Lab_05T

36-600

Fall 2022

To answer the questions below, it will help you to refer to Sections 10.3 and 10.5 of ISLR; it might also help you to refer to your previous lab work (and, as always, to Google).

Question 1

Let's create a fake data frame with three rows and three columns:

```
(df <- data.frame(x=1:3,y=1:3,z=1:3))
```

```
##   x y z
## 1 1 1 1
## 2 2 2 2
## 3 3 3 3
```

Computing *by hand*, what is the Euclidean distances between the fake datum of row 1 and the fake data of rows 2 and 3? And what is the Euclidean distance between the fake data of rows 2 and 3?

```
1 and 2: sqrt(3*(2-1)^2) = sqrt(3) = 1.732
2 and 3: the same
1 and 3: sqrt(3*(3-1)^2) = sqrt(12) = 3.464
```

Question 2

Now compute the Euclidean distances using the `dist()` function. Show the output from this function. Does that output match your hand-computed quantities? If not, go back and think re-do your calculation in Question 1. If so, then you now have a sense as to what the "distance between two data points" is, in practice. Note the appearance of the output: the distances are stored in a lower-triangular matrix.

```
dist(df) # yes, the distances match
```

```
##          1        2
## 2 1.732051
## 3 3.464102 1.732051
```

Dataset 1

Here we import some data on stars either in, or in the same general direction as, the Draco Dwarf Galaxy.

```
file.path <- "https://raw.githubusercontent.com/pefreeman/36-290/master/EXAMPLE_DATASET
S/DRACO/draco_photometry.Rdata"
load(url(file.path))
df <- data.frame(ra,dec,velocity.los,log.g,mag.g,mag.r,mag.i)
rm(file.path,ra,dec,velocity.los,log.g,temperature,mag.u,mag.g,mag.r,mag.i,mag.z,metalli
city,signal.noise)
```

`df` is a data frame with 2778 rows and 7 columns. See this README file (https://github.com/pefreeman/36-290/tree/master/EXAMPLE_DATASETS/DRACO) for a full description of the data and its variables.

Question 3

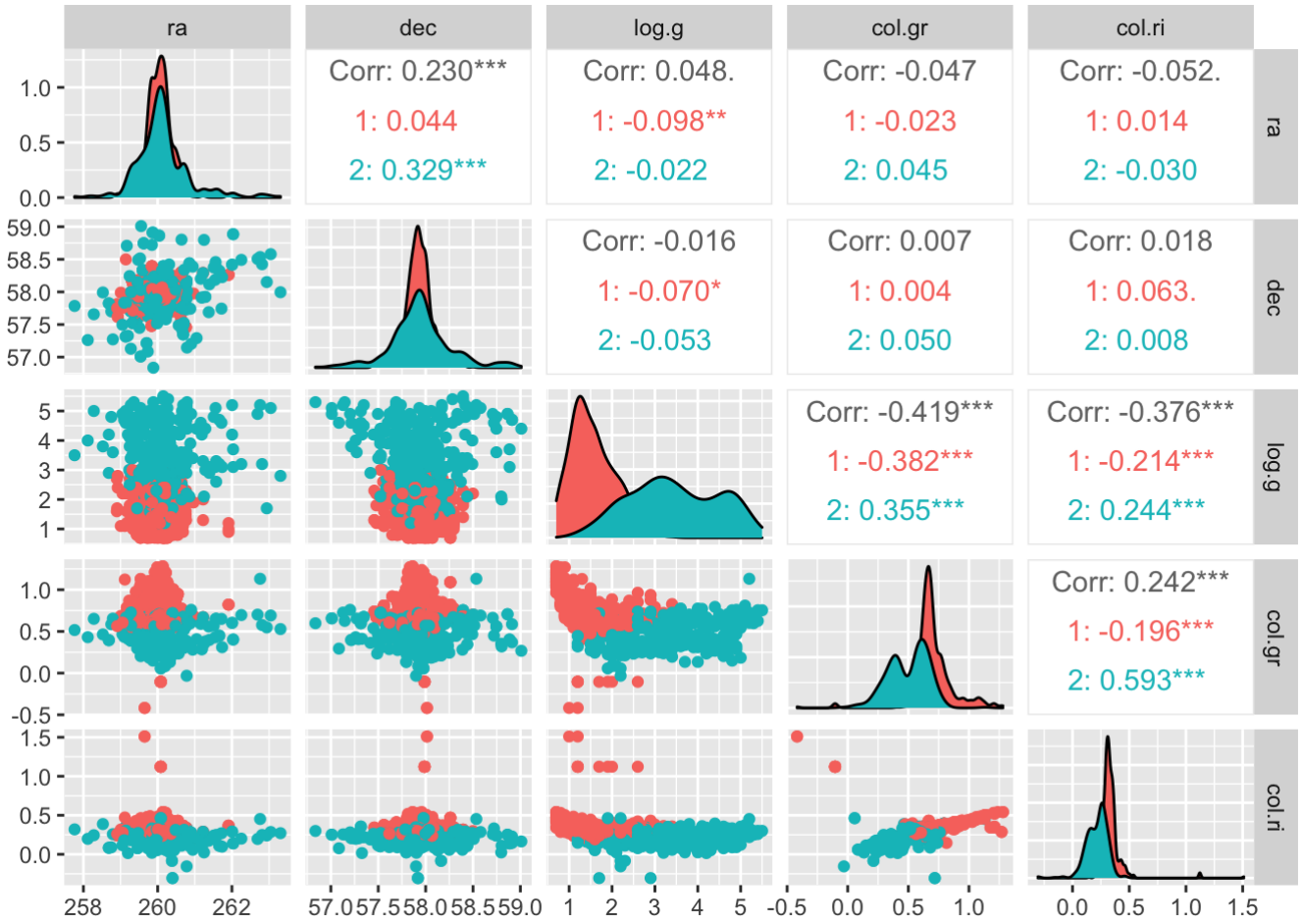
Use `dplyr` functions to filter the data frame such that it only contains values of `dec` > 56, values of `ra` < 264, and values of `velocity.los` between -350 and -250. (Put a space between the "<" and the "-" when you filter on the latter velocity, otherwise `R` will see the assignment operator and act accordingly.) Mutate the data frame to have g-r and r-i colors (call them `gr` and `ri`), then delete the magnitudes and `velocity.los`. (Pro tip: you can "negatively select" columns by putting minus signs in front of the column names.) Save the resulting data frame as `df.new`.

```
suppressMessages(library(tidyverse))
df %>%
  filter(.,ra<264 & dec>56 & velocity.los>-350 & velocity.los< -250) %>%
  mutate(.,col.gr=mag.g-mag.r,col.ri=mag.r-mag.i) %>%
  select(.,-mag.g,-mag.r,-mag.i,-velocity.los) -> df.new
```

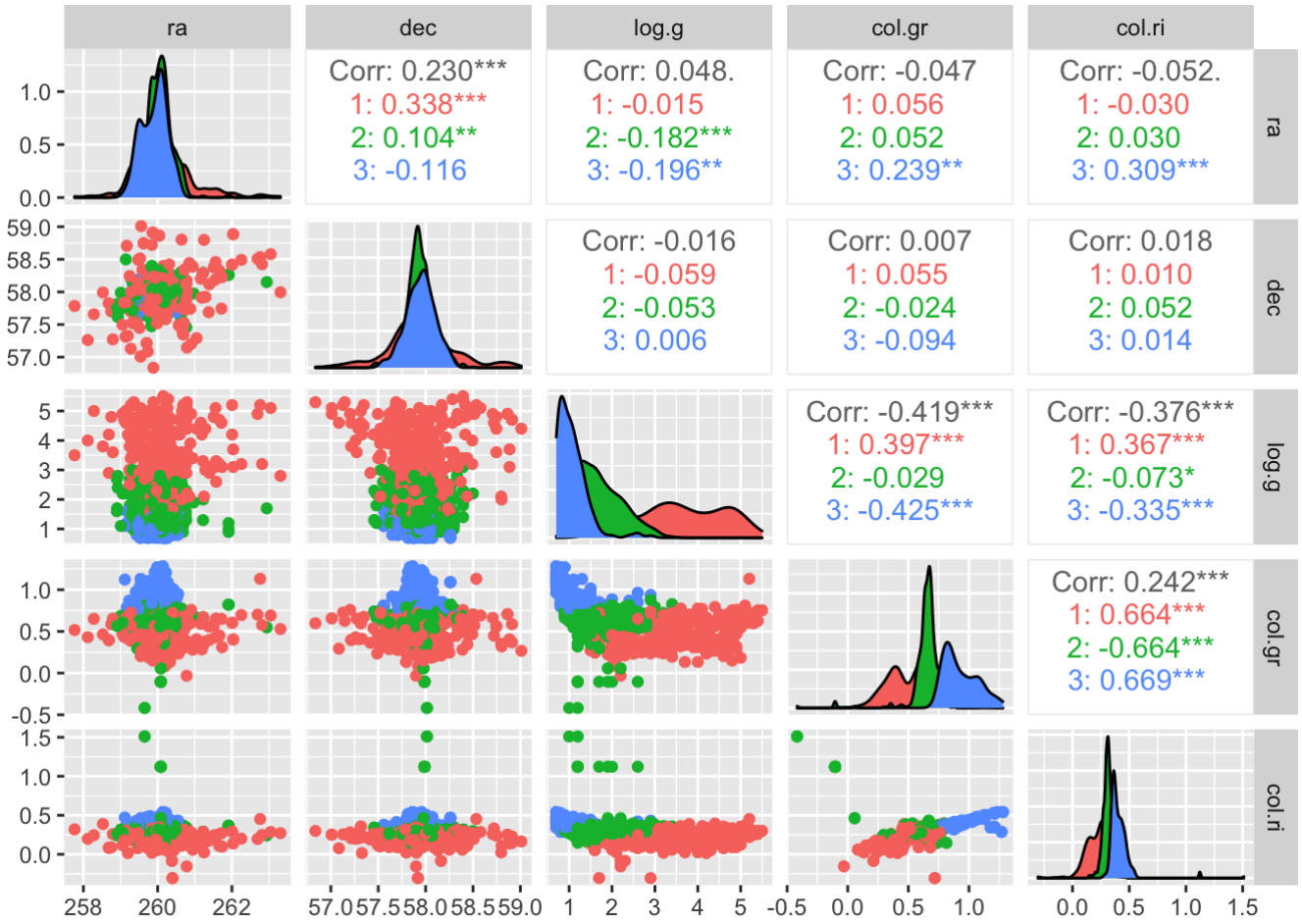
Question 4

Use the `kmeans()` function to cluster the data in your data frame. Try different values for k, and finally display results for what *you* would choose as its optimal value. The default for `nstart` is 1; that should be increased to something larger…play with the values for this argument. Display the results using `ggpairs()`. Pass this argument to `ggpairs()`: `mapping=aes(color=factor(km.out$cluster))`, where `km.out` is the output from K-means, and `cluster` is the number of the cluster to which a datum has been assigned. Remember to `scale()` your data! Also, note that `kmeans()` utilizes random sampling, so you should absolutely set a random number seed immediately before calling `kmeans()` to ensure reproducibility!

```
suppressMessages(library(GGally))
# To be clear, students only have to show one plot for one choice of K.
set.seed(101)
km2.out <- kmeans(scale(df.new),2,50)
ggpairs(df.new,progress=FALSE,mapping=aes(color=factor(km2.out$cluster)))
```




```
km3.out <- kmeans(scale(df.new),3,50)
ggpairs(df.new,progress=FALSE,mapping=aes(color=factor(km3.out$cluster)))
```



Question 5

For your final run of K-means, what are the number of groups and the number of data in each group? Also, what is ratio of the between-cluster sum-of-squares to the total sum-of-squares? (This is a measure of the total variance in the data that is "explained" by clustering. Higher values [closer to 100%] are better, but beware: the larger the value of K, the higher the ratio is going to be: you will be getting into the realm of overfitting.) (Hint: `print()` your saved output from `kmeans()` .)

```
print(km2.out)
```

```
## K-means clustering with 2 clusters of sizes 870, 348
##
## Cluster means:
##            ra         dec      log.g     col.gr     col.ri
## 1 -0.06560733 -0.01727773 -0.4876261  0.3182786  0.2972189
## 2  0.16401832  0.04319432  1.2190653 -0.7956966 -0.7430474
##
## Clustering vector:
##    [1] 2 2 1 1 2 1 2 1 1 1 1 1 1 1 1 2 1 1 2 1 1 1 1 1 1 2 1 2 1 1 2 1 1 1 1 1 1 1 1
1 1 1
##   [43] 1 1 1 1 2 1 1 1 1 1 1 1 1 1 2 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 1 1
2 1 1
##   [85] 1 1 1 1 2 1 1 1 1 1 1 1 1 1 1 2 1 2 1 2 1 1 1 1 2 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1
##  [127] 1 1 1 1 1 2 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1
##  [169] 1 1 2 2 2 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 1 1 1 1 1 1 1 1
1 1 1
##  [211] 2 1 1 1 1 1 1 1 1 2 1 1 1 1 1 1 1 2 1 1 2 1 2 1 1 1 2 1 1 1 1 1 1 1 2 1 2 1 1
1 1 1
##  [253] 1 1 1 1 1 1 1 1 2 1 1 1 1 2 2 1 1 1 1 1 1 2 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1
##  [295] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 1 1 1 1 1 1 1 1 1 1 1 2 1 1 1 1 2 1 1 1 2 1 1 1
1 1 1
##  [337] 1 1 1 1 1 2 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 1 2 1 1 2 2 1 1 1 1
1 1 1
##  [379] 1 1 1 1 2 1 2 2 1 1 1 1 1 2 1 2 1 1 1 2 1 1 1 1 1 1 2 2 2 1 1 1 1 1 1 1 1 1 1
1 2 2
##  [421] 1 2 2 1 1 1 1 1 1 1 1 1 1 1 1 2 1 1 1 1 1 1 1 1 1 2 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 2
##  [463] 2 2 1 1 1 1 2 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1
##  [505] 1 1 1 1 1 1 1 1 1 1 1 2 2 2 1 1 1 2 2 2 2 2 2 2 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1
##  [547] 1 1 2 1 2 2 2 2 2 1 1 1 1 1 1 1 1 1 1 1 2 2 1 2 2 1 1 1 1 1 1 1 1 1 1 1 2 1 1
1 2 1
##  [589] 1 1 1 1 1 1 1 1 1 1 1 1 2 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1
##  [631] 1 1 2 1 2 1 1 2 1 1 2 2 2 2 1 1 1 1 1 1 1 1 1 2 1 1 1 1 1 2 1 1 1 1 1 1 1 1 1
1 1 1
##  [673] 1 1 1 1 1 1 1 1 2 1 2 2 2 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 1 1
2 1 1 1
##  [715] 1 1 1 1 1 1 1 1 2 2 1 1 1 1 1 1 1 1 1 1 1 1 2 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 2 2 2
##  [757] 2 2 1 2 1 1 1 1 1 1 2 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 1 1 1 1 1 1 1 1 1 1 1 2 2
1 1 1 1
##  [799] 1 1 1 1 1 1 1 1 1 2 2 1 1 2 1 2 1 2 1 1 1 1 1 1 1 1 1 1 2 1 1 2 2 2 2 1 1 2 1
2 2 2 1
##  [841] 1 1 2 2 1 1 1 1 2 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2 1 1 1 1 1 2 2 1 1 1 1 1 1 1 1
2 2 2 1
##  [883] 1 1 2 2 2 1 2 2 1 1 2 1 1 1 1 2 2 2 2 2 2 2 1 1 2 2 2 1 2 2 2 1 1 1 2 1 2 2 2
2 2 2 2
```

```
##  [925] 2 2 2 2 2 2 1 2 2 2 2 2 1 1 2 1 2 2 2 2 2 2 2 2 1 1 1 1 1 2 1 1 1 1 1 2 1 2 2
2 2 2
##  [967] 2 2 1 1 1 1 2 2 2 2 2 2 2 2 2 1 1 1 1 2 2 1 1 1 1 1 1 1 1 2 2 2 1
##  [ reached getOption("max.print") -- omitted 218 entries ]
##
## Within cluster sum of squares by cluster:
## [1] 2421.503 2347.989
##  (between_SS / total_SS =  21.6 %)
##
## Available components:
##
## [1] "cluster"      "centers"      "totss"        "withinss"     "tot.withinss"
## [6] "betweenss"    "size"         "iter"         "ifault"
```

```
print(km3.out)
```

```
## K-means clustering with 3 clusters of sizes 300, 739, 179
##
## Cluster means:
##             ra         dec      log.g      col.gr     col.ri
## 1  0.170740835  0.04361334  1.3717092 -0.81913484 -0.8149672
## 2  0.001693028 -0.02791883 -0.3310594 -0.02460787  0.1321051
## 3 -0.293147475  0.04216768 -0.9321781  1.47444506  0.8204721
##
## Clustering vector:
##    [1] 1 1 2 2 1 2 1 3 3 2 3 3 3 3 2 1 2 2 1 2 3 2 2 2 2 1 2 2 2 2 2 2 2 2 2 2 2 2
3 2 2
##   [43] 2 2 2 2 1 2 2 2 2 2 2 2 2 2 1 3 3 3 3 3 3 2 2 2 2 2 2 2 2 2 2 1 1 1 1 1 2 2
1 2 2
##   [85] 2 2 2 2 1 2 2 2 2 2 2 2 2 2 2 1 2 2 2 1 2 2 2 2 2 2 3 3 3 3 3 3 2 2 2 3 2 2 3
3 3 3
##  [127] 3 2 2 3 2 1 2 2 2 3 3 2 2 2 2 2 2 2 3 3 3 3 3 3 3 2 2 3 2 2 2 3 3 3 2 2 2 2 3
3 2 2
##  [169] 3 3 1 1 2 2 2 2 2 2 2 2 2 2 2 2 2 2 3 2 2 2 2 2 2 2 2 2 2 2 2 2 1 2 2 2 2 2 2 2
2 2 2
##  [211] 1 2 2 2 2 2 2 2 2 2 1 2 2 2 2 2 2 2 2 1 2 2 1 2 1 2 2 2 1 2 2 2 2 2 2 2 2 2 2 2
2 3 3
##  [253] 2 2 2 2 2 2 2 2 3 1 3 2 3 3 1 1 3 3 3 3 2 3 2 2 2 2 2 2 2 2 3 3 3 3 2 2 2 2 2 2
2 2 3
##  [295] 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 1 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 1 2 2 2 1 2 2
3 2 2
##  [337] 2 2 3 2 2 1 2 2 2 2 3 2 2 3 3 3 3 3 2 2 2 2 2 2 2 2 2 2 2 2 2 2 1 2 2 1 1 2 2 2
2 3 3
##  [379] 2 2 2 2 2 2 2 1 2 2 2 2 2 2 1 2 1 2 2 2 1 2 2 2 2 2 2 1 1 1 2 2 2 2 3 3 3 3 3 3
3 1 1
##  [421] 2 1 1 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 1 2 2 2 2 2 2 2 2 2 2 2 2
2 2 1
##  [463] 2 1 2 2 2 2 2 2 2 2 3 3 2 3 3 3 3 3 3 3 3 2 2 2 2 2 2 2 2 2 2 2 3 3 3 3 3 3 3
2 2 2
##  [505] 2 2 2 3 3 2 3 3 3 3 3 1 1 1 2 2 2 1 2 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2 2 2 3 2 2
2 2 2
##  [547] 2 3 1 2 1 1 1 1 1 2 2 2 3 3 3 3 2 2 2 3 3 1 1 2 1 1 3 3 3 3 3 3 3 3 3 3 3 1 3
3 2 2
##  [589] 2 2 2 2 2 2 2 2 2 3 3 2 2 2 2 3 3 2 2 3 3 3 3 3 3 3 3 3 2 2 2 2 2 2 2 3 3 3 3
3 3 2
##  [631] 2 2 1 2 1 2 2 1 2 2 1 1 1 1 3 3 3 3 3 2 2 3 2 1 3 3 3 2 3 2 2 2 2 2 2 2 2 2 2
2 2 2
##  [673] 3 3 3 2 2 3 3 2 2 1 1 1 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 1 2 2 2
2 2 2
##  [715] 2 2 2 2 2 2 2 2 2 1 3 3 2 3 2 2 2 2 2 2 2 2 2 1 2 2 2 2 2 2 2 2 2 2 2 2 2 3 2 2 2
1 1 1
##  [757] 2 1 2 2 2 2 2 2 2 2 2 1 2 2 2 2 2 2 2 2 2 2 2 2 2 1 1 1 2 2 2 2 2 2 2 2 2 2 2 1 2
2 2 2
##  [799] 2 2 2 2 2 2 2 2 2 1 1 2 2 1 2 1 3 1 2 2 2 2 2 2 2 2 2 2 1 2 2 1 1 1 1 2 2 1 3 1
1 1 2
##  [841] 2 2 1 1 2 2 2 2 2 1 2 2 2 2 2 2 2 2 2 2 2 2 2 1 1 1 1 2 2 3 3 2 1 1 2 2 2 2 2 2 2
1 1 2
##  [883] 2 2 1 1 1 3 1 1 3 3 1 2 2 2 2 1 1 1 1 1 1 1 2 2 1 1 1 2 1 1 1 2 2 2 2 2 1 1 1
```

```
1 1 1
##  [925] 1 1 1 1 1 1 2 1 1 1 1 1 2 3 1 2 1 1 1 1 1 1 1 1 2 2 2 2 2 1 2 2 2 2 2 1 2 1 1
1 1 1
##  [967] 1 1 2 2 2 2 2 1 2 2 1 1 1 1 1 2 2 2 2 2 1 2 2 2 2 2 2 2 2 2 1 1 2 2
##  [ reached getOption("max.print") -- omitted 218 entries ]
##
## Within cluster sum of squares by cluster:
## [1] 2152.8616 1857.6937  324.3033
##  (between_SS / total_SS =  28.8 %)
##
## Available components:
##
## [1] "cluster"      "centers"      "totss"        "withinss"     "tot.withinss"
## [6] "betweenss"    "size"         "iter"         "ifault"
```

```
If I set K=2, then (348,870), 21.6%; for K=3: (300,739,179), 28.8%. Everyone's numbers w
ill differ.
```

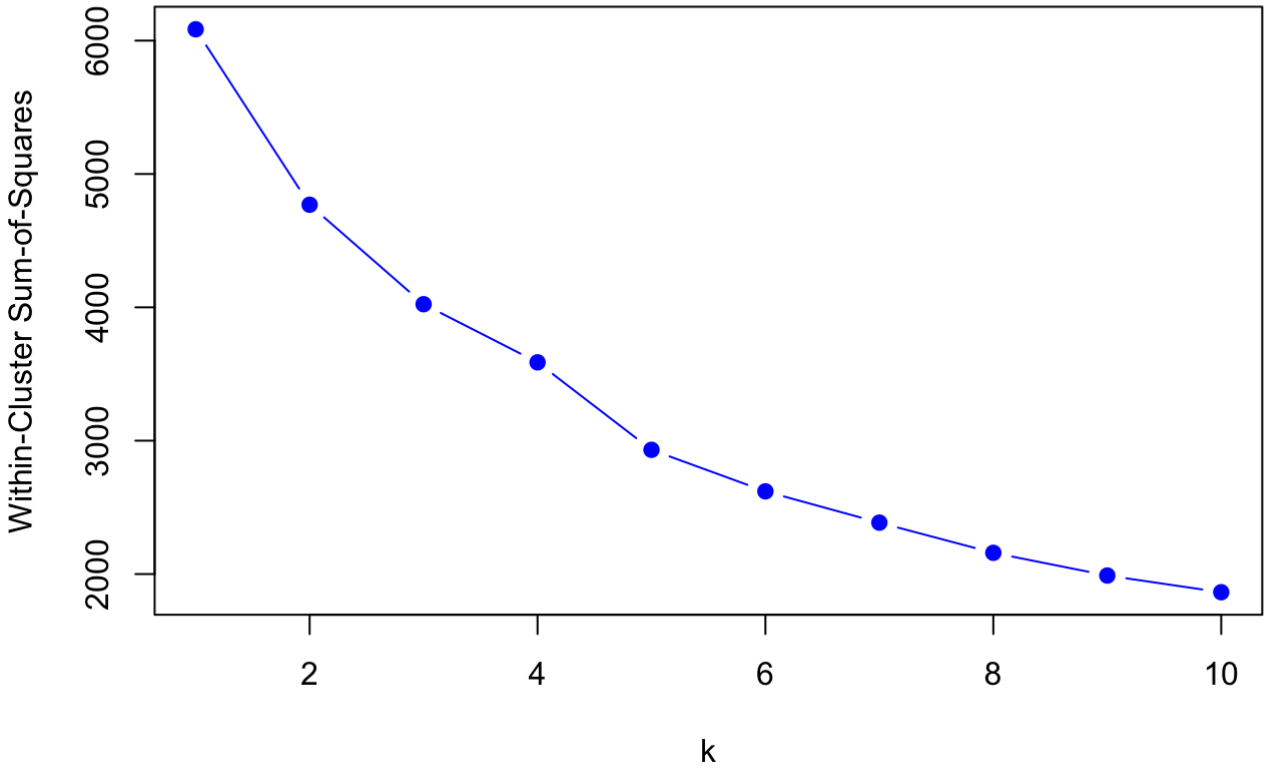
Question 6

Now utilize one of the methods presented in class for determinining the optimal value of k, while noting that some methods provide more concrete results than others.

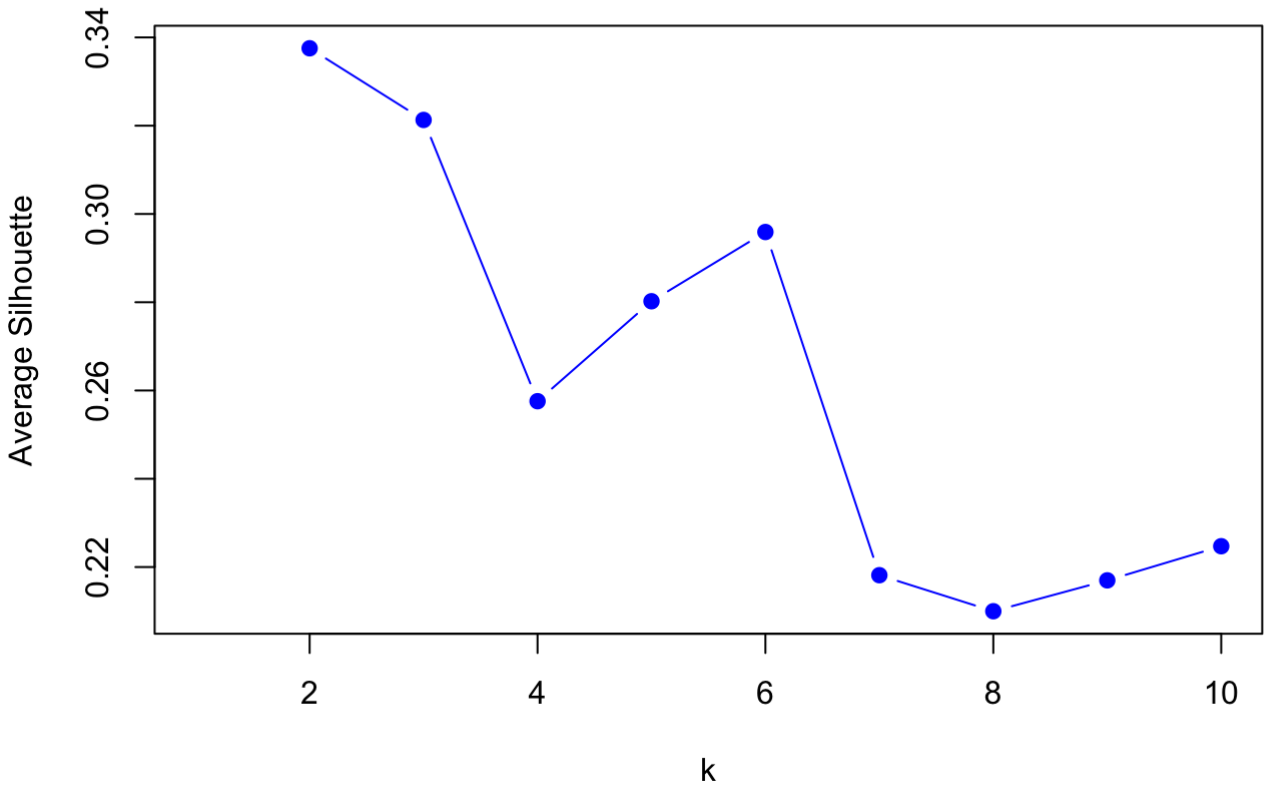
```
# Elbow
wss <- rep(NA,10)
for ( ii in 1:10 ) {
  km.out <- kmeans(scale(df.new),ii,nstart=20);
  wss[ii] <- km.out$tot.withinss;
}
```

```
## Warning: did not converge in 10 iterations

## Warning: did not converge in 10 iterations
```

```
plot(1:10,wss,xlab="k",ylab="Within-Cluster Sum-of-Squares",pch=19,col="blue",typ="b")
```




```
# ...there is no really obvious elbow here!

# Silhouette
library(cluster)
ss <- rep(NA,10)
for ( ii in 2:10 ) {
  km.out <- kmeans(scale(df.new),ii,nstart=20);
  ss[ii] <- mean(silhouette(km.out$cluster,dist(scale(df.new)))[,3]);
}
plot(2:10,ss[2:10],xlab="k",ylab="Average Silhouette",pch=19,col="blue",typ="b",xlim=c(1
,10))
```




```
# ...k=2

# Gap Statistic
suppressMessages(library(factoextra))
gs <- clusGap(scale(df.new),FUN=kmeans,nstart=20,K.max=10,B=50)
```

```
## Clustering k = 1,2,..., K.max (= 10): ..
```

```
## Warning: did not converge in 10 iterations
```

```
## done
## Bootstrapping, b = 1,2,..., B (= 50)  [one "." per sample]:
## ....
```

```
## Warning: did not converge in 10 iterations
```

```
## ......................
```

```
## Warning: did not converge in 10 iterations
```

```
## ....
```

```
## Warning: did not converge in 10 iterations
```

```
## .......
```

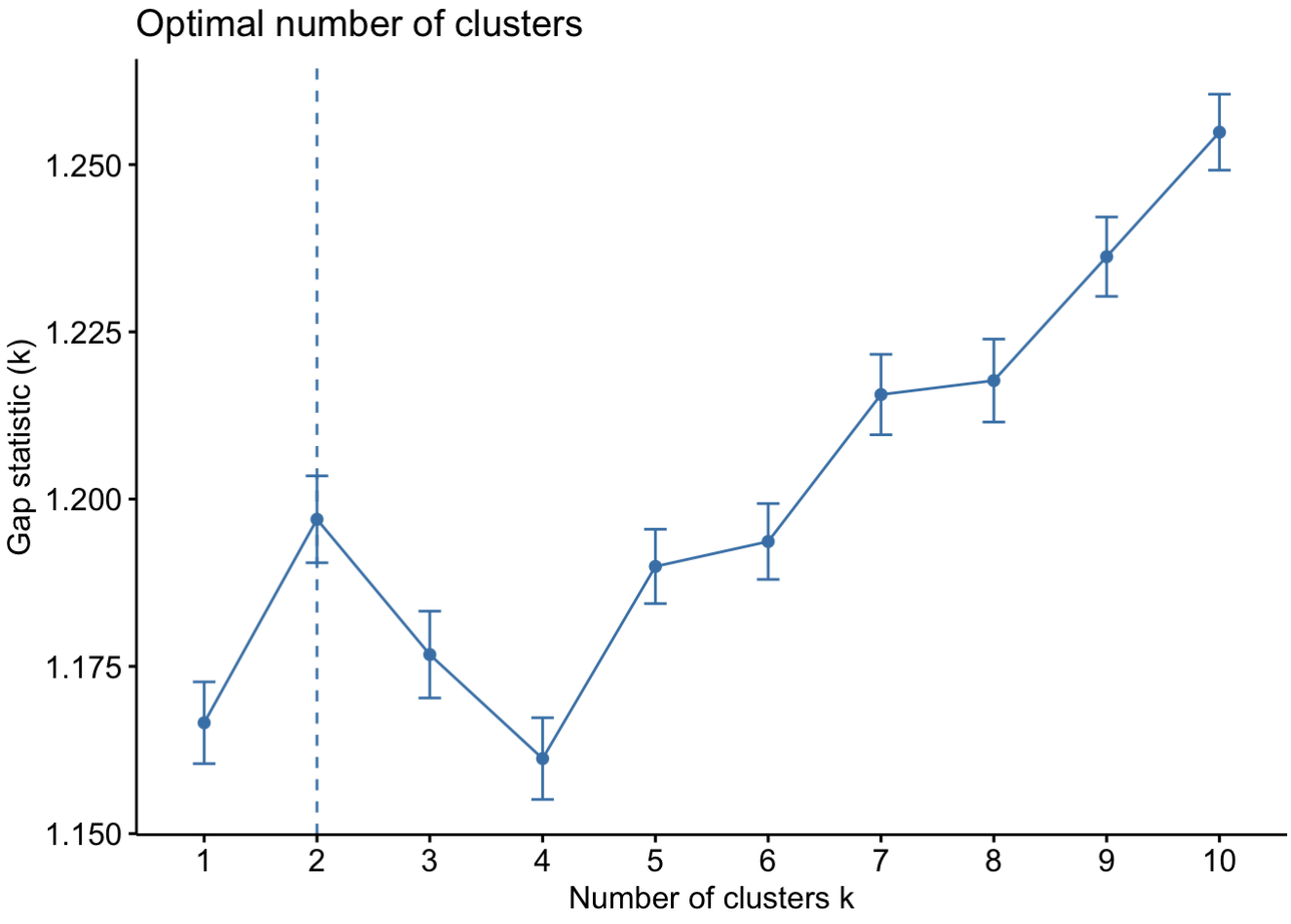
```
## Warning: did not converge in 10 iterations
```

```
## ............
```

```
## Warning: did not converge in 10 iterations
```

```
## .. 50
```

```
fviz_gap_stat(gs)
```




```
# ...k=2...although this is not the absolute maximum value (but parsimony matters!)
```

Dataset 2

Question 7

Now input the diamonds dataset (`diamonds.csv`); download it from the `DATA` directory on `Canvas` . Once the data frame is input, remove the columns `X` , `cut` , `color` , and `clarity` from it, and call the resulting data frame `df.new` .

```
df <- read.csv("diamonds.csv",stringsAsFactors=TRUE)
df %>% select(.,-X,-cut,-color,-clarity) -> df.new
```

Question 8

The diamonds dataset is too large to input into K-means as is. (I have discovered this myself, to my sorrow!) So we will randomly select 1000 rows from the data frame and work with those. First, run this code (while noting that you can change the seed to whatever you like):

```
set.seed(303)
s <- sort(sample(nrow(df.new),1000))
```

Then utilize the `slice()` function from `dplyr` to select the rows recorded in the vector `s` , and save the output to `df.small` .

```
set.seed(303)
s <- sort(sample(nrow(df.new),1000))
df.new %>% slice(.,s) -> df.small
```

Question 9

Now we can do K-means. But we'll do things a little differently this time: here, we will first use the gap statistic to estimate the optimal value of k. Keep that value in mind for the next question below.

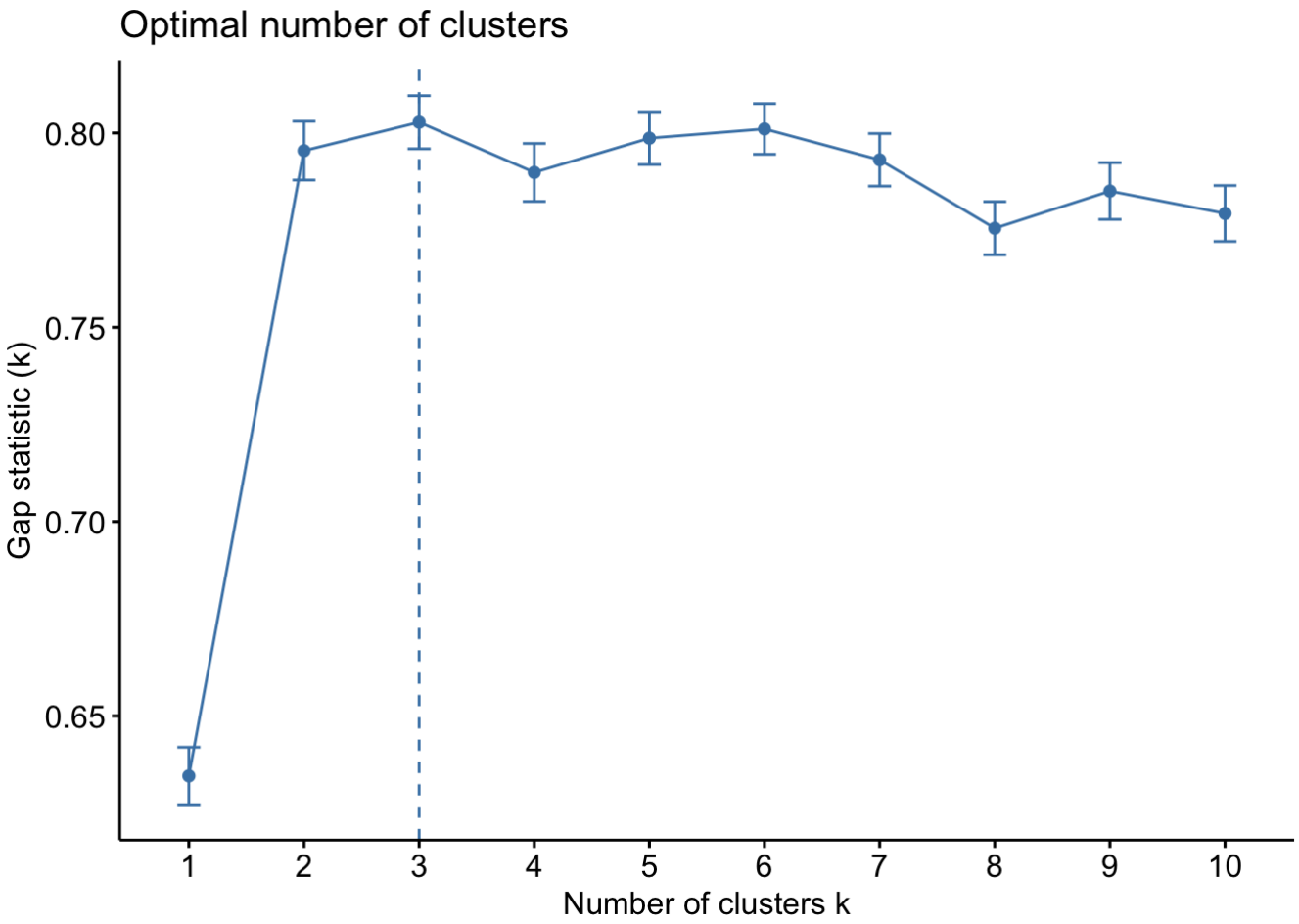
```
set.seed(202)
gs <- clusGap(scale(df.small),FUN=kmeans,nstart=20,K.max=10,B=50)
```

```
## Clustering k = 1,2,..., K.max (= 10): .. done
## Bootstrapping, b = 1,2,..., B (= 50)  [one "." per sample]:
## ..................
```

```
## Warning: did not converge in 10 iterations
```

```
## ............................... 50
```

```
fviz_gap_stat(gs)  # Note...some people may get k=2 or other values...it depends on the
 seed!
```

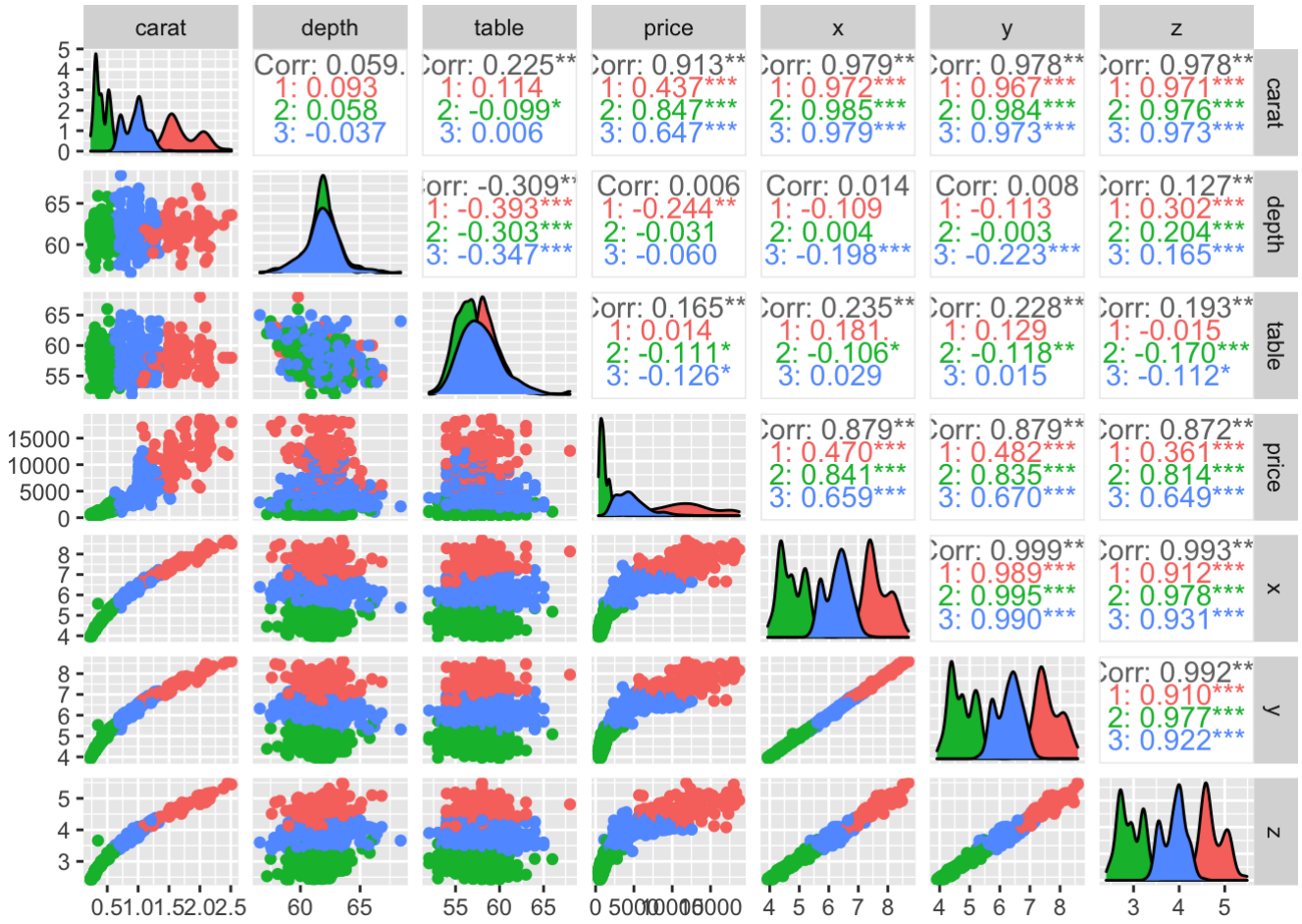


Question 10

In Question 4, we explored using different values of k, and through the use of `ggpairs()` we determined that for the stellar data, k should be 2 or 3. In Question 8, on the other hand, we let the gap statistic function determine the optimal value for k...and now here we will use `ggpairs()` to visualize the results of assuming this optimal value only. Do this below.

You need not state an answer to this question, but: does it appear that there are natural clusters in the data that were uncovered using K-means, or does it appear that the data were simply split into pieces? (Look at, e.g., `price` versus `carat`; that scatter plot *might* influence your thinking. Note that there not necessarily a right answer here. Interpretation, meet ambiguity.)

```
set.seed(202)
km.out <- kmeans(scale(df.small),3,20)
ggpairs(df.small,progress=FALSE,mapping=aes(color=factor(km.out$cluster)))
```


carat
depth
table
price
x
y
z
Corr: 0.059.
1: 0.093
2: 0.058
3: -0.037
Corr: 0.225**
1: 0.114
2: -0.099*
3: 0.006
Corr: 0.913**
1: 0.437***
2: 0.847***
3: 0.647***
Corr: 0.979**
1: 0.972***
2: 0.985***
3: 0.979***
Corr: 0.978**
1: 0.967***
2: 0.984***
3: 0.973***
Corr: 0.978**
1: 0.971***
2: 0.976***
3: 0.973***
Corr: -0.309**
1: -0.393***
2: -0.303***
3: -0.347***
Corr: 0.006
1: -0.244**
2: -0.031
3: -0.060
Corr: 0.014
1: -0.109
2: 0.004
3: -0.198***
Corr: 0.008
1: -0.113
2: -0.003
3: -0.223***
Corr: 0.127**
1: 0.302***
2: 0.204***
3: 0.165***
Corr: 0.165**
1: 0.014
2: -0.111*
3: -0.126*
Corr: 0.235**
1: 0.181.
2: -0.106*
3: 0.029
Corr: 0.228**
1: 0.129
2: -0.118**
3: 0.015
Corr: 0.193**
1: -0.015
2: -0.170***
3: -0.112*
Corr: 0.879**
1: 0.470***
2: 0.841***
3: 0.659***
Corr: 0.879**
1: 0.482***
2: 0.835***
3: 0.670***
Corr: 0.872**
1: 0.361***
2: 0.814***
3: 0.649***
Corr: 0.999**
1: 0.989***
2: 0.995***
3: 0.990***
Corr: 0.993**
1: 0.912***
2: 0.978***
3: 0.931***
Corr: 0.992**
1: 0.910***
2: 0.977***
3: 0.922***