(2)

(a) Note $A \sim 9985 \times 563 \Rightarrow U \sim 9985 \times 9985$, $S \sim 9985 \times 563$, $V \sim 563 \times 563$

Lets first show $USV^T = \sum_{i=1}^{563} s_i U^{(i)} (V^{(i)})^T$ by block multiplication:

Lets write U as a 1×9985 block matrix of its columns and V as a 1×563 block matrix of its columns

$$USV^T = \begin{bmatrix} U^{(1)} & \cdots & U^{(9985)} \end{bmatrix} \left[\begin{array}{ccc|c} s_1 & & 0 & \\ & \ddots & & 0 \\ 0 & & s_{563} & \\ \hline & 0 & & 0 \end{array}\right] \begin{bmatrix} (V^{(1)})^T \\ \vdots \\ (V^{(563)})^T \end{bmatrix}$$

Note A has rank 563 $\Rightarrow$ 563 non-zero singular values

$$= \left[\begin{array}{ccc|ccc} U^{(1)} & \cdots & U^{(563)} & U^{(564)} & \cdots & U^{(9985)} \end{array}\right] \left[\begin{array}{ccc|c} s_1 & & 0 & \\ & \ddots & & 0 \\ 0 & & s_{563} & \\ \hline & 0 & & 0 \end{array}\right] \begin{bmatrix} (V^{(1)})^T \\ \vdots \\ (V^{(563)})^T \end{bmatrix}$$

$$= \begin{bmatrix} U^{(1)} & \cdots & U^{(563)} \end{bmatrix} \begin{bmatrix} s_1 & & 0 \\ & \ddots & \\ 0 & & s_{563} \end{bmatrix} \begin{bmatrix} (V^{(1)})^T \\ \vdots \\ (V^{(563)})^T \end{bmatrix} = \begin{bmatrix} s_1 U^{(1)} & \cdots & s_{563} U^{(563)} \end{bmatrix} \begin{bmatrix} (V^{(1)})^T \\ \vdots \\ (V^{(563)})^T \end{bmatrix}$$

$$= s_1 U^{(1)} (V^{(1)})^T + s_2 U^{(2)} (V^{(2)})^T + \cdots + s_{563} U^{(563)} (V^{(563)})^T = \sum_{i=1}^{563} s_i U^{(i)} (V^{(i)})^T$$

Now lets show that $A = USV^T$:

Let $X \in \mathbb{R}^{563}$. Lets show $AX = USV^T X$. Now, since the columns of V form an orthonormal basis for $\mathbb{R}^{563}$ it suffices to show that $Av^{(j)} = \sum_{i=1}^{563} s_i U^{(i)} (V^{(i)})^T v^{(j)}$ $\forall$ $v^{(j)}$ columns of V.

Let $v^{(j)}$ be given. Then $Av^{(j)} = USV^T v^{(j)} = \sum_{i=1}^{563} s_i U^{(i)} (V^{(i)})^T v^{(j)}$

$= \sum_{i=1}^{563} s_i U^{(i)} ((V^{(i)})^T v^{(j)}) = s_j U^{(j)}$ since $V^{(i)} \cdot V^{(j)} = 0$ if $i \neq j$ and $= 1$ o/w

And by defn of $U^{(j)}$ we know $s_j U^{(j)} = Av^{(j)}$ so we are done.

$A = USV^T = \sum_{i=1}^{563} s_i U^{(i)} (V^{(i)})^T$ ∎

(3)(a)

$$\min_{m\in\mathbb{R}^n} \sum_{i=1}^{N} \|x^{(i)} - m\|^2 = \min_{m\in\mathbb{R}^n} \sum_{i=1}^{N} (x^{(i)} - m)(x^{(i)} - m) =$$

$$= \min_{m\in\mathbb{R}^n} \sum_{i=1}^{N} \left[(x^{(i)})^T x^{(i)} - 2m^T x^{(i)} \right] + N(m^T m)$$

$$f(m) = N(m^T m) + \sum_{i=1}^{N} (x^{(i)})^T x^{(i)} - 2\sum_{i=1}^{N} m^T x^{(i)} = N(m^T m) + \sum_{i=1}^{N} (x^{(i)})^T x^{(i)} - 2\sum_{i=1}^{N} (x^{(i)})^T m$$

Now we can see $f(m)$ is quadratic for m, i.e., of the form $m^T A m + b^T m + C$.

In this case $A = N(I)$, $b = -2\sum_{i=1}^{N} x^{(i)}$, $C = \sum_{i=1}^{N} (x^{(i)})^T x^{(i)}$

And so we have a closed form solution for its gradient

$$\nabla f(m) = 2Am + b \Rightarrow \nabla f(m) = 2N(Im) - 2\sum_{i=1}^{N} x^{(i)} = (2N)m - 2\sum_{i=1}^{N} x^{(i)}$$

Now setting this to zero, $\nabla f(m) = 0$

$$\Rightarrow (2N)m - 2\sum_{i=1}^{N} x^{(i)} = 0 \Rightarrow m = \sum_{i=1}^{N} \frac{x^{(i)}}{N}$$

Now since for any given $m \in \mathbb{R}^n$, $N(m^T I m) = N(m^T m) = N\|m\|^2 \geq 0$

we have that $m = \sum_{i=1}^{N} \frac{x^{(i)}}{N}$ is a min for $f(m)$ ▨

HW14

Ines Pancorbo

4/27/2020

2 b)

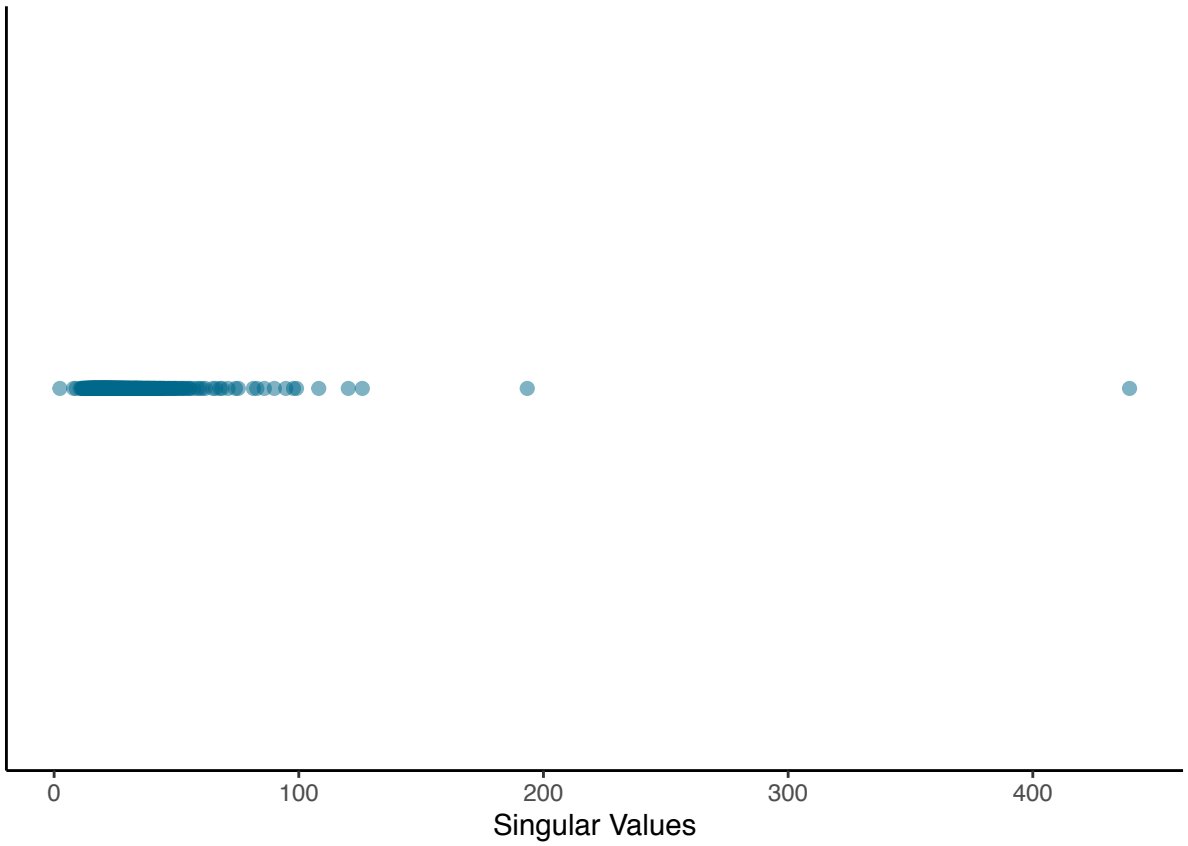
```
library("ggplot2")
A <- as.matrix(read.csv("user-shows.txt", header = F, sep = ' ', stringsAsFactors = T))
colnames(A) <- NULL
rownames(A) <- NULL

# rank of matrix A
qr(A)$rank
```

```
## [1] 563
```

```
# compute svd of A
svd <- svd(A)

# get singular values of A
singular_values <- svd$d
length(singular_values)
```

```
## [1] 563
```

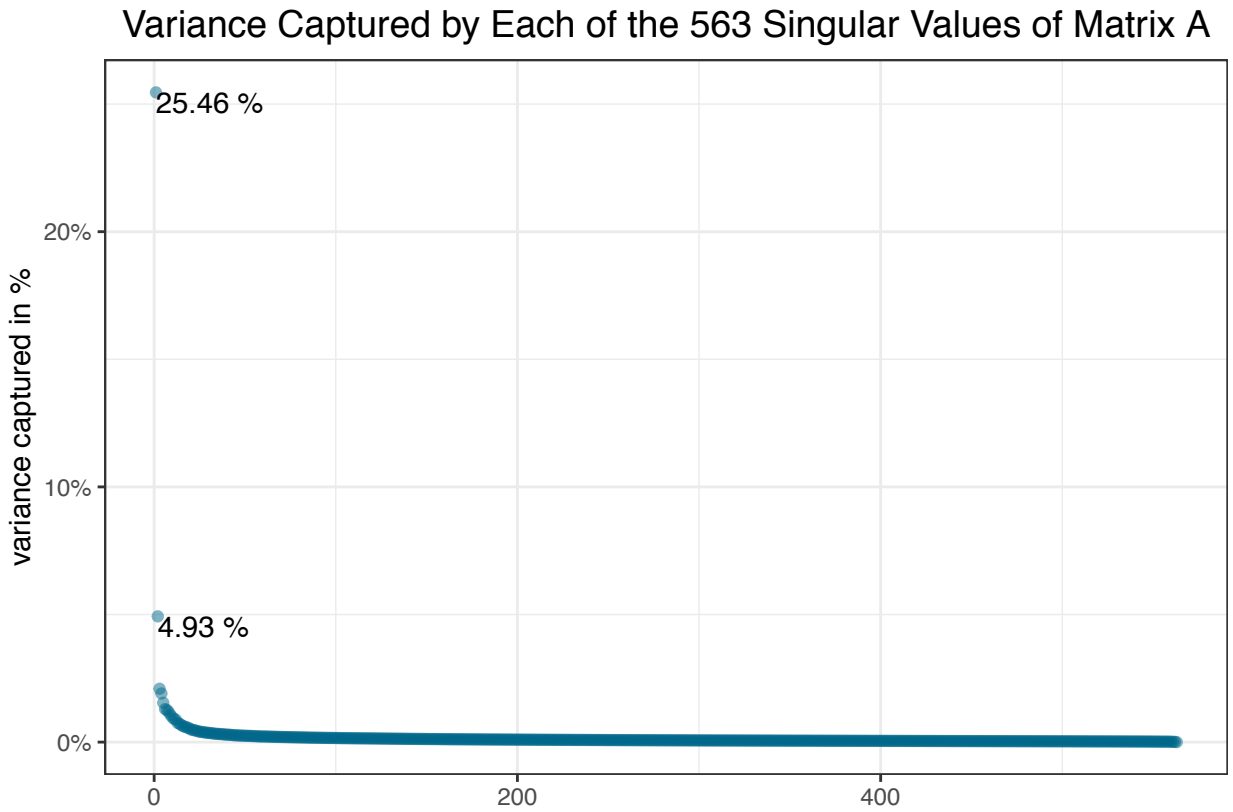
```
# plot singular values of A
ggplot() +
  geom_point(aes(x = singular_values,
                 y = rep(0, length(singular_values))),
             size = 2, color = 'deepskyblue4', alpha = 0.5) +
  theme_classic() +
  theme(axis.text.y = element_blank(),
        axis.ticks.y = element_blank() ) +
  xlab("Singular Values") + ylab("")
```



How many singular values would accurately approximate this matrix?

From the plot of singular values above it looks like there are two singular values that dominate, so it would make sense that the best rank 2 approximation of A (so 2 singular values) would "accurately" approximate this matrix. Accuracy would mean a balance between the amount of variance the k rank approximation (k singular values) preserves and the rank. Ideally, you would want a low rank (so for example, you can project onto a low-dimensional subspace) and a corresponding approximation that preserves much of the variance. However, there is usually a trade-off between variance and rank (the more variance you want to preserve, usually the higher rank you need). For our case it would be nice to see if the 2 rank approximation of A captures much (say approx 80% +) of the variance in A.

```
variance_captured <- function(k) sum(singular_values[k]^2)/sum(singular_values^2)
x <- seq(1, ncol(A), 1)
y <- sapply(x, variance_captured)
df <- data.frame(x = x, y = y)

p <- ggplot(df, aes(x, y))
p + geom_point(alpha = 0.5, color = 'deepskyblue4') +
  geom_text(data = subset(df, y > y[3]),
            aes(x, y, label = paste(sprintf("%0.2f", round(y*100, digits = 2)),"%")),
            size = 4, hjust = 0, vjust = 1) +
  theme_bw() +
  xlab("") + ylab("variance captured in %") +
  scale_y_continuous(labels = scales::percent_format(accuracy = 1)) +
  ggtitle("Variance Captured by Each of the 563 Singular Values of Matrix A") +
  theme(
      plot.title = element_text(size = 14, hjust = 0.5)
  )
```



The plot above tells us the same thing as the previous plot: There are two singular values that dominate, and we can see how the rank 1 approx to A $s_1 u^{(1)}(v^{(1)})^T$ captures 25.46% of the variance in A, and the rank 1 approx to A $s_2 u^{(2)}(v^{(2)})^T$ captures 4.93% of the variance in A. Both these approximations capture more variance than any of the other rank 1 approximations. We can also see that the rank 2 approximation of A $s_1 u^{(1)}(v^{(1)})^T + s_2 u^{(2)}(v^{(2)})^T$ captures around 30% of the variance in A. We can argue that 30% is small but we are considering a rank 2 approximation to A. The next plot shows how much higher the rank would have to be (how many more singular values to keep) in order for us to capture more of the variance of A.

```
cum_variance_captured <- function(k) sum(singular_values[1:k]^2)/sum(singular_values^2)
x <- seq(1, ncol(A), 1)
y <- sapply(x, cum_variance_captured)
df <- data.frame(x = x,y = y)

# How many singular would preserve around 80% of A's variance?
y[y > 0.8][1]
```

```
## [1] 0.8005798
```

```
match(y[y > 0.8][1], y)
```

```
## [1] 194
```

So we can see that to capture around 80% or more of the variance of A we would have to keep at least 194 singular values (i.e., we can think of projecting onto a 194 or higher dimensional subspace).

```
# How many singular would preserve around 50% of A's variance?
y[y > 0.5][1]
```

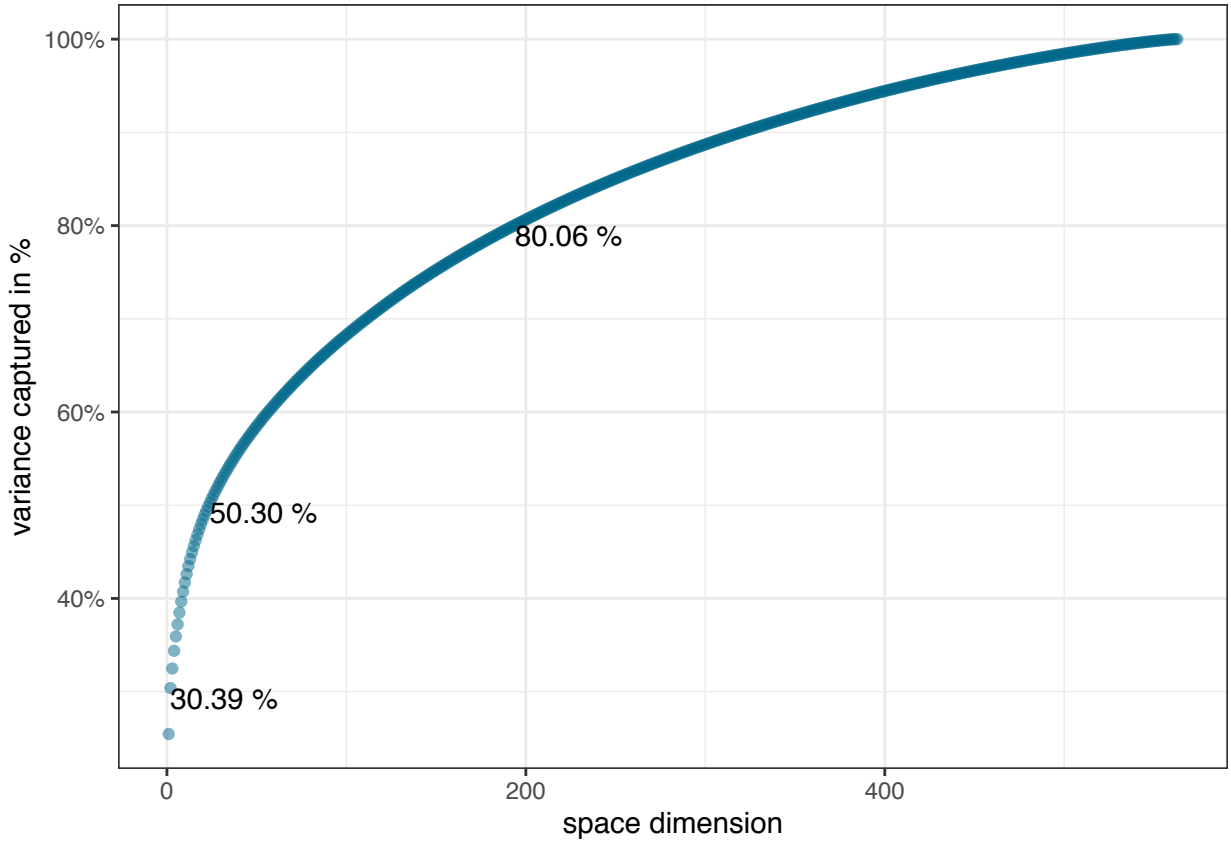
```
## [1] 0.5030221
```

```
match(y[y > 0.5][1], y)
```

```
## [1] 24
```

So we can see that to capture around 50% or more of the variance of A we would have to keep at least 24 singular values (i.e., we can think of projecting onto a 24 or higher dimensional subspace).

```
p <- ggplot(df, aes(x, y))
p + geom_point(alpha = 0.5, color = 'deepskyblue4') +
  geom_text(data = df[df$x %in% c(2, 24, 194),,],
            aes(x,y,label =
                  paste(sprintf("%0.2f", round(y*100, digits = 2)),"%")),
            size = 4, hjust = 0, vjust = 1) +
  theme_bw() +
  xlab("space dimension")+ylab("variance captured in %") +
  scale_y_continuous(labels = scales::percent_format(accuracy = 1))
```



From the plot, if we wanted to keep 80% or more of A's variance we would have to approximate A with a 194 rank or higher matrix (thinking in terms of this problem of shows/users, projecting onto a 194 or higher dimensional subspace), which might be excessive. Even if we only wanted to preserve around half of the variance we would have to approximate A with a 24 rank matrix (project onto a 24 dimensional subspace in terms of this problem of shows/users). So we can argue that keeping 2 singular values would "accurately" approximate A, accurate in the sense that we are trying to balance rank and variance preserved by that rank approximation to A.

2 c)

Matrix A is a 9985 by 563 matrix. The rows correspond to TV users and the columns correspond to TV shows. So we have 563 TV shows and 9985 TV users. And therefore, A maps TV users to TV shows.

We are interested in (1) projecting the users onto a two dimensional PCA space and (2) projecting the shows onto a two dimensional PCA space. Like we did in previous HWs we could do the following for (1) and (2):

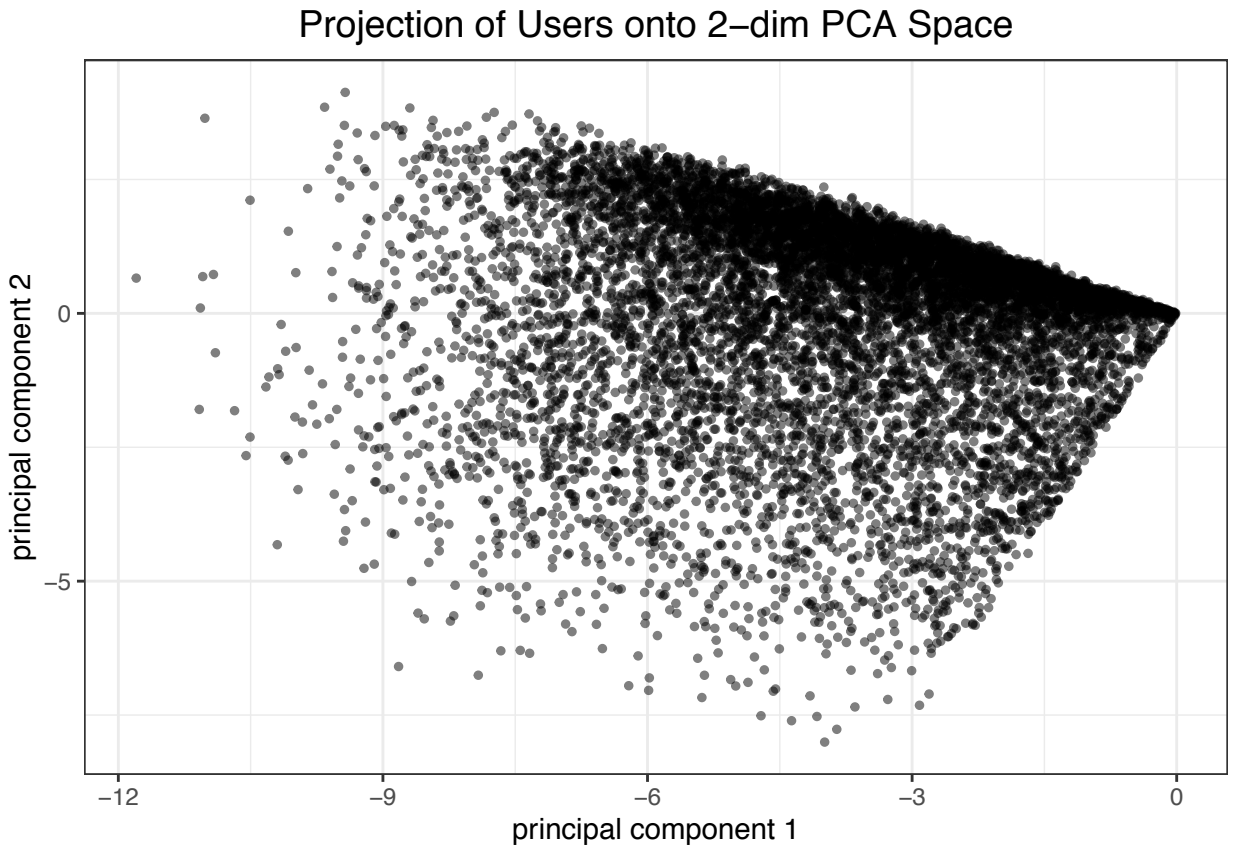
For (1) we would deal with matrix A since the rows of that matrix are the users. To compute its first two principal components we would compute the first two dominant eigenvectors, $q^{(1)}, q^{(2)}$ of A's covariance matrix, A^TA. We would then project onto the space spanned by $q^{(1)}, q^{(2)}$. Let $Q = (q^{(1)}q^{(2)})$. Then we would compute AQ, where the first and second columns would have the coefficients needed to plot the 2-dimensional projections of the users.

For (2) we would deal with matrix A^T since the rows of that matrix are the shows. To compute its first two principal components we would compute the first two dominant eigenvectors, $k^{(1)}, k^{(2)}$ of A^T's covariance matrix, AA^T. We would then project onto the space spanned by $k^{(1)}, k^{(2)}$. Let $K = (k^{(1)}k^{(2)})$. Then we would compute A^TK, where the first and second columns would have the coefficients needed to plot the 2-dimensional projections of the shows.

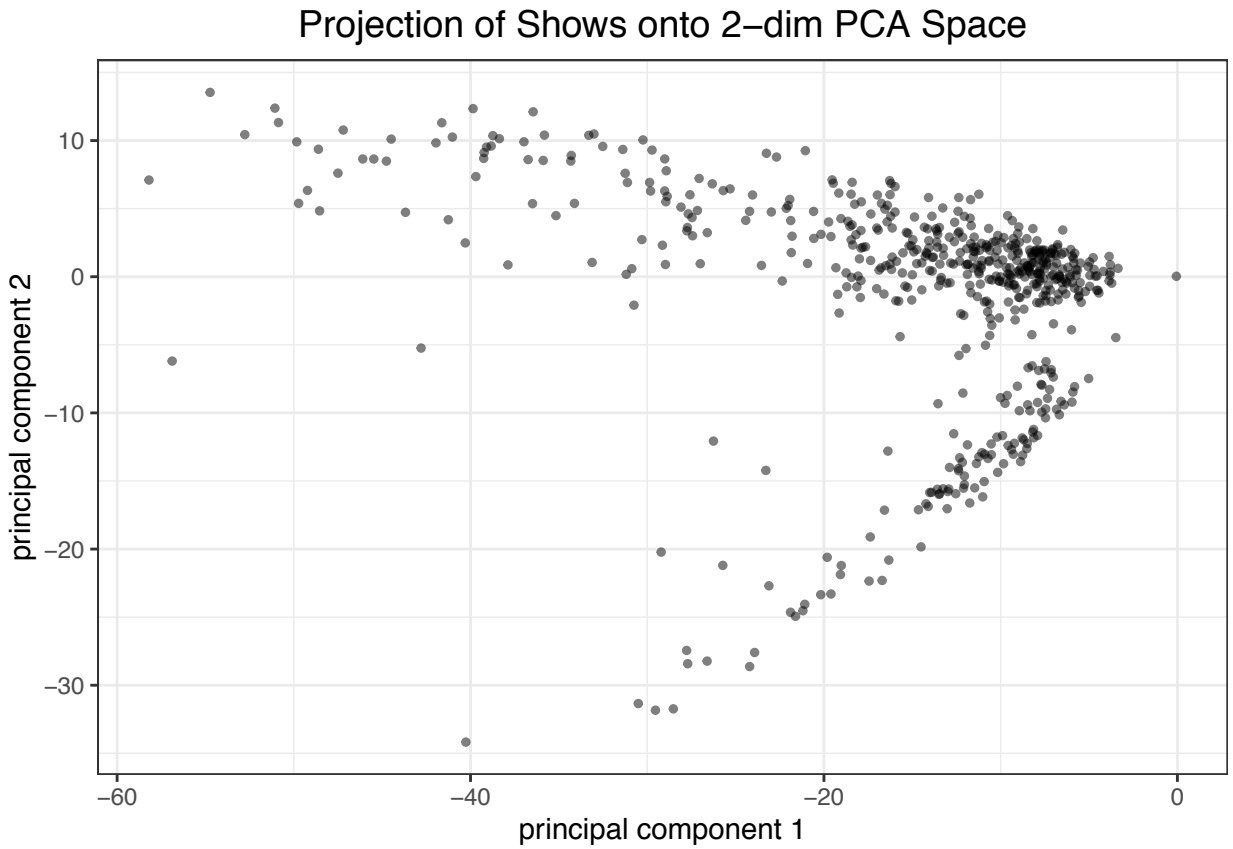
However, as seen in the lecture videos there is another way to do (1) and (2) via the SVD of A. Given the SVD of A, i.e., $A = USV^T$ we know the columns of V are the eigenvectors of A^TA, i.e., the principal components of A. So we will do (1) by projecting the users onto the two dimensional space spanned by $v^{(1)}, v^{(2)}$. Similarly, we know the columns of U are the eigenvectors of AA^T, i.e., the principal components of A^T. So we will do (2) by projecting the shows onto the two dimensional space spanned by $u^{(1)}, u^{(2)}$.

```r
# get U and V
U <- svd$u
V <- svd$v

Q <- V[,1:2]
K <- U[,1:2]
```

```r
# 2-dim coefficients for users
users_p <- A %*% Q

# 2-dim coefficients for shows
shows_p <- t(A) %*% K


# plotting users
ggplot() +
  geom_point(aes(x = users_p[,1], y = users_p[,2]), alpha = 0.5, size = 1) +
  theme_bw() +
  xlab("principal component 1") + ylab("principal component 2") +
  ggtitle("Projection of Users onto 2-dim PCA Space") +
  theme(
      plot.title = element_text(size = 14, hjust = 0.5)
  )
```




```
# plotting shows
ggplot() +
  geom_point(aes(x = shows_p[,1], y = shows_p[,2]), alpha = 0.5, size = 1) +
  theme_bw() +
  xlab("principal component 1") + ylab("principal component 2") +
  ggtitle("Projection of Shows onto 2-dim PCA Space") +
  theme(plot.title = element_text(size = 14, hjust = 0.5))
```




```
# let's save our 2-d show coordinates in a dataframe
coefficients_shows <- data.frame(x = shows_p[,1], y = shows_p[,2])
```

To recommend movies for Alex an approach would be to consider shows close to the shows we know he likes

```
show_names <- as.matrix(read.csv("shows.txt", header = F, sep = ' '))
```

```
set.seed(23)
# We could find the 5 closest shows to one show Alex likes
# But lets get 5 shows Alex likes, and find a show close to each
# of these 5
shows <- which(A[500,] == 1)[runif(5, 1, 50)]
shows
```

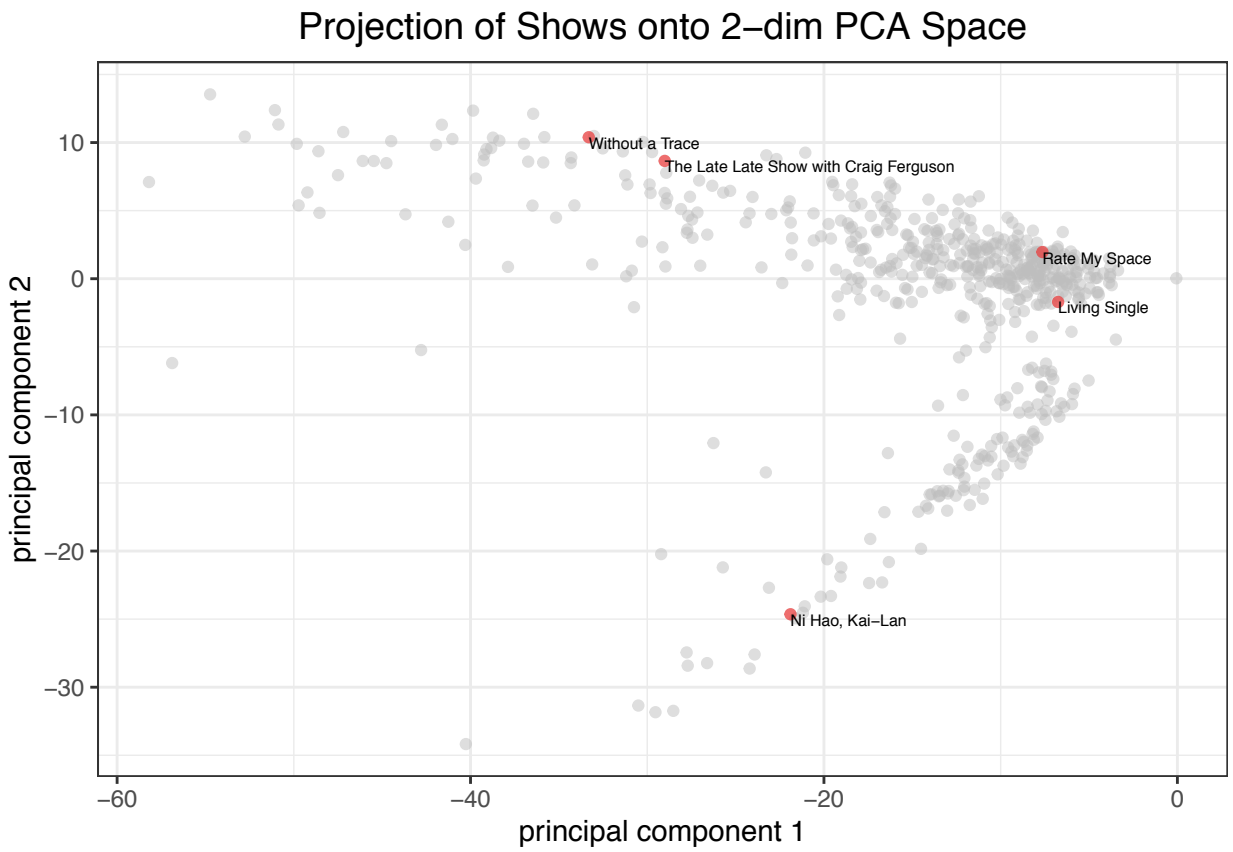
```
## [1] 236 140 156 244 258
```

```
# Let's get the names of these shows
show_names[shows,]
```

```
## [1] "Living Single"
## [2] "Without a Trace"
## [3] "Ni Hao, Kai-Lan"
## [4] "Rate My Space"
## [5] "The Late Late Show with Craig Ferguson"
```

```
# Let's plot those 5 shows Alex likes in red and all other shows in grey
p <- ggplot(coefficients_shows, aes(x, y))
p + geom_point(color = 'grey', alpha = 0.5) +
  geom_point(data = coefficients_shows[shows,],
             aes(x, y), color = 'red', alpha = 0.5) +
```

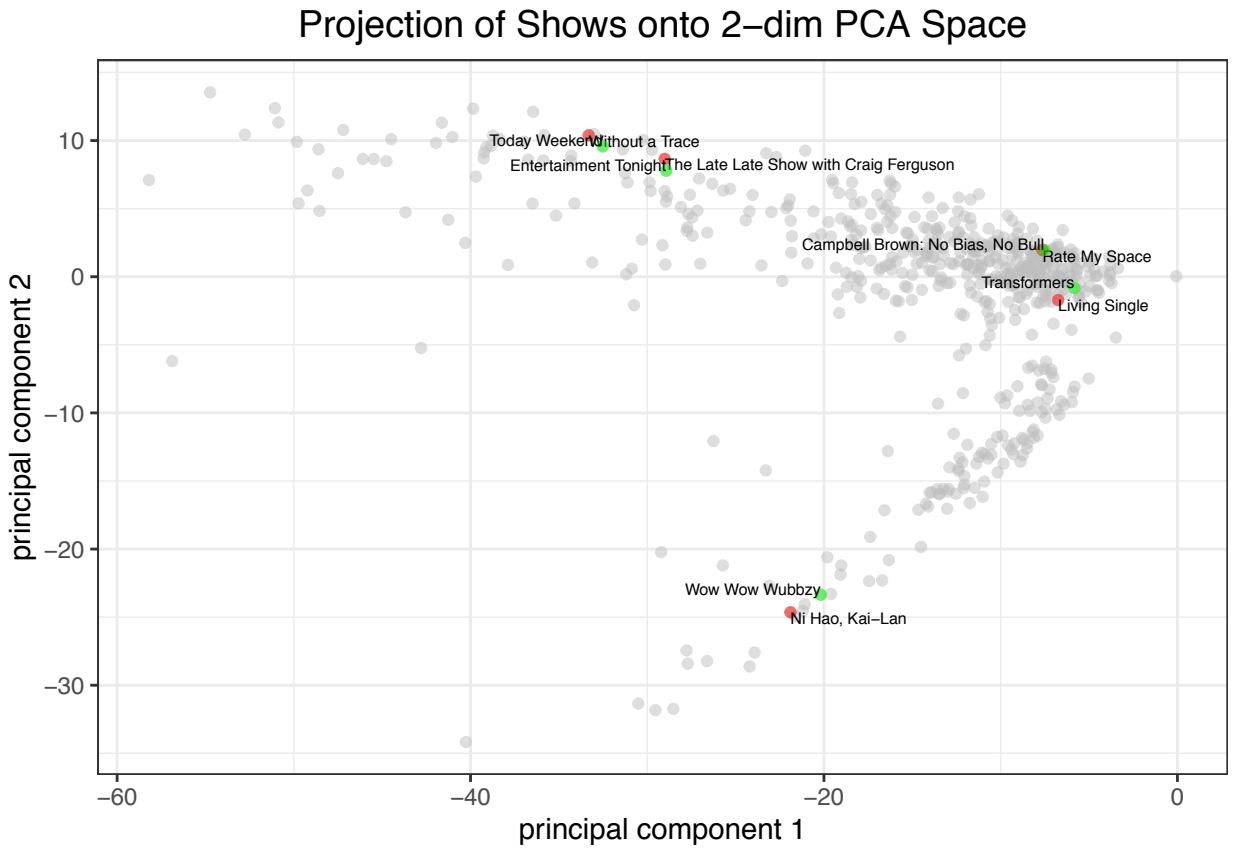
```
  geom_text(data = coefficients_shows[shows,],
            aes(x, y, label = show_names[shows,]), size = 2, hjust = 0, vjust = 1) +
  theme_bw() +
  xlab("principal component 1") + ylab("principal component 2") +
  ggtitle("Projection of Shows onto 2-dim PCA Space") +
  theme(
      plot.title = element_text(size = 14, hjust = 0.5)
  )
```




```
norm <- function(x) sqrt(sum(x^2))
# getting the 2-dim coordinates for the 5 shows Alex likes
show1 <- shows_p[shows[1],]
show2 <- shows_p[shows[2],]
show3 <- shows_p[shows[3],]
show4 <- shows_p[shows[4],]
show5 <- shows_p[shows[5],]

# getting all coordinates for the first 100 shows for which
# Alex's ratings where zeroed out
c <- t(cbind(shows_p[1:100,1], shows_p[1:100,2]))

# calculating the distances between each the first 100 shows
# and each of the 5 selected shows
# Then, sorting distances and choosing the
# smallest distance, returning the corresponding index
recommended1 <- sort(apply(c-show1, MARGIN = 2, FUN = norm),
                     decreasing = F, index.return = T)$ix[1]
```

```r
recommended2 <- sort(apply(c-show2, MARGIN = 2, FUN = norm),
                     decreasing = F, index.return = T)$ix[1]
recommended3 <- sort(apply(c-show3, MARGIN = 2, FUN = norm),
                     decreasing = F, index.return = T)$ix[1]
recommended4 <- sort(apply(c-show4, MARGIN = 2, FUN = norm),
                     decreasing = F, index.return = T)$ix[1]
recommended5 <- sort(apply(c-show5, MARGIN = 2, FUN = norm),
                     decreasing = F, index.return = T)$ix[1]

# collecting all indexes of the recommended shows
r <- c(recommended1, recommended2, recommended3, recommended4, recommended5)
```

```r
# Lets plot the shows we know Alex likes (in red) and
# the shows recommended (in green)
p <- ggplot(coefficients_shows, aes(x, y))
p + geom_point(color = 'grey', alpha = 0.5) +
  geom_point(data = coefficients_shows[shows,],
             aes(x,y), color = 'red', alpha = 0.5) +
  geom_text(data = coefficients_shows[shows,],
             aes(x, y, label = show_names[shows,]), size = 2, hjust = 0, vjust = 1) +
  geom_point(data = coefficients_shows[r,],
             aes(x,y),color = 'green', alpha = 0.5) +
  geom_text(data = coefficients_shows[r,],
             aes(x, y, label = show_names[r,]), size = 2, hjust = 1, vjust = 0) +
  theme_bw() +
  xlab("principal component 1") + ylab("principal component 2") +
  ggtitle("Projection of Shows onto 2-dim PCA Space") +
  theme(
      plot.title = element_text(size = 14, hjust = 0.5)
  )
```



Let's check if Alex actually likes those shows labeled green

```
# Read in the shows Alex rated
# We just care about the first 100, which were zeroed out
shows_Alex_rated <- as.matrix(read.csv("alex.txt", header = F, sep = ' ', stringsAsFactors = T))[1:100]


data.frame(show_liked = show_names[shows],
           recommended_show = show_names[r],
           real_rating_of_recommended = shows_Alex_rated[r],
           pred_rating_of_recommended = rep(1,5))
```

```
##                                 show_liked                recommended_show
## 1                             Living Single                     Transformers
## 2                           Without a Trace                    Today Weekend
## 3                           Ni Hao, Kai-Lan                    Wow Wow Wubbzy
## 4                             Rate My Space Campbell Brown: No Bias, No Bull
## 5 The Late Late Show with Craig Ferguson             Entertainment Tonight
##   real_rating_of_recommended pred_rating_of_recommended
## 1                          0                          1
## 2                          0                          1
## 3                          1                          1
## 4                          0                          1
## 5                          1                          1
```

We can see that it turns out that Alex only likes 2 out of the 5 shows recommended. So we are 40% accurate. If we were to recommend at random: we note that out of the 100 first shows, he actually likes 19 of these. So at random we would recommend 5/5 shows he likes with an accuracy of 0.015% and recommend 2/5 shows

he likes with an accuracy of 19.38%. So the SVD performs better than random when recommending shows for Alex.

```
# how many shows Alex likes out of first 100
length(which(shows_Alex_rated[1:100] == 1))
```

```
## [1] 19
```

```
# random accuracy
choose(19,2)*choose(81,3)/choose(100,5)*100
```

```
## [1] 19.37867
```

```
choose(19,5)/choose(100,5)*100
```

```
## [1] 0.01544479
```

3 b)

```
MyKmeans <- function(x, K){
  # Let's work with dataframes to ease plotting
  # Randomly assign datapoints to clusters
  data <- data.frame(x, assignment = sample.int(K, nrow(x), replace = T))
  # Create empty dataframe for mean of clusters
  centers <- data.frame(matrix(, nrow = ncol(x), ncol = K))

  iter <- 1
  repeat{
    # compute cluster centers
    for (i in 1:K){
      centers[, i] <- colMeans(subset(data[,1:ncol(x)], data$assignment == i))
    }

    ############################ plotting to visualize ##############################
    print(ggplot(data, aes(V1, V2, colour = as.factor(assignment))) +
            geom_point() +
            scale_colour_manual(values = c("seagreen1", "indianred1")) +
            annotate("point", x = t(centers)[, 1], y = t(centers)[, 2],
                     size = 6, shape = 17, colour = c("seagreen2", "indianred1")) +
            theme_bw() +
            ggtitle(paste("Iteration = ", iter)) +
            xlab("") + ylab("") +
            theme(legend.title = element_blank(),
                  plot.title = element_text(size = 14, face = 'bold', hjust = 0.5)))
    ##################################################################################

    old_assignment <- data$assignment
    # assign points to closest centers
    for(i in 1:nrow(x)){
      data$assignment[i] <- which.min(apply(x[i,]-centers, MARGIN = 2, FUN = norm))
    }

    # stopping condition
    if(identical(old_assignment, data$assignment)){
      break
```

```
    }

    iter <- iter + 1
  }
  return(list(data = data, centers = centers, iter = iter))
}
```

Explain why the kmeans algorithm is a descent algorithm.

The kmeans algorithm is as follows:

Initialize assignments, $a^{(i)}, i = 1, 2, \cdots, N$, randomly

Repeat until stopping condition {

For j,

$$\mu_j := \frac{\sum_{i=1}^{N} \chi_{\{a^{(i)}=j\}} x^{(i)}}{\sum_{i=1}^{N} \chi_{\{a^{(i)}=j\}}}$$

For i,

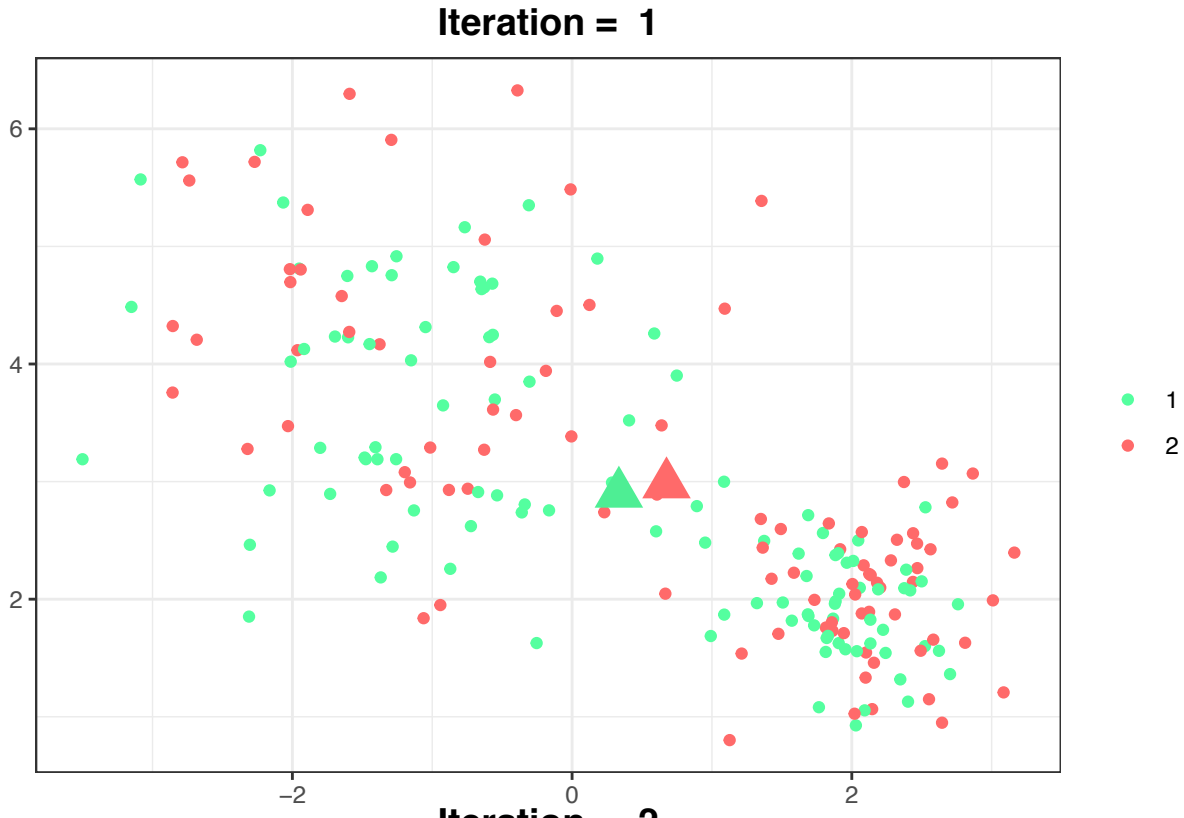
$$a^{(i)} := \arg\min_j ||x^{(i)} - \mu_j||$$

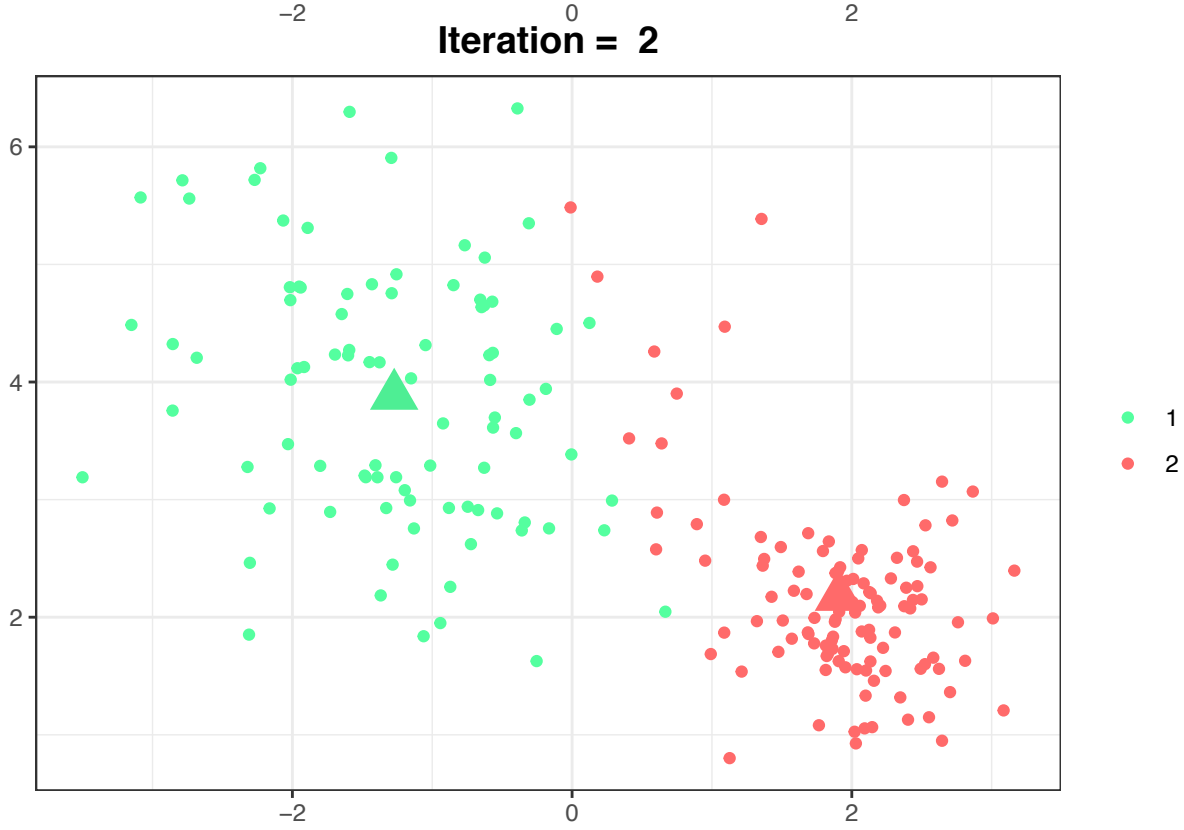
}

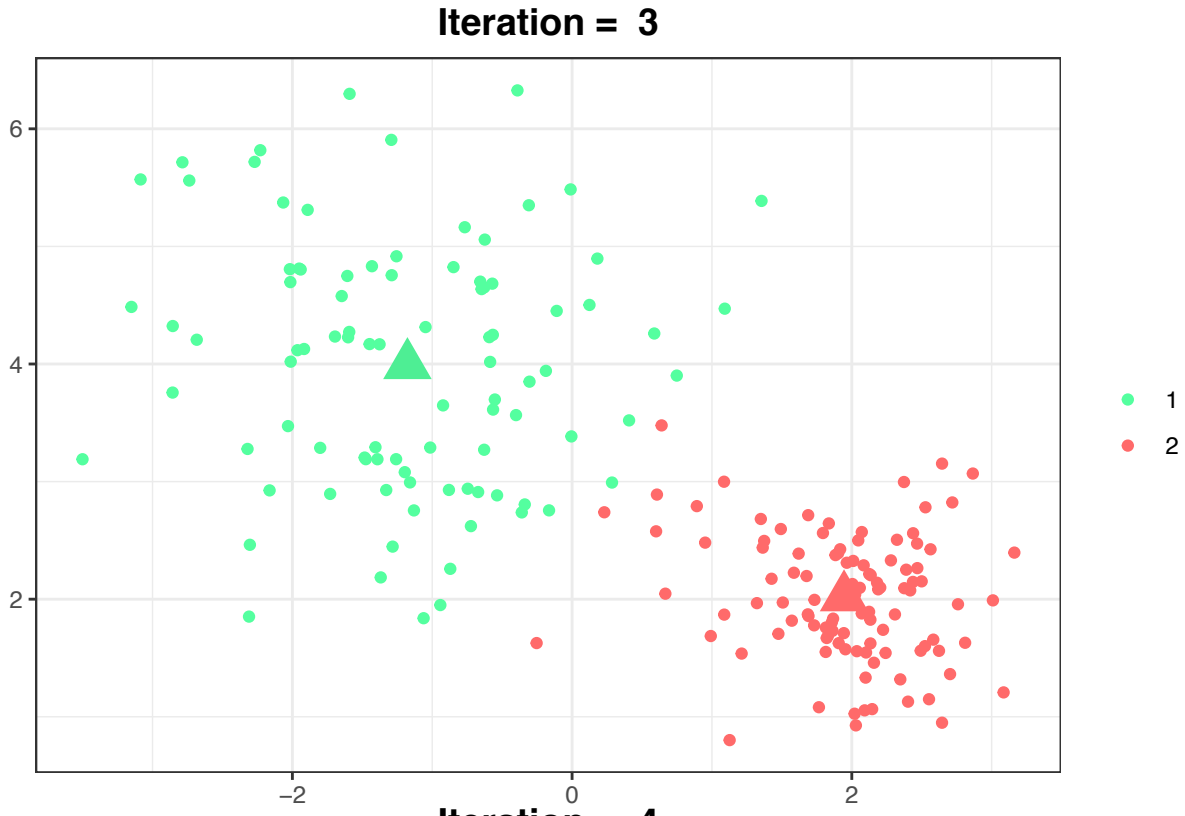
Defining the loss function, $L(a, \mu) = \sum_{i=1}^{N} ||x^{(i)} - \mu^{a^{(i)}}||^2$, which measures the sum of squared distances between each of the N datapoints $x^{(i)}$ and its assigned cluster center $\mu^{a^{(i)}}$. From the algorithm above, the inner-loop minimizes $L(a, \mu)$ first with respect to μ while fixing a (showed to be the mean in 3a) and then minimizes $L(a, \mu)$ with respect to a while fixing μ. So, at each iteration of the loop, $L(a, \mu)$ monotonically decreases, which implies we have a descent algorithm. Essentially, kmeans algorithm is an example of a coordinate descent algorithm. However, $L(a, \mu)$ is non-convex so we are not guarenteed to converge to the global minimum (which is why R's built-in function has an option to run kmeans many times, using different random values for the initialization, and returns the assignments and cluster centers that resulted in the lowest $L(a, \mu)$).

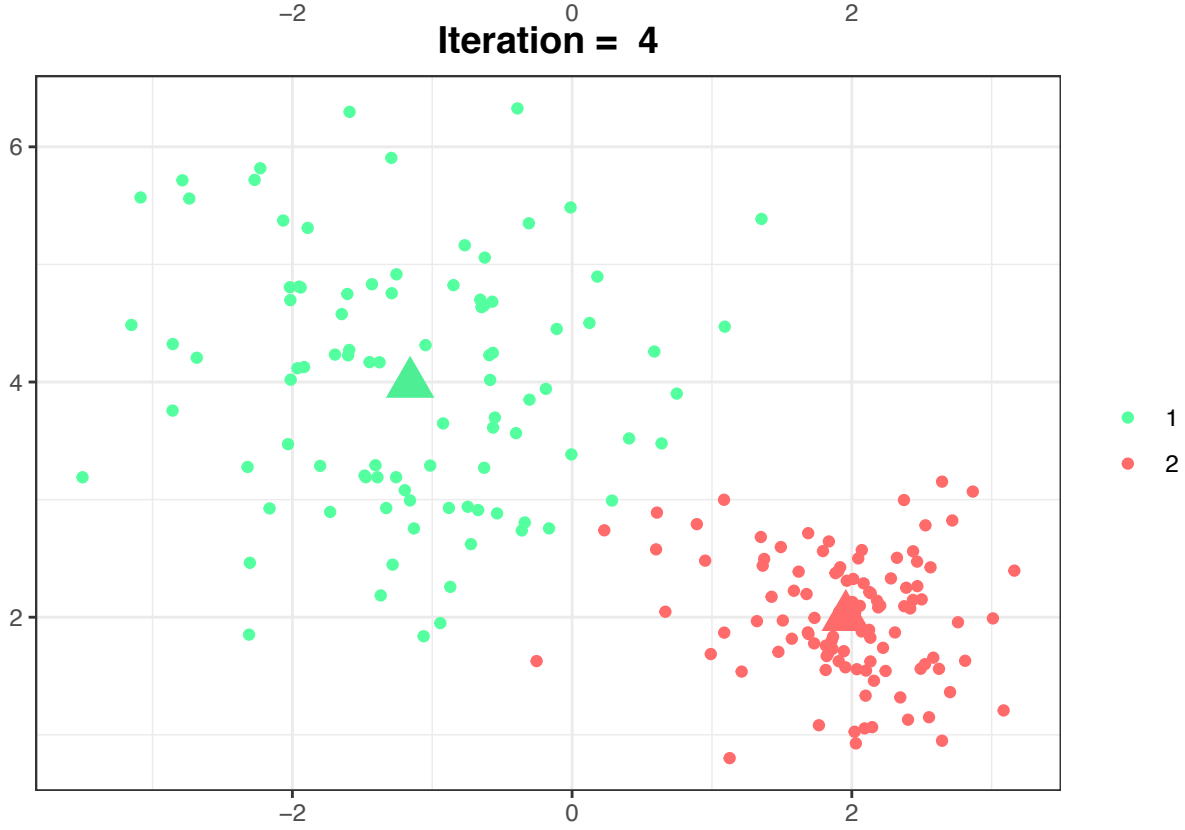
3 (c)

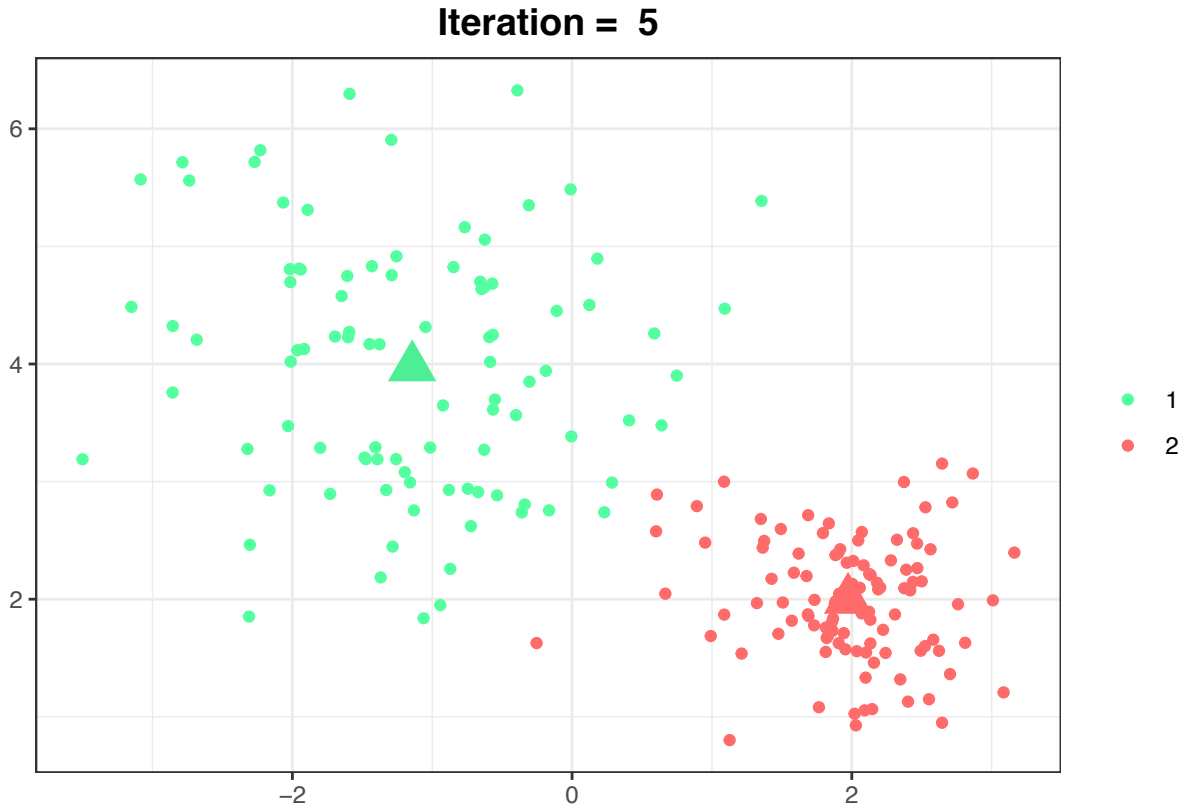
```
set.seed(2)
x <- as.matrix(read.csv("synthetic kmeans data.csv", header = T, stringsAsFactors = T))
kmeans_soln <- MyKmeans(x,2)
```


Iteration = 1
6
4
2
−2
0
2
1
2


Iteration = 2
6
4
2
−2
0
2
1
2


Iteration = 3
6
4
2
−2
0
2
1
2


Iteration = 4
6
4
2
−2
0
2
1
2




```
# number of iterations
kmeans_soln$iter
```

```
## [1] 5
```

```
# optimal means are at
kmeans_soln$centers[,1]
```

```
## [1] -1.143436  3.972099
```

```
kmeans_soln$centers[,2]
```

```
## [1] 1.973343 1.995245
```

```
# let's compare with R's kmeans function
# to check we get the same result
r <- kmeans(x, 2, nstart = 20)
r$centers[1,]
```

```
##       V1       V2
## 1.973343 1.995245
```

```
r$centers[2,]
```

```
##        V1        V2
## -1.143436  3.972099
```

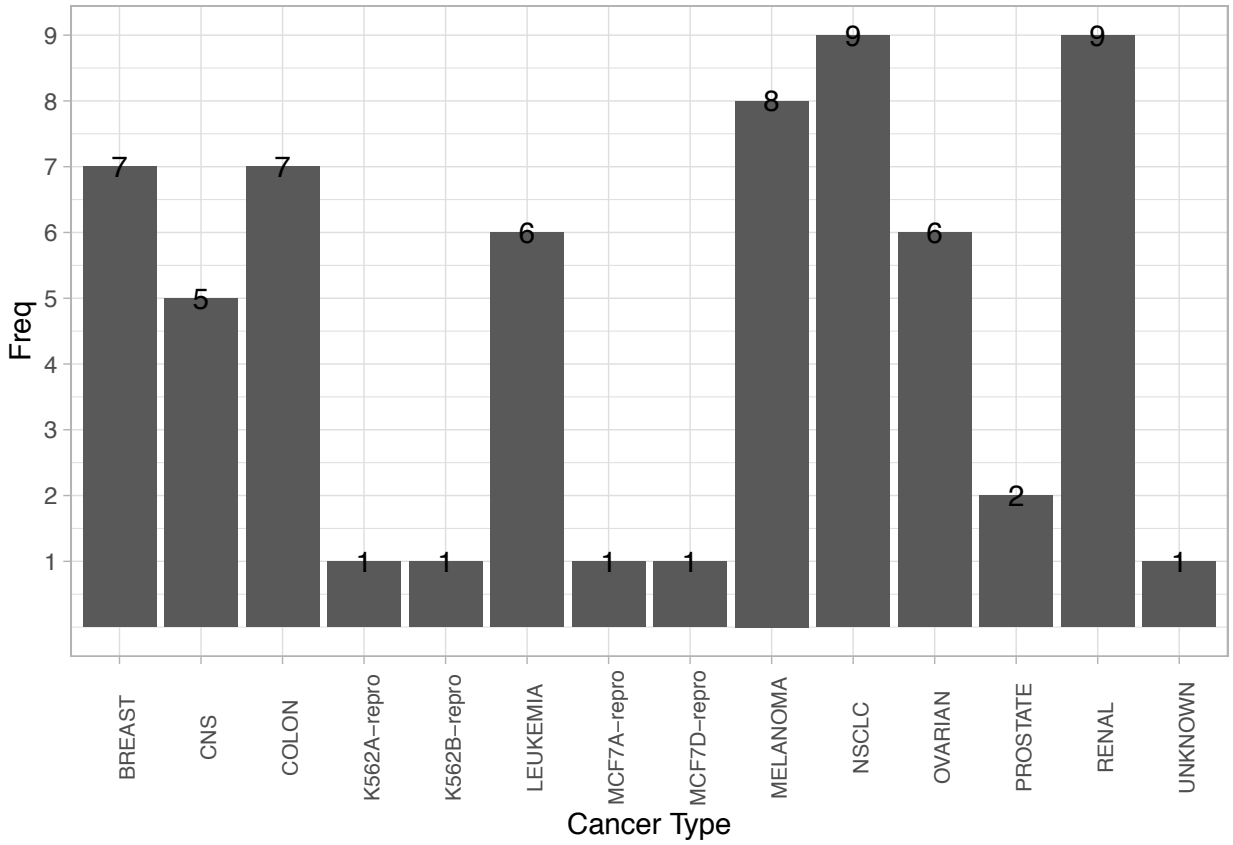
3 (d)

```
data <- read.csv("tumor microarray data.csv", header = T)
```

```
points <- data[,2:ncol(data)]
cancer <- data[,1]
```

See below for a breakdown of cancer cells by cancer type

```
cancer_freq_table <- data.frame(table(cancer))
ggplot(cancer_freq_table, aes(cancer, Freq)) +
  geom_col() +
  scale_y_continuous(breaks = seq(1, 10, 1)) +
  geom_text(aes(label = Freq)) +
  xlab("Cancer Type") +
  theme_light() +
  theme(axis.text.x = element_text(size = 8, angle = 90),
        legend.position = "none")
```



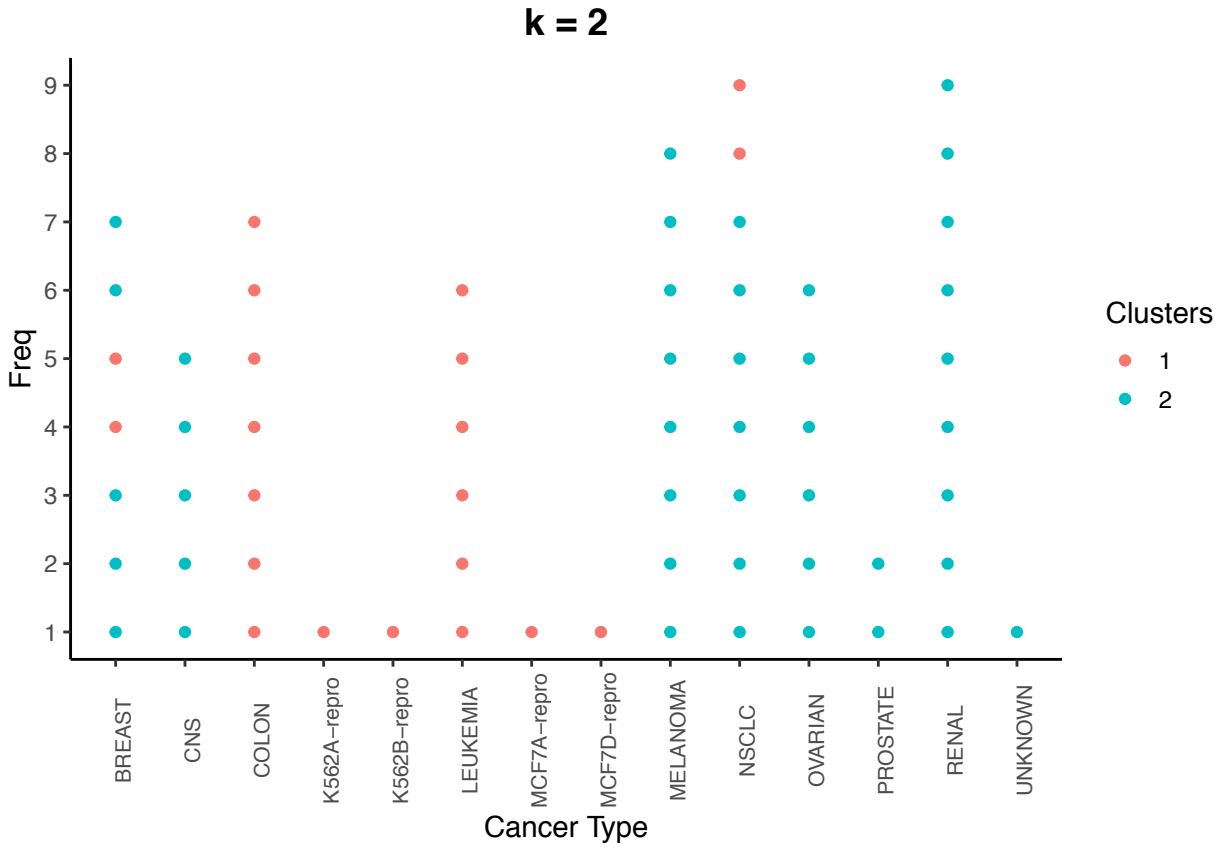
Now lets see for different k if the clusters formed separate the cancers

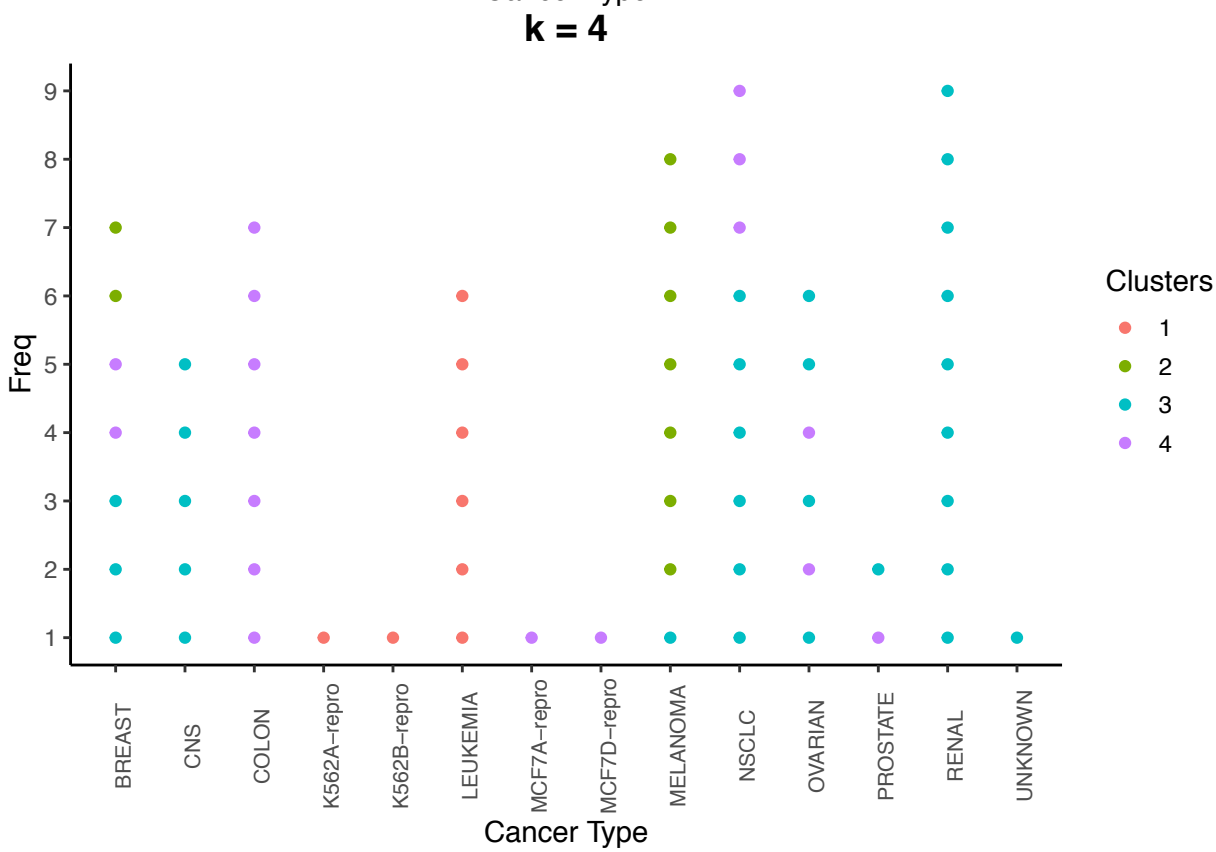
```
library("dplyr")
```

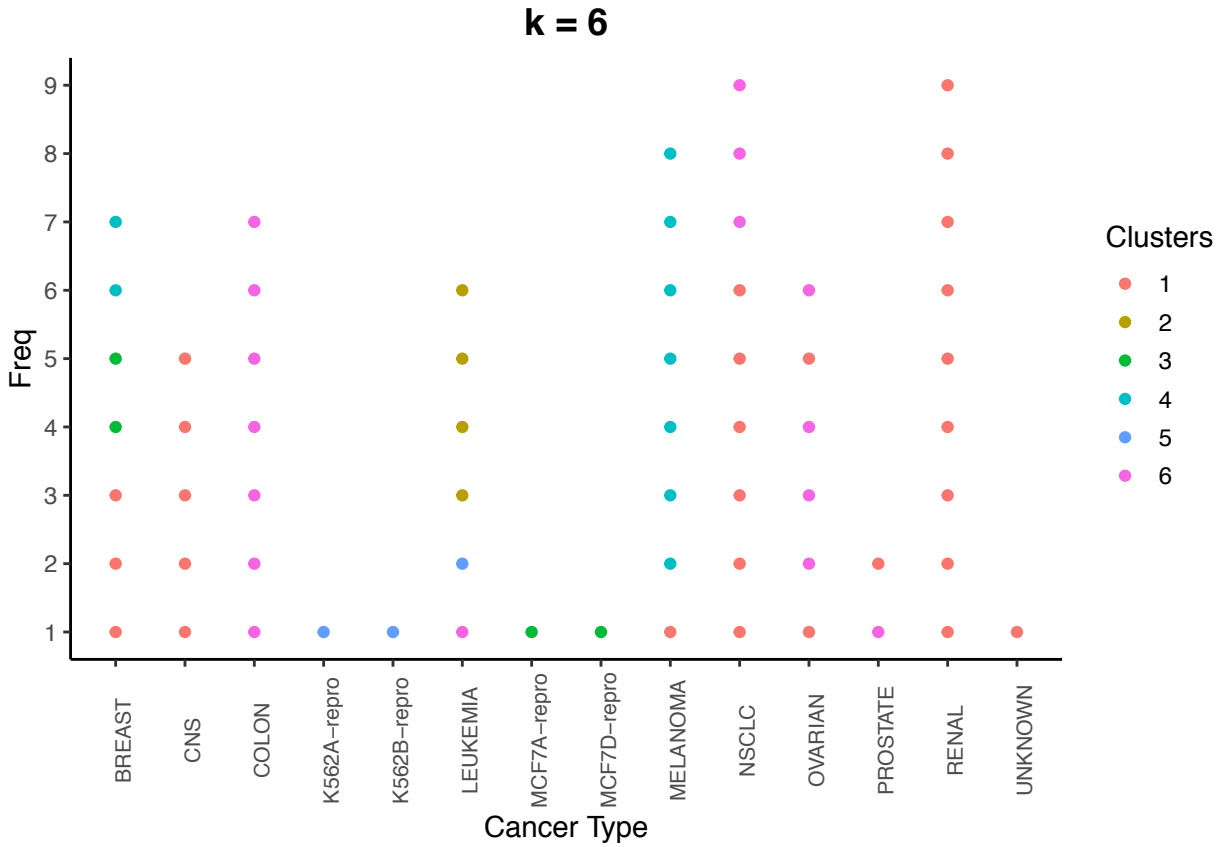
```
##
## Attaching package: 'dplyr'
## The following objects are masked from 'package:stats':
##
##     filter, lag
## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

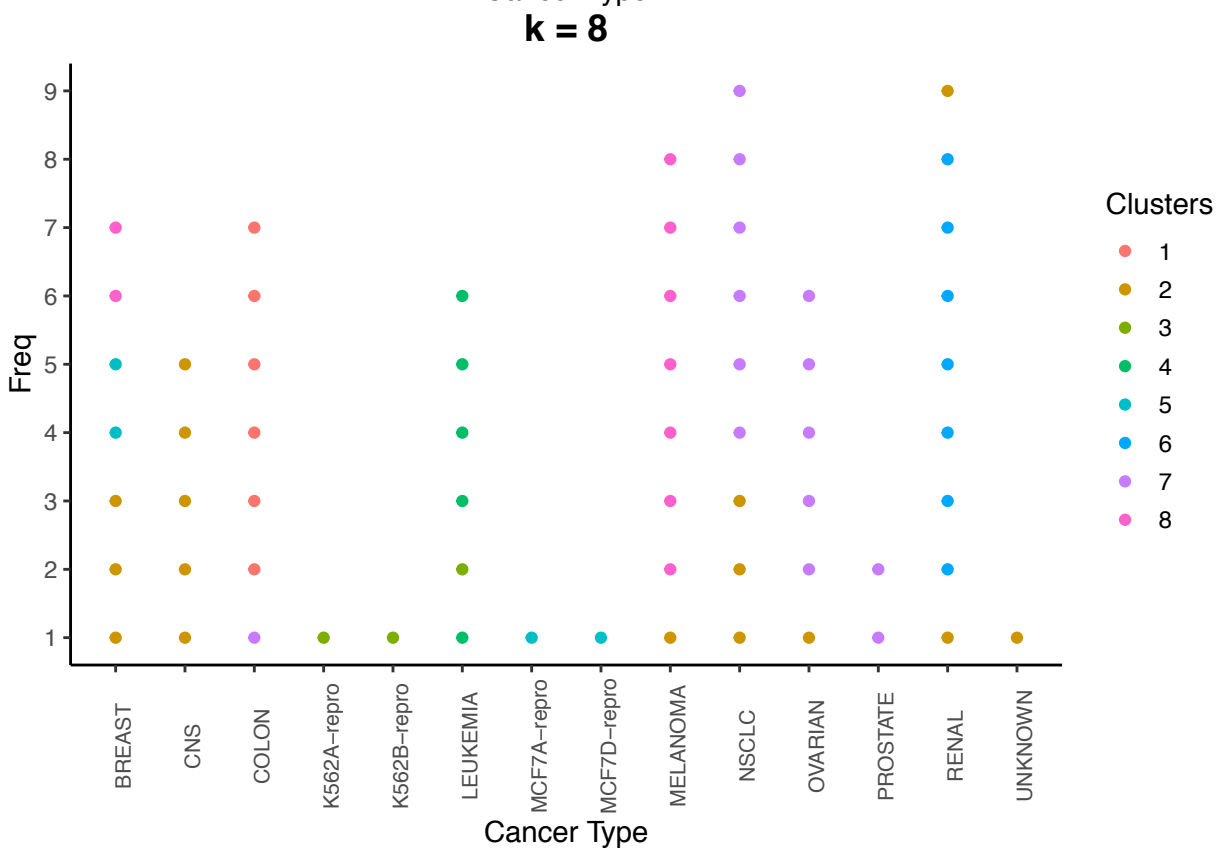
```r
kmeans_plot <- function(k){
  kmeans_soln <- kmeans(points, centers = k, nstart = 30)
  df <- data.frame(cancer_type = cancer, cluster = kmeans_soln$cluster)
  df <- df[order(df$cancer),]
  df <- df %>%
    group_by(cancer_type) %>%
    mutate(x = row_number())
  df <- mutate(df, y = as.integer(cancer_type))

  cancer_types <- unique(df$cancer_type)

  print(ggplot(df, aes(y, x, color = as.factor(cluster)), size = 8) +
    geom_point() +
    xlab("Cancer Type") + ylab("Freq") +
    theme_classic() +
    scale_x_continuous(breaks = seq(1, length(cancer_types), 1),
                       labels = cancer_types) +
    scale_y_continuous(breaks = seq(1, length(cancer_types), 1)) +
    ggtitle(paste("k =", k)) +
    scale_colour_discrete("Clusters") +
    theme(axis.text.x = element_text(size = 8, angle = 90),
          legend.key.size = unit(1,"line"),
          plot.title = element_text(size = 14, face = 'bold', hjust = 0.5)))
}
```

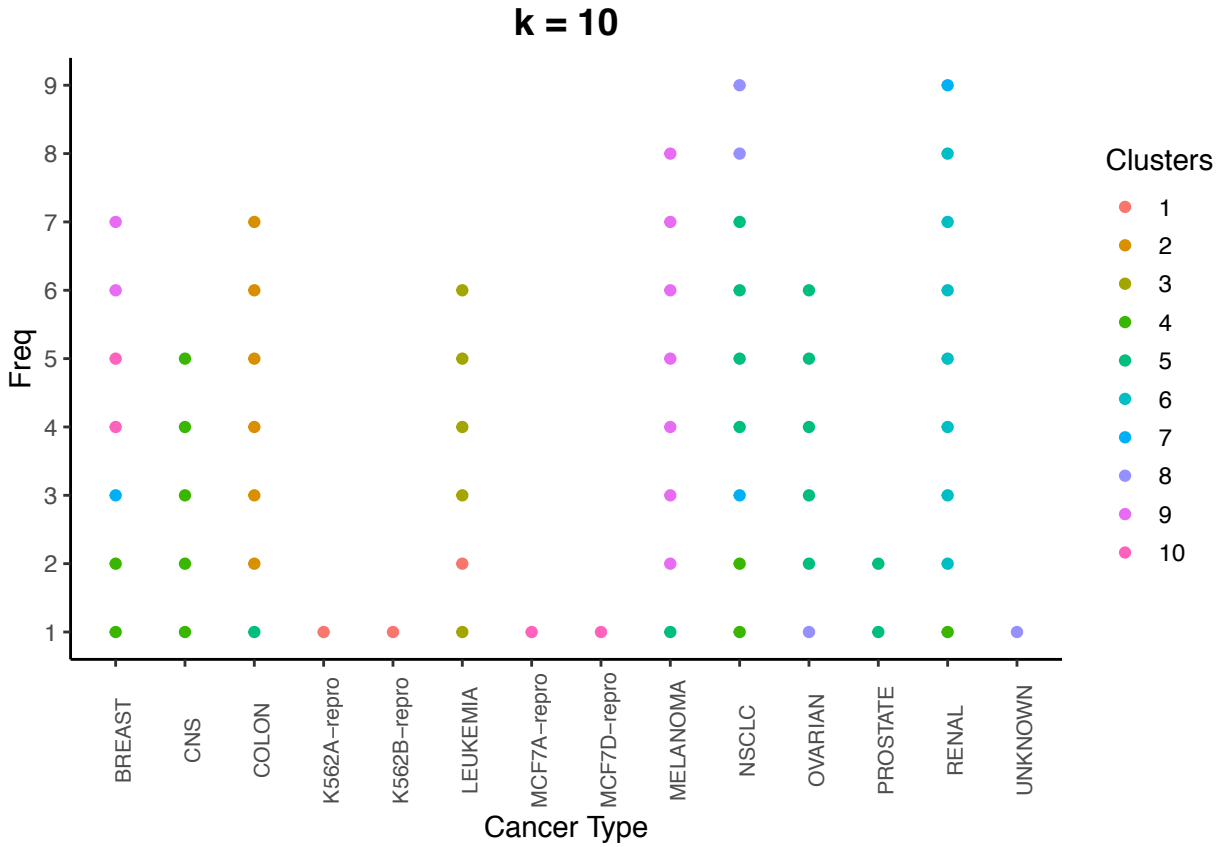
```r
# We know there are 14 different types of cancer
# Lets choose k = even integers between 2 and 18
k <- seq(2,18,2)
for(i in k){
  kmeans_plot(i)
}
```

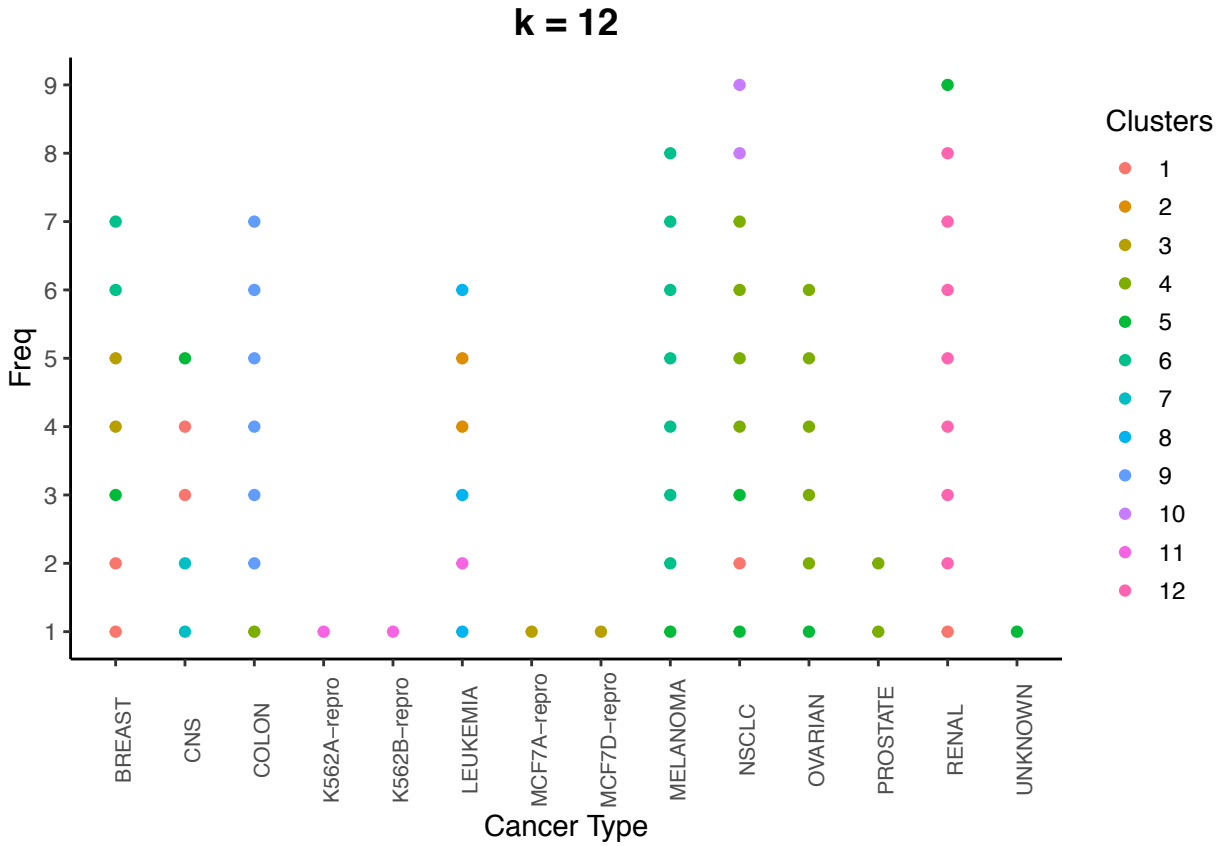

k = 2
Freq
Cancer Type
Clusters
1
2
BREAST
CNS
COLON
K562A−repro
K562B−repro
LEUKEMIA
MCF7A−repro
MCF7D−repro
MELANOMA
NSCLC
OVARIAN
PROSTATE
RENAL
UNKNOWN

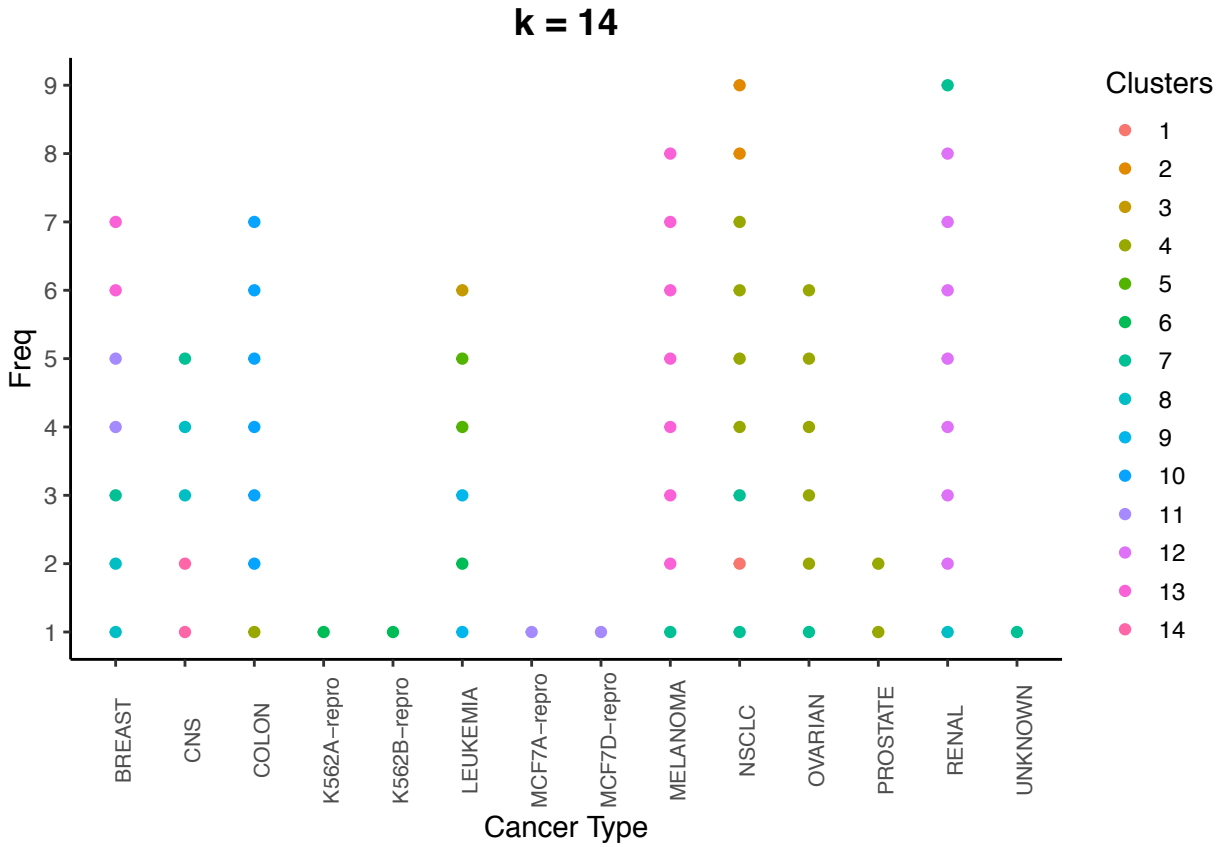

k = 4
Freq
Cancer Type
Clusters
1
2
3
4
BREAST
CNS
COLON
K562A−repro
K562B−repro
LEUKEMIA
MCF7A−repro
MCF7D−repro
MELANOMA
NSCLC
OVARIAN
PROSTATE
RENAL
UNKNOWN

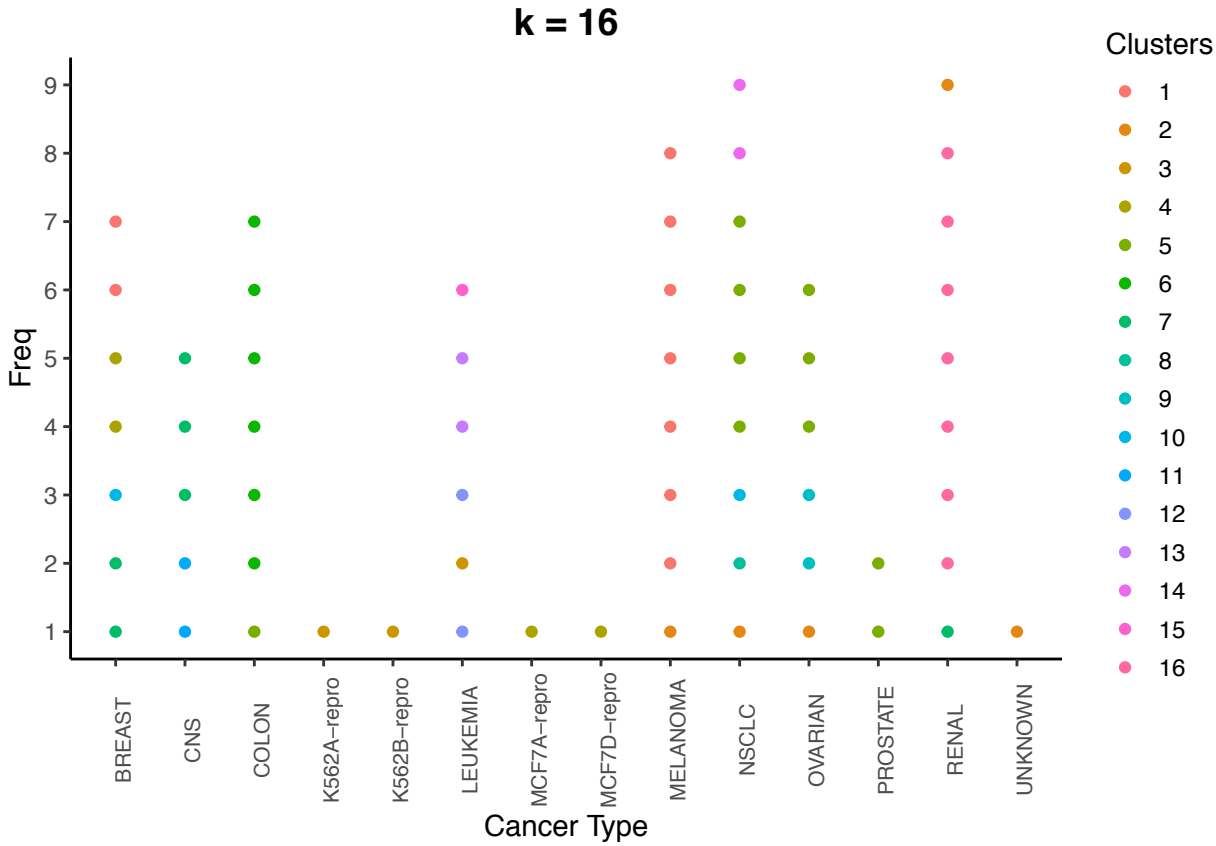

k = 6
Freq
Cancer Type
Clusters
1
2
3
4
5
6
BREAST
CNS
COLON
K562A−repro
K562B−repro
LEUKEMIA
MCF7A−repro
MCF7D−repro
MELANOMA
NSCLC
OVARIAN
PROSTATE
RENAL
UNKNOWN

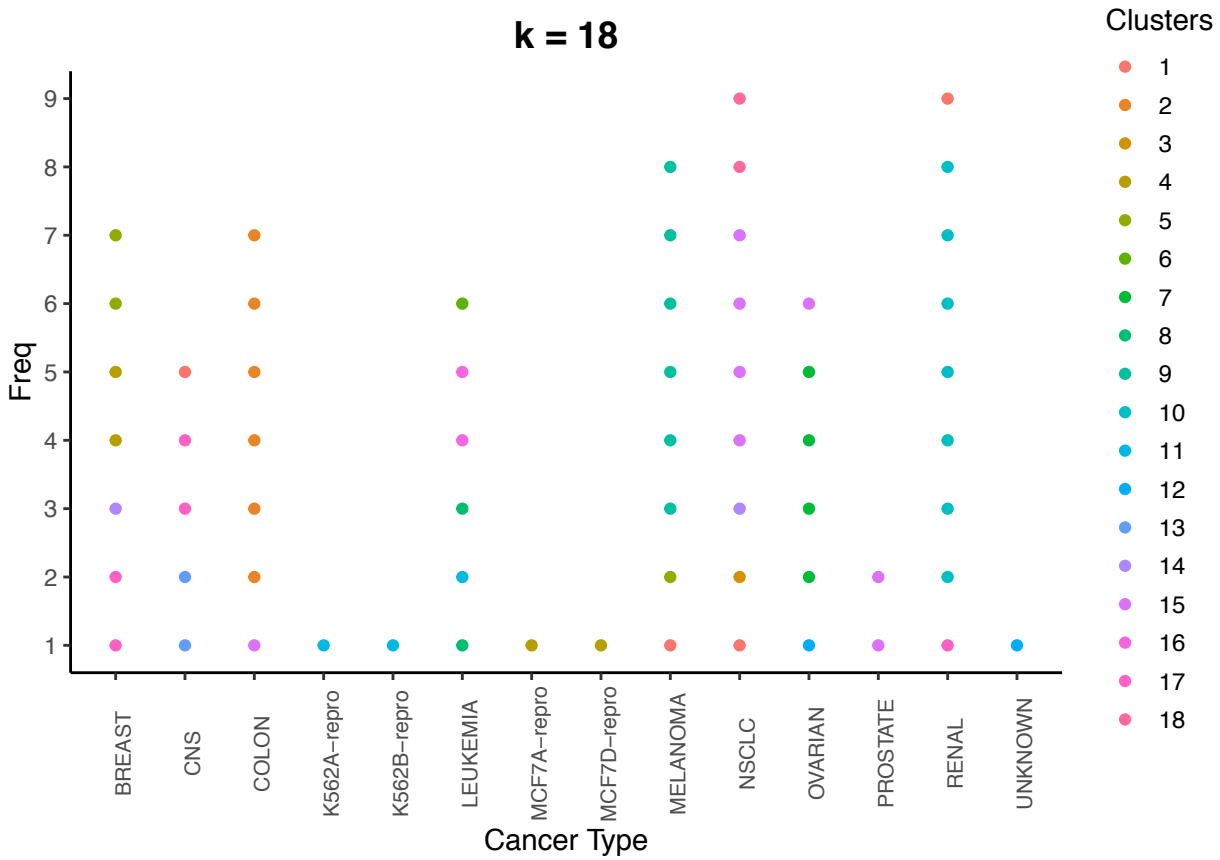

k = 8
Freq
Cancer Type
Clusters
1
2
3
4
5
6
7
8
BREAST
CNS
COLON
K562A−repro
K562B−repro
LEUKEMIA
MCF7A−repro
MCF7D−repro
MELANOMA
NSCLC
OVARIAN
PROSTATE
RENAL
UNKNOWN


k = 10
Freq
Cancer Type
BREAST
CNS
COLON
K562A−repro
K562B−repro
LEUKEMIA
MCF7A−repro
MCF7D−repro
MELANOMA
NSCLC
OVARIAN
PROSTATE
RENAL
UNKNOWN
Clusters
1
2
3
4
5
6
7
8
9
10


k = 12
Freq
Cancer Type
BREAST
CNS
COLON
K562A−repro
K562B−repro
LEUKEMIA
MCF7A−repro
MCF7D−repro
MELANOMA
NSCLC
OVARIAN
PROSTATE
RENAL
UNKNOWN
Clusters
1
2
3
4
5
6
7
8
9
10
11
12


k = 14
Freq
Cancer Type
BREAST
CNS
COLON
K562A−repro
K562B−repro
LEUKEMIA
MCF7A−repro
MCF7D−repro
MELANOMA
NSCLC
OVARIAN
PROSTATE
RENAL
UNKNOWN
Clusters
1
2
3
4
5
6
7
8
9
10
11
12
13
14


k = 16
Freq
Cancer Type
BREAST
CNS
COLON
K562A−repro
K562B−repro
LEUKEMIA
MCF7A−repro
MCF7D−repro
MELANOMA
NSCLC
OVARIAN
PROSTATE
RENAL
UNKNOWN
Clusters
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16



In the plots above, each point represents one of the 64 cancer cells in the data. And they are grouped together by cancer type.

In general we can see that kmeans is successful at grouping together cells of the same cancer. For example, for k small (for example, $k = 2, 4$ even 6) the clusters formed do separate some of the cancers. For example, for $k = 4$, all points corresponding to LEUKEMIA cancer are together in cluster 2, all points corresponding to COLON cancer are in cluster 3, all points corresponding to RENAL cancer are in cluster 4. However, three things to note:

(1) As k changes, cluster memberships change.

(2) There are 14 different types of cancer. So using a k of 2 (or a k "small"), even though you see most of the cells of the same cancer type grouped together you won't see them grouped together with cells corresponding only to that cancer type. For k of 2 you essentially see all cells of type BREAST, CNS, MELANOMA, NSCLC, OVARIAN, PROSTATE, RENAL, and UNKNOWN in one cluster and the cells of the other 5 types in another cluster. If you use a k of 14, you don't see 14 clusters, each one corresponding to cancer type, either though. But with k of 14 kmeans is still able to do a realtively ok job at grouping cells by cancer type. For example, cells corresponding to RENAL almost form their own cluster, as do cells corresponding to COLON or MELANOMA.

(3) Some interesting trends: (1) Some cells of certain cancer types are always grouped together in the same cluster no matter the value of k. For example, from the plots, the middle 7 points (cells) in RENAL are clustered together for all values of k tried. This applies to the top 6 points (cells) in COLON. (2) It looks like certain cells of NSCLC, OVARIAN, and PROSTATE are grouped together for any value of k.

Another note:

You can usually find the optimal value of k. There are different methods. R has a built-in function for the "elbow method." The elbow method is based on minimizing the total within-cluster sum of squares. The total

within-cluster sum of squares measures the compactness of the clustering and we want this to be as small as possible. A kink in the plot of the total within-cluster sum of squares would tell you the optimal k. In this case, there is no clear kink (maybe at $k = 4, 9$?) so it is hard to conclude on an optimal k value.

```
library("factoextra")
```

```
## Welcome! Want to learn more? See two factoextra-related books at https://goo.gl/ve3WBa
```

```
# Elbow method
fviz_nbclust(points, kmeans, method = "wss") +
  labs(subtitle = "Elbow method")
```