Assignment 5

Due at 11:59pm on November 25.

You may work in pairs or individually for this assignment. Make sure you join a group in Canvas if you are working in pairs. Turn in this assignment as an HTML or PDF file to ELMS. Make sure to include the R Markdown or Quarto file that was used to generate it. Include the GitHub link for the repository containing these files.

Working directory at: https://github.com/TroyLiuUofM/zeng1-liu2-a5.git

```r
library(censusapi)
```

```
Warning: package 'censusapi' was built under R version 4.3.3
```

```r
library(tidyverse)
```

```
Warning: package 'ggplot2' was built under R version 4.3.3
```

```
Warning: package 'tibble' was built under R version 4.3.3
```

```
Warning: package 'purrr' was built under R version 4.3.3
```

```
-- Attaching core tidyverse packages ------------------------ tidyverse 2.0.0 --
v dplyr     1.1.4     v readr     2.1.4
v forcats   1.0.0     v stringr   1.5.1
v ggplot2   3.5.2     v tibble    3.3.0
v lubridate 1.9.2     v tidyr     1.3.0
v purrr     1.0.4
-- Conflicts ------------------------------------------ tidyverse_conflicts() --
x dplyr::filter() masks stats::filter()
x dplyr::lag()    masks stats::lag()
i Use the conflicted package (<http://conflicted.r-lib.org/>) to force all conflicts to becom
```

```
library(magrittr)
```

```
Warning: package 'magrittr' was built under R version 4.3.3
```

```

Attaching package: 'magrittr'

The following object is masked from 'package:purrr':

    set_names

The following object is masked from 'package:tidyr':

    extract
```

```
library(factoextra)
```

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

Exploring ACS Data

In this notebook, we use the Census API to gather data from the American Community Survey (ACS). This requires an access key, which can be obtained here:

https://api.census.gov/data/key_signup.html

```
cs_key = "55d7b56135bd20c3d94c7f4b56d448aa88b66406" #Troy please don't leak my key, lol
acs_il_c <- getCensus(name = "acs/acs5",
                    vintage = 2016,
                    vars = c("NAME",
                             "B01003_001E",
                             "B19013_001E",
                             "B19301_001E"),
                    region = "county:*",
                    regionin = "state:17",
                    key = cs_key) %>%
            rename(pop = B01003_001E,
                   hh_income = B19013_001E,
                   income = B19301_001E)
head(acs_il_c)
```

```
  state county                        NAME    pop hh_income income
1    17    067   Hancock County, Illinois  18633     50077  25647
2    17    063    Grundy County, Illinois  50338     67162  30232
3    17    091  Kankakee County, Illinois 111493     54697  25111
4    17    043    DuPage County, Illinois 930514     81521  40547
5    17    003 Alexander County, Illinois   7051     29071  16067
6    17    129    Menard County, Illinois  12576     60420  31323
```

Pull map data for Illinois into a data frame.

```r
il_map <- map_data("county", region = "illinois")
head(il_map)
```

```
       long      lat group order   region subregion
1 -91.49563 40.21018     1     1 illinois     adams
2 -90.91121 40.19299     1     2 illinois     adams
3 -90.91121 40.19299     1     3 illinois     adams
4 -90.91121 40.10704     1     4 illinois     adams
5 -90.91121 39.83775     1     5 illinois     adams
6 -90.91694 39.75754     1     6 illinois     adams
```

Join the ACS data with the map data. Not that `il_map` has a column `subregion` which includes county names. We need a corresponding variable in the ACS data to join both data sets. This needs some transformations, among which the function `tolower()` might be useful. Call the joined data `acs_map`.

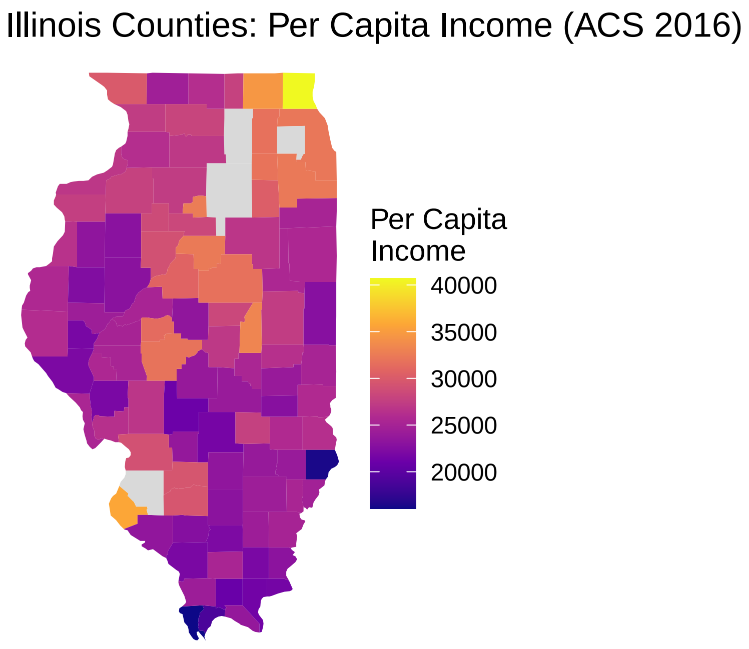
```r
# Pull county name from NAME, e.g. "Cook County, Illinois" -> "cook"
acs_il_c <- acs_il_c %>%
  mutate(
    county_name = str_to_lower(str_replace(str_extract(NAME, "[^,]+ County"), " County", ""))
  )

acs_map <- il_map %>%
  left_join(acs_il_c, by = c("subregion" = "county_name"))

# quick check
acs_map %>% select(subregion, pop, income) %>% head()
```

```
  subregion   pop income
1     adams 66949  26053
2     adams 66949  26053
```

```
3      adams 66949  26053
4      adams 66949  26053
5      adams 66949  26053
6      adams 66949  26053
```

After you do this, plot a map of Illinois with Counties colored by per capita income.

```
ggplot(acs_map) +
  geom_polygon(aes(long, lat, group = group, fill = income), color = NA) +
  coord_quickmap() +
  scale_fill_viridis_c(option = "plasma", na.value = "grey85") +
  labs(title = "Illinois Counties: Per Capita Income (ACS 2016)",
       fill  = "Per Capita\nIncome") +
  theme_void()
```

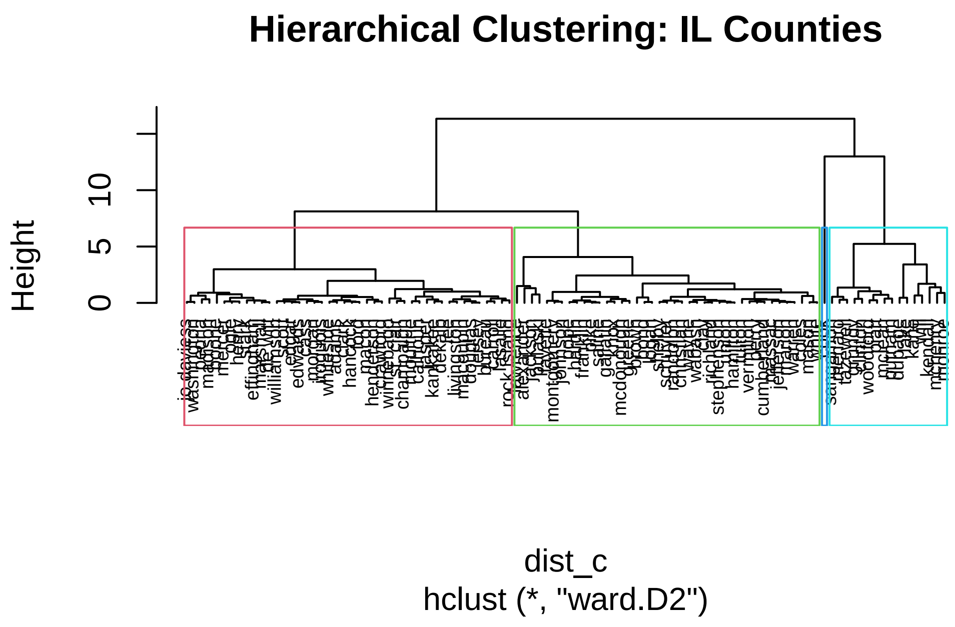


Hierarchical Clustering

We want to find clusters of counties that are similar in their population, average household income and per capita income. First, clean the data so that you have the appropriate variables to use for clustering. Next, create the distance matrix of the cleaned data. This distance matrix can be used to cluster counties, e.g. using the ward method.

Plot the dendrogram to find a reasonable number of clusters. Draw boxes around the clusters of your cluster solution.

```r
#Clean data
clust_dat_c <- acs_il_c %>%
  select(subregion = county_name, pop, hh_income, income) %>%
  drop_na()

rownames(clust_dat_c) <- clust_dat_c$subregion
Xc <- scale(clust_dat_c %>% select(pop, hh_income, income))   # standardize

dist_c <- dist(Xc, method = "euclidean")
hc_c   <- hclust(dist_c, method = "ward.D2")

# dendrogram
plot(hc_c, cex = .6, hang = -1, main = "Hierarchical Clustering: IL Counties")

# reasonable number of clusters
Kc <- 4

# cut tree into Kc clusters
clusters_c <- cutree(hc_c, k = Kc)

# draw boxes around clusters in the dendrogram
rect.hclust(hc_c, k = Kc, border = 2:(Kc + 1))
```



```
# check cluster sizes
table(clusters_c)
```

```
clusters_c
 1  2  3  4
44 16 41  1
```

Visualize the county clusters on a map. For this task, create a new `acs_map` object that now also includes cluster membership as a new column. This column should be called `cluster`.

Census Tracts

For the next section we need ACS data on a census tract level. We use the same variables as before.

```
acs_il_t <- getCensus(
  name     = "acs/acs5",
  vintage = 2016,
  vars     = c("NAME","B01003_001E","B19013_001E","B19301_001E"),
  region   = "tract:*",
  regionin= "state:17",
  key      = cs_key
) %>%
  mutate(across(everything(), ~ifelse(. == -666666666, NA, .))) %>% #fun() not working, fixe
  rename(
    pop       = B01003_001E,
    hh_income = B19013_001E,
    income    = B19301_001E
  )

head(acs_il_t)
```

```
  state county  tract                                        NAME  pop
1    17    031 806002 Census Tract 8060.02, Cook County, Illinois 7304
2    17    031 806003 Census Tract 8060.03, Cook County, Illinois 7577
3    17    031 806400    Census Tract 8064, Cook County, Illinois 2684
4    17    031 806501 Census Tract 8065.01, Cook County, Illinois 2590
5    17    031 750600    Census Tract 7506, Cook County, Illinois 3594
6    17    031 310200    Census Tract 3102, Cook County, Illinois 1521
  hh_income income
1     56975  23750
2     53769  25016
3     62750  30154
4     53583  20282
5     40125  18347
6     63250  31403
```

k-Means

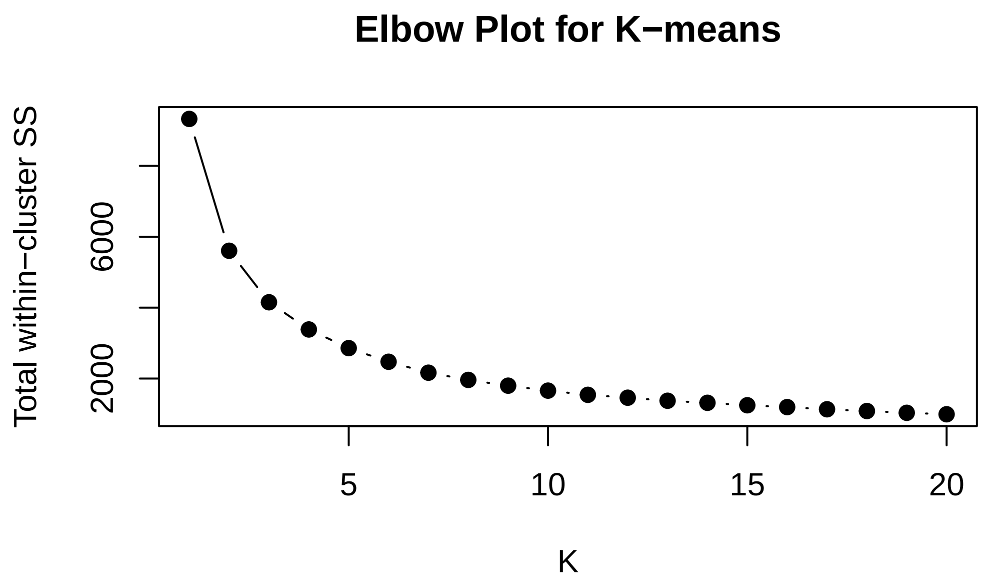
As before, clean our data for clustering census tracts based on population, average household income and per capita income.

```
# Extract county name from NAME
acs_il_t <- acs_il_t %>%
  mutate(
    county = str_extract(NAME, "[^,]+ County"),
    county = str_replace(county, " County", ""),
    county = str_to_lower(county)
  )

clust_dat_t <- acs_il_t %>%
  select(GEO_ID = NAME, county, pop, hh_income, income) %>%
```

```r
  drop_na()

Xt <- scale(clust_dat_t %>% select(pop, hh_income, income))
```

Since we want to use K Means in this section, we start by determining the optimal number of K that results in Clusters with low within but high between variation. Plot within cluster sums of squares for a range of K (e.g. up to 20).

```r
#Choose K: elbow (WSS) up to K = 20
kmax <- 20
wss  <- numeric(kmax)

for (k in 1:kmax) {
  km <- kmeans(Xt, centers = k, nstart = 20)
  wss[k] <- km$tot.withinss
}

plot(1:kmax, wss, type = "b", pch = 19, xlab = "K", ylab = "Total within-cluster SS",
     main = "Elbow Plot for K-means")
```



Run `kmeans()` for the optimal number of clusters based on the plot above.

```r
K_opt <- 5

set.seed(123) #for reproducability
km_final <- kmeans(Xt, centers = K_opt, nstart = 25)
clust_dat_t$cluster <- km_final$cluster

table(clust_dat_t$cluster)
```

```

   1    2    3    4    5 
 276 1016  274  744  799 
```

Find the mean population, household income and per capita income grouped by clusters. In addition, display the most frequent county that can be observed within each cluster.

```r
clust_summary <- clust_dat_t %>%
  group_by(cluster) %>%
  summarise(
    mean_pop = mean(pop),
    mean_hh_income = mean(hh_income),
    mean_income = mean(income),
    most_freq_county = names(sort(table(county), decreasing = TRUE))[1]
  )

clust_summary
```

```
# A tibble: 5 x 5
  cluster mean_pop mean_hh_income mean_income most_freq_county
    <int>    <dbl>          <dbl>       <dbl> <chr>           
1       1    3896.        122368.      67665. " cook"         
2       2    2686.         37123.      19778. " cook"         
3       3    7838.         86010.      38154. " cook"         
4       4    5381.         49260.      23275. " cook"         
5       5    3610.         73195.      35913. " cook"         
```

As you might have seen earlier, it's not always clear which number of clusters is the optimal choice. To automate K Means clustering, program a function based on `kmeans()` that takes K as an argument. You can fix the other arguments, e.g. such that a specific dataset is always used when calling the function.

```r
run_kmeans <- function(K) {
  set.seed(123)  # for reproducibility
  km <- kmeans(Xt, centers = K, nstart = 25)

  clust_dat_t$cluster <- km$cluster

  clust_summary <- clust_dat_t %>%
    group_by(cluster) %>%
    summarise(
      mean_pop = mean(pop),
      mean_hh_income = mean(hh_income),
      mean_income = mean(income),
      most_freq_county = names(sort(table(county), decreasing = TRUE))[1]
    )

  return(list(
    kmeans_result = km,
    cluster_data  = clust_dat_t,
    summary = clust_summary
  ))
}
```

We want to utilize this function to iterate over multiple Ks (e.g., K = 2, ..., 10) and – each time – add the resulting cluster membership as a new variable to our (cleaned) original data frame (`acs_il_t`). There are multiple solutions for this task, e.g. think about the `apply` family or `for` loops.

```r
acs_data <- clust_dat_t

# Iterate over K = 2 to 10
for (K in 2:10) {
  set.seed(123)
  km <- kmeans(Xt, centers = K, nstart = 25)

  acs_data[[paste0("cluster_K", K)]] <- km$cluster
}
```

Finally, display the first rows of the updated data set (with multiple cluster columns).

```r
head(acs_data)
```

```
                                      GEO_ID county  pop hh_income income
1 Census Tract 8060.02, Cook County, Illinois   cook 7304     56975  23750
2 Census Tract 8060.03, Cook County, Illinois   cook 7577     53769  25016
3    Census Tract 8064, Cook County, Illinois   cook 2684     62750  30154
4 Census Tract 8065.01, Cook County, Illinois   cook 2590     53583  20282
5    Census Tract 7506, Cook County, Illinois   cook 3594     40125  18347
6    Census Tract 3102, Cook County, Illinois   cook 1521     63250  31403
  cluster cluster_K2 cluster_K3 cluster_K4 cluster_K5 cluster_K6 cluster_K7
1       4          2          3          4          4          3          6
2       4          2          3          4          4          3          6
3       5          2          2          1          5          5          3
4       2          2          2          2          2          5          2
5       2          2          2          2          2          4          2
6       5          2          2          1          5          5          3
  cluster_K8 cluster_K9 cluster_K10
1          3          7           8
2          3          7           8
3          2          6           7
4          2          6           7
5          6          4          10
6          2          6           7
```