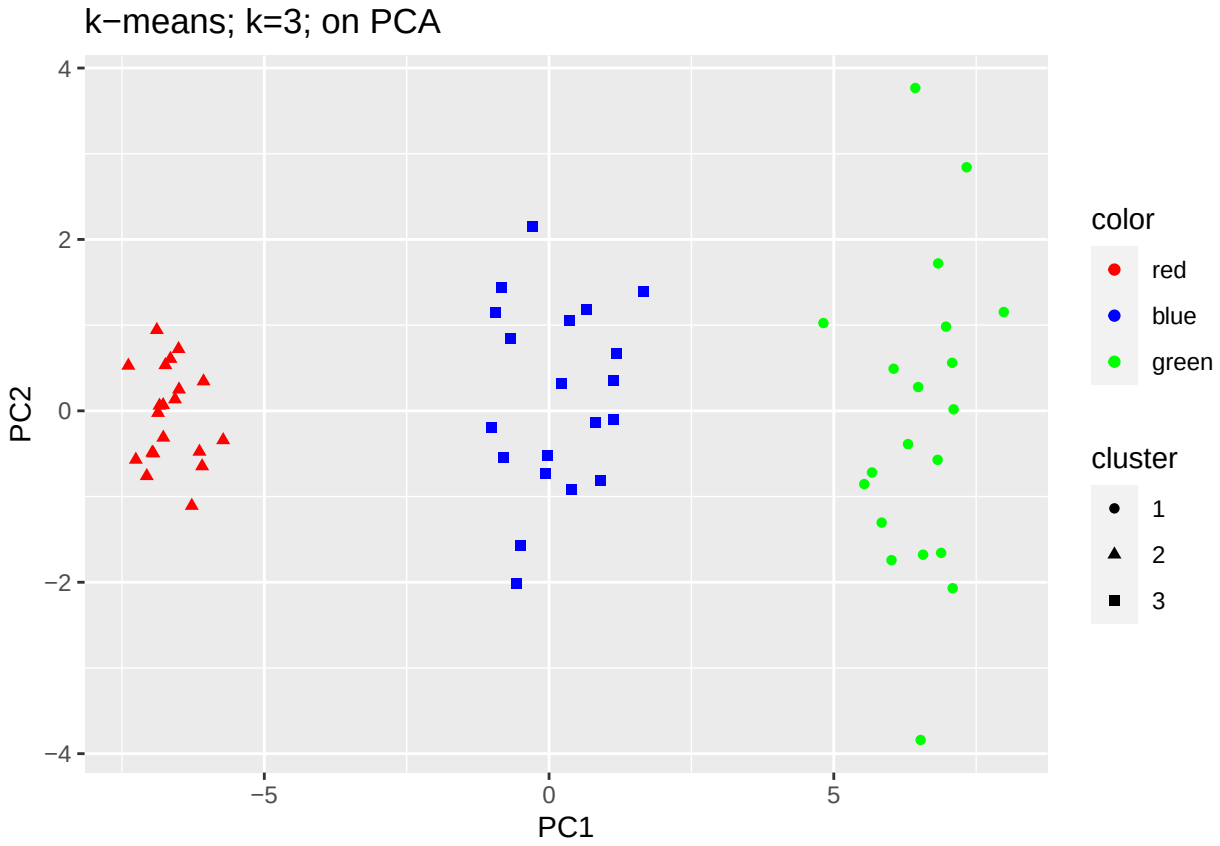
```
set.seed(1)
km.out <- kmeans(df[-1], 4)

qplot(PC1, PC2, main="k-means; k=4", color=color, shape=cluster,
      data=cbind(
          df_pcs,
          cluster=factor(km.out$cluster))) +
    scale_color_manual(values=c("red"="red",
                                "blue"="blue",
                                "green"="green"))
```



Since the clustering algorithm must partition the data into 4 clusters, one of the three clusters is divided into two. The space between the original classes is large enough that k-means will not span cluster assignments across classes.

(f) Now perform K-means clustering with $K = 3$ on the first two principal component score vectors, rather than on the raw data. That is, perform K-means clustering on the 60 $\times$ 2 matrix of which the first column is the first principal component score vector, and the second column is the second principal component score vector. Comment on the results.

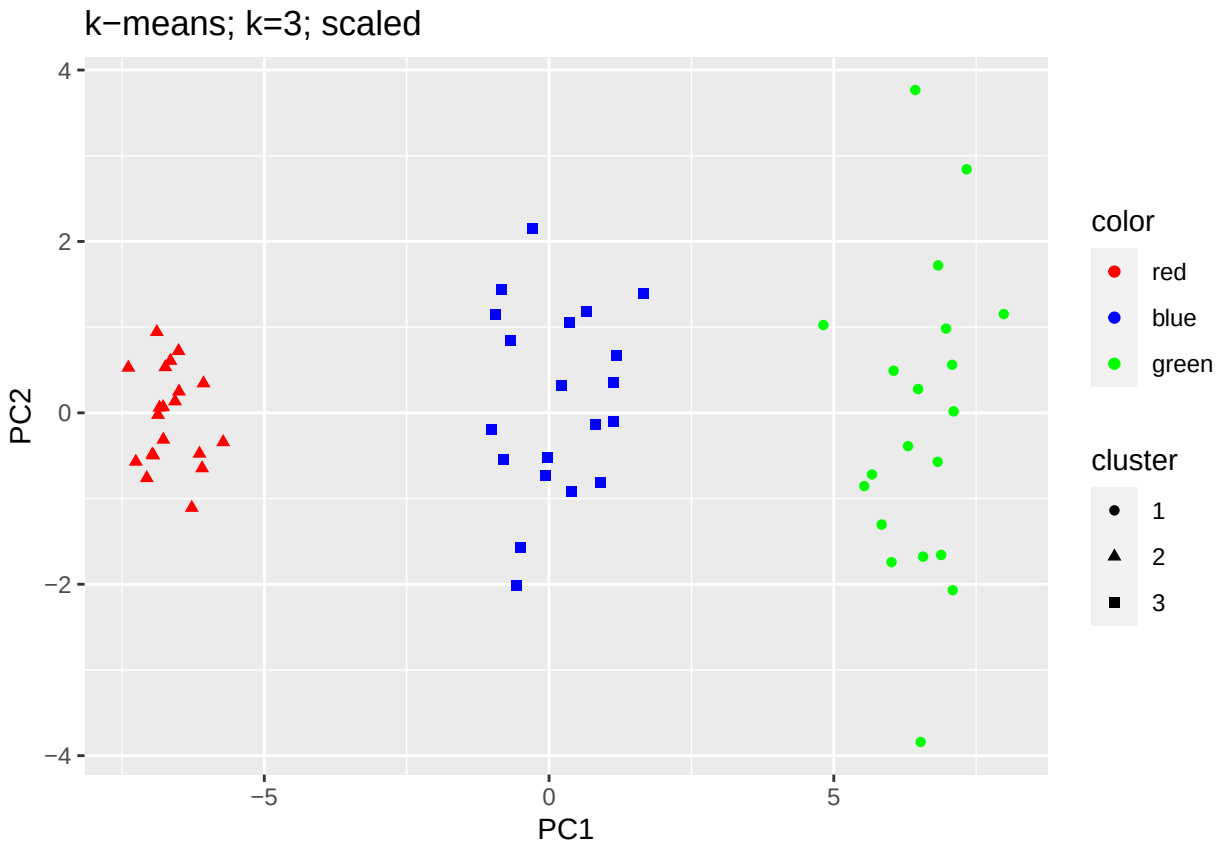
```
set.seed(1)
km.out <- kmeans(df_pcs[c(1,2)], 3)

qplot(PC1, PC2, main="k-means; k=3; on PCA", color=color, shape=cluster,
      data=cbind(
          df_pcs,
          cluster=factor(km.out$cluster))) +
    scale_color_manual(values=c("red"="red",
                                "blue"="blue",
                                "green"="green"))
```



The results are the same. The original k-means 3 way colustering performed without error, so it's not surprising that applying it to the PCA-applied data, data that is more cleanly distinguished by class, did not cause regress.

(g) Using the scale function, perform K-means clustering with $K = 3$ on the data *after scaling each variable to have standard deviation one*. How do these results compare to those obtained in (b)? Explain

```
# TODO: scale
set.seed(1)
scaled_xs <- scale(df[-1])
km.out <- kmeans(scaled_xs, 3)

qplot(PC1, PC2, main="k-means; k=3; scaled", color=color, shape=cluster,
      data=cbind(
          df_pcs,
          cluster=factor(km.out$cluster))) +
    scale_color_manual(values=c("red"="red",
                                "blue"="blue",
                                "green"="green"))
```



For this particular data, the results do not change. In the general case, scaling is important when continuous variables are represented at different orders of magnitude, especially for some notions of "distance", like euclidian distance, where the larger scaled variables would exert more influence on the distance.

4. This problem involves the `OJ` data set, which is part of the `ISLR2` package.

(a) Create a training set containing a random sample of 800 observations, and a test set containing the remaining observations.

```
library(ISLR2)
library(dplyr)

splitData <- function(df, trainProportion) {
    df <- df %>% mutate(id = row_number())

    set.seed(1)
    train <- sample_frac(df, trainProportion)
    test <- anti_join(df, train, by='id')

    df <- df %>% mutate(id=NULL)

    return(list(train=train, test=test))
}

split <- splitData(OJ, 800/nrow(OJ))
```

(b) Fit a support vector classifier to the training data using `cost` $= 0.01$, with `Purchase` as the response and the other variables as predictors. Use the `summary()` function to produce summary statistics, and describe the results obtained.

```
library(e1071)
smodel <- svm(Purchase ~ ., data=split$train, cost=0.01, kernel="linear")
summary(smodel)
```

```
##
## Call:
## svm(formula = Purchase ~ ., data = split$train, cost = 0.01, kernel = "linear")
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  linear
##        cost:  0.01
##
## Number of Support Vectors:  433
##
##  ( 218 215 )
##
##
## Number of Classes:  2
##
## Levels:
##  CH MM
```

```
nrow(split$train)
```

```
## [1] 800
```

Since cost is low, the margins will be very wide, most of the data will lie on or within the decision margin

(about 75% of it in this case). Additionally, since we use a linear kernel (making the SVM a support vector classifier), the model is less flexible than it could be.

(c) What are the training and test error rates?

```
library(caret)
preds_train <- predict(smodel, split$train)
confusionMatrix(preds_train, split$train$Purchase)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  CH  MM
##         CH 422  76
##         MM  63 239
##
##                Accuracy : 0.8262
##                  95% CI : (0.7982, 0.8519)
##     No Information Rate : 0.6062
##     P-Value [Acc > NIR] : <2e-16
##
##                   Kappa : 0.6334
##
##  Mcnemar's Test P-Value : 0.3088
##
##             Sensitivity : 0.8701
##             Specificity : 0.7587
##          Pos Pred Value : 0.8474
##          Neg Pred Value : 0.7914
##              Prevalence : 0.6062
##          Detection Rate : 0.5275
##    Detection Prevalence : 0.6225
##       Balanced Accuracy : 0.8144
##
##        'Positive' Class : CH
##
```

```
preds_test <- predict(smodel, split$test)
confusionMatrix(preds_test, split$test$Purchase)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  CH  MM
##         CH 151  35
##         MM  17  67
##
##                Accuracy : 0.8074
##                  95% CI : (0.7552, 0.8527)
##     No Information Rate : 0.6222
##     P-Value [Acc > NIR] : 3.54e-11
##
##                   Kappa : 0.5756
##
##  Mcnemar's Test P-Value : 0.0184
```

```
##
##             Sensitivity : 0.8988
##             Specificity : 0.6569
##          Pos Pred Value : 0.8118
##          Neg Pred Value : 0.7976
##              Prevalence : 0.6222
##          Detection Rate : 0.5593
##    Detection Prevalence : 0.6889
##       Balanced Accuracy : 0.7778
##
##        'Positive' Class : CH
##
```

These results are better than I expected given the number of support vectors.

(d) Use the `tune()` function to select an optimal cost. Consider values in the range 0.01 to 10.

```
set.seed(1)
tsmodel <- tune(svm, Purchase ~ ., data=split$train, kernel="linear",
                ranges = list(cost=c(0.01, 0.05, 0.1, 0.5, 1, 5, 10))
                )

summary(tsmodel)
```

```
##
## Parameter tuning of 'svm':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##  cost
##   0.1
##
## - best performance: 0.17375
##
## - Detailed performance results:
##    cost   error dispersion
## 1  0.01 0.17500 0.03435921
## 2  0.05 0.17750 0.02813657
## 3  0.10 0.17375 0.02972676
## 4  0.50 0.17375 0.02664713
## 5  1.00 0.17500 0.02500000
## 6  5.00 0.17625 0.02729087
## 7 10.00 0.18125 0.03019037
```

Cost of 0.10 gave the best results on the training data.

(e) Compute the training and test error rates using this new value for `cost`.

```
smodel <- svm(Purchase ~ ., data=split$train, cost=0.1, kernel="linear")

preds_train <- predict(smodel, split$train)
confusionMatrix(preds_train, split$train$Purchase)
```

```
## Confusion Matrix and Statistics
```

```
##
##           Reference
## Prediction  CH  MM
##         CH 424  71
##         MM  61 244
##
##                Accuracy : 0.835
##                  95% CI : (0.8074, 0.8601)
##     No Information Rate : 0.6062
##     P-Value [Acc > NIR] : <2e-16
##
##                   Kappa : 0.6525
##
##  Mcnemar's Test P-Value : 0.4334
##
##             Sensitivity : 0.8742
##             Specificity : 0.7746
##          Pos Pred Value : 0.8566
##          Neg Pred Value : 0.8000
##              Prevalence : 0.6062
##          Detection Rate : 0.5300
##    Detection Prevalence : 0.6188
##       Balanced Accuracy : 0.8244
##
##        'Positive' Class : CH
##
```

```
preds_test <- predict(smodel, split$test)
confusionMatrix(preds_test, split$test$Purchase)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  CH  MM
##         CH 154  31
##         MM  14  71
##
##                Accuracy : 0.8333
##                  95% CI : (0.7834, 0.8758)
##     No Information Rate : 0.6222
##     P-Value [Acc > NIR] : 2.685e-14
##
##                   Kappa : 0.6335
##
##  Mcnemar's Test P-Value : 0.01707
##
##             Sensitivity : 0.9167
##             Specificity : 0.6961
##          Pos Pred Value : 0.8324
##          Neg Pred Value : 0.8353
##              Prevalence : 0.6222
##          Detection Rate : 0.5704
##    Detection Prevalence : 0.6852
##       Balanced Accuracy : 0.8064
##
```

```
##        'Positive' Class : CH
##
```

This seemed to slightly increase the generalizability of the model, since it performed better on the test data.

(f) Repeat parts (b) through (e) using a support vector machine with a radial kernel. Use the default value for gamma.

```
set.seed(1)
tsmodel <- tune(svm, Purchase ~ ., data=split$train, kernel="radial",
                ranges = list(cost=c(0.01, 0.05, 0.1, 0.5, 1, 5, 10))
                )

summary(tsmodel)
```

```
##
## Parameter tuning of 'svm':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##  cost
##   0.5
##
## - best performance: 0.17375
##
## - Detailed performance results:
##    cost   error dispersion
## 1  0.01 0.39375 0.04007372
## 2  0.05 0.21125 0.03458584
## 3  0.10 0.19125 0.02949223
## 4  0.50 0.17375 0.02664713
## 5  1.00 0.17875 0.02360703
## 6  5.00 0.18875 0.02729087
## 7 10.00 0.19500 0.03184162
```

```
smodel <- svm(Purchase ~ ., data=split$train, cost=0.5, kernel="radial")

preds_train <- predict(smodel, split$train)
confusionMatrix(preds_train, split$train$Purchase)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  CH  MM
##         CH 438  71
##         MM  47 244
##
##                Accuracy : 0.8525
##                  95% CI : (0.826, 0.8764)
##     No Information Rate : 0.6062
##     P-Value [Acc > NIR] : < 2e-16
##
##                   Kappa : 0.6869
##
```

```
##  Mcnemar's Test P-Value : 0.03423
##
##             Sensitivity : 0.9031
##             Specificity : 0.7746
##          Pos Pred Value : 0.8605
##          Neg Pred Value : 0.8385
##              Prevalence : 0.6062
##          Detection Rate : 0.5475
##    Detection Prevalence : 0.6362
##       Balanced Accuracy : 0.8388
##
##        'Positive' Class : CH
##
```

```
preds_test <- predict(smodel, split$test)
confusionMatrix(preds_test, split$test$Purchase)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  CH  MM
##         CH 151  34
##         MM  17  68
##
##                Accuracy : 0.8111
##                  95% CI : (0.7592, 0.856)
##     No Information Rate : 0.6222
##     P-Value [Acc > NIR] : 1.364e-11
##
##                   Kappa : 0.5846
##
##  Mcnemar's Test P-Value : 0.02506
##
##             Sensitivity : 0.8988
##             Specificity : 0.6667
##          Pos Pred Value : 0.8162
##          Neg Pred Value : 0.8000
##              Prevalence : 0.6222
##          Detection Rate : 0.5593
##    Detection Prevalence : 0.6852
##       Balanced Accuracy : 0.7827
##
##        'Positive' Class : CH
##
```

This model performs very similarly but slight worse than the linear kernel model. We see higher accuracy on the training set and lower accuracy on the testing set, implying the model could be too flexible.

(g) Repeat parts (b) through (e) using a support vector machine with a polynomial kernel. Set degree = 2**.**

```
set.seed(1)
tsmodel <- tune(svm, Purchase ~ ., data=split$train, kernel="polynomial", degree=2,
                ranges = list(cost=c(0.01, 0.05, 0.1, 0.5, 1, 5, 10))
                )
```

```r
summary(tsmodel)
```

```
##
## Parameter tuning of 'svm':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##  cost
##     5
##
## - best performance: 0.18625
##
## - Detailed performance results:
##    cost   error dispersion
## 1  0.01 0.39375 0.04007372
## 2  0.05 0.35000 0.03818813
## 3  0.10 0.32875 0.05337563
## 4  0.50 0.19625 0.03230175
## 5  1.00 0.20000 0.03679900
## 6  5.00 0.18625 0.02729087
## 7 10.00 0.18875 0.03197764
```

```r
smodel <- svm(Purchase ~ ., data=split$train, cost=0.5, kernel="polynomial", degre=2)

preds_train <- predict(smodel, split$train)
confusionMatrix(preds_train, split$train$Purchase)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  CH  MM
##         CH 446 110
##         MM  39 205
##
##                Accuracy : 0.8138
##                  95% CI : (0.785, 0.8402)
##     No Information Rate : 0.6062
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.5938
##
##  Mcnemar's Test P-Value : 9.773e-09
##
##             Sensitivity : 0.9196
##             Specificity : 0.6508
##          Pos Pred Value : 0.8022
##          Neg Pred Value : 0.8402
##              Prevalence : 0.6062
##          Detection Rate : 0.5575
##    Detection Prevalence : 0.6950
##       Balanced Accuracy : 0.7852
##
##        'Positive' Class : CH
```

```
##
```

```
preds_test <- predict(smodel, split$test)
confusionMatrix(preds_test, split$test$Purchase)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  CH  MM
##         CH 150  46
##         MM  18  56
##
##                Accuracy : 0.763
##                  95% CI : (0.7076, 0.8124)
##     No Information Rate : 0.6222
##     P-Value [Acc > NIR] : 5.867e-07
##
##                   Kappa : 0.4671
##
##  Mcnemar's Test P-Value : 0.0007382
##
##             Sensitivity : 0.8929
##             Specificity : 0.5490
##          Pos Pred Value : 0.7653
##          Neg Pred Value : 0.7568
##              Prevalence : 0.6222
##          Detection Rate : 0.5556
##    Detection Prevalence : 0.7259
##       Balanced Accuracy : 0.7209
##
##        'Positive' Class : CH
##
```

The polynomial kernel performed worse across the board.

(h) Overall, which approach seems to give the best results on this data?

While all models performed comparably, the linear kernel seemed to provide the most balanced results!